EDGEX FOUNDARY PROJECT AND ISSUE MANAGEMENT STANARD

Revision 0.54

Friday, January 17, 2019

AUTHORS:

- Bryon Nevis, Security Architect - Security WG, Intel Corporation
- Lisa Rashidi-ranjbar, Release Czar - Devops WG, Intel Corporation

CONTRIBUTORS:

- Michael Hall, Tony Espy, Ian Anderson, Leonard Goodell, Diana Atanasova,

Table of Contents

Figures

Tables

Background

As activity in the EdgeX Foundry project has increased, it has been increasingly difficult for interested stakeholders to comprehend the progress that has been made towards a particular release and the work that it still outstanding. The Linux Foundation does not supply a standard set of project management tools that could be used for this work and a free-to-use for open-source commercial tool has not been identified. Consequently, the project has decided to settle on—for now—the free tools that come with GitHub. Unfortunately, the tools that come with GitHub provide only mechanism and no policy:

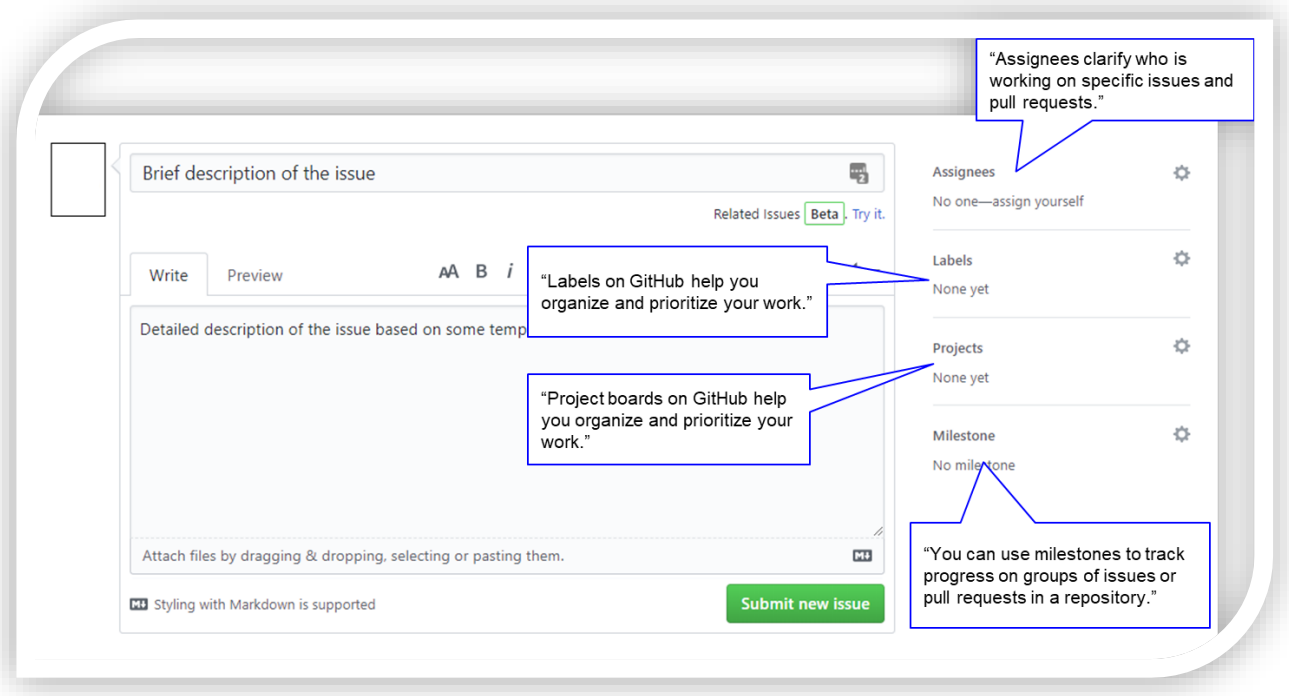


FIGURE 1: GITHUB NEW ISSUE FORM AND GITHUB-SUPPLIED DOCUMENTATION

This document proposes a project and issue management standard for use within EdgeX community. It is written from a data quality perspective: actionable data is high-quality data. High-quality data demands documentation of the business process around the data and providing clear definitions as to what the data means and how it is used. High-quality data requires that participants opt-in to these definitions and take an active role in ensuring that the data in the system is managed in compliance with the standard. It will then be possible to build reports on top of the underlying data to provide an accurate representation of the state of the project at any given time.

