



# TeleStax Open Source Playbook

Ensuring Effective Collaboration

2016

---

Jean Deruelle,

CTO

TeleStax, Inc

350 Cambridge Ave, Suite 250

Palo Alto, CA

## [Overview](#)

### [Open Source Responsibilities](#)

[Work in the open](#)

[Have a clear and public roadmap](#)

[Create Great Documentation](#)

[Ship! - Release Cycles](#)

[Versioning](#)

[Help the community](#)

[Marketing](#)

[Contributors](#)

[Licensing](#)

### [Technologies used by the RestComm platform](#)

### [Development Process](#)

[Everything starts with an issue](#)

[Design](#)

[Implementation](#)

[Code Quality](#)

[Code Best Practices](#)

[Create your own Fork](#)

[Create your own Feature or Fix Branch](#)

[Tests](#)

[Non Regression Tests](#)

[Manual Tests](#)

[Performance Tests](#)

[Documentation](#)

[Source Documentation](#)

---

- [Editing the Documentation](#)
- [Inserting UML Diagrams](#)
- [Outputting the Documentation](#)
- [Theming](#)
- [Hosting](#)
- [Continuous Documentation](#)
- [Process](#)
- [Committing code](#)
- [Pushing changes to your online clone](#)
- [Pull Request](#)
- [Pull Request Review](#)
- [Q&A Process](#)
  - [Continuous Integration](#)
  - [Continuous Delivery](#)
- [Release Process](#)
  - [Creating a release on github](#)
  - [Make a Release Announcement](#)
  - [Market your release](#)
- [Hiring Process](#)
- [Git Useful Commands](#)
  - [Git Cloning & Branching](#)
  - [Git Committing](#)
  - [Pushing changes to your online clone](#)
  - [Pull Request](#)
  - [Pull Request Review](#)

## Overview

Open Source is at the core of everything we do and is the foundation of our history and company culture. Although we have mentioned our office address, it is mainly for reference as the entire company and contributors are working in an open and distributed manner from day one ([similar to many others nowadays](#) with [very good reasons for doing it](#)), now across more than 30 countries at the time of writing.

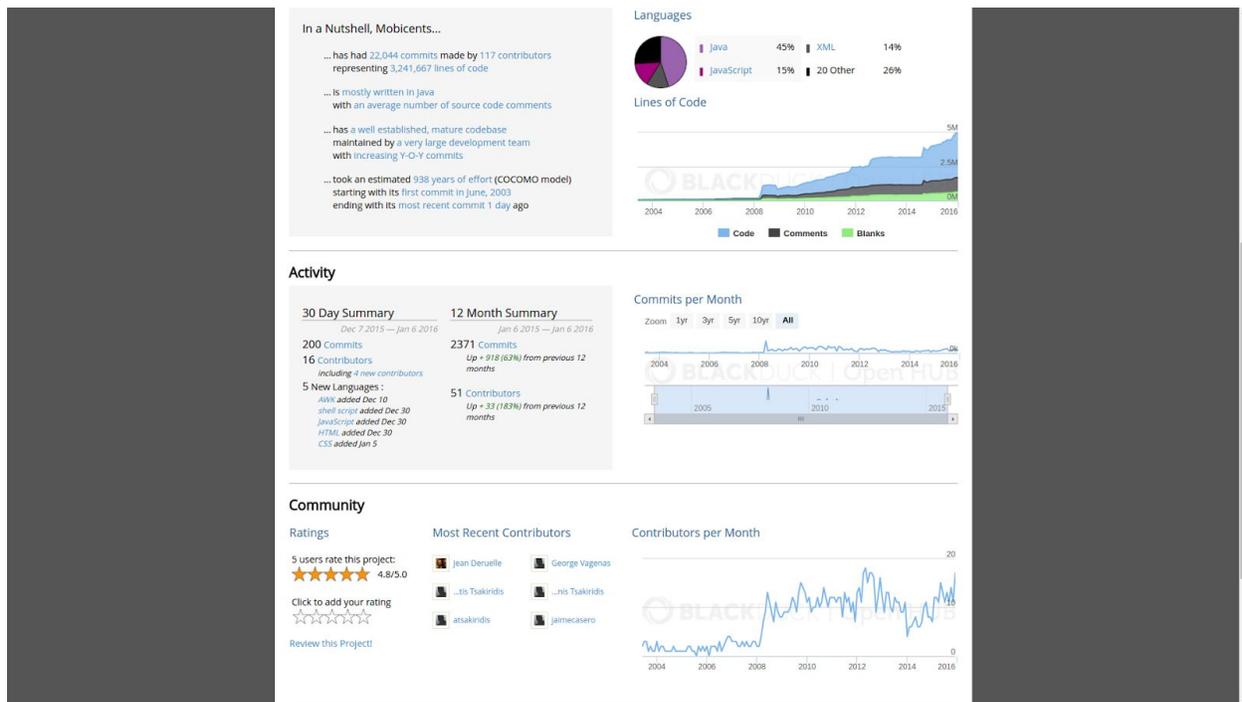
## Global Team and Partners



This Playbook contains Best Practices on evolving your open source project and growing an active community around it based on our 13 years of building the biggest Open Source Communications Platform : Mobicents [which has just been rebranded to RestComm](#).

