# 1 - Introduction

## 1.1 - Overview

This document gives the SA3LI working procedures for using the Forge in conjunction with the 3GPP CR process.

## 1.2 - Aims

SA3LI uses the Forge to meet several key objectives agreed by group

1. Raise the quality of accepted CRs by providing a consistent mechanism for validating and syntax checking formal-language components of CRs before they are accepted
2. Avoid implementation errors during the publication process by providing an automated means of merging formal-language changes
3. Provide a developer-friendly way of obtaining SA3LI's formal language deliverables

## 1.3 - Approach

CR authors are asked to submit their changes as Merge Requests to the Forge (in addition to the regular 3GPP process), as part of the normal meeting process. This allows the aims to be met in the following ways

* Merge Requests can be subject to automated testing, including syntax-checking and compilation, ensuring that CRs can be proven to compile before being accepted by the meeting
* Merge Requests can be automatically checked against each-other to highlight potential conflicts, allowing these to be resolved before the close of the meeting
* The Forge provides a set of comprehensive, industry-standard tools for analysing the changes proposed, making comprehensive review by technical delegates much easier.
* Merge Requests provide a way for CRs to be implemented automatically within the formal-language parts of the spec, reducing the scope for error (while still allowing the output to be subject to the normal 3GPP review and publication process).
* Developers can easily clone, copy and compare formal-language artefacts stored on the Forge using the same industry-standard tooling that many use in regular development (git).

# 2 - Repositories

## 2.1 - Repository structure

SA3-LI uses a single repository for its formal-language deliverables. There are two instances of the repository

* A production repository at <https://forge.3gpp.org/rep/sa3/li>
* A trial repository at <https://forge.3gpp.org/rep/sa3/li-trial>

*Note – only the trial repository is in use. Work will be moved to the production repository when ready.*

## 2.2 - Directory structure

The repository has the following directory structure:

* A subdirectory for each specification with formal-language schemas
	+ A subdirectory for each published release.
		- An ASN.1 or XSD file for each formal-language module or schema in that release of that schema
* A "testing" subdirectory for shared test fixtures used for automated checking.

In this way, the repository structure reflects the state of the most-recently published specifications for each given release.

|  |  |
| --- | --- |
| **EXAMPLE** | The following diagram illustrates the directory structure. For brevity, only one directory has been fully expanded. (Repository root) ├─ 33128 │ ├─ r17 │ │ ├─ TS33128Payloads.asn │ │ ├─ TS33128IdentityAssociation.asn │ │ ├─ TS33128IdentityAssociation.asn │ │ ├─ urn\_3GPP\_ns\_li\_3GPPIdentityExtension.xsd │ │ ├─ urn\_3GPP\_ns\_li\_3GPPStateTransfer.xsd │ │ └─ urn\_3GPP\_ns\_li\_3GPPX1Extensions.xsd │ ├─ r16 │ └─ r15 ├─ 33108 └─ testing |

## 2.3 - Branch configuration

The current state of published 3GPP SA3LI deliverables, across all releases, is represented by the "main" branch.

Other branches are created and merged during the CR process (see clause 3).

## 2.4 - Repository roles

The Forge CR process involves the following roles:

* **Maintainer** – Responsible for creating, managing and removing meeting branches, accepting CRs in the Forge once agreed by the meeting, and resolving post-meeting issues

(3GPP Technical Officer, ETSI Forge staff, or delegated authority, as directed by SA3LI chair)

* **CR Author** – Responsible for creating, submitting and updated CRs to the Forge.

(any delegate permitted to contribute CRs to the meetings)

## 2.5 - Repository configuration

Annex A contains more detailed information on how the repositories are configured, including user permissions and default branch behaviours.

# 3 - CR process

## 3.1 - Preparing for a meeting

The Maintainer creates a meeting branch that represents the baseline for a plenary meeting at which a given set CRs will be agreed. If there is more than one SA3LI meeting in a given plenary cycle, then the same meeting branch will be used.

The branch name follows the branching convention given in clause 4.2:

meeting/[*meeting title*]/

|  |  |
| --- | --- |
| **EXAMPLE** | CRs considered at **SA3#82-LI-e-a** (2021-07-12) will be considered at the next plenary meeting, which is **SA3#93e** (2021-09-14).Before **SA3#82-LI-e-a** the Maintainer therefore creates a branch called meeting/SA93eCRs considered at **SA3#82-LI-e-b** (2021-09-01) will also be considered at **SA3#93e**. The Maintainer does not need to create an additional meeting branch. |

## 3.2 - Drafting a CR

The CR Author reserves a CR on the 3GPP portal, following the existing 3GPP process. This assigns a CR number to the CR.

The CR Author creates a branch specific to their CR, following the branching convention given in clause 4.2.

cr/[*deliverable*]/[*CR number*]

This branch is created using the appropriate meeting branch (rather than main) as a baseline.

The CR Author immediately creates a Merge Request (MR) for the new branch, targeting the same meeting branch. The CR Author gives the MR a title following the naming convention in clause 4.1.

[*deliverable*] CR [*CR number*] – [*CR title*]

The CR Author (and any collaborating parties) commits the desired changes to their branch

Changes made to the CR branch are visible to anyone with access to the ETSI Forge. Delegates may comment on and, if appropriate, contribute to, the changes in the CR branch.

|  |  |
| --- | --- |
| **EXAMPLE** | A CR Author wishes to contribute a CR to TS 33.128 for **SA3#82-LI-e-a**. They reserve a document on the portal, and are assigned CR number **987**. The CR will be considered at the next plenary meeting, **SA3#93e**.Given this information, the CR Author creates the following:**A new CR branch*** Name: cr/33128/0987
* Baseline: meeting/SA93e

**A new MR**, * Source branch: cr/33128/0987
* Target branch: meeting/SA93e
* Titled "TS 33.128 CR0987 – My proposed change"
 |

## 3.3 – Submitting a CR to a meeting

The CR Author should ensure that the current commit passes any automated syntax and drafting rule checks before submitting the CR for consideration by the meeting. CRs which do not pass automated checking may be accepted at the discretion of the Chair (see clause 3.4).

When ready for submission, the CR contributor prepares a CR form as per the existing 3GPP CR process. The CR author places a link to both the MR and the latest commit in their CR branch in the "Other comments" section of the CR form.

The CR author includes the changes to the formal language schema as part of the CR form, using change-marked text as per the existing 3GPP process. CR authors are strongly encouraged to generate this change-marked text from Forge tooling, rather than attempting to keep the CR Form and Forge MR aligned manually.

## 3.4 – Consideration of the CR at the meeting

The CR Author presents both the CR form and the associated changes in the 3GPP Forge to the meeting.

If the CR Author decides to revise the contribution, they makes any necessary changes by making additional commits to the CR branch. Draft CR forms may be created following the procedures in clause 3.3, and the uploaded to the drafts folder following the existing 3GPP process.

CRs should pass automated checking before being accepted by the meeting, with exceptions being made at the Chair's discretion. Special attention should be given to CRs which fail the automated merge test check. This indicates that two or more CRs conflict with each other. In some cases this is unavoidable (e.g. if two CRs both add parameters to the same structure), but CR Authors should ensure that any unavoidable merge conflicts are resolved prior to agreement.

## 3.5 – Post-meeting treatment of agreed CRs

Once the final SA3LI meeting in a given plenary cycle is finished, the Maintainer accepts the MR for each agreed CR, with the following settings:

* Source branch is **not** deleted.
* Source commits are squashed.

The Maintainer then resolves any merge conflicts, seeking assistance from CR authors where necessary.

The Maintainer then performs the following post-meeting steps:

* Object Identifiers for any ASN.1 modules that contain changes are incremented
* XSD version tags in XSD namespaces of any XSD schemas that contain changes are incremented
* ASN.1 Structures with new or changes tags are checked to ensure tag numbers are sequential (unless otherwise agreed).
* Tabs are replaced with spaces
* Indentation and alignment errors are corrected

The meeting branch now contains a corrected set of CRs that can be submitted to SA plenary, and be used to generate a draft version of the specification.

## 3.6 – Post-plenary procedures

Once the changes are confirmed by SA and the new version of the spec is published, the Maintainer merges the agreed meeting branch into the relevant release branch with the following settings:

* Source branch is deleted.
* Source commits are **not** squashed.

*NOTE – this is the opposite to the settings used in accepting the CRs.*

The Maintainer tags the head of the main branch following the tagging convention given in clause 4.

* A tag for each deliverable agreed for publication at the plenary meeting
* A tag indicating the output of the plenary meeting

These tags make it easy to retrieve or compare specific versions of a specification, or the output of specific meetings.

The Maintainer removes the meeting branch and any remaining CR branches.

|  |  |
| --- | --- |
| **EXAMPLE** | After **SA3#93e**, new versions of **TS 33.128** are agreed for publication. A Release 16 version is agreed at **16.7.0**, and a Release 17 version is agreed at **17.2.0**.After merging the meeting branch back into main, the Maintainer creates the following tags for the head commit* A tag for the meeting output named output/SA93e
* A tag for the newly-published TS 33.128 r17 spec, named spec/33128/17.2.0
* A tag for the meeting output named spec/33128/16.7.0
 |

# 4 – Conventions

## 4.1 – Rendering convention

|  |  |  |
| --- | --- | --- |
| **Element** | **Description** | **Example** |
| *meeting title* | Name of a meeting, given with only alphanumeric characters (i.e. all hyphens and hashes removed) | SA93eSA382LIea |
| *deliverable* | Name of a published deliverable, containing only the numeric characters (i.e. 'TS', spaces and periods omitted) | 3312833108 |
| *CR number* | A CR number, given as a four digit number padded with zeroes | 0987 |
| *version* | A version of a given deliverable, including release number, given as dotted decimals | 17.1.0 |