Issue Management Standard (GitHub)

In order to achieve the high data quality needed for reporting and project management purposes, the project must attach meaning to the various fields and the participants must be disciplined in their issue management processes. For example, here is the new issue form (Figure 1) with more meaningful descriptions:

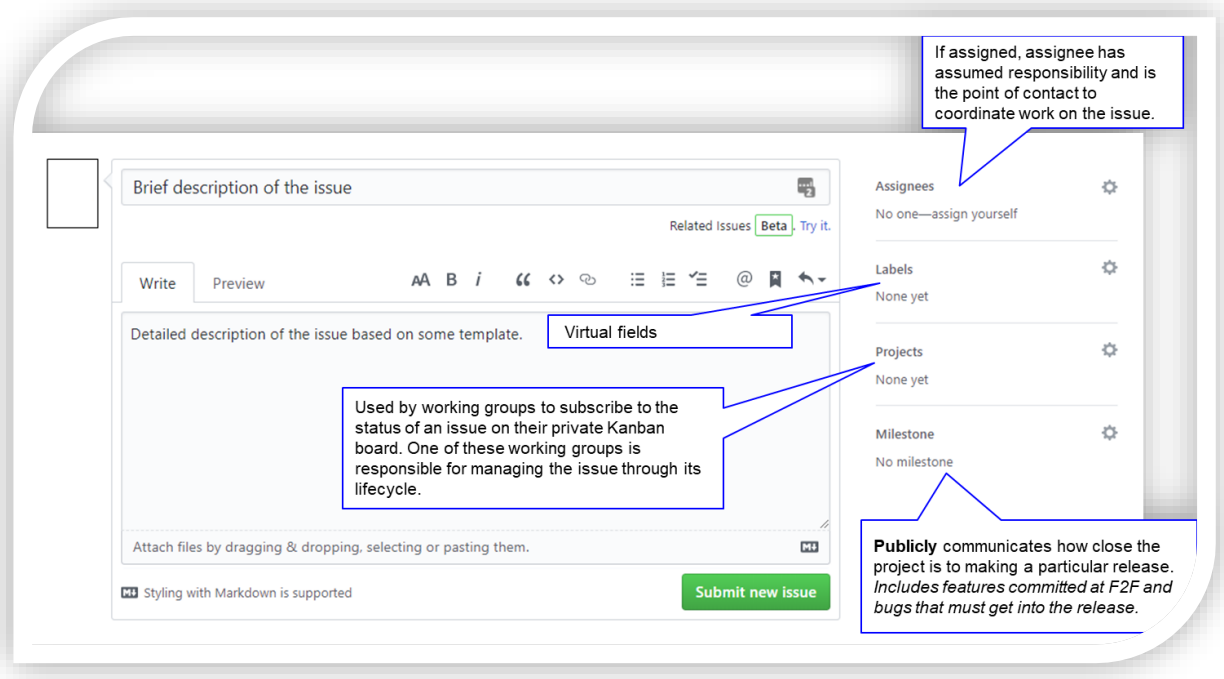


FIGURE 2: GITHUB NEW ISSUE FORM WITH MORE MEANINGFUL DATA DEFINITIONS

GitHub Issue and Pull Request Field Definitions

Definition: Issue is an umbrella term that represents lifecycle-managed textual discussion about the GitHub-hosted software and documentation. An issue can be used to discuss planned work, software defects, questions about the product, or other textual discussion. Issues have fields and metadata associated with them that can be used for filtering or searching, and have lifecycle information where the lifecycle of an issue terminates in a "closed" state. Issues are not code.

Definition: Pull Request contains all the fields of an Issue, but additionally a reference to code that is being proposed for delivery to the Git repository.