The community has just reached a [very impressive growth milestone](#).

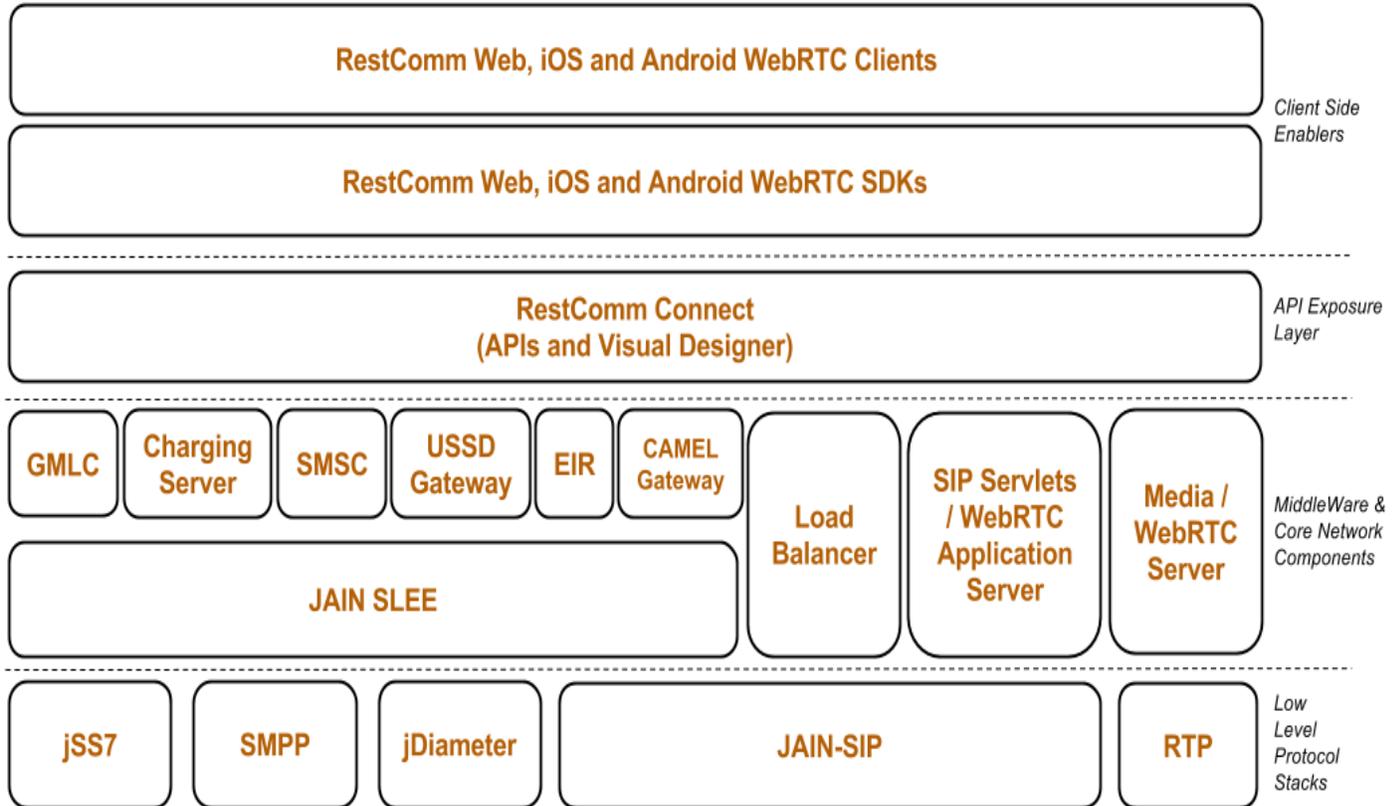
The independent WebSite [OpenHub](#) shows at the time of this writing that the codebase tripled since TeleStax was created and that it would take 51 Million Dollars to rebuild the project for an average salary per year of \$55K.



In a Nutshell, RestComm... has had [22,150 commits](#) made by [117 contributors](#) representing [3,240,672 lines of code](#), mostly written in Java with an average number of source code comments has a well established, mature codebase maintained by a very large development team with increasing Y-O-Y commits took an estimated [938 years of effort](#) (COCOMO model) starting with its [first commit in June, 2003](#) ending with its [most recent commit about 24 hours](#) ago

TeleStax leads and maintains the following Open Source projects that constitute the RestComm Open-Core family. Telestax offers a carrier grade, commercial quality product under the brand TelScale which is based on the RestComm open core. See our [Open Source or Enterprise Grade](#) page for more.

