

**Start Your Own Open Source Project using
SourceForge
By Liviu Tudor**

Preface

This book is the synthesis of the work that I have carried out myself in starting up an open source project and driving this through sourceforge.net – from concept to implementation to making it available on sourceforge.net and then recruiting a team of enthusiasts and managing the development and the release process from there on.

It is meant to help others who are interested in developing a project and making it available to the community using sourceforge.net, however, they might be scared at the prospect of what a complex task this can become.

The original project this book is about is still alive on SourceForge; the project is called “*aws-s3-version-mgmt*” and you can check it out for yourself on SourceForge by visiting this link: <https://sourceforge.net/projects/awsversionmgmt/>. (And if you are that way inclined, by all means join our community of developers and contribute to it too!) The book will make often references to various tools and features offered by SourceForge and will provide links to these sections on the above-mentioned project site. So be warned ahead that I am un-ashamingly promoting this open-source project by doing so, but ultimately it’s for a good cause as well as I feel it provides a better context for the readers of this book.

Why SourceForge? There are many other open-source incubator sites out there – *github.com* being one of the most prominent ones at the moment – however, having been a SourceForge user for a long time (I did find myself many times downloading and using components hosted there!), and bearing in mind my rather poor knowledge of *git*, I opted for SourceForge when it came to starting this project. As many entrepreneurs around the world will tell you, when it comes to starting a project, you need to iterate quick and often, and as such, you need to start by using what you know – as the project evolves you can start looking at leveraging other tools and frameworks, but to start with choose the one that’s going to get you quicker to the goalpost. Hence SourceForge, in my case.

However, a lot of the issues discussed in this book are valid for any other open source project / hosting combination; yet when I will contribute my first open source project on *github* I will quite like release a new booklet on using that for your open source project.

As I just mentioned, this book references one of the open source projects which I am actively (still) working on – as it happens this is a Java project and uses Maven¹ as a build tool. As such, a lot of the configurations and information here is targeted at similar projects (Java/Maven setup) – you will still find hopefully a lot of the information here useful even if your project is not a Java one or not using Maven,

however, if you are, you will find examples of configuration pieces to use and instructions on how to achieve certain things for a maven-ized Java project.

Lastly, it's worth mentioning that the experience of bringing up to life an open source project is not that far from a startup – you get to wear a lot of hats, from the initial idea, prototyping, proof of concept, then managing the project throughout its lifecycle, building the team of developers who will work alongside you on it, help and mentor them, listen to their ideas and incorporate them in your roadmap, plan releases, assign tasks based on the strengths you have noticed in your developers and finally creating an audience (market) for your project in the outside world. (Potentially even providing support to your users in the outside world too.) So for all those entrepreneurs wannabe who I see strutting their stuff around Mountain View, read on, you might find this helpful!

Contents

Preface	2
Contents	4
Project Lifecycle.....	5
Idea.....	6
Research.....	9
Prototyping.....	11
Publish on SourceForge	13
Creating a Project on SourceForge	14
Project Metadata.....	18
Adding Tools to the SourceForge Project.....	25
Mailing Lists.....	28
Uploading the Project Source Code.....	30
Setup the Build Process	32
Maven-specific Setup.....	34
Deployment	36
References and Links.....	42

Project Lifecycle

As with any other software development projects, the open source projects are subject to the same lifecycle:

1. Idea
2. Research
3. Prototype
4. Functionality Planning
5. Alpha/Beta Versions
6. Stable Version / Release
7. Iterate

If the project is started from day zero by a team then you probably won't deviate too much from the above high-level plan – and to be honest, in that case you probably don't need the help of this book.

If however, you find yourself to be a single developer right at step 1, then this book will hopefully build your project up into a useful one for the open source community!

I will therefore step through each one of the above and explain what I did to accomplish the task and move on to the next and give pointers on how this process can be applied to other projects.

Idea

Like with many other projects, an open source project starts with an idea. This can be based on an ingenious idea you just came up with (time travel?) or it could be something as simple as providing a different mean to accomplishing something for which there are already tools out there, but you just want to provide the public with alternatives.

Bear in mind that while a lot of times we dream of coming up with an absolutely ground breaking idea and providing something that was never thought of, these ideas come to us rarely, however, there is a lot of improvement space left around each project out there. Even more, the open source community needs choices, no one likes to be anchored into a technology or tool – we use these because they help us achieve certain tasks. We start using them to the point we get comfortable with them even though they might have some major drawbacks – and at that point we will prefer those tools and frameworks to others; as such, once we reach that point there will be nothing to convince us that it's better to use emacs over vi or vice versa! And that is perfectly fine, we all have our preferences, and if you are thinking that you will change people's habits and preferences, you will find that this is a rather steep hill to climb – though not impossible! However, I advise you do not target these users and developers in the first stages of your project – you need to look at new adopters, in the above example, people who never used a terminal-based text editor, and as such haven't made up their mind about the whole vi-versus-emacs battle. These are the guys who are open to exploring alternatives – different ways of doing the same thing. (And in the above example, if you look at nano this is pretty much how this got the traction it has nowadays.) And these early adopters later on can turn the attention of “old-timers” to your project – but we'll get to that later on.

For now just understand that there's no shame in this ecosystem to provide a new implementation for a framework or tool already “out there” – whether it is the fact that you think that curl can do with that extra parameter in the command line or whether it's the fact that a certain command-line utility cannot be easily integrated in scripts. Bear in mind that due to the high numbers of techies out there, if your idea comes from an internal frustration with a particular tool (maybe you think to accomplish a particular task it's too of a cumbersome command line syntax, maybe it doesn't easily integrate with your other tools, or maybe it is missing entirely the functionality you want or whatever the frustration it might be), there will be quite likely others out there experiencing the same! And as such, your idea is quite likely not a silly one!

So don't let the presence of other "similar" tools or framework out there put you off – you came up with an idea, and stick to it. You would like a tool that accomplishes a specific task in a specific way and you think that tool is going to make your life easier – so go for it, that's your idea! Don't think at this stage if your project will get any traction or any users – in the first stages of the project you will have to work on it alone as if you are developing this for yourself only. Build it and they will come as they say – people won't be able to turn their attention to your project until it's available for them, and until then they will live happy in the "knowledge" that they don't need your tool.

As Henry Ford² geniously said: "If I had asked people what they wanted, they would have said faster horses". Same here: people don't know they need your project until it's available to them. As such, throughout the initial steps of an open-source project you will find yourself on your own – but don't let this put you off: you are building this tool or framework for your own benefit at least! You know that at the end of this cycle, when the first version is available, **YOU** will be the first user of it, so your work is not going to be in vain at all!

So once you come up with the idea, don't give up – if you think you, yourself, would use a tool or framework like that, then go for it.

In my case – or should I say in the case of this project – the idea came from my frustration with the lack of tools and UI's that Amazon is offering around their versioning³ feature on S3. As I had explained at the time in my blog post⁴ which triggered this project to start, the problem with versioning-enabled S3 buckets is that most (if not all) file browsers for S3 will only list the most recent version of the bucket, and do not provide any support for any previous versions. As such, imagine that you have 100 files that are updated daily in the bucket, after a month (~30 days), each file will have 30 versions -- and as such, the total number of objects stored in the S3 bucket will be $30 \times 100 = 3,000$ objects. Out of those, using any of the today's browser, you will only be able to see 30 objects (the latest versions). Even more, deleting those 30 latest versions will show in any S3 browser the bucket as empty, however, when you try to delete the bucket, Amazon will rightly report the bucket as non-empty and you will not be allowed to delete the bucket -- which still has $29 \times 100 = 2,900$ objects in it, sitting there, taking space and costing you money. (Please note that at the time I'm writing this, even Amazon's own web based S3 browser shows the bucket as empty in this case!)

Even more confusingly, let's say you have a file **abc.txt**, which has 100 versions, and you delete the latest one (what actually happens behind the scenes is S3 creates a new empty version which is called a "delete marker"). At this point, the file "disappears" from your S3 browser and you are convinced it's deleted. A few months later you create a new file called **abc.txt** again, totally different from the original one. At this point this new file inherits all the previous versions of the previous **abc.txt** file even though they are not related -- which can cause confusion!

The lack of UI (or any other sort) tool for this makes a lot of room for human error and confusion – and at the time the project was started as well as at the time I’m writing this book, even Amazon’s own web-based AWS S3 bucket explorer didn’t address this problem. I have checked with a few of my techie friends, most of them weren’t using the S3 versioning or they didn’t know of any tools.

So as per my own advice above, I thought: *“OK, this is not a ground-breaking idea. Amazon offers API support for versioning-enabled buckets, so it’s not something that can be labeled ‘innovative’ to put a utility together and maybe there aren’t too many users out there using that functionality. **But it will save me lots of time and frustration having a tool like this!**”* And that was the point where the idea appeared: I needed to write a tool to assist me with management of these S3 buckets!

Based on my initial scoping of this issue, and the fact that no one seemed to use the versioning feature, I have thought at the time that I am going to be on my own with this tool – no one else was going to use it or need it, let alone help me work on it! However, I went ahead with this because I needed it!

Research

This part typically refers to answering the question: “Is this even possible? And if yes, how?”. In most cases you will find yourself digging through pages on the net around the subject, investigating API’s and so on.

In my case it was pretty straight forward – I simply used the Amazon’s SDK for Java⁵ and spent a bit of time investigating what’s possible and what not in terms of versioning in S3.

Typically this phase gets entangled quite a bit with the “*Idea*” phase – as your idea tends to develop and grow as you do your research. Also, it is possible that at first glance your idea might “shrink” a bit – as you might find that a few things won’t be that easy to do. If you’re tempted to shrink down your idea, don’t! Write down all the things you wanted from your tool/framework – and as your research goes on and you find that some of these will be quite challenging, mark them down somehow. This will be issues that you can tackle later on when you start looking at planning the work on it, but do not cut on your initial idea because of your research shows a huge leap that needs to me made ahead to achieve certain tasks!

It is a good idea at this stage to not only rely on Google searches for your research – there are a lot of technical forums out there and I would encourage you to try to reach out to other techies who might use or need similar tools and frameworks to you. Who knows, maybe they are just as frustrated as you about the lack of that “something” out there and you might even find at this early stage someone to lend you a helping hand in creating it! (I will get back to this point later on, it tends to be the case with a lot of open-source projects that you need to get close enough to that first release for developers to start turning their attention to your project, but occasionally you can be pleasantly surprised by finding others who are offering to help right away!)

Do not hesitate at all during this stage to let others know that you intend to build an open-source project around this idea – trust me, no one will “steal” your idea from you! Even if let’s say that you come up with something so good that someone can see right away a commercial opportunity in building something similar and selling it (or the services), trust me, it’s much harder to transform a piece of code into money than you think: they will have to find funding and resources (developers, computers, networking, office space and so on) on their side, and this takes time! By which point you’ve quite likely advanced with your project. And even if they do have all of that available for them right away, they have to overcome a major problem: the fact that they are offering a paid service and hitting “the market” with it going against

your free, open-source “offering”. And it won’t be that easy for them to achieve that easily. (I’m not saying it’s impossible, I’m just saying that your open source project will always have a place in the market if you find yourself in such a sweet spot, so don’t worry at all about being loud and clear and open about this project idea to anyone who asks!)

It might be the case that some techies will point you towards slightly similar projects – or might offer a helping hand as I said. Or they might discover this little script they put together ages ago when they came across something similar – and that might give you that extra 0.1% boost when you finally sit down to write your code. Be active about your project on all the channels you normally use to do your daily job – I find for instance that LinkedIn groups and stackoverflow.com are great tools for that.

The outcome of this research phase should be that you have an idea of how to go about implementing your project, what sort of functionality you will be plugging into it and what tools/frameworks you need to use as well as what tools/frameworks to keep an eye on. (If there’s something similar out there that does something very well why not offer it yourself too?)

Once this is done, you can move onto prototyping and finally starts seeing your project growing!

Prototyping

This is the part I personally enjoy most – probably same as with most coders: sitting down and writing code. You have at this point an idea of what is possible, what is easy and what is hard to do. You are finally sitting down and stringing code together trying to make the “easy” bit “accomplished” and the “hard” bit “possible”.

What goes in your prototype is up to you – it’s most likely the very first tiny bit of thought that your idea came out of. There is no one who can tell you what goes in there and what not, as such, attack this at any angle you want. Bear in mind though that this prototype will evolve soon into your alpha version – so try to have at least one bit of functionality working. You will find out that as you start working on that a few other collaterals crop up which you will have to implement as well – and this is what’s growing slowly your prototype into a fully-fledged application or framework, which then can be used by others too!

Don’t bother at this point with things like source control, nightly integration, documentation and so on – just focus on writing the code, you will get to do all that later on.

In my case, the first problem I came across was not being able to delete an S3 bucket that otherwise looked empty in all S3 browsers! So the first bit of code that I wrote for this was a “purge bucket” bit of functionality. I simply wrote a function to iterate through all the versions in a bucket and delete them from S3. Then once I was done with that tiny function, of course, I thought: *“well, I need to wrap it up in an executable program”*. So out came the `main()` function. Now since I wrote a `main()` function, I might just as well accept a few parameters, so I’m not relying on hardcoded access keys and bucket names! Since I’m going to do that I might just as well provide a structured set of command line parameters, something like `--bucket=...` or similar – a syntax that most users are familiar with. So before I knew it I plugged in the Apache Commons CLI⁶. That then required a bit of code written around building the command-line options, dealing with help screens, parsing errors and so on. Before I knew it, my simple requirement of purging an S3 bucket became a stand-alone Java application, with a nice syntax in the command line, which can be parameterized to suit anyone with an AWS account!

At this point it’s time to stop! (Yup, you read it correctly!) It’s time to stop because you have grown your idea into a prototype: a piece of code that can be used by others to accomplish a tiny thing, but while small, it’s enough to prove that actually your idea can (and has begun to!) grow into a more complex problem. Sure, you will have still tons of functionality that you’d like to implement at this stage and add to your project. But if you got your project into this stage where someone else can pick

up your code and can use it to achieve one of the tasks you had in mind for your project, then you have reached your goal of prototyping: yes, it is possible to do what you want to do and here you go, you've just proved part of it with this bit of code you've written!

At this point, it's time to hit the "market" with it – and this is where the lifecycle of the project begins to look a bit different to your standard one!

Publish on SourceForge

If you have completed your prototype and you have some code that “does stuff”, it’s time to publish this to sourceforge.net and transform it into a “proper” open source project from there on! Don’t be tempted to take the approach of publishing everything after you completely finished the project – and here’s why:

- First of all, you get free source control – whether you use SVN or GIT on sourceforge, you will get a free source code repository, full with history, rollback and so on. (How many times you decided to do a refactoring of the code which then proved to be rubbish and you wish you had a previous version to revert to and start again? Well, once you have a source repository that obviously becomes possible!)
- SourceForge tracks down all the activity on a project – from commits to source code to tickets updates, wiki and so on; based on this activity it promotes on various parts of the website projects – be it “Most Recently Updated Projects” or “Most Popular” or “Active Projects”. You want sourceforge to start seeing a lot of activity on your project sooner rather than later – and as such occasionally push your project in front of the numerous developers who find themselves using the website every day.
- Setting up the project in sourceforge provides you with a central point for “brain dumping” all of your ideas – whether it’s using the wiki, the ticketing system, forums or any other applications.
- Starting working with sourceforge in the early stages of the project also allows you to familiarize yourself with certain features of the site that maybe you haven’t used before. I have personally used a mix of JIRA, Redmine, Bugzilla and a few other issue tracking systems before, so it took me a while to get acquainted to sourceforge’s own issue tracking system. Same for their wiki.
- Last but not least, doing so provides you visibility on the web! Bear in mind that Google indexes a website once every couple of weeks maybe – quite likely much more often with something like sourceforge – so the moment your project is on sourceforge, it becomes available for Google to index. And once Google indexes it, anyone searching for matters related to your project might come across your project on sourceforge! It’s all good you finishing your project and then publishing it – but then you have to wait at least 2-3 weeks for Google to index it, then you might have to wait another couple of weeks for Google to check back make sure your website hasn’t disappeared in the meanwhile and from there on to start pushing it into its search results – this means you can have 1-2 months after your project is released and published on sourceforge (if you take the approach “publish after release”) and appearing in search results! Hitting sourceforge.net right away with your

project means that while Google and other crawlers are doing their job indexing your project you carry on coding along on your side, safe in the knowledge also that your code is stored in a code repository, it's backed up and so on!

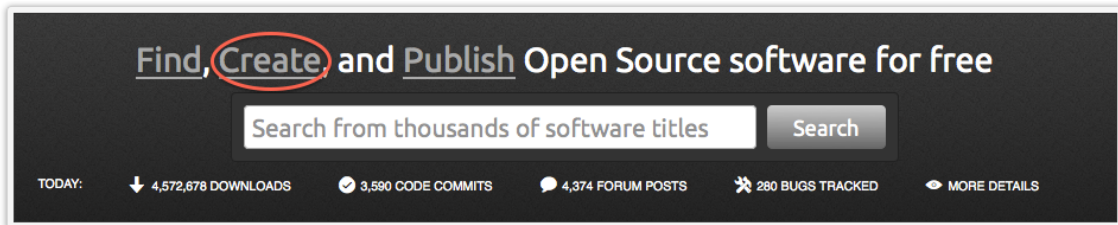
- Last but not least, publishing your project to sourceforge makes collaboration possible right away! Maybe in a week time one of your friend turns around and says *"hey, how's that S3 versioning thing you were working on coming along? We have decided to use S3 versioning in our app but I still haven't found anything out there for helping me with the cleanup – you still working on it? Do you need a hand with it?"* If that happens and all of your source code is still "safe" in your laptop, you cannot but say *"well, errr, yes, I'm still working on it. If you want to help I can give you a tarball with the sources, then you send me back just the files you modified.... I guess I'll have to figure out a way to merge them with my changes hmmm.... Or how about we spend some time putting this somewhere we can both access it?"* And you end up on ... sourceforge.net :) The trouble is, now you have to tell your friend to wait a bit while you put it on sourceforge – *but hang on, I'm in the middle of finishing this big piece of code, I'll do it over the weekend.* And trust me, I speak from experience, your friend might not be that patient and you just missed your opportunity of benefiting from a little help from him since after the weekend he won't be available due to some production outage or his holiday coming up, or his mom visiting, or his girlfriend's birthday ... and the list can go on! You want to be able when the opportunity arises for someone to help you to just say, sure here's the project on sourceforge, give me your username and I'll add you to it! And publishing on sourceforge right away eliminates all these future problems for you while providing you with a comprehensive set of tools for your project!

Creating a Project on SourceForge

Publishing your project is a pretty simple process and I'll walk you through it shortly, but first it's worth noticing that from here on you will be wearing multiple hats and you're moving out of your comfort zone of being just a developer! You become the project manager, admin, marketer, planner, recruiter, developer and tester all in one! Some developers might find this overwhelming – it might not be in their nature, it is after all the eternal "developer turning manager-wannabe" and some can't hack it! But the first steps in open source projects are very similar to the first few months in a startup – everyone is wearing multiple hats and everyone is helping with everything. That "everyone" unfortunately in this case is you – so you have to become the project manager, marketing person, CTO and CEO in one for a little while – in order to see your project come to life. Just as in the case of a startup, your motivation is seeing your project come to life and start being used in the outside world – and this should be the force that drives you to accomplish all of these tasks which are not related really to software engineering and development,

but are absolutely crucial to a project. If you don't execute this step and don't set up your project in sourceforge, it will be a struggle to get visibility and traction – makes it more difficult to keep track of releases, issues fixed and so on.

Getting back to the initial matter, publishing a project on sourceforge is very simple – create an account if you haven't got one (don't worry, accounts are free!) – and then in the main screen simply click on “Create”:



This will take you to the pretty simple create project screen:

Create a Project

Project Name

Please enter a value

URL Name

Please use only letters, numbers, and dashes 3-15 characters long.

http://sourceforge.net/p/

<input checked="" type="checkbox"/> Wiki Documentation is key to your project and the wiki tool helps make it easy for anyone to contribute.	<input checked="" type="checkbox"/> Files & Stats Use the largest free, managed, global mirror network to distribute your files, and follow the download trends that enable you to develop better software.
<input checked="" type="checkbox"/> Git Git is a distributed version control system designed to handle everything from small to very large projects with speed and efficiency.	<input checked="" type="checkbox"/> Tickets Bugs, enhancements, tasks, etc., will help you plan and manage your development.
<input checked="" type="checkbox"/> Forums Collaborate with your community in your forum.	<input type="checkbox"/> Blog Share exciting news and progress updates with your community.
<input type="checkbox"/> SVN Enterprise-class centralized version control for the masses.	<input type="checkbox"/> Mercurial Mercurial is a distributed source control management tool that efficiently handles projects of any size and offers an easy and intuitive interface.

I have read and agree to the [Terms of Use](#), and acknowledge that non-compliant projects will be removed.

Create

Most of the options above you can revisit at any point once the project is created, the main part here is selecting a name for your project. Comes as no surprise that this name is supposed to be rather self-explanatory, but what is not known, and I found out this through my own experience with sourceforge, is that when searching for projects on sourceforge, their search engine matches projects on project name first – so if your project is dealing with Amazon’s S3 for instance, make sure you include “s3” somewhere in the name. You can use dashes to separate words in the project name – for instance in my case I came up with “aws-s3-version-mgmt” – feel free to make up your own which includes a bit of SEO so to speak. Based on this, SourceForge will create your project its own **web space**, using the format `http://sourceforge.net/projects/<YOUR PROJECT NAME>`. In my case, not aware initially of the search implications of sourceforge, my first project name was rather uninspired and it was just `aws-version-mgmt` – which sourceforge

“flattened” to “awsversionmgmt” so the url for the project was created as :
<http://sourceforget.net/projects/awsversionmgmt/>

Also note that automatically a separate project **website**⁷ using the naming `http://<PROJECT NAME>.sourceforge.net` - in my case this became <http://awsversionmgmt.sourceforge.net>

During this step you can also select your project source code repository as well - I personally am very comfortable with SVN and I chose that as my first repository, however, if git or mercurial is your weapon of choice, go with that. I strongly suggest you select the “Wiki”, “Files and Stats” and “Tickets” features too at this stage:

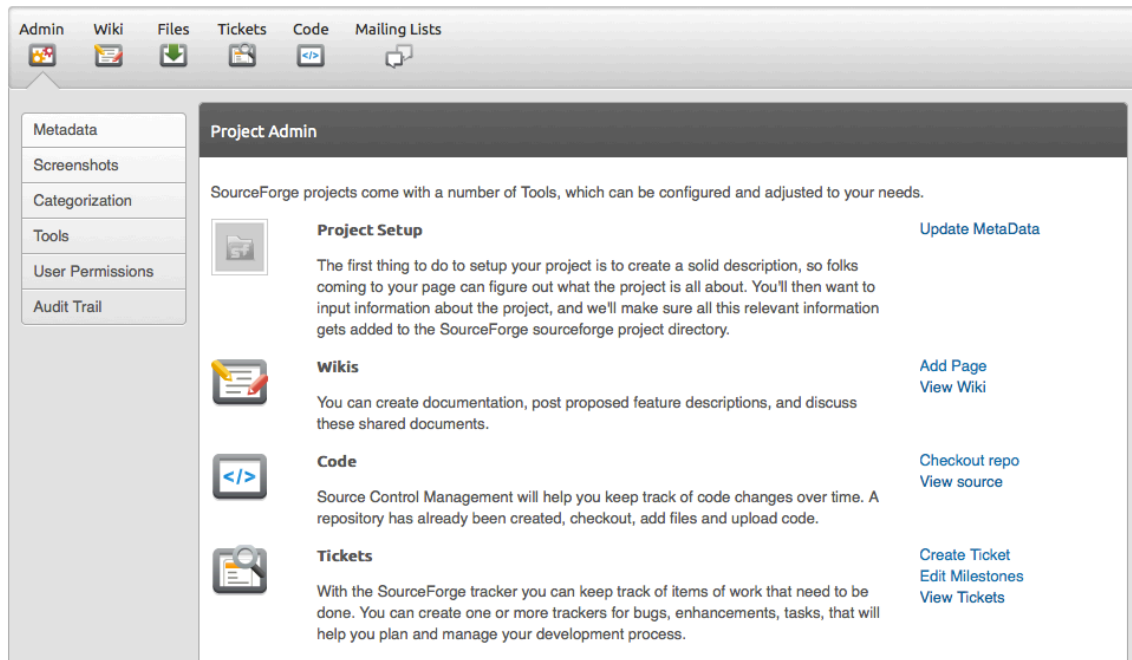
- The Wiki will provide you with a very easy yet effective interface for brain-dumping all of your ideas about the project for the community to see. It also helps you crystalize ideas into functionality, so it's a crucial component of your project development. Looking into the future, it also provides a common place for your developers to express their design / architecture / coding / features ideas, and when that moment comes that you will have others joining your project, you want everything to be set up for them already.
- The ticketing functionality allows you to plan your project more effectively - it's great to dump your ideas on the wiki in an unstructured format - but then when you start looking at which one of those are ready for “production”, it helps to have an organized way of doing so. Your ideas from the wiki will ultimately grow to become tickets in the ticketing system - these tickets will then be assigned to versions/releases/milestones as your project moves forward and it helps you see at any point how far/close you are from making a release.
- “Files and Stats” is absolutely vital for your projects - while you might not be interested in the stats that tell you how many users have downloaded your app, you are very interested in making the app available for download! So you need this more than anything else!

As I said before though, don't worry about not selecting everything that you need - you can at any point revisit your web space setup and add or remove components to it (like tickets, wiki etc).

Once you create your project you will start receiving email notifications from sourceforge as the features you have selected are made available to you. Please be aware that certain features might take a while to be made available - occasionally you might have to wait 1-2 hours for sourceforge to set that up for you. Just be patient and wait for the confirmation emails: you are beginning to take ownership of this project and as such you need to start wearing multiple hats to help your project come to life! As a tip, one thing that I found that is made available pretty much right away is the Wiki, so if you have selected other features that might take time to

become available, while waiting for that, start setting up your Wiki and do a brain-dump of all of your ideas on it!

Once your project is created, you will become automatically the project admin – so you have a free reign of changing the project as you see fit or adding tools to it. It's now time to start setting up your project! You will be given access to a shiny new Project Admin screen, which will look something like this:



Project Metadata




First thing I advise you to do is to set up your project metadata – log into your admin section (see above) to do so and select to update the project metadata – this will open a screen like the following:

Admin Wiki Files Tickets Code Mailing Lists


Metadata
Screenshots
Categorization
Tools
User Permissions
Audit Trail

Project Admin

Name:

Icon:   

Username:

Project Status: Active 

Homepage:

Export Control: [Why?](#)

This project incorporates, accesses, calls upon or otherwise uses encryption software with a symmetric key length greater than 64 bits ("encryption"). This review does not include products that use encryption for authentication only.

Short Summary:

9 characters left

Full Description:

194 characters left

Support Page:

None

Wiki

Tickets

URL:

Twitter Handle:

Facebook page:

Most of the details set in this screen are being presented in the project web space page that visitors would see when they hit your project – so it is important that you make the information in this relevant and easy on the eye. As such I definitely recommend you choose an icon for your project – 1 image says more than 1,000 words right? Set your “Homepage” to the project website (see discussion about web space vs. website in the footnotes) – you will get to edit that website later on, but make sure it’s set for now. Then ensure things like the short summary and the full description are phrased such that they explain clearly what your project is trying to accomplish. I would suggest here to stick to something similar to the “elevator pitch”, that is typically a good format for this section – more details can always be offered in the wiki or the project website.

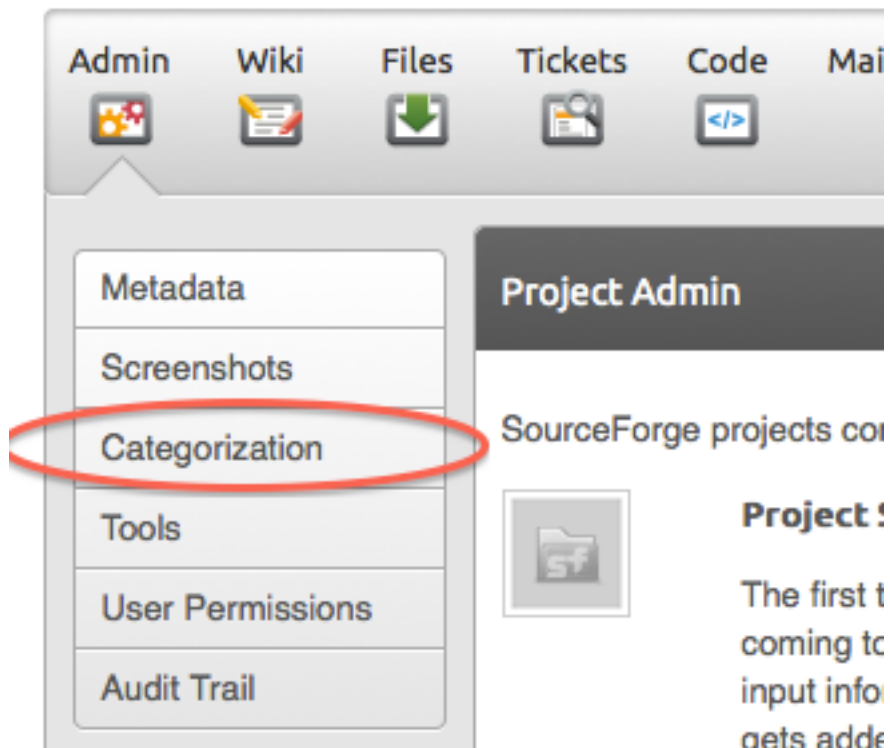
If you use Twitter and or Facebook – at this stage I would suggest you go ahead and fill in YOUR details regarding these 2 social media environments; I’m guessing you are proud of your project and as such you will be tweeting and posting on facebook about it. If your project becomes that big that it requires its own entities in the social media space, then at that point you can decide to set those up and update these

pages to reflect so – but in the initial stage you want to concentrate on implementing your project, not on spending time managing social media!

There is also the subject of “support page” – this really depends on the type of project you are going to implement – perhaps it makes sense for you to offer support after all! Other projects might not need such a feature – however, in all cases I advise you select at this point your “support” offering to be wiki: you will simply dump all the knowledge and insights about your project on the wiki. You are in the early stages of your project and cannot spend your valuable time at this moment into telling a user how to set their classpath or what’s the shortcut in Eclipse to invoke your main class. Offer the information on the wiki and don’t let “support” tickets at this point interrupt you – if you get prompted for assistance, direct the users to the wiki at this stage or engage in an email conversation if you think that they do have a valid point. Once you will have a stable project and a team with you, you can start looking at offering support – be it through the ticketing system, mailing lists, forums and so on. For now though, stick to the simple stuff.

Oh, one last thing, please make sure your project status is “Active” – no one has ever downloaded projects which are no longer active or supported by the developers and you don’t want your project to be perceived as that since you’re just starting on it!

Having finished with your project metadata there is another must-do step, which to me still falls under project meta-data, however, the sourceforge folks called it “*Categorization*”:



This bit allows you to define a few more key points about your project – things like labels which help define your project (yup, these are used by their search tool!), what sort of topics does it fall under (is it a filesystem-oriented application, does it deal with networking and so on), databases used, programming languages, OS and so on. All of these are important for targeting your **developer** audience – these are people who you might want to either use your tool or join your team (or both!), so it's important you set this information to correctly categorize your application such that it reaches the right people.

Somehow in this set of meta-data that you have to set up, there are 2 sections which I feel are somehow misplaced:

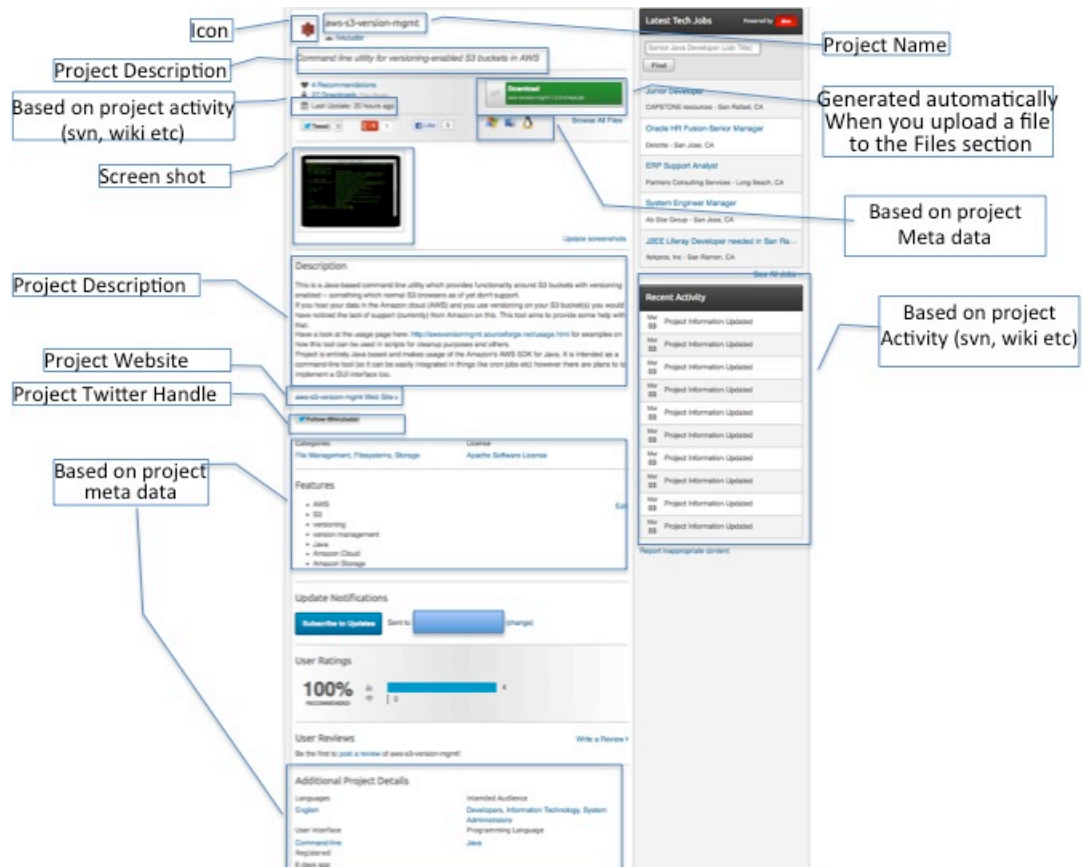
- *Intended Audience* – This helps SourceForge get this project in front of the relevant audience on the site, as such make sure you include all the possible users of your framework/tool: is it just sysadmins? Is it only Java developers? Or can it be extended to project managers too? You can always revisit this section – and in fact as your project grows you might **have** to! – but it's always good to have in your mind (and in here) an idea of who you might want to try out your app first!
- *Development Status* – This informs the audience where you are with your project. As you are creating this project you will be in stage “1-Planning” however, this stage appears on the project web space, so as you advance your project please please please make sure you keep this up to date! It is, after all, a known fact that users are easily put off by projects which are in pre-alpha, or beta mode – whereas a “Production/Stable” attracts more users, since it means you already have a release out there! Don't worry too much about it at this stage, just make a mental note that you will have to revisit this section as you advance your project through its lifecycle.

One important though not major aspect of the project in the categorization screen is also the licensing of your project! After all you are offering your project for free to the world and you want to make it clear to everyone that you adhere to some standard licenses known in the community already (be it GNU, Apache or something else). I personally prefer the Apache license – it's widely spread and understood and I somewhat find it natural to use when I'm using Apache tools as well during the build (like maven or ant for instance). However, please select whichever is right for you – though make sure you reference the same license in your project build file later on!

Last but not least, on the same idea that an image is worth more than 1,000 words, I strongly advise you to provide a screenshot of your project – if your project has a UI, then it should be pretty clear what your screenshot to contain. In the case of a web-based project, just take a screenshot of the browser. If it's a command-line utility take a screenshot of the Terminal (or Command Prompt) window. If it's just a framework, it can get a bit tricky: how do you take a screenshot of a library? In that

case I would advise you take a screenshot (terminal or in your IDE) of your unit tests passing – if your naming convention is pretty good, then at a glance this should show the user what sort of calls can be made to your library based on the unit tests names!



Having filled in all the metadata for the project, here's a preview of how this metadata will contribute to the first impression users will have when visiting your project web space:



As you can see every little bit counts – you will find that projects which are missing meta or information struggle to attract developer and user attention and build traction. Put yourself in their shoes: it's like offering your customers to buy a Mac, but the only information you're offering them is a picture of a cardboard box and a price, no technical specs, no photos of the product, no mention about software versions included and so on. The more info you offer about your project to the public, where you include it in your project meta and description, or offer it on the project website, the more confidence you build in your user and developer base – and the more likely it becomes to build traction.

Now have a look by comparison with the following (I have hidden the project and developer details since this is not a case of "name and shame" but rather just showing by comparison 2 projects with and without metadata filled in):

Planning


 

♥ [Add a Review](#)
↓ [0 Downloads](#) (This Week)
📅 [Last Update: 14 hours ago](#)

[Tweet](#) 0 [+1](#) 0 [Like](#) 0

Description

No description

 [Web Site >](#)


Categories	License
CASE	GNU General Public License version 2.0 (GPLv2)

Update Notifications

[Subscribe to Updates](#)

User Reviews

[Write a Review >](#)

Be the first to post a review of 

Additional Project Details

Languages	Intended Audience
Polish	Developers
User Interface	Programming Language
Other toolkit, Win32 (MS Windows), X Window System (X11)	Java
	Registered
	2007-12-13

Now looking at this, ask yourself realistically: for a project with no description, no explanation, no files to download, and which has been created in 2007 (6 years ago!!!) but is still in Planning stage – how much confidence does that inspire to you? Would you work on this if the guys running this project will approach you and say

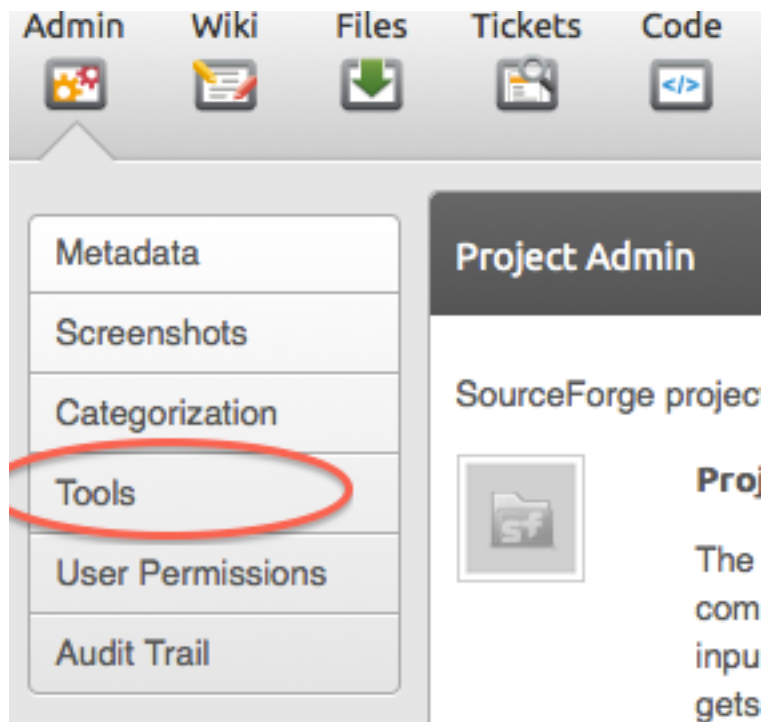
“hey, we could do with a pair of hands, wanna get involved?”. (As a side note, the project website link doesn't work either!)

The above is pretty much a “dead project” – maybe the people who started it are still working on it, but they will struggle terribly to find another developer to join their ranks and help! So unless they can put in the effort to take the project out of the planning stage, have some code and some information available, this project is going to be spiraling down: because there is no traction all the work falls on you and you only, because of that you get more and more things to be done, because of that you are making less and less progress every day to the point where you look ahead and there is such a long road to your first release that it's demoralizing to the point you probably give up and abandon it.

On the other hand, a bit of effort put into defining your project clearly, supplying a bit of information and visuals can go a long way – such that the first impression doesn't put off people, on the contrary!

Adding Tools to the SourceForge Project

I've mentioned earlier on that I would suggest you choose at least *Wiki*, *Tickets* and *Files and Stats* from the initial "Create Project" screen – if you chose not to, as mentioned, you can revisit that decision at any point from the *Admin* screen by going to the *Tools* section in the menu:



This will open up a screen which allows you to add a whole set of applications to your project to assist you with your project planning and development. At this point I strongly recommend you install the apps mentioned above (if you haven't done so at project create time) and also add the following to your project:

- *MySQL Databases* – if your project relies on a database, you will most likely need this, so sourceforge hosts your mysql databases
- *Mailing Lists* – I found this to be a very effective way of communicating to both your developers and your users, and as such I suggest you use it. As and when your project grows and you decide to offer support to your user base, you can use something like SourceForge's "Support" app, until then I suggest you set up a mailing list and use regular email for that. (I will come back to setting up the mailing list later on).
- If however you prefer a web-based communication tool to email based, you can set up the "Discussion" app as well – which allows you ultimately to build a forums section for your project and interact with users and developers that

way. I personally prefer the mailing lists, but that's just a personal preference rather than a recommendation.

- You can also add the “*Summary*” app – this simply adds a button labeled “*Summary*” in the admin toolbar which takes you straight to the project summary page – this is the page shown above, which all visitors see when looking at your project. This tool offers an easy way to quickly view what your changes look like when you're updating the project metadata / description / screenshots etc – simply make the changes and save them then click on *Summary* to get an idea right away of what your changes look like.
- Also feel free to add the “*Reviews*” app – do not be fooled by the confusion here: app reviews are enabled by default in sourceforge, this app doesn't enable that for you, it simply gives you an admin interface into viewing and managing the reviews given to your project. (The reviews functionality is a very useful one as it allows you to receive feedback from the community – I strongly encourage you to pay close attention to the feedback given to your project, since a bad review means a user had a bad experience with your project and your goal, with the open source project, is to make users' lives easier not more frustrating. Reach out to the user who gave you the review, find out what the issue is and fix it – while the bad review might stay there the same user might come back with an awesomely positive follow-up review. And having a follow-up review only shows to other users that you are committed to this project – thus increasing the confidence level in your project.)
- Note that sourceforge offers also a “*Blog*” tool as well and you might be tempted to add this as well to the application. At this point I would suggest you don't! Here's why: blogs take a while to get visibility on the net – if you have a personal blog already, you are better off blogging there about this project, future designs, features and so on – and that blog post will reach your existing audience right away. If you set up a blog on sourceforge, you will find yourself having to maintain now:
 - Your personal blog (if you have one)
 - The project wiki
 - The project website
 - The project blog
 - (Optionally) maintain any social media feeds you have to mention the project regularly in order to increase its visibility.
 - Tickets on sourceforge
 - Any mailing list you might have on sourceforge
 - (obviously) the code

These tasks can become overwhelming at some point – and potentially depressing, when you find yourself updating more web content than code! As such I would strongly suggest you don't give into temptation at this stage – if you reach version 3 of your project and you have at that point at least 3-4 other developers, you can then add the Blog app to your project since in between the few of you it's much easier to

get out a blog post once a month or so. (Again, bear in mind that a blog which hasn't been updated in months can turn users into skeptics as it might suggest the project is no longer active – no one in the open source community likes inactive projects since it typically means if there's something wrong with the project you got nowhere to turn for help!)

For now I suggest you stick to these applications only – as you can see, if you need any others you can add them to the project at any point, for now though, I suggest you keep them to a minimum necessary, as you have for a while to maintain both the code and the project infrastructure and manage the project and you don't want to end up being spread too thinly over too many things! However, you do want to set up a foundation which would make developer onboarding down the line very easy – such that the source repository is ready, information is on the wiki, roadmap is planned in tickets and so on.

Mailing Lists

While it's true you won't need the mailing list facility really until you get developers to join your project, I suggest you add this to your project right away – for a very simple reason: it takes sourceforge anywhere in between 2 to 4-6 hours to finalize the setup of a mailing list! So in foresight of having everything ready for when the first developer joins in, I'd suggest you set it all up now.

Start simple again – build just a simple “developers” mailing list, and once it's ready add yourself to it. You will be the administrator for it and I suggest a few settings for your list:

- Set it up such that all user registration requires you (the admin) to approve it – this helps with preventing spam as well as you will be notified initially every time a new developer joins, so you can have a clear idea at any point who is joining your list. Maybe you want to reach out to each subscriber individually, welcome them to the project – if you do this, it might be worth while to have a default email template for this! Or maybe you just want to email them to find out if they are a developer or a user of your project – if you notice at some point a mix of developers and users on your list it might be then the time to create a separate list for just the users and separate them from the developers! (Users will typically be interested in how to use your project, they might ask for features or for help in setting up – developers on the other hand will quite likely raise design questions, discuss about a particular unit test failing – things that are not relevant to your users.)
- Also make sure you do **NOT** allow emails from non-subscribers – if you do that, all you need is for a spammer to get access to your list email address and then your list will be filling up shortly with spam. (Remember that **all** of the emails on this list are indexed and archived by sourceforge and the archives can be browsed on the web! As such a list full of spam will decrease the confidence in your project, since people will think that there is no “administrator” guarding the project!)
- Depending on the kind of emails you expect to be sent around on this email list, set a decent max email size in the “*General Options*” settings, under “*Maximum length in kilobytes of a message body*” – sourceforge offers you the choice to select “0” for “no limit” but I advise against that, since it simply gives your subscribers the chance to send huge attachments around, which can be a waste of storage and bandwidth (bear in mind these emails get in turn broadcasted to all subscribers – some of which might have filters set on their inboxes not to accept emails bigger than XYZ!). I find the default 40Kb can be a bit restrictive (especially when dealing with rich text format or HTML), so I suggest you look somewhere around 64Kb – 100Kb. Though, depending on your project, other values might be suitable for you.

The rest of the default settings for the mailing lists I found to be ok, but feel free to tweak it here and there if you need to.

One last thing, as I said, it can take sourceforge a few hours to create your mailing list – so first just proceed to setting up the list and then I'd advise you to allow it 24 hours before you go to setting up the rest of the options described above. This way, by the time you get to making changes to the configuration you will be 100% sure that the list and all the components around it has been set up and finalized.

Uploading the Project Source Code

Having worn your project manager/admin hat for a little while and finished setting up your project in sourceforge, it's time now to become a sysadmin / devops for a little while and finally upload your code into sourceforge! I will assume for the purpose of this exercise that you went ahead with Subversion – as I explained, this is the reason why I stuck with sourceforge myself, since SVN is my first choice in source control (for now).

SourceForge creates a blank repository for you – as with all SVN repositories, you will have to create the `trunk/`, `branches/` and `tags/` directories first. (If you are not familiar with this, have a look at the SVN book here: <http://svnbook.red-bean.com/en/1.7/svn.reposadmin.create.html>). Once you have created the basic structure, follow the steps in the same link above to import your code into `trunk/` - - this is finally when you make your code available to the open source community!

Make sure you only import your sources – in other words perform a `mvn` or `ant` clean or similar and get rid of all the output generated by your build. (No need to store your jar files or generated libs in SVN, as and when you make a release you can put these in the Files section!)

One thing is worth mentioning here – I personally use `svn+ssh` – I have generated my own ssh key and uploaded it into sourceforge to use with each project I work on there. Sourceforge used to offer `svn` access over just standard HTTPS, however, having not used that myself for a while I don't think this is possible anymore. As such, you are strongly advised to set yourself up an ssh key in sourceforge – this article on SourceForge's own wiki should help with that: <https://sourceforge.net/apps/trac/sourceforge/wiki/Subversion%20client%20instructions> as well as this one: <https://sourceforge.net/apps/trac/sourceforge/wiki/SSH%20keys>

One thing is worth mentioning here: if you are using SVN + SSH, there is a config switch you might need to attend to for your `svn`; this is only needed if you encounter a message like the following every time you use the `svn` command line:

```
$ svn update  
At revision xyz  
Killed by signal 15.  
$
```

(The important bit being the “*Killed by signal 15*”!)

To suppress that message, edit your `~/subversion/config` file and look for the section `[tunnels]` which might look something like this:

```
### Section for configuring tunnel agents.
[tunnels]
### Configure svn protocol tunnel schemes here. By default, only
### the 'ssh' scheme is defined. You can define other schemes to
### be used with 'svn+scheme://hostname/path' URLs. A scheme
### definition is simply a command, optionally prefixed by an
### environment variable name which can override the command if it
### is defined. The command (or environment variable) may contain
### arguments, using standard shell quoting for arguments with
### spaces. The command will be invoked as:
### <command> <hostname> svnserv -t
### (If the URL includes a username, then the hostname will be
### passed to the tunnel agent as <user>@<hostname>.) Here we
### redefine the built-in 'ssh' scheme to avoid an unfortunate
### interaction with the "ControlMaster auto" feature (for
### details, see Debian Bug #413102):
ssh = $SVN_SSH ssh -o ControlMaster=no
### If you wanted to define a new 'rsh' scheme, to be used with
### 'svn+rsh:' URLs, you could do so as follows:
# rsh = rsh
### Or, if you wanted to specify a full path and arguments:
# rsh = /path/to/rsh -l myusername
```

All that is needed is to add a ``-q`` (quiet mode) option to `ssh` – so change the highlighted line something like this:

```
ssh = $SVN_SSH ssh -q -o ControlMaster=no
```

And that should do it.

At this point, having imported your prototype code into source forge, you have a point in `svn` to always rollback to, should any changes you make later on prove unsatisfactory in the future. Unfortunately this task doesn't stop here!

Setup the Build Process

Having uploaded the code it means you have offered your prototype code to the open-source community, however, this doesn't necessarily mean that you are ready to onboard new developers! There are a few things you need to adjust to make that a much easier, streamlined process:

- *Build process*: it's all good your code running fine in your IDE, but how about other developers? Can you just force everyone to use the same setup as yourself? Let's say you use Eclipse and an emacs user turns up – then what? You just tell them they can't join unless they install Eclipse? It's important at this stage to start thinking about how your project is going to be built – use a tool that doesn't commit anyone to any IDE – be it ant, make, maven or whatever is appropriate for your project. If you're programming in Java, maven seems to be pretty much the standard nowadays – however, ant is not too far behind. So put together a build file for your project and test it, make sure your project builds correctly then finally add the build file to SVN. Now should any emacs user decide to join the project, all you have to do is point them at the `pom.xml` or the `build.xml` or whatever your build file is. As a suggestion, if you go with Java, I suggest nowadays you go with maven – even though I am still a beginner at that, I do find it much easier to manage a project using it, and it plugs straight into Eclipse, IntelliJ and the likes. However, that's just my suggestion – as long as you use a build tool that's IDE-independent you should be good. You can also take the approach “*I'd like to use tool XYZ, however I don't know it that well but I know tool ABC*” – in that case I suggest you start with ABC and use that in the beginning; once you have a few developers joining your project, you can delegate the task at migrating from ABC to XYZ to one of them!
- *Code style*: we all have our preferred ways of writing code (does the bracket go on the same line as the method name or new line, camel case or capital case and so on), and as such to you it seems very easy to write and read your own code. Other developers might have different preferences though to yours and as such working with your code might be a bit difficult for them. So having uploaded the prototype code, it's now time to think about what code style will your project. You can take the approach that since it's your project, everyone should use **your** code style – however, that's not in the spirit of collaboration and open source, and even more, it might put certain developers off, if your code style is rather unique. I strongly suggest therefore that you select a coding standard that is well known on the net – be it the Oracle Java coding conventions⁸, or Apache conventions or something else. Even if developers looking to join your project will find these standards not as natural as their own, they will be much more likely to use it, on the basis that it is a well established and known standard rather than your own one. Once you decide on a code style, use a tool like Checkstyle⁹ or similar

and plug it into the build process to ensure this code style is met. This enables all developers to run a coding standard check when they build and test the code – and as such address any issues with their code not being inline with the coding style you have chosen. Be sure to run the tool against your own code too and fix any such issues and commit them back into SVN – having done that you now have a point in time where you know that all of your code is written 100% in line with the coding style you have chosen.

- *Code quality standards:* you know your code and you have been coding for a while, and as such you know how to protect yourself from infinite loops and the likes, but who is to guarantee that all developers who will be joining your project know all these? You should expect a mix of language and technology proficiencies from the dev's looking to join in – it's not uncommon for beginners to join open-source projects, looking to develop their skills. As such you need to set up some code quality standards checks as part of the build process – if you use java there are a few plugins and tasks for maven and ant which I recommend strongly you use:
 - FindBugs¹⁰
 - PMD¹¹
 - CPD (Copy Paste Detector) – part of PMD

Set these up in your build file to automatically generate some reports and signal to developers any potential issues with the code. FindBugs and PMD are very good at finding things like unused variables, infinite loops and misused design patterns. CPD will signal piece of identical code used throughout your project – when that happens typically it's a sign you need to review your code and refactor it a bit.

At this step, having established your code is bugs-free and in line with the coding standards you set up, I suggest you go through all of the source files and update your documentation and comments in the code such that it's easier for anybody else to understand your project's code. If you're a java coder, this means ensuring every method and member is javadoc'd – and if you are using Checkstyle in your project, you can have checkstyle verify that this is the case and warn you when you are missing comments. (If you are using java it is important to have your javadoc complete and up-to-date because later on, we will be publishing all of the javadoc's on the project website – as you will see later on!)

When setting up the project build file, don't forget to reference your project license in it – maven has dedicated support for this, by using a construct like this for instance (if you're using maven as your build tool) in order to reference the Apache license:

```
<licenses>
  <license>
    <name>The Apache Software License, Version
2.0</name>
    <url>http://www.apache.org/licenses/LICENSE-
2.0.txt</url>
    <distribution>repo</distribution>
```

```
        <comments>A          business-friendly          OSS
license</comments>
    </license>
</licenses>
```

Though similar mechanisms are available for other build tools – just check which one applies to you. It’s always a good idea to add your license to your build file – in most cases this will be inserted somewhere in the deliverables / release files, such that even if someone doesn’t visit the project website to check the license but end up downloading your file (from a mirror site perhaps?) can still figure out your licensing model from the deliverables and be encouraged to use your project based on that.

Maven-specific Setup

If you are working on a java project and using maven as your build tool, there are a few other issues you are encouraged to add to your *pom.xml* file, which will help later on with your project setup:

First of all make sure you fill in the basic information in your pom about the project: set the *<name>* of the project to correspond to the name you gave your project in sourceforge, the *<description>* I suggest you set to the same one you filled in sourceforge too. The *<url>* you will have to set up to the url of your project web space – e.g. <https://sourceforge.net/projects/awsversionmgmt/>

Next set up the *<organization>* element in the pom – this should be you to start with and feel free to unashamingly promote your personal website in there: after all this is your work! Something like this will work:

```
<organization>
    <name>Livi Tudor</name>
    <url>http://liviutudor.com</url>
</organization>
```

It’s worth setting up the *<scm>* element as well – this will help you later on with things like automated deployments and so on. If you are using subversion on sourceforge, your *<scm>* should look something like this:

```
<scm>
    <connection>scm:svn:https://svn.code.sf.net/p/awsversionmgmt/code/trunk/</connection>
    <developerConnection>scm:svn:https://svn.code.sf.net/p/awsversionmgmt/code/trunk/</developerConnection>
    <url>http://svn.code.sf.net/p/awsversionmgmt/code/</url>
</scm>
```

(Obviously replace *awsversionmgmt* with your project Unix name in the above.)

If you have set up the “*Tickets*” app in your sourceforge project, then it’s worth adding the `<issueManagement>` element to your pom too, which should look like this:

```
<issueManagement>
  <system>sourceforge</system>
  <url>http://sourceforge.net/p/awsversionmgmt/tickets/</url>
</issueManagement>
```

I strongly advised you earlier on to select “*Files and Stats*” as one of the tools to be added to your project setup – if you haven’t added yet to your project, please go ahead and do it now. Once you do it, you can set up your `<distributionManagement>` in your pom too – which later on will help you with automated deployments:

```
<distributionManagement>
  <repository>
    <id>sf-net-repository</id>

    <url>scp://shell.sf.net/home/frs/project/a/aw/awsversionmgmt/maven<
/ur>
  </repository>
</distributionManagement>
```

(We will revisit shortly the `<distributionManagement>` setup when we get to the project site – but for now this should do. One thing worth noticing here is the way the file system is split up in subdirectories – first based on the first letter of your project – **a** in the case of *awsversion* management – then based on the first 2 letters – **aw** in this case – and finally the full name – so **a/aw/awsversionmgmt** ; you will have to adapt this path according to your own project name.)

Last but not least, add yourself to the developers list:

```
<developers>
  <developer>
    <id>livt</id>
    <email>me AT liviutudor DOT com</email>
  </developer>
</developers>
```

As people will start paying attention to your project and want to join in you can add them to this list – such that they appear in the automatically generated project reports (we’ll get to that bit later).

If you have set up your mailing list already, it's worth adding that to your pom in `<mailingLists>`; sourceforge doesn't offer a mechanism to subscribe and unsubscribe to the list by sending emails to specific addresses, so just include the mailing list in `<subscribe>` and `<unsubscribe>`, same as you would with `<post>`:

```
<mailingLists>
  <mailingList>
    <name>Developers List</name>
    <subscribe>awsversionmgmt-
developers@lists.sourceforge.net</subscribe>
    <unsubscribe>awsversionmgmt-
developers@lists.sourceforge.net</unsubscribe>
    <post>awsversionmgmt-developers@lists.sourceforge.net</post>

    <archive>https://sourceforge.net/mailarchive/forum.php?forum_name=aws
versionmgmt-developers</archive>
  </mailingList>
</mailingLists>
```

(Again, replace the links and the email addresses with your own.)

Deployment

The purpose of most open-source software projects (on SourceForge and otherwise) is to provide some tool or libraries or framework the user can download and use. In the case of Java projects this typically refers to a bunch of classes – in most cases grouped together in a jar file. (Typically these are referred to as *deliverables* or *binaries*.) This file is normally generated by your build process – be it ant, maven or some other tool.

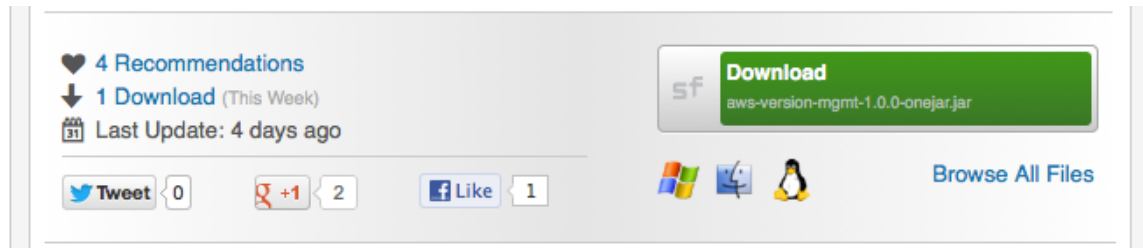
In fact let me rephrase that: this **has** to be generated by your build process! What I mean by that is the fact that you need a build process which automates most of the tasks – including generating these binaries; rather than having them packaged by some manual process. A manual process requires the person building the packages to remember a set of steps, each one with their particularities. Since you are going to set up the build process, these steps would probably stick in your mind very well and you might argue that you can't possibly forget them. Also you would argue that you could put it on the project wiki – and that's true, you can! However, imagine that your project develops enough traction to get the attention of a few developers – and they might or might not pay attention to the wiki page or they might even forget some of the steps involved. And in an open-source environment you can't really take the approach of "I'll be the only one performing builds" – or if you do you will find that you end up consuming a lot of your time performing admin tasks (i.e. creating a release and uploading it) rather than concentrating on implementing and coding

your project! There are lots of articles out there who explain why this is so I'm not going to stress on this, but just enough to state that you have to tailor your build process such that it produces all the binaries somewhere in some directory.

By now you should have "installed" on your SourceForge project the "*Files and Stats*" add-on – as I explained earlier on this gives you 2 powerful features for your project:

- The "*Stats*" module provide you (and your project visitors) with some basic statistics about the number of downloads, recommendations, things like shares on social media channels and so on
- The "*Files*" module provides you with a storage / deployment / download area where you can publish files and make them available for download by your users. This is the area that you need to configure your project to deploy your binaries to. Once a file is uploaded into the "*Files*" section of the project, the "*Download*" button automatically gets updated to offer a one-click download of the latest uploaded file in this section.

For example, in the case of the *aws-s3-version-mgmt* project, having just released version 1.0.0 of the project (which resulted in *aws-version-mgmt-1.0.0.one-jar.jar* file to be uploaded in the "Files" section), the "Download" button automatically gets updated to point to this file:



Needless to point out, this is a very convenient feature for users of your project, who in most cases don't want to navigate through lots of pages to download your project, but rather want to download it quickly and start using it right away. (If you're like me, then you are probably used to this pattern: download the software, install it and run it and only if it doesn't work – or doesn't work the way I thought it should – go and consult the wiki/manual/documentation/etc.)

However, this feature has a few implications:

- If your application consists of a few deliverables, then you need to find a way to group them into one single file – which would then appear on the download button. This means that if your binaries consist of a few files, you need to package them together before deployment (use something like tar, zip etc).
- The last file that gets uploaded to the file section has to be the "main" deliverable. This means that if during your deployment process you deploy a few files (for instance if you're generating a jar with the sources, another one with the javadocs and another one with the compiled java classes), you want

to make sure the last file you deploy is the one that users can download and run your application from it right away (in this case it will be the compiled java classes packaged in a jar file).

Packaging

I'm sure whichever build tool you are using there are options to choose from, but if maven is your build tool, I have found so far 2 ways of achieving the above:

1. By using the maven assembly plugin ¹²– this is a maven plugin which allows you to have your project packaged into tar/gz/bzip2/jar/zip/war files and also allows customization over directory structures and contents.
2. By using the one-jar maven plugin ¹³– this is another plugin which packages all of your jar dependencies and code into one executable jar – such that users can just invoke it via `java -jar ...`. That is quite useful if your project is an application (rather than a framework, or a set of applications etc)

There are probably more than one way of deciding which one of these 2 you should use with your maven setup, but here's a quick rule of thumb from me on these:

- If what you're building is a Java application – as in a single java app, not a collection of apps – then in most cases you want to go with the *onejar-maven-plugin* : if you're providing an application for download you want your users to be able to double-click a file and start the app right away, and an executable jar allows you to do just that! If you distribute your app as a tarball with directories for libraries and so on, then the script for running your app becomes a bit convoluted (you'll have to specify the classpath etc etc etc) – using this plugin, all the user has to type once they downloaded the one-jar file is: `java -jar /path/to/one-jar.jar` – and voila! Remember, from a user experience point of view, it's important that your users get to use your app as soon as possible from the moment they made the decision they want to give it a go – have them go through page after page of instructions on how to configure and run your app loses their interest in lots of cases, so a simple yet effective way of packaging your application like this provides a huge user experience improvement!
- If however what you're providing is a framework, or a set of applications or some libraries, then onejar-maven-plugin is probably not the best way – while you might still be able to package everything using it (ultimately by default it will create a jar containing your application compiled code AND the dependency jars), the end result might not be the best way to deliver your framework to the users. Typically with things like libraries and frameworks you probably have a certain way of setting up files in directories, maybe include some shell scripts, some configuration files, some logging setup – all of this is not really possible with the onejar-maven-plugin, however the maven assembly plugin allows you to setup directories, copy files over and then finally package everything into a tarball, or zip file, jar or all sorts of other “standard” package formats.

There is potentially a 3rd way of packaging your application as well, however, this is a rather “specialized” way of doing it: RPM! There is a maven rpm plugin ¹⁴which allows you to package your project into an rpm – however, obviously doing so limits your distribution to Linux distros; if your project is specifically targeted at Linux (CentOS, RedHat, etc) then you might want to consider this – however, we won’t go into details about this plugin here.

In the case of *aws-version-mgmt*, since this is a command line application, I’ve gone down the route of onejar-maven-plugin – as such, here’s an outline of the maven configuration I’ve used for it together with some explanations:

```
<repositories>...</repositories>
...
<!-- Declare the repository where to download the onejar plugin from -->
<pluginRepositories>
  <pluginRepository>
    <id>onejar-maven-plugin.googlecode.com</id>
    <url>http://onejar-maven-
plugin.googlecode.com/svn/mavenrepo</url>
  </pluginRepository>
</pluginRepositories>
...
<build>
  ...
  <!-- use the onejar plugin to package everything -->
  <plugin>
    <groupId>org.dstovall</groupId>
    <artifactId>onejar-maven-plugin</artifactId>
    <version>1.4.4</version>
    <executions>
      <execution>
        <configuration>
          <!-- Set the mainClass so when user launches java -jar one-jar.jar this class gets
executed -->
          <mainClass>com.liviutudor.aws.versionmgmt.Program</mainClass>
          <!-- Set to true to have the package phase generate the one-jar file -->
          <attachToBuild>true</attachToBuild>
        </configuration>
        <goals>
          <goal>one-jar</goal>
        </goals>
      </execution>
    </executions>
  </plugin>
  ...
```

```
</build>
```

As you can see, there's nothing really to the configuration – apart from declaring the main class to be set in the manifest (this is the class which gets called when using *java -jar* in the command line), and attaching the plugin to the maven package phase there's nothing else to be done.

Uploading Files to SourceForge

Having decided on the way to package your project (one-jar, tarball, zip etc), and set up your pom accordingly, it's finally time to now look at uploading this package to the SourceForge "Files" section. Of course you can do this manually, via FTP and all sorts of other mechanisms, however, as I said before, all of this (packaging + upload + version rev'ing) needs to be done automatically by your build process, so it's very easy to maintain. And all of this can be done by using the maven release plugin¹⁵ together with the maven wagon¹⁶ build extension.

The maven release plugin already does the donkey work for us:

- Runs a full build (you can configure what phases to run, but typically you want to do a full clean package, so run unit tests and generate the package file)
- Updates your pom version – it takes the user through a series of prompts but it makes some smart assumptions about it, so in most cases you can just accept the default values it suggests
- Tags your source control repository with the version number
- Uploads your package file(s) to your maven repo

So based on this, all that is needed is really to configure the bits regarding the source control integration and package upload.

The SVN / source control integration setup has already been done previously when we set up the `<scm>` element of the project. Also, we have touched on the `<distributionManagement>` element too – this defines the url where your maven artifacts are being uploaded when you go through a maven release. As you can see it has 2 components: `<repository>` and `<site>` -- at this stage it is the `<repository>` bit we are interested in, since this is where your artifacts will end up being uploaded.

```
<distributionManagement>
  <repository>
    <id>sf-net-repository</id>
    <url>scpexe://shell.sf.net/home/frs/project/a/aw/awsversionmgmt/maven</url>
  </repository>
  <site>
    <id>awsversionmgmt.sf.net</id>
```



```
        <url>scpexe://shell.sf.net/home/project-  
web/awsversionmgmt/htdocs</url>  
    </site>  
</distributionManagement>
```

In order to use SCP / SSH for uploading files, we need to simply reference the Maven Wagon extension in the build:

```
<build>  
<extensions>  
    <extension>  
        <groupId>org.apache.maven.wagon</groupId>  
        <artifactId>wagon-ssh</artifactId>  
        <version>1.0</version>  
    </extension>  
</extensions>
```

and finally configure the maven release plugin as well – which pretty much consists of including it in the <plugins> sections of your pom and setting up a bunch of properties – in the case of aws-version-mgmt project this looks like this:

```
<plugins>  
...  
<plugin>  
    <groupId>org.apache.maven.plugins</groupId>  
    <artifactId>maven-release-plugin</artifactId>  
    <version>2.4.1</version>  
    <configuration>  
        <!--what goals to run during the release -->  
        <preparationGoals>clean verify install</preparationGoals>  
        <!--if you're releasing snapshots,you want this to set to true to avoid  
releasing the same snapshot version twice with different code; it simply  
generates an extra date/time stamp to the version -->  
        <allowTimestampedSnapshots>true</allowTimestampedSnapshots>  
    </configuration>  
</plugin>  
</plugins>
```

Now with this set up, you can simply do the usual *mvn release:prepare*, *mvn release:perform* and you'll see that auto-magically your one-jar file gets uploaded (after your application artifact jar too!) and the Download button will automatically point to it. Also, your svn repo gets tagged and pom gets updated!

References and Links

Throughout the book there are a few terms and software packages mentioned, this chapter is meant to provide more information about these.

¹ Maven – a Java build tool from Apache Software Foundation; visit the project website for more information: <http://maven.apache.org/>

² Henry Ford – the founder of Ford Motor Company, see this article on Wikipedia for details: http://en.wikipedia.org/wiki/Henry_Ford

³ Read more about the versioning feature in S3 here: <http://aws.amazon.com/about-aws/whats-new/2010/02/08/versioning-feature-for-amazon-s3-now-available/>

⁴ You can read the blog post I'm referring to following this link: <http://liviutudor.com/2013/02/27/utility-for-version-enabled-aws-s3-buckets/>

⁵ The Amazon SDK for Java documentation is freely available at <http://docs.aws.amazon.com/AWSJavaSDK/latest/javadoc/index.html>

⁶ The Apache Commons CLI provides an API for parsing the command line parameters. Read more about it here: <http://commons.apache.org/proper/commons-cli/>

⁷ When talking about “web space” and “web site” in sourceforge, the differences are rather subtle and can be confusing at first: the web space is a directory that sourceforge creates off it's <http://sourceforge.net/projects/> directory – and this is the space intended mostly for the developers, project admins, etc of the project. This is where things like ticketing, source code, wiki etc are made available. The website of the project is rather intended to be used by the **users** of your projects – these guys are quite often not interested in design issues being discussed by your developers, or what's going to be plugged into the next release. They want to see a bunch of “HOW TO” documents, FAQ's and quite likely a download button. While there's nothing from preventing a developer publishing the same information on both the web space and the website, the envisaged targeted audience is different for both.

⁸ Oracle's Java coding conventions can be found here: <http://www.oracle.com/technetwork/java/codeconv-138413.html>

⁹ Read more about checkstyle on the project website (yes, on sourceforge!): <http://checkstyle.sourceforge.net/>

¹⁰ FindBugs' website: <http://findbugs.sourceforge.net/>

¹¹ PMD (“Project Mess Discovery”) website: <http://pmd.sourceforge.net/>

¹² Maven assembly plugin project page: <http://maven.apache.org/plugins/maven-assembly-plugin/>

¹³ The onejar-maven-plugin project page: <http://code.google.com/p/onejar-maven-plugin/>

¹⁴ Maven RPM Plugin project page: <http://mojo.codehaus.org/rpm-maven-plugin/>

¹⁵ Maven release plugin project page: <http://maven.apache.org/maven-release/maven-release-plugin/>

¹⁶ Maven Wagon is a transport abstraction plugin for maven – read more about it on the project page: <http://maven.apache.org/wagon/>