## **Introduction**

This is a Proposal for a Simultaneous Release Plan for the release of TF. Please read it carefully to understand the requirements to participate and the requirements for each milestone.

## **Definitions**

### **APIs**

* **APIs:** Application Programming Interface. For the purposes of TF an API is any form of interface that is exposed to other TF apps/services and/or exposed to third party services or external applications.
* **API Classifications:**For the purposes of being declared **Stable**, **Provisional**, or **Tentative**, an API is a collection of code that provides some high-level functionality, e.g., flow programming or data store access. When listed on a release plan, APIs should be given a short name, classified into one of the three categories, and have the supporting bundles (if/when they exist) listed as well.
	+ **Stable API:** An API that can be accessed external to your project, existed in a previous version of TF, and will continue to exist in the current version of TF with no changes. By definition, all Stable APIs are frozen throughout this entire release cycle. Note that all APIs are assumed to be **Stable APIs** unless called out as **Provisional** and/or **Tentative** in a release plan.
	+ **Provisional API:** An API that can be accessed external to your project and is introduced in the current release, or an externally accessible API that existed in a previous version of TF but is being modified for the current release.
	+ **Tentative API:** A **Provisional API** that will be provided in a best effort manner, but which may or may not be delivered in the final release. The Go/NoGo status of Tentative APIs must be made by M2.

### **Milestones**

* **Functionality Freeze (M2):** *(used to be called feature freeze)* No new externally visible functionality is to be added to the current release of TF. All provisional APIs are at least functional (at a beta-quality level) if not yet fully tested.
* **API Freeze (M3):** All externally accessible APIs (Stable and Provisional) may not be modified. An API exception process will allow for critical changes to APIs after API freeze with the consent of the consuming projects.
* **Code Freeze (M4):** No new features/functionality are to be allowed into the current release. Only errors/bugs identified in the bug system are allowed. The exceptions to this include new tests, and documentation. Distribution packaging must be complete. Errors/bugs found after Code Freeze are still bugs and they may be created and worked on. This includes packaging bugs found as well.
* **String Freeze (M4):** All text strings used within TF may not be changed. Final documentation and localization teams may rely on these strings not changing for the current release.
* **Release Candidate (RC):** A fully-built, complete version of the current TF release. This includes all code, documentation, and packaging that make up the final user-deliverable set of artifacts. After RC0, new RCs will be continually built, e.g., once per day, to enable rapid testing and fixing of bugs.
	+ Note that this definition makes the dates for RCs and the final release as targets, but they may need to be adjusted based on project readiness and any remaining blocking bugs.
	+ While we will build daily release candidates, the notion of RC#s (increasing in number at a longer cadence, e.g., weekly) will remain to aid in planning when bug fixes are expected.
	+ During the RC process blocking bugs will be tracked both on a spreadsheet and in JIRA.
	+ During the RC process regular, e.g., daily, meetings may be held on IRC to identify and address critical issues as they arise.
	+ Projects not in the autorelease and the distribution feature index by RC0 cutoff will be dropped from the release.

### **Features**

* **Feature:** A logical grouping of code and functionality in a project. A Feature is usually any component or grouping.
* **Top-Level Feature:** A Feature that provides one of the major pieces of functionality delivered by a project. In general, this should not require an understanding of the project's internals to know when/how to install it. Most projects will have a small number of, maybe even only one, Top-Level Features. In many cases the Top-Level Feature will actually only be a meta-feature grouping together lower-level features, which would be less obvious for an outsider to consume.
* **User-Facing Feature:** A Top-Level Feature that somebody looking to install and run TF should know about. They should be able to install the Feature and should be able to tell that it's been installed in the form of new user interface elements, support for new southbound devices, or other mechanisms.