## 4.2 – Branching convention

|  |  |  |
| --- | --- | --- |
| **Branch Name** | **Description** | **Example** |
| main | Current published deliverables | main |
| meeting/{*meeting title*} | Branch used as a baseline for CRs being considered at a given SA plenary meeting, and an intermediate merge target for CRs which are agreed. | meeting/SA93e |
| cr/{*deliverable*}/{*CR number*} | Branch representing a CR | cr/33128/0987 |

## 4.3 – Tagging convention

|  |  |  |
| --- | --- | --- |
| **Tag name** | **Description** | **Example** |
| spec/{*deliverable*}/{*version*} | Tags a published version of a given deliverable | spec/33128/17.1.0 |
| output/{*meeting title*} | Tags the output of a given meeting | output/SA93e |

# Annex A – Repository configuration

We propose each repository is configured as follows:

## Permissions

Delegates with a Forge account and who request access to the repository are given Developer access.

Developers may fill the CR Author role, and do the following

* Create new public branches
* Push commits to unprotected branches
* Create merge requests
* Contribute to the Issues and Wiki areas

Only Maintainers are permitted to the following:

* Accept Merge Requests on protected branches (meetings and releases)
* Create tags

## Merge Requests

The repository Merge Method is set to “Merge commit with semi-linear history”. This only allows merges when fast-forward merging is possible, but allows the user to rebase automatically via the GUI.

Merge Requests are allowed to be merged even if the CI/CD pipeline fails. This allows CRs that fail the pipeline to still be agreed, although this should only be done by exception.

## Branches

The default branch is set to main

main and meeting branches are protected.

## CI/CD

A CI/CD pipeline is created to automatically check all commits for ASN.1 or XSD syntax errrors and contraventions of the drafting rules.

# Annex B - FAQ

1. Why do you have a single repository, rather than one per deliverable?

Repositories incur a maintenance overhead for ETSI Forge staff and 3GPP delegates involved in maintaining them. Having a single repository minimises this overhead, and makes it very easy to share test fixtures between deliverables as required.

1. Why are you still using the Trial repository and not the production one?

We are still in the process of exposing the Forge to new delegates, and vice versa. This means we are regularly encountering new edge cases and issues. Since the production repository only allows ETSI staff to make changes, it makes more sense to do this learning in the Trial repository, where technically-minded delegates can help out by fixing things during a meeting.

1. Why do you have directories for each release, rather than a branch for each release?

Either would work, but the directory structure is much easier to understand for delegates who are not familiar with git - which is almost all of them.

For developers, who often have to work against multiple releases concurrently, it also makes it easy to access multiple releases simultanesouly, without having to clone the repository multiple times.

1. Why bother creating a meeting branch? Surely it will, by definition, be the same as main?

In most cases it will, but it provides a useful baseline if changes are made to main after a plenary cycle starts. More importantly, it provides a consistent merge target for MR/CRs being considered in the same plenary cycle. This is used by the automated testing processes to work out which CRs are being considered together, in order to perform speculative merging to spot merge conflicts before the CRs are agreed.

1. Can a 3GPP Technical Officer really be expected to do all these things?

Probably not. We expect that technical delegates will be asked to assist the Technical Officer in performing some of these tasks.

1. Why do create an MR before you've pushed any commits to the CR branch?

Gitlab MRs carry more information than git branches, such as source and destination branches, pipeline statuses and labels. These can be used via the Gitlab API to provide tooling to assist CR Authors in their work, and by the automated testing to do things like detection of merge conflicts.

The Gitlab web interface for MRs also provides some useful tools, such as comment threads and a simplified diff view, making it easier for delegates to understand what a given CR is proposing to change.

We ask that the MR is created at the beginning of the change request so that this information is available for the whole life of the CR.

1. Why do you squash commits for CR/MRs but not for the meeting branch?

CR/MR commits are squashed to make the commit history manageable. However, when merging down after a meeting it is useful to retain traceability to a particular CR (e.g. in git blame).

1. Why do you retain the source brances for CR/MRs but not for the meeting branch?

SA plenary could decide to reject a particular CR. Retaining the original branches up to this point makes it a little easier to unpick things if this happens. Once SA plenary have agreed things, then all the intermediate branches (both meetings and CRs) can be removed safely.

1. How do I get a version of a specification which isn't the latest in a release?

This is what tags are for.

<https://forge.3gpp.org/rep/sa3/li-trial/-/tags>

Let's assume you want to look at the deliverables in TS 33.128 v16.1.0. You won't find it if you browse the main branch of the repository, because the 33128/r16 directory contains the latest published version in each release – and v1.6.1.0 isn't the latest R16 version of TS 33.128.

Instead, you use the fact that each published version is tagged in the repository, in just the same way that software versions are tagged in normal development repositories. The tagging convention (see clause 4.3) tells us that the correct tag is spec/33128/16.1.0. You can use the Forge UI or git command line tool to view, compare and download the repository as it was at that point in time.