See below for the full list of open source projects, TeleStax is driving



---

## Open Source Responsibilities

Creating an effective Open Source community requires resource investments - both in time and money. Just like anything that produces value however, it's worth the effort. Done poorly and you're not doing much more than a code dump. Done well, it can develop a life and culture of its own.

In this playbook, we will outline the responsibilities that we, at Telestax, abide by and enforce for the greater good of everyone involved in developing and cultivating this effective Open Source community. Continue reading to learn more about creating and sustaining roadmap, documentation, versioning, and more in Open Source

You can watch this good video on [growing an open source community](#) by Dan Allen.

With great power comes great responsibilities:

### 1. Work in the open

- a. All design, code, documentation, tests are in the open so we must guarantee great quality and involve the community as much as possible in design discussions and feedback on new features, etc. This is extremely important to create and grow a strong community. Any new feature or design should be sent for review and feedback through a thread on our public Google Group <https://groups.google.com/forum/#!forum/restcomm>.
- b. Every member of the team should be connected to the public gitter channel <https://gitter.im/RestComm/Restcomm-discuss>. It is very important so we can keep the contributors engaged and benefit from public brainstorming.
- c. Every Wednesday at 5 PM CET, TeleStax is holding a community status update for all the projects of the platform where anyone can join and listen or chime in.

### 2. Have a clear and public roadmap

- a. Needless to say this one is extremely important for everyone to know where the project is headed and how potential contributors can contribute.
- b. Ask community and customers for feedback on the roadmap but as project leaders you set the vision and direction.
- c. Define Milestones in your Github project (by example) <https://github.com/RestComm/RestComm-Connect/milestones>

- 
- d. We use the excellent [Waffle.io](https://waffle.io) for visualization and maintaining of the current roadmaps for our open source projects (by example <https://waffle.io/RestComm/RestComm-Connect>). You can filter per milestone, per person, ...
  - e. It's good to make extensive use of GitHub Labels. We have created a number of specific labels that help us filter better the milestones and associated issues. There is two specific ones that are worth spending some time on.
    - i. [Help Wanted Label](#): This label tags all issues that contributors can take to help the project. Ideally, new labels with difficulty for each issue should be added so new contributors can take the easy ones and as they get more advanced they can pick harder issues and eventually get hired.
    - ii. [Technical Debt Label](#): No matter what you do, there is always technical debt accumulating for any project for various reasons. Instead of denying it and having quality issues, we label clearly those issues and enforce that they will be scheduled and taken care of in roadmap planning.

### 3. Create Great Documentation

- a. Even if it's a higher initial investment to document the project it is mandatory. (Team members can help by creating issues for missing docs.) R&D team members should always have a draft of docs that can be tested and polished by documentation team.
- b. Any feature that is worked on need to be documented and usually documented with examples.
- c. The better the project is documented the more people can find their way around by themselves. This also means less community and customer support questions there will be, thus freeing up time to move forward on the roadmaps faster

### 4. Ship! - Release Cycles

- a. Aim for 1 to 1.5 months release to stay agile and keep the community engaged with fixes to bugs reported and new features. One feature and one technical debt issue per person working on the project per release is good practice. The rest of the time is spent on bug fixing and improving

---

some areas, which may have been left behind because of customer support overload, travelling or other factors.

- b. It is very important to follow the *release early, release often* strategy to keep the community engaged and allow for small fast incremental improvements instead of big monolithic releases with big impact on stability or changes.

## 5. Versioning

If you're a project lead, we are following the [Apache Conventions for Versioning](#) i.e., Major.Minor.Patch. So usually your Maven pom file will have a version similar to Major.Minor.Patch-SNAPSHOT. However, our [Continuous Delivery](#) will be responsible for continuously releasing every time there is change in the master branch of the project. As a result, the final version will be Major.Minor.Patch.BuildNumber.

## 6. Help the community

- a. Go to your forum every day to show users that the forum is alive.
- b. Never leave a post on your forums or StackOverFlow without response more than four business days. Ideally wait one or two business days so that community members can answer first and get involved (in order to spot the next potential contributors to recruit in 7).
- c. Make sure you are notified of any new topics on your project's tag questions on StackOverflow
- d. Never leave an issue from community in the issue tracker without response for more than four business days. Before prioritizing it on the roadmap, always ask the reporter if they want to contribute a fix (in order to spot the next potential contributors to recruit in 7) otherwise people will ask you to fix their code and monopolize your time (and prohibiting you from being able to get necessary work done on the project).
- e. Ideally issues reported by community should be fixed in the current release cycle except if they require a big refactoring. Issue reported by community should be treated as second priority after customers support tickets in order to keep the community engaged and using/growing the project. Since most of our prospect leads are inbound leads usually generated by community users going production this is extremely important. A number of customers disclosed they were tracking our response time in community to check if we would be providing good quality support before buying anything from us.