#### **Stable and Extended Features (optional)**

* **Stable Feature:** A Top-Level Feature that meets the following criteria. The TSC is expected to review each potentially stable feature during the project's release review and ensure that it meets these requirements. **Note:** Any of these requirements can be waived by the TSC for a given feature. However, the TSC is encouraged to find a way to make the test fit the situation before granting a waiver.
	+ The feature must lie within the declared scope of the project.
	+ The feature is part of a Mature project (as defined in the project lifecycle).
	+ The feature was present in the previous release.
	+ The feature is only dependent on other stable features (or sub-components of other stable features).
	+ The feature provides adequate documentation.
	+ The feature has 75% or higher test coverage.
	+ The feature must have at least one automated test for each of:
		- *functionality* (previously called a system test) to show the basic functionality works
		- *cluster compatibility* to show the feature works in a 3-node cluster using the clustered data store
		- *scalability* to show how large a system, e.g., number of hosts, switches, or links, the feature can handle
		- *performance* to show how many operations, e.g., transactions, flows, linkstate events, per second the feature can handle
		- *longevity/stability* to show the feature can run for a period of time under load without ill effect
		- In each case, the tests must show no unexplained regressions from previous releases.
	+ The feature is backward compatible with the previous release of the feature, e.g., any APIs that were not deprecated in the previous release still exist with the same signatures.
	+ The feature has no known vulnerabilities that are older than a week and classified as important by the security response team or [high by their CVSS score in a CVE](https://nvd.nist.gov/cvss.cfm). If a fix for such a vulnerability lies outside of TF, the TSC may choose to relax the requirement on a case-by-case basis.
	+ The feature commits to providing a migration strategy from the previous release. This will ideally take the form of scripts or automatic upgrade support, but could also come in the form of documentation.
* **Extended Feature:** A top-level feature that is a part of the release and does not meet the Stable Feature criteria.

### **Release Distributions**

* **Stable Distribution:** A distribution containing the collection of Stable Features as they are compiled into the Stable Release Feature Repository hosted in the Integration project.
* **Extended Distribution:** A distribution containing the collection of both Stable and Extended Features as they are compiled in the Extended Release Feature Repository hosted in the Integration project.

### **Project Offsets**

Projects are classified into one of 3 offsets:

* **offset zero:** deadlines are at the dates listed
* **offset one:** deadlines are *generally* at the listed dates + 1 week
* **offset two:** deadlines are *generally* at the listed dates + 2 weeks
	+ *Note that the intent is that offset two projects have no other projects depending on them in this release*

This is intentionally flattening the actual dependency graph

* The idea is to hit an 80/20 point where projects can have some lag to get new APIs from those they depend on
	+ If projects are in the same offset but need APIs from each other this should be noted and planned (possibly by asking for them sooner than would be required) as part of the API request negotiation at M1

The intent is for projects that form key infrastructure of all other projects (e.g., controller) to operate at **offset zero**, projects which provide key network services to operate at **offset one**, and projects that others don't depend on to operate at **offset two**.

Deadlines for **Release Candidates** (RC0, RC1 and RC2) and the release are the same regardless of offset. Deadlines for M1 through M4 are offset by +1 week and +2 weeks.

## **Requirements for Participation**

In order to participate in the simultaneous release, a project must do the following things.

1. Planning
	* Projects must declare their intent to participate in the Simultaneous Release by M0. This is accomplished by sending the first milestone readout mail and adding the project to the table on the wiki.
	* Participating projects must publish a candidate Release Plan shortly after M0, and declare their final Release Plan by M1.
		+ Participating project Release Plans must contain Milestones that minimally line up with the Simultaneous Release Plan Milestones
		+ Release plans should contain a complete list of the exposed APIs including the properties defined above, e.g., the name of the interface, a short name, and the list of supporting bundles.
		+ Per-project release plans now include sections for cross-project negotiation of provided APIs and for noting cross-project incompatibilities.
			1. Projects are required to negotiate cross-project dependencies for any new or modified APIs.
			2. Projects are encouraged to think about any cross-project incompatibilities and how to resolve them, if possible, as part of their release plans.
2. Leadership & Communication
	* Each project must elect a Project Lead as described in the TSC charter.
		+ Phil Robb or Casey Cain will help projects with this process and it must be completed by M0.
		+ The results of the election, and other changes to the project lead during this release, should be reported by
			1. Updating the project facts for the project on its main wiki page
			2. Updating the participating projects table of this release
			3. Sending an e-mail to the tsc list
	* The Project Lead is expected to be responsible for the the project meeting deadlines, interacting with other projects, and interacting with the TSC
	* The Project Lead will be subscribed to the release mailing list and must respond to requests sent to the project in a timely fashion—defined as **two business days**.
		+ If Project Leads are not be able to do so, they should (i) have somebody else stand in and do this on their behalf, (ii) send a mail to the release mailing list indicating this and the time period, and (iii) note the same information in the participating projects section of the release plan.
	* The project lead is expected to, at a minimum, read the release mailing list, read the TSC meeting minutes. The project lead is strongly encouraged to attend these meetings if at all possible and some representative from the project is expected to attend each IRC meeting if at all possible.
	* In addition to the Project Lead, each project must designate a Test Contact and Documentation Contact to handle test related communication.
	* All release-critical correspondence that requires a response will have a subject line containing "PLEASE RESPOND BY <DATE> <UTC-TIME>" or "URGENT RESPONSE REQUIRED/NEEDED"
		+ Please limit traffic to correspondence directly relating to the release
		+ The TSC collects response time metrics for projects both to inform our planning and to measure project maturity going forward.
3. Service Release Participation
	* All projects participating in the release are also required to participate in the stability releases described in the schedule after the formal release.
4. Quality
	* No later than M1, each project must verify that the project builds and passes test for each new patch pushed to gerrit.
	* No later than M1 as part of the participating projects must push their binary artifacts to the repository
	* No later than M1, each project must have a Jenkins Job which rebuilds and retests to an appropriate level when a project it depends on publishes new artifacts, i.e., a Jenkins "-integration" job.
	* No later than M1, each project primarily written in Java must be reporting unit and/or integration test coverage via sonar. (See instructions on reporting test code coverage)
		+ Projects, especially ones that form key infrastructure for other projects, are strongly encouraged to set goals for code coverage and reported bugs. Doing so will be seen favorably when evaluating projects for advancement in the Project Lifecycle.
		+ Stable Features must have appropriate unit and/or integration test coverage of at least 75% prior to M4.
5. Testing
	* In addition to setting up appropriate -verify, -merge, and -integration jobs by M1, projects are expected to provide adequate unit, integration and system tests.
		+ Stable Features must have established integration and system tests as required for Mature project Stable Features.
	* The coverage provided by unit tests and integration tests should be reported by M1. (See instructions on reporting test code coverage)
	* Participating projects must describe a basic system test per top-level feature and a comprehensive system test including functionality, cluster, scalability, performance, longevity/stability per stable feature prior on M2.
	* Participating projects must run at least one basic automated system test for each top-level feature and several automated system tests including functionality, cluster, scalability, performance, longevity/stability for each stable feature by M4.
		+ Note: The system test requirements can be waived by the TSC for a given feature if for example the top-level feature is tested through another top-level feature.
		+ Note: Projects running system test outside TF (external Lab) are required to report system test results in a timely fashion after release creations, e.g., weekly, RC, and formal releases.
	* System tests are expected to reliably pass. If a system test turns out to be unstable and intermittently fails, it must be fixed or disabled. If intermittent system tests are seen as having value to the project, they can be written and run on-demand by the project, but won't be run as part of the automated CSIT suite.
	* Further details and requirements can be found in the schedule and project integration and test requirements below.
6. Documentation
	* Each participating project is expected to identify the kinds of documentation that would be useful (e.g., installation guide, user guide, developer guide) and provide them as part of the release.
	* More details on the expectations can be found in the schedule and project documentation requirements below.
7. Code Hygiene
	* No uses of System.out.println in non-test code.
	* No dependencies on 3rd party (non-TF) snapshot versions
	* Willing to use agreed-upon versions for dependencies (both 3rd-party and TF), i.e., to eliminate version skew
	* Willing to find source code for 3rd-party dependencies and/or move to versions or alternatives for which source code can be found.
8. Distribution
	* All projects must support a distribution model including defining features and adding them to integration repository no later than M2.
	* No later than M2, each project must have a “distribution-check" Jenkins Job to verify changes in the code do not break integration distribution.
9. Meeting Deadlines
	* All projects are expected to meet the deadlines laid out in the schedule below.
		+ To indicate this, the project lead/contact is expected to provide and send a milestone readout to the release mailing list by 23:59:59 UTC on the date listed for the the appropriate offset at each milestone.
		+ Most information will be communicated by filling out appropriate information in the Tracking Spreadsheet, but a mail should still be sent indicating that the information has been filled in. Any other information or questions can be included in that mail.
	* If a project cannot make a deadline, the project lead/contact must write a summary of what was missed, why, the course correction that will be taken, and its impact on other projects.
		+ For **offset two** project this is mainly intended to be reflective and to help inform the release process.
		+ For **offset zero** and **offset one** projects, this should be completed within 24 hours of missing the deadline and must be presented to the TSC at the first TSC meeting after the deadline.
	* All Milestone deliverables will be verified by the release management staff and/or the TSC.
		+ *NOTE: For deliverables defined only in the project's release plan—and not as a requirement in this document—the release management staff and/or TSC will verify that the status of the deliverables has been reported. release management staff and/or the TSC may also, but are not required to, verify the delivered functionality.*

###  **Project Documentation Requirements**

This section is excerpted from Project Documentation Requirements, to see it in its full context, go to that page.

#### **Kinds of Documentation**

These are the expected kinds of documentation and target audiences for each kind.

* **User/Operator:** for people looking to use the feature w/o writing code
	+ Should include an overview of the project/feature
	+ Should include description of available configuration options and what they do
* **Developer:** for people looking to use the feature in code w/o modifying it
	+ Should include API documentation,
* **Contributor:** for people looking to extend or modify the feature's source code
	+ *Note: should be documented on the wiki not in reStructuredText*
* **Installation:** for people looking for instructions to install the feature after they have downloaded the TF release
	+ *Note: audience is the same as User/Operator docs*
	+ For most projects, this will be just a list of top-level features and options
		- We'd also like them to note if the options should be checkboxes (i.e., they can each be turned on/off independently) or a drop down (i.e., at most one can be selected)
		- What other top-level features in the release are incompatible with each feature
		- This will likely be presented as a table in the documentation and the data will likely also be consumed by automated installers/configurators/downloaders
	+ For some projects, there is extra installation instructions (for external components) and/or configuration
		- In that case, there will be a (sub)section in the documentation describing this process.
* **HowTo/Tutorial:** walk throughs and examples that are not general-purpose documentation
	+ Generally, these should be done as a (sub)section of either user/operator or developer documentation.
	+ If they are especially long or complex, they may belong on their own
* **Release Notes:**
	+ Release notes are required as part of each project’s release review. They must also be translated into reStructuredText for inclusion in the formal documentation.

#### **Requirements for projects**

Projects MUST do the following

* Provide documentation including
	+ Developer documentation for every feature
		- Most projects will want to logically nest the documentation for individual features under a single project-wide chapter or section
		- This can be provided as a single .rst file or multiple .rst files if the features fall into different groups
		- This should start with ~300 word overview of the project and include references to any automatically-generated API documentation as well as more general developer information (as described above).
	+ User/Operator documentation for every every user-facing feature (if any)
		- *Note: This should be per-feature, not per-project. Users shouldn't have to know which project a feature came from.*
		- Intimately related features, can be documented as one noting the differences
	+ Installation documentation
		- Most projects will simply provide a list of user-facing features and options.
	+ Release Notes (both on the wiki and reStructuredText ) as part of the release review.
* This documentation will be contributed to the docs repo (or possibly imported from the project's own repo with tooling that is under development)
	+ Projects MAY be ENCOURAGED to instead provide this from their own repository if the tooling is developed
	+ Projects choosing to meet the requirement this way MUST provide a patch to docs repo to import the project's documentation
* Projects MUST cooperate with the documentation group on edits and enhancements to documentation
	+ Note that the documentation team will also be creating (or asking projects to create) small groups of 2-4 projects that will peer review each other's documentation. Patches which have seen a few cycles of peer review will be prioritized for review and merge by the documentation team.

#### **Timeline for Deliverables from Projects**

* **M2:** Documentation Started
	+ Identified the kinds of documentation that will be provided and for what features
		1. Release Notes are not required until release reviews at **RC2**
	+ Created the appropriate .rst files in the docs repository (or their own repository if the tooling is available)
	+ Have an outline for the expected documentation in those .rst files including the relevant (sub)sections and a sentence or two explaining what will go there
		1. Obviusly, providing actual documentation in the (sub)sections is encouraged and meets this requirement
	+ Milestone readout should include
		1. the list of kinds of documentation
		2. the list of corresponding .rst files and their location, e.g., repo and path
		3. the list of commits creating those .rst files
		4. the current word counts of those .rst files
* **M3:** Documentation Continues
	+ The readout at M3 should include the word counts of all .rst files with links to commits
	+ The goal is to have draft documentation complete so that the documentation group can comment on it.
* **M4:** Documentation Complete
	+ All (sub)sections in all .rst files have complete, readable, usable content.
	+ Ideally, there should have been some interaction with the documentation group about any suggested edits and enhancements
* **RC2:** Release notes
	+ Projects must provide release notes as .rst pushed to the master branch of the docs project

## **Milestones, Release Candidates, and Service Releases**

* Milestones are spaced roughly 4 weeks apart taking into account significant holidays.
* Release Candidates (RC) are spaced roughly 1 week apart
* Service Releases are roughly 4, 12, 20, and 30 weeks after the Formal Release and are intended to continue at least until the after the next formal release of the TF, presumably .

### **Schedule Framework**

This Simultaneous Release plan has been drafted based on the Schedule Framework

### **Schedule**

2The deadline to meet and report the results of each milestone is at 23:59:59 UTC on the listed day. That corresponds to 4p or 5p pacific time.

Official release template is: Release Plan Template.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Milestone** | **Offset 0 Date** | **Offset 1 Date** | **Offset 2 Date** | **Events** |
| M0 | Date | N/A | N/A |  **Simultaneous Release Open**1. Contact Freeze
	* Projects must have declared intent to participate in Simultaneous Release
	* Projects must have elected their Project Leads and specify a Test Contact
	* Participating Projects must have published a candidate Release Plan for public comment ( Release Plan Template )
* *Note: the date for M0 will normally be at least one day after the TSC approves the release plan.*
* *Note that the release plan includes details about negotiating inter-project dependencies, expectations, and incompatibilities.*
 |
| Last call for project proposals |  |  |  | 1. This is the latest date a project proposal can be sent to the project-proposals list and still have the required two week public comment period before its project creation review at the last TSC meeting before the M1/M2/M3 milestone. Project proposals submitted after this date will not be able to become formal projects by M1/M2/M3 and thus will not be able to participate in the release.3
 |
| M1 |  |  |  | 1. Participating Projects must have declared their final Release Plan with all sections fully completed.
2. Projects that need extra configuration or resources other than those available in the TF CI infrastructure must have opened helpdesk tickets to add them.
3. Project Checklist completed (for *all* projects, not just new ones).
4. Projects may apply for a system test waiver if they think they have top-level features not requiring system test or covered by other top-level features test.
5. Projects must specify whether they plan to use TF CI infrastructure for system test. It is recommended to use the TF CI infrastructure unless there is some HW or SW resource that cannot be installed there. Projects running system test in external Labs are required to report system test results in a timely fashion after release creations, e.g., weekly, RC, and formal releases.
6. Project must get acknowledged from all projects that it depends on.
 |
| M2 |  |  |  | 1. Feature/Functionality Freeze
	* Final list of externally consumable APIs defined and documented
		+ Projects must state for each TENTATIVE API they have (if any) whether they are formally planning to deliver it.
			- If so, it should be noted that it will be delivered.
			- If not projects requesting the API must be informed so that they can take corrective actions.
		+ Externally consumable APIs are available at beta-quality
	* All inter-project dependencies are resolved (all project functionality is declared as either "In" or "Out" of this release)
2. Features defined
	* Each feature should be tested in every appropriate jenkins job (at least -verify, -merge, and -integration)
	* Any feature repositories containing features intended for release must be added to the main features.xml file in the integration git repository as explained in the step-by-step guide
		+ Projects must have a distribution job to verify changes in code do not impact the integration distribution (this will be automatically setup by the releng/builder group).
	* Features that are intended to be "top-level", "user-facing" and/or "stable" must be called out in the milestone readout. These features will have additional requirements:
		+ Each "top-level" feature must have a developer guide section and a system test Each "user-facing" feature must have a user guide section Each "stable" feature must meet the requirements explained in the definitions section above.
	* Changing the name of a feature or removing a feature should be handled via an API freeze waiver after this point
3. Documentation Started
	* Identified the kinds of documentation to be provided, created docs files for them with outlines, and committed those files in an appropriate location.
4. Feature Test Started
	* Instructions can be found in the System Test Step by Step Guide (TBD).
	* Projects must have filled out a basic system test plan template for each top-level feature. Stable features have additional requirements for functionality, cluster, scalability, performance, longevity/stability.
 |
| M3 |  |  |  | 1. API Freeze: See more information in the definition above.
2. Documentation:
	* Project readouts MUST include a word count of each relevant doc file with a goal of draft documentation done.
3. Feature Test Continues
	* Participating projects Projects must have all extra SW configuration and resources required for system test installed in the TF CI.
 |
| M4 |  |  |  | 1. Code Freeze (bug fixes only from here as defined above)
2. Stability branch, i.e., stable/, must be cut and local project versions bumped on master to avoid overwriting SNAPSHOTS
	* *Note:* Branch cutting will occur sometime between offset 0 M4 and offset 2 M4 and may be either staggered by offsets or done all at once.
3. String Freeze (all externally visible strings frozen to allow for translation & documentation)
4. Documentation Complete: Only editing and and enhancing should take place after this point.
5. Feature Test Complete
	* Stable features should fulfill quality requirements listed in definitions section
	* Projects must run at least one basic automated system test job for each top-level feature and several automated system test jobs including functionality, cluster, scalability, performance, longevity/stability for each stable feature.
 |
| RC0 |  |  |  | 1. The build for RC0 will start at 23:59:59 UTC
	* At the start of the build for RC0, all projects must be in the distribution and autorelease.
		+ Between M4 for offset 2 projects and RC0 is a two week period for projects to finish adding to Integration Distribution and Autorelease and for projects to fix any errors in the Autorelease Build Job. At the beginning of this two week period, projects are given two week notice of potential drop. Projects that have not been successfully added to the Integration Distribution and Autorelease are dropped from the release. At the end of this two week period, we release RC0 for projects to begin their initial testing. At this time, all projects participating in the release must be in the distribution and autorelease.
2. During the RC process, regular, e.g., daily, meetings may take place to identify and address issues
3. During the RC process, blocking bugs will be tracked in JIRA and a common spreadsheet
 |
| RC1 |  |  |  | 1. The build for RC1 will start at 23:59:59 UTC
	* At the start of the build for RC1, all stable/ branches will be locked and only release engineering staff will be able to merge patches.
2. During the RC process, regular, e.g., daily, meetings may take place to identify and address issues
3. During the RC process, blocking bugs will be tracked in JIRA and a common spreadsheet
 |
| RC2 |  |  |  | 1. The build for RC2 will start at 23:59:59 UTC
	* At the start of the build for RC2, the release engineering staff will only merge patches that fix blocking bugs. All stable/ branches will remain locked and only release engineering staff will be able to merge patches and will only do so for patches that fix blocking bugs.
2. During the RC process, regular, e.g., daily, meetings may take place to identify and address issues
3. During the RC process, blocking bugs will be tracked in JIRA and a common spreadsheet
 |
| RC3 |  |  |  | 1. Participating Projects must hold their Release Reviews, including User Facing Documentation.
	* The build for RC3 will start at 23:59:59 UTC
2. All stable/ branches will remain locked and only release engineering staff will be able to merge patches and will only do so for patches that fix blocking bugs.
3. During the RC process, regular, e.g., daily, meetings may take place to identify and address issues
4. During the RC process, blocking bugs will be tracked in JIRA and a common spreadsheet
 |
| Formal Release |  |  |  | 1. Formal Release
	* *NOTE: The build to produce the formal release artifacts is likely to occur before the formal release date.*
2. After the release, except for projects that have opted-out, the release engineering staff will apply the release patch to the stable/ branch and bump versions.
	* *Note:* Any patches merged to stable/ after the auto-release build that produces the formal release artifacts, but before the release patch and version bumps are applied will have to be reverted and re-applied after the release and version bump patches. This shouldn't happen in as the stable/ branches will have been locked since RC1.
 |
| SR1 (Service Release 1 aka .1) |  |  |  | 1. First Service Release for . NOTE: This date is provisional, but will not move earlier. Please note, event based Updates (security/critical bugs) are distinct and may occur at any point.
	* To allow time for testing, a release candidate will be built before the service release and projects are expected to not merge patches except for blocking bugs between that time and the actual service release.
	* Blocking bugs will be tracked via JIRA and a spreadsheet.
2. After the release, projects MUST apply the release patch to the stable/ branch and bump versions. Unless a project opts out, this will be done automatically by the release team after the release.
	* *Note:* Any patches merged to stable/ after the auto-release build that produces the formal release artifacts, but before the release patch and version bumps are applied will have to be reverted and re-applied after the release and version bump patches.
 |
| SR2 (Service Release 2 aka .2) |  |  |  | 1. Second Service Release for . NOTE: This date is provisional, but will not move earlier. Please note, event based Updates (security/critical bugs) are distinct and may occur at any point.
	* To allow time for testing, a release candidate will be built before the service release and projects are expected to not merge patches except for blocking bugs between that time and the actual service release.
	* Blocking bugs will be tracked via JIRA and a spreadsheet.
2. After the release, projects MUST apply the release patch to the stable/ branch and bump versions. Unless a project opts out, this will be done automatically by the release team after the release.
	* *Note:* Any patches merged to stable/ after the auto-release build that produces the formal release artifacts, but before the release patch and version bumps are applied will have to be reverted and re-applied after the release and version bump patches.
 |
| SR3 (Service Release 3 aka .3) |  |  |  | 1. Third Service Release for . NOTE: This date is provisional, but will not move earlier. Please note, event based Updates (security/critical bugs) are distinct and may occur at any point.
	* To allow time for testing, a release candidate will be built before the service release and projects are expected to not merge patches except for blocking bugs between that time and the actual service release.
	* Blocking bugs will be tracked via JIRA and a spreadsheet.
2. After the release, projects MUST apply the release patch to the stable/ branch and bump versions. Unless a project opts out, this will be done automatically by the release team after the release.
	* *Note:* Any patches merged to stable/ after the auto-release build that produces the formal release artifacts, but before the release patch and version bumps are applied will have to be reverted and re-applied after the release and version bump patches.
 |
| SR4 (Service Release 4 aka .4) |  |  |  | 1. Fourth Service Release for . NOTE: This date is provisional, but will not move earlier. Please note, event based Updates (security/critical bugs) are distinct and may occur at any point.
	* To allow time for testing, a release candidate will be built before the service release and projects are expected to not merge patches except for blocking bugs between that time and the actual service release.
	* Blocking bugs will be tracked via JIRA and a spreadsheet.
2. After the release, projects MUST apply the release patch to the stable/ branch and bump versions. Unless a project opts out, this will be done automatically by the release team after the release.
	* *Note:* Any patches merged to stable/ after the auto-release build that produces the formal release artifacts, but before the release patch and version bumps are applied will have to be reverted and re-applied after the release and version bump patches.
 |

Please note that the TSC reserves the right to allow projects to enter the Simultaneous Release for a reasonable period of time after the M0 date. For example, the TSC may allow additional time if a project is delayed by the IPR Review process.

Projects running system tests outside the TF CI infrastructure are not required to run system tests and report the results on "-merge" and "-integration" Jenkins jobs, although if they can this is ideal. They are required to report system test results in a timely fashion after release creations, e.g., weekly, RC, and formal releases.

Please also note that projects that would like to spin out parts of themselves into additional projects may have those new projects join the Simultaneous Release at any point prior to M3 provided:

1. The TSC has been informed of this intent prior to M2
2. The original project's release Release Plan is apportioned between the original and new projects with no parts missing
3. The new projects have been proposed and approved by the TSC into one of the non-proposed life-cycle states in the normal manner by M2
4. The new projects have completed the requirements for all milestones before they joined the release, e.g., M0 and/or M1

Lastly, note that as the new projects are joining the release prior to M2, they must meet all the requirements for M2 at the normal time.