Issue Core Fields Definitions

The fields in GitHub issues and pull requests mean the following in the EdgeX Foundry repositories:

Field	Definition
author	An audit field that is automatically set to the creator of the issue or PR.
title	A brief summary of the contents of the issue or pull request Business rules: • Required always
description	The first comment box after the title is the description. It provides the long description regarding what the issue or pull request is about. For pull requests, the description contains the body of the commit message and additional metadata used by GitHub for automation purposes, such as "Fixes #" tags. Business rules: • Required always
comments	Every comment box after description forms a running log of the community discussion of the issue or pull request. Diagnostic information from automation or bots may appear in comments as well.
assignees	If assigned, the assignee has assumed responsibility and is the point of contact for coordinating work on the issue. The assignee must be a project member. For pull requests, this field has no special meaning: if set, it is usually set to the author. Business rules: • *Required* for all in-progress issues • *Recommended* for all release backlog issues (apply **help wanted** label for all unassigned backlog)
reviewers	(Pull requests only.) List of project members who are solicited to review the pull request.
labels	Labels are used to implement virtual fields. See "Virtual Field Definitions" below.
projects	Projects are used by EdgeX working groups to subscribe to the status of an issue on their project board. One of these working groups is responsible for managing the Issue through its lifecycle. An Issue may be on multiple boards simultaneously.
milestone	All milestones in EdgeX refer to releases. The milestone field is used to include the issue in this release's backlog. It indicates a commitment to resolve the issue by the named milestone. (For retroactive milestones, it would be used to flag the issue as "wontfix".) Milestone is a single-select field: only one milestone can be selected at a time.

TABLE 1: ISSUE AND PULL REQUEST FIELD DEFINITIONS

Virtual Field Definitions

The "labels" field is used to implement virtual fields. All virtual fields are **optional** unless otherwise specified. These are defined as follows:

Virtual Field	Definition
issue_type	Indicates the type of issue (single-select): • **bug** — A reactive change or broken functionality • **ci** — Has to do with testing, building, packaging, et cetera. • **enhancement** — A planned feature or change. • **question** — A question. • **request for comments** — Means "we have a proposal for a big architectural feature and we want some public feedback on it." • **security_audit** — The issue is about a security problem with the project that is okay to be publicly known. (There is a private vulnerability reporting process for confidential security bugs.) • **tech-debt** — The issue is about refactoring existing code to improve its design or to remove a previously-added workaround. Business rules: • Required always.
close_reason	Explains why an issue was closed (without a fix) (single-select): • **duplicate** — Is substantially the same as another issue. • **invalid** — Something is substantially wrong with the issue such that it can't move forward (for example, a bug report filed against a feature that is provided by a third-party). • **wontfix** — Used to document that the issue is valid but that not resources will be put into fixing the issue. Often used to indicate that a bug won't be fixed on a previous release. Business rules: • Required for all issues closed without a fix. • A comment explaining why the Issue was dispositioned is required.
subsystem_affected	Indicates the scope of the proposed change (multiple-select): • **app-services** • **core-services** • **device-services** • **documentation** • **export-services** • **security-services** • **snap** — Cross-cutting label for any Issue that requires changes to the snap packaging of EdgeX

	• **support-services** • **system-management**
release_affected	A multiple-select field used for bugs only that indicates a release of EdgeX which is affected by the issue. Note, there may be multiple release_affected labels if the issue exists in more than one EdgeX release. For feature development work, the milestone field is the official field used for release planning. (This changes the current practice so there will be a transition period where both fields are used.) If multiple release_affected tags exist and the issue is only fixed in one of these releases resulting in the issue being closed, if any of affected releases is either an active development release or an LTS release, then a new issue should be opened with release_affected set accordingly. This allows the issue to continue to be tracked in the other affected releases where it makes sense to do so. Values: • **california** • **california dot - 0.6.1** • **delhi** • **delhi dot - 0.7.1** • **edinburgh** • **fuji** • **geneva** • **geneva-f2f** — An additional tag that that signifies that the issue belongs to a committed roadmap item at the Geneva planning face-to-face. • **hanoi** • **hanoi-f2f** — An additional tag that that signifies that the issue belongs to a committed roadmap item at the Hanoi planning face-to-face. • **unscoped** — Indicates that the work is not targeted for any release. <u>Business rules:</u> • Required for all bug type issues requiring a code change <u>Exceptions:</u> • Special procedures are required (see below) for issues that affect multiple releases.
exception	Indicates an exceptional condition outside of the normal workflow (multiple-select): • **hold** — Used on pull requests only. Indicates that the pull request should not be merged despite successful pre-commit tests and approvals. Contact the PR author for additional details. • **good first issue** — Used to advertise an issue that is easy to approach and easy to finish and would be appropriate for a new contributor to the project to tackle.