---

## 7. Marketing

Be prepared to spend time on not developing your project, and evangelizing the project instead, which includes;

- a. Publish on social networks (Tweets, FB, LinkedIn, ...) at least weekly or when working on a new feature.
- b. Always do a blog post and publish it on social networks for every release you make.

## 8. Contributors

From our experience on Mobicents and RestComm, TeleStax team noticed three kinds of users:

- a. **Regular users** : who use your product and ask questions and after being comfortable also answer questions
- b. **Contributors** : who use your product, ask questions, answer questions and contribute code
- c. **Certified Partners:** At Telestax, our partners are essential to the success and growth of our business. As our business grows, we are more and more focused on building strong partnerships that strive to execute on a shared vision and strategy that is based on **trust** and **transparency**. The Certified Partner commits experts from their own team to be Contributors so that the partnership is built on the foundation of openness, community and competencies. This enables our partners to ensure they have insight and input on our product roadmap, access to sales, marketing, and technical resources from the Telestax core team. This also allows TeleStax to recommend to its customers, certified partners that are deeply involved and recognized experts. Read More on the [TeleStax Certified Partnership Program](#)

There is also multiple types of possible contributions, not necessarily code only :

- a. Using it in your product or project and providing feedback.
- b. Code & Algorithms: Core Projects, Incubator projects, Frameworks
- c. Use cases, feature requests: Roadmap influence
- d. Community Support, bug fixes, forum posts: Help to be helped

- e. Documentation: Everyone needs good docs, Code is a moving target.
- f. Testing (Perf, load, security, unit tests, interop, ...) / CI

For b, e and f it's mandatory that the Contributor signs [the TeleStax Contributor Agreement](#). **When a contributor contributes to the project, acknowledge him on a [public Acknowledgments page](#)**

As the Contributors ratio grows, you now become the leader of the project who gives the direction, the goals and works on keeping everything coherent and aligned.

Contributors are usually naturally attracted if the project lead executes correctly on doing all of the above i.e., great product with regular updates and bug fixes, great documentation, marketing and keeping the community and contributors engaged. But remember, contributors are a very small percentage of the user base usually and it takes time and effort to attract them but it's key to engage them as the best of them become part of the team as the company grows as it is our only [Hiring process](#).

## 9. Licensing

RestComm is licensed under AGPL : See community obligations on AGPL Code <http://info.protecode.com/bid/40416/AGPL-and-the-cloud-what-are-your-obligations>

Use the below AGPL license header for source code

```
/*
 * TeleStax, Open Source Cloud Communications
 * Copyright 2011-2016, Telestax Inc and individual contributors
 * by the @authors tag.
 *
 * This program is free software: you can redistribute it and/or modify
 * under the terms of the GNU Affero General Public License as
 * published by the Free Software Foundation; either version 3 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 Affero General Public License for more details.
 *
 * You should have received a copy of the GNU Affero General Public License
```

---

```
* along with this program. If not, see <http://www.gnu.org/licenses/>
*/
```

Modern IDE can automatically add license header for every new file you add to the project, check the links below:

- [https://wiki.eclipse.org/Development\\_Resources/How\\_to\\_Use\\_Eclipse\\_Copyright\\_Tool](https://wiki.eclipse.org/Development_Resources/How_to_Use_Eclipse_Copyright_Tool)
- <https://www.jetbrains.com/help/idea/15.0/generating-and-updating-copyright-notice.html>

