How A Company Can Participate In Open Source

A Handbook

Ron Goldman & Richard P. Gabriel
CONTENTS

Contents .................................................................................................................. iii
Preface ....................................................................................................................... ix
Acknowledgements .................................................................................................... xi
1. What is open source? ................................................................................................. 1
   An informal definition of open source software ....................................................... 1
   A quick history of open source ................................................................................ 1
   Philosophical tenets of open source ......................................................................... 2
      Historical roots of open source ............................................................................. 2
      Warning .................................................................................................................. 4
      Everyone together is smarter than your group alone ............................................. 4
      Open source means an open development process—starting with the design ...... 5
      The business model must reinforce the open source efforts ................................. 5
      Creating and involving outside developers requires internal resources ............... 6
      Open source is a gift economy .............................................................................. 6
      Open source is not business as usual ..................................................................... 7
   Common open source myths, misconceptions & questions ....................................... 8
      Myth 1: Open source development is something new. ........................................... 8
      Myth 2: Open source development is done by hobbyists and students. ............... 8
      Myth 3: Open source software is low quality....................................................... 8
      Myth 4: Large scale development isn’t efficient .................................................... 9
      Myth 5: If the software is free, then who’s going to pay you to write it? ............... 9
      Myth 6: By making our software open source we’ll get thousands of developers
            working on it for no cost. ................................................................................... 9
      Myth 7: Open source doesn’t scale ....................................................................... 10
      Myth 8: Does your company still own the source? ............................................... 10
   Open source and community ...................................................................................... 11
2. Why consider doing open source development? ..................................................... 17
   Business reasons for choosing open source .............................................................. 17
Contents

1. Visibility ...............................................................18
2. Ubiquity .............................................................18
3. Design discipline ....................................................18
4. Best standards development ......................................19
5. Design help ..........................................................19
6. Development help ....................................................19
7. Improved quality .....................................................19
8. Better way to do releases ..........................................20
9. Appeal to the community whose name is “The Open Source Community” ....................................................20
10. Guidance for proprietary products ...............................20
11. Get brand loyalty for your company’s hardware/software ......................20
12. Better relations with developers ....................................21
13. Get the hit effect working .........................................21
14. Community building for speculative reasons .......................21
15. Commoditize competition ........................................21
16. Avoiding lock-in .....................................................21
17. Risk reduction .......................................................22
18. Story-telling through connected activities .........................22
19. Statement of values ................................................23
20. Create a marketplace ................................................23
21. Creating a business model ........................................23
22. Following through on the business model .........................24
23. Measuring success ..................................................24

3. Licenses ........................................................................27
   What the license does ................................................27
   What the license does not do ......................................28
   A quick word on copyright .......................................28
   …and patents ...........................................................29
   The licenses ...........................................................29
   Proprietary licences ................................................29
   Sun Community Source License (SCSL) .........................30
   Open standard / proprietary code ................................31
   Sun Industry Standards Source License (SISSL) ...............31
   Sun Public License (SPL) ........................................32
   GNU General Public License (GPL) and GNU Lesser General Public License (LGPL) ...........................................32
   Berkeley Software Distribution (BSD) ..........................33
   Public Domain ........................................................34
   Supplementing the license—contributor forms ..................34
   Issues in choosing a license, dual licensing and joining existing projects .........................35

4. How to do open source development ................................39
   Public code archive ..................................................39
   Open mailing lists or newsgroups ..................................40
<table>
<thead>
<tr>
<th>Contents</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Let everyone know what is happening</td>
<td>40</td>
</tr>
<tr>
<td>Posting etiquette</td>
<td>41</td>
</tr>
<tr>
<td>Types and number of mailing lists/newsgroups</td>
<td>42</td>
</tr>
<tr>
<td>Project documentation</td>
<td>42</td>
</tr>
<tr>
<td>Project web site</td>
<td>43</td>
</tr>
<tr>
<td>Bug database</td>
<td>44</td>
</tr>
<tr>
<td>Software lifecycle</td>
<td>45</td>
</tr>
<tr>
<td>Planning and decision-making</td>
<td>45</td>
</tr>
<tr>
<td>Integrating contributions</td>
<td>46</td>
</tr>
<tr>
<td>How decisions get made varies among open source projects</td>
<td>47</td>
</tr>
<tr>
<td>Code reviews</td>
<td>47</td>
</tr>
<tr>
<td>Daily builds</td>
<td>47</td>
</tr>
<tr>
<td>Testing</td>
<td>48</td>
</tr>
<tr>
<td>Releases</td>
<td>48</td>
</tr>
<tr>
<td>Support</td>
<td>50</td>
</tr>
<tr>
<td>Making it happen</td>
<td>50</td>
</tr>
<tr>
<td>Building a community</td>
<td>52</td>
</tr>
<tr>
<td>Building trust</td>
<td>53</td>
</tr>
<tr>
<td>Establishing credibility with the open source crowd</td>
<td>54</td>
</tr>
<tr>
<td>Growing a community of users and developers</td>
<td>55</td>
</tr>
<tr>
<td>Give folks a good initial experience</td>
<td>56</td>
</tr>
<tr>
<td>A community web site</td>
<td>58</td>
</tr>
<tr>
<td>Who's in charge?</td>
<td>59</td>
</tr>
<tr>
<td>Meetings</td>
<td>62</td>
</tr>
<tr>
<td>Three principles for community building</td>
<td>62</td>
</tr>
<tr>
<td>Joining an existing open source project</td>
<td>64</td>
</tr>
<tr>
<td>License issues</td>
<td>64</td>
</tr>
<tr>
<td>Becoming a good community member</td>
<td>64</td>
</tr>
<tr>
<td>The open source development process</td>
<td>65</td>
</tr>
<tr>
<td>Getting your contributions accepted</td>
<td>65</td>
</tr>
<tr>
<td>5. Going with open source</td>
<td>67</td>
</tr>
<tr>
<td>Deciding to do open source</td>
<td>67</td>
</tr>
<tr>
<td>How to prepare to do open source at your company</td>
<td>69</td>
</tr>
<tr>
<td>Resources at your company to help you</td>
<td>69</td>
</tr>
<tr>
<td>Educate yourself and your team about open source</td>
<td>69</td>
</tr>
<tr>
<td>Educate your manager about open source</td>
<td>70</td>
</tr>
<tr>
<td>Develop your business model</td>
<td>70</td>
</tr>
<tr>
<td>Design appropriate review criteria</td>
<td>70</td>
</tr>
<tr>
<td>Choosing a license</td>
<td>70</td>
</tr>
<tr>
<td>Getting your source code ready to release</td>
<td>71</td>
</tr>
<tr>
<td>Design issues</td>
<td>72</td>
</tr>
</tbody>
</table>
Contents

Legal issues .................................................................72
Create a rational development plan ....................................73
Budgeting resources for your project: people and web site ........75
Creating a new web site ..................................................76
Making use of an existing web site .....................................77
Your web site needs an editor ..........................................78
Cross linking with your company's web pages ......................78
Getting approval from your company ..................................78
Getting approval to release Sun developed source code ..........79
Getting approval to incorporate downloaded source code ........80
Getting approval to participate in an existing open source project .81
What about small projects? .............................................82

6. How to build momentum .............................................85
Marketing your project ..................................................86
What is your project's story? ............................................86
The name matters .........................................................87
Getting the message out ................................................88
Going beyond standard marketing ....................................88
Focus on your users and contributors ................................88
Target your potential audience .........................................89
Make users successful ..................................................89
Encourage outside developers .........................................90
Communication creates community ....................................90
Make sure adequate resources are allocated .......................91
Community outreach .....................................................92
Welcome the unexpected ................................................92

7. What to avoid—known problems and failures ..................93
Not understanding open source .......................................93
Don't duplicate an existing effort ......................................94
Licensing issues ..........................................................95
Code issues ...............................................................95
Trying to control too much .............................................97
Marketing issues ........................................................99
Tension between an open source project and the rest of your company .100
Community issues .......................................................101
Lack of resources ........................................................103
Recovering from mistakes ............................................103

8. Conclusion .............................................................105
Appendix A: Resources .................................................107
Books for further reading ..............................................107
Open source web sites ..................................................107
# Contents

- Tools ................................................................. 108
- Licenses .................................................................. 108

## Appendix B: Open source licences ................................................. 109
- GNU General Public License (GPL) .................................. 109
- GNU Lesser General Public License (LGPL) ...................... 116
- Berkeley Software Distribution (BSD) .............................. 125
- Apache Software License ............................................. 126
- Sun Public License (SPL) ............................................. 128
- Sun Industry Standards Source License (SISSL) ................. 137
- Sun Community Source License (SCSL) ........................... 143

## Appendix C: Open source contributor forms ................................. 159
- Free Software Foundation Copyright Assignment Form ........ 159
- Apache Contributor Agreement ...................................... 163
- OpenOffice.org Contributor Assignment ......................... 165
- Project JXTA Contributor Assignment ............................. 166

## Appendix D: Article from XML.com ............................................. 169
Preface

In the fall of 1998 Bill Joy and Mike Clary asked us to help establish a community of developers and companies around Sun's new Jini network technology. Jini is a simple distributed computing model based on the Java programming language that enables services running on different computer systems to spontaneously interact with each other over the network. For Jini to succeed, it was clear that the underlying Jini protocols and infrastructure would need to become pervasive, and to accomplish that would require a strong community of participants and partners. Moreover Sun did not have the expertise to define Jini services in areas like printing, digital photography, storage, white goods, or the many other potential markets for products that would benefit from being network enabled.

As a preliminary step the Sun Community Source License (SCSL) had been developed to provide a framework where the source code for Jini could be safely shared. However a license and providing the source code were not enough—there had to be more to motivate people to participate. We worked with the Jini engineering and marketing teams to apply the lessons we had learned from being involved in various open source projects and from our studies of complexity science to create a true Jini Community.

We worked hard to help establish an identity for the community. To build identity and culture requires face-to-face meetings, so we organized community meetings that were held in interesting places with non-traditional speakers. The first Jini Community meeting was at the Aspen Institute in Aspen, Colorado. We played some games, we engaged a recording graphic artist, we broke into groups in unusual ways. We encouraged nonstandard thinking and encouraged speaking as individuals rather than as representatives of companies at times. The keynote speakers included Robert Dahl, a political scientist, talking “On Democracy,” and Thomas Petzinger, a Wall Street Journal columnnist, speaking about new business models based on cooperation.

We also helped to create a web site for the project (http://www.jini.org) to use as a meeting place, a newspaper, a repository of shared documents, a totem, and a place to share work and projects. Most of the shared assets of the community can be accessed there. The web site has a public part that advertises the community, and private parts for members only to help foster identity.

Common vocabulary leads to shared stories and then to shared beliefs. We created a pattern language on how the Jini Community works, which has served to create a bit of shared vocabulary—
terms like dangerous waterhole, cut & run, and microcosm. The pattern language is designed to show how to build (self-)governance structures and procedures on the fly which are of a variety that have proven to be comfortable and delightful in the past but which are tailored to the specific on-the-spot needs of the community.

We also worked with the community on developing a formal decision-making process and a “community process” for the purpose of ratifying definitions for Jini services. The former is to make it clear how community-wide decisions are made—how are proposals created, how are they voted on, how are appeals made—and the latter is to specify how the community can officially bless or endorse a service definition.

Since the spring of 2000 we have done similar work with other Sun-sponsored open source projects such as NetBeans, OpenOffice and Project JXTA. We have worked with each of those teams to help them to define their open source strategy along with how to implement it.

This handbook contains the lessons we have learned from helping to create and nurture communities around these Sun-sponsored open source projects. It also includes insights from studying other open source projects such as Linux, Apache and Mozilla. The handbook is in two parts. The first will describe what open source is, give some of its history, discuss business reasons for using open source, discuss the various licenses currently used by open source projects, and describe how an open source project works in a day-to-day manner. The second part will help you decide on whether open source is right for your project, and, if so, what steps you should take to proceed and some mistakes you should avoid.

This handbook is a work in progress and we welcome comments on how to improve it. Please send your suggestions to rgoldman@cs.stanford.edu. An online version of it can be found at: http://www.dreamsongs.com/OSHandbook.html
ACKNOWLEDGEMENTS

The authors wish to acknowledge everyone who has talked with us or written about their experiences with open source; we have heavily mined those works to produce this handbook.

We wish to thank Bill Joy and Mike Clary who initiated this work by asking us to help create a vibrant community of developers using Sun’s Jini Technology. We also want to acknowledge the strong support we received from Emily Suter and Ingrid Van Den Hoogen. We also thank the members of the Jini development team, especially Ken Arnold, Mark Hodapp, Jim Hurley, Charles Lamb, and Jimmy Torres. We also acknowledge some of the Jini Community members that helped us such as Cees de Groot, Alan Kaminsky, Michael Ogg, Aleta Ricciardi, Sylvia Tidwell Scheuring, Jerome Scheuring, and Bill Venners.

Thanks to Rob Gingell and Bonnie Toy for asking us to advise other groups at Sun who were considering using open source and supporting us while we completed this Handbook.

The members of Sun’s Open Source Program Office, Danese Cooper, James Duncan Davidson, and Eric Renaud, were very helpful in spreading the open source message at Sun.

We thank Brian Behlendorf and Mitchell Baker for sharing with us the lessons they had learned from the Apache and Mozilla projects. Also Tim O’Reilly for sharing his insights about open source.

We have benefited from comments on early versions of this handbook from a number of readers including Bob Sproull, Emily Suter, Danese Cooper, Lawrence Lee, Ed Burns, Steven Rubin, Philippe Lalande, Duncan Findlater, Michael Davey, and Rod Evans.

We also want to thank the various Sun groups that have let us experiment on them as they have explored using open source. The NetBeans team has been especially helpful, particularly Evan Adams, Kartik Mithal, Dirk Ruiz, and David Taber. From StarOffice/OpenOffice we want to thank Alan Templeton, John Heard, and Marissa Bishop. The whole JXTA team has been great to work with, thanks to Juan Carlos Soto, Emily Suter and Ingrid Van Den Hoogen. Also John Tollefsrud of the Sun GridEngine project.

Also thanks for their help to Guy Steele, Jan Hauser, Jennifer Umstattd, John Ravella, Thomas Petzinger Jr., Robert Dahl, Kathy Knutsen, Bill Lard, Clifford Allen, Mike Schilling, Douglas Begle, Thomas Benthin and Steve Graham.
Special thanks to Mary Alexander for editorial assistance and writing advice.
1. **What is Open Source?**

This parts will describe what open source is, give some of its history, discuss business reasons for using open source, discuss the various licenses currently used by open source projects, and describe how an open source project works in a day-to-day manner.

**An Informal Definition of Open Source Software**

There are many definitions of what constitutes open source. The basic idea is very simple: By making the source code for a piece of software available to all, any programmer can modify the software to better suit his or her needs and redistribute the improved version to other users. By working together, a community of both users and developers can improve the functionality and quality of the software.

Thus to be open source requires that anyone can get and modify the source code, and that they can freely redistribute any derived works they create from it. The different licenses have various wrinkles on whether modifications must also be made into open source or if they can be kept proprietary.

In this handbook we will also discuss cases that do not meet the definition of open source, but do involve some form of collaborative development of the source code, for example Java.

**A Quick History of Open Source**

In 1984 Richard Stallman formed the Free Software Foundation (http://www.gnu.org) and started the GNU project (GNU is an acronym for “GNU’s Not Unix”). He did so in response to seeing the collapse of the software-sharing community at the MIT AI Lab as everyone left to join companies making proprietary software. Stallman coined the term *free software* to express his philosophy that programmers should be allowed access to the source code so they could modify it to suit their needs. He developed the GNU General Public License (GPL) to assure that he would always be able to see and modify the source code for any software he wrote, along with the modifications made by anyone else who might work on it. A number of important Unix tools and utilities have been developed as part of the GNU project including the GNU GCC compiler and GNU Emacs.
In 1989 the University of California at Berkeley released their networking code and supporting utilities as Networking Release 1, the first freely redistributable Berkeley Software Distribution (BSD). This was followed in 1991 with Networking Release 2 that contained almost all of the source code for BSD Unix (all but six files were included).

In 1991 Linus Torvalds started work on the Linux kernel under the GNU GPL and around 1992 combined it with the not-quite-complete GNU system to produce a complete free operating system. In 1995 Red Hat Software was formed to sell CD-ROMs plus offer support and services for the users of Red Hat Linux.

In 1995 a group of webmasters created the Apache project, based on the NCSA web server. Apache 1.0 was released later that year. According to a June 2001 Netcraft web server survey, the Apache web server is more widely used than all other web servers combined and currently has over 60% of the web server market.

In 1998 Netscape announced they would release the source code to their Navigator browser and three months later did so. Partly in response to the Netscape announcement a group of leaders in the free software community, including Sam Ockman, John “maddog” Hall, Eric Raymond, and Larry Augustin, met to discuss how to better market the ideas of free software. They decided to use the term Open Source and working with others created the Open Source Definition (c.f. http://www.opensource.org/docs/definition.html), which is based on the Debian Free Software Guidelines written by Bruce Perens for the Debian Linux distribution. They created an organization, the Open Source Initiative, to further the ideas of open source and to certify licenses as being true open source. There is a continuing tension between those who support free software and those who favor open source due to different philosophical outlooks.

More details on the history of open source are available on-line and in books like Open Sources: Voices from the Open Source Revolution edited by Chris DiBona, Sam Ockman & Mark Stone or Rebel Code: Inside Linux and the Open Source Revolution by Glyn Moody.

**PHILOSOPHICAL TENETS OF OPEN SOURCE**

Understanding the philosophical tenets of open source is simpler once you know a little about its history, which is about as ancient as anything in computing.

**Historical roots of open source**

The open source philosophy has its roots in the era of timesharing at large research universities like MIT, Stanford, and CMU. In the 1960's and 1970's, these schools had research laboratories whose researchers and projects produced software—this included basic platforms such as operating systems and programming languages as well as higher-level, application-type software. This was necessary because either the research work itself was in the area of operating systems and programming languages, or there were no suitable commercial or “professionally” created systems filling these roles.

The computers these Labs used were almost exclusively timeshared machines, with a limited number of terminals connected to them. During the day the machine was heavily used not only for research but for writing papers, and for administrative, and clerical purposes. Yes, there was e-mail in
Imagine a 1 MIPs machine with about a megabyte of memory shared by 100–200 people. A lot of real work was done at night.

At night, doors are normally locked and the only people wandering the halls would be the janitors, who had keys to every office. Terminals were usually in offices, and if those offices were locked, the terminals were not available for use by anyone without a key. Almost certainly there were fewer terminals than people, and so the physical resources were shared as well as the computational ones. People would start large computations and head home, experimental operating systems might crash, experimental computations might crash the operating system—in short, things could get wedged easily.

Suppose that the terminal controlling an out-of-control computation was behind a locked door and the problem was sufficient to hamper other people working at 2am. What could you do? Go home until the person returns tomorrow? Time was too valuable, and there might be 50 people who have to go home because one person has locked his or her door. Or suppose the OS crashes in such a way that even rebooting doesn’t fix it—the permanent context has changed enough that the OS software needs to be fixed (perhaps a new peripheral has been added, like a robot arm). Do you wait until you can talk to the “programmer in charge” to make the change when you yourself are the expert about what to do?

No, you institute two changes from normal procedures:

• you do not allow offices to be locked
• you permit anyone to work on any of the sources

Horrifying by current standards, but it worked just fine and created a culture that had very high productivity. First, rather than having a single or several system administrators, such Labs developed quite a few, perhaps a few dozen for a 200-person Lab. These folks could do most anything sys admins do today, and in addition they were pretty darn good system programmers as well—they would typically have a part of the OS or other system software they were responsible for creating and maintaining. People who joined the Lab would have all the source code available to look at how things were done, and in fact, how to program well. What better way to learn about programming than by looking at the working code of master programmers?

The result was that the machines were productively used 24 hours a day, 365 (or 366) days a year with support. Operating systems, programming languages, and a lot of the software we take for granted today were developed incrementally over a few years time with the full-time or part-time help of hundreds of programmers.

Social pressure and a kind of hierarchy of merit was used as a filter on who could work on what. Vandalism, bad behavior, and harmful mischievousness rarely if ever took place, such was the strength of the code of ethics that developed in such installations.

Here’s a story we heard from the old days:

A new faculty member joined one of the “open-source-like” Labs and had a terminal in his office. Used to a more traditional environment, he started a job, locked the door, and went home for the night. The job ran away and locked up the system in such a way it needed to be killed...
from the terminal or the system would need to be rebooted. One of the system programmers had to break open the door, pulling the hinges out of the door jamb. The next morning the new faculty member found his door leaning against the wall, splinters all over the floor, a note on his seat explaining what happened, and another note from the director of the Lab informing him that his unrestricted funds account would be charged to have the door and jamb repaired.

This is the culture which became the open source movement. The attitude is that people will behave well and that there is a common good being promoted. Source code is part of a shared experience that builds and supports the culture.

**Warning**

Therefore, if as you read these tenets you feel that they don't match how you work, how you feel, and how you could work, then you will not be able to fit into the culture required to make a successful open source project. In that case, you should not pursue open-sourcing your project: Your project will not succeed, your project will embarrass your company in the public eye, and you will have mangled your project in such a way that either it will have to be abandoned or you will have to abandon it.

Harsh, yes, but open source is not for everyone nor for every project—at least not yet.

**Everyone together is smarter than your group alone**

Inviting people to look at source code puts them in a frame of mind to help out. Not only will you gain a lot of testers, but these testers will have the source code and many of them will be able to locate the source of a problem in the code itself. The sum total of people running the software will encompass a wide variety of experience and expertise, and it is not unreasonable to assume that every type of bug in your code has been seen by someone in the pool at one time or another. It’s like having a thousand people proofread a book for spelling and typographic errors.

You will mostly get highly informed and dedicated testers. The fact the source code is available means that many of them will be at least able to develop their own workarounds to problems, and so they will stick with the software longer. But just as importantly, you will get a number of people who will want to contribute, often substantially, to the core source code.

A very common reaction to the idea of someone outside the organization contributing to the code base is that you don't want “a bunch of losers messing things up.” Our experience is the opposite. First, there is no requirement in open source that every change proposed by someone must be taken back into the main source branch. So, you can select what you take, and you can modify or improve it if you wish. Second, it is your decision whether someone is trusted enough to check things directly into the source tree. You will have developed a relationship with all these people, looked at their work, worked with them to improve their skills and use the style you want, and in most cases you will wish you could hire them. Third, there will not be that many of them anyway. For Linux, which is the biggest open source project, there are on the order of 100 people who are permitted to check code into the source tree without supervision. Many of these people “own” parts of the code or modules, and they will supervise other contributors.
If it really is the case that all the people out there are losers with respect to the code in your project, then there are no replacements to hire if one of your developers goes down or leaves your company. Wow, if that's true, we hope your developers all have fabulous compensation packages and bodyguards.

**Open source means an open development process—starting with the design**

If you drop code into a public location and continue development on a private copy inside your company, you are not doing open source. You are running a source access program or simply providing extra documentation. To develop the right level of trust, the code that is out there must be the real source code that your developers are working on. You might have various modules checked out while the code is being worked on, but that must be true on a module-by-module basis, not the entire source base.

Moreover, your developers are simply part of the larger pool of developers. They are perhaps better focused, they might work together better, or they might have as their goals things of a common importance to your company, but they are otherwise exactly in the same boat as all the outside developers.

This means also that many if not all the otherwise internal development mailing lists need to be public. All design and implementation decisions, that affect the public source code must be done in the open. There can be no distinction between us and them.

In exchange for this you build incredible loyalty and an intense desire to move the code forward with a shared vision. You are getting immediate and totally relevant feedback from the market. If you are short of resources, perhaps they will be supplied from the outside. The highly influential book on people management, *Peopleware* (by Tom DeMarco and Timothy Lister), suggests that a good way to build a team is to have a shared project, where the project itself comes to symbolize the group. Perhaps it’s getting your newly assembled team together to cook an impromptu meal, but in the end it’s working a shared artifact that does the trick. That’s the same trick here. But it’s no trick, it is part of the social nature of people.

**The business model must reinforce the open source efforts**

You cannot charge people to use code that they are writing or have written. That means that you cannot simply take the code and charge money directly for it. You need to choose a business model that supports open source. The following are some classical open source business models:

- Bundle open source software with perhaps some other software and with support, charging for the bundle and for additional testing and quality
- Add value in the form of additional modules or surrounding software and sell that additional software bundled with the open source software
- Provide a service based on the open source software, such as a subscription service that updates customers’ sites with tested and assured code
- Sell consulting services that leverage the open source software
What is open source?

- Sell ancillary things like books, T-shirts, and mugs
- Sell hardware that runs the open source software particularly well
- Sell software that uses the open source software as a platform

Ubiquity, winning over hearts, thousands of eyes looking over the code, better platform security, and some additional outside help are some of the reasons to do open source. The classical case is that you need to make a platform ubiquitous for some other business reason, and so you use open source as the conduit to ubiquity.

Creating and involving outside developers requires internal resources

Many people think doing open source is a no-brainer: You can add developers to your team and not need to pay them. However you do need to attract, foster, and support those outside developers and that does require internal resources. First and foremost it requires your internal developers to take the time to participate on the public mailing lists, to answer questions from outside developers and to promptly respond to bug fixes and code contributions. Your developers will also need to document the system architecture so outside folks can find their way around in the source code. You may even need to rewrite the source code to be more modular, especially if it is currently one big monolithic mess.

Another cost is setting up and maintaining the infrastructure needed for an open source project: a CVS code archive, a bug database, various mailing lists and a web site for the project. For large projects these can each require a full-time person.

Open source is a gift economy

To understand open source it helps to make a distinction between a commodity economy, to which we are accustomed in a capitalist society, and a gift economy. In a gift economy, gifts are exchanged, forming a bond based on mutual obligation: In the simplest form of gift exchange, when one person gives a gift to another, the receiver becomes obligated to the giver, but not in a purely mercenary way—rather, the recipient becomes very much like a member of the giver’s family where mutual obligations are many, varied, and long lasting. A person may give a gift with the realistic expectation that someday a gift of equal or greater use value will be received, or that the recipient will pass on a further gift. In an open-source project, the gift of source code is reciprocated by suggestions, bug reports, debugging, hard work, praise, and more source code.

The commodity economy depends on scarcity. Its most famous law is that of “diminishing returns,” whose working requires a fixed supply. Scarcity of material or scarcity of competitors makes high profit margins. It works through competition.

The gift economy is an economy of abundance—the gifts exchanged are inexhaustible. Gift economies are embedded within non-economic institutions like kinship, marriage, hospitality, artistic patronage, and ritual friendship. A healthy Western family operates on a gift economy. In an open source project an individual’s status and reputation depends on the quality of the gifts they contribute.
For open source to succeed in communities that include corporations, the source-code gift-based economy needs to thrive alongside the commodity economy hovering on its boundaries.

**Open source is not business as usual**

In summary deciding to make a project open source means that it will be very different from your normal proprietary project. All decisions and design needs to be done in the open. Schedules will depend on people outside your company that you cannot control. The community that emerges will take on a life of its own, possibly taking the project in directions you neither anticipated nor desire.

Your business model and processes must be different or else you won’t succeed. You don’t do open source as an add-on to your usual process. Deciding to do open source is like deciding to get married and start a family: It takes a commitment and once you start down that path it will change how you live.
COMMON OPEN SOURCE MYTHS, MISCONCEPTIONS & QUESTIONS

The picture of open source software painted by the popular media tends to be superficial and simplistic. Open source is touted to be a miraculous way to produce software at no cost. For anyone developing software professionally all this open source hype no doubt seems pretty farfetched. Let’s take a closer look and try to shed some light on what open source is really all about.

Myth 1: Open source development is something new.

If you read the newspaper, open source seems to have started with the Linux operating system back in 1991 (or more likely, in 1997 or 1998 when whoever wrote the article finally heard about Linux). The actual facts are a bit different: open source is as old as computer programming. If you had wandered into places like MIT or Stanford in the 1960’s sharing of software was assumed. Early development of the ARPAnet was helped by freely available source code, a practice which has continued as it grew into today’s Internet. The Berkeley version of Unix that Bill Joy was a contributor to dates from the mid-70’s. All in all quite a distinguished lineage.

Myth 2: Open source development is done by hobbyists and students.

The stereotype for the sort of person who contributes to an open source project is that of a hobbyist or student, someone you maybe wouldn’t take too seriously. After all, full-time professional programmers don’t have time for such things. Well, first thing, students and hobbyists can often write very good code. Next lots of professionals do like to program on their own time. Moreover many professionals are paid by their day job to develop open source software: Both Sun and IBM have engineering teams contributing to various parts of the Apache web server. Most companies building PC peripherals now write the needed device drivers for Linux as well as Windows. In fact, the open source process encourages the replacement of contributions from poorer programmers with code from better ones.

Myth 3: Open source software is low quality.

How can a bunch of random programmers, with no QA folks, produce code with any degree of quality. Isn’t open source software full of bugs? Well there may initially be as many bugs in open source as in proprietary code, but because it’s open, more developers will actually look at the code, catching many bugs in the process. Also everyone using the code is essentially doing QA; they will report any bugs that they find, and since they have access to the source code, they will often also fix the bugs themselves.

If you still don’t believe that open source software is actually of higher quality than most commercial software just take a look at some open source software you use everyday. Assuming you make any use of the Internet, then you are relying on open source code like BIND, which is at the heart of the Domain Name Service (DNS), or sendmail, which probably transported most of the email you receive, or Apache, which in June 2001 was the software running on over 60% of the world’s web servers. Then there’s Linux, which has won several awards for quality, and has a notably longer mean-time between reboots than some other major PC operating systems.
Myth 4: Large scale development isn’t efficient.

Having thousands of people fixing bugs might work, but how can you possibly coordinate the work of that number of developers? Without central control how can it possibly be an efficient process? Well, that’s correct, but why does it need to be efficient? When you have limited resources, efficiency is important, but in an open source effort with lots of developers, if some go off and write a module that eventually is rejected, it doesn’t matter. Open source efforts often progress in small steps. If several people are working on different solutions to a problem, as long as one eventually produces a solution you are making forward progress. If two solutions are produced that’s even better, just pick the best. Also with the ease of email and news groups, the various people working on that problem will likely find each other and spontaneously self-organize to work together to produce a result better than any of them alone could have. All without any central control.

Myth 5: If the software is free, then who’s going to pay you to write it?

Why should your company pay you to write free software? Well, your company may already be doing that. Are you working on a product that is sold or distributed for free? Or are you working on something only used internally? Is the income generated from selling the software you write greater than the cost to produce it? The profit may come from other activities. Likewise for “free” software. Your company will continue to make its money from selling hardware (e.g. servers, storage and workstations); proprietary software; books; and consulting, training and support.

For example, O’Reilly and Associates sells enough Perl books to pay the main Perl developers to work on new Perl features. Several of the main Linux developers are employed by Red Hat, which makes its money by packaging up “free” software. Cygnus (now part of Red Hat) sells support for the GNU compiler and debugger, which its employees continue to develop and give away. Finally Sun sells servers, but gives away Java.

Look at the section on business models for more details of how your company can make money from open source software development. Keep in mind though, that roughly 90% of the money spent on software development is for custom software that is never sold; sale of commercial software is less than 10%.

Myth 6: By making our software open source we’ll get thousands of developers working on it for no cost.

That would be nice, but in reality most open source projects have only a few dozen core developers doing most of the work, with maybe a few hundred other developers contributing occasional bug reports, bug fixes and possible enhancements. Then there are the thousands of users, who may contribute bug reports and requests for new features. The users also post messages asking how to use the software and, hopefully, the more experienced users post answers to those questions. Some users may even help write documentation.

It’s true that you don’t need to pay any outside developers who choose to work on your project. However you will need to pay the cost of maintaining the infrastructure necessary for open source development (e.g. a CVS code server, a bug database, project mailing lists, and project web site),
along with people to integrate the contributions you get. You won't get something for nothing, but for a successful open source project you can get back more than what you put in.

**Myth 7: Open source doesn't scale.**

Experience with conventional, proprietary software development teaches that the larger the project the greater the number of resources needed for coordination and design. For an open source project where all of the discussion is via mailing lists and where there is no formal management structure, it seems that it would be impossible to efficiently organize the developers. Hence open source might work for small projects, but not for large ones.

However Brooks' Law that adding more developers to a late project will just make it later does not totally apply to open source because developers are no longer treated as a scarce resource that must be used efficiently. In his essay *The Mythical Man-Month*, Brooks points out that new developers must be trained, that larger teams require a greater overhead to communicate with each other, and that not every task may be partitioned. For an open source project it is important to distinguish between those developers who make up the core team—the module owners and few developers with check-in privileges—and the much larger number of occasional contributors. The core team is always going to be too small and all of the lessons of conventional software development apply to them, including Brooks' Law. However it is with the larger group of contributors that open source changes the rules: These are the folks who can track down obscure bugs and create fixes for them, they can help each other to get up to speed on the code, they can implement features from the project's wishlist, or they can explore and experiment with more radical modifications—all activities that free up the core team to focus on their own main work.

Instead of controlling and scheduling developers, open source relies on the developers self-organizing based on their interests and abilities. Instead of a management structure to coordinate everyone's work, open source development requires resources to evaluate and integrate developer contributions. Moreover, those resources can draw on the total community of developers and are not limited to any core group. To see this look at the success of some of the large open source projects, like Apache or Linux.

**Myth 8: Does your company still own the source?**

Yes, your company still owns any source code that it releases under an open source license because your company still owns the copyright. The open source license grants others the right to examine and use the source code, but it does not affect your company's ownership of the code. As the copyright owner, your company can release the source code under another license or use it in a proprietary product. Only if your company were to distribute the source code without a copyright notice, deliberately putting it into the public domain, would your company no longer own it.

However, source code contributed back to your company by outside developers is owned by the author, who holds the copyright for it. Under some licenses, like the Sun Community Source License (SCSL), your company would be able to use the contributed code without restrictions. Under an open source license, such as GPL or the Mozilla Public License (MPL), your company would be bound by the terms of the license just like any other developer.
OPEN SOURCE AND COMMUNITY

If you talk about community with almost anyone working on an open source project they will draw something like the following figure:

![Diagram showing a single community built around source code.](image)

Figure 1: Single community built around source code

They will tell you about how people start as just users and how some will become more involved by reporting bugs. Some may become developers who fix bugs or make minor enhancements. A few developers then get more involved and join the ranks of the core developers, being granted the right to check in changes to the actual project’s source code. This is a very code-centric view of the open source process as a hierarchy that has users at the periphery, occasional developers closer in, core developers even closer in, and the code at the center.

And in fact, this view makes the hierarchy into a funnel in which the goal is to convert people from one ring in the diagram to people in the next inner one—almost as if the goal were to turn people into code in the end, the highest form of existence.

If we step back a bit we can see that the people involved in an open source project take on various roles. Over time they may act as a user, a designer or a coder. So a better diagram would look like this:

![Diagram showing a single community built around source code.](image)

Figure 2: Still focused on code

Here the code is still at the center; users, designers, and coders can look at the code; but only so-called core coders deal directly with the code. We use the term coder rather than developer to emphasize the roles being played: A coder is someone who manipulates code, either by writing new code to
implement some design feature or by inspecting and correcting existing code someone else has written.

The thin, solid red lines indicate changes to the source code, the dotted lines indicate viewing the source code and interacting with the program, and the thicker, blue lines indicate communications between people in the different roles. Note that the ovals representing the different roles are not drawn to scale; the users circle should be much, much bigger for any healthy open source project.

This is still a view that focuses on the source code, but it brings out that design is separate from coding and that designers need not be coders. It also reflects the fact that people in the community may not simply advance from user to core developer in a linear progression, but adopt roles that make sense for what they are doing at the moment.

If we look at the ways that people interact when working on an open source project we see they do so in many ways, such as:

- online discussion
- meetings
- project web site
- documentation
- code

There can be discussions that do not involve the source code at all. Users can converse about how to best do their jobs, maybe only just touching on how the project’s tools can help them. Other conversations may focus on standards for protocols or APIs that the project uses. Still other conversations may address issues affecting the larger open source community. We might represent these multiple communities like this:

![Figure 3: Communities built around common interests](image)

An open source project has many different communities, each with a different purpose. For a successful project, each of these communities must be nurtured and helped to grow. In a successful community a vocabulary might spring up which is derived from the project’s technology, application area, and existing culture. Such a community will come to resemble a long-existing club with its own phrases, in jokes, rituals, and customs—an astute creator of such a community will know this, and
will help the community grow these aspects. One less astute will focus on the code, probably leaving out vital potential members of the community.

To make this more concrete let’s look at some of the different communities that exist within two Sun projects, Jini and NetBeans.