	• **help wanted** — Used to advertise the issue as something the project wants done, but the project doesn't have anyone assigned to work on it or does not have the required expertise to do it well.
priority	Priority is used to influence where to put resources and what features to cut when a release window is closing (single-select). The permissible values for priority and suggested guidance around their usage are: • **3-high** — Indicates a release-blocking issue • **2-medium** — Indicates cross-cutting project impact • **1-low** — Used for isolated changes

TABLE 2: VIRTUAL FIELD DEFINITIONS

Status / State Machine

There is no "status" field in GitHub. Issues are either "open" or "closed". However, issues may be assigned to project Kanban boards and can be moved from column to column in order to provide advanced status reporting. The current column of an issue is represented below as "status." Issues may freely move from state to state without restriction. Moreover, each working group may have its own customized state machine. The table presented below is the project-recommended values. Unless otherwise specified, issues must be manually moved from state to state.

Status	Meaning
new issue	Default state for incoming issues when assigned a project. New issues require triage into either "icebox" or "release backlog" status.
icebox	There are no current plans to work on the issue in the near-term.
release backlog	The issue is planned for the upcoming release or be retroactively fixed for a previous release.
in progress	Someone has started working on the issue
QA/Code Review	A state reserved for pull requests that have not been approved. GitHub automation automatically places new pull requests here.
Done	Terminal state for all issues and pull requests. GitHub automation will move issues and pull requests here automatically when they are closed.

TABLE 3: PROJECT-RECOMMENDED ISSUE STATUS VALUES

Figure 3 below shows the idealized state machine for issues. This is the recommended flow but working groups may customize this flow to one more suitable for their unique requirements.

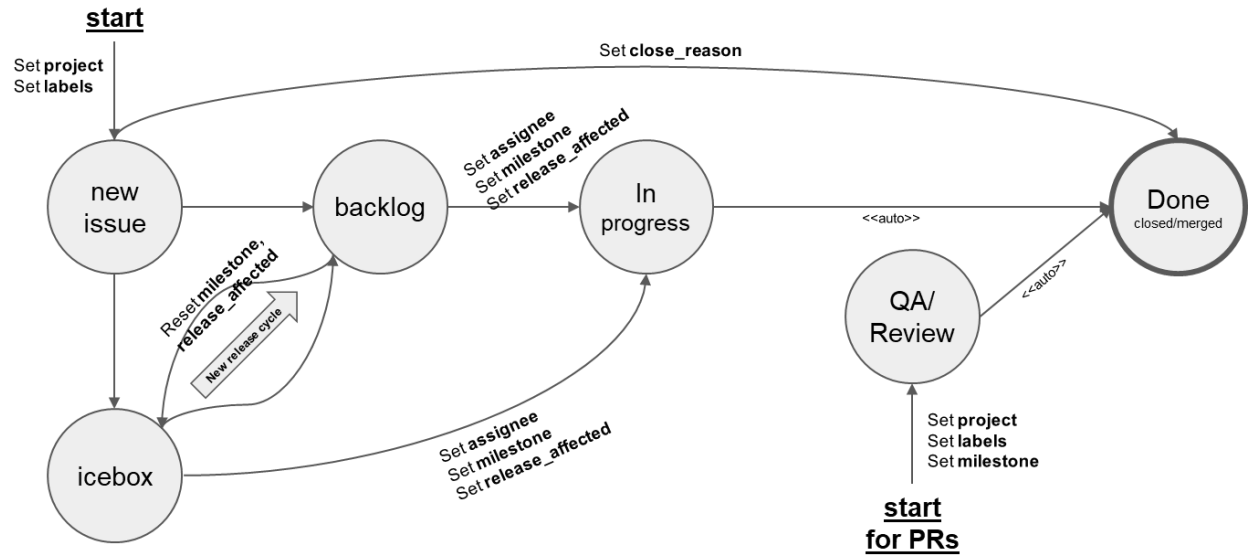


FIGURE 3: IDEALIZED STATE MACHINE

Bugs affecting more than one release

When a bug affects more than one release, multiple issues shall be filed, one against each affected release. (Typically, this will be against the current supported release and release-in-progress.)