## Technologies used by the RestComm platform

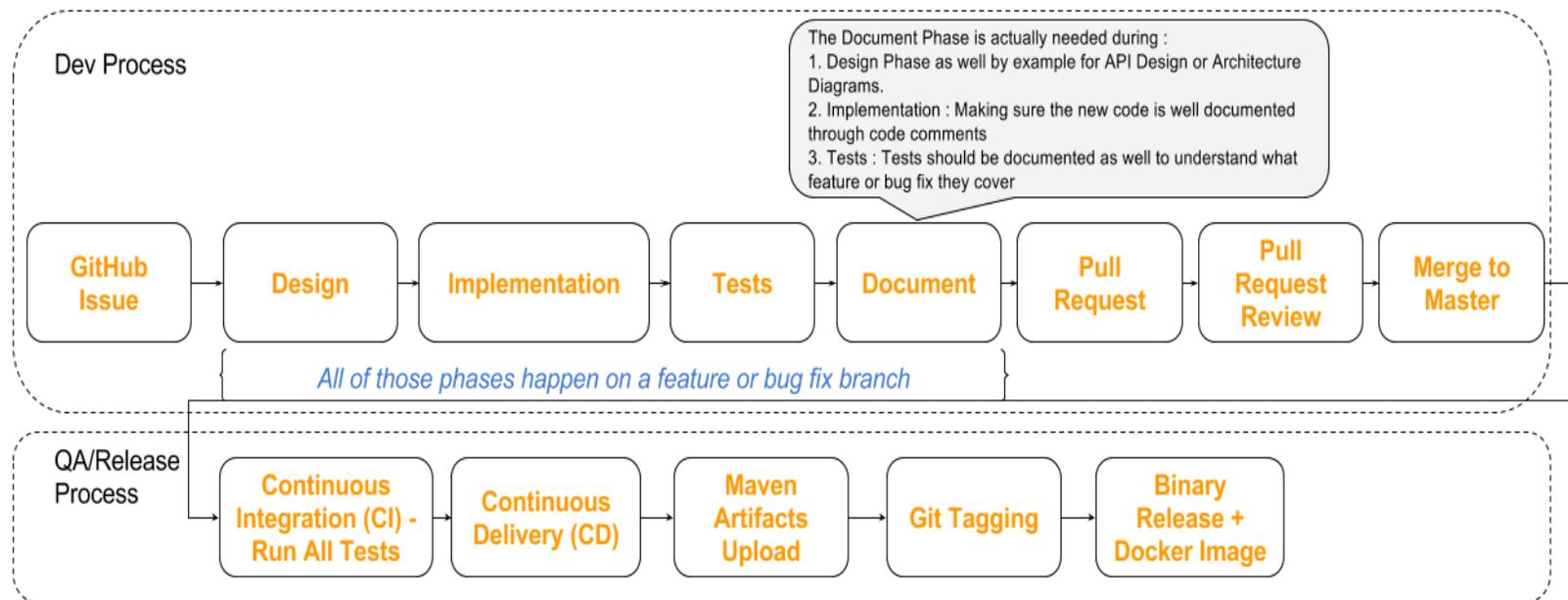
The Restcomm Open Source Platform is fully Java based. As such we use a variety of tools such as Java JDK, Eclipse or IntelliJ IDE for development, Maven and Ant as build manager, Git as decentralized version control system, Jenkins for Continuous Integration and Delivery. Please make sure you're familiar with those technologies if you want to contribute to the Platform

## Development Process

*Contributor* is used as term below to describe either a team member or community contributor

The model we've chosen for developing RestComm is somewhat akin to this one <http://nvie.com/posts/a-successful-git-branching-model/> where each contributor will create their own branch to work on an issue and do a Pull Request so it can be merged to master.

Here is a visual view of it



Here is a more indepth procedure below:

## 1. Everything starts with an issue

Before any work is done, an issue needs to be created in the corresponding github repository. For example <https://github.com/RestComm/Restcomm-Connect/issues/new>. The issue should explain the task that needs to be done.

If the issue is to be implemented by a contributor, then the contributor should provide his/her Github profile and username so he/she can be **added as a Collaborator** with **Read** access rights to the repository and the issue can be assigned to him/her.

Only then the next phase can start.

## 2. Design

- i. Any new feature should go first through a design round which includes producing a mix of architecture documents, diagrams and supportive documents which should be added to the github project contributors documentation (wiki pages or github Pages) so that future contributors can easily find their way around and understand the basis of a given feature.
- ii. Each feature should be brainstormed and discussed internally and publicly so the team mates and community can provide feedback

- 
- iii. After there is a common agreement or final decision made, it's time to move to implementation

### 3. Implementation

#### i. Code Quality

Each contributor should write good code that is well [tested](#) and [documented](#).

The maven process guarantees that the code is indented correctly as per our conventions.

The contributor must make sure that the [project license header](#) is present in all files contributed.

The contributor shouldn't in general commit code that is tied to multiple issues at the same time. 1 commit == 1 issue, it makes it easier to understand, review, break down and potentially backport a commit.

#### ii. Code Best Practices

1. Logging : Follow <https://wiki.base22.com/display/btg/Java+Logging+Standards+and+Guidelines> . Section Use Guarded Logging is specially important for performance.
2. Package Naming: use org.restcomm

#### iii. Create your own Fork

Each contributor creates his own fork of the RestComm project (they want to contribute to) repository. This clone is hosted on Github servers, and can be created by clicking **Fork** button from the github project for example <https://github.com/RestComm/Restcomm-Connect>

The contributor then makes a local clone of their GitHub fork, which is stored on their local machine. Instructions for checking it out is [https://github.com/<Contributor\\_github\\_account>/RestComm-Connect](https://github.com/<Contributor_github_account>/RestComm-Connect) (replace <Contributor\_github\_account> in that URL with your github account, example <https://github.com/deruelle/RestComm-Connect>)

#### iv. Create your own Feature or Fix Branch

The contributor then creates a new branch into their local clone and do the changes into their local branch for the contribution and [commit](#) them.