In the Jini project there are separate communities focused on creating Jini services in different application areas:

![Figure 4: Different Jini code development interests](image)

A small number of developers care about further developing the core Jini code. Others care only about areas like printing or enterprise applications or building networked appliances that use Jini to connect to services. People working in one application area often have little in common with those working in another area. For example, those people working on creating services for appliances in the home have a very different set of concerns from those using Jini to connect legacy enterprise applications.

Another example is to look at the various communities involved with the NetBeans project:

![Figure 5: Multiple NetBeans products and uses](image)

Besides those directly involved with developing the NetBeans source code, there are three additional groups at Sun working on the various editions of Forte for Java: Community Edition (CE), Internet Edition (IE) and Enterprise Edition (EE). There are also various third party companies developing NetBeans modules.

There are several distinct user communities. First there are those developers using NetBeans as an IDE to create Java applications—their concern is how to best use NetBeans to write Java programs. There is also a geographical information system (GIS) built using the NetBeans framework, so
users of that application are not involved with any programming; they want to discuss issues around using GIS. When the NetBeans IDE is enhanced to handle languages besides Java, like C/C++, then there will be a new user community of C/C++ developers.

For NetBeans to be a success all of the communities shown above need to be successful. Each must grow for NetBeans to prosper.

NetBeans provides another example of how an open source project can involve multiple communities, in this case three very different cultures:

- NetBeans
- Forte
- Sun

At Sun, the NetBeans open source project arose from the simultaneous acquisitions of NetBeans, a small startup company in Prague, Czech Republic, and Forte Software, a company based in Oakland, California. These two companies joined Sun’s existing Tools organization, bringing together three companies, each with different corporate cultures, values, day-to-day processes, and business goals. This was a challenging acquisition integration task.

Unifying three different corporate cultures and day-to-day processes would be challenging enough, however, the bigger challenge may have been that the Tools organization had become spread across three different sites (Menlo Park, Oakland, and Prague).

Distributed development is fundamental to the open source methodology. It is simply assumed. So, the open source methodology represented both a natural solution for the Tools organization’s physical distribution and a neutral, fourth methodology to bridge the differences in corporate cultures and processes.

As you read the next section and think about your project’s business goals and business model, you need to consider how you will involve the various groups that have an interest in your project:

![Figure 6: Possible constituencies](image)
Your business model must include a community focus.
2. **Why consider doing open source development?**

An open source project only works if everyone involved derives some benefit from it. That means that there must be a good reason for your group to make its work available, and for outsiders—users, developers, and other interested parties—to want to participate.

One very compelling reason is if there is an existing open source project that your work can build upon. By joining that project you avoid needing to redo large amounts of work. Plus, there is an existing community of outside developers that has an interest in what you are adding to their project. An example of this is when Sun contributed its Java Server Pages technology to the Apache web server project to create the Jakarta project (http://jakarta.apache.org). By doing this Sun assured that JSP would become available on the most widely used web server, that the Apache developers would not create a competitor to JSP, that it would be easier to work with other companies, like IBM, on JSP technology, and that JSP could build on the work already done for Apache. For the Apache community adding JSP was a way to add desired functionality and expand the community of Apache developers; it was also a vote of confidence in their work with Apache.

There are many reasons that will cause outside developers to participate. These include:

- They want to use the features that you are providing.
- They can influence development.
- They can modify your code to suit their needs.
- They can participate in a cool project.

These are also reasons why you might choose to join an existing open source project.

**Business reasons for choosing open source**

Deciding to manage a development project as an open source project requires making a careful determination of the project’s business goals—an open source project is generally more expensive to
operate than a proprietary project, and there are different scheduling and development processes that need to be adopted and which can affect product delivery.

The following is a partial catalog of possible business goals for an open source project:

**Visibility**

By running a project as an open source project, the existence and nature of the project are visible to the outside world. If the project is visible to a strategically important community—whether they participate in the project or not—your company is better able to “present” plans, goals, features, and statements of progress. Open source makes available additional communication channels, including more informal ones consulted by the target developer community.

For example, any sort of open, Java-related activity, such as the NetBeans project, enables the developers working on it to use the various Java newsgroups or weblogs like Slashdot to communicate directly with other developers.

**Ubiquity**

By providing an open source offering, a project can begin to spread that offering first to the outside users and developers for the project, out to the broader open source community, and then perhaps to the larger developer community. Both the no-cost and open nature of the offering and the transparent stance of the project to users of the offering contribute to the spread of the offering. The transparent stance conveys the message that your company’s developers don’t consider themselves the ultimate authority on the offering and not only value the contributions of the community but consider members of the community as equals.

For example an explicit goal of NetBeans was to create a common IDE platform that would be one of the top three most popular Java development environments. This is a ubiquity goal. Jini is another example of a project that attempted to establish a new, ubiquitous platform for services.

Another example is StarOffice where ubiquity includes users. A person might go to OpenOffice.org to download the latest version, but then stay to participate in the mailing lists discussions of how to best use the OpenOffice application and how it might be improved. Because of the feeling of community that arises from being part of those discussions, the person is much more likely to recommend OpenOffice to others.

**Design discipline**

Doing development open source style requires making almost all decisions on mailing lists open to the public. Design and implementation proposals are made in writing, are discussed by the community, and the resulting decisions are summarized in writing. All this written material is archived and available to everyone. By enforcing this level of discipline, the quality of the design and implementation decisions is likely to increase—simply because they are being done deliberately and in writing. The process of writing it all down creates a need to express issues in a clear way, which often forces one to better understand matters. This better understanding then leads to a better design. In many ways, the open source process resembles the literate programming methods Don Knuth and others have promoted over the years.
Best standards development

Some projects have as a goal developing a standard—API definitions, language extensions, or tool extensions. By opening up the process to everyone, it is more likely that the best advice and greatest adoption will take place.

The NetBeans IDE APIs had been designed in an open way to begin with, so an open source approach would simply continue this practice. A goal for OpenOffice was to create an XML standard for file formats.

Design help

By opening up the design process to the community, a project can not only get advice on direction but help on how to design the implementation for that direction. This can come in the form of what is generally considered design, as well as direction regarding features and usability from pure users. Feedback from users includes answering if the project is even worth doing at all.

We call users who provide good directions and even pure design, user/designers. This is a specialized case of the goal to foster innovation outside of your company.

Most developers don’t distinguish between development and design, so by aiming for development help, open source projects expect design help as well.

Getting design help was an important goal for the Jini project, since for Jini to succeed meant defining Jini services for printing and consumer appliances that fall outside of Sun’s areas of expertise.

Development help

One of the myths of open source is that an open source project will attract a lot of developers, so that the costs of operating a project using open source will be lower than using proprietary processes. This isn’t true primarily because software is still largely monolithic or made of modules that are too large and so most volunteer developers will not have the time or motivation to fully understand the code well enough to want to or be able to add significantly to it. However, finding bugs, fixing some bugs, and porting software to other platforms or making it work in new configurations are the sort of help a project can reasonably hope to get. This is also a specialized case of the goal to foster innovation outside of your company.

For a project like NetBeans that is structured as a set of modules communicating through API’s, one goal was to create a community of module developers. This has already happened: outside developers have indeed contributed new modules and continue to work on them.

Improved quality

By making releases available to users early in the development process, bugs are more likely to be caught and fixed. This results in greatly improved quality in the official release. Moreover the bugs that bother users the most are the ones that are actually most apt to be fixed.
Better way to do releases

The open source process of doing daily development releases and frequent stable releases in addition to periodic major, official releases, is often better suited to customer needs. This process allows cutting edge customers to immediately have access to the latest features, while more conservative customers can wait for major releases that are guaranteed to be more solid and bug free. Customers can choose the level of risk they are comfortable with.

Major releases are possibly done less frequently, but by having an open development process with more communication between users and developers the development can be done less frantically without everyone feeling that the project has slowed down.

Appeal to the community whose name is “The Open Source Community”

Some open source pundits and leaders talk about “the open source community” as if it were a clearly identifiable set of people. Some speak as if this community were homogeneous, making statements such as “the open source community rejects Jini,” when in fact, quite a few people in the “open source community” are members of the Jini community. Considered as a political entity, though, it is true that the “open source community” rejects the license used for Jini.

The idea of this goal is to gain acceptance and positive statements from the pundits and leaders in “the open source community” and various smaller communities within it. For example, the Linux community is a thriving Unix community within the broader open source community. By doing an open source project, the organization running the project can get closer to that community.

A prime example of this was releasing the StarOffice source code under the GNU LGPL. Using that license, which is a favorite of many in “the open source community,” improved Sun’s reputation.

Though this wasn’t an explicit NetBeans goal, the open source community is not a strong Java development community. In fact, much of this community is not a strong C++ development community, favoring C. Further, this community has been slow to pick up Java because of Sun’s ownership of it. By doing a full-up Java project as open source, Sun hopes to improve its standing with that community.

Guidance for proprietary products

Some open source licenses permit proprietary products to be based on an open source system, sometimes by adding features and sometimes by providing quality and support. Being able to develop the best such product can be assisted by commentary and feedback from the community.

This was a major goal for NetBeans, with the Forte for Java Internet and Enterprise Editions being for-sale products built on the NetBeans platform.

Get brand loyalty for your company’s hardware/software

By creating a community of users and developers using an open source project, your company gets a reputation as being a “good guy” with this and related communities, and that can create brand loyalty.
Every company-sponsored open source project has this goal whether stated or not. For example with NetBeans the goal was to increase the number of Java developers, which increases brand loyalty to Sun.

**Better relations with developers**

Open source breaks down the barriers between your company and outside developers. By including them in the development process, outside developers and users feel a part of the community working on the project. When done right this leads to a better relationship between them and your company.

Many of these developers and users are also key influencers in their own companies, so their having a closer relationship with your company can directly translate into their recommending other of your company’s products for use by their organization.

**Get the hit effect working**

By studying how movies and popular music become runaway hits, researchers have found that a large network of key influencers can cause a word-of-mouth wildfire to take off. By the right people speaking favorably about a work, those favorable words can spread and by bumping into the same person from different directions can cause that person to become a fan. This can happen with an open source project when key influencers are in the community and are favorable to its projects.

This is an important mechanism for ubiquity.

**Community building for speculative reasons**

In some cases your company is interested in getting into a new area where it may have little or emerging expertise and credibility. Starting an open source project can bring expertise and contacts from the community concerned with the area.

An example would be Project JXTA where Sun engineers are collaborating with many outside experts to explore possibilities of peer-to-peer computing (P2P). By working with a larger community, Sun hopes to better establish expertise in the P2P space and be able to take early advantage of business opportunities that arise based on P2P.

**Commoditize competition**

By providing a no-cost offering, any competition can be forced to play a commodity game where small advantages and brand loyalty, for example, can play a stronger role than in a high-value game.

OpenOffice is an example of this goal: Offering free office-productivity software makes it more difficult for other companies, such as Microsoft, to charge large amounts for proprietary software with similar features. It transforms office-productivity software from a unique product into a commodity.

**Avoiding lock-in**

A common strategy for companies is to offer a high-value product and create brand lock-in by either making it hard to move to another product or by providing special platform-specific capabilities. An
open source community will not tolerate proprietary lock-ins, and will generally move toward making the offering as portable as possible.

This was a major goal for NetBeans, where the competitors to NetBeans and Forte for Java were seen as promoting lock-in not only to their own IDEs but perhaps also to particular Java implementations.

Risk reduction

A concern to many companies is the risk they face if a software product they depend on is discontinued. With open source applications they know they can continue product development, even if the original developers stop working on it. In the past, the primary way of reducing the risk that a producer would abandon software or go out of business was a practice called software escrow wherein the source code was held by a third party as if in a vault whose contents were released to the consumer of the software upon certain contractually defined triggers.

Making an application open source also reduces the risk for the company that initially developed it. For example Cisco Systems decided to release CEPS, the Cisco enterprise print system software, under an open source license partly to reduce its dependency on in-house programming staff. Cisco wanted to make sure that if the original programmers left the company, the software would continue to be maintained. In fact one of the two original developers did eventually take a new job elsewhere, but since it is an open source project he has been able to continue to work on it. If a developer really identifies with a project, then you can ensure that person’s participation for a long time by making it open source.

Another way that a company reduces its own risk by making a product open source is that the outside developers who work on the project become attractive candidates to hire: they are knowledgeable about the product and have a proven track record. There are numerous examples where a major contributor is later hired by the company that started the open source project.

Story-telling through connected activities

This is a general marketing-based goal. Typically marketing takes place via collateral material, events such as JavaOne and product launches, and press releases. There are not very many ways to get small pieces of news into the world. Web sites have helped with this. An open source project has a web site, mailing lists, and community events, but they are aimed more at developers and users than at the press and analysts. It is possible to tell these smaller stories within the context of normal open source communications. For example, a developer could mention in a design consideration something that implies a new strategic direction for your company. Even simply having a developer at your company working hard at making some program or system work well on a competitor’s system conveys a story about your company’s values that is not possible to tell through normal marketing avenues.

This is an explicit NetBeans goal, though it is not likely that anyone thought of it this way. The Forte for Java product line is marketed partly through the open source project, and statements about the Forte for Java direction are made indirectly through the actions and discussions by NetBeans developers working in the NetBeans community.
Statement of values

By doing an open source project, your company is saying that it believes in the general open source principles about its benefits, and that the world beyond your company is full of expertise and innovation. It is a statement of humility. Every open source project achieves this result, at least to some degree.

The example above about working to enhance performance of a competitor’s product exhibits the value that your company cares about its customers and wants to provide them with the best products possible—even if they come from a competitor.

The business goal that is achieved by affirming your company’s values is indirect, helping to establish your company’s reputation. This in turn influences how potential customers and business partners think about your company and what they will expect in their dealings with your company.

Create a marketplace

By providing a ubiquitous open source platform, you can create a marketplace for add-ons, support, and other related products and services. Having an open source base platform greatly reduces the risk a player in the new marketplace must take on, because there is no owner of the underlying technology to possibly leave the market or go out of business.

One goal NetBeans had was to create a marketplace of NetBeans modules that Sun could participate in. For Sun to do well here required there to be a vigorous third-party marketplace.

Creating a business model

Every company is interested in activities that support its business. You must be able to explain how the relevant business goals for your project contribute to your company. Some of the goals have a direct connection to revenue, such as selling a proprietary product based on an open source project. Others have a more indirect connection, such as better relations with developers making it easier to sell your company’s hardware or services into an organization.

For example, Sun makes most of its money by selling hardware, so any software that encourages customers to buy Sun servers or workstations supports Sun’s main business model. It can do so directly by providing desired functionality and high quality, such as Solaris. It can do so by offering an alternative to software available on other platforms, such as StarOffice. It can try to level the playing field (or even tilt it in Sun’s favor) by establishing new standards, such as Java. It can try to create a new market where Sun can be a major player, such as Jini. It can do so by gaining developer mindshare, so more applications for Sun hardware will become available, such as NetBeans. If shifting to open source development will make these activities more likely to succeed then a strong case can be made to do so. As an aside, please note that each of these strategies may be best supported by a different type of licence; we’ll discuss how to choose a license in a later section.
Following through on the business model

Your business model justifies your use of open source both to managers within your company and also to outside developers. It is important that you can communicate your business model to both groups.

Many outside developers are suspicious of the motives of large companies and will view an open source project sponsored by such a company as an attempt to pull a fast one. To quell these fears you must be able to explain how your company plans to benefit from participating in the open source project. Only when they understand what your company plans to gain will they be comfortable supporting the project. If they don’t hear a sensible business reason for why your company is using open source, you can be sure that someone will suggest a variety of sneaky motives, often based on the fears open source developers have that companies will steal their work and put it into proprietary products. If you are not willing to disclose your main business reasons then you should rethink making your project open source.

You will also need to explain your business model to other groups within your company that you need to interact with. Other parts of your company will expect your project to be business-as-usual and will not be happy when they find out otherwise. For example, if another group inside your company has a product that depends on yours, then you need to explain to them that your company does not control the release schedule and that your project will not do a release if your community feels the code isn’t quite ready. You can expect there to be lots of pressure on your project to act like a normal proprietary project.

Measuring success

There are fairly standard ways of measuring whether a proprietary project is successful. These include product sales, market share, number of bugs, meeting schedules, and other familiar assessments. Many of these also apply to open source projects, but there are additional metrics that are also necessary.

Each of the business goals for your project should have an associated measure. Here are some possible metrics for various business goals:

- Ubiquity: number of users, number of mentions by the press, response by competitors
- Standards development: number of participating parties, number of parties adopting the standard
- Design help: number of additional user/designers, amount of email on mailing lists
- Development help: number of outside developers, number of bugs reported/ fixed, number/quality of contributions, amount of email on mailing lists
- Appeal to community: quality and number of comments in open source community

By default your project will be measured the same way a proprietary one is, which will probably result in the open source activities getting shortchanged. For a successful open source project you need to get management to agree to a different set of metrics. The performance reviews for each
employee participating in the open source project activities needs to include measures for their open-source-related activities. Note that just adding the new metrics in with all of the standard existing ones won't work; you cannot expect that people can successfully do both their old job and take on the new tasks required to create a healthy open source community.
2 Why consider doing open source development?
3. LICENSES

A variety of licenses have been created to meet the different needs of open source projects—the original Berkeley Unix was released under the BSD license; work on Emacs and Linux use the GNU GPL, as does StarOffice; Mozilla uses the MPL. Sun has two open source licenses—the Sun Public License (SPL) used for NetBeans and the Sun Industry Standards Source License (SISSL) used for NFS—and also a non-open source license, the Sun Community Source License (SCSL), used for Java and Jini. Your reasons for choosing open source development will determine which license is best for your project.

Note that the license is only a gate that people pass through. If people are not willing to agree to the terms of the license, then they don’t pass through the gate. However for those people who do accept, the license doesn’t specify how they will work together; it merely defines some very basic ground-rules.

What the license does

The license grants outside developers certain rights that establish what they can and cannot do with the source code. All of the licenses we will consider here grant the developer the right to use and modify the source code. Some licenses also include the right to use any Intellectual Property (IP), such as patents, that is required for the source code. Most licenses do not grant developers the right to take that IP and use it in a different application.

Each license also requires developers to assume certain responsibilities. For example, some licenses require that any bug fixes a developer makes to the source code must be contributed back to the original developer. Another common requirement is that any IP used in the source code that a developer contributes must be made available to other developers who use that code.

A major difference between licenses is what the developer must do to redistribute executable binaries built from modified versions of the source code. Many open source licenses require that the modified source code also be made available, at no or nominal cost, to anyone who wants to see it—this being the whole point of open source.

Some licenses allow the source code to be incorporated into a larger work that is not subject to the terms of the license, though the original source code that is included still is; other licenses consider
any additions to just be extensions of the original program and, as such, subject to all of the license

terms. This is a key factor if you plan on combining open source code with proprietary code.

Some of the Sun-created licenses—SCSL and SISSL—including compatibility requirements in the
license itself. This is an issue that may be addressed in other licenses in the near term.

Finally all of the licenses deal with various legal matters: warranty (there is none), liability (it’s lim-
ited), termination of the license (if you violate any of its terms), dealing with brands and trademarks
(they are not included with the license) and several other boilerplate issues familiar to anyone who
has read any license accompanying commercial software (e.g. clauses about governing law, dispute
resolution, U.S. Government use, international use, and severability).

What the license does not do

It is just as important to realize what the license does not do. The license describes certain boundary
conditions, but does not speak about how developers will actually work together. It is up to other
documents or traditions to describe the process of contributing code, making a new release, and
deciding what to do when disagreements arise. In fact many open source projects have a form that
anyone wishing to contribute code must sign before their code will be accepted. (This form is dis-
cussed below in the section “Supplementing the license—contributor forms” on page 34.) Later sec-
tions of this paper will look at the day-to-day functioning of an open source project and how to
build up a community of developers.

One other matter that most licenses do not touch on is how to ensure that modifications to the
source code maintain compatibility with established standards. One way this can be done is to
require that any distributed code pass a compatibility test in order to be granted the right to use a
logo or brand. If there is an established brand then this can be enough of a carrot for developers to
keep things compatible. Most often, compatibility is maintained because the community values it
and will not accept code contributions that deviate from the standard; anyone who wants to create
an incompatible version is thus forced to fork the code and start a new project.

A quick word on copyright…

The author of a piece of software owns the copyright to the code and that gives the author the right
to choose the terms of the license. The copyright owner may license the code to one group under one
set of terms, and to another group under a different set of terms. For example, Sun has made the
StarOffice source code available under both the LGPL and the SISSL; moreover Sun can also use
the same code in a proprietary product like StarPortal.

Since it is the copyright holder who gets to determine the license terms, there can be a problem if
there are multiple copyright owners. Every person who makes a change or addition to the original
source code becomes the copyright owner for the code they have written. When these changes are
contributed back to the open source project, their authors usually license them under the same terms
as the original source code, indeed many of the open source licenses require that to be the case.
Many open source projects will not even accept a contribution unless it is released under the same
license as the original code—having various pieces of the source code under different, possibly
incompatible, licenses would be too confusing and difficult.
A further problem arises if a project decides that it wants to change the license being used. For example, the Mozilla project is considering dual licensing of the source code under both the MPL and the GPL. However for any changes to the licensing to happen, every person who has contributed any source code, and hence owns the copyright to it, must agree to the change. Just tracking down all of the copyright holders can be a major task, let alone getting them to agree to the proposed licensing changes. Because of this some open source projects request, or even demand, that contributors also assign their copyright over to either the original author of the code or some agreed upon third party, such as the Free Software Foundation.

...and patents

Everything you know about patents from proprietary software development still applies to the world of open source projects. Since your code is “published” when it is made available to people working on the project, any international patents involved with the code need to be applied for beforehand, and any US patents filed within one year. By contributing the source code to the project, you may be granting anyone who distributes a product based on the project, the right to use your patent. Depending on the license you use, they may or may not have the right to use the source code, and the patent, in other work.

The licenses

We will now look at some licenses that have been used for various collaborative projects that Sun engineers have participated in. Note: this is only an overview of the various licenses—for the real details you need to read the license itself. The full text of each license is included in “Appendix B: Open source licences”.

Proprietary licences

Though very far from being open source, it is possible to do limited collaboration under a proprietary license. This can work when the collaboration is with a few companies, rather than being open to anyone. This has been referred to in the open source world as a gated-community.

For example, Sun’s Free Solaris Source License Program allows anyone who has signed the license to view and modify the Solaris source code. They are allowed to distribute their changes to other licensees only via a Sun secure web site, and are not allowed to incorporate any Solaris code into other products. While offering licensees only very limited rights, this can still be attractive to outside organizations that have compelling reasons to tailor the source code to their needs.

This type of license creates a strong class system, with the licensees having few rights in contrast to the organization issuing the license. This will severely limit the size and scope of contributions that licensees will be willing to share. The originating organization will gain few, if any, of the benefits of an open source project. There are two main reasons to use a proprietary license: If your company does not own all of the source code, so cannot legally make it available publicly, or if the technology involved is so advanced that it is important to minimize the number of people who are permitted to look at it.
Sun Community Source License (SCSL)

The Sun Community Source License (SCSL) was created in an attempt to combine the best of proprietary and open source development. Two major goals of the SCSL were that compatibility among deployed versions of the software be required and enforced through testing, and that proprietary modifications and extensions including performance improvements be allowed.

The SCSL defines two types of use: Research and Commercial Use. Research Use means that the source code is only used for research, development, educational or personal use. Commercial Use is when an executable based on the source code is used or distributed for any direct or indirect commercial or strategic gain or advantage. (The original version of the SCSL also included a third type of usage, Internal Deployment Use, which was in between Research and Commercial Use. In the most recent version of the SCSL, Internal Deployment has been merged with Commercial Use.)

In Research Use one can basically do anything to the source code, except give it to someone who has not also signed the SCSL. The only requirement is that any bug fixes be provided back to Sun.

Commercial Use adds a requirement that any code distributed must pass a compatibility test as provided in the appropriate Technology Compatibility Kit (TCK). Commercial Use also requires that an organization must sign separate Commercial Use and trademark licenses. Since those licenses may include royalty payments made to Sun, the SCSL does not meet the requirements of an open source license. The requirement that any distributed code must maintain compatibility also causes some open source leaders to reject the SCSL, since they see this as limiting what developers can do with the source code. Of course, Sun never claimed that the SCSL was an open source license—hence the term community source.

The SCSL grants the right to use any IP associated with the original source code, and requires that when a developer contributes source code they must also grant rights to any IP that it requires to other developers who wish to use that code.

The SCSL was originally developed as a license for Jini, but was then modified to also be the license used for providing developers access to Java. Because the needs of Java and Jini were different, the resulting common license became more complex. For example, to use the Java TCK requires signing a separate support agreement with Sun.

The added requirements of making the SCSL meet the needs for both Java and Jini, along with the sheer number of issues that the license covers, resulted in a long and complicated license. Many developers and companies have found this confusing, causing some to refuse to sign the license. A revised version of the SCSL was made available in October 2000 that, while still complex, is somewhat clearer.

The SCSL is intended for projects that are trying to develop an infrastructure technology, where compatibility and interoperability are crucial and where having a strong, committed party guiding the development of the technology is seen as a positive factor. Organizations that sign the SCSL are definitely in a second-class role compared to Sun, and that will limit the type and number of contributions they are willing to make. Also as the technology matures there will be a strong desire by developers to become equal members with Sun. Many of the companies working with Jini are very happy to have Sun taking the lead on developing the core Jini technology, while they devote their
efforts to Jini-based products. However they have also been reluctant to contribute enhancements they have made that would be useful to other Jini community members.

For a good overview of the ideas behind the SCSL please see the paper Sun Community Source License Principles by Richard P. Gabriel and William N. Joy which can be found at http://www.sun.com/981208/scsl/principles.html. A copy of the most recent version of the SCSL as used for Jini can be found at http://www.sun.com/jini/licensing/licenses.html.

Open standard / proprietary code

Not a license per se, this model of collaboration is limited to the joint development of a public standard. Individual organizations would then develop their own proprietary code to implement the agreed upon standards; this is pretty much business as usual. But it may be all that is required to get the job done. Note that the use of “open” here means that the details of the standard are available for anyone to use or implement. It does not mean that the process that created the standard was open to anyone.

An example of this is the public development of the ADA programming language, which was followed by several companies creating proprietary implementations of ADA compilers. Another example is the IEEE 754 floating-point standard that was openly developed, and then implemented in proprietary hardware by various chip manufacturers.

Here the collaboration is to create a community to reach agreement on what the standard should be. Once such a standard exists, proprietary efforts can compete to implement it.

Sun Industry Standards Source License (SISSL)

Going a big step further, the Sun Industry Standards Source License (SISSL) provides the source code to implement a specified standard, along with the right to use any IP associated with that source code. Developers can use and modify this source code and distribute executables built from it as they wish, but if they make any modifications that do not comply with the standard, then they must publish the differences and make available a reference implementation under the same terms as the SISSL.

Larger works may be created using the SISSL, so that it is possible to add proprietary code and not be required to publish it.

The SISSL does not include any provisions for developers contributing source code back to the community. It is up to the individual developer to choose which license they want to offer their modifications under.

The reasons to use the SISSL include developing momentum for a standard, and preventing other companies from hijacking the standard with proprietary extensions. If the code is freely available, that makes it easier for developers to adopt the standard. If they then modify it to be incompatible, the SISSL forces them to announce their changes and to publish the modified source code.
The SISSL is a true open source license and has been certified as such by the Open Source Initiative (http://www.opensource.org), a non-profit corporation dedicated to managing and promoting the Open Source Definition.

A copy of the SISSL can be found at http://www.openoffice.org/licenses/sisssl_license.html.

**Sun Public License (SPL)**

The Sun Public License (SPL) is practically the same as the Mozilla Public License (MPL). It merely changes all references to Mozilla to refer to Sun and includes documentation as part of the source code. It is also a true open source license.

Larger works may be created using the SPL, so that it is possible to add proprietary code and not be required to publish it. Note that any changes to files in the original source code are considered modifications that must be made available to the community; they are not considered part of a larger work. This means that while it is possible to add proprietary modules as part of a larger work, the interfaces in the original code to those modules must be made public.

The SPL grants the right to use any IP associated with the original source code, and requires that when a developer contributes source code they must also grant rights to any IP that it requires to other developers who wish to use that code.

The reason to use the SPL is to create a community of developers who can easily share modifications, but who might also want to make proprietary additions to go into products they would sell. For example, Sun released the source code for NetBeans, a Java IDE, under SPL hoping to create an active community of developers working to improve it, but allowing those same developers to sell any proprietary modules they develop.

A copy of the SPL can be found at http://www.sun.com/developers/spl.html.

**GNU General Public License (GPL) and GNU Lesser General Public License (LGPL)**

The license most associated with the idea of open source is the GNU General Public License (GPL). It was created by Richard Stallman for use by the Free Software Foundation to distribute the source code developed for the GNU project. The idea behind the license is that while organizations can sell computer software, the source code should be freely available for developers to learn from and to modify.

The GPL does not allow larger works to be created from the open source codebase. The source code for any modifications or extensions must also be released under the GPL. This is the famous *viral* nature of the GPL. Any developer who contributes code to a GPL project is assured that they will always be able to see the source code to any future extensions; no one will be able to take their code and use it in a proprietary product.

Some people worry that if they were to include a GPL'ed program on a CD, that any other programs on the CD would become *infected* and they would need to make the source code for them available. This is not the case: the GPL explicitly states that bundling GPL'ed code with other programs has no effect on the other programs. It is only when source code released under the GPL is incorporated
with other source code, compiled, and distributed that the other source code becomes subject to the
GPL. All three steps are required for the GPL to take effect.

If you use GPL'ed code in another program, but do not distribute that program, then there is no
requirement to make your source code available. It is the act of distributing the binary executables
that triggers the requirement to publish the source code. Individuals are welcome to modify GPL'ed
code for their own use. When the GPL was written this made sense, since only the developer could
use the resulting program if it was not distributed. With the growing number of programs that are
used to support web sites this is no longer the case; installing the program on a web site can make it
available to millions without actually distributing it. This is an area where the GPL may be modified
in the near future.

The GPL ties access to the source code with the right to use any IP in the source code, by insisting
that any patent associated with contributed code “must be licensed for everyone’s free use or not
licensed at all.” If a company makes a contribution to a GPL project, they must allow any IP they
own that is in the donated code, to be freely used by the members of the project. The GPL states
that if a patent license would not permit royalty-free redistribution of the program by all those who
receive copies of it, then no one can distribute it. This is “to avoid the danger that redistributors of a
free program will individually obtain patent licenses, in effect making the program proprietary.”

The Free Software Foundation has a second license, the GNU Lesser Public License (LGPL) which
is used for libraries. Any modifications made to the source code for the library must be made avail-
able as with the GPL, but the source code for any program that is only linked with the library does
not need to be made available. Thus a proprietary program can use LGPL’ed libraries.

The reason to use the GPL and LGPL is to make sure that any modifications to the original source
code remain available and can not be modified and then used privately in proprietary programs. An
example of the use of both licenses is in how HP has licensed its e-speak technology: The common
core portions of e-speak are under GPL, so any changes must be shared with all, while the libraries
are under LGPL, so people can use them freely with proprietary code they write to create new e-ser-
vices. Thus HP assures the basic technology is shared, but they can also encourage new services to
be written.

Copies of the various GNU licenses can be found at http://www.gnu.org/copyleft/gpl.html (GPL)
and http://www.gnu.org/copyleft/lesser.html (LGPL). The FSF also maintains a FAQ that
answers many questions about the use of the GPL and LGPL that can be found at http://

Berkeley Software Distribution (BSD)

A number of open source licenses are variations on the license used by the original Berkeley Soft-
ware Distribution (BSD) of Unix, which is based on simple copyright. Essentially every file has a
copyright notice listing the original author and a requirement that any versions of the source code
that are distributed include the original copyright notice. There is also a no-endorsement clause say-
ing that the names of the originators and contributors can not be used to endorse products derived
from the source code. Plus there is the usual disclaimer of any warranty.
The original BSD license also included an advertising clause that stated that any advertisements for derived products must include a statement saying the product was based on work done by the original contributor. This clause was removed from the BSD license in 1999, but still appears in licenses that were derived from the original BSD license.

Variants of the BSD license, such as the one used by the Apache project, add a clause saying that any derived products cannot use certain terms in the product name without prior permission. So products derived from the Apache source code cannot have the word “Apache” in their names unless they have permission from the Apache Group.

There are essentially no requirements for developers working with source code released under a BSD-style license. They can make any modifications they wish and redistribute the results however they choose. There are no incentives in the license to encourage developers to contribute their modifications back to the community.

The BSD-style license does not include any mention about the right to use the IP in the source code. Just because a company has contributed source code under a BSD-style license does not mean that they have given up their rights to any IP they own in the donated code. If someone donates code that includes IP they own, then they can require that anyone wishing to use their donation acquire a separate commercial license from them. Most open source projects would reject such a donation. Likewise if a company were to claim that some of the project’s code infringed on one of their patents, then the response of the open source project would be to remove the offending code, and to rewrite it so that it no longer infringed.

The reason to use a BSD-style license is to make the source code as easily available as possible to outside developers, while, possibly, retaining the right to be credited for the original work. Sun’s Project JXTA uses a variant of the Apache license.

A copy of the BSD license can be found at: http://www.opensource.org/licenses/bsd-license.html. A copy of the Apache license can be found at: http://www.apache.org/LICENSE.txt.

Public Domain

A final choice is to use no license or copyright whatsoever and put the source code into the public domain. From that point on anyone can essentially do anything with the code. (Note: recent rulings on there being an implicit copyright on any published material make it unclear as to how to put something into the public domain.)

The main reason to make the source code public domain would be if you had no plans for any further work on it. For example, end-of-life code that you are essentially dumping on the chance that it may be useful to someone, somewhere.

Supplementing the license—contributor forms

While the license specifies what rights someone has to use the source code, many open source projects also require that any developer wishing to contribute to the project sign an additional form. This form states that the developer has the right to contribute their source code, i.e. that they own it
and that it does not belong to someone else. If the developer works for a company, the form has the developer state that their company grants the right to use the contributed code to the project.

The form is also used to have the developer state that if any patents or third party IP are used by the contributed code then the project is granted the right to freely use it.

Some projects use the form to assign the copyright of the code over to the project.

The developer usually only needs to sign the form the first time they contribute code; the form then applies to any subsequent contributions.

Note that requiring developers to sign such a form before accepting their contributions goes beyond the scope of the license. The developer is still free to exercise their rights to the code, but if they want to participate with the specific project codebase, then they must sign the form. If the additional requirements of the form are not acceptable, then the developer may be motivated to join or create a new project, i.e. fork the project.

The actual text of contributor agreements for several open source projects can be found in “Appendix C: Open source contributor forms”.

Issues in choosing a license, dual licensing and joining existing projects

For a new project if you want people to be able to take your source code and include it in commercial products, then you probably want your project to use BSD or the SPL. If you are developing an infrastructure technology, then SCSL or GPL will be better suited to your needs—SCSL, if the technology needs a strong central organization to move it forward, or GPL, if the technology is more mature.

If compatibility is a major concern then the SISSL is a possibility, though a better strategy might be to use BSD or the SPL and require derived products to pass a compatibility test in order for them to use a desirable logo/brand.

One thing to note about licenses is that the copyright owner may release the same source code under multiple licenses. So because Sun owns the copyright to StarOffice, any of the StarOffice source code can be used in a Sun proprietary product, even though the code is available to others under the LGPL. Sun could also license the StarOffice code to some company for use in a proprietary product. However any modifications or bug fixes made by an outside developer working under the LGPL could not be used in any proprietary product, unless that developer had assigned the copyright over to Sun.

Note that there is nothing in the way the source code is licensed that would require any developer to assign their copyright to Sun. However to have a contribution included in the Sun maintained OpenOffice project requires such an assignment. So far, a year after the start of the project, developers have been willing to go along with this.

If there are two disjoint developer communities that cannot or will not use the same license then a dual licensing scheme to allow both communities to participate can make sense. However both groups need to benefit. For companies that will not participate under GPL and developers who want
to use the code with other GPL'ed code, only a dual license can bridge the gap. The Mozilla project started with code under a dual license of the Netscape Public License and the Mozilla Public License due to Netscape needing to have special access to the code in their proprietary products. Use of the NPL was intended to diminish over time and indeed has done so. Now Mozilla is shifting to a dual license of MPL and GPL in order to work better with other open source projects that use GPL.

For StarOffice, Sun has released the source code under a dual license that allows each developer to choose whether they will abide by the LGPL or by the rules of the SISSL. This is to allow companies that will not touch code under the LGPL to still be able to participate, though the main community will be the developers working under the LGPL. It is important that any developer contributions be made using the same dual license structure. Otherwise there would be code that could only be used by the LGPL developers, and other code only available to those developing under SISSL; in effect the project would fork.

If you are thinking of joining an existing open source project, then that project has already selected a license. You need to consider if you can achieve your business goals under that license. If you are planning to combine the open source code with proprietary code, then GPL would not work, but any of the other licenses, including LGPL, would be ok.