The following issue fields should be set:

- title = (as desired) + "in <release-name>"
- issue_type = bug
- priority = (as desired)
- release_affected = <current release>, <next release>
- milestone = <release-name>
- project = (depends)

If the bug is not going to be fixed in a particular release, the **close_reason** should be set to **wontfix** and an appropriate comment made as to the reason the bug was so dispositioned. Bugs that will be fixed should go through the normal bug handing process.

Release stabilization procedures

Release stabilization is the period of time between code freeze (when a git branch named after the release is made) and when the official EdgeX release is declared (container images and snaps published). During release stabilization the following procedures apply:

- Every PR must have an associated bug (against the code-freeze branch).
- A second bug must be filed against the next release for fixing the issue in the **master** branch.

The procedures for "Bugs affecting more than one release" should be used for the associated bugs.

Project management

The project management system draws on the high-quality data of the issue management system. The project management system is designed to provide high-level decision support for release activities, and to provide visibility to interested stakeholders on the state of the project.

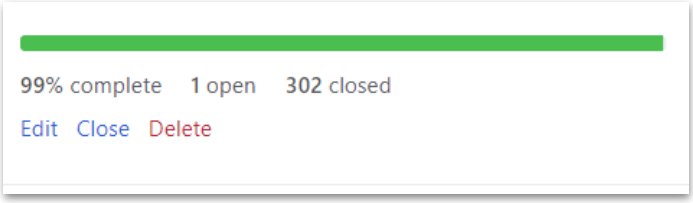
There are two primary mechanisms used for this:

- GitHub milestones — mostly for public consumption
- Project (Kanban) boards — the primary mechanism used by EdgeX project members

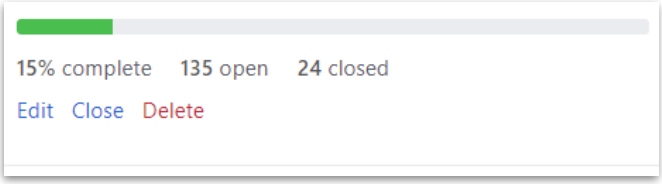
GitHub Milestones

GitHub milestones provide a public view of release progress and can give a coarse approximation as to how close the project is to the next release milestone.

For example, this what the GitHub milestone for Fuji release looked like in the week following the Geneva planning face-to-face.



Contrast this to the GitHub milestone for Geneva at approximately the same time:



One problem with GitHub milestones are that they are tracked on a per-repository basis. Since an EdgeX release is made of code form multiple repositories, looking at a single repository's milestone, such as edgex-go, is insufficient to get the health of the entire project.

Something more is needed.

Project (Kanban) Boards

GitHub provides a facility called a "project" that allows for organizing work. These projects can be created at a per-repository level or at the organization level. The project boards function as a Kanban, which customizable swim lanes and some basic automation for state transitions between swim lanes. In the Issue Management section we talked about how these project board are used to implement a more

granular issue state machine than "open" and "closed." In this section, we describe how these boards are used for project management.

Using Project Boards to inform release decisions

The following project boards have been created at the organization level:

- Systems Management WG
- Security WG
- QA/Test WG
- Device WG
- Core WG
- Certification WG
- Applications WG
- Devops WG

These project boards map 1:1 with the standing EdgeX working groups where contributions to the code base are coordinated.

The overall project state can be ascertained by navigating to each of the above project boards and counting the issues in each column.