---

**Note:** Please see [Git Useful Commands](#) if you're not familiar on how this works in Git

## 4. Tests

### i. Non Regression Tests

Every feature or bug fix needs to have at least one meaningful automated non regression test along with the code that is committed. If there is no tests, then the contribution will not be accepted. This is to guarantee the quality of the project as the codebase evolves.

### ii. Manual Tests

It may happen for multiple reasons that the code change needs to have a number of manual tests performed. Those should be clearly documented with the reason on why they can't be automated. In most cases, external systems can be mocked but it may happen this is harder to do for NAT related issues by example

### iii. Performance Tests

It's very important that each project contains a set of performance tests for typical use cases to assess the performance and identify any possible memory leak or performance bottlenecks.

## 5. Documentation

At TeleStax, we treat Documentation like any other piece of code and it follows the same processes and rules. We currently use the great [AsciiDoc](#) and [AsciiDoctor](#) to write, theme, publish and version our documentation.

### i. Source Documentation

Our source documentation (the raw content) is hosted on github itself which means anyone can actually contribute to it as well.

Our Source Documentation is bundled with every project at the same location ie docs/sources-asciidoc. By example, <https://github.com/RestComm/sip-servlets/tree/master/docs/sources-asciidoc/src/main/asciidoc>.

### ii. Editing the Documentation

Github is capable of interpreting AsciiDoc, which means you can actually preview the modified content as you update the documentation. You can also checkout

---

the documentation locally by cloning the repository and going into the location of the docs and edit it. Alternative editors like [AsciidocFx](#) (includes an editor and preview), [Atom](#) (with the asciidoc and asciidoc preview plugins) or [Brackets](#) (with the asciidoc preview plugin) can be used to edit the content locally. AsciidocFx being our favorite so far. You can then commit the documentation changes like any other piece of code and do a Pull Request for it.

### iii. Inserting UML Diagrams

The preferred way to insert UML diagram is using [PlantUML](#) files. This ensures the original diagram is tracked under version control on Github, since it is a plain text file. This way, anytime a diagram needs to be updated, the Github file may be changed.

A sample sequence diagram may be found at [JAIN SIP Sequence diagram](#).

There are cloud services available to view PlantUML diagrams. Use the [PlantUML form](#) to submit your diagram, and check how it would be rendered. Another useful service at [Gravizo](#) will allow you to integrate your diagram in existing documents. Of course you may try the [local setup](#), or any of the existing IDE integration plugins ( [Eclipse](#), [Netbeans](#), [IntelliJ IDEA](#)).

Again, find a sample generated diagram on Github at [JAIN SIP Project](#) site.

To insert an embedded diagram in AsciiDoc you need to enable the [AsciiDoc Diagram](#) extension. Since we are using Maven to trigger the document generation, you need to configure the [AsciiDoc Maven plugin](#) to include the diagram extension dependency, and include the extension by adding a “requires” configuration. This will only work if you add dependency to “asciidoctorj-diagram” version “1.5.0” to the plugin dependencies. Check sample pom file at [DiagramEnalbedPomFile](#)

You may insert the diagram directly/embeddedCode or indirectly/encodedURL. Indirect access should be preferred, as we can version control the diagram independently from referenced document, and we prevent duplications in case we want to insert the diagram in different documents. Another advantage, is that the document will be “automatically” updated when you change the target diagram (that is, on next document generation the same URL will deliver the new diagram). For indirect access, you can use the Plantuml server with this URL “<http://www.plantuml.com/plantuml/proxy?src=<urlToYourPlantUmlFile>>” . This web service will answer with a PNG image file, that may be inserted in HTML docs, or any URL image enabled doc. For example [Indirect Access Diagram](#) . Next picture shows how to insert the image in this document itself by using indirect access.



---

The documentation on [http://documentation.telestax.com/core/sip\\_servlets/SIP\\_Servlets\\_Server\\_User\\_Guide.html](http://documentation.telestax.com/core/sip_servlets/SIP_Servlets_Server_User_Guide.html) is updated everyday through a Continuous Integration job hosted on Cloudbees at <https://mobicents.ci.cloudbees.com/job/RestComm-Documentation/>. It fetches the latest documentation from every project, build it, theme it (by copying the latest theme updates from <https://github.com/RestComm/documentation>) and then commit it and push it to <https://github.com/RestComm/restcomm.github.io> which allows the changes to be visible publicly at <http://documentation.telestax.com>.

## vii. Process

Each feature and sometimes even bug fixes requires that the contributor writes documentation for the feature.

Contributor design documentation is usually done in [Design](#) and can be written in AsciiDoc too and people can comment on it through github. A Google document can be created too to be reviewed and commented upon during the review and then later on migrated to AsciiDoc (<https://chrome.google.com/webstore/detail/asciidoc-processor/eghlmnhjjbjodpeehjjcgfcjegcfbkh>)

User Facing Documentation is currently hosted at <http://documentation.telestax.com>.

User Facing Documentation follows the following process:

1. Everyone (public and team members) should open an issue for missing docs. This also goes for questions that are asked in forums or on Zendesk. This also includes flagging old and outdated information.
2. Developers/support team members should write a draft
3. Doc team will test and polish
4. Continuous Documentation will build and theme the documentation

## 6. Committing code

It's important to reference the Issue number created in [Everything starts with an issue](#) as Github automatically link that commit to the relevant issue. That allows for one issue to have the track of all relevant commits associated to it.

---

It's highly helpful when coming back to it later if an issue is detected or when backporting a particular issue to a different branch. (Important for Productization of the code base). See <https://github.com/blog/957-introducing-issue-mentions> and <https://help.github.com/articles/closing-issues-via-commit-messages/>.

As we use the excellent [Waffle.io](https://waffle.io) for visualization and maintaining of the current roadmaps for our open source projects (by example <https://waffle.io/RestComm/RestComm-Connect>), it is recommended to follow their naming convention and workflow at <https://github.com/waffleio/waffle.io/wiki/Recommended-Workflow-Using-Pull-Requests-&-Automatic-Work-Tracking>.

## 7. Pushing changes to your online clone

When a change is ready to be integrated back into the repository, that change is pushed from the developer's local clone to their Github Fork clone.

To avoid merge soup, please always rebase your branch first.

If the main repository has evolved since your last push to your clone repository, you will need to bring those changes into your repository as well as potentially merge them.

**Note:** Please see [Git Useful Commands](#) if you're not familiar on how this works in Git

## 7. Pull Request

First pull in all of the latest changes from upstream, apply them to your master branch, then rebase your feature branch against master before merging it into master and pushing it upstream. (**Note:** Please see [Git Useful Commands](#) if you're not familiar on how this works in Git)

Go to your online clone repo and navigate to the page with details on the branch to be reviewed. By example, [https://github.com/your\\_github\\_account/RestComm-Connect/tree/development](https://github.com/your_github_account/RestComm-Connect/tree/development) and click the green Pull Request button.

## 8. Pull Request Review

When the pull request is done, the project lead maintainer will receive a notification and the code will be reviewed.

If any further changes are suggested, a couple of iterations might be needed so the contributor will need to modify the code again, commit, push and comment on the issue.

The project lead maintainer may run a [Continuous Integration](#) on the specific branch to ensure it doesn't break any of the existing code based.

**The project lead maintainer should never accept a Pull Request that don't meet the process above i.e., Design Documents, Good code with Comments, Non Regression Tests and Documentation**

Once the change is approved, a committer of Restcomm will merge it back into the main repository (**Note:** Please see [Git Useful Commands](#) if you're not familiar on how this works in Git)

Even though this may sound complicated, this process makes code reviews easier and allows a lot of people to work on changes in parallel without breaking the stable master.

---

## Q&A Process

### 1. Continuous Integration

RestComm projects use the popular [CloudBees](#) service for all their Continuous Integration.

For those unfamiliar with Continuous Integration, it is recommended to read [https://en.wikipedia.org/wiki/Continuous\\_integration](https://en.wikipedia.org/wiki/Continuous_integration).

RestComm Continuous Integration runs for every project regularly when code is committed to the master branch. It ensures the code compiles correctly and all the automated tests are run and there is no regression of the code base then it proceeds to the next stage of releasing the software through the [Continuous Delivery](#)

### 2. Continuous Delivery

RestComm projects use the popular [CloudBees](#) service for all their Continuous Delivery. Each job, after running the tests, will release the maven artifacts to the Sonatype [Maven Repository](#) <https://oss.sonatype.org/content/repositories/releases/org/mobicents/> which is sync-ed to Maven Central and also creates a binary file of the project (if applicable) as well as tag the github repo with the corresponding version so that when issues are reported it's easy to find out the exact code corresponding base as opposed to our previous way of using SNAPSHOTs maven artifacts. This also guarantees any build is fully reproducible and can be rebuilt from scratch.

---

## Release Process

Since we are using Continuous Delivery, every build on the Continuous Integration services Cloudbees is potentially a release that can be used to formally create a stable release after the necessary smoke manual testing has been completed.

### 1. Creating a release on github

Download the latest binaries zip files from your Continuous Integration/Delivery job locally and go to [https://github.com/RestComm/<project\\_name>/releases/new](https://github.com/RestComm/<project_name>/releases/new) to create a new release. Specify the Release Title and Description and upload the binary zip files you just downloaded. You need to also select the relevant tag in Tag Version dropdown list which have been previously created automatically by the CI/CD job from which you downloaded the binary zip files. When done, click “Publish Release” button.

### 2. Make a Release Announcement

Please draft a release announcement on TeleStax website similar to <http://telestax.com/restcomm-media-server-4-1-0-released/>

### 3. Market your release

Post the release announcement on the Google Groups for the community <https://groups.google.com/forum/#!forum/restcomm> and <https://groups.google.com/forum/#!forum/mobicents-public>.

Post the release announcement on social networks : Twitter, LinkedIn, Facebook, Google+ or coordinate with [marketing@telestax.com](mailto:marketing@telestax.com).

You can also post the release announcement on related news website if relevant <http://www.theserverside.com/>, <https://dzone.com/>, <https://groups.google.com/forum/#!forum/discuss-webrtc>

## Hiring Process

Even though, TeleStax has a [traditional jobs board](#), for any of those jobs to be awarded the candidate has to go through the same process that everyone in the company went through (even the TeleStax co-founders), i.e., contribute to the community in one way or another.

---

Again contributions take various forms, it's not only about code contributions. Let's break it down into a couple examples :

- R&D Position : Obviously this one is one of the most technical one, requires design, testing, documentation, code contributions, community support on the [Google Group](#) or [StackOverflow](#) and spent a sensible amount of time participating.
- L1 Support Position: This one requires mainly to provide community support for usability type of questions. Pointing the users to the right piece of documentation mainly
- L2 Support Position: This one requires to provide the same as L1 Support + some contributions in terms of simple or more elaborate code fixes, documentation and testing as well.
- Solution Architect Position: This one requires similar contributions as L2 Support Position and an ability to see the bigger picture and help on roadmap ideas, and new feature design discussions.
- Management/Sales/Marketing Positions: This one requires that the candidate creates a RestComm Application through the RestComm Visual Designer or through the RestComm API that will improve existing processes or could be relevant in the market. Those applications would ultimately be available on the [RestComm Application Store](#). We and the community provide a number of [applications ideas](#) in the open to be developed that can potentially be sold to the operators we are working with. They can also contribute blog posts on [TeleStax blog post](#) as guests.

This is actually the best hiring process I ever stumbled upon as this contribution period allows both sides to understand if they like to/are comfortable:

- Working with the team - people first
- In a distributed manner - home office.
- Across so many timezones
- And so many languages and cultures - English is the common language.
- Understand our open source platform value proposition and model
- The development and contribution processes
- Be recognized by their peers
- Understand if they are a good fit technically (for engineering and support positions)
- Compete on a position in the open - democracy wins.
- Be hyper productive from day 1 once they have the job

So don't wait, ping us and start contributing ;)

---

## Git Useful Commands

### 1. Git Cloning & Branching

The contributor then creates a new branch into their local clone

```
git checkout -b feature-branch
```

### 2. Git Committing

Do the changes into their branch for their local branch for the contribution and commit them

```
git commit -a -m "commit message"
```

**IMPORTANT:** Please use the Github integration to use the commit message to tie the commits to the Issue you're working on. More information on that can be found at <https://help.github.com/articles/closing-issues-via-commit-messages>

**IMPORTANT:** When your change is pulled into the main RestComm source, the change description that you entered here will show up as changes in the main RestComm source, so please use a meaningful description - fixing bug, making changes, etc. are not ok, please instead use something like fixing transform bug caused by NPE, etc. so that it makes sense in the context of RestComm as a whole, not just your clone.

If you have any new files, make sure to use the following command before committing

```
git add <file or directory>
```

Same thing if you want to remove some files

```
git rm <file or directory>
```

### 3. Pushing changes to your online clone

When a change is ready to be integrated back into the repository, that change is pushed from the developer's local clone to their Github Fork clone.

```
git push origin feature-branch
```

To avoid merge soup, please always rebase your branch first.

If the main repository has evolved since your last push to your clone repository, you will need to bring those changes into your repository as well as potentially merge them.

---

You need to add a remote via which you will identify the upstream repository:

```
git remote add upstream
git@github.com:RestComm/RestComm-Connect.git
```

Now whenever you want to merge upstream changes into your clone, do the following:

```
git fetch upstream
git merge upstream/master
```

## 4. Pull Request

First pull in all of the latest changes from upstream, apply them to your master branch, then rebase your feature branch against master before merging it into master and pushing it upstream:

```
git checkout master
git fetch upstream
git merge upstream/master
git checkout awesome-feature
git rebase master
(fix any conflicts with upstream changes)
git push origin feature-branch
```

## 5. Pull Request Review

Once the Pull Request is approved, a committer of RestComm will merge it back into the main repository with the following commands.

```
git checkout -b feature-branch
git pull
https://github.com/<contributor_github_account>/RestComm-Connect/feature-branch
git checkout master
git merge feature-branch
```