Finally, we again stress that the success of an open source project is mainly influenced by how the day-to-day matters are handled. That a project like Apache, based on a minimal BSD-style license that does not even mention collaboration, can succeed, is proof that we need to look beyond the license.
## Summary of licenses

<table>
<thead>
<tr>
<th></th>
<th>Proprietary</th>
<th>SCSL</th>
<th>SISSL</th>
<th>SPL</th>
<th>GPL</th>
<th>LGPL</th>
<th>BSD</th>
<th>Public Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Can be mixed with proprietary software</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>IP used in contributions must be made available to all developers</td>
<td>✔ 1 ✔ ✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Modifications must be published</td>
<td>2, 3 4 5 ✔ ✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>When incorporated into a larger work license covers all of it</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Includes compatibility requirements</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Original developer has special rights</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Can redistribute binaries</td>
<td>✔ ✔ ✔ ✔ ✔ ✔ ✔ ✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Can redistribute source code</td>
<td>✔ ✔ ✔ ✔ ✔ ✔ ✔ ✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>

Notes:

1. Only if the IP is required by a modification that does not comply with the standard.
2. All bug fixes must be published.
3. If a modified or new interface specification (API) is shared with any third party, then the API must be published for all to see.
4. Only changes that do not comply with the standard must be published.
5. Only changes to files containing the original code or community contributions must be published.
6. May only be distributed to those who have signed the SCSL.
4. **How to Do Open Source Development**

The key point about open source is that the development process must really be open: All the developers, both internal and external, need to have the same access to the source code and be able to fully participate in discussions and decisions about its design. If you currently have a geographically dispersed workgroup then you are already probably aware of many of the issues involved. More traditional, co-located workgroups will likely need to change the way they currently work.

In this section we will look at the major pieces needed to do an open source project and also discuss the activities that go into creating a new release. As we do so note that there are a number of new jobs that need to be done. To have a successful open source project requires people to do those jobs, so you will need additional resources.

One other important point to keep in mind as you read on is that while most of the focus in this section is on developers, users are also a big part of the process. It is important to keep users involved in the process and not treat them as second-class citizens.

**Public Code Archive**

A prime requirement for an open source project is that the source code be publicly available:

- Any developer, whether inside or outside the company, should be able to get the latest version of the code anytime.
- A developer who is in charge of maintaining a module—the *module owner*—should be able to directly make changes to the source code.
- Contributions and bug fixes from developers who have not yet been granted write-access to the source code need to be integrated into the source code archive in a timely manner. This can be a major task and will most likely be done by the various module owners and other core developers.
- Frequent builds of the source code must be done, daily where possible, and made available for download by developers and users. Usually a recent build that was fairly stable is also kept available for download.
This is very different from the usual proprietary development model where the internal developers have their own private copy of the official source code that they periodically release to external developers. With everyone sharing the same source code archive, when any developer fixes a bug or makes a change it is immediately available to all of the other developers; internal developers do not get special access. You do not want to discourage potential contributors by having them spend time tracking down a bug and fixing it, only to find that someone else has already done so.

Most open source projects use CVS (Concurrent Versioning System) to maintain the repository of shared code. CVS provides a way for multiple developers to edit the source code without stepping on each other's changes. It also supports defining branches to create multiple versions of the code and makes it possible to roll back to an earlier version. CVS is itself an open source project and is available for free from either ftp://ftp.gnu.org/gnu/non-gnu/cvs/ or http://www.cvshome.org/downloads.html. Bonsai is a tool used by the Mozilla project (http://mozilla.org/bonsai.html) that lets developers perform queries on the contents of a CVS archive to determine recent changes and who was responsible for changing a particular line of code.

One job that must be assigned to a team member is to set up and maintain the CVS code server. Another job is to do the daily build and make sure no one has checked in code that breaks things. A third job, mentioned above, is to integrate bug fixes and contributions into the source code archive.

Open mailing lists or newsgroups

Another major aspect of an open source project is that the design is done in the open. All of the developers should be using a public mailing list or newsgroup for their discussions. Your internal developers should be participating there and not using some internal mailing list. Note: in the following discussion for brevity's sake we will often just talk about using a public email list, but we also include using a public newsgroup in addition to or instead of the mailing list.

Let everyone know what is happening

It is vital that community members are involved in discussions with your internal developers. If it appears that your internal developers are doing their work in closed internal meetings and exchanges of private email, then outside developers will feel like they are being treated as second-class citizens and will not participate as much as they might. That is not to say your internal developers can only communicate over public email lists or newsgroups, but when you have a meeting be sure to post notes from it to the list and consider inviting outside developers to the meeting, either to attend in person or over the phone. Also if you post the agenda beforehand, outside developers can express their viewpoints via email and they can be considered at the meeting. Note that the more design work is done via email, the easier it is to preserve it in a mailing list archive; often design decisions are never documented and this creates problems down the road when the assumptions underlying the design may change or new people join the project and need to understand why certain decisions were made the way they were.

There are other reasons to let everyone know what is being considered. An important one is to alert the community about any plans to make major changes to the code. The worst situation is when a module owner makes large changes to their module with no notification whatsoever. If other developers find out about the change because it breaks something they are working on, or they go to do
something with the module and find the code is totally different, then they are not going to be happy. They are going to feel that they cannot depend on that module, since to them it has changed in an arbitrary manner. Slightly better is if the module owner had announced that the code had been changed; the other developers may be just as bad off, but at least they know it. The best thing to do is to announce in advance what changes are being considered, engage in open discussion with anyone concerned, and then announce what was changed when the new code is checked in. Outside developers will feel part of the process, because they really are part of it—plus the quality of the design will likely be improved because the discussions included more viewpoints.

On an Apache project mailing list in July 2000 there was a heated exchange of messages because two groups, one from Sun and another from IBM, had been primarily talking amongst themselves instead of using the public email list. As a result, one group announced a new project to redo the architecture of a major component. The other group, who had been developing that component, basically said “Hey, who are you to tell us what to do? That’s our code and we’ll take care of it ourselves, thank you very much.” Lots of flames then flooded the mailing list. In this case eventually things had a satisfactory resolution, but the upset never needed to happen. It turns out that the group originally developing the component had privately discussed that it should be redesigned, but that they didn’t have the manpower internally to do so. Had they posted their discussion, they would have gotten lots of volunteers and retained their leadership position. Meanwhile the other group privately discussed their problems with the current design, but didn’t make them public until they announced the new project. Had they posted their comments earlier, everyone would have come out for a redesign and they wouldn’t have created a rift with the original development group.

(A short article on the exchange is available at http://www.xml.com/pub/a/2000/07/19/deviant/index.html and reproduced in “Appendix D: Article from XML.com”. The actual messages could previously be seen starting at http://xml-archive.webweaving.org/xml-archive-general/2091.html, but unfortunately when last checked were no longer accessible.)

Posting etiquette

In general, your internal developers have to be more careful about what they post than would a random, outside developer. Even though they may think they are speaking as an individual, everyone else will take what they say as your company’s official policy. If they flame someone, it will not be seen as the action of an individual, but rather as one of “those arrogant guys” from your company. However it is really important that your developers do participate. Some folks will never trust your company, but they will trust Ken, Yarda or Stefan, even if they all work for your company.

As a rule then, when sending a reply to a public mailing list or newsgroup remember that it will be read by everyone, taken out of context, and viewed suspiciously by folks trying to determine what your company is up to now. If you have a strong emotional feeling when writing the message, it might be best not to send it right away, but instead wait a day and see how it looks then. Often the first draft lets you get any angry feelings out of your system, and the second draft is the one that you should actually send.

One final point about messages is not to be defensive. If someone attacks your company’s actions it is much better to wait a bit and see if, as often happens, another outsider will come to your company’s defense. It will carry more weight than if the same message came from an employee of your company.
Types and number of mailing lists/newsgroups

Different mailing lists or newsgroups can be used for different purposes. A special moderated list for major announcements that only receives infrequent posts will be subscribed to by many users and developers. An unmoderated list with frequent postings on technical issues will have a smaller audience. In general it is better to have too few mailing lists, than to have too many—would you rather be at a party where everyone is packed into one room, animatedly talking, with groups moving into nearby rooms when appropriate, or a party where some rooms have only one or two people doing or saying nothing, and the rest are empty? The analogy to a party is also apt since you will need to have at least one person per list who acts as the host. This is a very important job that must be done well, since it helps to set the tone for the community. Part of the host’s job is to actively discourage flaming in order to make the list a safe place for people to post to.

A small project may only need a single mailing list for all project related discussions. For a large project each module might have several mailing lists. When the number of messages sent out each day grows too large, the people on the list will call for splitting it into several lists. Make sure though that it’s not just a temporary increase in traffic. A single thread that lots of folks chime in on can easily double the number of messages.

You will need a mailing list for users. Free software generally does not have a customer support hotline that users can call when they have a problem. They need a mailing list or newsgroup where they can post their questions. Developers or other users can then post a reply to help them out.

In order to encourage people to answer other folk’s questions, the Jini team had a policy of waiting two days before answering a general question. Those that could only be answered by the core team, mostly “why was this done” questions, were answered immediately. This worked very well and has resulted in a mailing list where community members naturally answer most posted questions with the core developers answering only the more difficult, technical ones.

It is important to keep an archive of each list. This is useful for new developers or new users so they can see if a particular issue has already been discussed. It’s also useful as a group memory. Be sure that it is easy to search the archives.

Project documentation

Besides the normal end-user documentation required for any software product, an open source project needs to have good internal documentation for developers. You want to make it as easy as possible for a new developer to learn their way around the source code. The easier it is to learn enough to get started, the more developers you’ll attract. Conversely, if the internal documentation is poor or nonexistent, many potential developers will become frustrated and give up.

Those developers who are working on the project as part of their normal day job will be prepared to devote the time needed to come up to speed on how the code is organized and how it works. For them poor documentation is just business as usual, but for someone with only a few hours of free time in their off hours who just wants to fix an annoying bug or make a minor enhancement, the quality of the documentation can make the difference between success and failure. Initially the new developer needs to be able to locate the relevant locations in the source code and learn enough about
how that code works to be able to modify it. If they are successful then they are likely to do more
work on the code. A web-based tool that can help developers search the source code is lxr, the Linux
Cross-Reference tool (http://lxr.linux.no).

This means that all developers that contribute to the source code need to be encouraged to docu-
ment their code. Someone also needs to write and keep up-to-date design documents—especially
needed is a high-level description of the software architecture. This could be an additional job for
any technical writers that are available to your project. Mailing list archives are also a good place for
new (and old) developers to learn about various design decisions.

Another important document is a project roadmap that describes the current development plans for
the overall project and the individual modules. The roadmap reflects what the core developers and
module owners plan to work on based on discussions on the community mailing lists. The roadmap
allows developers and users to get a sense of what changes they can expect and when they might
happen. They can either join in the official work then or, if their particular needs are for other
changes, they know that they need to organize additional efforts. Many developers will decide what
they will work on by consulting the project roadmap, so it is vital that the roadmap be kept up-to-
date. The roadmap, and the community discussions about what features are part of it, helps to focus
your project and give it direction.

You will also want to create wishlists for the overall project and for each individual module. The
wishlists are a good place for a developer to look when they want to find something interesting to do
that will help the project. The process of creating the wishlist encourages users to speak up and par-
ticipate in the design. Involving users as designers is essential if the project is to be really successful.

Your users will also need documentation. And it is likely that some of them are willing to help write
it. The user group mailing list is an excellent source of information; organizing information in it into
a FAQ is a good first step.

If you have professional technical writers working on the project, they will probably be the module
owners for the documentation. Just as with code, others can send them documentation suggestions
and corrections, but they are the ones to decide what goes into the manual. Of course, as with a code
module, a volunteer who makes a number of good suggestions can earn the right to edit the manual
directly.

Finally be sure that the documentation you provide is in a format that people can easily read such as
plain text or HTML. Don’t require your users and developers to need special software to read the
documentation. Especially software they might need to pay for!

Project web site

In addition to mailing lists each project needs a web site where prospective users and developers can
find out about the project, and where news about the project can be posted. The web site should be
the portal to all aspects of the project. The site can have user guides, tutorials, archives of the mailing
lists, and other documentation. And of course there should be a download page where folks can
obtain the latest version of both the source code and binaries of it ready for use. There may also be
pointers to web pages for commercial products associated with the project, though the main site should not have a strong commercial feel to it.

When someone first hears of your project, the project’s web site is where they will go to find out about it. The web site will help set their expectations about your project: What is the current project status? How can they download executables or source code? Does it seem professional? Are new developers welcomed? Is it a real community where they can fully participate, or is it more like a user group where their participation is limited? Is it an active effort or does it look dead? Why should they get involved? Is there help or tutorials available for beginners? The way the web site answers these questions will dictate whether they want to get involved as either a user or developer, or if they just leave.

For a small project the web site might be just a few pages. For a large project the site might be quite large, with each module having its own set of pages. Maintaining the web site and making sure the content on it is kept current is another infrastructure job that must have a person assigned to it. For large projects an additional person is needed to be the editor-in-chief.

We will say more about the project web site in the section on creating a community of developers.

Bug database

Bugs happen. Being able to keep track of outstanding bugs is a must. The bug tracking tool you choose should be as easy as possible for developers to use, otherwise they may not. Some developers will prefer a bug tool that they can use via email so that bug reports are mailed to them and they can reply via email. Other developers will prefer a web-based bug database.

Since users are a prime source of bug reports it needs to be easy for them to report a bug. Keep in mind that they have already suffered by discovering the bug—they may have lost their work and undoubtedly lost time—so don’t make it painful for them to submit a bug report too. Also the more encouragement you can give to users to report bugs well, the more testers you effectively have. It may be good practice to have a developer responsible for reading the user mailing list and filing bug reports for problems reported there. It may make sense to have a separate mailing list just for bug reports.

The bug database may also be a good place to keep track of suggested enhancements. Any developer should be able to record an idea for a future improvement. Module owners should make sure that good ideas that come up in the mailing list are also recorded. Periodically the suggestions recorded in the bug database should be used to update the wishlists for the overall project and for the individual modules.

An example of a bug tracking system used by various open source projects is Bugzilla (http://www.mozilla.org/bugs/). It was originally created for the Mozilla project and the source for it is freely available. The NetBeans open source project is currently using Issuezilla, which is based on Bugzilla.

The work to maintain the bug database is another infrastructure job.
Software lifecycle

We've been looking at some of the infrastructure that supports an open source project. Now let's look at some of the activities that need to happen as the project progresses.

Planning and decision-making

All of the planning and decision-making for an open source project should take place either on the project's public mailing lists or at public community meetings. Small groups can, and should, get together in private to work up proposals and suggestions, but as soon as possible those need to be opened for community feedback.

Public discussion will generally take longer to make a decision than a proprietary development group would, but since the diversity of the viewpoints will be greater for an open source effort, the resulting decision will likely be of higher quality. This can translate into a shorter overall development cycle, since subsequent work will not as likely need to be discarded because the real issues came up after, rather than during, the discussion period.

With public discussion it is crucial that someone be sensitive to when the discussion seems to be winding down. At that point it is important to post a wrap up message that summarizes the main issues and the consensus on what should be done. Part of the wrap up is to list who has volunteered to actually do the work. Often the person who initiated the discussion will be the one to wrap things up, but some developers—especially those used to working in a hierarchical company—will expect someone else to make a decision and tell them what to do, which is not the way open source works.

One of the most common reasons for a software project to fail is because it does not meet the needs of its intended audience. This is much less likely to occur if the actual users of the project can join in the discussions where the project's functionality is being defined. For a really successful open source project, it is probably the users that started this discussion in the first place. Once the need is clearly articulated and possible solutions have been debated, only then can the developers work on a plan to implement a solution. Or if there are several proposed solutions, then the developers may choose to experiment by implementing trial versions of each, so that the users can try them all out to decide which is the best. Note that this process is called user-centered design or evolutionary design and is too seldom practiced by proprietary projects; the open source process naturally encourages a user-centered approach.

Design decisions, and the rationale behind them, need to be recorded in the project's design documents. There should be a person assigned to keep them always up-to-date. Scheduling decisions should be recorded in a project roadmap document. As mentioned above, the roadmap allows developers and users to get a sense of what changes they can expect and when they might happen. For a developer looking for a way to help, the roadmap can point them at work that needs to be done. It is very important that the roadmap be kept current; it is the canonical source of decision information on the project.

Scheduling takes on a very different flavor for an open source project since many people are volunteers. Internal developers can be assigned to tasks based on your company's priorities, but outside developers choose what they will work on and set their own schedules. We will say more later about
how to do a release, but two points should be mentioned here. First, for a normal company-run project the features that will go into the next release are defined, a schedule is made up and the features are then coded. For an open source project this process is reversed: new features are implemented and when there are a significant number of them, then a release date is set. Second, setting a release date will motivate people, who are working on almost finished modules, to get them done in time for the next release. This process of setting a release date and then including in the release only those modules ready by that date is sometimes referred to as the “train model.” Any modules not ready must wait for the next train, i.e. release.

The reliance on volunteers is another reason that it is important to assure that the project uses good modular design to enable decentralized development. In his book *Open Source Development with CVS*, Karl Fogel points out that “free projects optimize in favor of a distributed burden, lessening the vulnerability of the module to any one person’s schedule (or lapse in judgement for that matter).” [pg. 147]

Deciding what internal developers focus on is a matter of private planning. However if they are all working on matters peripheral to the main community activity, that can be bad—project leadership requires working to help achieve community goals. Leadership is not always important. For example your company might join an existing open source project with the sole aim of porting it to Solaris or adding a few features. But for an open source project that your company initiates, ignoring the wishes of the community can be fatal: at best your company will be seen as irrelevant and the project will fork, while at worst the community will cease to participate and the project will be seen as a public failure that only your company cares about.

In the end, it’s the people who write the code and who integrate the contributions that have the final say. The phrase “show me the code” often comes up in open source project discussions.

**Integrating contributions**

In a successful open source effort, developers will be contributing bug fixes and new features. Part of the job of every module owner is to review these contributions and integrate them into the code base. Accepting a contribution may involve some work, but is fairly straightforward. Rejecting a contribution however requires some sensitivity.

Some proposed new features may introduce more complications then they are worth or they may be questionable in the first place. The module owner needs to ensure the quality of the code they are responsible for and sometimes will need to reject a contribution. For the health of the community though, the person making the contribution should not feel rejected. They have donated their time to make their contribution and you want to encourage them to keep working on the project. Requesting that they make some changes to make their work better is one possibility. Another is to have an official part of the CVS archive for non-standard contributions; others can see them and incorporate them in their builds if they want to. Many open source developers have gone on from receiving constructive rejections of their work to later become productive project members. A healthy community helps to educate and develop its members.

Module owners must also be aware of the attitude of the community to their decisions. The module owner is allowed to make decisions only as long as they have the community’s respect. Their author-
It is largely based on merit. If enough other developers disagree with the direction they are taking the project they will either be replaced or the project will split into two factions—that is the code will fork. It becomes even more interesting when a company sponsors an open source project, as it will likely assign one of its developers to be the module owner. The module owner then has a manager to answer to in addition to the community. Moreover that manager may or may not be responsive to community feedback.

**How decisions get made varies among open source projects**

The majority of open source projects have less than twenty developers working on them and rely on a single person, usually the originator of the code, to act as a “benevolent dictator” making the major decisions. For larger projects decisions are often delegated to the senior developer in charge of each code module. Other open source projects use a more democratic method. For example the Apache Group votes on changes to the software and on who can become a member of the group.

We’ll say more on governance in the section “Who’s in charge?” on page 59 when we discuss community issues.

**Code reviews**

One aspect of integrating changes from developers is that the module owner needs to do an informal review of the code before accepting a contribution or bug fix. But who reviews the module owner’s code? Actually all of the code is continually being looked at by various developers. If there is code that is badly written, inefficient, confusing or buggy, then someone is likely to rewrite it. Discussion about implementation issues on the project mailing lists is another place that code review can take place.

An open source project effectively has an ongoing, informal peer review of its code. This is what allows there to be a continuous evolution of the code. A proprietary project is much more likely to have important sections of code that only the original author has ever looked at.

**Daily builds**

One of the most crucial aspects required for an open source project to be successful is that a developer can make a small change to the latest source code, compile it and see the outcome of their change. If they can’t do this, they will be much less motivated to continue working on the project. This is why it is important to have working code that does something useful before asking developers to join a new open source project. However a problem can arise if someone has checked in code that includes fatal bugs or, even worse, causes the code not to build correctly; then no one else can test any of their work.

It is vital that there be someone who is responsible for making sure that the build is not broken. This buildmaster needs to identify the cause of any problems and fix the build as soon as possible, probably by reverting the bad files to earlier good versions. The buildmaster then needs to contact whoever checked in the bad code and get them to fix it.

The Mozilla project has a special tool called tinderbox (http://mozilla.org/tinderbox.html) that consists of a farm of machines whose sole purpose is to continually check out and build the source.
tree, combined with a continually updating web page that shows the status of the builds. This way, people can know when it is safe to check in new changes.

To facilitate testing it is important that the latest build (executable) be available for download by the testers. You want the smallest delay possible between when a developer checks in a fix for a bug, and when the testers can download a new version containing that bug fix. The more immediate feedback everyone can receive the better. In the early days of Linux, Linus Torvalds on occasion made several releases of the Linux kernel in a single day. That’s giving users and developers immediate gratification.

It is also very important to have the most recent, stable build available for download. This is for your cutting edge users who want the latest features that have been initially tested, but have not yet gone through a full release cycle.

Testing

In a very real sense everyone who uses an open source application is part of the QA effort. As mentioned above when describing the bug database, the easier you make it for users to report a bug, the more likely they are to do so. Moreover, in addition to reporting the bug, some developers will even track down the cause and submit a fix for the bug.

For some open source projects there is no formal testing; all the bugs are reported from actual use. For other efforts, testers are recruited as part of the release cycle. In some sense people are volunteering to help test code whenever they download the latest build or the last stable build. The cutting edge users and developers are the first line of bug finders.

Looking at the quality of Linux, especially in comparison with other PC operating systems, one sees how successful this user testing can be. Another example is that the early users of Mozilla’s alpha and beta releases have been reporting about 1,000 bugs per month.

Having users doing the testing has extra benefits since the testing is not just that new functionality has been correctly implemented, but also that the functionality meets the user’s actual needs. This is combining usability testing with QA.

It’s important to thank your testers. Their efforts do not get the same visibility as developers contributing code or folks who write documentation. They should be acknowledged somewhere on the project’s website and also in a README file that is part of the standard distribution.

For a larger project with more resources, doing some form of automatic regression testing can really help make sure old bugs do not recur. If full-time QA people are available to do more formal testing then so much the better.

Releases

In one sense every time someone checks in a change to the source code, that is a new release. This is why open source projects talk about continuous releases. For the active developer this is great: They are guaranteed to be working on the latest code. They won’t waste time fixing a bug that someone
else has already fixed. Other people can start using their contributions immediately (and report any bugs they find in those contributions).

For users continuous releases are less desirable. Users want some stability to the programs they rely on. However how much stability is desired varies from one user to the next. Some want to be able to use the latest features, while others want something really solid and bug free. When a newcomer wants to know about your project, it should be easy for them to locate and download an executable that is known to work and be well tested. That gives them something to try out that will provide them the best initial experience possible.

To satisfy these conflicting needs what is often done is a series of frequent, small incremental releases using code that has been mostly debugged, with infrequent major releases after the remaining bugs have been discovered and fixed. So the more adventurous users become the QA team to flush out the bugs in the minor releases.

When to do a major release is determined by the current state of the code: Have a number of serious bugs been fixed? Have significant new features been added? Has it been a long time since the last release? If the community decides the answer is yes then the project enters a release cycle aimed at creating a stable version suitable for release.

The release process for an open source project is very similar to that used for proprietary products. The main difference is that the open source process is looser. For example if the code for a new module is fairly stable and does useful things, then it may be included in a release even though the documentation for it is slim or nonexistent.

The job of coordinating the release process is often handled by selecting someone to be the release manager. This includes recruiting testers, coordinating the testing process, and even making sure that the testers are properly acknowledged afterwards.

The release manager’s job also includes helping to decide what goes into the release and what is not yet ready. This generally involves a code freeze, so new functionality is not being added. Once it is decided to do a new release, many developers who are in the midst of writing a new module will be motivated to finish it quickly so that it can be included in the release. Motivating developers is good, but finishing the implementation of a module is not sufficient; it also must be debugged so that it is stable. Part of the release manager’s job is to allow only stable code into the release. Even some bug fixes may not be included if the benefit of fixing the bug is outweighed by the possibility of new bugs that may be introduced by the fix. During the release cycle is not the time to make big changes to the code.

Unlike many proprietary projects, ongoing development is likely to continue during the release process. Developers will continue to work on new and experimental modules that will not be included in the release. Projects that use a source control system such as CVS can start up a branch for the release activity while normal development continues on the main trunk.

When most of the known bugs have been fixed and the release is becoming stable, it is often a good idea to put out a beta release. More people are willing to try out a beta version since it has already undergone substantial testing. This second batch of testers will help catch the remaining bugs.
Finally when the last major bugs are fixed the release is ready to be packaged up for distribution and announced to the world. It is important that every release be given a unique release number so that everyone knows which is the newest version, and so bugs can be reported against the correct version of the source code. Some projects, such as Linux, have adopted the convention of giving even version numbers to stable releases and odd numbers to untested ones. So in August 2000, the latest stable Linux kernel was version 2.2.16, the development version was 2.3.66, and the beta for the next stable kernel was version 2.4.0-test6.

A company like Red Hat makes its living from packaging up the major Linux releases and selling them to users who value stability, ease of installation and product support. If your company plans on offering a branded product based on the open source code, then it would be following a similar model.

Support

If one looks at any of the open source licenses it is very clearly stated that open source software comes with no support whatsoever. This is generally discussed as an opportunity for third parties to sell support, and indeed companies such as RedHat make good money doing so.

But the support story is not so clear-cut. In fact the greatest source of support is from the users and developers working on the project. These are the people who care about the software the most, and also the people who know it best. In a successful open source project, the main mailing list is where people can ask questions about problems they are having and quickly get answers. Some of the answers may not be the best, but generally with a little persistence people do get the information to solve their problems. As long as people realize that they are asking for help, rather than demanding support, then the mailing list can be a major benefit for users of open source software. For those users that need someone to solve their problems for them, purchasing support from a third party is the way to go.

This is another way that user feedback can help improve the software. Developers get to see the problems real users are having and modify the code accordingly. It's also just a small step from asking how to do something, to suggesting something useful that the program could do.

Making it happen

As you can see lots of work is needed if an open source project is to be successful. Here is a list of some of the jobs that you must commit resources to:

- Evangelist/community coordinator to encourage and coordinate developers, get publicity for the project, increase community involvement, host mailing lists, and in general just keep the project moving along. We'll say more about these activities in the section on creating a community of developers.

- Module owners to be responsible for the development of the code and to integrate contributions and bug fixes from other developers. They also need to participate on the project mailing lists.

- Infrastructure support for the CVS archive, mailing lists, bug database, and web site.

- Web site editor to keep the web site alive with new content.
• People to document the system architecture and record reasons for design decisions.
• A buildmaster to oversee the build process and fix problems with broken builds.
• A release manager to coordinate release activities.

Going open source is not a way to get something for nothing. It takes real work to make an open source project successful. But putting in the required effort can yield a project that grows much more quickly than if you had done it all by yourself. At the least you should be able to double or triple the number of developers working on it. For example a year after going open source, the Tomcat project had 31 developers with commit access, only 9 of whom were Sun employees.
BUILDING A COMMUNITY

Ok. You've announced your project. You've got a web site, open mailing lists, a CVS archive, and a bug database. Your developers are standing by, eager to join in any discussions, and ready to accept bug fixes and new features. What do you do next?

What you want is a healthy community involved with your project. This involvement takes many forms. First you need users, people to actually use the software you are creating. John Ousterhout, the creator of TCL/TK, has said:

If an open source software package has a large enough number of users (where “large enough” seems to be around 5,000–10,000), you can be confident that it is functional, reliable, and fairly well documented. To see why this is true, consider the users. If they are non-expert users, there’s no way that 5,000–10,000 of them will use a package unless it has these properties (furthermore, the developer would go crazy dealing with all of their questions and bug reports). On the other hand, if the users are expert and there are that many of them, then if there are problems they will simply fix them, which they can do since the package is open-source.

Having a large number of users makes things real and applies pressure on the developers. If you don’t have users, you won’t get developers. If you don’t have users, then no one really cares about your project anyway and it will not be successful.

The distinction between a user and a developer will depend on your project. For a project like Netscape’s Mozilla browser there’s a big difference: Users run the browser; developers write the code that is the browser. For a project building a platform like Java there is less of a distinction: Users write Java programs; developers extend the classes in the official Java packages. Someone writing a Java class library is somewhere in between.

Besides just using software, which is of course the whole point of developing it, the users also help in many ways. First, they provide support to other users by discussing how to best use the software in their work, by helping to write product documentation and by answering questions posted to the various mailing lists. Second, they can contribute any user-level customizations they make to the product, such as templates, macros, or preference settings. Third, they can help shape the future development of the product by suggesting improvements, identifying problem areas, and participating in design discussions. Finally, they help test the product by reporting the bugs they encounter; for most open source projects users are the QA department.

Some of your users will have sufficient technical skills to also work on the source code. They may likely start out by contributing a fix for a bug that was bothering them. From there they may get more involved by contributing code for a minor improvement or participating in design discussions on the mailing lists. Then they may join in the ongoing work being done by the core developers, possibly earning the right to commit code to the CVS repository. Finally, they may become a module owner.

There is a tendency in open source projects to focus on the code, with the result that anyone who is not a developer is often treated as a second-class citizen. The expertise and energy of users is a huge resource just waiting to be tapped.
Keep in mind that your users and outside developers have diverse backgrounds. Some may be computer novices, while others may have more experience than your most senior people. Some will have worked on other open source projects and have firm ideas about how they are run; for others your project will be their first involvement with open source and they will have only hazy ideas of how to participate and will need to be educated in how open source really works.

As with all online communities there is a natural progression from newbie, to regular member, to knowledgeable elder. The tone of the community you create will determine whether people's involvement will grow or not. If suggestions to the mailing lists are routinely flamed, or if code contributions are harshly rejected, then people will not want to participate. If the community encourages participation, even when the contribution is not very good, then more people will be willing to participate and the level of everyone's participation will generally improve over time. Many open source projects report that some of the current core developers started out by making somewhat clueless or half-baked contributions, but, by receiving encouragement and constructive feedback from other community members, they improved to the point of becoming key participants.

Very roughly you can expect that as the level of participation increases, the number of people at any given level will be an order of magnitude less than the next lower one. For example, over 40,000 people have downloaded the Jini source code, but only around 4,000 have subscribed to the jini-users mailing list. Of those around 400 people might post an occasional message, with most of the posts coming from a core group of around 40, of which close to 4 are considered as elders.

**Building trust**

If you were just a private individual starting up a new open source project, then you might be able to just announce your project and then sit back and wait for contributions to start rolling in. For a company though, especially a large company, there's a suspicion that must be overcome before outside developers will feel comfortable contributing to the project. Basically you have to earn their trust and prove that you do not have hidden motives. There is a big difference between creating a successful users group and growing a successful open source project.

If a potential developer feels that you will be making money by selling the code that they contribute, then they are apt to be offended, and rightly so. And if they think that they might need to pay you in order to use a product that incorporates their contributions, then don't expect to receive many contributions.

You must clearly communicate what your business model is and that any money you make is for value that you are adding, e.g. a higher quality product, or additional proprietary features. If you really are profiting from the outside contributions made to the project, then, even if you can explain how they also will benefit, it will be an uphill battle to recruit outside developers.

If developers must pay licensing or royalty fees, they will be less likely to contribute back to the project. If the license itself is too restrictive or seems to favor your company too much, outside participation will suffer. If publicity seems to slight contributions from external developers, then those developers will create negative press and leave.
The more you actually live a true open source lifestyle, the easier it will be to earn the trust of your outside developers. Any incidents where your developers, or other people in the company, put a business-as-usual spin on things will make it harder for the project to succeed.

For example, Sun’s reputation with the open source community increased with the release of NetBeans under the Sun Public License and even more with the release of StarOffice under the LGPL and the support of Sun to the GNOME project. This overcame some of the bad feelings many developers had towards Sun because of the way Sun has retained control of Java. However if any future problems arise between Sun and outside developers in the NetBeans or OpenOffice projects, the old suspicions will immediately come out and need to be addressed.

It is vital that the your company’s developers participate in the project’s mailing lists. This is how the community will come to know who they are and what sort of person they are. Remember that trust is built up between individuals. How your company’s developers interact with community members will set the tone for the entire community. Since initially all of the module owners will be internal developers, how they accept and reject outside contributions will either create an atmosphere of real community or convince outsiders that it is an effort only by your company.

As soon as possible you want to have qualified outside developers receive check-in privileges. So keep an eye out for any outside developers who are contributing reasonable code and after they make a few contributions consider granting them CVS commit access. The sooner some module owners are outside developers the better. So welcome anyone contributing what could be a new module. These are signals to everyone that it is a true open source project. For example, a defining moment for the Tomcat project was the 3.1 Release where the release manager was from IBM and not from Sun. That let everyone know that Tomcat was no longer a Sun-only project.

**Establishing credibility with the open source crowd**

Beyond the day-to-day activities of your project, there are some other things you can do to establish better credibility with the larger open source community. The larger and more visible your project is, the more important it is for people working on other open source projects to see you as being part of this larger open source community.

First off it will help if you are using a license that has been blessed by the more well known spokesmen for open source, such as Richard Stallman, Tim O’Reilly, or Eric Raymond. If your project uses the Sun Public License or the GNU GPL, then they will be able to go on record as saying your project is true open source and a good thing. If your project is using a version of the Sun Community Source License, they and the press will be sure to point out that your project is not open source.

Second, you want to establish connections with other open source projects. Does your project use tools or modules developed by another open source project? Will your project produce tools or modules that another open source project can benefit from? The OpenOffice project started with the intent to integrate its office productivity suite with the GNOME desktop, so it should be involved in the GNOME community, possibly contributing code and bug fixes back to the GNOME project. Every open source project developing in Java could be using the NetBeans Java IDE to make their work easier. The more your project is seen as supporting other open source efforts, the more you will be considered a good member of the larger open source community.
You also want to establish dialogs with open source developers not involved with your project. Have your internal developers both read and post to places like Slashdot (http://slashdot.org). The more people see individuals from your company participating in open source forums, the harder it will be for them to see your company as a faceless large corporation. It is fine for your company's developers to express whatever views they have, but they should strive for some restraint and avoid too much flaming. As mentioned in the section on “Email etiquette,” they need to remember that even though they are posting as an individual, people will read their note as being your company's official policy. When your company's actions are attacked, they should wait a bit to give outside people a chance to defend your company, which is more effective then if it comes from an employee of your company. Of course if the criticism is valid they should immediately acknowledge it and address how to fix things.

Because many open source developers are suspicious of large companies, recognize that it will take time to earn their trust and overcome any past mistakes.

Growing a community of users and developers

You have a basic choice in whether to sit back and let potential developers find you, or whether to go out and actively try to recruit new participants. In either case once someone starts to contribute to your project, you need to actively encourage them.

If you decide to go out and look for folks then you need to know what type of person you are looking for. Who is your target audience? Or rather target audiences, since you need to grow both the number of active developers and the number of people using the resulting software.

To attract users you need something that they can use. If all you have produced are some major pieces that would go into an actual product, then there may not be anything to use yet. This was a major problem with the Mozilla project when it first started. Without a working Mozilla browser there were no users and it was much harder to attract developers.

Once you have a working product you increase your users just like with any other product: You make it better—easier to use, more features, better performance, fewer bugs, and better documentation. And you market it. Since the source code is publicly available, there should be a free version that users can easily download and install.

A good user mailing list or newsgroup will also help grow the number of satisfied users, who will then help future novices. It is extremely important early on to help set a good tone on the mailing list. By encouraging group participation you help to grow a real community. Out of this user community will come your future contributors, documentation writers, evangelists, and customers.

When the project is started, having a single “communal” mailing list is best. Resist the temptation to overdesign by creating many lists before any community has developed. As the volume grows on the mailing list, the community itself will decide when it is time to create additional lists.

For a traditional open source project people start out as users and become more involved because they value the product and wish to improve it. Again your developers need to encourage people to participate in the design of new features and also in their implementation. For minor contributions
it can sometimes be frustrating to experienced developers who know that they could do it faster and better by themselves, but a few kind words now and then can really help.

Another source of contributors is other companies. If a company benefits from using an open source product, then it may be willing to devote resources to that project. Actively seeking other companies to partner with may be a good strategy for growing the pool of contributors. Note that when a company gets involved it may add many new developers who are focused on one or two modules. It is important that the module owner is not overwhelmed by their participation on the mailing lists and can promptly incorporate their contributed code and bug fixes. It isn’t good if they have to wait weeks for their changes to get into the official CVS code archive. As soon as possible some of their more experienced developers should be granted check-in privileges.

One other source of developers is universities. Providing assistance—hardware, software or funding—to a university research program can enlist the aid of large groups of students.

Many of these suggestions involve active work by core developers to work with and mentor people outside of your company. This can only be done by personal contacts, so it is important that your developers know that their performance reviews will include their community development work. If they receive the message it’s important to work with outside developers, but they are only evaluated on how much code they’ve written, then it is likely that they will minimize their community participation. Without strong outside leadership to pick up the slack, the community would then stagnate and the project would become an effort done mostly by your company alone.

Give folks a good initial experience

It is often difficult for a user to download and install a new piece of software. This can be due to a variety of reasons such as poorly written directions, the number of steps required for a successful install, or the difficulty of correctly configuring the software to the specific computer system. If your project is really compelling to potential users, then they will be highly motivated and willing to overcome whatever obstacles are in their way. For example, the first spreadsheet programs on the early personal computers created an urgency in many people who so needed the functionality of a spreadsheet that they were willing to ignore how difficult to use they were. Of course, if your project is that exciting then it may be that your business model should be based on commercial sale and not open sourcing it.

For most projects, the first obstacle a person encounters may be enough to stop them from going any further. If they never hear about your project or can’t find the project’s web site then the project has already essentially failed. If it’s not clear how to download and install it, folks will give up and look elsewhere. If they can’t figure out how to use the program to at least do something simple, or if it crashes on them right away, not only are they apt to drop it then and there, but they’ll also let other people know what a loser the program is.

It is really important to make sure that people have a good initial experience as they locate, download, and try out your project’s product. Making sure that the initial “out of the box” experience is a good one can be the difference between a successful project and one with no users. Make sure that it is easy for users to locate the correct version of the product to download. There should be a link on the project’s home page that will take them directly to the correct download page. It should be crystal
clear which download is the last stable release and which is the cutting-edge, buggy release that newcomers don't want. If there are known to be major problems with a release then you want to have a list of all of the major bugs or missing features posted somewhere that users can easily find—in fact they should see the list before they try to download, so they can decide if the release is even worth their time to try out.

Any product installation should be simple and straight forward. There should be a clearly written tutorial to get them started using the program. After they have had a taste of success with it they will be more willing to put up with and overcome problems.

Now trying to guarantee your users a good initial experience is certainly true for commercial products, but why should an early version of an open source project have to be so polished? It's certainly true that lots of open source advocates are sophisticated computer users and expect early releases from open source projects to be a bit rough. However, they only have so much time to devote to trying yours, and overcoming each obstacle eats into the time and energy they have available. Too many obstacles and they will give up and do something else. Remember they don't need your program—they already make do without it and are just interested in trying it out.

In some ways this is a good situation because it forces you early on to think about how people will actually install and use your program, instead of waiting until the end of the development cycle to tackle it, if then. Dealing up front with user concerns always makes for a better final product.

Creating a good initial experience also goes for developers. What sort of reception do they get when they make their first bug report, contribute their first bug fix, or initially post to the mailing list? Are they welcomed or flamed? Does a more senior developer mentor them or reject their effort? Even if their contribution is ultimately rejected, it is crucial that they feel that their effort was appreciated and that they were not rejected.

An example of how to encourage outside contributors happened early on in the Jini project. One of the first code submissions had major problems. Rather than reject it Ken Arnold—the module owner—basically rewrote it. Ken then talked privately with the original author about the need for the changes and then announced to the world that the author had made this great contribution to the project—without mentioning that it had been rewritten.

Other obstacles developers face include how difficult it is to understand and work with the source code: Is it modular? Is there readable system-level documentation of the program's architecture and data structures? Is the build process straight forward or long and complex? Do they need special tools to compile or work on the code? All are potential problems that can make them decide not to devote their free time to working on your project.

Having to overcome obstacles also eats into your project in other, more subtle, ways. Lots of energy on the Jini mailing list during its first three years has been taken up addressing problems all newcomers seem to face when they first try to run an example Jini service. The problems lie with the underlying complexity of how the operating system handles multicasting (or not, in the case of Windows 2000), the network configuration, having the correct security policies set in the Java virtual machine, and other such issues outside of Jini's control, but that are needed for Jini to function. Until the Jini project includes tools to automatically detect these problems, and then help users track down solutions to them, the beginner is faced with a major effort before achieving any success with the Jini
technology. Some make use of the Jini FAQ and mailing list archives to solve their problems, while many more post their plight to the main mailing list. Going into its third year the community continues to be incredibly supportive of new user problems, but it does take up time of more senior members of the Jini community.

A community web site

Peoples’ first exposure to your project is apt to be through the project’s web site. People working on the project will use the web site as a place to meet, to post their work, and to find out about the current status of the project. The information presented on your web site and how it is organized can help your project to be more successful. One very important job the web site can do is to help create a sense of community.

At a minimum your web site needs to have pages that:

- describe your project and its goals
- provide access for downloading the most recent releases, the latest experimental builds, and the current source code
- explain how to get involved
- tell how to sign up for the project mailing lists and a searchable archive of previous messages
- list the key developers working on the project and how to contact them
- list recent news items concerning the project

Larger projects will also have web pages for:

- tutorials and examples of how to use your product
- user documentation
- developer documentation, including an up-to-date project roadmap
- accessing the CVS archive
- reporting, searching and editing the bug database
- a list of frequently asked questions about the project (FAQ)
- descriptions of each module
- pointers to web pages for commercial products associated with the project

Many of the above listed items will be created by your community members either directly (e.g. someone writes an FAQ), or through community activities (e.g. online discussion of what milestones are in the project’s roadmap). Other pages report on what community members are doing.

Some projects will allow participating developers to create additional web pages, possibly giving them a personal directory to store files in where they can upload their own web pages or exchange files with other developers.
As you can see the pages on the web site need to cover a large and diverse range of material. This material needs to be organized to meet the different needs of the various parts of the community. So there may be one section of the web site dedicated to users and another more focused on developer needs. The project's home page needs to clearly direct visitors to the web pages appropriate for their interests.

Moreover the look and feel of the site will likely change as one moves from the very public home page to more specialized, remote parts of the web site. In a person's home there is an intimacy gradient from the public parts of the home to the more private regions visited only by family and close friends. Similarly the web site's outer face will usually be more polished and standardized, while further in, individual developers may use a variety of designs for the web pages that describe their projects.

As the project grows and changes, so too must the web site. The community itself should be a major source of suggestions on how the web site can be improved. Make sure it is easy for community members to suggest changes to the site and participate in its continual redesign. Some of this is easy to do by just posting on the web pages an email address for suggested changes. That is good mainly for minor corrections or adding pointers to web pages on other web sites. For more substantial changes it may be better to have a web site subproject that volunteers can join.

There is also a constant need to keep the web site fresh. New content in the form of articles, status updates, news stories, and usage tips must be added regularly. Many people do not subscribe to the project mailing lists, but rely on the project web site to keep them up to date on things. The web site is like a newspaper and if the same old articles are there week after week, then people will stop visiting. One important project role is that of web site editor. Just as a newspaper needs an editor, so does your web site. The editor needs to solicit articles from developers, gather news related to the project, highlight work being done, profile key developers, and generally make sure that there is a good reason for anyone interested in your project to regularly visit your web site.

Large projects, like OpenOffice, NetBeans and Jini, will want their own independent web site; they will also have the budget to pay for one. Smaller projects will want to join an existing web site that hosts open source projects, such as SourceForge (http://sourceforge.net). Sun has a web site for smaller Sun open source projects called SunSource.net (http://sunsource.net).

Who’s in charge?

As your project grows and develops, many decisions will need to be made. These will range from those affecting a single developer, such as whether a bug fix they have submitted is accepted or not, to ones affecting every developer and user, such as when to do a major release and what features will go into it. How these decisions are made and who it is that makes them will either strengthen or undermine your project’s sense of community.

If people feel that they are involved in the decision-making process and that their viewpoints are heard and respected then the community will generally accept whatever decision is made. If people feel that a decision is being rammed down their throats then they will object and, in the worst case, go elsewhere, possibly forking the source code and starting a competing project.
The exact decision-making process varies from one open source project to the next, but in many it is based on the idea of a meritocracy: those that have demonstrated their competency through their work on the project are the ones who make the decisions. In many cases the project lead, often the originator of the code, has the final say. Likewise, each module owner makes decisions that affect their module. This will work only as long as the “benevolent dictator” can maintain the respect of the developer community, otherwise the community will call for their replacement.

Some open source projects have the senior developers vote on major decisions. For the Apache Group, three positive votes and no negative votes is necessary for any decision. Moreover anyone voting no must give an explanation of why they are vetoing the proposal. A vote is also required to add a new member to the group of developers that can vote. The Apache Project guidelines and voting rules can be found at http://dev.apache.org/guidelines.html.

Note that even in a meritocracy there will be lots of politics at play. First, there is the question of who judges whether someone has merit or not. There is no objective test that a prospective contributor must pass. Instead they must somehow prove themselves to the current core group. If their approach to the project is sufficiently different than the core group’s then they may never be judged worthy. Second, having a good idea is not enough, it must be presented to the community in a clear manner that meshes with the current vision for the project. Even though a single module owner may make the final decision on whether to incorporate a new feature, being able to muster community support in favor of the feature can strongly influence the outcome. In both cases there is a tension between the group’s ability to maintain a narrow enough focus so as to be able to move forward, and being open to new innovation.

It should also be noted that a meritocracy based on the ability to write code has the direct effect of disenfranchising users—including users who are qualified developers, but who do not have the time or interest to contribute code to the project. Giving a say only to those who have contributed their time and effort has a certain appeal, especially if you are one of those people, but it has a negative result if the interests of those contributors is different from the pure users. For example, a programmer’s needs for a text editor are quite different than a person writing technical articles. So an open source project developing a new text editor could easily put all its energy into adding features that make it easier to write computer programs, rather than other features such as footnotes or a bibliography. That’s fine if the vision of the project was to create an editor for programmers, but if the original intent was to provide an open source replacement for proprietary programs like Microsoft Word then the project would be a failure. To be successful requires all the users having a direct say in setting the priorities for the project. Letting everyone vote on which bugs are most important to fix is a small example of this. In any case developers get the final vote by deciding what they will spend their time working on.

For an open source project where the majority, if not all, of the senior developers work for a single large company, the above decision-making processes can run into problems. Many open source developers are suspicious of the motives of large companies, and they will suspect the senior developers of caring more about their company than about the good of the open source project. This makes decision-making a problem until the outside developers get to know the senior developers as individuals and have reason to trust them. Even then people will perceive the situation as potentially unfair.
To counter this a more democratic process may be required. One solution is to establish a neutral governing board that has the final say, and to which anyone can appeal a decision. Both the Jini and NetBeans projects have done this, establishing boards where Sun has only 1/3 of the total votes and the rest of the board consists of well-respected community members. This has gone a long way toward gaining the trust of outside developers by showing that Sun cares about the success of the open source project and is not trying to pull a fast one. The Jini project has gone even further by putting major decisions to a vote of the entire community.

Jini is a simple distributed computing model based on the Java programming language. Among other things, it was intended as a model for services—small bits of functionality—to discover each other dynamically and create a network of interoperating program parts. These services could be housed within devices—physically separate computing platforms as far as Jini is concerned. Because such a model is new, the definition of each service is likely to require a maturation period, in which an initially immature service is defined and released, with perhaps numerous improvements subsequently being released. Service definitions are specified by Java interfaces: API's, in other words.

A problem facing the Jini community was how to maintain the consistency and quality of Jini services, where all services must conform to the core Jini specifications—it’s like TCP/IP—but where newly developed services need room to experiment and innovate. The community is working to develop a process that is fair, lightweight, and adaptable. It is lightweight because most people don’t need to use it—they just write up specifications for their new Jini services and try to get people to use them. Only when they want something ratified, or if they want to make a change to the core Jini services, do they need to have the larger community vote.

To be fair means giving individuals and small companies a say equal to that of large companies. It also means recognizing that a large company may have many thousands of customers using its Jini-based products and so it deserves a greater say than a lone individual representing just their own interests. To balance these conflicting needs members are divided into a general house and a commercial house. Both houses must approve any proposed standard. Acting as a sort of Supreme Court to oversee the process and further ensure fairness is a nine member, Technical Oversight Committee (TOC). The initial draft for this Jini Community Decision Process (JDP) is described in: http://jini.org/docs/Constitution.html. A more detailed version of the JDP was under development in August 2001 and can be viewed at: http://www.jini.org/docs/rules/.

One final point is that no current open source license describes how the community is run—if a project forks, one branch could vote on everything, while the other could appoint an individual to make all the major decisions. If people do not like the governance structure you choose for your project, they may decide to set up a separate community with a decision-making process that they find more acceptable.

Likewise the community has no direct say over changing the license—only the copyright owner of the code can do that. However community discussions on the project’s mailing list in favor of changes to the licensing can be effective. The Mozilla project is an example where community feedback helped to shape the initial license and where it has also prompted adopting a dual license structure. To give the community direct say the copyright could be assigned to a foundation that was responsible to the community, i.e. the foundation’s governing board is elected by the members of the community. Then the community would have ultimate control over making changes to the license.
Meetings

Mailing lists, web sites, and other flavors of online communications are good, but to have a real community it is important for community members to actually meet each other in person. Being able to put a face with a name, and to really interact with someone—hearing their voice, seeing their facial expressions—is what is needed to create a real sense of community. Meeting in person also helps to see people as individuals and to break down the barriers between companies.

Community meetings help the project to develop its own culture, which reinforces the community by helping to define a group identity and building stronger bonds between members. Each community will have its own flavor based on its shared values, philosophy, personalities of major members and other unique properties. Attending a community meeting gives members a shared experience. It helps to create a larger context by exposing people to aspects of the project that they would otherwise not have seen.

Meetings are excellent places to:

- discuss and vote on major issues—sometimes it is much easier and quicker to reach consensus when meeting in person
- have individuals or subgroups present their work to the larger community
- conduct demonstrations of new products or subprojects
- connect developers and users
- start up a new subproject—people can more easily share their enthusiasm when in person then when writing email messages
- acknowledge people who have contributed to the project
- perform informal market research and test out new ideas
- generate excitement for the project

So it is very important that your project schedule regular community meetings. These can be full events in themselves or just a part of some larger event. They can involve the entire community or just members of a subproject. If your project has working groups, committees, or a governing board that can be scheduled as part of a larger project-wide meeting, that will make it easier for individuals to justify travel to participate in them.

Three principles for community building

We will end this section with three basic community design principles from Amy Jo Kim’s book *Community Building on the Web* [pp. xv-xvi]. We highly recommend this book, which describes the strategies needed to create a thriving online community.

First, design for growth and change. Don’t overdesign your project up front. Start off small and focused. Grow in response to the needs of your community members.
Second, create and maintain feedback loops. You need to listen to your community in order to meet its needs.

Third, empower your community members over time. As your project “grows and matures, your members can and should play a progressively larger role in building and maintaining” the project.

Taken together, these three principles will help you to nurture and guide the excitement and energy of your users and developers so as to grow a successful project with an engaged community. The results will be much more than you could have done by yourself.
JOINING AN EXISTING OPEN SOURCE PROJECT

So far we have been discussing open source projects that your company starts and provides the leadership for. If an open source project that your work can build on already exists, then your needs are apt to be better suited by joining that project and not by trying to start a new effort. You would definitely want to join an existing open source project if your goal is to port an open source application to Solaris, or to add a new feature to an existing open source application or library.

Joining an already established open source project is much easier than starting your own project since someone has already set up the infrastructure (web site, CVS code archive, mailing lists, and bug database) and a community of developers is already working on it. You can focus your efforts on adding the new functionality you need.

License issues

Before joining an open source project you need to check that the license that they are using is compatible with your business goals. For example, suppose that you are planning to combine the existing open source code with proprietary code. If they are using a GPL-style license then you would not be able to do so. However, if they are using any of the other common open source licenses, including LGPL, then you would be ok.

Becoming a good community member

In theory your involvement can be as minimal as contributing a few bug fixes or small new features and that’s it. Generally your business goals will require you to do a bit more and you will need to actively participate in the project and establish yourself as a valued community member.

Each open source project has its own flavor, just as each Internet newsgroup or mailing list does. People are known to the community by their previous posts and contributions. It is very important that before you start posting to the project mailing lists, you first lurk for a bit. Read over some of the mailing list archives to get a feel for how the project operates: who is influential, what are the current major issues, and how are issues decided. You should also search the archives to see what has already been discussed about your particular areas of interest.

When you first post to the project to announce how you want to be involved, it is very important that you acknowledge the good work already done, and explain how you will be adding value. You want to avoid appearing arrogant. You may also need to alleviate people’s fears that your company is somehow trying to rip them off—sometimes just explaining why it is important to your company is enough to do that.

Keep in mind that you are establishing your reputation as an individual working on the open source project. Other people working on the project don’t care about your position at your company. What they care about is the quality of your posts and code contributions. The more they respect you, the easier it will be to work with them.
The open source development process

The key to open source development is that it is an open process. This is especially to your benefit when joining an existing open source project. By looking at the project's website you should be able to learn everything you need about how the project is organized. In particular you can read the project roadmap to get an understanding of what development is planned. You can identify the individuals responsible for the modules you will be working on. You can read the mailing list archives to get a sense of the project's history and character.

As you begin to work on the project, you need to make sure that you maintain this openness. Make sure that you are using the project's public mailing lists and not a list internal to your company. Involve the community in any major design decisions that you need to make.

Being open in your intentions is quite important in your dealings with the project's module owners. These are the people who have the final say as to whether your proposed contributions are accepted into the official code base. That being the case, it is important that you post a message announcing your intentions—what you want to do and how you plan to do it—before doing any major work. This will give the module owner and the other developers a chance to comment on your plans and make suggestions as to how it fits, or does not fit, in with the community development plans. It is likely that you may need to make your work more general or extend the functionality to satisfy the rest of the project. It is also likely that the revised plan will be better overall. Note that if all you want to do is implement an API to your own proprietary code, that will not get much, if any, community support.

Some of the discussion may become a bit heated; many developers express themselves quite bluntly. Don't let the flames get to you. Remember that many of the people you are dealing with are volunteers and some may be quite inexperienced. Your replies should be aimed at the community and not at the "idiot" who just flamed you. Refer to the section on "Posting etiquette" on page 41 for more on how to handle yourself when posting.

For more details of the day-to-day mechanics of an open source project, please see the section "How to do open source development" on page 39.

Getting your contributions accepted

To be effective when doing open source development it is very important to create realistic expectations on both sides. Above we talked about making sure that the community, and especially the module owners, had a clear expectation of what you want to contribute to the project. This greatly increases the chances that your code will be accepted when you finish it. Another factor in your favor is that most people running an open source project want their application to become popular and be used everywhere, so if your company is interested in modifying it a bit so it will be accessible to more people, you already have a leg up.

It is also important that you have realistic expectations about how the open source project functions. For example, it may release a stable version once a year and if it just finished a release cycle, then anything you contribute won't be officially released until the next year's stable version. Likewise if the project is in the middle of a release cycle, there may not be much community interest in discussing
the wonderful new features that you are planning on—you may need to wait until the release is done before people have the energy to discuss major new features.

Another potential problem you face is that the particular module owner you are working with may be very slow in accepting your contributions. This may be because the module owner is busy with other matters or doesn’t have much time to devote to the project. It may also be that your team of several developers is writing code much faster than the single module owner can digest it. If this is the case then you need to gain the trust of the module owner so that you can be allowed to check code in directly. It helps if you respect the coding style currently used in the module when you write bug fixes, patches or small changes.

A major role of the module owner is to guarantee the quality of the code base by screening contributed code. Some module owners are happy to share this screening function with developers who have shown that they are competent. While the module owner has the last word on what goes into their module, they are fine about letting delegated individuals check in the code when it is ready, as long as the general design had been discussed on the project’s mailing list beforehand.

Sometimes though, a module owner refuses to delegate and this can create a major problem. Some module owners insist on understanding every line of code in their module, which is fine for a small module, but may be impossible for a very large one. This is especially true for some original project owners, who never were willing to delegate responsibility to the various modules, but instead insist on being a bottleneck that all major changes need to go through. If this is the case and the module/project owner is unwilling to change, you may need to muster community support for sharing responsibility. Otherwise your only option is to fork the project. That is what happened with Emacs back in 1992: The XEmacs project came about when Lucid Inc. tried to contribute major changes to the display code and Stallman refused to accept it. For more details see: http://www.jwz.org/doc/lemacs.html. Nine years later the two projects still remain separate.
5. **GOING WITH OPEN SOURCE**

The aim of this part is to help you decide on whether open source is right for your project, and, if so, what steps you should take to proceed and what mistakes you should avoid.

**DECIDING TO DO OPEN SOURCE**

The first part of this handbook has described what is involved in an open source project, and now you are in a position to make an intelligent decision if open source is appropriate for your project. Here are some of the questions that you need to answer:

**Do you buy the lifestyle?**

Are you willing to do your work in the open and collaborate with people outside your company? Will you use public mailing lists instead of internal ones? Are you willing to spend significant amounts of time discussing design and direction with outside community members? Is everyone working on your project willing to work this way? Are your managers (and their managers) willing to change how your project is run? Are you willing to give up control and let the community make decisions?

**Does your business model support it?**

What are your business goals? Are you trying to make a platform ubiquitous, gain developer mind-share, create a for sale product, sell support or services, or what? How does open source help you achieve these goals? How will your business model develop over time? Do you need to provide support for your product? If so, how will you support it?

**Can you legally open source your code?**

Does your company own the rights to all of your source code? Does it include any third party code? Does it make use of any third party intellectual property (e.g. patents or trade secrets)?
Is your product/project something that the open source community will care about?

Who is your target audience? Why will they care about your project? What do they get out of it? What licenses are they currently using for related projects? Will your license be compatible with theirs? Will you be competing with an existing open source project? Is there a working version of your code that people can use right now, or is it still in development?

Is your source code open source friendly?

Large monolithic applications that are poorly documented require lots of study before a developer can make even a small change. How easy is it for a new person to learn their way around your code base? Is your code modular? Is there sufficient project documentation? Are you willing to refactor the code? Are the main developers of the code willing to answer questions?

Can you get the resources?

Doing open source requires some basic infrastructure. Can you provide the people to maintain the CVS archive, project mailing lists and archives, web site, and bug database? Can you support a community coordinator, a web site editor, and a buildmaster? Do you have a budget so you can pay to have your project’s web site hosted?

Do you have realistic and measurable goals?

How will you know if your project is a success? How can you convince your managers and others at your company that you are achieving your goals? For an open source project success is more than just when you release and what features were implemented. You also need to include the health of the community. What numbers are meaningful for your project goals (e.g. market share, number of active outside developers, number of user contributions, or amount of email sent to mailing lists)?

What support do you need from your manager?

Is your manager willing to allocate sufficient resources? Will they assess you on the success of the open source effort? If the open source community needs more time to make a decision or get a release out, is your manager willing to slip schedules, or are you being told you must meet deadlines for your company’s internal schedules at all costs? How about other managers at your company that are in charge of projects that depend on the results of your project?
HOW TO PREPARE TO DO OPEN SOURCE AT YOUR COMPANY

In case you turned to this section first, please look at the questions in the previous section “Deciding to do open source” on page 67. You will need to have answers to those questions before you can complete this section or the next.

Resources at your company to help you

The Open Source Program Office, which includes Danese Cooper and Eric Renaud, exists to help you launch and maintain your open source project. They have started a web site at http://opensource.eng and an internal mailing list, opensource@sun.com. They can be very helpful in advising you on what license to use, in negotiating with hosting providers (including CollabNet and SourceForge) to host your project’s web site, in training your team, and in many other matters. Danese has been involved with every open source project released from Sun since 1998.

The authors of this handbook—Ron Goldman and Richard P. Gabriel—are also available to advise you. We provide expertise in helping you to develop your business model and create a real community around your source code.

You should also talk with the folks who are running or participating in open source projects, such as NetBeans or OpenOffice. They are the ones who are in the best position to help you understand what you are getting yourself into.

Educate yourself and your team about open source

The best way to prepare to do open source is to participate in an existing open source project. You should find an open source project that interests you and join it. Download the project’s code and play with it. Follow the mailing lists. Maybe fix a bug or contribute a small modification. The other members of your team should also participate in open source projects that interest them.

Doing so for a few weeks will give you and your teammates a good idea of what an open source project really feels like. Also you will all gain an appreciation for what it’s like to be an outsider. In your interactions with the project’s module owners and core developers were you treated courteously? Were your suggestions/contributions dealt with promptly? Did you feel accepted/encouraged? This should give you plenty to think about when you are running your own open source project.

Another way to learn about the open source culture first hand is to join in discussions on places like Slashdot (http://slashdot.org). Read the postings to see the range of people who make up the open source community, from clueless newbies to experienced professionals. Also post your own comments on topics you personally care about. Please avoid flaming. If you post with an email address from your company, you might check if your comments are treated differently. Don’t feel that you need to defend your company—that’s best done by people who don’t work for your company.

Not all developers are suited for the open way that open source projects must be run. Getting a taste from participating on someone else’s project may aid in identifying if you, or someone on your team, is not appropriate for open source.
Educate your manager about open source

Developing open source software is quite different from how proprietary software is created. You will definitely need the support of your manager if you are to succeed. So your manager, and probably their manager, will also need to understand how open source works and what its advantages and disadvantages are.

You need to talk with your manager and make sure that he or she does not believe any of the various myths and misconceptions about open source. (C.f. “Common open source myths, misconceptions & questions” on page 8.)

Does your manager understand how an open source development process will change how you do your work? Does he or she see how the business model for the project is supported by open source? Most importantly does your manager agree that the benefits gained by creating an open source community around your project is worth more than your company keeping total control?

Develop your business model

If you haven’t already developed the business model for your project, now is the time to do so. Your business model should make it clear what your business goals are and how they fit with your company’s goals. It should explain how using an open source development model helps you achieve your goals. For more information please refer to “Business reasons for choosing open source” on page 17.

Design appropriate review criteria

You and your manager will need to establish proper criteria to measure the success of your project. You need to include both the traditional metrics on the code itself, and also ways to measure the success of the community working on your project. Based on your business model you need to decide what numbers are meaningful, e.g. release dates, feature sets, market share, number of active outside developers, number of user contributions, or amount of email sent to mailing lists.

You and your manager also need to meet with your Human Resources representative to design an appropriate performance review that measures developers on both their contributions designing and writing code, plus their contributions making the open source community a success by responding to messages on project mailing lists, incorporating contributions and bug fixes from outside developers, and other community-related activities.

Choosing a license

Here are some questions that will help you to pick which open source license to use:

• Does your project currently use any open source code? If so, then you need to use the same license that the open source code uses, or another license that is compatible with it.

• Will your project need to work with any other existing open source groups or code? If so, then you need to use a license that is compatible with theirs. For example, working with GNOME or Linux groups will be easier if your project uses GPL or LGPL, while working with an Apache-related project it would be best to use a BSD-style license.
• Who do you see as your potential contributors: companies or individuals? If companies, then
don’t use GPL as many companies are still reluctant to use that style of license, instead think
about using SPL or BSD so proprietary products are possible. If individuals, then GPL would
probably be a good choice. If you want to minimize barriers as much as possible, consider a BSD-
style license.

• Do you plan to charge people to use the technology? That automatically means you cannot use an
open source license; you must either use a proprietary license or a variant of the SCSL. If you are
willing to let anyone use the technology for free, but instead want to charge for use of a name or
logo, then you can use any license for the technology and have a separate logo license for anyone
who is willing to pay to use the logo.

• Does your code define a platform or infrastructure? If so, then SCSL or GPL will be better suited
to your needs—SCSL, if the technology needs a strong central organization to move it forward,
or GPL, if the technology is more mature.

• Are you mainly trying to establish a standard? If so, then you may not need to develop any source
code, and so don’t need any license. Instead you need to establish a standards group. If the stan-
dard already exists and you want to promote it, then using the SISSL may be appropriate.

• Will your company or others want to include your source code in commercial products? Both the
SPL and any BSD-style licenses support creating proprietary larger works.

• Is compatibility critical? If so then SISSL or a variation on the SCSL is appropriate. Alternatively,
the license should not try to control the technology at all, but rather you could require that any
products pass a set of compatibility tests before they can use the trademarked name.

• Are you worried some predator will try to “embrace & extend” the technology? If so, then using
the GPL or SISSL would force them to reveal any extensions they might make. Using the SCSL
would not even allow them to release an incompatible product, but SCSL also may minimize
community involvement.

• Are you or others at your company unwilling to give up control of the technology? SCSL was
designed to allow Sun to keep control, but be aware that the refusal to let the community take
charge may limit your effort. Wanting to keep control is a bad sign that often indicates that you
should not consider using open source at all.

• Is there any IP that you do not want to make publicly available? If you want to limit the technol-
ogy to only be used by the project, then you need to use SPL, SISSL or SCSL.

• Are there two disjoint communities that you need to work with? If so, then you may need to use a
dual license, so each community uses the license best suited to it. However this will complicate
your life, so if at all possible try to use a single license.

Note: you also need to decide on what a developer must agree to in order to contribute source code
to your project. Refer to the section “Supplementing the license—contributor forms” on page 34 for
more details.

Getting your source code ready to release

If you are starting a new open source project, then you need to have working code in order to attract
people to your project. Your code probably was not developed from scratch with the intent of mak-
ing it available to the public, so you will need to get it ready before you can release it. There are two major aspects of the code to consider: design and legal ownership.

Design issues

One aspect of proprietary code development is that most of the code is only seen by the developer who originally wrote it. It is not apparent to anyone else if the code has gotten ugly, if a rough prototype intended to be thrown away has instead become the production code, or if changes and patches have been force fit over an antiquated design in order to add new functionality. The prospect of opening up the source code to the scrutiny of outside developers can be quite unsettling.

Before the source code is released it is important to do a thorough design review of it to avoid embarrassment. We would all like the code to be perfect and polished, but that is never the case and not something that any open source developer expects. What you want to check is that the code is not truly awful, and, if chunks of it are, they may need to be worked on. Sometimes all that is needed is a comment explaining matters, e.g. here is a quick hack to provide functionality needed to test another module, or this only handles the basic case and needs to have special cases added later. If outside developers understand why a design or implementation decision was made, they will not make unreasonable criticisms of the code. However they will be justifiably harsh if the code is really bad and there are no extenuating circumstances.

You want to think of the source code as a work in progress, an approximation of what you want that is going to grow and change over time. Some parts will be exquisite jewels, while others will be lumps of coal. What is most important is that it be possible for an outside developer to make small changes to the code (for bug fixes or minor improvements) without breaking everything.

Part of the design review then is to make sure that the code is open source friendly. One of the most important aspects of that is that the source code builds a working application that people can use right now. If the code is still in development and can not be used to do anything interesting yet, then it may need more work before it is ready for release. Also if the code is not already modular, you may need to make it so. Large monolithic applications that require one to know all of the code in order to modify a small part are not suited to open source development.

Another part of the design review is to examine the project documentation. If you do not already have a good architectural overview, clear instructions for installation, and a documented build process, then you will need to write those documents. Your user community will also need a good user’s manual.

Legal issues

Before you can release any source code, you need to do due diligence on the code to make sure there are no legal problems. You must first be sure that your company legally has the right to release the source code. This means that you must check all of the code to see if any of it was developed by a third party, or if it makes use of any third party intellectual property (e.g. patents or trade secrets). If any third party work is found, you need to check the terms of the license that your company has to use it to see if you can distribute it or not.
For example, the StarOffice product contained third party code for printing and spell checking that Sun could not legally make available. So the OpenOffice open source project started out with no printing or spell checking code. They have since started to write new code for printing, and have incorporated a new spell checker developed by another open source project.

The above example of StarOffice illustrates the need to check the source code itself rather than assuming that all of the code used by a company that your company acquired, like Sun acquired StarOffice, is now fully owned by your company. Code can be “third party” because it was third party to a company your company acquired. The only safe way to identify all third party code is to actually go look at all of the source code.

Searching your code for third party IP can take a lot of time. Some of the searching can be done with automated tools (e.g. using `grep` on all of the files for the word “copyright”), but much must be done by hand. It is often very useful to bring fresh eyes to this task, so that it is not just the engineers who have been working on the code that conduct the search. However, it is not clear whose budget to use to pay for the time engineers from other projects at your company will spend examining the code base.

It is also important to identify any of your company’s IP in the source code that you might not want to make generally available. For example, patents may not yet have been filed for it. Which IP can be shared and which cannot is a business decision.

Finally it is necessary to scrub the code of any nasty comments or obscenities. It can be embarrassing to your company if people find disparaging comments about other companies or individuals. If your code base is not too large, this is tedious, but not too time consuming; if, like StarOffice, you have several million lines of code to scrub, this can be a major undertaking.

One other legal issue is what you will call the open source application if it needs a different name than your company’s branded product. You should have your company’s trademark department check on any names you plan on using. It often does not make sense to trademark an open source product.

**Create a rational development plan**

If your business model involves building a branded product (whether proprietary or not) on top of your open source project, then you need to be very clear on how the development of the two efforts will be tied together. You need to decide how you will merge changes in one effort into the other. Will you periodically take a snapshot of the open source project and incorporate that into the branded product? Or will you do your sampling only after a major release of the open source project? Will you incorporate the entire open source code base or only selected parts of it? What is your process?

Because the open source project involves volunteer developers from outside of your company, you will not be able to dictate its release cycle; the community will decide on the schedule for doing a release and what features will be in it. If your derived, branded product has a fixed release date, what will you do if the open source release is delayed? Can you slip your schedule for the branded product?
The most “rational” process would be to base the branded version on a stable release of the open source project. Changes made to the branded product would then flow back into the open source code base. This process guarantees that both code bases are as similar as possible and minimizes problems of version skew. It also forces the schedule for the branded product to be tightly coupled with the open source project release cycle. Graphically it looks like this:

![Figure 1: Branded product built on stable open source project release](image)

where the bottom line is the ongoing open source development effort and the top line is the branded development effort. The numbers label different events:

1. Incorporating changes from the open source version into the branded product code base.
2. Start of the open source release process, solid line is work on the next stable release, while the dotted line is ongoing development (new or experimental) aimed at some future release.
3. End of the open source release cycle—a stable version is now available. Start of the branded release process, based on the stable open source version.
4. Merge of open source stable version with the main open source development branch.
5. End of branded product release cycle—a stable version of the branded product is now available.
6. Merge of stable branded product code back into open source code base. Note: some changes might have been merged back earlier (between points 4 and 6).

A major assumption here is that there will be no major differences between the open source version and the corresponding parts of the branded product. The branded version may include additional functionality, but those parts in common with the open source version will be a strict subset of the “official” open source release. The only exceptions would be temporary “improvements” in the branded version, such as bug fixes or updated documentation, that would be migrated back to the open source code base.

An alternative development process would be to maintain two separate code bases and have the developers of the branded product decide both when to merge in code from the open source project and when to submit changes back to it. This certainly gives you the most flexibility, but also the most
chaos as the two code bases will evolve independently. Graphically this process looks like the following:

![Figure 2: Branded product independent of open source project release cycle](image)

where again the bottom line is the ongoing open source development effort and the top line is the branded development effort. The numbers label these events:

1. Incorporating changes from the open source version into the branded product code base.
2. Incorporating changes from the branded product into the open source code base.
3. Start of the open source release process, solid line is work on the next stable release, while the dotted line is ongoing development (new or experimental) aimed at some future release.
4. End of the open source release cycle—a stable open source version is now available.
5. Merge of open source stable version with the main open source development branch.
6. Ongoing branded release process.
7. End of branded product release cycle—a stable version of the branded product is now available.

Note that the development of the open source version and the branded product are happening in parallel; there is no relationship between the release cycles of the two efforts. Code is taken from one effort and incorporated into the other whenever some developer thinks is a good time.

The assumption here is that the two versions can have different features. They are now two separate products, only sharing a family resemblance. Over time their differences may increase or decrease.

**Budgeting resources for your project: people and web site**

“How to do open source development” describes a number of tasks that you need to do in order to have a successful project. You need to assign people to do the following:

- Evangelist/community coordinator to encourage and coordinate developers, get publicity for the project, increase community involvement, host mailing lists, and in general keep the project moving along.
Module owners to be responsible for the development of the code and to integrate contributions and bug fixes from other developers. They also need to participate on the project mailing lists.

Infrastructure support for the CVS archive, mailing lists, bug database, and web site.

Web site editor to keep the web site alive with new content.

People to document the system architecture and record reasons for design decisions.

A buildmaster to oversee the build process and fix problems with broken builds.

A release manager to coordinate release activities.

For a new project all of the above will initially need to be done by your company’s employees. As your project grows outside developers may volunteer to take on some of the tasks. If you are joining an existing open source project then you may only need to handle the module owners’ tasks.

You project’s web site provides the place for your community of developers to live. For more information on what the web site needs to do refer to “A community web site” on page 58. If you are starting a new open source project then you will also need to create a new web site. If you join an existing project then there is probably an existing web site that you will be able to build on.

Creating a new web site

If you even suspect that you might want to create a new web site for your project, immediately acquire the rights to the domain name. Do it now.

Your web site will need to provide the following services:

- a CVS server for the source code
- a mailing list server with a searchable archive
- a newsgroup feed of the mailing lists
- a bug tracking system
- disk space for developers and projects.

For security reasons your web site needs to be hosted outside of your company’s firewall. Currently you have three choices:

- pay a company that does web hosting for open source projects to run your web site
- use one of the free open source project hosting sites
- do it yourself

Which one you pick will depend on your budget and project needs.

Companies like CollabNet (http://www.collab.net) provide all of the services your open source project will need. They also charge several hundred thousand dollars per year to do so. This includes a person to check that the community is functioning properly. It can also include a web site editor. If your project has any special needs then you can contract for additional features. For a large project
having CollabNet host the web site may make sense. Both NetBeans and OpenOffice use CollabNet. Note it will generally take three to four weeks to finalize a contract for your project.

Sun has a special web site called SunSource.net (http://sunsource.net) for smaller, Sun-sponsored, open source projects. This web site is also being hosted by CollabNet. Because the web infrastructure is shared by a number of projects, the cost is much less than for setting up and running a separate, large project’s web site. However because the infrastructure is shared it may be harder to add special features for your project. Contact the Sun Open Source Program Office for details on how to get your project included on SunSource.net.

Since many open source projects are composed of developers working in their free time with no sponsoring organization, there have sprung up web sites, like SourceForge (http://sourceforge.net) that provide free hosting of open source projects. This is great for developers with no budgets, but it means you are limited to whatever basic features the hosting company provides, which may include a random banner ad on your web pages. You can purchase additional services, such as more disk space for your CVS archive, if you need them. For a small project with limited resources, SourceForge may be the only possible alternative.

All of the tools used to host an open source project are themselves available as open source. So if you choose to, you can do it yourself and set up your own web site. You would need to contract with a company that does web hosting, but you would do all of the work to setup web pages, mailing lists, CVS trees, and your bug database. The web site for jini.org was at one point run on a Sun workstation owned by the Jini group that was co-located at Best.com.

Before your web site can go live, you need to have your company’s legal and trademark folks take a look at it to check that there are not any legal or trademark problems. You should reserve time with them so they can look over your web pages at least a week before you want to make the web site accessible to the public.

Making use of an existing web site

If you are joining an existing open source project then you will probably be able to use the project’s web site and not have to create a new site of your own. If you are joining in on the work of ongoing modules, then all of the mailing lists, CVS archives, and bug databases should already exist; you just need to subscribe to the mailing lists and start contributing code.

If you are developing new modules, then you will likely need a new mailing list for each module. If it is a large project then it very likely that there is an easy, (possibly automated) way to create a new subproject, complete with archived mailing lists and a directory in the CVS tree. Some open source efforts, such as Apache, require that you are either a member of the Apache Software Foundation (ASF), or have a member’s sponsorship, in order to start a new project.

Adding a new project to an existing web site generally will not put a strain on the site’s resources. If you expect your project will be a significant drain due to the size of the code base or the traffic on the discussion lists, then you might want to consider donating server machines or a monetary contribution. Talk to the web site maintainers and find out what they need to make the web site better. It’s always good to volunteer resources, but don’t assume they will be accepted (e.g. a Linux-based web
site will probably not be interested in a Solaris-based web server). Having your company listed as a supporter of the web site is also a plus.

**Your web site needs an editor**

Your project may be making great progress as developers work on the source code, but if no one is reporting the improvements to your web site and general mailing lists, then to the outside world your project will look dead. A stale web site will turn people away, so you need to have someone assigned to keep the web site home page up-to-date with the latest project news. You want your web site to attract both users and potential developers. The larger your project, the more important it is to have a person whose job it is to let everyone know what's happening. Editing the web site may be part of the job for the community coordinator.

**Cross linking with your company's web pages**

You want to have pointers to your project's web pages from appropriate places on your company's main web site. On sun.com for example, the *Forte for Java* web pages have links to netbeans.org, and the *Software: Desktop & Webtop* page has links to both StarOffice and OpenOffice. The editor of your web site should coordinate with the people managing your company's web site so that it has up-to-date news of your project. Often when your project is first announced there will be lots of stories and links on various of your company's web pages, but months later nothing will have changed. Stale references make it seem like your project has not progressed and give people the wrong idea.

You also want to make sure that any link from your open source web site back to a commercial web site is appropriate. Having a banner ad for Sun on netbeans.org would be in very bad taste, as would giving Sun preferential treatment with links to Sun products. However having a page of links to companies offering commercial NetBeans plug-in modules, including ones from Sun, would be ok.

**Getting approval from your company**

*Note: this section describes the approval process being developed for the Software Systems Group (SSG). Once the SSG process is finalized, a similar policy for all of Sun will presumably be established.*

Before you can join an existing open source project or start a new one, you must first get approval from Sun's upper management. If you've done the steps above then you have already done most of the work required for the approval process and you should have no trouble getting your project approved.

Currently there seem to be three different policies depending on whether you wish to:

- Release Sun developed source code to outside developers. (This includes creating a new open source project around the Sun contributed code.)
- Download source code from outside of Sun for use in Sun products.
- Have Sun employees participate in an existing open source project.
All three cases involve similar issues and any open source project that Sun participates in will almost certainly involve both releasing Sun developed code to outside developers and incorporating code from external developers into a Sun product. An SMI Licensing Steering Committee is currently working to create a single unified policy.

Pointers to the current versions of these policies can be found on the SWAT Legal home page at http://software.central/legal/. (The policy about Sun employees participating in an existing open source project is still being written and has not yet been posted to the web page.)

**Getting approval to release Sun developed source code**

If you are planning on making Sun developed source code available to the public, either by starting a new open source project, or by contributing it to an existing one, you first need to complete the steps specified in the Sun Public Source Code Release Policy (http://software.central/legal/publicsource-new.html). The details may change, but it currently consists of two steps: (1) a Business Analysis section that explains your business strategy and makes the case for doing open source, and (2) a Due Diligence section to guarantee that the source code does not include any embarrassing comments or third party encumbrances. Only after both steps have been successfully completed can the project be started. In fact no announcement of Sun’s intent to release the code should be made until both steps have been completed.

**Requirements of the Business Analysis section**

The Business Analysis section requires you to provide answers to questions in two main areas: the suitability of the source code for public distribution and the strategic significance for Sun of releasing the code. Questions of suitability include:

- Identifying the source code.
- Telling what stage of development the code is in—is it mature code?
- Is the source code virus free?
- Is it adequately documented?
- Is it sufficiently modular and buildable so that outside developers will be able to use it?
- Identifying the engineers who will participate on the Due Diligence team.

Questions of strategic significance include:

- What are the business goals for making the source code public?
- Identifying the audience/community that will use the code.
- Identifying the factors critical to the success of the effort, including resources needed.
- Which licensing model do you intend to use.
- What the financial impact to Sun will be.
- What is the target date to announce the release of the source code.
This information is then reviewed by the SSG Public Source Review Committee, which is made up of your Business Unit’s Vice President and executives from the Software Systems Group including the Chief Technology Officer, Technology Evangelism and Marketing, and Corporate Strategy & Planning. If approved, it then goes to the Division President for approval. Only then is the due diligence process started.

Requirements of the Due Diligence section

Due diligence includes a review of the source code and documentation by the Chief Counsel and technical staff to determine if any third party technology or proprietary information has been incorporated into Sun’s source code. This is done by examining the source code thoroughly to determine the origin of each part. Any code not developed by Sun is checked to see that it is under a license that allows Sun to include it in a public release, and, if not, the code must be replaced or relicensed.

The due diligence effort also includes:

- Confirming that appropriate copyright, trademark and patent filings have been made.
- Determining if there are any export restrictions.
- Removing any inappropriate, derogatory or offensive statements or references in the source code.

Upon completion of the due diligence process, the Chief Counsel will send a report to the SSG Public Source Review Committee. If there are no problems, it then goes to the Division President for approval. If approved, you may then announce and release the source code.

Getting approval to incorporate downloaded source code

If you are planning on downloading source code that has been released under an open source license, then before you can incorporate that code into a Sun product, you must first complete the steps specified in the Web Download Policy (http://software.central/legal/web-download-new.html). The purpose of this policy is to ensure that your Business Unit, any affected functional groups (e.g. Support), and Sun management are making informed decisions when they acknowledge and accept the risks and obligations dictated by the license that the code uses.

These risks include:

- No warranties, which means that if the source code infringes the intellectual property rights of a third party, Sun could be required to stop shipping products containing the downloaded code, and could be forced to pay monetary damages.
- No support, so if needed, Sun must resolve performance, customer support and maintenance, internationalization and localization issues. (Of course, providing support for an open source product can be an excellent business opportunity.)
- Limiting what Sun can do with a product based on the open source code, i.e. Sun may not be able to release the larger work under a desired license.
- Imposing additional requirements on Sun, such as that all source code in any derived work must be made publicly available.
• Indiscriminate dissemination of the downloaded source code within Sun, which could permit it
to be unknowingly used in some other, inappropriate Sun product.

Three steps must be completed before you can start using the downloaded source code in a Sun
product. First your Business Unit must:
• Identify and describe the technology you wish to use.
• Explain the strategic significance of the technology: What gap does it fill? What is its benefit to
Sun?
• Identify which Sun products will use the technology and how they will use it. Will it be embed-
ded in the product or just bundled with it? If embedded, will the downloaded code be used as is
or will it be modified?
• Describe how Sun will distribute the technology, e.g. in source or binary. Will the technology’s
functionality be segregated from Sun proprietary code?
• State if there will be any license fees and/or royalties for the Sun products that will include the
technology.
• Explain how the technology will be maintained and supported.
• Describe the proposed licensing model for any Sun products developed using it.

Then Corporate Strategy & Planning must:
• Ensure that the Business Unit’s strategic rational and business purposes for downloading the
technology are consistent with Sun’s long-term technical and business objectives, and
• Coordinate with the Business Unit and Sun’s Maintenance and Support division to ascertain that
the appropriate support and/or engineering resources are available to support the downloaded
source code once it has been embedded in a Sun product.

And finally your Business Unit’s Chief Counsel needs to:
• Identify licensing requirements, such as needing to publish the modified source code.
• Identify the legal risks associated with using the Web Download Technology.
• Identify any steps or measures to minimize the risks or comply with the licensing requirements.

Only then can the downloaded source code be incorporated into a Sun product. Once it has then
your Business Unit will be responsible for making sure that any requirements imposed by the down-
load license are fully complied with.

**Getting approval to participate in an existing open source project**

If you are planning on having Sun employees participate in an existing open source project, then
before any work is done, you need to provide answers to a number of questions about the project. It
is not currently clear who is involved in approving such a request. In some cases all that may be
required is that the direct manager give their approval to the participating engineers. This policy is
still being finalized and has not yet been posted anywhere.
If Sun developed code will be contributed to the project, or if the code developed by the project will be incorporated into a Sun product, then one, or both, of the above policies may apply and require an additional approval process. Work is in progress to have a single policy for all Sun open source efforts, so hopefully things will become simpler sometime soon.

For now the questions that need to be answered before any Sun engineers may participate include details of the open source project:

• Describe the overall open source project.
• What are the primary reasons that Sun should participate?
• What are the precise goals and objectives for Sun personnel working on the project?
• What open source license does the project use?
• How is IP handled? Does the project allow contributors to retain copyright on contributed code?
• What other companies are participating in the project?

There are also a series of questions about risks to Sun’s intellectual property rights:

• What procedures will you put in place to prevent any of the open source code developed by Sun personnel from subsequently getting mixed with Sun’s proprietary code?
• What is the likelihood that Sun personnel developing open source code, will later be asked to develop similar functionality in Sun proprietary code?
• What existing Sun patents may fall within the scope of the project?
• What steps will you take while working on the project to identify possible future patents?
• Will participation in the project increase the risk that Sun Personnel could disclose trade secrets?

Finally there are some questions about other risks and contract issues that apply more to projects run by an industry consortium than to a traditional open source project.

Once these questions are answered and your management grants approval, then you can start to participate in the open source effort.

What about small projects?

For a small project all of the above may seem like overkill. However the same steps must be followed whether you are releasing 500 lines of code or 500,000. Of course it is much easier to do due diligence on 500 lines of source code so it can be released.

For code that is no longer being supported (EOL), but that you don’t want to die, the amount of effort needed to make it open source may seem to be orders of magnitude more than the work you expect your group will put into the code in the future. Remember that open source is not magic: It takes a lot of work to make a successful open source project. If you just want to dump the code and abandon any future work on it, then you should not be thinking about open sourcing it. Much better would be to simply release it as sample code on your company’s web site.
To be successful open source requires fostering a community around the code, and that takes work. At a minimum, someone who understands the code must filter suggested changes and help developers work with it. There also needs to be an archived mailing list or newsgroup, and at least a handful of web pages, including one that lets people download the source code and executable versions.

Most open source projects are fairly small, consisting of maybe 20 core developers and several hundred users. So the effort required is only the discussion and coding needed to move the project along. As long as the user community is happy with the rate of progress, the project is alive and healthy.

Since a small project will have a limited budget (and an EOL one will have no budget), it will probably not be able to afford to have a dedicated web site. Instead it makes sense to have the project become part of SunSource.net (http://sunsource.net) where a number of smaller, Sun open source projects are hosted. If the cost for joining SunSource.net is still too much, an alternative is to use a free web site provider, like SourceForge (http://sourceforge.net).
5 Going with open source
6. **HOW TO BUILD MOMENTUM**

The previous sections of this handbook have described all the elements that go into creating and running an open source project. In this section we will discuss some things you can do to increase your project’s chances for success. The next section will focus on what not to do based on lessons we’ve learned from seeing failures and mistakes made in various projects.

The ideas below apply to any open source project, however we want to emphasize that a company engaging in an open source project has many additional resources that can be used to further the project in ways that are not available to a project that just has individual volunteers. Besides developers a company also employs technical writers, user interface experts, graphic artists, web site designers, technology evangelists, marketing people, project managers, and people with other skills, all of whom can help with different aspects of the project. Once they understand the open source process, these folks can make major contributions to help the project be a success. All of the activities that your company does to support a proprietary product—marketing, advertising, support, and training—are also needed for an open source project.

As you read over this section remember what your business goals are and focus on how to achieve them. Creating a successful open source project is only part of your strategy. As the project grows and develops it will take on a life of its own. You need to balance the needs of the open source community against your business goals. For the most part they should be aligned in a common purpose and developers from both inside and outside of your company will work together to achieve them. Where they are orthogonal, you can let the community proceed on its own. If they are opposed, you need to be very up front about communicating your plans and reach some agreement that does not alienate or frustrate your community.

For example, one of the business goals of the NetBeans project is for Sun to sell proprietary modules that add functionality to the basic NetBeans IDE. Developing similar modules as open source in NetBeans would undermine this goal, since the functionality would then be available for free. However outside developers have every right to expect that any useful modules they create can become part of NetBeans. So the decision was made that any module could be developed for NetBeans. Also it was decided that when a NetBeans module duplicated the functionality of one of Sun’s proprietary modules that the Sun version would be opened sourced. In this way the level of NetBeans would continually go up.
Marketing your project

As with any product, marketing plays a key role in the success of your project. For you to be successful people need to hear of your project, understand something of what it does and see how it can help them. Getting the message out about your project is one job marketing can help with.

What is your project’s story?

Before your project is ever announced to the world you need to come up with a compelling story that describes what your project is aiming at—something to get people excited about it. This is the message that will attract people to your project both as users and potential developers. If the story is not appealing, then people will not be interested in participating and won’t get involved.

For most open source projects this story is focused on the technology being developed. For example the Apache project exists to build the best web server:

\[\text{The Apache Project is a collaborative software development effort aimed at creating a robust, commercial-grade, featureful, and freely-available source code implementation of an HTTP (W eb) server. (http://httpd.apache.org/ABOUT_APACHE.html)}\]

\[\text{The goal of this project is to provide a secure, efficient and extensible server which provides HTTP services in sync with the current HTTP standards. (http://httpd.apache.org)}\]

Other projects have a more user-centered focus, for example:

\[\text{The GNOME project was born as an effort to create an entirely free desktop environment for free systems. From the start, the main objective of GNOME has been to provide a user friendly suite of applications and an easy-to-use desktop. (http://www.gnome.org/intro/findout.html)}\]

You need to look at what sort of people you are hoping to attract to your project. If you want to mainly target end-users then you need to talk about what they can do with the applications you are developing. If you want to focus on developers who will contribute code to your project then emphasizing the technological aspects may work, but remember most people working on open source projects do so because they need to use the resulting applications—your developers will also be users.

Creating a good story is hard work. The story needs to express the shared purpose behind your project. It will attract people who share this vision and are willing to work to make it real. Moreover the story should infuse all aspects of the project so that it provides guidance whenever a strategic decision must be made. In literary theory such a story is called a topos and can be defined as:

\[\text{a conventionalized expression or passage in text which comes to be used as a resource for the composition of additional texts} \]

—The New Princeton Encyclopedia of Poetry and Poetics

“Topos” literally means a place or location in Greek, and a topos is a place from which similar stories can be woven. An example from literature is the Garden of Eden. Anyone in a Western Judeo-Christ
tian culture asked to tell a story about the Garden of Eden would likely come up with a story that was consistent with the “vision” of creation the Garden of Eden presents.

To give an example of the power of a good story consider the story developed for the Jini project. Jini technology is a thin layer built on top of Java that allows applications to be written as remote services that can be dynamically deployed and located over a network. As such the obvious story is that Jini is another middleware framework for distributed applications. This has a technology focus that would have severely limited the spread of the Jini message—indeed the term middleware causes even developers to yawn. Instead the Jini marketing team built a story around what Jini technology would mean to users that generated lots of excitement in developers, the press and the marketplace.

The Jini topos speaks of a world of “intelligent services,” “simply connect” and “spontaneous networks” accompanied by a set of home, office, and automobile scenarios derived from this topos. The concepts of “simply connect” and “spontaneous networks” address how people relate to the technology. The Jini topos was highly effective and created a thriving community and associated technology—as hoped—along with thriving E-Speak, UPnP, and SOAP communities and associated technologies.

This type of story is especially important for any project with ubiquity as a goal. For Jini success did not rest in extensive development of the core Jini protocols, but rather in nurturing communities to create a wide range of Jini-based services. The story needed to inspire companies to want their products, be they printers, cameras or whatever, to “simply connect.”

Contrast this with the Apache topos described earlier that speaks of web servers and technology for making them effective. Consequently, while Jini projects tend to fan out creating new services for people to use, Apache projects tend to focus on adding web server functionality and adopting related technologies such as Java Server Pages, servlets, and XML.

At its best the topos as a story-making story attracts and provides room for experimentation and variation beyond what the initiators envisioned.

The name matters

What you call your open source project is important. You need to choose a name that will attract developers. In effect you are creating a brand identity just like any commercial product. You want to create an identity that matches your target audience and that is consistent with your story. Most importantly you don’t want a name that is too slick or connected with a commercial product.

This can be complicated if part of your business plan is to develop a commercial product based on the open source version. You need to recognize the different naming requirements for each version.

For example, when Sun was deciding how to name its various Java IDE products, it used the more formal “Forte for Java” name for the Sun-branded versions, and the more developer-friendly “NetBeans” name for the open source version. Keeping the NetBeans name was important for historical reasons, but as a name it does not really describe the project—a Java IDE and more generally a portable tools platform—so another name might have been better.
Getting the message out

Once you have a good story about your project you need to tell it to people. This is a traditional role for marketing. When you have a major piece of news, such as when you first announce your project, your company’s marketing machine can see that the world hears about it. For very important announcements this can include major stories in the press. For instance the initial stories about Sun’s Jini technology appeared in the New York Times, Business Week and Wired. When top Sun executives like Scott McNealy and Bill Joy give talks or meet with the press, they make sure to talk about what’s new with Jini, NetBeans, OpenOffice and the other open source projects Sun is working on. Major stories are also featured on Sun’s web site, along with links to stories elsewhere on the web.

In addition to the big splash-type of announcement, you also want to maintain ongoing, low-key press coverage. This is essential since there is often a long time between the initial announcement and the final release. You don’t want people to think that your project is dead, so be sure to publicize ongoing activities such as community meetings, working group meetings, and attaining significant milestones. These can be as short as a single sentence in a column of industry news in a magazine. Such stories should also be featured on the home page of your project’s web site. Note that some PR folks are only interested in handling major stories and will balk at the smaller scale needed for ongoing coverage. If they won’t do it, then you need to find someone else who will.

Going beyond standard marketing

The conventional marketing channels—major newspapers and magazines, trade press, and trade shows—are important, but an open source project has other ways to reach potential users and developers including newsgroups, mailing lists, webzines and weblogs. You want to use all of these channels to market your project. You should encourage your developers and users to post to whatever online forums are appropriate.

It is important to use a different writing style for material sent over these alternative channels. Anything that has a slick marketing feel will be rejected and will probably give people a negative view of your project. Avoid hype. What does work is honest talk from developer-to-developer or user-to-user. As such it is the opposite of the typical anonymous marketing message broadcast to a target audience. Instead it is a message from a real person attempting to engage in an ongoing conversation with other individuals. Each message helps establish the reputation of the writer. Your developers/users need to become known and respected members of the community in order to best communicate about your project. A big test of their honesty is being able to admit errors or mistakes and to acknowledge the successes of other projects.

One last point is that it is really important to give credit to the folks who did the work. If a number of outside developers contributed to your open source project be sure to acknowledge their efforts.

Focus on your users and contributors

One of the most crucial factors in your project’s success is releasing a useful product: one that works and solves a real problem. This is exactly the same problem faced by any company releasing a commercial software application. Open source works best with at least several thousand users, so your
product must first attract that many. Then the open source process will encourage those users to provide feedback that will make your project more in tune with the needs of potential users.

If you have such a large code base that you plan on releasing it in several installments in order to give you time to scrub it, then make sure that the first release can build a complete and useful product. Try to release the parts that are most valuable to users and developers first. As you release subsequent pieces that will generate additional news stories about your project.

**Target your potential audience**

Just like any product you need to identify what sort of person would benefit from using your product. You need to make sure your message reaches those people. This is where you can make use of standard marketing and advertising.

You also want to target other companies that would benefit either from using your product directly or by using it to add value to their customers. For example any company doing Java development could use NetBeans to increase the productivity of their programmers. Other companies are creating commercial products that run on top of NetBeans. In both cases companies have an incentive to assign their own developers to work on NetBeans to add in features they need, and indeed a number of companies have already done so. For any open source project, companies provide a good source of potential developers.

**Make users successful**

Installing and learning to use a piece of new software is always a major hurdle for people. As is discussed in the section “Give folks a good initial experience” on page 56, it is important that you make things as easy as possible. Every person that can’t install your application, or who is unable to get it to run, will be sure to mention those problems to other potential users—you want the message sent by word-of-mouth to be that people should try your product, not that it should be avoided.

Good documentation, tutorials and training will also help your users to succeed. The ideas behind open source encourage them to help each other, so provide ways for them to do so via your project’s web site: Create user mailing lists. Compile lists of frequently asked questions (FAQs). Do everything you can to encourage the formation of an active user community. There may even be more than one, so you may need to foster several developing communities. They might be separated by what they do with the product, e.g. Java vs. C programmers, or by other factors, e.g. French vs. German speaking. For large projects work with a publisher to get a book written about how to best use your software.

Do releases as frequently as makes sense. Small frequent releases give people a greater sense of progress than infrequent large ones.

You also want to let your users help make your project more successful. You need to listen to their needs when designing future versions of your product. In fact your users should be part of your project’s design team.
Encourage outside developers

When you first open your source code to the outside world the initial reaction of outside developers is usually very positive. You can expect comments like “Wow! You really meant it! There’s a lot here!” After a few weeks the warts will start to show as they realize how much work will be required to add features they want or to improve performance or to do whatever they care about. You need to expect this change in attitude and react appropriately: Acknowledge the problems and don’t be defensive.

Any developer beginning to work on your project needs help to come up to speed. New developers at your company can often just walk across the hall to ask one of the original developers. Outside developers do not have that option. You need to provide them with a systems overview, internal documentation, and email access to the original developer team.

Make sure your developers welcome outside comments and do not respond with flames. As has been said before your team needs to encourage outside contributors. Some have lots of expertise and need to be treated as peers. Others may lack experience and need mentoring. The tone you set when you respond to the first suggestions, bug reports, bug fixes, and outside contributions will determine what role people outside your team will play in your project. It is much easier to start off right, then to try to recover later.

Communication creates community

It is important that everyone involved with your project—be they users, designers, developers, or evangelists—know what is happening with it. They need to know what’s new, and be able to participate in discussions about various project issues, plans, and designs. As is mentioned in the section “Let everyone know what is happening” on page 40, it is crucial that your internal developers use public mailing lists for their discussions so the entire community can participate.

To repeat some of the points made in the section on “Open mailing lists or newsgroups” on page 40, your project should start off with as few mailing lists as possible in order to focus things. This might mean one moderated list for infrequent project announcements, one unmoderated list for all user-related questions and comments, including design suggestions, and one unmoderated list for all general developer-related messages (mostly implementation issues).

Your team needs to set a friendly tone on these lists to encourage others to participate (c.f. the section on “Posting etiquette” on page 41). You also want to encourage people from outside of your company to respond to questions from other users/developers. We recommend the Jini team’s policy of waiting two days before answering a general user question. Those questions that can only be answered by the core team should be answered immediately. This should result in a mailing list where users naturally answer most posted questions with the core developers answering only the more difficult, technical ones.

Mailing lists are great for discussions, but not everyone has the time to follow them. To keep people informed on current issues you want a project newsletter. This newsletter should be “published” on a regular schedule, ranging from weekly to monthly depending on how active your project is. The contents of the newsletter should be featured on your project's web site. An example newsletter topic might be a “contributor spotlight” that focuses on some individual and their work on the project.
To create a real community for your project means that people get to know one another. Some of this can happen over mailing lists, but far better are face-to-face meetings. Try to have a community meeting as soon as possible and at regular intervals afterwards. For smaller projects you might need to hold your community meeting as part of some larger event, for example as a BOF at JavaOne. Try to hold meetings in different locations so more of your community can attend.

A community meeting is a great way to raise the overall level of excitement about your project. Try to avoid dull status reporting sessions. Plan for more interactive formats to take advantage of people being together. Users/developers usually have lots of questions they would like to ask the core developers, so be sure they get a chance. Introduce major contributors so people can associate a face with their name. You also want your developers to become known as individuals. Reserve time for people to schedule birds-of-a-feather (BOF) sessions during the meeting. Include time for people to network.

Community meetings are also a great place to hold working group meetings. Getting the major players into a room can really speed up the time needed to decide on open issues. Be sure to report back via email to the full community on any decisions made at the meeting.

Other meeting options include phone conferences and online chat sessions. An open source project has by definition a distributed work group and you need to explore ways to facilitate communications among them.

Make sure adequate resources are allocated

An open source project requires more in the way of resources than a conventional proprietary one. If you skimp on providing for the additional needs, then you will slow down, or even cripple, your project. You want the rate limiting factor for your project to be the enthusiasm and effort of your community, not that the person maintaining the project's web site is too busy to update it or your internal developers are so overworked they have no time to integrate outside contributions and bug fixes. To start up an open source project implies a commitment to engage with your community and to support them.

Of course you do have limited resources, so you need to identify when your lack of resources is the bottleneck and see if the community can provide the additional manpower. For example, what if one of your developers is in charge of a module, but has so much of his own work to do that he is unable to promptly reply to bug fixes or modifications submitted by outside developers? In such a case one or more other developers needs to be allowed to also check in changes to that module. Very likely this will be one of the outside developers who submitted a change to the module. As soon as possible you want to have qualified outside developers receive check-in privileges.

However that same overworked module owner must make time to participate in any community discussions relating to his module. If not then another person who will engage with the community needs to take over ownership of the module. It is ok to delegate work on the module to others, but leadership about the module's development requires active participation.

This is a major area for good project management. A good project manager will make sure that when the project is being hurt by a lack of adequate resources that either more resources from within your company are allocated or that the community is solicited to provide the additional help.
Community outreach

Your project does not exist in a vacuum. You want to strengthen your connections with other open source projects. You want to do this to be a good citizen and also because it will benefit your project. It will also benefit your company that many of your software engineers have established a good reputation with the open source community.

Being active in the larger world of open source will help you to better run your open source project. Your project infrastructure undoubtedly uses many open source tools such as CVS, Bugzilla, and Perl, so you may want to be involved in the debates on their future development. Seeing how other projects are run will give you ideas for what your project should or should not do.

You also want to market your project to other open source efforts. They provide a natural audience for your product and a source of potential contributors. For example if your product runs on Linux then you want to work with those groups creating Linux distributions so that your product is included—both StarOffice and Forte for Java, Community Edition are part of the Linux distributions from RedHat, Turbolinux and Caldera.

Welcome the unexpected

Finally one of the strengths of open source is that contributors will always surprise you. When someone submits a new module that takes things in a new direction, be willing to devote some resources to making it a bigger success. If some people start using your product in ways you didn’t expect, check it out and see if there’s a whole new market waiting for you. And most of all listen to what your community tells you it wants and needs.
7. **What to Avoid—Known Problems and Failures**

As you work to move your project forward it is inevitable that you will make some mistakes. How you respond to them can be critical to your project’s success. If you are honest with your community about what was done and take steps to correct matters, then your project can move forward with strengthened mutual respect between you and the community. If you refuse to admit that there is a problem or ignore the wishes of the community, then the community will have less trust in you and your intentions. This is especially critical for large companies where the outside developers often start off with a low level of trust concerning the company’s motives. Any time you act to reinforce their fears hurts your project’s chances of success. When your actions demonstrate your respect for the community’s opinion, you earn the trust of your community and improve the chances of your project recovering from any future problems.

In this section we will look at some problems that have caused various open source projects to stumble. Hopefully by being aware of them you can avoid them happening in your project.

**Not understanding open source**

The first place that problems arise is when people making decisions about whether to use open source for a project do not understand how open source works. As a result they decide not to use open source because they imagine there are insurmountable problems, or, even worse, they choose open source but have unrealistic expectations. The first case has the company missing a possible opportunity, while the second probably dooms the project to failure.

One common fear is that by choosing to make a project open source the core developers will be swamped by having to answer thousands of email messages from outside users/developers. This is one reason that engineers working on proprietary projects are often kept isolated from the customers. Indeed the core team will need to spend some of its time engaging with the community and answering questions, however the bulk of the questions that arise on project mailing lists will end up being answered by another community member—the community naturally provides the necessary resources.
Another common misconception is a fear that by open sourcing one product that everything connected with it, including totally separate programs, must also be open sourced. Microsoft and others who are opposed to open source have tried to play on this fear in various statements and articles where they have called open source both “un-American” and “a cancer.” The reality is that even under a strong open source license like the GPL only the actual source code of the application is affected. Other open source licenses, like the LGPL, Mozilla, or BSD, make it easy to combine proprietary and open source code. There are real issues about what IP an open source project will share, but they are not as horrific or all-devouring as people uninformed on the details of open source might fear. Note: Because no open source license has yet been tested in court, lawyers are prone to be overly cautious when they first read an open source license.

Many people think that the point of doing open source is to get hundreds of outside developers to work for them for free. When this doesn’t happen, and it almost certainly won’t, they will declare the project a failure. Moreover people thinking that open source is just a way to cut costs will not allocate sufficient resources to the project—starving it to death.

The other common mistake is to think that a company can open source a project to get good PR, while playing the same old proprietary game. The outside world will quickly see this for the sham it is, and no one will join the project.

Doing an open source project for the wrong reasons will likely get you a lot of bad press coverage and alienate outside developers.

A different type of problem is starting to occur as more people learn about open source: People are now thinking that any open source project they create must work exactly the same as an existing open source project. Projects such as Apache, Mozilla, and Linux are being held up as exemplars of the only way to run an open source project. While it is important to learn from successful open source projects, we are still very much just beginning to understand the landscape of open source projects. To really succeed requires adopting an open source attitude and innovating new ways to do things.

**Don’t duplicate an existing effort**

Generally speaking, there will only be one open source project in a given area. The few instances where two open source projects exist that are trying to produce similar results, is due to some political issue: GNOME happened because of unhappiness with the license used by KDE, XEmacs came about because of a disagreement in how to add display support to Emacs, and Linux started up while AT&T was legally battling with Berkeley over BSD.

There are only so many open source developers interested in a given area and it is hard enough to create one community of them, let alone two. So if there is already an open source project with goals similar to yours, then you should join that project. If there is some very good reason why you cannot join with them—and it had better be an extremely compelling reason, such as they are using GPL and your business aim is to produce a proprietary version—then you should rethink whether you should be doing open source at all.
If you decide to proceed it is very important that you establish good relations with the developers of the existing project. The last thing you want is to have them bad-mouthing your project on forums like Slashdot. Also you probably share many common goals with them and there are apt to be many places you can work together, e.g. establishing common file formats.

There will sometimes be cases where it is necessary to create a competing open source effort. For example, when Sun released the StarOffice code it overlapped with two existing open source projects: Gnumeric and AbiWord. In the case of Gnumeric, the lead developer for it was engaged in conversation about this overlap before OpenOffice was announced, and, as a result, he views the OpenOffice project in a good light and is probably receptive to working with Sun if an opportunity to do so occurs. However, no one on the AbiWord project was consulted before or after the OpenOffice announcement; moreover one of the AbiWord developers tried to open a discussion about developing a common XML file format, but never received any replies from the StarOffice engineers. The result is that the AbiWord developers became suspicious of Sun’s motives and did not expect to work with Sun, and both efforts will be the worse because of this.

One final point about creating a competing open source project: Do not expect any people working on existing projects to switch to your project. Your developers will come from those people who have not yet gotten involved with either project, and you will be in direct competition with the other project for potential new developers. In the example above, none of the Gnumeric or AbiWord developers switched over to work on OpenOffice.

Licensing issues

The license you chose can create problems if it is unfamiliar to your target audience or inappropriate for your project.

It is a bad idea to create a new license for your project. If people are unfamiliar with a license they tend to be suspicious of it and shy away from the project. When Sun first introduced the Sun Community Source License (SCSL) no one quite knew what to make of it, plus it was hard to understand. Using a new license means also taking on a large community education effort—time that might be better spent working on your project.

The Mozilla project has decided that they want to be able to engage more with a number of projects that use the GPL. As a result they are in the messy process of contacting everyone who has contributed any code to Mozilla to get their permission to dual license the code under both MPL and GPL. This highlights two things: the problem of letting contributors hold the copyright for their contribution and how the choice of license can limit who will work on the code.

Code issues

One of the first hurdles an open source project faces is whether the code is complete and builds a working application. This is the classic complaint about Mozilla—the source code Netscape released was incomplete and could not make a working web browser. The Mozilla team was committed to the work, so the project has continued. As a working browser has begun to emerge, more casual contributors have also been able to participate.
A second hurdle is making sure that the user succeeds when first trying out your application. This was a noticeable problem with Jini, not because of a problem with the Jini code, but rather due to a problem configuring Windows systems to properly support multicasting. The lesson here is that even though your code is fine, problems with other software may cause your project to suffer.

Another stumbling block occurs when outside developers have trouble navigating the source code. This could be because the code is too difficult to understand, in which case it often needs to be rewritten to be more modular. It can also be because there is inadequate system documentation, or just because of the sheer size of the codebase. Developers who want to work on the OpenOffice project have a steep learning curve due to the tremendous amount of code involved. This is a particularly common problem as a project moves from being proprietary to open source because of the vastly different levels of developer involvement—even an incredibly well documented proprietary project still assumes a full-time commitment from a new developer.

A related problem cropped up in the JXTA project. The JXTA source code was deliberately released early so that the community could help to design it. This led to some developers being confused about which aspects of JXTA were not yet defined and which had just not been documented yet.

A different type of problem often happens when a project has a slow release cycle. One way that the outside world judges how well a project is doing is by how long it has been since the last release. If a project goes too long without a release, people will start to consider the project dead or a failure. For example, many people thought that the Mozilla project was in trouble after its first year because no working browser had been released. The frequent releases over the last year have dispelled this feeling and now people outside of the Mozilla project are likely to think that it is alive and making progress.

If your project has not released anything for a long time, you should consider putting less into your next release and trying to do smaller, more frequent releases. You should also make sure that your project web site is updated regularly—a stale web site is another indication of a dying or dead project.

Another potential problem relates to subprojects. As we discussed in the section on “Don’t duplicate an existing effort” on page 94, there will usually be only one open source project in a given area. This also applies within an open source project: once someone contributes a new module to your project it will be difficult for a similar module to compete. Stated another way, the first code to show up usually wins. For example as part of the Apache XML Project, the Xerces XML parser appeared first making it difficult for another XML parser, the Crimson subproject, to be adopted. In fact the best parts of Crimson are now being merged into the Xerces codebase and then the Crimson subproject will disappear.

A final problem comes about from thinking that developing the code is all that is required. If all of your attention is on developing the code you are likely to lose track of your main business goals. For example if ubiquity is a goal then marketing the technology to potential partners should have a high priority—internal development resources should be focused on helping partners succeed. If you can get a large company like IBM or HP to adopt the technology, you can be sure they will contribute their own developers to help work on the code. Unfortunately the natural tendency is to try to do everything yourself and larger opportunities are often neglected.
Another way this problem can hurt your project is if the developers, both internal and external, are the only ones to decide on the project’s future direction. The whole idea of open source as a meritocracy focused on coding tends to disenfranchise anyone who does not contribute code, which means that users, UI designers, marketing people, and the like are often not able to fully participate. This results in software that is designed by developers for developers and is why much open source software is not appropriate for the general public. If you are seeking “world domination,” then this can be a major, major problem that you need to overcome.

One way to make the focus go beyond the code is to actively make roles for non-developers like user interface (UI) designers and documentation writers. For example, NetBeans has a community-wide mailing list dedicated to the design and discussion of UI issues. There is also a NetBeans project focusing on UI development and issues, plus a process for other developers to get UI design assistance from the members of the UI project. When the NetBeans project first started there was a hesitancy to add non-developer roles like this because that wasn’t something that the high profile open source projects like Apache or Linux did. You might encounter a similar resistance.

**Trying to control too much**

One of the surest ways to cripple an open source project is for the original developer to refuse to give up control. At best this turns potential outside developers into a user group and at worst kills the project. There are many ways that the original developers can try to keep control.

A common mistake is to think that the internal development group does not need to change how it works. Very often the internal developers continue to use a private company mailing list to carry on any discussions they might have about project issues. While not every email message needs to be sent to the public project mailing list, if essential design discussions are carried out in private then one of the main benefits of open source—feedback from users and outside developers—is lost. A further problem with this is that when the internal developers do post to the rest of the community it is then in the context of announcing what they have decided, rather than as a starting point for community discussion. For a successful open source project the internal development team needs to shift their email discussions to the project’s public mailing lists.

The other major way that internal developers do not change their development process is by keeping a private code archive, often one that relies on a proprietary code versioning tool instead of CVS. Some projects do not even have a public code archive, just tar files with the source code for the latest release. This sends a clear message to potential outside developers that they are second-class citizens who can look at the code, but never touch it directly. In fact they cannot even see the most recent changes, but need to wait until the public copy is updated. A real open source project has one public code archive that all developers use, though not everyone will have full write-access to it.

Another problem arises when developers want their design or code to be perfect before letting anyone else see it. There is a major shift in attitude needed to go from a proprietary development environment where, in many cases, no one ever looks at your code, to an open source world where potentially everyone can see what you’ve done, warts and all. Whether it’s a single developer or the entire internal development team, waiting too long before releasing a design or code is a mistake that hurts the overall project. It is important that everyone think in terms of a “work in progress,” devel-
7 What to avoid—known problems and failures

opened in the open to maximize possible feedback from others. Remind folks that nothing is final, it is always subject to modification as the project’s needs change.

A more general instance of this is common to all sorts of projects, both proprietary and open source, where developers and managers try to make a master plan and then carry it out. Planning is good, but carried to extremes where it does not adapt to change, it hurts your project. For example some open source projects try to figure out all of the appropriate mailing lists and set them up so they are ready when the project is first announced. Unfortunately the needs of a project that has just started up are often quite different from those of one that has been running for a while. A better approach is to start with one or two main mailing lists and only when the number of daily messages grows too large should more lists be created. Engineering in general often tries to do “the right thing” in one step, rather than take a number of smaller steps in whatever direction currently seems best.

A very bad problem occurs when the internal developers do not have respect for the outside developers. If any member of the core team thinks of the outside developers as bozos that should be ignored, then you can be sure that the community will pick up on that attitude and react appropriately. You need to remember that there are more smart people outside of your company than inside it, and one major reason for making your project open source is to be able to engage with them as much as possible.

This problem can often be seen by core developers refusing to engage with outside developers in design discussions on the project’s mailing lists—often the core developers don’t even read the public mailing lists. Refusing to listen to community concerns hurts the project because it both kills a sense of community and results in developing the wrong product. Keep in mind that something like half of all software projects fail, not for technical reasons, but because they produce a result that does not match the need of their target audience. Feedback from the community in an open source project guarantees that what is produced is what is wanted. Make sure you listen to what your community says; they are probably right.

The day-to-day operation of a project offers many places where mistakes are made by trying to keep control. Creating a real community requires that the community participate in making decisions. There is no real community when the only developers who can check in code to the CVS archive all work for your company, where every important module is owned by a developer employed by your company, or where any governance board consists solely of your company’s employees. So try to grant outside developers the right to check in code as soon as possible, make an outside developer in charge of a module as soon as possible, have outside developers on your governance board.

You can alienate outside developers if your company’s employees are automatically given check-in privileges, while outside developers have to earn them by making good contributions. This is a mistake that a number of open source projects with a core team from a single company have committed. It is really important that new developers are treated according to the same standards regardless of what company they work for. Sometimes a new developer at your company may have made contributions that the outside community doesn’t see because he or she hands their code over to the module owner in private, so be sure to make the activity of new folks visible to all.

Another mistake some open source projects make is when the module owners are too slow to accept outside contributions and check them into the official code archive. Or even worse, refuse to accept
them. This might be because the module owner is so busy they have no time to deal with any contributions from the community. It might also be because the module owner is being too controlling over what goes into their module. Part of the module owner's job is to maintain the quality of the code, but another equally important job is to be responsive to the directions set by the community.

If another company is contributing so much code that the module owner becomes a bottleneck, then it is really important that one or more of the outside contributors be granted the right to also check code in. The alternative is that no one will make additional contributions or more likely that the outside developers will fork the project as happened when the XEmacs effort started a second project to develop Emacs faster and along a different direction than the original Emacs development effort.

In summary one of the worst things you can do to your project is to try to control it too much. While other mistakes will slow down or limit your project, too much control by you will make it likely that someone will fork the project. Rather than trying to control everything it is much better to provide positive leadership. If you set goals that the community can agree to, then everyone will be working together to advance the project. If you don't engage the community, you will lose it as everyone stops trying to work with you and goes elsewhere. One final point is that not all engineers and managers are cut out for open source: if they refuse to give up control then they may do more harm than good for your project.

**Marketing issues**

The biggest mistake many open source projects make is not actively marketing themselves and the applications they are creating. No marketing means fewer users and fewer contributors. For a low-key effort this may be fine, but if you really want your project to take off people need to hear about it.

When they hear about it is also important. Some companies make a big announcement well before the project is underway and the code is available. This can drum up quite a bit of interest in your project, but if there's nothing for potential outside contributors to do, then it can be a bit of a wasted opportunity. For example, in mid-July 2000 Sun announced that the StarOffice source code would be made available and created a project web site at OpenOffice.org. This generated lots of interest and many people flocked to the openoffice.org mailing lists. Unfortunately there was no documentation about the soon-to-be released code and only a few, low-level StarOffice developers participated on the mailing lists. As a result potential community members ended up spinning their wheels and went down many dead-end topics speculating about how StarOffice worked until the code was finally released in mid-October and more members of the StarOffice development team joined the mailing lists. If you were having a party at your house you would not think of making your guests wait outside for many hours before you let them in, and if they had to be outside for some time you would go out and welcome them, not leave them to fend for themselves.

Big announcements that get major press coverage are great for your project—assuming they have real content and are not just hype. However many marketing and PR people think that is the only type of story worth doing. As a result, in the long periods between major announcements many people might think that your project is dead since they haven’t heard any recent news about it. This happened to the Jini project in the several year interval between when the technology was first announced and when products based on Jini began to appear in the marketplace. Any new technology will have a similar interim period, so following a huge initial announcement with a very long and
noticeable silence will invite people to think the project has failed. You need to make sure that the many small, but significant, events associated with your project are properly communicated to the public. Frequent small mentions of your progress will let people know that your project is alive and well. An ideal place to communicate current news of your project is to post it on your project’s web site.

Letting your project web site become stale is a surefire way to make people think that your project is dead. Many people interested in your project will never subscribe to the project’s mailing lists—they will periodically surf to your project’s web site to see what’s happening. If the project’s home page hasn’t changed they will conclude that nothing is happening and are less likely to check again. Make sure that someone is assigned to update your web site on a regular basis.

The worst marketing mistake a company can make is to not give proper credit for the work done by outside developers. For example, back in December 1999 at a large trade show, Sun announced the release of a Linux version of the Java 2 Platform, Standard Edition (J2SE) that was done in collaboration with Borland Software Corporation (then named the Inprise Corporation). The press release failed to mention that much of the work that lay behind J2SE was done by the Linux Blackdown Porting Team, a group of open source programmers devoted to Java. It turned out that this major omission was because the marketing person writing the press release was unaware of the work done by the Blackdown project. Sun immediately released another announcement giving proper credit to the Blackdown team and also the Sun engineers in charge of the Linux port personally apologized to the Blackdown group. Promptly admitting the mistake and correcting it was crucial for Sun to regain trust with the open source community.

The above highlights another mistake in that Sun did not publicize participation by Sun engineers in the Blackdown work and other open source projects. If people inside of Sun had known then the mistake above would never have happened. Likewise those in the open source community are unaware of Sun’s many contributions to open source projects causing Sun to not receive credit it has earned. In fact according to the Orbiten Free Software Survey (http://orbiten.org/ofss/01.html) done in May 2000, Sun was ranked as the second largest contributor of code to open source projects. To get the credit it deserves your company needs to let others know of its contributions to open source projects, but without overemphasizing or hyping its participation.

Tension between an open source project and the rest of your company

It’s not enough that everyone working on your open source project act appropriately with the outside community, you also have to deal with the rest of your company. If others inside of your company do not understand why you are using open source and the implications of that to them, then your project can run into major trouble. Open source projects need different resources than proprietary ones and have different constraints in how they operate. If this is not understood by those outside your project, then they will work against you in the normal internal politics that goes on inside every company.

One mistake your open source project can make is to not market itself within your company. Moreover your message needs to have a different emphasis than the one used for people outside of your company. You need to communicate how the goals of your project help further your company’s business goals. If you fail to do this then your project will be seen as irrelevant and other proprietary
efforts will be given precedence. This marketing must be an ongoing activity, especially when there are changes in upper management.

You also must educate people at your company about what metrics will be used to measure the success of your project. These are likely to be quite different than the more familiar metrics used by proprietary projects. Anyone opposed to your project, for whatever reason, will have a much harder time proving it a failure if everyone understands the appropriate metrics for measuring it.

A common management mistake we have seen is to have people trying to serve two masters. When a developer is supposed to be helping to grow the community and also be writing code to meet an internal deadline for some proprietary work, everyone knows that the community involvement will be put on hold. It is necessary to separate developers into those focused on the open source activities of the project and those doing work on proprietary projects. Otherwise local pressure from managers and coworkers will ensure that the proprietary aspects are given priority. A failure to include community-related metrics in the performance reviews of those involved with open source activities is another mistake that gets made and also results in those community activities being neglected.

The worst type of mistake that projects make is to try to cheat or put something over on their community. This could be as minor as sneaking code into the codebase or something major like blocking a competing company from checking in code they need. Any actions that treat the open source project as if it were a proprietary effort owned by your company will destroy the community's trust of your company—both for your project and also for other open source projects that your company is involved in. Anyone who thinks that they can get good PR for an open source project, while playing the same old proprietary game is sadly mistaken. If someone tries to avoid, ignore or otherwise fool your project's community, you need to let them know that your company has made a public commitment to your open source community and any monkey business will get your company bad publicity. Sometimes a person may even have good intentions, but any activity that does not show respect to the community and the approved community processes should be avoided.

Another example of the tension between the proprietary side of your company and your open source project is apt to arise over your project's schedule. Proprietary efforts within your company will expect you to have control of when your project does a release and what features are included in it. If their products depend on yours this may cause them to make unrealistic plans. You need to educate them so they understand that your community has a major say in the release process.

Community issues

Some of the worst mistakes that can hurt the growth of your project's community have already been mentioned above. These include mistakes like the internal developers continuing to work in a proprietary style without engaging the community; trying to plan out the entire community ahead of time rather than letting it develop more organically; starting up the project web site and mailing lists before you are ready to participate; and acting as if you owned the entire project.

Another set of mistakes stem from a belief that the community will arise by itself. That will never happen. A common failure is to think that the community doesn't need face-to-face meetings—that email communication alone will create the needed connections between people. This just isn't the case. Meeting in person allows folks to put faces with names and to see each other as people; it helps
to break down the barriers between your company employees and outside contributors. A related mistake is thinking that all that is needed to have a community meeting is to announce it a few months in advance and then it will just happen. This doesn't work. Like any meeting, it is important to plan out what topics will be discussed at the meeting and post an agenda well in advance. It is also necessary to talk up the meeting in community discussions. People need to be motivated to want to attend, especially for the first few meetings when the community is just getting established. Creating a vibrant community only can occur if you and your team nurture it by encouraging contributors, holding community meetings, and supporting community activities.

Another problem is if your company’s developers send harsh or acrimonious messages—flames—to a project mailing list. Flaming rarely improves matters and the community will consider the flame as coming from your company. Make sure you and your teammates follow good posting etiquette and avoid attacking outside developers. Even if you know that the person you would flame is comfortable with that sort of exchange, remember that public messages are seen by everyone, many of whom will not appreciate flames. Since the mailing lists are the major way people in your community will communicate, it is crucial that everyone feel safe posting to them. Flaming and harsh criticism in messages will discourage many people from participating.

A similar problem is that any opinions or comments posted by an employee of your company are often taken as being official company policy. You need to be sure to indicate what is just your personal thoughts and what is really company policy.

Another set of problems center on how community members can create new projects for modules they are working on. On jini.org any registered community member can create a new project, however old projects that die are not cleaned up. As a result it is very confusing to a Jini developer who wants to join an existing project as they are confronted with a list of over 150 projects, many of which are defunct. For any open source project someone in the community needs to be responsible for pruning inactive projects. You also want to evangelize active projects so potential participants know about them. Both the Jini and JXTA project home web pages have a section to spotlight one or two community projects.

The opposite problem is making it too hard for someone to start up a new project. Both NetBeans and JXTA have an approval process for creating a new project. They require public discussion before a new project can be approved. This works to avoid duplication and to inform the rest of the community about the new project, but in a way that may prevent some types of projects from being started. People are generally conservative, and when a project sounds like it is outside what is normal or usual, they will tend to discount or discredit it. So, if you ask people what they think, if the project is innovative in some way, it will tend to be disparaged. Whereas, if you just let the project proceed, then maybe once people can try out a working version of it then its value will become apparent. That is, a deliberate process will screen out innovative projects through a kind of fear. Hence a better approach might be to allow anyone to create a new project, then have a community coordinator monitor the project’s health and should it die, move it to a list of inactive projects.

A final community-related problem concerns one-size-fits-all project infrastructure tools. Make sure that the company providing the infrastructure for your project (web site, CVS tree, mailing lists, and bug database) does not treat your project as a generic open source project. Every open source project will naturally do things slightly differently, and, where this is significant, you don’t want your project...
constrained by your infrastructure provider. For example NetBeans discovered that when CollabNet upgraded their project to use SourceCast that the new project roles and CVS roles clashed. Some open source projects limit a developer's CVS access to just those modules that they have been granted access to, which is how SourceCast had been set up. NetBeans used a different model where once a developer was trusted to commit changes to one module, then they were given access to every other module too. The NetBeans model encourages community members to trust each other and relies on social, rather than technical, means to enforce what code changes are allowed. After the infrastructure change to keep this preferred way of working, NetBeans had to require CollabNet to modify things. This is an example of two things: That technological concerns should not dominate design, and that there is not one true way to do open source.

Lack of resources

The underlying cause of a number of the mistakes mentioned above is a lack of resources, most notably people to work on some necessary aspect of your project. Those people can be employees of your company that are assigned to work on your project or they can be volunteers, including employees of other companies.

Many projects do not have anyone assigned to be a community coordinator or evangelist. That can compound any resource problems because the community coordinator would be the person to recruit volunteers from the community to help out. For a smaller project the project owner can take care of this, while for bigger projects it might be part of the product manager's job. For large projects it can become a full-time job.

The most common reason that a project's web site becomes stale is because no one has time to update it. Outside contributions can get ignored when module owners are too busy. Someone needs to write the internal documentation needed to orient potential contributors. Make sure that you don't let critical tasks remain undone: assign an employee of your company or find an outside volunteer.

Recovering from mistakes

Everyone and every project makes mistakes. It's what you do after a mistake that really matters. What you need to do is:

- Admit that a mistake was made.
- Take steps to set things right.

Honestly discussing with the community what happened and how to make sure it does not happen again is the only way to proceed if you want to have a successful open source project. Being defensive, refusing to admit there is a problem or refusing to fix things are just further mistakes that will alienate potential users and contributors and erode their trust of you.
8. CONCLUSION

Open source is a way of building closer relationships with your customers and better relationships with your developers. An open source project breaks down organizational boundaries both within your company and between your company and the outside world. Open source results in software that better meets the needs of its users, in both greater fit and higher reliability.

In an article taking a broad look at open source Brian Behlendorf wrote:

> What the open-source community has proven is that individuals—and, by extension, companies—can work together on a much more discrete, iterative level. ... People are starting to understand that having interaction between developers on a daily basis during development is far more valuable than waiting through the typical product-revision cycles. It may seem chaotic at times; for programmers, balancing the requirements of their employers with that of the other participants may be a constant challenge. But it can work.


There are many ways to run an open source project. When first starting out it is a good idea to model your project on one of the established open source projects, such as Apache or Linux. Once you have gotten the proper feel for things, you will be able to evolve how your project is run so as to better meet your specific needs. Just as your project will change as it matures, there is also an arc of maturity for all of your company’s sponsored open source projects: With the first project, Jini, Sun was just learning to let go of control. With Tomcat, NetBeans and OpenOffice the basic skills of running an open source project began to spread within Sun. Only then did it became possible to do more experimental projects like JXTA. It takes a while for trust in the open source approach to spread within an organization. But once it does, then the scope of possible projects that can be open sourced increases, as does the ability to innovate in how those projects are run.

Open source is an evolving methodology with its own arc of maturity. As existing open source projects face new problems they need to innovate in how they operate. The Apache project is grappling with how to resolve conflicts over competing approaches. Linux is trying to increase compatibility among Linux distributions and prevent the fracturing that plagued Unix in the 1980’s and 1990’s. Open source projects are experimenting with various governance methods, new licenses, and different ways to work together.
We believe that as the focus of programming shifts from stand-alone applications to interacting web services that we will see a new way of developing software arise that we call mob software. Mob software takes the self-organizing and open-ended aspects of open source and goes much, much further with them. Imagine every person, not just programmers, as able to modify the software they use. With millions of contributors sharing a multitude of improvements, mob software will transform how we create and think about software. For more thoughts about mob software please refer to our essay Mob Software: The Erotic Life of Code available at http://www.dreamsongs.com/Essays.html.

Open source is not for everyone, but if you have the right attitude then it can be a major success factor for your project. You must be willing to give up control and share decision making with your community. Working together you can create something much better than you could by working alone. Good luck!
APPENDIX A: RESOURCES

BOOKS FOR FURTHER READING

*Community Building on the Web*; Amy Jo Kim; Peachpit Press; 2000.

*Open Source Development with CVS*; Karl Fogel; CoriolisOpen Press; 1999.

*Open Sources: Voices from the Open Source Revolution*; edited by Chris DiBona, Sam Ockman & Mark Stone; O'Reilly & Associates; 1999.

*Rebel Code: Inside Linux and the Open Source Revolution*; Glyn Moody; Perseus Publishing; 2001

*The Cathedral & the Bazaar*; Eric Raymond; O'Reilly & Associates; 1999.


OPEN SOURCE WEB SITES

The latest version of this Handbook can be found at: http://www.dreamsongs.com/OSHandbook.html

Free Software Foundation home page: http://www.gnu.org

Open Source Initiative: http://www.opensource.org

Slashdot: http://slashdot.org

Apache Software Foundation: http://www.apache.org

Eric Raymond's *The Cathedral and the Bazaar* can be found at: http://www.tuxedo.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/index.html

CollabNet: http://www.collab.net

SourceForge: http://sourceforge.net
TOOLS
Bonsai to query about changes to a CVS archive: http://mozilla.org/bonsai.html
Bugzilla bug tracking tool: http://www.mozilla.org/bugs/
Tinderbox to show build status of source tree: http://mozilla.org/tinderbox.html
The Linux Cross-Reference tool, lxr, is available from: http://lxr.linux.no/

LICENSES
Apache: http://www.apache.org/LICENSE.txt
BSD: http://www.opensource.org/licenses/bsd-license.html
GPL: http://www.gnu.org/copyleft/gpl.html
LGPL: http://www.gnu.org/copyleft/lesser.html
SCSL (Jini version): http://www.sun.com/jini/licensing/licenses.html
SISSL: http://www.openoffice.org/licenses/sissl_license.html
APPENDIX B: OPEN SOURCE LICENCES

Below is the full text for all of the open source license mentioned in this handbook. Included with each is a pointer to where the license may be found.

GNU GENERAL PUBLIC LICENSE (GPL)
taken from: http://www.gnu.org/copyleft/gpl.html

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place - Suite 330, Boston, MA 02111-1307, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation’s software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.
To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author’s protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors’ reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone’s free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

**TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION**

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The “Program,” below, refers to any such program or work, and a “work based on the Program” means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term “modification.”) Each licensee is addressed as “you.”

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program’s source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.
2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
c) Accompany it with the information you received as to the offer to distribute corresponding
source code. (This alternative is allowed only for noncommercial distribution and only if you
received the program in object code or executable form with such an offer, in accord with Sub-
section b above.)

The source code for a work means the preferred form of the work for making modifications to it. For
an executable work, complete source code means all the source code for all modules it contains, plus
any associated interface definition files, plus the scripts used to control compilation and installation
of the executable. However, as a special exception, the source code distributed need not include any-
thing that is normally distributed (in either source or binary form) with the major components
(compiler, kernel, and so on) of the operating system on which the executable runs, unless that com-
ponent itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated
place, then offering equivalent access to copy the source code from the same place counts as distribu-
tion of the source code, even though third parties are not compelled to copy the source along with
the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided
under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is
void, and will automatically terminate your rights under this License. However, parties who have
received copies, or rights, from you under this License will not have their licenses terminated so long
as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else
grants you permission to modify or distribute the Program or its derivative works. These actions are
prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Pro-
gram (or any work based on the Program), you indicate your acceptance of this License to do so, and
all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient auto-
atically receives a license from the original licensor to copy, distribute or modify the Program sub-
ject to these terms and conditions. You may not impose any further restrictions on the recipients'
exercise of the rights granted herein. You are not responsible for enforcing compliance by third par-
ties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other rea-
son (not limited to patent issues), conditions are imposed on you (whether by court order, agree-
ment or otherwise) that contradict the conditions of this License, they do not excuse you from the
conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations
under this License and any other pertinent obligations, then as a consequence you may not distrib-
ute the Program at all. For example, if a patent license would not permit royalty-free redistribution
of the Program by all those who receive copies directly or indirectly through you, then the only way
you could satisfy both it and this License would be to refrain entirely from distribution of the Pro-
gram.
If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and “any later version,” you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS
WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

one line to give the program’s name and an idea of what it does.
Copyright (C) yyyy name of author

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) yyyy name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'. This is free software, and you are welcome
to redistribute it under certain conditions; type `show c’ for details.

The hypothetical commands `show w’ and `show c’ should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w’ and `show c’; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program `Gnomovision’ (which makes passes at compilers) written by James Hacker.

signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.
GNU LESSER GENERAL PUBLIC LICENSE (LGPL)

taken from: http://www.gnu.org/copyleft/lesser.html

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

[This is the first released version of the Lesser GPL. It also counts
as the successor of the GNU Library Public License, version 2, hence
the version number 2.1.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages—typically libraries—of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.
To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the “Lesser” General Public License because it does Less to protect the user’s freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a “work based on the library” and a “work that uses the library.” The former contains code derived from the library, whereas the latter must be combined with the library in order to run.
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called “this License”). Each licensee is addressed as “you.”

A “library” means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The “Library,” below, refers to any such software library or work which has been distributed under these terms. A “work based on the Library” means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term “modification.”)

“Source code” for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library’s complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a) The modified work must itself be a software library.

   b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

   c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.
d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute
the source code, even though third parties are not compelled to copy the source along with the object code.

5. 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, is called a “work that uses the Library.” Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a “work that uses the Library” with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a “work that uses the library.” The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a “work that uses the Library” uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a “work that uses the Library” with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer’s own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable “work that uses the Library,” as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)
b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the “work that uses the Library” must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

   a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

   b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so,
and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients’ exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and “any later version,” you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.
14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

**NO WARRANTY**

15. Because the Library is licensed free of charge, there is no warranty for the Library, to the extent permitted by applicable law. Except when otherwise stated in writing the copyright holders and/or other parties provide the Library “as is” without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of the Library is with you. Should the Library prove defective, you assume the cost of all necessary servicing, repair or correction.

16. In no event unless required by applicable law or agreed to in writing will any copyright holder, or any other party who may modify and/or redistribute the Library as permitted above, be liable to you for damages, including any general, special, incidental or consequential damages arising out of the use or inability to use the Library (including but not limited to loss of data or data being rendered inaccurate or losses sustained by you or third parties or a failure of the Library to operate with any other software), even if such holder or other party has been advised of the possibility of such damages.

**END OF TERMS AND CONDITIONS**

How to Apply These Terms to Your New Libraries.

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

*one line to give the library’s name and an idea of what it does.*

Copyright (C) year name of author

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License.
License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a “copyright disclaimer” for the library, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the library `Frob' (a library for tweaking knobs) written by James Random Hacker.

signature of Ty Coon, 1 April 1990
Ty Coon, President of Vice

That's all there is to it!
BERKELEY SOFTWARE DISTRIBUTION (BSD)
taken from: http://www.opensource.org/licenses/bsd-license.html

Copyright (c) 1998, The Regents of the University of California. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. Neither name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
APACHE SOFTWARE LICENSE

taken from: http://www.apache.org/LICENSE.txt

/*----------------------------------------------------------------------------
 * The Apache Software License, Version 1.1
 *
 * Copyright (c) 2000 The Apache Software Foundation. All rights
 * reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. The end-user documentation included with the redistribution,
 *    if any, must include the following acknowledgment:
 *       "This product includes software developed by the
 *        Apache Software Foundation (http://www.apache.org/)."
 *    Alternately, this acknowledgment may appear in the software
 *    itself, if and wherever such third-party acknowledgments normally
 *    appear.
 *
 * 4. The names "Apache" and "Apache Software Foundation" must
 *    not be used to endorse or promote products derived from this
 *    software without prior written permission. For written
 *    permission, please contact apache@apache.org.
 *
 * 5. Products derived from this software may not be called "Apache",
 *    nor may "Apache" appear in their name, without prior written
 *    permission of the Apache Software Foundation.
 *
 * THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
 * OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
 * USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
 * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *-----------------------------------------------------------------------------

Draft 11/2/01 — Do Not Distribute
* This software consists of voluntary contributions made by many
* individuals on behalf of the Apache Software Foundation. For more
* information on the Apache Software Foundation, please see
*
* Portions of this software are based upon public domain software
* originally written at the National Center for Supercomputing
* Applications, University of Illinois, Urbana-Champaign.
*/
SUN PUBLIC LICENSE (SPL)
taken from: http://www.sun.com/developers/spl.html

SUN PUBLIC LICENSE
Version 1.0
April 26, 2000

1. Definitions.

1.0.1. "Commercial Use" means distribution or otherwise making the Covered Code available to a third party.

1.1. "Contributor" means each entity that creates or contributes to the creation of Modifications.

1.2. "Contributor Version" means the combination of the Original Code, prior Modifications used by a Contributor, and the Modifications made by that particular Contributor.

1.3. "Covered Code" means the Original Code or Modifications or the combination of the Original Code and Modifications, in each case including portions thereof and corresponding documentation released with the source code.

1.4. "Electronic Distribution Mechanism" means a mechanism generally accepted in the software development community for the electronic transfer of data.

1.5. "Executable" means Covered Code in any form other than Source Code.

1.6. "Initial Developer" means the individual or entity identified as the Initial Developer in the Source Code notice required by Exhibit A.

1.7. "Larger Work" means a work which combines Covered Code or portions thereof with code not governed by the terms of this License.

1.8. "License" means this document.

1.8.1. "Licensable" means having the right to grant, to the maximum extent possible, whether at the time of the initial grant or subsequently acquired, any and all of the rights conveyed herein.

1.9. "Modifications" means any addition to or deletion from the substance or structure of either the Original Code or any previous Modifications. When Covered Code is released as a series of files, a Modification is:

A. Any addition to or deletion from the contents of a file containing Original Code or previous Modifications.

B. Any new file that contains any part of the Original Code or previous Modifications.
1.10. “Original Code” means Source Code of computer software code which is described in the Source Code notice required by Exhibit A as Original Code, and which, at the time of its release under this License is not already Covered Code governed by this License.

1.10.1. “Patent Claims” means any patent claim(s), now owned or hereafter acquired, including without limitation, method, process, and apparatus claims, in any patent Licensable by grantor.

1.11. “Source Code” means the preferred form of the Covered Code for making modifications to it, including all modules it contains, plus any associated documentation, interface definition files, scripts used to control compilation and installation of an Executable, or source code differential comparisons against either the Original Code or another well known, available Covered Code of the Contributor’s choice. The Source Code can be in a compressed or archival form, provided the appropriate decompression or de-archiving software is widely available for no charge.

1.12. “You” (or “Your”) means an individual or a legal entity exercising rights under, and complying with all of the terms of, this License or a future version of this License issued under Section 6.1. For legal entities, “You” includes any entity which controls, is controlled by, or is under common control with You. For purposes of this definition, “control” means (a) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (b) ownership of more than fifty percent (50%) of the outstanding shares or beneficial ownership of such entity.

2. Source Code License.

2.1 The Initial Developer Grant.

The Initial Developer hereby grants You a world-wide, royalty-free, non-exclusive license, subject to third party intellectual property claims:

(a) under intellectual property rights (other than patent or trademark) Licensable by Initial Developer to use, reproduce, modify, display, perform, sublicense and distribute the Original Code (or portions thereof) with or without Modifications, and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using or selling of Original Code, to make, have made, use, practice, sell, and offer for sale, and/or otherwise dispose of the Original Code (or portions thereof).

(c) the licenses granted in this Section 2.1(a) and (b) are effective on the date Initial Developer first distributes Original Code under the terms of this License.

(d) Notwithstanding Section 2.1(b) above, no patent license is granted: 1) for code that You delete from the Original Code; 2) separate from the Original Code; or 3) for infringements caused by: (i) the modification of the Original Code or (ii) the combination of the Original Code with other software or devices.
2.2. Contributor Grant.

Subject to third party intellectual property claims, each Contributor hereby grants You a world-wide, royalty-free, non-exclusive license

(a) under intellectual property rights (other than patent or trademark) Licensable by Contributor, to use, reproduce, modify, display, perform, sublicense and distribute the Modifications created by such Contributor (or portions thereof) either on an unmodified basis, with other Modifications, as Covered Code and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using, or selling of Modifications made by that Contributor either alone and/or in combination with its Contributor Version (or portions of such combination), to make, use, sell, offer for sale, have made, and/or otherwise dispose of: 1) Modifications made by that Contributor (or portions thereof); and 2) the combination of Modifications made by that Contributor with its Contributor Version (or portions of such combination).

(c) the licenses granted in Sections 2.2(a) and 2.2(b) are effective on the date Contributor first makes Commercial Use of the Covered Code.

(d) notwithstanding Section 2.2(b) above, no patent license is granted: 1) for any code that Contributor has deleted from the Contributor Version; 2) separate from the Contributor Version; 3) for infringements caused by: (i) third party modifications of Contributor Version or (ii) the combination of Modifications made by that Contributor with other software (except as part of the Contributor Version) or other devices; or 4) under Patent Claims infringed by Covered Code in the absence of Modifications made by that Contributor.

3. Distribution Obligations.

3.1. Application of License.

The Modifications which You create or to which You contribute are governed by the terms of this License, including without limitation Section 2.2. The Source Code version of Covered Code may be distributed only under the terms of this License or a future version of this License released under Section 6.1, and You must include a copy of this License with every copy of the Source Code You distribute. You may not offer or impose any terms on any Source Code version that alters or restricts the applicable version of this License or the recipients’ rights hereunder. However, You may include an additional document offering the additional rights described in Section 3.5.

3.2. Availability of Source Code.

Any Modification which You create or to which You contribute must be made available in Source Code form under the terms of this License either on the same media as an Executable version or via an accepted Electronic Distribution Mechanism to anyone to whom you made an Executable version available; and if made available via Electronic Distribution Mechanism, must remain available for at least twelve (12) months after the date it initially became available, or at least six (6) months after a subsequent version of that particular Modification has been made available to such recipients. You are responsible for ensuring that the Source Code version remains available even if the Electronic Distribution Mechanism is maintained by a third party.
3.3. Description of Modifications.

You must cause all Covered Code to which You contribute to contain a file documenting the changes You made to create that Covered Code and the date of any change. You must include a prominent statement that the Modification is derived, directly or indirectly, from Original Code provided by the Initial Developer and including the name of the Initial Developer in (a) the Source Code, and (b) in any notice in an Executable version or related documentation in which You describe the origin or ownership of the Covered Code.


(a) Third Party Claims.

If Contributor has knowledge that a license under a third party's intellectual property rights is required to exercise the rights granted by such Contributor under Sections 2.1 or 2.2, Contributor must include a text file with the Source Code distribution titled “LEGAL” which describes the claim and the party making the claim in sufficient detail that a recipient will know whom to contact. If Contributor obtains such knowledge after the Modification is made available as described in Section 3.2, Contributor shall promptly modify the LEGAL file in all copies Contributor makes available thereafter and shall take other steps (such as notifying appropriate mailing lists or newsgroups) reasonably calculated to inform those who received the Covered Code that new knowledge has been obtained.

(b) Contributor APIs.

If Contributor’s Modifications include an application programming interface (“API”) and Contributor has knowledge of patent licenses which are reasonably necessary to implement that API, Contributor must also include this information in the LEGAL file.

(c) Representations.

Contributor represents that, except as disclosed pursuant to Section 3.4(a) above, Contributor believes that Contributor’s Modifications are Contributor’s original creation(s) and/or Contributor has sufficient rights to grant the rights conveyed by this License.

3.5. Required Notices.

You must duplicate the notice in Exhibit A in each file of the Source Code. If it is not possible to put such notice in a particular Source Code file due to its structure, then You must include such notice in a location (such as a relevant directory) where a user would be likely to look for such a notice. If You created one or more Modification(s) You may add your name as a Contributor to the notice described in Exhibit A. You must also duplicate this License in any documentation for the Source Code where You describe recipients’ rights or ownership rights relating to Covered Code. You may choose to offer, and to charge a fee for, warranty, support, indemnity or liability obligations to one or more recipients of Covered Code. However, You may do so only on Your own behalf, and not on behalf of the Initial Developer or any Contributor. You must make it absolutely clear than any such warranty, support, indemnity or liability obligation is offered by You alone, and You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Ini-
tial Developer or such Contributor as a result of warranty, support, indemnity or liability terms You offer.

3.6. Distribution of Executable Versions.

You may distribute Covered Code in Executable form only if the requirements of Section 3.1–3.5 have been met for that Covered Code, and if You include a notice stating that the Source Code version of the Covered Code is available under the terms of this License, including a description of how and where You have fulfilled the obligations of Section 3.2. The notice must be conspicuously included in any notice in an Executable version, related documentation or collateral in which You describe recipients' rights relating to the Covered Code. You may distribute the Executable version of Covered Code or ownership rights under a license of Your choice, which may contain terms different from this License, provided that You are in compliance with the terms of this License and that the license for the Executable version does not attempt to limit or alter the recipient's rights in the Source Code version from the rights set forth in this License. If You distribute the Executable version under a different license You must make it absolutely clear that any terms which differ from this License are offered by You alone, not by the Initial Developer or any Contributor. You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of any such terms You offer.

3.7. Larger Works.

You may create a Larger Work by combining Covered Code with other code not governed by the terms of this License and distribute the Larger Work as a single product. In such a case, You must make sure the requirements of this License are fulfilled for the Covered Code.

4. Inability to Comply Due to Statute or Regulation.

If it is impossible for You to comply with any of the terms of this License with respect to some or all of the Covered Code due to statute, judicial order, or regulation then You must: (a) comply with the terms of this License to the maximum extent possible; and (b) describe the limitations and the code they affect. Such description must be included in the LEGAL file described in Section 3.4 and must be included with all distributions of the Source Code. Except to the extent prohibited by statute or regulation, such description must be sufficiently detailed for a recipient of ordinary skill to be able to understand it.

5. Application of this License.

This License applies to code to which the Initial Developer has attached the notice in Exhibit A and to related Covered Code.

6. Versions of the License.

6.1. New Versions.

Sun Microsystems, Inc. (“Sun”) may publish revised and/or new versions of the License from time to time. Each version will be given a distinguishing version number.
6.2. Effect of New Versions.

Once Covered Code has been published under a particular version of the License, You may always continue to use it under the terms of that version. You may also choose to use such Covered Code under the terms of any subsequent version of the License published by Sun. No one other than Sun has the right to modify the terms applicable to Covered Code created under this License.

6.3. Derivative Works.

If You create or use a modified version of this License (which you may only do in order to apply it to code which is not already Covered Code governed by this License), You must: (a) rename Your license so that the phrases “Sun,” “Sun Public License,” or “SPL” or any confusingly similar phrase do not appear in your license (except to note that your license differs from this License) and (b) otherwise make it clear that Your version of the license contains terms which differ from the Sun Public License. (Filling in the name of the Initial Developer, Original Code or Contributor in the notice described in Exhibit A shall not of themselves be deemed to be modifications of this License.)

7. DISCLAIMER OF WARRANTY.

COVERED CODE IS PROVIDED UNDER THIS LICENSE ON AN “AS IS” BASIS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, WARRANTIES THAT THE COVERED CODE IS FREE OF DEFECTS, MERCHANTABLE, FIT FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE COVERED CODE IS WITH YOU. SHOULD ANY COVERED CODE PROVE DEFECTIVE IN ANY RESPECT, YOU (NOT THE INITIAL DEVELOPER OR ANY OTHER CONTRIBUTOR) ASSUME THE COST OF ANY NECESSARY SERVICING, REPAIR OR CORRECTION. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS LICENSE. NO USE OF ANY COVERED CODE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

8. TERMINATION.

8.1. This License and the rights granted hereunder will terminate automatically if You fail to comply with terms herein and fail to cure such breach within 30 days of becoming aware of the breach. All sublicenses to the Covered Code which are properly granted shall survive any termination of this License. Provisions which, by their nature, must remain in effect beyond the termination of this License shall survive.

8.2. If You initiate litigation by asserting a patent infringement claim (excluding declaratory judgment actions) against Initial Developer or a Contributor (the Initial Developer or Contributor against whom You file such action is referred to as “Participant”) alleging that:

(a) such Participant’s Contributor Version directly or indirectly infringes any patent, then any and all rights granted by such Participant to You under Sections 2.1 and/or 2.2 of this License shall, upon 60 days notice from Participant terminate prospectively, unless if within 60 days after receipt of
notice You either: (i) agree in writing to pay Participant a mutually agreeable reasonable royalty for
Your past and future use of Modifications made by such Participant, or (ii) withdraw Your litigation
claim with respect to the Contributor Version against such Participant. If within 60 days of notice, a
reasonable royalty and payment arrangement are not mutually agreed upon in writing by the parties
or the litigation claim is not withdrawn, the rights granted by Participant to You under Sections 2.1
and/or 2.2 automatically terminate at the expiration of the 60 day notice period specified above.

(b) any software, hardware, or device, other than such Participant’s Contributor Version,
directly or indirectly infringes any patent, then any rights granted to You by such Participant under
Sections 2.1(b) and 2.2(b) are revoked effective as of the date You first made, used, sold, distributed,
or had made, Modifications made by that Participant.

8.3. If You assert a patent infringement claim against Participant alleging that such Particip-
pant’s Contributor Version directly or indirectly infringes any patent where such claim is resolved
(such as by license or settlement) prior to the initiation of patent infringement litigation, then the
reasonable value of the licenses granted by such Participant under Sections 2.1 or 2.2 shall be taken
into account in determining the amount or value of any payment or license.

8.4. In the event of termination under Sections 8.1 or 8.2 above, all end user license agree-
ments (excluding distributors and resellers) which have been validly granted by You or any distribu-
tor hereunder prior to termination shall survive termination.

9. LIMITATION OF LIABILITY.

UNDER NO CIRCUMSTANCES AND UNDER NO LEGAL THEORY,
WHETHER TORT (INCLUDING NEGLIGENCE), CONTRACT, OR OTHERWISE,
SHALL YOU, THE INITIAL DEVELOPER, ANY OTHER CONTRIBUTOR, OR ANY
DISTRIBUTOR OF COVERED CODE, OR ANY SUPPLIER OF ANY OF SUCH PAR-
TIES, BE LIABLE TO ANY PERSON FOR ANY INDIRECT, SPECIAL, INCIDENTAL,
OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER INCLUDING, WITH-
OUT LIMITATION, DAMAGES FOR LOSS OF GOODWILL, WORK STOPPAGE,
COMPUTER FAILURE OR MALFUNCTION, OR ANY AND ALL OTHER COMMER-
CIAL DAMAGES OR LOSSES, EVEN IF SUCH PARTY SHALL HAVE BEEN
INFORMED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION OF
LIABILITY SHALL NOT APPLY TO LIABILITY FOR DEATH OR PERSONAL
INJURY RESULTING FROM SUCH PARTY’S NEGLIGENCE TO THE EXTENT
APPLICABLE LAW PROHIBITS SUCH LIMITATION. SOME JURISDICTIONS DO
NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSE-
QUENTIAL DAMAGES, SO THIS EXCLUSION AND LIMITATION MAY NOT
APPLY TO YOU.

10. U.S. GOVERNMENT END USERS.

The Covered Code is a “commercial item,” as that term is defined in 48 C.F.R. 2.101 (Oct.
1995), consisting of “commercial computer software” and “commercial computer software documen-
tation,” as such terms are used in 48 C.F.R. 12.212 (Sept. 1995). Consistent with 48 C.F.R. 12.212
and 48 C.F.R. 227.7202-1 through 227.7202-4 (June 1995), all U.S. Government End Users acquire
Covered Code with only those rights set forth herein.
11. MISCELLANEOUS.

This License represents the complete agreement concerning subject matter hereof. If any provision of this License is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable. This License shall be governed by California law provisions (except to the extent applicable law, if any, provides otherwise), excluding its conflict-of-law provisions. With respect to disputes in which at least one party is a citizen of, or an entity chartered or registered to do business in the United States of America, any litigation relating to this License shall be subject to the jurisdiction of the Federal Courts of the Northern District of California, with venue lying in Santa Clara County, California, with the losing party responsible for costs, including without limitation, court costs and reasonable attorneys' fees and expenses. The application of the United Nations Convention on Contracts for the International Sale of Goods is expressly excluded. Any law or regulation which provides that the language of a contract shall be construed against the drafter shall not apply to this License.

12. RESPONSIBILITY FOR CLAIMS.

As between Initial Developer and the Contributors, each party is responsible for claims and damages arising, directly or indirectly, out of its utilization of rights under this License and You agree to work with Initial Developer and Contributors to distribute such responsibility on an equitable basis. Nothing herein is intended or shall be deemed to constitute any admission of liability.

13. MULTIPLE-LICENSED CODE.

Initial Developer may designate portions of the Covered Code as “Multiple-Licensed”. “Multiple-Licensed” means that the Initial Developer permits you to utilize portions of the Covered Code under Your choice of the alternative licenses, if any, specified by the Initial Developer in the file described in Exhibit A.
Exhibit A—Sun Public License Notice.

The contents of this file are subject to the Sun Public License Version 1.0 (the "License"); you may not use this file except in compliance with the License. A copy of the License is available at http://www.sun.com

The Original Code is _________________. The Initial Developer of the Original Code is Sun Microsystems, Inc. Portions Copyright _____ Sun Microsystems, Inc. All Rights Reserved.

Contributor(s): _________________________________.

Alternatively, the contents of this file may be used under the terms of the _____ license (the ![License]), in which case the provisions of [_____] License are applicable instead of those above. If you wish to allow use of your version of this file only under the terms of the [_____] License and not to allow others to use your version of this file under the SPL, indicate your decision by deleting the provisions above and replace them with the notice and other provisions required by the [_____] License. If you do not delete the provisions above, a recipient may use your version of this file under either the SPL or the [_____] License.”

[NOTE: The text of this Exhibit A may differ slightly from the text of the notices in the Source Code files of the Original Code. You should use the text of this Exhibit A rather than the text found in the Original Code Source Code for Your Modifications.]
Sun Industry Standards Source License
Version 1.1

1.0 DEFINITIONS

1.1 “Commercial Use” means distribution or otherwise making the Original Code available to a third party.

1.2 “Contributor Version” means the combination of the Original Code, and the Modifications made by that particular Contributor.

1.3 “Electronic Distribution Mechanism” means a mechanism generally accepted in the software development community for the electronic transfer of data.

1.4 “Executable” means Original Code in any form other than Source Code.

1.5 “Initial Developer” means the individual or entity identified as the Initial Developer in the Source Code notice required by Exhibit A.

1.6 “Larger Work” means a work which combines Original Code or portions thereof with code not governed by the terms of this License.

1.7 “License” means this document.

1.8 “Licensable” means having the right to grant, to the maximum extent possible, whether at the time of the initial grant or subsequently acquired, any and all of the rights conveyed herein.

1.9 “Modifications” means any addition to or deletion from the substance or structure of either the Original Code or any previous Modifications. A Modification is:

   A. Any addition to or deletion from the contents of a file containing Original Code or previous Modifications.

   B. Any new file that contains any part of the Original Code or previous Modifications.

1.10 “Original Code” means Source Code of computer software code which is described in the Source Code notice required by Exhibit A as Original Code.

1.11 “Patent Claims” means any patent claim(s), now owned or hereafter acquired, including without limitation, method, process, and apparatus claims, in any patent Licensable by grantor.

1.12 “Source Code” means the preferred form of the Original Code for making modifications to it, including all modules it contains, plus any associated interface definition files, or scripts used to control compilation and installation of an Executable.
1.13 “Standards” means the standards identified in Exhibit B.

1.14 “You” (or “Your”) means an individual or a legal entity exercising rights under, and complying with all of the terms of, this License or a future version of this License issued under Section 6.1. For legal entities, “You” includes any entity which controls, is controlled by, or is under common control with You. For purposes of this definition, “control” means (a) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (b) ownership of more than fifty percent (50%) of the outstanding shares or beneficial ownership of such entity.

2.0 SOURCE CODE LICENSE

2.1 The Initial Developer Grant
The Initial Developer hereby grants You a world-wide, royalty-free, non-exclusive license, subject to third party intellectual property claims:

(a) under intellectual property rights (other than patent or trademark) Licensable by Initial Developer to use, reproduce, modify, display, perform, sublicense and distribute the Original Code (or portions thereof) with or without Modifications, and/or as part of a Larger Work; and

(b) under Patents Claims infringed by the making, using or selling of Original Code, to make, have made, use, practice, sell, and offer for sale, and/or otherwise dispose of the Original Code (or portions thereof).

(c) the licenses granted in this Section 2.1(a) and (b) are effective on the date Initial Developer first distributes Original Code under the terms of this License.

(d) Notwithstanding Section 2.1(b) above, no patent license is granted: 1) for code that You delete from the Original Code; 2) separate from the Original Code; or 3) for infringements caused by: (i) the modification of the Original Code or (ii) the combination of the Original Code with other software or devices, including but not limited to Modifications.

3.0 DISTRIBUTION OBLIGATIONS

3.1 Application of License.
The Source Code version of Original Code may be distributed only under the terms of this License or a future version of this License released under Section 6.1, and You must include a copy of this License with every copy of the Source Code You distribute. You may not offer or impose any terms on any Source Code version that alters or restricts the applicable version of this License or the recipients’ rights hereunder. Your license for shipment of the Contributor Version is conditioned upon Your full compliance with this Section. The Modifications which You create must comply with all requirements set out by the Standards body in effect one hundred twenty (120) days before You ship the Contributor Version. In the event that the Modifications do not meet such requirements, You agree to publish either (i) any deviation from the Standards protocol resulting from implementation of Your Modifications and a reference implementation of Your Modifications or (ii) Your Modifications in Source Code form, and to make any such deviation and reference implementation or Modifications available to all third parties under the same terms as this license on a royalty free basis within thirty (30) days of Your first customer shipment of Your Modifications.
3.2 Required Notices.
You must duplicate the notice in Exhibit A in each file of the Source Code. If it is not possible to put such notice in a particular Source Code file due to its structure, then You must include such notice in a location (such as a relevant directory) where a user would be likely to look for such a notice. If You created one or more Modification(s) You may add Your name as a Contributor to the notice described in Exhibit A. You must also duplicate this License in any documentation for the Source Code where You describe recipients' rights or ownership rights relating to Initial Code. You may choose to offer, and to charge a fee for, warranty, support, indemnity or liability obligations to one or more recipients of Your version of the Code. However, You may do so only on Your own behalf, and not on behalf of the Initial Developer. You must make it absolutely clear than any such warranty, support, indemnity or liability obligation is offered by You alone, and You hereby agree to indemnify the Initial Developer for any liability incurred by the Initial Developer as a result of warranty, support, indemnity or liability terms You offer.

3.3 Distribution of Executable Versions.
You may distribute Original Code in Executable and Source form only if the requirements of Sections 3.1 and 3.2 have been met for that Original Code, and if You include a notice stating that the Source Code version of the Original Code is available under the terms of this License. The notice must be conspicuously included in any notice in an Executable or Source versions, related documentation or collateral in which You describe recipients' rights relating to the Original Code. You may distribute the Executable and Source versions of Your version of the Code or ownership rights under a license of Your choice, which may contain terms different from this License, provided that You are in compliance with the terms of this License. If You distribute the Executable and Source versions under a different license You must make it absolutely clear that any terms which differ from this License are offered by You alone, not by the Initial Developer. You hereby agree to indemnify the Initial Developer for any liability incurred by the Initial Developer as a result of any such terms You offer.

3.4 Larger Works.
You may create a Larger Work by combining Original Code with other code not governed by the terms of this License and distribute the Larger Work as a single product. In such a case, You must make sure the requirements of this License are fulfilled for the Original Code.

4.0 INABILITY TO COMPLY DUE TO STATUTE OR REGULATION
If it is impossible for You to comply with any of the terms of this License with respect to some or all of the Original Code due to statute, judicial order, or regulation then You must: (a) comply with the terms of this License to the maximum extent possible; and (b) describe the limitations and the code they affect. Such description must be included in the LEGAL file described in Section 3.2 and must be included with all distributions of the Source Code. Except to the extent prohibited by statute or regulation, such description must be sufficiently detailed for a recipient of ordinary skill to be able to understand it.

5.0 APPLICATION OF THIS LICENSE
This License applies to code to which the Initial Developer has attached the notice in Exhibit A and to related Modifications as set out in Section 3.1.
6.0 VERSIONS OF THE LICENSE

6.1 New Versions.
Sun may publish revised and/or new versions of the License from time to time. Each version will be given a distinguishing version number.

6.2 Effect of New Versions.
Once Original Code has been published under a particular version of the License, You may always continue to use it under the terms of that version. You may also choose to use such Original Code under the terms of any subsequent version of the License published by Sun. No one other than Sun has the right to modify the terms applicable to Original Code.

7.0 DISCLAIMER OF WARRANTY

ORIGINAL CODE IS PROVIDED UNDER THIS LICENSE ON AN “AS IS” BASIS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, WARRANTIES THAT THE ORIGINAL CODE IS FREE OF DEFECTS, MERCHANTABILITY, FIT FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE ORIGINAL CODE IS WITH YOU. SHOULD ANY ORIGINAL CODE PROVE DEFECTIVE IN ANY RESPECT, YOU (NOT THE INITIAL DEVELOPER) ASSUME THE COST OF ANY NECESSARY SERVICING, REPAIR OR CORRECTION. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS LICENSE. NO USE OF ANY ORIGINAL CODE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

8.0 TERMINATION

8.1 This License and the rights granted hereunder will terminate automatically if You fail to comply with terms herein and fail to cure such breach within 30 days of becoming aware of the breach. All sublicenses to the Original Code which are properly granted shall survive any termination of this License. Provisions which, by their nature, must remain in effect beyond the termination of this License shall survive.

8.2 In the event of termination under Section 8.1 above, all end user license agreements (excluding distributors and resellers) which have been validly granted by You or any distributor hereunder prior to termination shall survive termination.

9.0 LIMIT OF LIABILITY

UNDER NO CIRCUMSTANCES AND UNDER NO LEGAL THEORY, WHETHER TORT (INCLUDING NEGLIGENCE), CONTRACT, OR OTHERWISE, SHALL YOU, THE INITIAL DEVELOPER, ANY OTHER CONTRIBUTOR, OR ANY DISTRIBUTOR OF ORIGINAL CODE, OR ANY SUPPLIER OF ANY OF SUCH PARTIES, BE LIABLE TO ANY PERSON FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER FAILURE OR MALFUNCTION, OR ANY AND ALL OTHER COMMERCIAL
DAMAGES OR LOSSES, EVEN IF SUCH PARTY SHALL HAVE BEEN INFORMED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION OF LIABILITY SHALL NOT APPLY TO LIABILITY FOR DEATH OR PERSONAL INJURY RESULTING FROM SUCH PARTY’S NEGLIGENCE TO THE EXTENT APPLICABLE LAW PROHIBITS SUCH LIMITATION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THIS EXCLUSION AND LIMITATION MAY NOT APPLY TO YOU.

10.0 U.S. GOVERNMENT END USERS

U.S. Government: If this Software is being acquired by or on behalf of the U.S. Government or by a U.S. Government prime contractor or subcontractor (at any tier), then the Government's rights in the Software and accompanying documentation shall be only as set forth in this license; this is in accordance with 48 C.F.R. 227.7201 through 227.7202-4 (for Department of Defense (DoD) acquisitions) and with 48 C.F.R. 2.101 and 12.212 (for non-DoD acquisitions).

11.0 MISCELLANEOUS

This License represents the complete agreement concerning subject matter hereof. If any provision of this License is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable. This License shall be governed by California law provisions (except to the extent applicable law, if any, provides otherwise), excluding its conflict-of-law provisions. With respect to disputes in which at least one party is a citizen of, or an entity chartered or registered to do business in the United States of America, any litigation relating to this License shall be subject to the jurisdiction of the Federal Courts of the Northern District of California, with venue lying in Santa Clara County, California, with the losing party responsible for costs, including without limitation, court costs and reasonable attorneys' fees and expenses. The application of the United Nations Convention on Contracts for the International Sale of Goods is expressly excluded. Any law or regulation which provides that the language of a contract shall be construed against the drafter shall not apply to this License.
EXHIBIT A—Sun Standards License

The contents of this file are subject to the Sun Standards License Version 1.1 (the "License"); You may not use this file except in compliance with the License. You may obtain a copy of the License at ______________________________.

Software distributed under the License is distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License.

The Original Code is ______________________________________.

The Initial Developer of the Original Code is: Sun Microsystems, Inc.

Portions created by: ______________________________________

are Copyright (C): ______________________________________

All Rights Reserved.

Contributor(s): ______________________________________

EXHIBIT B— Standards

The Standard is defined as the following:


OpenOffice.org Application Programming Interface Specification, located at http://api.openoffice.org
Sun Community Source License
Version 3.0

Jini Technology Specific Attachment v 1.0
(SCSL3/Jini TSA 1.0)

Contents

I. Definitions
II. Purposes
III. Research Use Rights
IV. Restrictions & Community Responsibilities
V. Governance

Attachment A-1: Community Member Certificate Attachment
Attachment A-2: Student Acknowledgment
Attachment B: (Additional Terms/Responsibilities)
Commercial Use Supplement to Sun Community Source Code License
Technology Specific Attachment to the Sun Community Source License

I. DEFINITIONS


“Community Member” means You, Original Contributor and any other party that has entered into and has in effect a version of this License (or who is similarly authorized and obligated by Sun) for the Technology with Original Contributor.

“Contributed Code” means Error Corrections, Shared Modifications and any other code other than Reference Code made available to all Community Members in accordance with this License.

“Contributed Code Specifications” means the functional, interface and operational specifications and documentation for Contributed Code.

“Contributor” means any Community Member who makes available Contributed Code.

“Covered Code” means any and all code (including Modifications) implementing all or any portion of the Technology Specifications.

“Error Correction(s)” mean Modifications which correct any failure of Community Code to conform to any aspect of the Technology Specifications.
“Interfaces” means classes or other programming code or specifications designed for use with the Technology comprising a means or link for invoking functionality, operations or protocols and which are additional to or extend the interfaces designated in the Technology Specifications.

“Modifications” means any (i) change or addition to Covered Code, or (ii) new source or object code implementing any portion of the Technology Specifications, but (iii) excluding any incorporated Reference Code.

“Original Contributor” means Sun Microsystems, Inc., its affiliates, successors and assigns.

“Reference Code” means source code for the Technology designated by Original Contributor at the Technology Site from time to time.

“Research Use” means research, evaluation, development, educational or personal and individual use, excluding use or distribution for direct or indirect commercial (including strategic) gain or advantage.

“Shared Modifications” means those Modifications which Community Members elect to share with other Community Members pursuant to Section III.B.

“Technology Specifications” means the functional, interface and operational specifications and documentation for the Technology designated by Original Contributor at the Technology Site from time to time.

“Technology” means the technology described in and contemplated by the Technology Specifications and which You have received pursuant to this License.

“Technology Site” means the website designated by Original Contributor for accessing Community Code and Technology Specifications.

“You” means the individual executing this license or the legal entity or entities represented by the individual executing this license. “Your” is the possessive of “You.”

II. PURPOSES

Original Contributor is licensing the Reference Code and Technology Specifications and is permitting implementation of Technology under and subject to this Sun Community Source License (the “License”) to promote research, education, innovation and product development using the Technology.

COMMERCIAL USE AND DISTRIBUTION OF TECHNOLOGY IS PERMITTED ONLY UNDER OPTIONAL SUPPLEMENTS TO THIS LICENSE.

III. RESEARCH USE RIGHTS

A. From Original Contributor. Subject to and conditioned upon Your full compliance with the terms and conditions of this License, including Sections IV (Restrictions and Community Responsibilities) and V.E.7 (International Use), Original Contributor:
1. grants to You a non-exclusive, worldwide and royalty-free license to the extent of Original Contributor's copyrights and trade secret rights in and covering the Reference Code and Technology Specifications to do the following for Your Research Use only:
   a) reproduce, prepare derivative works of, display and perform the Reference Code, in whole or in part, alone or as part of Covered Code;
   b) reproduce, prepare derivative works of and display the Technology Specifications;
   c) distribute source or object code copies of Reference Code, in whole or in part, alone or as part Covered Code, to other Community Members or to students; and
   d) distribute object code copies of Reference Code, in whole or in part, alone or as part of object code copies of Covered Code, to third parties.

2. will not, during the term of Your License, bring against You any claim alleging that Your using, making, having made, importing or distributing Community Code for Your Research Use, insofar as permitted under Section III.A.1 of this License, necessarily infringes any patent now owned or hereafter acquired by Original Contributor whose claims cover subject matter contained in or embodied by the Reference Code or which would necessarily be infringed by the use or distribution of any and all implementations of the Technology Specifications.

3. grants to You a non-exclusive, worldwide and royalty-free license, to the extent of its intellectual property rights therein, to use (a) Original Contributor's class, interface and package names only insofar as necessary to accurately reference or invoke Your Modifications for Research Use, and (b) any associated software tools, documents and information provided by Original Contributor at the Technology Site for use in exercising the above license rights.

B. Contributed Code. Subject to and conditioned upon compliance with the terms and conditions of this License, including Sections IV (Restrictions and Community Responsibilities) and V.E.7 (International Use), each Community Member:

1. grants to each Community Member a non-exclusive, worldwide and royalty-free license to the extent of such Community Member's copyrights and trade secret rights in and covering its Contributed Code, to reproduce, modify, display and distribute Contributed Code, in whole or in part, in source code and object code form, to the same extent as permitted under such Community Member's License with Original Contributor (including all supplements thereto).

2. will not, during the term of the Community Member's License, bring against any Community Member any claim alleging that using, making, having made, importing or distributing Contributed Code as permitted under this License (including any supplements) infringes any patents or patent applications now owned or hereafter acquired by such Community Member which patents or patent applications are infringed by using, making, having made, selling, offering for sale, importing or otherwise transferring the Contributed Code (“Community Member Patents”). This covenant shall apply to the combination of the Contributed Code with other Covered Code if, at the time the Contributed Code is posted, such addition of the Contributed Code causes
such combination to be covered by the Community Member Patents. The covenant shall not apply to any other combinations which include the Contributed Code or to the use or distribution of modified Contributed Code where the modifications made by the Community Member add to the functions performed by the Contributed Code in question and where, in the absence of such modifications, there would be no infringement of a Community Member Patent.

3. grants to Original Contributor, in addition to the rights set forth in Sections III.B.1 and III.B.2, the right to sublicense all such rights in Contributed Code, in whole or in part, as part of Reference Code or other technologies based in whole or in part on Reference Code or Technology and to copy, distribute, modify and prepare derivative works of Contributed Code Specifications, in whole or in part, in connection with the exercise of such rights.

C. Subcontracting. You may provide Covered Code to a contractor for the sole purpose of providing development services exclusively to You consistent with Your rights under this License. Such Contractor must be a Community Member or have executed an agreement with You that is consistent with Your rights and obligations under this License. Such subcontractor must assign exclusive rights in all work product to You. You agree that such work product is to be treated as Covered Code.

D. No Implied Licenses. Neither party is granted any right or license other than the licenses and covenants expressly set out herein. Other than the licenses and covenants expressly set out herein, Original Contributor retains all right, title and interest in Reference Code and Technology Specifications and You retain all right, title and interest in Your Modifications and associated specifications. Except as expressly permitted herein, You must not otherwise use any package, class or interface naming conventions that appear to originate from Original Contributor.

IV. RESTRICTIONS AND COMMUNITY RESPONSIBILITIES

As a condition to Your license and other rights and immunities, You must comply with the restrictions and responsibilities set forth below, as modified or supplemented, if at all, in Attachment B, Additional Research Use Terms and Conditions.

A. Source Code Availability. You must provide source code and any specifications for Your Error Corrections to Original Contributor as soon as practicable. You may provide other Contributed Code to Original Contributor at any time, in Your discretion. Original Contributor may, in its discretion, post Your Contributed Code and Contributed Code Specifications on the Technology Site. You may post Your Contributed Code and/or Contributed Code Specifications on another website of Your choice; provided, source code of Community Code and Technology Specifications must be provided to Community Members only and only following certification of Community Member status as required under Section IV.D.

B. Notices. You must reproduce without alteration copyright and other proprietary notices in any Covered Code that You distribute. The statement, “Use and Distribution is subject to the Sun Community Source License available at http://sun.com/software/communitysource” must appear prominently in Your Modifications and, in all cases, in the same file as all Your copyright and other proprietary notices.
C. Modifications. You must include a diff file with Your Contributed Code that identifies and
details the changes or additions You made, the version of Reference Code or Contributed Code You
used and the date of such changes or additions. In addition, You must provide any Contributed
Code Specifications for Your Contributed Code. Your Modifications are Covered Code and You
expressly agree that use and distribution, in whole or in part, of Your Modifications shall only be
done in accordance with and subject to this License.

D. Distribution Requirements. You may distribute object code of Covered Code to third parties
for Research Use only pursuant to a license of Your choice which is consistent with this License. You
may distribute source code of Covered Code and the Technology Specifications for Research Use
only to (i) Community Members from whom You have first obtained a certification of status in the
form set forth in Attachment A-1, and (ii) students from whom You have first obtained an executed
acknowledgment in the form set forth in Attachment A-2. You must keep a copy of each such cer-
ificate and acknowledgment You obtain and provide a copy to Original Contributor, if requested.

E. Extensions.

1. You may create and add Interfaces but, unless expressly permitted at the Technology
Site, You must not incorporate any Reference Code in Your Interfaces. If You choose to
disclose or permit disclosure of Your Interfaces to even a single third party for the pur-
poses of enabling such third party to independently develop and distribute (directly or
indirectly) technology which invokes such Interfaces, You must then make the Inter-
faces open by (a) promptly following completion thereof, publishing to the industry, on
a non-confidential basis and free of all copyright restrictions, a reasonably detailed, cur-
rent and accurate specification for the Interfaces, and (b) as soon as reasonably possi-
ble, but in no event more than thirty (30) days following publication of Your
specification, making available on reasonable terms and without discrimination, a rea-
sonably complete and practicable test suite and methodology adequate to create and test
implementations of the Interfaces by a reasonably skilled technologist.

2. You shall not assert any intellectual property rights You may have covering Your Inter-
faces which would necessarily be infringed by the creation, use or distribution of all
reasonable independent implementations of Your specification of such Interfaces by a
Community Member or Original Contributor. Nothing herein is intended to prevent You
from enforcing any of Your intellectual property rights covering Your specific imple-
mentation of Your Interfaces or functionality using such Interfaces other than as specif-
ically set forth in this Section IV.E.2.

V. GOVERNANCE.

A. License Versions.

Only Original Contributor may promulgate new versions of this License. Once You have accepted
Reference Code, Technology Specifications, Contributed Code and/or Contributed Code Specifi-
cations under a version of this License, You may continue to use such version of Reference Code, Tech-
nology Specifications, Contributed Code and/or Contributed Code Specifications under that
version of the License. New code and specifications which You may subsequently choose to accept
will be subject to any new License in effect at the time of Your acceptance of such code and specifications.

B. Disclaimer Of Warranties.

1. COVERED CODE, ALL TECHNOLOGY SPECIFICATIONS AND CONTRIBUTED CODE SPECIFICATIONS ARE PROVIDED “AS IS,” WITHOUT WARRANTIES OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, WARRANTIES THAT ANY SUCH COVERED CODE, TECHNOLOGY SPECIFICATIONS AND CONTRIBUTED CODE SPECIFICATIONS ARE FREE OF DEFECTS, MERCHANTABILITY, FIT FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT OF THIRD PARTY RIGHTS. YOU AGREE THAT YOU BEAR THE ENTIRE RISK IN CONNECTION WITH YOUR USE AND DISTRIBUTION OF ANY AND ALL COVERED CODE, TECHNOLOGY SPECIFICATIONS AND CONTRIBUTED CODE SPECIFICATIONS UNDER THIS LICENSE. NO USE OF ANY COVERED CODE, TECHNOLOGY SPECIFICATIONS OR CONTRIBUTED CODE SPECIFICATIONS IS AUTHORIZED EXCEPT SUBJECT TO AND IN CONSIDERATION FOR THIS DISCLAIMER.

2. You understand that, although each Community Member grants the licenses set forth in the License and any supplements hereto, no assurances are provided by any Community Member that Covered Code or any specifications do not infringe the intellectual property rights of any third party.

3. You acknowledge that Reference Code and Technology Specifications are neither designed nor intended for use in the design, construction, operation or maintenance of any nuclear facility.

C. Limitation Of Liability.

1. Infringement. Each Community Member disclaims any liability to all other Community Members for claims brought by any third party based on infringement of intellectual property rights. Original Contributor represents that, to its knowledge, it has sufficient copyrights to allow You to use and distribute the Reference Code as herein permitted (including as permitted in any Supplement hereto) and You represent that, to Your knowledge, You have sufficient copyrights to allow each Community Member and Original Contributor to use and distribute Your Shared Modifications and Error Corrections as herein permitted (including as permitted in any supplements to the License). You agree to notify Original Contributor should You become aware of any potential or actual infringement of the Technology or any of Original Contributor's intellectual property rights in the Technology, Reference Code or Technology Specifications.

2. Suspension. If any portion of, or functionality implemented by, the Reference Code, Technology or Technology Specifications becomes the subject of a claim or threatened claim of infringement (“Affected Materials”), Original Contributor may, in its unrestricted discretion, suspend Your rights to use and distribute the Affected Materials under this License. Such suspension of rights will be effective immediately upon Original Contributor's posting of notice of suspension on the Technology Site. Original Con-
tributor has no obligation to lift the suspension of rights relative to the Affected Materials until a final, non-appealable determination is made by a court or governmental agency of competent jurisdiction that Original Contributor is legally able, without the payment of a fee or royalty, to reinstate Your rights to the Affected Materials to the full extent contemplated hereunder. Upon such determination, Original Contributor will lift the suspension by posting a notice to such effect on the Technology Site. Nothing herein shall be construed to prevent You, at Your option and expense, and subject to applicable law and the restrictions and responsibilities set forth in this License and any Supplements, from replacing Reference Code in Affected Materials with non-infringing code or independently negotiating, without compromising or prejudicing Original Contributor's position, to obtain the rights necessary to use Affected Materials as herein permitted.

3. Disclaimer. ORIgINAL CONTRIBUTOR’S LIABILITY TO YOU FOR ALL CLAIMS RELATING TO THIS LICENSE OR ANY SUPPLEMENT HERETO, WHETHER FOR BREACH OR TORT, IS LIMITED TO THE GREATER OF ONE THOUSAND DOLLARS (US$1000.00) OR THE FULL AMOUNT PAID BY YOU FOR THE MATERIALS GIVING RISE TO THE CLAIM, IF ANY. IN NO EVENT WILL ORIGINAL CONTRIBUTOR BE LIABLE FOR ANY INDIRECT, PUNITIVE, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES IN CONNECTION WITH OR ARISING OUT OF THIS LICENSE (INCLUDING, WITHOUT LIMITATION, LOSS OF PROFITS, USE, DATA OR ECONOMIC ADVANTAGE OF ANY SORT), HOWEVER IT ARISES AND ON ANY THEORY OF LIABILITY (including negligence), WHETHER OR NOT ORIGINAL CONTRIBUTOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE AND NOTWITHSTANDING FAILURE OF THE ESSENTIAL PURPOSE OF ANY REMEDY.

D. Termination.

You may terminate this License at any time by notifying Original Contributor in writing.

All Your rights will terminate under this License if You fail to comply with any of the material terms or conditions of this License and do not cure such failure in a reasonable period of time after becoming aware of such noncompliance.

If You institute patent litigation against a Community Member with respect to a patent applicable to Community Code, then any patent licenses or covenants granted by such Community Member to You under this License shall terminate as of the date such litigation is filed. In addition, if You institute patent litigation against any Community Member or Original Contributor alleging that Reference Code, Technology or Technology Specifications infringe Your patent(s), then the rights granted to You under Section III.A above will terminate.

Upon termination, You must discontinue all uses and distribution of Community Code, except that You may continue to use, reproduce, prepare derivative works of, display and perform Your Modifications, so long as the license grants and covenants of this license are not required to do so, for purposes other than to implement functionality designated in any portion of the Technology
Specifications. Properly granted sublicenses to third parties will survive termination. Provisions which, by their nature, should remain in effect following termination survive.

E. Miscellaneous.

1. Trademark. You agree to comply with Original Contributors Trademark & Logo Usage Requirements, as modified from time to time, available at the Technology Site. Except as expressly provided in this License, You are granted no rights in or to any Sun, Jini, Jiro or Java trademarks now or hereafter used or licensed by Original Contributor (the “Sun Trademarks”). You agree not to (a) challenge Original Contributor’s ownership or use of Sun Trademarks; (b) attempt to register any Sun Trademarks, or any mark or logo substantially similar thereto; or (c) incorporate any Sun Trademarks into Your own trademarks, product names, service marks, company names or domain names.

2. Integration and Assignment. Original Contributor may assign this Research Use License to another by written notification to the other party. This License represents the complete agreement of the parties concerning the subject matter hereof.

3. Severability. If any provision of this License is held unenforceable, such provision shall be reformed to the extent necessary to make it enforceable unless to do so would defeat the intent of the parties, in which case, this License shall terminate.

4. Governing Law. This License is governed by the laws of the United States and the State of California, as applied to contracts entered into and performed in California between California residents. The United Nations Convention on Contracts for the International Sale of Goods shall not apply. Nor shall any law or regulation which provides that a contract be construed against the drafter.

5. Dispute Resolution.

   a) Any dispute arising out of or relating to this License shall be finally settled by arbitration as set forth in this Section, except that either party may bring an action in a court of competent jurisdiction (which jurisdiction shall be exclusive), relative to any dispute relating to such party’s intellectual property rights. Arbitration will be administered (i) by the American Arbitration Association (AAA), (ii) in accordance with the rules of the United Nations Commission on International Trade Law (UNCITRAL) (the “Rules”) in effect at the time of arbitration, modified as set forth herein, and (iii) by an arbitrator described in Section 5.b who shall apply the governing laws required under Section V.E.4 above. Judgment upon the award rendered by the arbitrator may be entered in any court having jurisdiction to enforce such award. The arbitrator must not award damages in excess of or of a different type than those permitted by this License and any such award is void.

   b) All proceedings will be in English and conducted by a single arbitrator selected in accordance with the Rules who is fluent in English, familiar with technology matters pertinent in the dispute and is either a retired judge or practicing attorney having at least ten (10) years litigation experience. Venue for arbitration will be in San Francisco, California, unless the parties agree otherwise. Each party will be required to produce documents relied upon in the arbitration and to respond to no more than twenty-five single
question interrogatories. All awards are payable in US dollars and may include for the prevailing party (i) pre-judgment interest, (ii) reasonable attorneys' fees incurred in connection with the arbitration, and (iii) reasonable costs and expenses incurred in enforcing the award.

6. U.S. Government. If this Software is being acquired by or on behalf of the U.S. Government or by a U.S. Government prime contractor or subcontractor (at any tier), the Government's rights in this Software and accompanying documentation shall be only as set forth in this license, in accordance with 48 CFR 227.7201 through 227.7202-4 (for Department of Defense acquisitions) and with 48 CFR 2.101 and 12.212 (for non-DoD acquisitions).

   a) Covered Code is subject to US export control laws and may be subject to export or import regulations in other countries. Each party shall comply fully with all such laws and regulations and acknowledges its responsibility to obtain such licenses to export, re-export or import as may be required. You must pass through these obligations to all Your licensees.
   b) You must not distribute Reference Code or Technology Specifications into countries other than those listed on the Technology Site by Original Contributor, from time to time.

READ ALL THE TERMS OF THIS LICENSE CAREFULLY BEFORE ACCEPTING. IF YOU ARE AGREEING TO THIS LICENSE ON BEHALF OF A COMPANY, YOU REPRESENT THAT YOU ARE AUTHORIZED TO BIND THE COMPANY TO THE LICENSE.

WHETHER YOU ARE ACTING ON YOUR OWN BEHALF OR THAT OF A COMPANY, YOU MUST BE OF MAJORITY AGE AND OTHERWISE COMPETENT TO ENTER INTO CONTRACTS.

ATTACHMENT A-1
COMMUNITY MEMBER CERTIFICATE

You certify that You are a Licensee in good standing under the Sun Community Source License for the ______________ Technology (fill in applicable Technology and Version) (the “License”) and that You agree to use and distribute code, documentation and information You may obtain pursuant to this certification only in accordance with the terms and subject to the conditions of the License.

Signature:____________________

Printed Name and Title:_____________________

Company _____________________
ATTACHMENT A-2
STUDENT ACKNOWLEDGMENT

You acknowledge that this software and related documentation has been obtained by your educational institution subject to the Sun Community Source License (the "License"). You have been provided with access to the software and documentation for use only in connection with your course work as a matriculated student of your educational institution. Commercial use of the software and documentation is expressly prohibited.

THIS SOFTWARE AND RELATED DOCUMENTATION CONTAINS PROPRIETARY MATERIALS OF SUN MICROSYSTEMS, INC. PROTECTED BY VARIOUS INTELLECTUAL PROPERTY RIGHTS. YOUR USE OF THE SOFTWARE AND DOCUMENTATION IS LIMITED.

Signature:____________________

Printed Name:_____________________

ATTACHMENT B (ADDITIONAL TERMS/RESPONSIBILITIES): NONE
COMMERCIAL USE SUPPLEMENT TO SUN COMMUNITY SOURCE CODE LICENSE
GENERAL TERMS

I. PURPOSE AND EFFECT.

This Commercial Use Supplement General Terms ("CUSupp") is required for Commercial Use of Covered Code and, once signed by You and Original Contributor, is operative for all Technologies specified in all Technology Specific Attachment(s) hereto. The rights and responsibilities set forth in this CUSupp are additional to those in Your License.

II. DEFINITIONS.

"Commercial Use" means uses and distributions of Covered Code for any direct or indirect commercial or strategic gain or advantage.

"Compliant Implementation" means Covered Code that fully implements the Technology Specifications and complies with Sun's requirements.

"Compliance Materials" means the test programs, guides, documentation and other materials identified in the Technology-Specific Attachment(s) for use in establishing that Covered Code is a Compliant Implementation.

"Technology-Specific Attachment(s)" means an attachment or attachments to this CUSupp which contains terms and conditions specific to the Technology therein identified as well as the specifics of the Compliance Materials and requirements for such Technology.

III. COMMERCIAL USE RIGHTS.

A. Commercial Use. Subject to and conditioned upon Your compliance with the terms and conditions of Your Research Use license and the additional terms and conditions set forth in this CUSupp and associated Technology-Specific Attachment(s), including the provisions of Section IV, below, Original Contributor hereby adds to those rights enumerated under Section III.A.1 of the Research Use license the rights to, within the specified Field of Use denoted in the Technology Specific Attachment:

   e) use the Compliance Materials to determine whether Covered Code constitutes a Compliant Implementation;

   f) use, reproduce, display, perform and distribute internally source and object code copies of Compliant Implementations for Commercial Use;

   g) reproduce and distribute to third parties and Community Members through multiple tiers of distribution object code copies of Compliant Implementations for Commercial Use;

   h) reproduce and distribute the source code of Compliant Implementations to Community Members licensed for Commercial Use of the same Technology; and

   i) reproduce and distribute a copy of the Technology Specifications (which may be reformatted, but must remain substantively unchanged) with Compliant Implementations for Commercial Use.
B. Covenant. In addition, Original Contributor’s covenant under Section III.A.2 is hereby expanded to include Your using, making, having made, selling, importing and distributing Compliant Implementations for Commercial Use insofar as permitted above.

IV. ADDITIONAL REQUIREMENTS AND COMMUNITY RESPONSIBILITIES.

As a condition to the Commercial Use rights granted above, You must comply with the following restrictions and community responsibilities (in addition to those in the License):

F. Certification. You may distribute source code of Compliant Implementations for Commercial Use only to Community Members from whom You have first obtained a certification of status in the form set forth in Attachment A-1. You must keep a copy of each such certificate and acknowledgment You obtain and provide a copy to Original Contributor, if requested.

G. Compliance Materials. Depending on the Technology licensed, Your access to and use of the Compliance Materials may be subject to additional requirements such as entering into a support agreement and trademark license. Such additional requirements, if any, are as set out in the Technology-Specific Attachment. You agree to comply fully with all such applicable requirements.

H. Compatibility. Only Compliant Implementations may be used and distributed for Commercial Use.

I. Commercial Distribution Requirement. You may distribute object code copies for Commercial Use as herein contemplated under a license agreement of Your choice which is consistent with Your rights and obligations under the License and this CUSupp. You may provide warranties, indemnities and/or other additional terms and conditions in Your license agreements, provided that it is clear that such additional terms and conditions are offered by You only. You hereby agree to hold each Community Member harmless and indemnify them against any liability arising in connection with such terms and conditions. You will pay all damages, costs and fees awarded by a court or arbitrator having jurisdiction over the matter or any settlement amount negotiated by You and attributable to such claim.

J. End User License Terms. You must include the following terms and conditions in end user license agreements accompanying copies of Compliant Implementations distributed for Commercial Use hereunder:

1. Software contains copyrighted information of Sun Microsystems, Inc. and title is retained by Sun.

2. Use, duplication or disclosure by the United States government is subject to the restrictions set forth in the Rights in Technical Data and Computer Software clauses in DFARS 252.227-701(c)(1)(ii) and FAR 52.227-19(c)(2) as applicable.

K. Defense. Original Contributor will have the right, but not the obligation, to defend You, at Original Contributor’s expense, in connection with a claim that Your Commercial Use of Reference Code is an infringement of a third party’s intellectual property rights, in which case You agree to cooperate with Original Contributor and Original Contributor will pay all damages costs and fees awarded by a court or tribunal of competent jurisdiction, or such settlement amount negotiated by Original Contributor and attributable to such claim.
L. Notice of Breach or Infringement. Each party shall notify the other immediately in writing when it becomes aware of any breach or violation of the terms of this License, or when You become aware of any potential or actual infringement by a third party of the Technology or Sun’s intellectual property rights therein.

M. Proprietary Rights Notices. You must not remove any copyright notices, trademark notices or other proprietary legends of Original Contributor or its suppliers contained on or in the Covered Code, Technology Specifications and Contributed Code Specifications.

N. Relationship. The relationship created is that of licensor and licensee only. You hereby waive the benefit of any law or regulation dealing with the establishment and regulation of franchises or agencies.

Agreed:

You

By: _____________________________

Name: __________________________

Title: ___________________________

Date: ___________________________

Original Contributor

Sun Microsystems, Inc.

Name: __________________________

Title: ___________________________

Date: ___________________________
TECHNOLOGY SPECIFIC ATTACHMENT TO THE SUN COMMUNITY SOURCE LICENSE
JINI TECHNOLOGY CORE PLATFORM
Version: 3.0/Jini TSA 1.0

1. Effect. This Technology Specific Attachment to the Commercial Use Supplement applies to the Jini Technology Core Platform as described on the Technology Site. The rights and responsibilities set forth in this Technology Specific Attachment are additional to those in Your License and the CUSupp. The term “License” hereinafter refers to the License, the CUSupp, and this Technology Specific Attachment.

2. Additional Requirements and Responsibilities. In addition to the requirements and responsibilities specified in the License, and as a condition to exercising the rights granted therein, You agree to the following additional requirements and responsibilities:

   a) Distribution of Source Code. Source Code of a Compliant Implementation authorized for distribution for Commercial Use may be distributed only to another Commercial Use Licensee of the same Technology. You must include a prominent notice with every copy of Source Code of Covered Code that You distribute indicating that use is limited to Licensees in good standing and is subject to the terms and conditions of this License. You may not offer or impose any terms on any Source Code of Covered Code that alters the recipient’s rights, requirements, and responsibilities under the recipient’s License.

   b) Upgraded Code. From time to time, Original Contributor may post Upgraded Code to the community web-server described at the Technology Site. Upgraded Code as used in this Section 2 means new versions of the Technology designated by Original Contributor as an upgrade to the Technology at the Technology Site. Wherever commercially and technically reasonable, You agree that each release by You of a product comprising or incorporating a Compliant Implementation will implement the most current Upgraded Code available no less than one hundred and twenty (120) days prior to Your Commercial Use of such Compliant Implementation. If you determine that it is not commercially or technically reasonable to incorporate Upgraded Code as contemplated, You are then not required to incorporate such Upgraded Code provided that, from the date one hundred twenty (120) days after such Upgraded Code is first made commercially available, You may not thereafter, for Commercial Use, distribute any new Modifications or Interfaces, alone or integrated with Covered Code or other code, unless and until such time as You incorporate the then-current Upgraded Code and pass the associated TCK. You may, however make Error Corrections and You may correct the adverse effect of a failure of Your Modifications and Interfaces to perform their functions. It is Original Contributor’s intent to maintain compatibility between Covered Code and Upgraded Code.

   c) Additional Services. If you provide any services, in the form of interfaces or otherwise, whose functions are substantially similar to those core services whose Interfaces are provided under this License (the “Standard Service Interfaces”), then You must also support the Standard Service Interfaces in Your product or technology. In addition, You must support Standard Service Interfaces associated with Upgraded Code in connection with Your Commercial Use of Compliant Implementations in the same manner as required for Upgraded Code under Section 2.b.
d) Branding. Compliant Implementations used for Commercial Use may, at Your option, be branded with the Technology compliance logo under a separate trademark license to be executed by You and Original Contributor concurrent with execution of this Technology Specific Attachment.


a) Support to You. Technical support is not provided to You by Original Contributor under this License. You may obtain one or more support programs, if available, from Original Contributor relating to the Technology which are described on the Technology Site.

b) Customer Support. You are responsible for providing technical and maintenance support services to Your customers for Your products and services incorporating the Compliant Implementation.

4. Royalties and Payments.

Royalty per Unit $: None.

5. Compliance Materials; Use Restrictions.

Compliance Materials: Jini Technology Core Platform Compatibility Kit

a) The Compliance Materials for the Technology may be accessed at the Technology Site.

b) You are not authorized to create derivative works of the Compliance Materials or use the Compliance Materials to test any implementation of the Technology Specifications that are not Covered Code. You shall only use the Compliance Materials for purposes of verifying compatibility with the Technology Specifications. You must not publish your test results or make claims of comparative compatibility with respect to other implementations of the Technology Specifications. In consideration for the license grant in Section III.A of the CUSupp, You agree not to develop Your own tests which are intended to validate conformance with the Technology Specifications.

c) Notwithstanding subsection 6.b above, You may use and modify the Source Code of programming code contained in the Compliance Materials for the sole purpose of creating error corrections or modifications to the Compliance Materials to propose to Original Contributor for inclusion in the Compliance Materials. You may not use such modified code for any other purpose including, without limitation, in testing Covered Code pursuant to Section III.A. Any changes to the Compliance Materials which you propose to Original Contributor shall be subject to the license grant set forth in Section III.B.

6. Requirements for Determining Compliance.

a) Development Restrictions. A Compliant Implementation:

i. must fully comply with the Technology Specifications for the Technology to which this Technology Specific Attachment applies; and

ii. must not modify or extend the required public class or public interface declarations whose names begin with “java”, “javax”, “jini”, “net.jini”, “sun.hotjava”, “COM.sun” or their equivalents in any subsequent class, interface and/or package naming convention.
adopted by Original Contributor. It is specifically suggested that You name any new Java packages using the “Unique Package Naming Convention” as described in “The Java Language Specification” by James Gosling, Bill Joy, and Guy Steele, ISBN 0-201-63451-1, August 1996. Section 7.7 “Unique Package Names,” on pages 125 and 126 of this specification says in part:

You form a unique package name by first having (or belonging to an organization that has) an Internet domain name, such as “sun.com”. You then reverse the name, component by component, to obtain, in this example, “com.sun” and use this as a prefix for your package names, using a convention developed within your organization to further administer package names.

b) Covered Code. All Covered Code must constitute a Compliant Implementation prior to any Commercial Use (other than pre-deployment testing), whether originating with You or acquired through a third party. Successful compatibility testing must be completed by You, or by a third party authorized by Original Contributor to conduct such tests, using the most current version of the applicable Compliance Materials available no less than one hundred and twenty (120) days prior to Your Commercial Use. If You make any further Modifications to any Covered Code previously determined to be a Compliant Implementation, You must retest the new Covered Code to ensure that it continues to be a Compliant Implementation. For this retest, You may use the same version of the Compliance Materials as used originally or, at your option, You may use a more current version of the Compliance Materials.

c) Test Results. Upon Original Contributor’s written request, You agree to provide to Original Contributor or the third party test facility, if applicable, Your test results that demonstrate that Covered Code is a Compliant Implementation, and that Original Contributor may publish or otherwise distribute such test results.

Agreed:

You

By: _____________________________
Name: __________________________
Title: ___________________________
Date: ___________________________

Original Contributor

Sun Microsystems, Inc.

Name: __________________________
Title: ___________________________
Date: ___________________________
APPENDIX C: OPEN SOURCE CONTRIBUTOR FORMS

Below is the full text of the contributor agreements for several open source projects. Included with each is a pointer to where it may be found.

FREE SOFTWARE FOUNDATION COPYRIGHT ASSIGNMENT FORM
from: http://ftp.xemacs.org/beta/FSF/assign.changes

(The following is sent in an email message to a potential contributor from a developer working on an FSF project. For more details see: http://www.gnu.org/prep/maintain_6.html).

The way to assign copyright to the Foundation is to sign an assignment contract. This is what legally makes the FSF the copyright holder so that we can register the copyright on the new version. I'm assuming that you wrote these changes yourself; if other people wrote parts, we may need papers from them.

If you are employed to do programming (even at a university), or have made an agreement with your employer or school saying it owns programs you write, then you and we need a signed piece of paper from your employer disclaiming rights to the program.

The disclaimer should be signed by a vice president or general manager of the company. If you can't get at them, anyone else authorized to license software produced there will do. Here is a sample wording:

Digital Stimulation Corporation hereby disclaims all copyright interest in the changes and enhancements made by Hugh Heffner to the program “seduce,” also including any future revisions of these changes and enhancements.

Digital Stimulation Corporation affirms that it has no other intellectual property interest that would undermine this release, or the use of the Program, and will do nothing to undermine it in the future.
<signature of Ty Coon>, 1 April 1987
Ty Coon, President of Vice, Digital Stimulation Corp.

(If your employer says they do have an intellectual property claim that could conflict with the use of
the program, then please put me in touch with a suitable representative of the company, so that we
can negotiate what to do about it.)

IMPORTANT: When you talk to your employer, *no matter what instructions they have given
you*, don't fail to show them the sample disclaimer above, or a disclaimer with the details filled in
for your specific case. Companies are usually willing to sign a disclaimer without any fuss. If you
make your request less specific, you may open Pandora's box and cause a long and unnecessary delay.

Below is the assignment contract that we usually use. You would need to print it out, sign it, and
snail it to:

Richard Stallman
545 Tech Sq rm 425
Cambridge, MA 02139
USA

Please try to print the whole first page below on a single piece of paper. If it doesn't fit on one printed
page, put it on two sides of a single piece of paper.

Don't forget to put down the date when you sign! Spell out the month name—don't use a number
for the month. Dates using a number for the month are ambiguous; 2/8/95 means one thing in the
US and another in Europe.

Snail a copy of the employer’s disclaimer as well.

Please send me email about what you decide to do. If you have any questions, or would like some-
thing to be changed, ask rms@gnu.org via email.

ASSIGNMENT

For good and valuable consideration, receipt of which I acknowledge, I, NAME OF PERSON,
hereby transfer to the Free Software Foundation, Inc. (the “Foundation”) my entire right, title, and
interest (including all rights under copyright) in my changes and enhancements to the program
NAME OF PROGRAM, subject to the conditions below. These changes and enhancements are
herein called the “Work.” The work hereby assigned shall also include any future revisions of these
changes and enhancements hereafter made by me.

Upon thirty days’ prior written notice, the Foundation agrees to grant me non-exclusive rights to
use the Work (i.e. my changes and enhancements, not the program which I enhanced) as I see fit;
(and the Foundation’s rights shall otherwise continue unchanged).

For the purposes of this contract, a work "based on the Work" means any work that in whole or
in part incorporates or is derived from all or part of the Work.
The Foundation promises that all distribution of the Work, or of any work “based on the Work,”
that takes place under the control of the Foundation or its assignees, shall be on terms that explicitly
and perpetually permit anyone possessing a copy of the work to which the terms apply, and possess-
ing accurate notice of these terms, to redistribute copies of the work to anyone on the same terms.
These terms shall not restrict which members of the public copies may be distributed to. These
terms shall not require a member of the public to pay any royalty to the Foundation or to anyone
else for any permitted use of the work they apply to, or to communicate with the Foundation or its
agents in any way either when redistribution is performed or on any other occasion.

The Foundation promises that any program “based on the Work” offered to the public by the
Foundation or its assignees shall be offered in the form of machine-readable source code, in addition
to any other forms of the Foundation’s choosing. However, the Foundation is free to choose at its
convenience the media of distribution for machine-readable source code.

The Foundation promises to give or send me, upon reasonable prior notice and payment of a fee
no more than twenty times the cost of the necessary materials and postage, a copy of any or all of the
works “based on the Work” that it offers to the public or that it has offered within the past six
months, or that it distributed for the first time within the past six months. For works that are pro-
grams, the machine-readable source code shall be included. My request shall detail whether I wish to
receive all such works or specific works. My choice of works to request may affect the cost and there-
fore the fee.

I hereby agree that if I have or acquire hereafter any patent or interface copyright or other intel-
lectual property interest dominating the program enhanced by the Work (or use of that program),
such dominating interest will not be used to undermine the effect of this assignment, i.e. the Foun-
dation and the general public will be licensed to use, in that program and its derivative works, with-
out royalty or limitation, the subject matter of the dominating interest. This license provision will be
binding on my heirs, assignees, or other successors to the dominating interest, as well as on me.

I hereby represent and warrant that I am the sole copyright holder for the Work and that I have
the right and power to enter into this contract. I hereby indemnify and hold harmless the Founda-
tion, its officers, employees, and agents against any and all claims, actions or damages (including
attorney’s reasonable fees) asserted by or paid to any party on account of a breach or alleged breach
of the foregoing warranty. I make no other express or implied warranty (including without limita-
tion, in this disclaimer of warranty, any warranty of MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE).

Agreed: [signature] Date: [Write the month with LETTERS]

For the Free Software Foundation,

Richard Stallman, President:

Please print the following as a separate page.
Please email a copy of the information on this page to fsf-records@gnu.ai.mit.edu, if you can, so that our clerk doesn't have to type it in. Use your full name as the subject line.

[For the copyright registration, what country are you a citizen of? What year were you born? Please write the information here; sending it separately (e.g. in a message) makes extra clerical work for us.]

[Please write your email address here.]

[Please write your snail address here, so we can snail a copy back to you.]

[Which files have you changed so far, and which new files have you written so far?]
APACHE CONTRIBUTOR AGREEMENT

taken from: http://www.apache.org/foundation/records/minutes/1999/board_minutes_1999_11_01.txt

The Apache Software Foundation
Contributor License Agreement

Thank you for your interest in The Apache Software Foundation (the “Foundation”). In order to clarify the intellectual property license granted with contributions of software from any person or entity (the “Contributor”), the Foundation would like to have a Contributor License Agreement on file that has been signed by the Contributor, indicating agreement to the license terms below. This license is for your protection as a Contributor of software to the Foundation and does not change your right to use your own contributions for any other purpose.

If you have not already done so, please complete this Agreement and send it by facsimile to the Foundation at +1-410-803-2258, or send a photocopy by regular mail to The Apache Software Foundation, 1901 Munsey Drive, Forest Hill, MD 21050-2747, U.S.A. Please read this document carefully before signing and keep the original for your records.

Full name: ___________________________

Mailing Address: ___________________________ Telephone: ___________________________

_________________________________

_________________________________ Facsimile: ___________________________

_________________________________

Country: ___________________________ E-Mail: ___________________________

I hereby accept the following terms and conditions:

1. Your “Contributions” means all of your past, present and future contributions of object code, source code and documentation to the Foundation, however submitted to the Foundation.

2. You hereby grant to the Foundation a non-exclusive, irrevocable, worldwide, no-charge, transferable copyright license to use, execute, prepare derivative works of, and distribute (internally and externally, in object code and, if included in your Contributions, source code form) your Contributions.

3. You represent that you are legally entitled to grant the above license. If your employer(s) have rights to intellectual property that you create, you represent that you have received permission to make the Contributions on behalf of that employer, or that your employer has waived such rights for your Contributions to the Foundation.
4. You represent that, except as disclosed in your Contribution submission(s), each of your Contributions is your original creation. You represent that your Contribution submission(s) include complete details of any license or other restriction (including, but not limited to, related patents and trademarks) associated with any part of your Contribution(s) (including a copy of any applicable license agreement). You agree to notify the Foundation of any facts or circumstances of which you become aware and which would make Your representations in this Agreement inaccurate in any respect.

5. You are not expected to provide support for your Contributions, except to the extent you desire to provide support. You may provide support for free, for a fee, or not at all. Your Contributions are provided AS-IS, without warranty of any kind (either express or implied) including, without limitation, any implied warranty of merchantability and fitness for a particular purpose and any warranty of non-infringement.

Please sign: ________________________________ Date: _____________________
OPENOFFICE.ORG CONTRIBUTOR ASSIGNMENT

Whereas, Contributor owns and has sufficient rights to contribute the software source code and other related intellectual property.

Contributor agrees to the following:

1. Contributor hereby assigns to Sun all worldwide common law and statutory rights associated with the copyrights, copyright application, copyright registration and moral rights in the Contribution.

2. Contributor represents that he/she/it is legally entitled to grant the above assignment and that he/she/it is not providing any code that violates any law or breaches any contract.

3. Contributor agrees to take such reasonable actions as may be required, if any, to perfect the assignment of the Contribution as stated herein, including the execution and delivery of any additional instruments of assignment if appropriate and necessary.

4. Contributor agrees that this copyright assignment is in effect for all contributions of software source code and other related intellectual property.

Signed: ________________________________ Date: ________________

Printed Name: ________________________________

Please send a signed copy of this assignment by facsimile to Sun Microsystems at: +1-650-562-2629 or mail it to:

Sun Microsystems for OpenOffice.org
c/o Max Lanfranconi
901 San Antonio Road
MailStop CUP03-410
Palo Alto CA 94303-4900
USA
PROJECT JXTA CONTRIBUTOR ASSIGNMENT

taken from: http://www.jxta.org/project/www/sun_contrib_4-25-01.pdf

Sun Project JXTA Software Contributor License Agreement

Thank you for your interest in Project JXTA (the “Project”). In order to clarify the intellectual property license granted with contributions of software from any person or entity to the Project (a “Contributor”), Sun requires that a Contributor License Agreement be on file that has been signed by each Contributor, indicating agreement to the license terms below. This license is for your protection as a Contributor of software to the Project and does not change your right to use your own contributions for any other purpose.

If you have not already done so, please complete this Agreement and send it by facsimile to the Project Office or send a photocopy by regular mail to:

By Fax: JXTA Project Community Manager
By Mail: The JXTA Project Community Manager
Sun Microsystems, Inc.
(415) 863-4691
Mail Stop USFO-05
901 San Antonio Rd.
Palo Alto, CA 94303
(415) 626-4616

Please read this document carefully before signing and keep the original for your records.

Your Full name: _____________________ e-mail: _____________________
Mailing Address: ____________________ Telephone: ____________________
________________________________ Facsimile: ____________________
________________________________ Country: _____________________

Contributor and Sun Microsystems, Inc. (“Sun”) hereby accept and agree to the following terms and conditions:

1. “Contributions” means all past, present and future contributions of object code, source code and documentation to the Project, however submitted by Contributor to the Project, excluding any submissions that are conspicuously marked or otherwise designated in writing by Contributor as “Not a Contribution.”

2. Contributor hereby grants to Sun’s Project JXTA, a world-wide, royalty-free, non-exclusive license:

(a) under Contributor’s intellectual property rights (other than patent or trademark) Licensible by Contributor, for Sun and any third party developer of Project JXTA to use, reproduce, modify, dis-
play, perform, sublicense and distribute the Contributions created by such Contributor (or portions thereof) with or without modification; and

(b) under Contributor’s patent claims infringed by the making, using, or selling of Contributions made by that Contributor either alone and/or in combination with other Contributions (or portions of such combination), to make, use, sell, offer for sale, have made, and/or otherwise dispose of: 1) Contributions made by that Contributor (or portions thereof); and 2) the combination of Contributions made by that Contributor with other software and hardware.

(c) the licenses granted in Sections 2(a) and 2(b) above are effective on the date Contributor delivers the Contributions to Sun’s Project JXTA.

(d) notwithstanding Section 2 (b) above, no patent license is granted: 1) for any code that a licensee has deleted from the Contribution; 2) separate from the Contribution; or 3) for infringements caused by: (i) third party modifications of a Contribution or (ii) the combination of Contributions with other software or other devices if such combination causes the infringement.

Except for the rights granted to Sun’s Project JXTA in this paragraph, Contributor reserves all right, title and interest in and to your Contributions.

3. Contributor represents that Contributor is legally entitled to grant the above licenses. If Contributor’s employer(s) have rights to intellectual property that Contributor creates, Contributor represents that Contributor has received permission to make the Contributions on behalf of that employer, or that Contributor’s employer has waived such rights for Contributor’s Contributions to the Project.

4. Contributor represents that, except as disclosed in Contributor’s Contribution submission(s), each of Contributor’s Contributions is Contributor’s original creation. Contributor represents that Contributor’s Contribution submission(s) include complete details of any license or other restriction (including, but not limited to, related patents and trademarks) associated with any part of Contributor’s Contribution(s) (including a copy of any applicable license agreement). Contributor agrees to notify Sun of any facts or circumstances of which Contributor becomes aware that would make Contributor’s representations in this Agreement inaccurate in any respect.

5. Contributor is not expected to provide support for Contributor’s Contributions, except to the extent Contributor desires to provide support. Contributor may provide support for free, for a fee, or not at all. Contributor’s Contributions are provided as-is, with all faults defects and errors, and without warranty of any kind (either express or implied) including, without limitation, any implied warranty of merchantability and fitness for a particular purpose and any warranty of non-infringement.

Please sign: ___________________________ Date: ______________
APPENDIX D: ARTICLE FROM XML.COM

The following article was originally published on XML.com at http://www.xml.com/pub/a/2000/07/19/deviant/index.html. Reprinted with permission from O'Reilly & Associates, Inc.

Codename Spinnaker
By Leigh Dodds

The Apache XML developer lists have seen some turmoil recently over a proposed plan to develop their next generation XML parser. However a phoenix may have been borne out of the ashes: this week we report on “Spinnaker,” a.k.a. the Xerces Refactoring Initiative.

Sparring over Spinnaker

In a bold announcement on the general Apache developer list, James Davidson outlined his intention to start a project, code-named “Spinnaker,” with some ambitious goals:

Spinnaker is an attempt to create a next generation Apache XML Parser based on all the lessons learned from the current versions of Xerces and Crimson.

The announcement listed goals for the Spinnaker effort itself. Amongst these was the desire to produce a parser suitable for inclusion in the Java Development Kit, a goal that apparently had internal backing from Sun.

To say that the announcement was welcomed with less than open arms by some of the IBM developers would be an understatement:

Looks like a “coup d’etat” to me.

This was the reaction of Arnaud Le Hors. Andy Clark, another IBM developer, questioned the wisdom of beginning such a project at the start of the weekend:

Is it possible that, in the future, we hear about submissions to the tree *before* everyone goes home on Friday? I want us all to work together on the future of the Xerces parser instead of being surprised by a new source tree over a weekend.
These opening remarks set the tone for a heated debate, which threatened to turn into an IBM versus Sun squabble over who was playing fair by open source rules. Abrasive remarks were exchanged, with both sides quickly disavowing any corporate-lead motives, whilst questioning those of the other side. James Davidson believed the timing of his announcement was irrelevant:

This is open source, Apache style. People work whenever they work and that's the way this all works. Most Apache developers don't work on the main sources during the typical M-F 8-4 (local time) window. They work when they get time, or the muse is with them, or whatever. There are no limits, it's a 24/7 shop and to be blunt, conformance with a corporate schedule isn't part of the mandate.

Arnaud Le Hors singled out the method of initiating the project for particular criticism:

The problem is not about Sun vs IBM. It is not about corporate vs individuals. Not about creating a new project. Not about working on week-ends. Not about questioning the current goals or implementation. The problem is about making decision on your own. Not communicating. Not consulting others in the project.

Ironically, it seems that a lack of communication, along with a good deal of miscommunication, was the root cause of the argument. This is symptomatic of an internal divide within the Apache parser development community. Stefano Mazzocchi, in valiant attempts to defuse the situation, highlighted the cause of the divide:

...Tomcat, Xerces and Xalan didn't start as Apache projects and their development community was _imposed_ and did not emerge from the community of volunteers.

Both Xerces and Crimson are “donations” to the Apache effort. However, these donations haven’t been made into the arms of a waiting team of Apache volunteers. Both projects are still largely driven by development teams at IBM and Sun. Mazzocchi’s key point is that simply providing a public CVS server does not make for a good open source project. If the development teams continue as they have done prior to opening the source (e.g., by having off-line design meetings), then there’s little chance for a developer community to form around the project.

Mazzocchi suggested that the Tomcat developers have managed to make the transition from internally driven development to a full open source community, whereas the Xerces, Xalan, and Crimson teams have yet to achieve it. Rajiv Mordani further illustrated this point:

...since most of you’ll work in the same office there is a lot of things that happen in there that should be actually done on the mailing lists. I don’t remember seeing a mail going out proposing the implementation of schemas for [example]. The only info that was sent out is that the repository will be a little unstable for the next few days as schema support is being added to Xerces so please use the tagged version of the workspace.

It would be wrong to single out just the IBM developers in this regard. It appears that the Sun team is as much at fault—although, as Brett McLaughin commented, it’s the external perspective that matters:
What I meant to focus on, and what we may have miscommunicated on, is the _perception_ of what is happening… The _perception_ …is what we have to fight in an open source project.

As tempers cooled, it quickly became apparent that the perception that neither team is interested in the other’s software was incorrect. Once the rhetoric was laid aside, a lot of common ground was discovered. At that point the real work started.

To help push matters forward, Sam Ruby suggested that the project should be known as the Xerces Refactoring Initiative (XRI)—a name he described as “intentionally boring.” Stefano Mazzocchi produced a “virtual press release” to explain the common purpose of the initiative:

> The Apache XML community started a “Refactoring Initiative” to create [a] next generation XML parser for Java. This will be known as “codename Spinnaker” (or simply spinnaker) and will be the collective design whiteboard where the worldwide Apache community of individuals will openly develop such [a] new XML parser. The final result of this RI will be determined when “final” status is reached and, as always, decided by the community.

**Taking the Initiative**

Turf wars aside, what are the driving forces for XRI, and what are its goals?

In his opening announcement, James Davidson listed several perceived problems with Xerces. These included performance problems under HotSpot, and code complexity. Both of these appear to be the result of efforts put in by the Xerces team to optimize the parser for the Java 1.1 platform. While the efforts have clearly paid off, the optimizations do make the code harder to understand, and limit the ability of HotSpot to automatically optimize performance on Java 1.2 and 1.3 virtual machines.

Edwin Goei observed that code complexity is one reason why additional contributions haven’t been made by the XML community:

> The main objection I have to the current Xerces code is not about which VM it runs on, but on the ability for a developer to understand the code and make changes to it. I think one major reason there has not been more developer participation is because the code is difficult to understand … I looked at two other parsers that I believe are easier to understand: Aelfred2 (not Apache) and Crimson/ProjectX… With Xerces it takes much more effort to understand the code and I prefer not to make changes to code I don’t understand.

This observation was echoed by Brett McLaughlin who suggested that developers are turning their efforts elsewhere:

> I have lots of users on the JDOM lists who think Xerces is great, but are totally scared by the code—unfortunately, I can’t blame them. They would rather spend their time working on JDOM, or something similar.

The complexity of Xerces was one issue that wasn’t disputed. Arnaud Le Hors explained that the IBM Xerces team had been considering a redesign all along:
The fact is that we’re also interested in a new version of Xerces which is more modular. As a matter of fact, we’ve given it quite some thoughts already, and have even written down a first draft of a design document on it that we’ll be happy to send out as input.

We don’t think the current model is perfect. We refer to it as “the star model.” Basically every piece knows about the others and the flow of information is far from being linear. Instead, we’re thinking of a pipeline model. Where every piece works as a box with an input and output stream working independently of the others. It will be hard to make this as efficient as the current parser. But this, among other things, would let one to easily plugin the validator or not.

Once the common ground had been found, the requirements began to emerge thick and fast. Scott Boag described his vision of what the initiative would involve:

I think the point is to build/refactor a next generation, commercially viable, parser based on the current state of the art (including and especially Schema support), and the collected requirements. It is indeed pointless, in my opinion, to talk about “adapting” other code bases—what is going to occur is a mining operation from Xerces and Crimson. Anything that’s open source with the right license is open to be mined for ideas…

Arved Sandstrom agreed that the search for ideas should be expanded to all parsers with available source and suitable licensing:

There’s stuff that microparsers like nanoxml can contribute to the discussion. Python XML parsers (and there is more than one) are quite good. If we are talking James Clark, let’s not forget expat; this is a very good parser and represents the core of the Perl XML processing family.

…I think there is potential here for making this project best-of-breed when it comes to showing that open-source can do process.

The project has now taken on a life of its own: differences seem to have been set aside (at least temporarily) to focus instead on the technology. Ed Staub has taken on the task of co-ordinating the collection and publication of the XRI requirements. The discussion has already thrown out some interesting ideas such as Grammar caching (avoiding reparsing of the same DTD multiple times), and compiling an XML Schema into a custom parser.

There’s obviously a long road ahead for XRI/Spinnaker but it should yield some interesting results. Don’t go looking for a Xerces 2 just yet, and don’t despair, as Xerces 1 is not about to be abandoned: the IBM team is currently hard at work completing XML Schema support.

Perhaps the key benefit of this project, despite its rocky beginnings, is that the internal fragility of the Apache XML may be removed. With effort from both the Xerces and Crimson teams, as well as the wider developer community, we’ll not only benefit from a next-generation parser, but also a more stable and collaborative process underpinning the development of a vital component in the XML infrastructure.