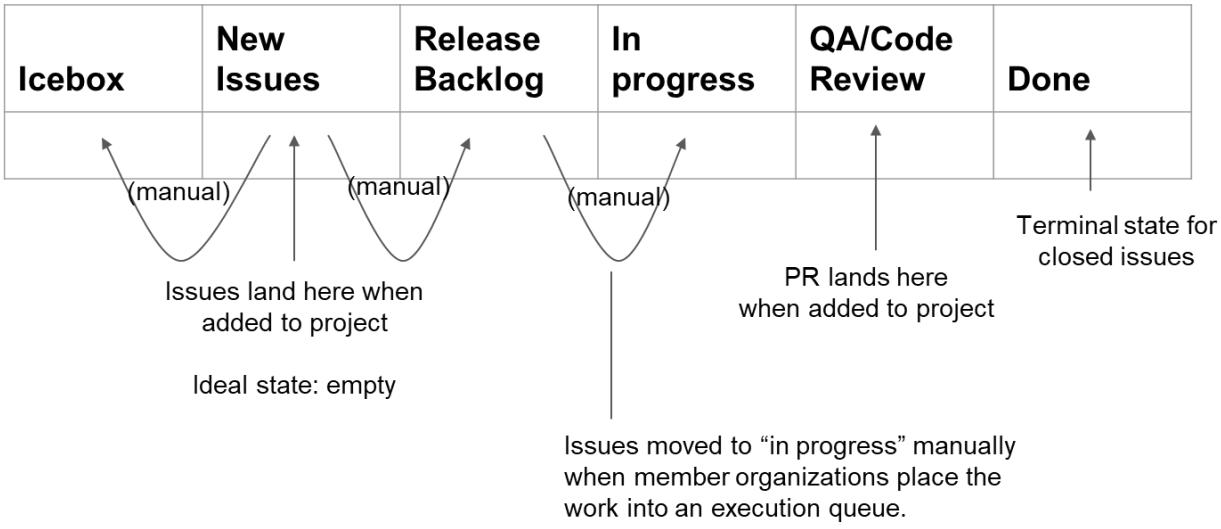


FIGURE 4: PROJECT BOARD STANDARD SWIMLANES

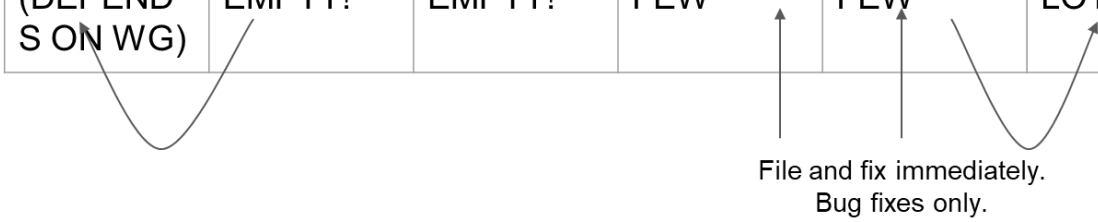
The project is ready for code-freeze when the "release backlog" is empty, and there are very few issues remaining in the "in progress" and "QA/Code review". The project boards should remain in this state until the release is cut.

Project board states at other important project milestones

End of a release cycle

At the time of code freeze, the project boards should resemble the following state. Notably, the backlog should be empty, and the project should only be working on bug fixes for the impending release. The project may stay in this state for several as the entire framework is regressed and bug fixes made.

Icebox	New Issues	Release Backlog	In progress	QA/Code Review	Done
(DEPENDS ON WG)	EMPTY!	EMPTY!	FEW	FEW	LOTS!



After code-freeze, it is technically possible to begin making changes for the next release. This early work should be done by directly picking items out of the "icebox" and putting them into "in progress." It is expected that as the project matures in its release practices, the duration the uncomfortable state between code freeze and release will shorten from several weeks to several days. There is no good way with the current tools to filter the project boards based on release.

Icebox	New Issues	Release Backlog	In progress	QA/Code Review	Done
(DEPENDS ON WG)	EMPTY!	EMPTY!	NEXT REL. ONLY	NEXT REL. ONLY	LOTS!

A new release cycle is started by an announcement that the official release has been made.

Beginning of a release cycle

At the beginning of a release cycle, working groups pull selected backlog from the icebox into the release backlog. This backlog is presumed to have been groomed prior to the planning face-to-face and ratified there, then, when the release branch for the previous release is made, the done items from the

previous release are archived, and the backlog for the next release is pulled into the release backlog column:

From this point forward, issues gradually transition to the "done" state: