Section D ITS Open Source Process

D.1 https://ite-org.github.io/NTCIP-8008/latest/

AASHTO / ITE / NEMA

Table of contents

Section D Front Matter

D.1

Interim for Field Release (IFR)

NTCIP 8008 v1.0.0

National Transportation Communications for ITS Protocol

ITS Open-Source Process

January 6, 2026

D.1 Notices

D.1.1 Copyright

D.1.2 Content and Liability Disclaimer

The information in this publication was considered technically sound by the consensus of persons engaged in the development and approval of the document at the time it was developed. Consensus does not necessarily mean that there is unanimous agreement among every person participating in the development of this document.

AASHTO, ITE, and NEMA standards and guideline publications, of which the document contained herein is one, are developed through a voluntary consensus standards development process. This process brings together volunteers and seeks out the views of persons who have an interest in the topic covered by this publication. While AASHTO, ITE, and NEMA administer the process and establish rules to promote fairness in the development of consensus, they do not write the document and they do not independently test, evaluate, or verify the accuracy or completeness of any information or the soundness of any judgments contained in their standards and guideline publications.

AASHTO, ITE, and NEMA disclaim liability for any personal injury, property, or other damages of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of, application, or reliance on this document. AASHTO, ITE, and NEMA disclaim and make no guaranty or warranty, express or implied, as to the accuracy or completeness of any information published herein, and disclaims and makes no warranty that the information in this document will fulfill any of your particular purposes or needs. AASHTO, ITE, and NEMA do not undertake to guarantee the performance of any individual manufacturer or seller's products or services by virtue of this standard or guide.

In publishing and making this document available, AASHTO, ITE, and NEMA are not undertaking to render professional or other services for or on behalf of any person or entity, nor are AASHTO, ITE, and NEMA undertaking to perform any duty owed by any person or entity to someone else. Anyone using this document should rely on his or her own independent judgment or, as appropriate, seek the advice of a competent professional in determining the exercise of reasonable care in any given circumstances. Information and other standards on the topic covered by this publication may be available from other sources, which the user may wish to consult for additional views or information not covered by this publication.

AASHTO, ITE, and NEMA have no power, nor do they undertake to police or enforce compliance with the contents of this document. AASHTO, ITE, and NEMA do not certify, test, or inspect products, designs, or installations for safety or health purposes. Any certification or other statement of compliance with any health or safety-related information in this document shall not be attributable to AASHTO, ITE, or NEMA and is solely the responsibility of the certifier or maker of the statement.

D.1.3 Trademark Notice

NTCIP is a trademark of AASHTO / ITE / NEMA. All other marks mentioned in this project are the trademarks of their respective owners.

March 20, 2025

D.1 Acknowledgements

This document was prepared though an open-source standards development process with the following active contributors:

Check out the full list of contributors here.

In addition, the following submitted comments during the process:

- k-vaughn

The resultant document is maintained by the NTCIP Base Standards, Profiles and Protocols (BSP2) Working Group (WG), a subdivision of the Joint Committee on the NTCIP. The Joint Committee on the NTCIP is organized under a Memorandum of Understanding among the American Association of State Highway and Transportation Officials (AASHTO), the Institute of Transportation Engineers (ITE), and the National Electrical Manufacturers Association (NEMA). The Joint Committee on the NTCIP consists of six representatives from each of the standards development organizations (SDOs) and provides guidance for NTCIP development.

March 20, 2025

D.1 Foreword

D.1.1 Overview

This document is an NTCIP Open-Source NTCIP Process, Control, and Information Management document provided as Interim for Field Release (IFR).

Open-source documents are developed using the ITS Open-Source Process, as defined in NTCIP 8008. This process provides an open standards development process that accepts issues reported by the community and resolved by peer-reviewed contributions from the community. The open source process concludes with the resultant material being approved by the defined approval process.

IFR documents are approved through a streamlined process focused on the technical experts of the community (e.g., those participating in the open-source development process) rather than through a formal ballot of industry managers.

NTCIP Process, Control, and Information Management documents define the practices and policies used by the NTCIP Joint Committee and its working groups in developing and maintaining NTCIP publications.

This document defines the process for developing projects for the ITS community using an open-source environment (e.g., GitHub). The project can produce any type of product, such as a guide, a technical specification, a test procedure (e.g., including code), etc.

The approval process for the resultant open-source product is based on the target level of specification. For example, an IFR specification undergoes a less formal approval process than a full standard.

D.1.2 Approvals

IFRs are peer reviewed within the open-source process with final approval by an associated WG established by the NTCIP Joint Committee.

Approval information is provided within the online environment.

For more information about NTCIP standards, visit the NTCIP Web Site at www.ntcip.org.

D.1.3 User Comment Instructions

Comments can be submitted at any time. In preparation of this NTCIP standards publication, input of users and other interested parties was sought and evaluated.

Comments on open-source projects can be submitted either on the discussions or issues tab of the project.

Discussions can be initiated at any time and anyone in the community can respond, all within a public environment. Responses to discussion comments are strictly informative and may not be accurate. Discussion comments can lead to the submittal of issues that need to be resolved to clarify the standard.

Issues can be submitted at any time. Issues are triaged by the project maintainer, who will evaluate their merit, classify them (e.g., as a bug, documentation issue, ommission), and in most cases respond to the submitter. Once ready, issues will be available for contributors to volunteer to address. When a volunteer has a proposed solution, it can be submitted to the project and approved in a relatively short period (when compared to the traditional standards approval process). However, updates to the projects are still version controlled so that users can reference a specific version of the project without fear of it changing.

Comments should use the templates provided on the website; otherwise they may be ignored.

D.1.4 History

For a history of the project, see the projects releases page.

July 17, 2025

D.1 Introduction

This site defines the ITS Open-Source Process as used by several projects within the ITS standards community. The process follows general practices within the larger open-source community; however, this document:

- provides a step-by-step overview of the process, so that those unfamiliar with open-source processes can better understand the process and become contributors,
- formalizes the process (e.g., by clearly defining what are requirements), and
- tailors the process (e.g., by defining the preferred tools to be used).

This document contains one normative annex.

The following keywords apply to this document: AASHTO, ITE, NEMA, NTCIP, open-source, process.

This document uses only metric units.

October 30, 2024

Section 4 General

4.1 Scope

This document specifies the process used to produce open-source documents within the field of Intelligent Transportation Systems (ITS).

The process follows general practices within the larger open-source community; however, this document:

- provides a step-by-step overview of the process, so that those unfamiliar with open-source processes can better understand the process and become contributors,
- formalizes the process (e.g., by clearly defining what are requirements), and
- tailors the process (e.g., by defining the preferred tools to be used).

The process to approve the resultant product is defined elsewhere (e.g., NTCIP 8001).

The ITS Open-Source Process is based on the practices defined by open-sauced. However, whereas open-sauced is written as an informative guide and describes how systems can work; this document is written as a specification to define how the ITS Open-Source Process will work. While still providing a discussion of the issues; it highlights the requirements and notable options along the way by stating each in its own paragraph and boldfacing the keywords "shall" and "may" to clearly designate requirements and options. The remaining text provides further guidance and can include additional options that do not necessitate specific numbering.

We recognize that onboarding to a new project can be challenging, especially if you're new to open source development. Be patient, and don't be discouraged by setbacks or mistakes. You'll become more comfortable and confident in your contributions with persistence and practice.

4.2 References

The following documents are referenced by this document. At the time of publication, the editions indicated were valid.

4.2.1 Normative References

Normative references contain provisions that, through reference in this text, constitute provisions of this document. All standards are subject to revision, and parties to agreements

based on this standard are encouraged to investigate the possibility of applying the most recent editions of the standard listed.

- ISO/IEC/IEEE 24765:2017: *Systems and software engineering — Vocabulary, 2017*
- GitHub
- MkDocs
- Materials for MkDocs
- ReqView
- Python

4.2.2 Other References

Other references are included to provide a more complete understanding of this document and its relationship to other documents.

4.2.2.1 Other Resources for Contributors

This document standardizes and tailors certain aspects of the information contained in open-sauced; however, it is not a complete replacement of that material. If you wish to learn more about open-source development, the following materials may be of interest:

- What is open-source?
- Why open-source?
- The Secret Sauce
- Types of Open-Source Contributions
- Open Source Guides
- Introduction to GitHub and Open Source Projects

4.2.2.2 Other Resources for Maintainers

If you wish to learn more about open-source maintenance, the following materials may be of interest:

- Understanding the Role of an Open Source Maintainer
- How to Communicate and Collaborate Effectively
- Building Community
- Maintainer Power Ups
- Building Your Team
- The Power of Open Source Metrics
- Contributor Ladder Template
- Maintainer Community

4.3 General Statements

The remainder of this document is broken into the following chapters:

- **Commenting Process:** Details the process of contributing to open-source projects and provides step-by-step processes for using the preferred tools of the ITS open-source projects.
- **Contribution Process:** Details the process of contributing to open-source projects and provides step-by-step processes for using the preferred tools of the ITS open-source projects.
- **Maintenance Process:** Details the rules that project maintainers are to follow when managing an ITS open-source project. This includes processes for setting up new projects, managing issues and pull requests, maintaining quality, and coordinating with standard development organizations.
- **Approval Process:** Defines the approval stages for ITS open-source projects and the processes required for approval for each stage and subsequent tagging and publication of versions.[1]
- **Documentation Conventions:** Annex B defines the preferred styles, processes, and tools for developing documentation for ITS open-source projects, including projects that are 100% documentation (e.g., the ITS Open-Source Process project).
- **Code Conventions:** Annex C defines the styles, processes, and tools for developing computer code for ITS open-source projects, including Python and ASN.1.
- **Requirements Management:** Defines preferred ways to use requirement management tools to produce content that can be easily integrated into the ITS open-source pojects while providing clear traceability.

4.4 Glossary

For terms not defined here, English words are used in accordance with their definitions by the merriam-webster online dictionary. Electrical and electronic terms not defined in this section or in Webster's New Collegiate Dictionary are used in accordance with their definitions in ISO/IEC/IEEE 24765:2017.

backlog: A backlog is a list of tasks that need to be completed within a project. Typically, these are tasks that are not yet assigned to a developer and are waiting to be worked on. Sometimes, these could be tasks that were open weeks or months ago and are still waiting to be worked on.

branch: A branch is a separate version of the code that's created for development purposes. Branches allow contributors to experiment with changes without affecting the main codebase. When changes are ready to be merged into the main codebase, they're typically submitted as a pull request.

bug: A bug refers to an error, flaw, or defect in code that adversely affects the proper functioning of the software. Open source projects often depend on contributions from the community to identify and rectify these bugs.

clone: Cloning is the process used to copy an existing Git repository into a new local directory. The `git clone` command will create a new local directory for the repository, copy all the contents of the specified repository, create the remote tracked branches, and checkout an initial branch locally. By default, Git clone will create a reference to the remote repository called origin.

code freeze: A code freeze is a period of time where no new code is added to a project. It is often used to prepare for a release and ensure that the code is stable and ready for production.

code review: A code review is when a maintainer or contributor will review the work of another contributor. This is a great way to ensure that the code is high quality and meets the standards of the project.

containerization: Containerization is a way of packaging and running applications. Instead of installing an app directly on your computer, you put it in a container that includes everything it needs to work. This container can then run on your computer alongside other containers. It's a way to organize and run multiple applications on the same machine, making it easier for developers to manage and scale their applications.

continuous integration (CI): Continuous integration (CI) is a development approach in which developers regularly merge code into a shared repository. For each change, an automated build and test process is run to detect errors as quickly as possible.

continuous deployment (CD): Continuous deployment (CD) is often associated with continuous integration (CI) and refers to keeping your application deployable at any point or even automatically releasing to production. CD means that every change which passes the automated tests is deployed to production automatically.

contributor: A contributor is anyone who makes changes, additions, or suggestions to an open source project. Contributors can be developers, designers, writers, testers, or anyone else who helps to make the project better.

core member: A core member is a contributor who has been granted additional privileges or responsibilities within an open source project. Core members are typically trusted contributors who have demonstrated a deep understanding of the project and have made significant contributions to its development.

docs: Docs is an abbreviation for "documentation". It primarily explains how to implement and use a product or an open source project. It also provides information on how to contribute to the project and expectations for contributors. Documentation is often written using Markdown, a lightweight markup language.

fork: A fork is a copy of a repository. When you fork a repository, you create a new copy of the codebase that you can modify and experiment with without affecting the original codebase.

GitHub actions: GitHub Actions are a way to automate tasks within your software development life cycle. GitHub Actions are event-driven, meaning that you can run a series of commands after a specified event has occurred. Examples of GitHub Actions include running tests, deploying to production, and sending notifications.

GitHub discussions: GitHub Discussions are a way to have conversations about your project directly in GitHub. They are a great way to discuss ideas, ask questions, and share knowledge with your community.

issue: An issue is a problem or bug that needs to be addressed in the code. Issues can be created by anyone, and they're often used to keep track of bugs, feature requests, and other tasks that need to be done.

linting: Linting is the process of running a program that will analyze code for potential errors. A popular linting tool used frequently is ESLint. You can setup an action to run ESlint against each pull request that comes in to check for potential errors before it makes it into production.

maintainer: A maintainer is a person or a group of people responsible for maintaining a specific open source project. Maintainers are typically responsible for reviewing and accepting or rejecting contributions from other contributors. They also have the authority to make final decisions about the direction and scope of the project.

markdown: Markdown is a lightweight markup language commonly used for creating formatted text documents. It is widely used for creating documentation and README files in software development due to its simplicity and readability.

merge: Merging is the process of combining changes from one branch into another. When a pull request is accepted and merged, the changes made in the pull request become part of the main codebase.

onboarding: Onboarding documentation helps new team members or collaborators quickly become familiar with a project's structure, goals, and processes.

OSS Projects: OSS stands for "Open Source Software" projects. These are software projects where the source code is made available to the public, allowing anyone to view, use and modify the software.

pull request: A pull request is a request from a contributor to a maintainer for changes made to the code to be pulled into a codebase.

quality assurance: Quality assurance in open source projects involves testing, reviewing, and ensuring the software meets the desired standards. Community members often contribute to testing and reporting issues to improve the software's quality.

release candidate: A release candidate is a beta version of software with the potential to be a final product. It is typically the last version before the final release.

release notes: Release notes are documents that detail changes, enhancements, bug fixes, and new features in each software release. They inform users and stakeholders about what to expect in a new version of the software.

repository: A repository is a central location where code is stored and managed. In open source, repositories are often hosted on platforms like GitHub, GitLab, or Bitbucket. Each repository can contain one or more projects, and contributors can submit changes to the code by making pull requests.

style guide: A style guide is a set of rules and conventions that define the preferred formatting, writing style, and visual elements used in documentation and other content. This helps maintain consistency and clarity across documents, making them easier to read and understand.

versioning: Versioning is the process of assigning either unique version names or numbers to new releases of your project. Some versions are released as "major" versions, while others are released as "minor" versions.

---

[1] TODO: Move the approval process to NTCIP 8001 ↩

July 23, 2025

Section 4 Overview

4.1 General

It might be tempting to think that managing an open-source standard can be achieved by using any general-purpose word processor that supports a collaborative development environment. However, one will quickly discover that these tools lack key features that are required for industry standards. For example, the two biggest issues are:

- Every edit made to a draft standard needs to be reviewed before being incorporated into a final product. General-purpose, collaborative word processors typically allow anyone with edit rights to edit the document without any traceability. If this was used for a draft standard, anyone with editing rights could make a minor change in the text immediately before the document was approved. The minor textual change could easily be missed by most reviewers while impact of a minor textual change could be significant (e.g., changing "shall" to "shall not"). Thus, rather than allowing peers to edit a common document, the process proposed in this document requires contributors to offer alternative text in a format where their proposal can be viewed and compared against the existing draft before it is incorporated into the final text.
- Standards are used for deployments that can remain in operation for decades. It is important that industry professionals can quickly access any prior released version of the standard so that they can understand devices that were previously deployed. General-purpose word processors generally only deal with one version of a document while the GitHub environment recommended by this document allows a complete version control history that allows users to display any archived version of the document.

To overcome these challenges, this document recommends using a `git` environment (in the case of NTCIP, one hosted at GitHub) that leverages markdown and Material for MkDocs as the primary documentation environment.

Managing a project within this open-source environment involves four major activities as described in the following clauses:

1. Establishing the project
2. Processing comments
3. Processing contributions
4. Approving releases

4.2 Establishing the Project

Figure D-1 provides an overview of the process to establish a new open-source project.

```
%%{init: { 'sequence': { 'mirrorActors': false } }}%%
sequenceDiagram
  participant Proposer
  participant Committee
  participant WG as Working Group
  participant Maintainer
  participant Repo as Open-Source Project Repository

  Proposer ->> Committee: Propose project
  Committee ->> WG: Establish WG
  Committee ->> Maintainer: Assign maintainer
  Maintainer ->> Repo: Establish public repository
  Maintainer ->> Repo: Upload initial baseline
  Maintainer ->> WG: Suggest project plan
  WG -->> Maintainer: feedback
  Maintainer ->> Repo: Post project plan
  Maintainer ->> Repo: Create appropriate branches for work
```

Figure D-1 Establish a new open-source project

When someone identifies a need for a new shared resource (e.g., industry standard, reusable code, website, etc.) within ITS, they can develop a proposal and submit it to an appropriate committee. The proposal can be relatively simple (e.g., a statement of goals and structure) or a complete prototype.

If the proposal is accepted by the committee, the committee will assign a working group and one or more maintainers who will become responsible for leading the project. This will often include the individual proposing the project. A maintainer will establish the open-source project repository on the standards development organization's open-source website (e.g., GitHub account) and upload the initial project files.

> **Note**
>
> The maintainer is a key role in the project. If the maintainer is not available for any reason, it can delay the triage of identified issues. It is the responsibility of the committee to ensure that the maintainer either has sufficient resources or has sufficient backup to provide a high degree of confidence that there is not an artificial bottleneck when contributors wish to address problems.

Once the initial upload is provided, the maintainer will work with the working group to refine the vision for the project and establish the set of baseline issues as a part of the project plan. The project plan will also define the planned release schedule, which can be based on a calendar schedule, reaching milestones, or achieving other metrics. Members of the WG are encouraged to submit their issues directly so that the originator can be properly captured and to encourage WG members to become familiar with the process; however, the Maintainer can submit comments on the behalf of others, if needed.

The Maintainers are also responsible for properly managing the GitHub repository structure according to project policies. For example, this might include establishing a `draft` branch and ensuring all contributions are merged into this branch as appropriate. It might also define rules on when to tag committed versions as a release or pre-release and ensuring that GitHub actions properly run to generate updates to the website with all appropriate information.

The goal should be to have both the generated website and the GitHub page to default to the latest approved version (if an approved version exists) and should allow viewing any prior formal release or the most recent working draft (i.e., pre-release). Other versions can be provided as well at the WG's direction. The generated website should include a banner at the top of each page indicating the status of the currently displayed version, unless it is the latest approved version.

4.3 Process comments

Figure D-1 provides an overview of how comments are processed for an open-source project.

```
%%{init: { 'sequence': { 'mirrorActors': false } }}%%
sequenceDiagram
  participant Commenter
  participant WG as Working Group
  participant Maintainer
  participant Repo as Open-Source Project Repository

  Proposer ->>> Repo: Review materials
  Proposer ->>> Repo: Submit comment
  Repo -->> Maintainer: Notify
  Maintainer ->>> WG: Seek guidance
```

```
  WG -->> Maintainer: Provide feedback
  Maintainer ->> Repo: Perform triage
```

Figure D-1 Process comments for an open-source project

Users of open-source projects often have questions, encounter bugs, request features, or provide feedback on usability. Submitting comments is the primary way for the community to help guide the development of the project. Comments can be submitted at any time.

When comments are submitted, maintainers (and other followers) are notified. If the comment is submitted as an issue (as opposed to a discussion item), the maintainer triages the issue by determining its relevance, classification (e.g., bug, documentation issue), and priority. If needed, the maintainer can discuss the issue with the commenter or sponsoring WG to ensure consensus from the broader community.

Each project should identify its goals for triaging submitted issues. By default, projects should have a goal of triaging all comments within one month of their submittal, but the exact timeline might vary based on available resources, the criticality of the project, and other factors. If a submitted issue is not triaged within this timeline, the submitter should contact the parent standards development organization for guidance.

As a result of the review, the issue can be accepted, merged with another issue, split into multiple issues, or rejected (e.g., if it does not fit with the project's goals). Once the triage is complete, the maintainer adds tags as appropriate to the issue so that it can properly be managed.

4.4 Process Contributions

Figure D-1 provides an overview of processing contributions to an open-source project.

```
%%{init: { 'sequence': { 'mirrorActors': false } }}%%
sequenceDiagram
  participant WG as Working Group
  participant Maintainer
  participant Contributor
  participant Repo as Open-Source Project Repository

  Contributor ->> Repo: Review open issues
  Contributor ->> Repo: Claim issue
  Repo -->> Maintainer: Notify
  Contributor ->> Repo: Create copy
  Repo -->> Copy: Copy
  Contributor ->> Copy: Make edits
  Contributor ->> Repo: Submit pull request
  Repo -->> Maintainer: Notify
  Maintainer ->> Copy: Review
  Maintainer ->> WG: Optionally coordinate
  WG -->> Maintainer: Feedback
  alt if acceptable
    Maintainer ->> Copy: Merge
    Copy -->> Repo: Merge
  end
```

Figure D-1 Process contributions to an open-source project

Open-source projects encourage contributions from the community, allowing others to solve issues or implement features. Contributors gain experience and recognition, while the project benefits from a broader range of solutions.

Interested contributors browse the list of open issues, claim one they are interested in, and start working on a solution. When they have developed and tested their proposed solution, they submit a request for the maintainer to "pull" a copy of their changes from their site. This is known as a pull request (PR).

When a PR is submitted, the maintainer is automatically notified and is responsible for reviewing the request to ensure that it:

- can be safely merged with the project without overwriting other changes,
- solves the stated problem without introducing bugs, and
- meets the project's guidelines (e.g., coding standards).

During the review process, the maintainer can communicate with the contributor if questions arise or with the WG to ensure consensus on the details of the proposed change. If the process identifies any issues with the proposed change, it can be returned to the contributor to make additional revisions. If the changes are deemed to be satisfactory, the maintainer can accept the pull request and the changes will be merged into the open-source project.

4.5 Approve Releases

Figure D-1 provides an overview of the process to approve a new release of an open-source project.

```
%%{init: { 'sequence': { 'mirrorActors': false } }}%%
sequenceDiagram
  participant AG as Approval Group
  participant Maintainer
  participant Repo as Open-Source Project Repository

  Maintainer ->> AG: Suggest release (suggested release number)
  AG ->> Repo: Review materials
  AG -->> AG: Vote
  AG -->> Maintainer: Report results
  alt if approved
    Maintainer ->> Repo: Tag as identified release number
  else
    Maintainer ->> Repo: Address identified issues
  end
```

Figure D-1 Approve releases for an open-source project

Releasing a project allows users to access a stable, tested version with new features, bug fixes, or improvements. It also provides a versioned snapshot that is easier to manage and distribute.

Once all expected changes have been made to fulfil a defined stage in the project plan, the maintainer will follow the project's defined process for obtaining approval of the current draft as a formal release (e.g., v1.1.3) from the identified approval group (e.g., perhaps selected experts for a patch, the WG for a new feature, or the parent committee for non-backwards compatible changes). The exact approval group is defined in the project's plan.

If approval is received, the maintainer:

- documents changes in release notes (if not already included);
- if it is a full release, moves the version to the main branch; and
- tags the current version as a new release (e.g., "v1.1.3").

Assuming the maintainer has properly configured the repository with the necessary GitHub action, defining a new release tag will automatically generate an update to the website and a new PDF. Further, if `mike` is included in the configuration, the generated website will make previous versions of the website (and PDF) available from a dropdown menu.

If approval is not received, the maintainer ensures that all of the identified issues are properly recorded on the issues page and continues the process of addressing issues through contributions.

This collaborative process allows open-source projects to evolve through contributions from users and developers worldwide, promoting continuous improvement while ensuring transparency and accountability.

July 23, 2025

Section 4 Commenter Responsibilities

4.1 Overview

4.1.1 General

Comments on projects using the ITS Open-Source Process are always welcome, no matter how seemingly major or minor. Comments are key to improving products. The ITS Open-Source Process is designed to facilitate and encourage users to submit comments and therefore the commenting process is kept simple.

Within the ITS Open-Source Process, comments can be submitted in either the discussions or issues tab of the project repository.

4.1.2 Discussions

The discussions tab provides an open forum where interested parties can discuss ideas, ask and answer questions, and formulate ideas. The discussions tab does not directly propose any change to the project but can often nurture ideas that ultimately result in refining the overall vision of the project, identify problems or ambiguities in the project contents, develop consensus on project priorities, etc.

Discussions can be started by anyone at any time. Discussions can result in refining the concept of one or more issues before formally submitting them as issues.

4.1.3 Issues

Every project should follow a plan. Within the ITS Open-Source Process, the plan is documented by defining issues that are to be addressed, preferably according to assigned priorities.

The issues tab provides an open forum where any interested party can propose specific issues that need to be addressed by project contributors. The issues can be anything from a missing comma to requesting an entirely new feature. All proposed changes to a project are supposed to be initiated by submitting an issue.

When an issue is submitted, the project maintainer is responsible for triaging the issue. Triaging includes reviewing the issue, determining if the issue fits within the project plan, potentially parsing or merging the issue to create easily manageable tasks, assigning appropriate priority and tags (e.g., bug, ambiguity, editorial) to the issue, and gaining consensus on the approach.

This process can involve working with others on the project team to ensure consensus on the decisions being made.

Once an issue has been reviewed and accepted, anyone can claim ownership of the issue and begin resolving it. Given the complexities of version control when there are potentially multiple contributors, it is wise to separate issues into distinct bite-sized tasks that can be addressed with a reasonably short turn-around.

4.2 Submitting a Comment

4.2.1 Read the README file

Before commenting, commenters **should** be familiar with the project as documented in the README file.

4.2.2 Respect the CODE_OF_CONDUCT

When commenting, commenters **shall** respect the rules within the CODE_OF_CONDUCT file.

4.2.3 Use discussions if no change is proposed

For comments that do not actively propose a specific change to the project, the commenter **shall** initiate a discussion using the project's discussion template.

4.2.4 Use issues to propose changes

For comments that actively propose a specific change to the project, the commenter **shall** submit an issue using the project's appropriate issue template (e.g., bug fix, documentation improvement, new feature).

4.2.5 Comply with templates

The commenter **shall** comply with all instructions on the selected commenting template without deleting any fields.

> **Note**
>
> Inclusion of all fields facilitates processing of the comment and prevents automatic rejection. If a section of the template is not applicable, either explain why it is not needed or write "N/A".

Tip

The specific templates offered can vary from project to project, but the templates often include the following fields:

- **Title:** A short descriptive phrase to allow readers to quickly assess the comment
- **Description:** The details of the comment, especially those not captured in other fields of the template. If you wish to work on the issue that you are submitting, you should indicate this in the description. However, you should not start this work until the issue has been triaged to ensure it fits with the overall project plan. When reporting a bug, the description needs to be sufficiently detailed so that the reader can reproduce the anomaly.

July 23, 2025

Section 4 Contributor Responsibilities

4.1 Overview

Contributions on projects using the ITS Open Source Process are always welcome, no matter how large or small. However, before contributing, it's important to familiarize yourself with the following resources of the project:

Some of this information is standardized in this document, but specific projects can extend or make exceptions to the process and will always have their own project-specific goals. Contributors are responsible for being familiar with the information contained in the following project files, as stored in the project's root directory:

- **README.md**: Provides an overview of the specific project,
- **CODE_OF_CONDUCT.md**: Identifies the code of conduct for the project,
- **CONTRIBUTING.md**: Identifies project-specific rules for contributing, and
- **LICENSE.md**: Identifies the license agreement for project files

For projects following the ITS Open-Source Process, the last two files will typically only identify exceptions or extensions to the rules defined by this document.

A simplified process for contributing to an ITS open-source project is shown in Figure D-1.

```
%%{init: { 'sequence': { 'mirrorActors': false } }}%%
sequenceDiagram
  participant GHS as Source GitHub
  participant GHC as Contributor's GitHub
  participant Contributor

  Contributor ->> GHS: 4.2.4 Fork()
  GHS -->> GHC: Copy()
  Contributor ->> GHS: 4.2.6 Claim Issue()
  GHS -->> Contributor: Response
  Contributor ->> GHC: 4.2.8 Edit()
  Contributor ->> GHC: 4.2.9 Sync()
  GHC -->> GHS: Sync()
  GHS -->> GHC: Update()
  Contributor ->> GHC: Resolve Conflicts()
  Contributor ->> GHC: 4.2.11 Commit()
  Contributor ->> GHS: 4.2.13 Pull Request()
  note over GHS, Contributor: See Section 5 for merging into main branch
```

Figure D-1 Simplified contribution process

Figure D-1 provides a more advanced process that includes all options discussed in this section.

```
%%{init: { 'sequence': { 'mirrorActors': false } }}%%
sequenceDiagram
  participant GHS as Source GitHub
  participant GHC as Contributor's GitHub
  participant LC as Contributor's Local Copy
  participant LWD as Contributor's Local Branch
  participant Contributor

  Contributor ->> GHS: Fork()
  GHS -->> GHC: Copy()
```

```
Contributor ->> GHC: Clone()
GHC -->> LC: Copy()
Contributor ->> GHS: Claim Issue()
GHS -->> Contributor: Response
Contributor ->> LC: Branch
LC -->> LWD: new branch
Contributor ->> LWD: Edit()
Contributor ->> GHC: Sync()
GHC -->> GHS: Sync()
GHS -->> GHC: Update()
Contributor ->> GHC: Pull()
GHC -->> LWD: Update()
Contributor ->> LWD: Resolve Conflicts()
Contributor ->> LWD: Test()
Contributor ->> LWD: Commit()
Contributor ->> LWD: Push()
LWD -->> GHC: Push()
Contributor ->> GHS: Pull Request()
note over GHS, Contributor: See Section 5 for merging into main branch
```

Figure D-1 Contribution process for a significant change

4.2 Prerequisites

4.2.1 Join the relevant working group

Those wishing to contribute **should** join the relevant working group.

> **Note**
>
> Most projects using the ITS Open-Source Process are led by working groups (WGs) within standards development organizations (SDOs). The lead WG and SDO is typically identified within the README file in the root directory of the project repository. Contributors are strongly encouraged to join the corresponding working group to promote better communication among community members and to develop a common vision for the project.

> **Example**
>
> The ITS Open-Source Process project is led by the Base Standards and Profiles 2 (BSP2) WG of the National Transportation Communications Interface Protocols (NTCIP) Joint Committee (JC).

NTCIP Guidance

Within the NTCIP, any interested party can send an email to ntcip@nema.org. The email should indicate (1) the working group of interest, (2) the stakeholder sector (e.g., infrastructure owner operator, other government, consultant, device manufacturer, management station developer, etc.), (3) contact information, and (4) a short biography.

4.2.2 Install Software

4.2.2.1 Git

Those wishing to contribute **may** install Git on their local computer.

Tip

Changes can be made directly through the GitHub website; but most users find it easier manage changes on their local computer, especially if the changes are significant.

4.2.2.2 Graphical User Interface

Those wishing to contribute **may** install a graphical user interface (GUI) for Git on their local computer.

Tip

These tools often making the learning curve of using Git much easier. GitKraken is a very user-friendly tool that has a free version for public projects (such as NTCIP).

Example

Example GUIs for beginners include:

- GitHub Desktop: Very beginner level with minimal user interface
- SourceTree: Beginner-friendly with support for advanced Git functionality
- GitKraken: Beginner-friendly with advanced options and modern UI with useful video clips to explain how to perform tasks.

NTCIP Guidance

While contributors are allowed to use the CLI or any GUI of their preference, this document references GitKraken videos due to(1) the user-friendly design of GitKraken, (2) the high-quality help (including videos) available for GitKraken.

4.2.2.3 Development Environment

Those wishing to contribute **may** install the development environment on their local computer.

Note

Git is available for all major development platforms, including Windows, Mac OS, and Linux. Git allows proper version control among multiple contributors. Git can be downloaded from https://git-scm.com/downloads.

Git natively uses a command line interface (CLI), which can be difficult for beginners. There are a variety of graphical user interfaces (GUIs) that are available to assist with interfacing with Git.

Tip

Contributions can be rejected if they do not conform to contribution rules, including errors in compiling (for code) or rendering (for documentation). Installing the complete development environment is useful to allow the contributor to ensure that the materials will work as intended and due not result in automatic rejection.

4.2.3 Establish an Account on the Repository Hosting Platform

Those wishing to contribute **shall** have their own account on the same `git` hosting platform used by the main repository.

Those wishing to contribute **shall** have a user name that can be used to readily identify the contributor per their identity used within the working group.

Note

All edits originate within the contributor's account and all contributions can be traced back to the contributor.

NTCIP Guidance

Create an account on GitHub. This requires a valid email address but is free for open-source work.

4.2.4 Fork the repository

Those wishing to contribute **shall** fork the repository to their own account.

Note

The main repository is shared by the entire open-source community. Individual contributors are not allowed to directly edit this file as that would create a chaotic environment. Forking a repository creates a copy of the repository on the repository hosting platform within the contributor's account. The contributor can then edit the copied repository (as described below). The contributor's repository will inherit the visibility of the project being forked (i.e., for open-source projects, it will be public). This allows the open-source community to review the proposed changes prior to accepting their incorporation into the community repository.

Github Guidance

Press the "Fork" button in the upper-right portion of the shared repository's home page (e.g., https://github.com/<account>/<project>). For complete details, see the Fork a Repository article on GitHub help.

Note

Each project has a LICENSE.md file that defines its copyright. Projects are encouraged to use CC-BY for documents, BSD 3-clause for code, and the NTCIP MIB copyright for MIBs.

4.2.5 Clone the repository

The contributor **should** clone (i.e., copy an instance of) the forked repository to their local machine where edits are to be made.

Note

While a fork creates a copy on the host platform (which can be viewed by others), cloning your forked repository creates a copy of your forked repository on a local machine. This allows the contributor to edit files on a local machine rather than directly in the online environment.

Contributors can edit their copy of the repository directly through the GitHub interface; however, this requires a live connection to the Internet and working on a local machine is often easier for significant edits.

GitKraken Guidance

- GitKraken Guidance

4.2.6 Claim an Issue

Before starting on any changes to the project, a contributor **shall** claim an associated issue.

A contributor **shall** not claim an issue that has the label "triage".

A new contributor to a project **may** claim an issue tagged as good first issue, or beginners only.

Experienced contributors **shall not** claim issues tagged with the label "beginners only".

Experienced contributors **should** avoid issues tagged with the label "good first issue".

Note

Taking ownership of an issue:

- Notifies maintainers that work is starting to address the issue,
- Allows efficient communication by allowing the maintainers and contributor to discuss the issue and proposed changes early in the update cycle,
- Provides a historical record of the steps taken to address the issue,
- Helps to block inappropriate pull requests as any pull request without an associated issue can be easily rejected.

When you're new to a project, it's a good idea to start with small, manageable tasks, fixing bugs, adding tests, or updating documentation. These will often be tagged with the text "good first issue" or "beginners-only". This will help you become familiar with the material and development workflow without getting overwhelmed. The goal is to reserve at least some of these issues for new contributors or until the end of the project; if everyone solves these problems first, it makes it more challenging for contributors to gain experience.

GitHub Guidance

If an issue is not assigned and it is not labeled with "triage", it is generally assumed to be available for anyone to work on. Take control of the issue by submitting a comment of `.take` on the selected issue. When an issue is assigned, it will be indicated under the "Assignees" section of the issue.

Note

Projects can implement additional rules regarding the assignment of issues. Always review the project's contributing guidelines to ensure you are aware of any variations from this standard process.

Tip

If you get stuck while working on your changes or need other clarification, you can always ask for help using the discussions tab of the project. For example, you can get help for the ITS Open-Source Process project at its Discussion Tab.

4.2.7 Create a Branch

Prior to starting work on a claimed issue, the contributor **may** create a separate branch for all edits related to that singular issue.

Note

Creating a separate branch facilitates tracking of changes and allows easier roll-backs of the project to known states. It can also be useful to separate tasks if the contributor is working on multiple issues at the same time. However, contributors need to be aware that separate branches that edit the same file will require manual review to properly merge edits.

4.2.8 Make Edits

Once the contributor has claimed an issue and has the desired branch activated, the contributor **shall** make changes in the local branch according to project guidelines.

Note

Annex B, C, and D provide preferred guidelines that can be referenced for different types of contributions.

If you have questions or concerns during the process (especially between meetings of the corresponding WG), you can use the Discussions tab associated with the project. These forums can be very useful in knowledge sharing and forming consensus, however, users should be aware that the discussions tab does not represent official decisions of the WG.

Tip

Avoid addressing any other issues as this (1) makes the change larger and delays completion of your primary task, (2) can overlap with changes being made by others, and (3) complicates version control by not clearly documenting when specific changes were made. However, in some cases, it may be appropriate to address multiple small and similar issues at once. For example, multiple grammar issues in documentation can be grouped into a single pull request.

4.2.9 Sync and Pull Updates

Prior to submitting a PR, the contributor **shall** sync the latest updates incorporated into the shared project and pull these updates into the contributor's working branch, resolving any conflicts as needed.

Note

Because multiple contributors can be working on the same project simultaneously, care must be taken to ensure that each contributor has the latest version of files prior to proposing their changes to be incorporated into the shared repository. This is done by first synchronizing any changes from the shared repository with the contributor's forked version and then pulling those changes down into the contributor's local branch. During this process, the Git environment will highlight any conflicts (e.g., if the contributor and someone else changed the same line of the same file). When this occurs, the contributor will need to resolve each conflict prior to finalizing the merge.

GitHub Help

To update your local copy, first update your forked (origin) repository:

- Go to your forked repository on GitHub.
- Click the "Sync fork" button.
- Click the green "Update branch" button.

Next, pull the latest changes in the main branch in the origin repository to update your local working branch by following these steps in your terminal:

- `git checkout YOUR-BRANCH-NAME`
- `git pull origin main`

4.2.10 Test the Updates

After synchronizing, pulling, and merging the latest updates but prior to submitting the PR, the contributor **should** install any necessary dependencies and test the changes to ensure that the changes provide the intended operation without any new bugs.

Note

It is critical that updates are tested prior to being incorporated into the final code (this includes ensuring that documentation files render correctly). Specific projects can define their own testing process and procedures. You can find the instructions on how to run a project locally in the README file or in the contributing guidelines.

Maintainers have the right to reject any contribution that fails properly compile.

4.2.11 Commit the Update

Prior to pushing the proposed changes to online repository, the contributor **shall** commit the changes in the working directory.

The contributor **may** perform interim commits during the development of the proposed changes.

The contributor **should** use the Conventional Commits specification for structuring commit messages.

Here are some examples of Conventional Commit messages:

- feat: add password reset functionality
- docs: update installation instructions
- chore(build): update dependencies
- fix(login): resolve issue with incorrect password validation
- refactor(api): streamline error handling in user service

Note

Committing changes ensures that the changes are logged in the contributor's local git account and is required prior to pushing the material to the contributor's online copy.

4.2.12 Push the Update to Contributor's Online Repository

Once the contributor has completed the proposed revisions and has created a local commit, the contributor **shall** push the proposed changes to the contributor's online repository.

Note

The changes need to be posted to the online repository so that other users can review the changes prior to their incorporation into the shared repository.

4.2.13 Make Pull Request

Once the contributor has completed the above steps, the contributor **shall** complete a pull request.

The contributor's pull request **shall** comply with the selected pull request template for the project, completing each field.

The contributor **shall** verify that GitHub does not report any action bot or other failures upon submitting the PR.

Note

In order to ensure that changes made to the shared repository fit with the project plan, follow submittal guidelines, and are free of bugs, it is important that they are reviewed before being incorporated. As such, rather than allowing each contributor to push changes to the shared repository without any review, they **request** the maintainer to **pull** the proposed changes. The request initiates the review process, and if successful, the changes will be pulled. The request for the maintainers to pull the updates is called a "pull request" (PR). In order to ensure that these requests are valid and useful, they must comply with the pull request template (e.g., identify the issue that the change claims to address).

It is especially important that the PR identifies the issue that the PR claims to address and must prefix the issue number with "Fixes #" to ensure that the issue is closed once the change is accepted.

The contributor will need to correct any errors that occur during the submittal process to ensure that the PR is received by the maintainers.

Warning

A PR may be marked as invalid and closed if:

- the issue is not assigned to the contributor who opened the PR,
- no issue is linked to the PR,
- the PR template is incomplete, or any section in the template is deleted, or
- changes are made directly in the default (`main`) branch.

4.2.14 Cooperate with Reviewers

The contributor **shall** work with the review team to address any questions, concerns, or problems that arise.

The contributor **may** appeal any direction received from the reviewers to the parent WG.

The contributor **shall** accept the direction of the review process, including any appeals.

Note

After a pull request has been submitted, reviewers can have questions or concerns (e.g., failure to comply with style guidelines). In addition, if multiple proposals are received in a short period, last minute changes can cause merge errors that need to be resolved. The contributor is typically the person most qualified to make revisions to the proposed changes without introducing errors. Although expected to be rare, there can be instances where the contributor and reviewers have different opinions about how a change should be implemented. The shared project is managed by the entire team and the contributor needs to respect the decisions made by the full team.

4.3 What Happens Next?

After your contribution has been submitted and reviewed, one of the following outcomes may occur:

1. **Your contribution is accepted**: If your contribution is approved by the project maintainers, it will be merged into the main branch of the codebase.
2. **Your contribution requires changes**: Sometimes, the project maintainers may request changes to your contribution before it can be accepted. This could be due to coding issues, conflicts with other changes, or a need for additional documentation. In this case, make the requested changes and resubmit your pull request.
3. **Your contribution is rejected**: In some cases, your contribution may not align with the project's goals or requirements, or it may not be the best solution to a problem. If your contribution is rejected, don't be discouraged. Take the feedback you received as an opportunity to learn and improve. You can always try contributing to another project or submitting a different contribution to the same project.

4.4 Keeping Branches Up to Date

It is highly recommended that you update your remote and local branches habitually. That way, your branch will have the latest update when merged into the `main` branch of the original (`upstream`) repository.

The best times to update your branches are before you push your changes to the remote repository and while you're waiting for your pull request to be reviewed.

In general, it is preferable to make small incremental changes to the project and to provide the updated materials as soon as possible after taking control of an issue. The longer the duration between checking out the project and submitting a pull request the higher the chance that another contributor will make competing changes in one of your files, which may need to be manually inspected to properly merge the changes.

4.5 Merge Conflicts

Merge conflicts are something you'll commonly encounter when contributing to an open source project. When two branches have made different changes to the same line(s) in the same file(s), Git cannot automatically determine which change to keep, resulting in a conflict.

When a merge conflict occurs, Git adds conflict markers (`<<<<<<<` , `=======` , and `>>>>>>>`) to indicate the conflicting lines from different branches. Everything between the `<<<<<<<` and `=======` is the changes that you worked on (current changes). And everything between the `=======` to `>>>>>>>` is the incoming changes from the remote `main` branch.

You need to pay attention to the conflicts and decide how you want to resolve them. You can keep only your change, incoming change, or both changes.

4.5.1 Tips to Prevent Resolving Merge Conflicts Repeatedly

Some open source repositories, such as OpenSauced's guestbook and pizza-verse repositories, have high contribution activities in the same files that can cause merge conflicts.

Below are some tips to prevent you from resolving merge conflicts repeatedly when contributing to open source projects:

4.5.1.1 1. Following Instructions

Ensure you follow the instructions in the project's README or Contributing Guide, and don't miss any step.

4.5.1.2 2. Pull Request Form

Complete the template form and fill in all areas when creating a pull request.

4.5.1.3 3. Resolving Merge Conflicts Immediately

If a branch has merge conflicts that must be resolved, the merge button is automatically disabled. So, maintainers are not able to merge the pull request.

When you notice merge conflicts in your pull request or if a maintainer asks you to resolve merge conflicts, fix them immediately. The sooner you resolve the conflicts, the sooner maintainers can review and merge your pull request.

4.5.2 Merge Conflicts in the Guestbook Repository

Since the primary purpose of the OpenSauced guestbook is to add your name to `.all-contributorsrc` and the `README.md` files, there is a high chance that you will encounter merge conflicts.

The conflicts happen when maintainers have merged pull requests before yours while you're working on your changes or waiting for your pull request to be reviewed. And you need to resolve them before your pull request can be merged.

4.5.2.1 Resolving Merge Conflicts

Before resolving merge conflicts, you must first

update your branches. Then, follow these steps:

1. In the `.all-contributorsrc` file:
2. Click the "Accept Both Change" option on the top of your workspace in VS Code.
3. Move your profile details to the end of the contributors' array and fix anything necessary.
4. In the `README.md` file:
5. Click the "Accept Incoming Change" option on the top of your workspace in VS Code for each conflict in this file.
6. Run `npm run contributors:generate`.

You will now see that the all contributors badge has been incremented, and your profile is generated at the end of the contributors' list in the `README.md` file.

1. Add and commit your changes.

   git commit -am "Resolve merge conflicts"

2. Push your commits to your remote branch.

   git push

4.6 Setup

4.6.1 Install Git

Download and install the git program appropriate for your platform. Using default options should be fine unless you have a particular preference (e.g., for text editor).

4.6.2 Create a GitHub account

Go to github.com, create an account and sign in

4.6.3 Fork the Desired Repository

4.6.4 Install GitKraken

Recommended: Download and install GitKraken. Link GitKraken to your GitHub Account

4.6.5 Clone Repository

Make sure to select a directory where you want to store the local copy of the repository. This directory needs to be empty

July 23, 2025

Section 4 Maintainer Responsibilities

When starting a project, the maintainer **shall**:

- Establish the open-source project on GitHub under the designated organizational account
- Upload the initial project files

4.1 Overview

The maintainer for an open-source project fulfills many responsibilities, including setting up the project, managing issues, reviewing submittals, and leading the development community. In addition, the maintainer is often a prime contributor.

4.2 Establish Repository

The maintainer **shall** work with the sponsoring SDO to establish the open-source repository for the project.

Example

NTCIP repositories are hosted athttps://github.com/ite-org/.

4.3 Configure Project Settings

4.3.1 Issues and Discussions

The maintainer **shall** ensure that the issues and discussion pages are enabled for the ITS open-source project.

Note

Within GitHub, issues are enabled by default but the discussions tab is disabled. To enable, go to the settings tab and select discussions in the general section.

4.3.2 Pages

If the project includes documentation, the maintainer **shall** ensure that GitHub pages is enabled for the project.

GitHub Process

To activate GitHub Pages using MkDocs, create a gh-pages branch. The go to the settings tab and select pages from the left-hand menu. Set Source to "deploy from a branch" and then select the "gh-pages" branch and the /root directory.

4.3.3 Dependabot

If the project includes code, the maintainer **may** configure Dependabot to report issues or create pull requests to update dependencies with security vulnerabilities.

Note

Dependabot is a GitHub feature that monitors the project's dependencies and reports any possible security vulnerabilities. To learn more about this feature, please read through the GitHub documentation.

4.4 Set Up Project Files

4.4.1 Overview

The maintainer **shall** ensure the following files are provided in the repository when starting the project and maintained throughout the project:

- README.md
- CODE_OF_CONDUCT.md
- CONTRIBUTING.md
- LICENSE.md
- SECURITY.md
- .github/CODEOWNERS
- appropriate issue templates in .github/ISSUE_TEMPLATE/
- appropriate PR templates in .github/PULL_REQUEST_TEMPLATE/
- appropriate saved replies

Additionally, if the site includes documentation, the maintainer **shall** ensure the following files are provided in the repository when starting the project and maintained throughout the project:

- .gitignore
- mkdocs.yml
- .github/workflows/deploy.yml (generates the documentation)
- docs/index.md
- docs/stylesheets/extra.css
- overrides/main.html (adds a status badge to the page)
- overrides/partials/nav.html (changes the navigation heading to be "Contents")
- overrides/partials/toc.html (changes TOC heading to be the document name)

Finally, the site **shall** include installation guidance, which **may** be contained in the README.md file, a separate INSTALLATION.md file, or in project documentation.

GitHub Help

A template repository containing all of these files, which can be used to initialize new projects, is stored on the ITE GitHub Site.

The recommended directory structure for NTCIP projects is as follows:

```
<Repository>
    ├── .github
    │   ├── ISSUE_TEMPLATE
    │   │   └── <Various yml templates as needed>
    │   ├── workflows
    │   │   ├── auto_assign.yml
    │   │   ├── triage_label.yml
    │   │   └── update_release.yml
    │   ├── CODEOWNERS
    │   └── PULL_REQUEST_TEMPLATE.md
    ├── .vscode
    │   └── settings.json
    ├── docs
    │   ├── images
    │   │   └── <Various embedded images as needed; preferred format is svg>
    │   ├── javascripts
    │   │   └── <Various javascripts as needed (e.g., tablesort.js)>
    │   ├── mermaid
    │   │   └── <Auto-generated SVG files produced by Mermaid for PDF file>
    │   ├── stylesheets
    │   │   ├── extra.css
    │   │   └── pdf.css
    │   ├── acknowledgements.md
    │   ├── foreword.md
    │   ├── general.md
    │   ├── index.md
    │   ├── introduction.md
    │   ├── notices.md
    │   └── <Other md files as needed for left side nav; at least one for each section>
    ├── images
    │   └── <Any images needed for GitHub site but not used in actual documentation>
    ├── overrides
    │   ├── partials
    │   │   ├── nav.html
    │   │   └── toc.html
    │   └── main.html
    ├── resources
    ├── templates
    │   ├── main.html
    │   └── pdf-styles.scss
    ├── .gitignore
    ├── .markdownlint.json
    ├── CODE_OF_CONDUCT.md
```

```
    ├── CONTRIBUTING.md
    ├── LICENSE.md
    ├── README.md
    ├── SECURITY.md
    ├── markdown-custom.js
    └── mkdocs.yml
```

4.4.2 Readme.md File

The README.md file **shall** contain an introduction to the open-source project. A good readme file should be clear, concise, up-to-date, and detailed. This file is located in your root directory and is displayed as the homepage of the repository within GitHub.

The README.md file **shall** contain the following information:

- project title
- information on how to access the current documentation for the project
- information on how to access prior releases of the project
- project summary, including its status and overview
- Acknowledgements of relevant funding sources, sponsors, and other open-source projects
- installation guidance
- tech and tools used in the project
- link to the code of conduct
- link to discussion forum for the project
- link to the issues page for the project and the types of issues accepted for the project
- link to the contributing guidelines
- link to the open source license

Note

A README file is written in the Markdown language, a popular language used in open source documentation like READMEs. The readme file does not use any of the special codes introduced by MkDocs or Materials for MkDocs.

Examples of good README files

- OpenSauced App
- Astro documentation
- freeCodeCamp
- ITS Open-Source Process

4.4.3 Installation Guidance

The installation guidance **may** be contained within the README.md file (e.g., if it is simple), be a separate file, or reference a section within the project documentation.

This guide identifies the tools and technology used by the project and includes instructions for the following:

- forking the repository
- cloning the repository
- installing the dependencies
- setting up the environment variables
- setting up the database, if applicable
- running the project locally

The best way to test your guide is by setting up the project locally using your guide. If you encounter issues getting your project to work, you will discover it quickly and can update the documentation to add or clarify the missing piece.

Example

OpenSauced Contributing Guidelines

4.4.4 Code of Conduct File

The CODE_OF_CONDUCT.md file **shall** define the rules and behaviors that are to be followed for the project.

The CODE_OF_CONDUCT.md file **should** consist of a reference to the ITS Open-Source Code of Conduct with any exceptions and extensions identified.

Note

It is expected that exceptions and extensions to the code of conduct will be rare.

4.4.5 Contributing File

The CONTRIBUTING.md file **shall** define the rules for contributing to the project.

The CONTRIBUTING.md file **should** consist of a reference to the ITS Open-Source Contributor Responsibilities with any exceptions and extensions identified.

The CONTRIBUTING.md file **should** identify specific types of conventions that apply to the project.

Example

The ITS Open-Source Process project only has documentation and while the resulting specification discusses coding conventions, the project does not include any code, ASN.1 or MIBs.

Note

Areas where exceptions and extensions are expected to occur include:

- documentation conventions (e.g., extensions for consistency in presenting project-specific information),
- coding conventions (e.g., for languages not discussed in the ITE Open Source Process documentation, special naming conventions),
- testing and linting requirements
- the process to claim ownership pf issues (e.g., WG approvals),
- guidelines for commit conventions
- requirements for creating pull requests (e.g., fields that need to be included), and
- requirements for pull requests to be approved (e.g., WG approvals)

4.4.6 License File

> **Per The Legal Side of Open Source**
>
> Making your GitHub project public is not the same as licensing your project. Public projects are covered by GitHub's Terms of Service, which allows others to view and fork your project, but your work otherwise comes with no permissions.
>
> If you want others to use, distribute, modify, or contribute back to your project, you need to include an open source license. For example, someone cannot legally use any part of your GitHub project in their code, even if it's public, unless you explicitly give them the right to do so.

The LICENSE file **shall** be a well-known Free and Open Source license.

For NTCIP documentation projects, including standards and ASN.1, the license **should** be CC BY 4.0.

Note

The CC BY license is designed for documentation and other creative works where users are allowed to use, distribute, modify, and contribute but any derivative works are required to give attribution to the source of the material and cannot "implicitly or explicitly assert or imply any connection with, sponsorship, or endorsement by the licensor."

For NTCIP projects involving code for a computer program, the license **should** be the Gnu Lesser General Public License Version 3.

Note

The LGPL v3 license is designed for compilable software that runs on a machine. Like the CC BY license, it allows users to use, distribute, modify, and contribute material as long as they give attribution to the original source and does not provide any rights to the names, trademarks, or logos of the original source.

4.4.7 Security file

The SECURITY.md file **shall** indicate how to provide reports of security issues through private channels to prevent exposure of the vulnerabilities prior to their fix.

4.4.8 Code Owners File

The .github/CODEOWNERS file **shall** conform to the rules defined in the official GitHub documentation.

Note

This ensures that the correct maintainers are notified when PRs are submitted.

Example

By opting-in to "require approval" and "require review from code owners", a WG can require a majority of voting members of the WG to approve any pull request before it can be merged into the protected branch. This can reduce the chance of merging pull requests that can break production.

4.4.9 Issue Templates

The maintainer **shall** define appropriate issue forms.

The maintainer **shall** develop the forms using YAML per the GitHub instructions.

Note

Issue forms allow the maintainer to ensure commenters provide key information (or at least provide text for specific fields) when they report issues making the triage process, and the review of pull requests easier to perform. Additionally, future contributors can benefit from these templates by understanding the history of changes made, which can help them debug or understand the code involved.

You can create various issue forms, such as bug reports, feature requests, documentation updates, etc. Each form can specify which fields are required, such as the steps for reproducing the bug or a details section for a feature request. The form can also be designed to automatically attach specific labels like `feature`, `needs triage`, or `bug` to quickly identify the type of issue.

Example

`.github/ISSUE_TEMPLATE/documentation_bug.yml` or `.github/ISSUE_TEMPLATE/documentation_enhancement.yml`

4.4.10 Pull Request Templates

The maintainer **shall** define appropriate pull request templates.

Note

GitHub currently only supports markdown templates for pull requests rather than YAML forms. Nonetheless, the templates serve a similar purpose in that they guide contributors in providing specific and structured information when opening pull requests in your project.

Example

.github/PULL_REQUEST_TEMPLATE/PULL_REQUEST_TEMPLATE.md

What type of PR is this? (check all applicable)

- [x] 🍕 Feature
- [ ] 🍾 Bug Fix
- [ ] 📝 Documentation Update
- [ ] 🎨 Style
- [ ] 👷 Code Refactor
- [ ] 🔥 Performance Improvements
- [ ] ✅ Test
- [ ] 🤖 Build
- [ ] 🔁 CI
- [ ] 📦 Chore (Release)
- [ ] ⏩ Revert

Related Tickets & Documents

closes #12

Mobile & Desktop Screenshots/Recordings

Figure D-1 PR Template Example

> **Tip**
>
> You can learn more about creating a pull request template on the official GitHub documentation.

4.4.11 Saved Replies

Sometimes, you repeatedly write the same reply to issues or pull requests. Clear communication between maintainers and contributors is crucial. So, when you write all comments manually, your messages will no longer be consistent and may be unclear. You can create saved replies when you frequently respond to issues and pull requests with the same comments.

Saved replies allow you to create a reusable response to issues, pull requests, and discussions and use it across repositories. It will save you time responding to contributors while keeping the consistency of your message. You can always modify your replies if necessary.

Read the GitHub documentation for complete instructions about how to create saved replies.

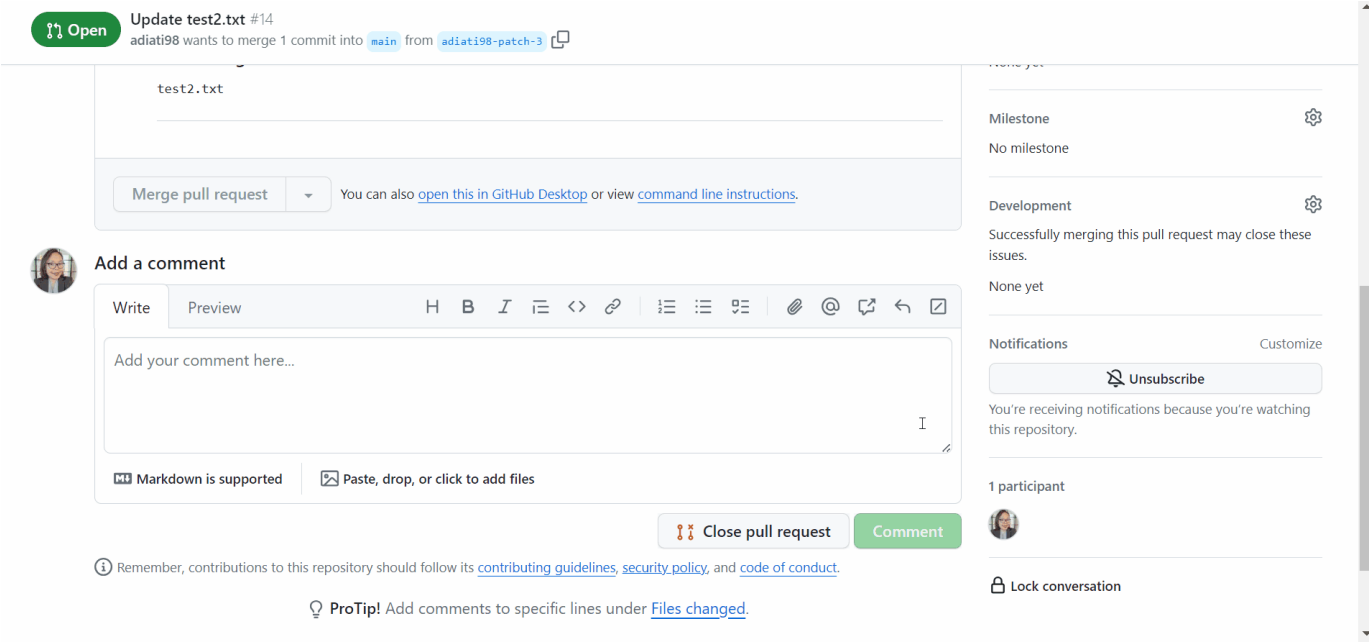

Figure D-1 Saved replies

4.4.12 Gitignore File

If there are any files that are likely to exist within the project directory that should not become a part of the repository, the maintainer **shall** include a `.gitignore` file.

Example

Projects that include documentation that is based on MkDocs typically have a `site` directory generated by MkDocs as a part of the process to generate a static website for review. This directory is not intended to be part of the registry as GitHub will produce its own generated site. In this case, the project is to include a `.gitignore` file that contains the line "site/", as for this project.

4.4.13 Mkdocs.yml File

If the project includes documentation using MkDocs, the maintainer **shall** define a `mkdocs.yml` file.

Note

The mkdocs.yml file defines directives to the mkdocs engine when generating the static website. For example, it defines

- name of the site,
- URL of the site,
- navigation menu,
- theme,
- extensions/plugins used, and
- other details

Example

Mkdocs file for ITS Open-Source Process

4.4.14 Deploy.yml File

If the project includes documentation using MkDocs, the maintainer **shall** define a `.github/workflows/deploy.yml` file that deploys the MkDocs site to GitHub Pages.

Note

The deploy.yml file can be used to automate actions when pull requests are merged with the project. For example, for projects that use MkDocs, the deploy file can ensure that the GitHub Pages site is updated with the new material when a pull request is merged.

Example

Deploy file for ITS Open-Source Process

4.4.15 Index File

If the project includes documentation using MkDocs, the maintainer **shall** define a `docs/index.md` file that provides a cover page for the document.

4.4.16 Extra.css File

If the project includes documentation using MkDocs, the maintainer **shall** define a `docs/stylesheets/extra.css` file that defines the heading styles to be used for the body and the annexes (i.e., adding section numbers in front of headings).

The maintainer **shall** ensure that this badge only changes per the approval process defined for the project.

4.4.17 Main.html File

If the project includes documentation using MkDocs, the maintainer **shall** define a `overrides/main.html` file that provides a badge that identifies the status of the project files.

The maintainer **shall** ensure that this badge only changes per the approval process defined for the project.

Note

By default, MkDocs does not display a status badge. Adding this badge ensures that users are aware of the status of the material located on the page.

Example

Main.html file for ITS Open-Source Process

4.4.18 Nav.html File

If the project includes documentation using MkDocs, the maintainer **shall** define a `overrides/partials/nav.html` file that overrides the title assigned to the left-hand navigation menu to be "Contents".

Note

By default, MkDocs entitles the left hand navigation (i.e., navigation of pages) with the title of the site. To better align with NTCIP formats, the nav.html file is provided to change this to "Contents".

Example

Nav.html file for ITS Open-Source Process

4.4.19 Toc.html File

If the project includes documentation using MkDocs, the maintainer **shall** define a `overrides/partials/toc.html` file that overrides the title assigned to the navigation menu on the right side of the screen.

Note

By default, MkDocs entitles the right-hand navigation (i.e., the contents of the current page) as "Contents". To better align with NTCIP formats, the standard toc.html file changes this to the title of the current page.

Example

Toc.html file for ITS Open-Source Process

D.5 Branch Management

The maintainer **shall** coordinate with the WG to determine when to use separate branches for the project.

The maintainer **shall** store the current release in the `main` branch.

The maintainer **shall** store the documentation website in the `gh-pages` branch.

For projects that only display a single release (e.g., a website project), the maintainer **may** use the `main` branch for all updates.

For projects that have not yet released version 1.0.0, the maintainer **may** use the `main` branch for all updates.

For projects that use provide multiple releases on the website and have already released version 1.0.0 or later:

1. the maintainer **shall** use a branch other than `main` or `gh-pages` for any version under development (including pre-releases)
2. the maintainer **should** use a separate branch for each development effort that has started but is not expected to be included in the next official release
3. the maintainer **may** use a separate branch for distinct development efforts
4. the maintainer **should avoid** using separate branches for separate development efforts that are expected to modify the same file

The maintainer **should** consider dividing a file into distinct parts when it needs to be modified by separate development efforts.

Note

Modifying the same file in different branches will result in a need to manually merge the changes, which creates extra work. In cases, this can be avoided by dividing the file into distinct parts. In other cases, this can be difficult to achieve and manual merging will either be required or the longer-term development effort delayed until the first development effort has completed its updates.

4.6 Define Project Structure

Once the project is established, configured, and the loaded with the initial project files, the maintainer **shall** establish the plan for the project by defining issues along with any appropriate stub files that can provide further guidance.

Note

This includes defining the plan for all aspects of the project (e.g., documentation and code).

4.7 Issue Triage

4.7.1 Overview

Once issues are being created for the project, the maintainer will need to triage these issues to ensure that they contribute to the project plan and so that they can be claimed by contributors.

Learning to triage issues is essential for any open-source maintainer. This involves going through the existing list of open issues and prioritizing them in order of importance. Some open

issues will be critical bug fixes, while others might be nice to have feature requests. Sometimes, you might have issues opened for things that are not a right fit for the project.

4.7.2 Triage Pre-Assessment

4.7.2.1 Overview

Prior to performing any detailed triage, it is important to screen reported issues that are not appropriate for further investigation.

4.7.2.2 Dealing with Spam

When a comment is spam, clearly combative, or unhelpful, the maintainer **should** avoid direct engagement, label the issue as `spam`, close the issue, and move on.

> **Example**
>
> This project is terrible! Nothing works, and your code is garbage. I can't believe anyone would use this. Fix it ASAP!!!

4.7.2.3 Insufficient Information Issues

When a comment does not provide concrete details about the issue, the maintainer **should** respond by requesting more information.

> **Example**
>
> If it is a bug report, ask for more details on reproducing it. If it is a feature request, ask for clarification on style or functionality changes.

If the commenter does not respond within a week, the maintainer **should** message them again for more details.

If a few weeks pass and the issue is not considered critical, the maintainer **may** close the issue.

4.7.2.4 Stale Issues

The maintainer **may** may label issues that have not been worked on for months as `stale`.

Example

An issue reported on a portion of a project that has been significantly edited by other contributions.

If the maintainer wishes to resurrect a `stale` issue, the maintainer **shall** go through the normal triage process, including adding and removing labels as appropriate.

If the maintainer believes a `stale` issue no longer applies, the maintainer **should** close the issue. This process **may** be automated (e.g., using an action like Close Stale Issues and PRs.

4.7.2.5 Ensuring Proper categorization

For issues that have sufficient information and are not `spam` or `stale`, the maintainer **shall** add and/or remove labels as appropriate for proper management.

Note

Most ITS standardization projects are expected to have a small number of contributors, in which case, the following set of labels are generally appropriate. Projects with more contributors should consider a fuller range of labels as adopted by the open-source community.

The following labels **should** be considered for most specification projects:

1. General Type Labels

- **github automation:** an issue related to the automated github scripts in testing or generating the documentation.
- **question:** Indicates a general inquiry or a request for clarification about how something works. Questions should be moved to the discussion tab, but the issue can be labeled with `question` and closed.

2. Type Labels for Documentation

- **documentation bug:** Identifies a reported problem or flaw in the documentation.
- **documentation enhancement:** Refers to a suggestion or request to improve or add informative text in the documentation.
- **new user need:** Suggests a new user need to be added in the document.
- **user need modification:** Suggests a modification to a user need in the document.
- **new requirement:** Suggests a new requirement to be added in the document.
- **requirement modification:** Suggests a modification to a requirement in the document.
- **dialog modification:** Suggests a modification to a dialog in the document.
- **ASN.1 modification:** A change to the ASN.1 or MIB.

3. Type Labels for Code

- **bug:** Identifies a reported problem, flaw, or unexpected behavior in the code.
- **enhancement:** Refers to a suggestion or request to improve or add features to the project.
- **feature:** Used for issues proposing new functionality or significant changes.
- **refactor:** A change in the codebase that improves its structure or readability without altering its functionality.
- **test:** Issues related to unit tests, integration tests, or overall testing improvements.

4. Priority Labels (Help prioritize issues based on urgency or importance)

- **critical:** Indicates urgent issue related to an existing release that needs to be addressed immediately (e.g., perhaps requiring its own release to formalize in a timely manner).
- **high priority:** Indicates urgent issue that needs to be addressed as soon as possible to allow for other tasks to proceed for the current update but not critical.
- **medium priority:** Important issue to address prior to next release that relates to functionality but not high priority.
- **low priority:** Non-urgent issues that may be tackled if there's extra time or resources available.

5. Status Labels (Track the progress of an issue or pull request)

- **in progress:** The issue is currently being worked on.
- **blocked:** Work on this issue is delayed or cannot proceed due to a dependency or external factor.
- **needs discussion:** Indicates that the issue or pull request requires further conversation or clarification before proceeding.
- **triage:** Newly created issues that need to be reviewed, categorized, and prioritized.
- **ready for review:** The pull request is awaiting review by project maintainers.
- **duplicate:** Marks an issue as being identical or closely related to an already existing issue.
- **wontfix:** Indicates that the maintainers have decided not to address the issue, either due to scope, relevance, or priority.

6. Difficulty or Effort Labels (Classify the expected effort required to address the issue)

- **good first issue:** Meant for new contributors; these are usually easy-to-solve problems with clear instructions.
- **beginner-friendly:** Similar to "good first issue," these are relatively simple problems that beginners can address.
- **help wanted:** Indicates that maintainers need assistance with the issue, open to contributions.
- **complex:** Issues that are challenging, requiring significant experience or effort to resolve.

7. Version or Milestone Labels (Track issues by release or milestone)
    - **compatibility:** Associates an issue or pull request with a specific version or release milestone.
    - **next release:** Indicates that the issue is planned for inclusion in the upcoming release.
    - **future:** Refers to issues or features planned for future releases beyond the current roadmap.

4.7.3 Triaging Bugs

The maintainer **should** verify the existence of any reported bug.

> **Note**
>
> If the maintainer expects to assign the issue to himself, the verification step can be postponed until the action is undertaken to correct the bug. If the bug cannot be verified, reply to the issue's original poster to gain more information and context.

4.7.4 Triaging Feature Requests

The maintainer **shall** ensure that any new feature request fits into the vision for the project.

The maintainer **shall** communicate with the original poster of the issue to determine how to best deal with the issue, including:

- assigning to the originator, if they express a willingness to contribute a solution,
- assigning to the originator and someone else (e.g., a maintainer), if the originator is willing to contribute to a solution but is unwilling to develop and propose it,
- adding a `help wanted`, if the maintainer wishes to look for another contributor, or
- assigning it to himself or another core team member, if the maintainer expects the issue to be addressed by them in a timely manner.

> **Note**
>
> Complex issues are best assigned to core team members.

4.7.5 Triaging Duplicate Issues

When assigning a `duplicate` label, the maintainer **should** respond to the originator of the issue.

Example

Thank you for taking the time to open this issue. Another team member is working on this feature, which will be added soon. As a result, we are going to close this issue.

4.7.6 Triaging Rejected Issues

When assigning a `wontfix` label, the maintainer **should** respond to the originator of the issue.

Example

Thank you for being so interested in our project. The feature you have proposed would not be a good fit for this project's current scope and direction. At this time, we will not be moving forward with this feature.

4.8 Reviewing Pull Requests

4.8.1 Overview

It is the maintainer's responsibility to ensure that the suggested code or documentation update meets the standards of the project and doesn't introduce any new issues for the project. You will also need to work with the contributor to help solve issues they encounter.

4.8.2 Pull-Request Pre-Assessment

4.8.2.1 Spam Pull Requests

For any pull request deemed to be spam, the maintainer **should** label the request as `spam`, close the pull request, and not respond to the contributor.

Example

- whitespace changes to the README file or other files
- random changes to files without an accompanying issue or explanation
- numerous links to unrelated websites or promotes products/services
- plagiarized content from other sources without permission or proper attribution

4.8.2.2 Low-Quality Pull Requests

For a pull request deemed to be of low quality, the maintainer **should** reach out to the author, explaining what needs to be added and what changes need to be made.

Example

- unfinished pull requests that do not address the entire issue
- code that does not fit within the established style guide for the project
- incomplete pull request forms that do not provide sufficient information on what changes were made
- address multiple issues at once and make it challenging to review

Note

Most of the time, low-quality pull requests are due to contributors not being aware of the rules to follow and need extra explanation and time to improve their pull requests.

4.8.2.3 Stale Pull Requests

If the maintainer is unable to get a response from a contributor regarding an issue after a repeated attempts over several weeks, the maintainer **may** reassign the associated issue; if there is an associated pending pull request, the maintainer **shall** either close the pull request or take it over (i.e., by using the code in the pull request as the starting point for additional modifications).

4.8.3 Testing

4.8.3.1 Overview

The maintainer **shall** review each contribution to ensure that all tests pass, the contribution works as expected without introducing errors.

4.8.3.2 Automated Testing

The maintainer **should** maintain automated tests to protect against errors.

GitHub Help

GitHub allows maintainers to set up an automated test suite that runs on every pull request and merges into the main branch. Good automated test suites can help catch bugs from going into production and breaking the application.

Example

Is Website Vulnerable finds publicly known security vulnerabilities in JavaScript libraries' websites

4.8.3.3 Failing Automated Tests

If a maintainer receives a pull request that fails automated testing, the maintainer **should** wait a few days to allow the contributor to resolve the issue independently.

If the contributor does not resolve the issue within a few days, the maintainer **shall** contact the contributor to see if they need help.

If the failing test is unrelated to the contributor's changes, the maintainer **shall** let the contributor know that the error is safe to ignore and that it will be fixed in another pull request.

If the contributor fails to respond after repeated attempts over several weeks or months, the maintainer **should** close the pull request and move on.

Note

If multiple contributors fail the same set of tests, the tests may need improvement.

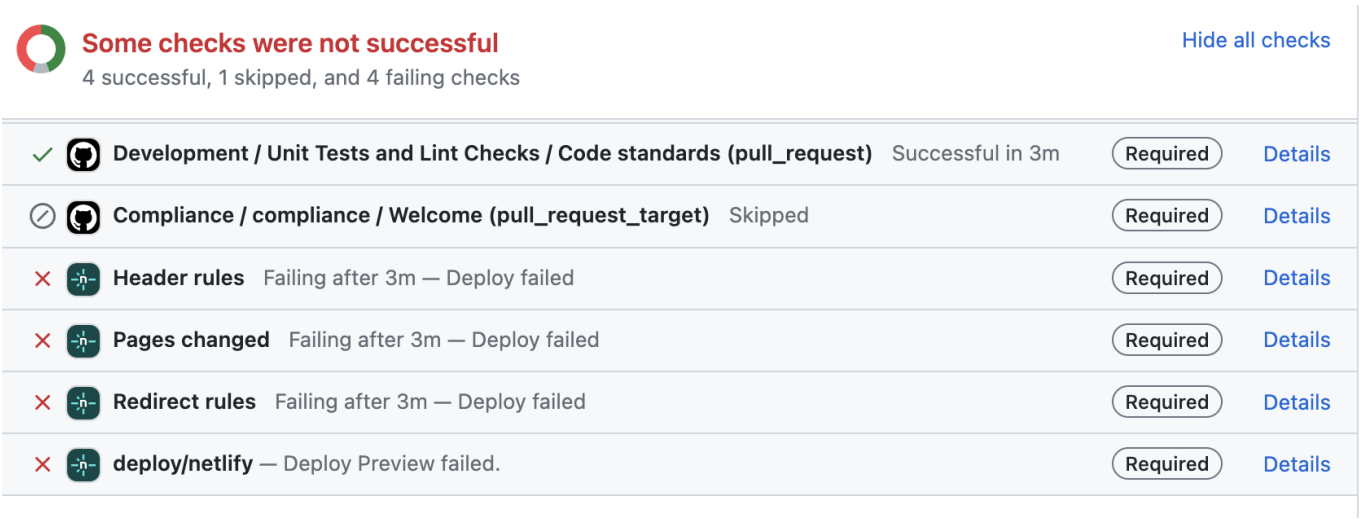

Figure D-1 Failed automated tests

4.8.3.4 Code Reviews

Prior to accepting any pull request, the maintainer **shall** review all proposed changes to ensure that they are designed to address the reported issue as claimed and that they conform to the projects coding and documentation guidelines.

> **Note**
>
> Performing this test before manual testing is useful as it will provide insights into the types of manual tests that are most appropriate.

4.8.3.5 Manual Testing

Prior to accepting a significant pull request, the maintainer **should** manually test the project using his local machine.

> **Note**
>
> If a pull request involves a small change to documentation or code, manual testing is less important, but the maintainer is the last line of defense before a pull request is merged in, which can potentially introduce new issues.

4.8.4 Effective Feedback

If the maintainer discovers a problem with the pull request, the maintainer **shall** respond to the contributor with a detailed account of the problem.

> **Note**
>
> Sometimes, it helps to include a screenshot or screen recording. If the automated tests did not catch the problem, it might be worth creating a separate issue to update the test suite.

The maintainer **shall** indicate the importance of each problem identified in the contribution with `critical`, `high`, `medium`, or `low`.

Note

Feedback provides a public log of how an issue has been addressed and needs to be informative, constructive, and helpful for the reviewer, contributor, and others that might read it at a later date.

Example

High: Please use more descriptive variable names for better readability. For example, revise the code to replace variable `d` with `duration`.

4.8.4.1 Missing Tests

Significant contributions of code **should** be accompanied with tests to help ensure that everything is working as expected.

Note

If the contribution does not include such tests, reach out to them to determine how they tested their contribution and let them know what parts need to be tested.

4.9 Creating a Release

The maintainer **shall** create a pre-release for each version of the document to be reviewed by the responsible WG or committee.

New pre-releases **should** be provided when major changes are made and at least one week prior to WG meetings, unless the WG has established a different notification period.

The maintainer **shall** create a formal release for each version of the document approved for release by the responsible WG or committee.

GitHub Help

Within GitHub, this can be achieved through the **Releases** tab.

The maintainer **shall** assign a tag to the release that indicates the version number per Semantic Versioning 2.0.0.

> **Note**
>
> This produces a version number in the format of `<major>.<minor>.<patch>[-<pre-release>]` format, where
>
> - the `major` number increments (and the other values reset to 0) when non-backwards compatible changes are made,
> - the `minor` version increments (and patch resets) when features are added in a backwards compatible manner,
> - the `patch` increments when backward compatible fixes are made without any new features, and
> - an optional `pre-release` code (preceded by a hyphen) indicates versions under development and must have a sequential alphanumeric identifier

The maintainer **shall** attach a PDF and zip archive of the website in versioned directories in the gh-pages branch.

```
gh-pages/
├── index.html (latest version)
├── v1.0.0/
├── v1.0.1/
├── v1.1.0/
└── v2.0.0/
```

> **GitHub Help**
>
> This can be automated with GitHub actions.

> **Note**
>
> When deploying a new version, ensure the previous version is moved into its own directory before overwriting the index.html and other files for the new release.

Example GitHub Action

```
name: Deploy MkDocs Site and Generate PDF
on:
  push:
    tags:
      - 'v*'

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v3

- name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.x'

- name: Install dependencies
        run: |
          pip install mkdocs-material
          pip install weasyprint

- name: Build MkDocs site
        run: mkdocs build

- name: Generate PDF
        run: |
          weasyprint site/index.html site/docs.pdf
        env:
          WEASYPRINT_BASEURL: 'https://yourusername.github.io/repository-name/'

- name: Deploy to GitHub Pages
        uses: peaceiris/actions-gh-pages@v4
        with:
          github_token: ${{ secrets.GITHUB_TOKEN }}
          publish_dir: ./site

- name: Upload PDF to release
        if: github.ref_type == 'tag'
        uses: actions/upload-artifact@v3
        with:
          name: docs-${{ github.ref_name }}.pdf
          path: site/docs.pdf
```

4.10 Building a Community

4.10.1 Overview

A vital component of any open source project is its community. Building a strong community can help accelerate the growth of your open source project. As new contributors discover and start to contribute to your project, you will want to create spaces for communication and collaboration.

If your project is on GitHub, you can use GitHub Discussions as a way for contributors to post questions and facilitate conversations. WG meetings should be advertised on the discussion board to encourage participation.

4.10.2 Promptly Respond and Address Concerns

Maintainers **should** establish a schedule to review incoming issues and pull requests and post this schedule within the discussion forum.

> **Note**
>
> It is important to set expectations as to how fast maintenance issues are likely to be addressed. This timeline might vary considerably across ITS open-source projects and during teh lifetime of any one project (e.g., there are often periods of more active development and less active development).

The maintainer **should** set up GitHub Actions to automate responses that welcomes each new contributor and provides an estimated time by which the issue triage or PR review can be expected

> **Note**
>
> You can learn how to set one up inthis article.

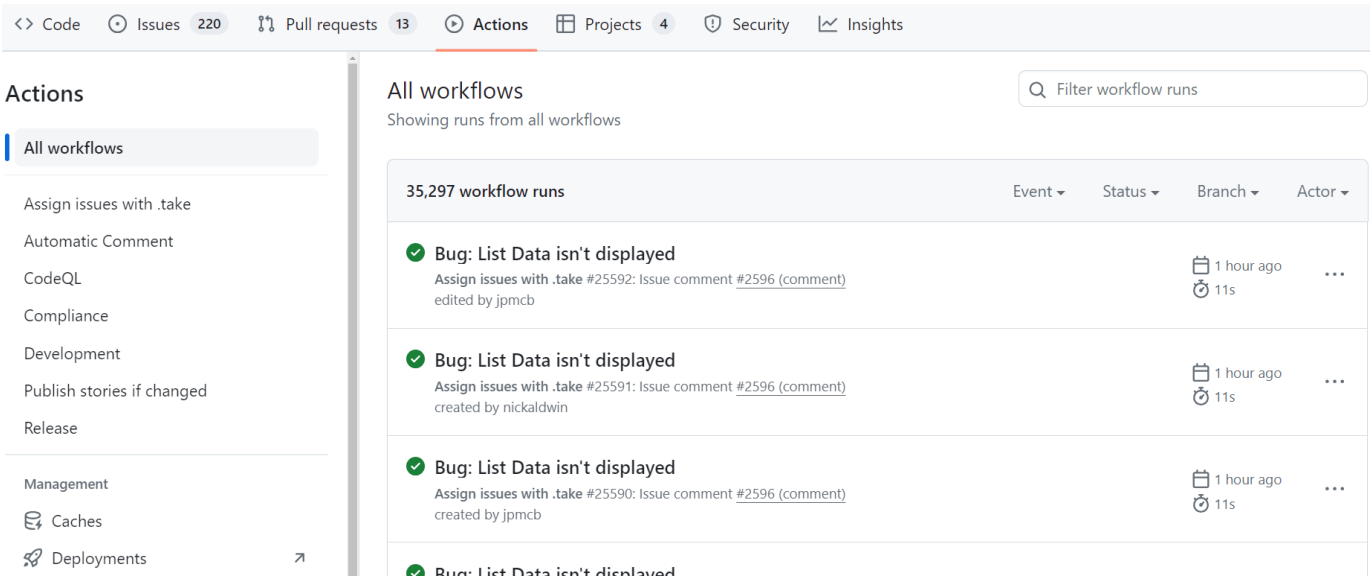

Figure D-1 Create GitHub Action

4.11 Advanced Features

4.11.1 Overview

Leveraging GitHub Actions to bring Continuous Integration / Continuous Delivery or Deployment (CI/CD) into your workflow directly in your repository will let you run code, test, build, and deliver or deploy software with simple and secure workflows. Automating these tasks will speed up your deployment process.

Using Git, GitHub, and GitHub Actions to build a CI/CD pipeline should give you confidence in your code.

Below are some helpful resources to help you build a CI/CD pipeline with GitHub Actions:

- GitHub Docs: The complete CI/CD solution
- How to build a CI/CD pipeline with GitHub Actions in four simple steps

There are many types of actions that you can set up for your project, depending on what you need. Below are some GitHub Actions that you usually find across repositories:

4.11.2 Linter

Most open source repositories have linters that run on each pull request. Linter is a tool for detecting potential errors and maintaining a consistent code style in a project. Super-Linter is one of the most used actions. This action can help you maintain code quality and achieve a more readable and consistent style.

4.11.3 Code Scanning Tools

Code scanning is a tool for detecting security vulnerabilities, possible bugs, and errors in code. You can use GitHub's code scanning feature and configure tools like CodeQL, which GitHub maintains, or third-party scanning tools such as SonarQube.

4.11.4 Creating and Customizing Actions

Actions of note:

- GitHub Marketplace
- Take Action: allows contributors to assign themselves to an issue by typing the `.take` command in the issue's comment.
- Triage Action: blocks the Take Action whenever a `needs triage` or `core team work` label exists.

You can read more about GitHub Actions and how to create one in the official documentation.

4.11.5 Projects

Keeping track of your issues is getting more challenging as your project progresses. A great tool that can help you organize and track your issues is Projects on GitHub. With projects, you can efficiently manage your project's features, roadmaps, or releases as they're built from and integrated with issues and pull requests that you add.

You can choose a template for your project. One of the templates is the "Kanban" template. Here, you can create notes and place the existing issues and pull requests in the "Backlog",

"Ready", "In progress", "In review", and "Done" columns. This will make it easy for you and your team to track the progress of your project.

To create a new project:

1. Navigate to your GitHub profile and click the "Projects" tab.
2. Click the green "New project" button.
3. Choose a template.
4. Name your project and click the "Create project" button at the bottom.

Please read the GitHub documentation to learn more about adding items to your project.

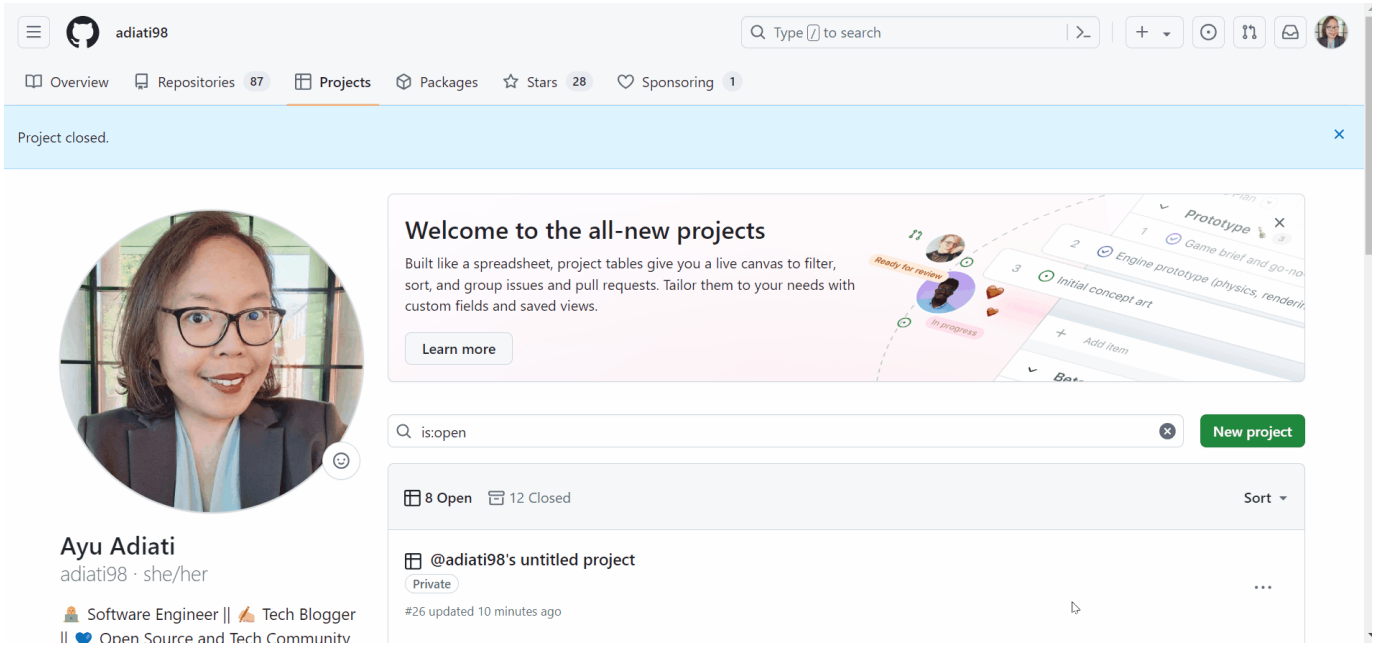

Figure D-1 GitHub Project Boards

July 23, 2025

Section 4 WG Responsibilities

4.1 Overview

Each major stage of the open-source process is reviewed by a WG or committee to ensure a base level of consensus. The specific group that is required to provide consensus and the level of consensus required dependent upon the standardization path adopted for the project.

Example

An NTCIP experimental specification can be approved at the NTCIP WG level for all stages while an NTCIP standard requires Joint Committee approval for the project approval and release approval.

The stages within the open-source process include:

- project approval
- issue prioritization
- pull-request approval
- release approval

Committee Requirement

When starting a project, the committee shall:

- Identify a responsible working group
- Identify one or more maintainers

Update style

WG Requirement

The WG shall define the time within which the maintainer is expected to triage and respond to submitted comments.

4.2 Project Approval

An appropriate WG or committee **shall** approve the formation of a project prior to establishing the SDO GitHub repository for the project.

The appropriate WG or committee **should** be identified in policies adopted by any SDO adopting the ITS Open-Source Process.

Note

A contributor can establish their own GitHub repository for the project before formal approval to allow WG members to gain a better idea of what is being proposed.

NTCIP Guidance

NTCIP 8001 identifies the appropriate WG or committee for NTCIP open-source projects.

4.3 Project Tailoring

The WG **shall** tailor parameters for the project, including:

1. the project schedule
2. the time within which the maintainer is expected to triage and respond to submitted issues and discussion items
3. whether the project is to use the `mike` version control system and, if so, which versions are to be maintained within this system

Note

Standards should always use a version control system to maintain historic versions of a standard. However, this process can also be used to maintain informative websites that might not need to provide historic versions.

Example

The list of versions maintained can include: - every release approved by the WG (with latest pointing to the most recently approved version) - every pre-release since the last release - other designated interim releases (e.g., UCD versions)

4.4 Issue Prioritization

A WG **should** oversee the prioritization of significant issues for each of its open-source projects.

A WG **may** provide guidance to its maintainer as to what constitutes a significant issue and how various issues should be handled.

Note

Many issues can be prioritized by the maintainer without involving the WG; however, when major issues arise that affect the direction of the project, it is best to obtain direction from the WG to ensure resources are managed properly.

4.5 Pull-Request Approval

The WG responsible for the open-source project **shall** approve each pull request prior to its merge into the SDO repository.

The WG responsible for the open-source project **shall** establish its policies on what constitutes a pull-request approval.

Note

A pull-request approval typically requires simple majority with no sustained objections.

GitHub Guidance

This can be achieved by requiring a minimum number of approvals within GitHub among a designated set of voting members.

4.6 Approve Releases

The WG responsible for the open-source project **shall** approve each version of a project prior to it being tagged as a release.

The WG responsible for the open-source project **shall** establish its policies on what constitutes a release approval.

Note

Releases can be made from any branch, not just the main branch.

Example

A release approval can be as simple as WG consensus or can require a formal ballot according to the processes adopted by the full committee.

July 17, 2025

Annex D Contributor Covenant Code of Conduct

Each entity that participates in the development of this repository as a commenter, contributor, maintainer, or manager agrees to encourage a harassment-free environment and to act and interact in ways that contribute to an open, welcoming, and healthy community.

D.1 Scope

This Code of Conduct applies within the scope of GitHub, and also applies when an individual is officially representing the community in public forums.

D.2 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders responsible for enforcement at ntcip@nema.org.

D.3 Details

For additional guidelines on the application of this code, see the Contributor Covenant.

D.4 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 2.1.

July 17, 2025

Annex D Documentation Conventions

D.1 Exceptions Allowed

Unless otherwise stated in the project-specific CONTRIBUTIONS.md file, each project based on this specification **shall** develop documentation as defined by this annex.

D.2 Development Environment

D.2.1 Overview

In addition to the development tools needed to manage and submit any contribution within the Git environment (e.g., Git, GitHub), developing project documentation requires the following tools:

- **A text editor,** which is used to create and edit markdown and yaml files,
- **Python,** which is required to run MkDocs,
- **MkDocs,** which is an open-source tool for translating a set of markdown files into a static website, and
- **Materials for MkDocs,** which is an open-source tool that extends the markdown language to support additional features that are useful for developing the look and feel of the project's documentation.

This combination of tools has been selected because it:

- is designed to be easy to install and use,
- requires minimal setup,
- works well with Git and GitHub,
- supports search functionality,
- can produce a static website,
- when coupled with add-ons, can produce PDFs
- has an active development community

It is recommended to establish this development environment prior to making any edits. Generating the documentation website locally from a known baseline allows the contributor to verify that the development environment is working correctly prior to introducing edits to the files. Contributors are required to generate the documentation locally to verify that their proposed changes do not introduce any errors to the project. The MkDocs development

environment allows users to see their changes in real time so that any errors can be addressed quickly.

D.2.2 Text Editor

Any text editor can be used to produce markdown and yaml files. These files are to have the extensions of ".md" and ".yml", respectively.

Contributors are encouraged to use Visual Studio Code, which is an open-source editor, with the following extensions enabled as it provides a reasonably close rendering of the final display format:

- Markdown Preview Enhanced by Yiyi Wang, this extension provides a markdown previewer with support for diagrams, math (LaTeX), mermaid, charts, and more;
- markdownlint by David Anson, this extension assists in ensuring markdown files follow consistent formatting rules; and
- YAML by Red Hat, this extension provides syntax highlighting, validation, and autocomplete for YAML files.

While any text editor can be used, this suite of tools offers a free solution that is designed to render the markdown in real-time while assisting the user in producing high quality code. However, users should be aware that the toolset still does not attempt to render some of the more advanced features of Materials for MkDocs. The final look and feel can be obtained using the MkDocs server.

D.2.3 Python

MkDocs requires Python 3.8 or higher. You can check to see if Python is already installed and its version with the following command:

```
python --version
```

The most recent version of Python can be installed from official Python website.

Once installed, you should verify by running both the `python --version` and `pip --version` commands. PIP should be installed as a part of the Python package.

D.2.4 MkDocs

Running the MkDocs server locally allows the contributor to see proposed changes in real-time and test them thoroughly prior to submitting pull requests. To install MkDocs, run

```
pip install mkdocs
```

Once installed, verify its installation with:

```
mkdocs --version
```

Once you have verified the installation, start the MkDocs server by changing to the directory containing your cloned copy of the project repository and running

```
mkdocs serve
```

Once the server is running, you can direct a web browser to localhost port 8000 to see the development version of the website. This site will be updated in realtime as you update files in the repository. If you want to create a static site, However, to render all elements within the project correctly, you will need to install Materials for MkDocs.

D.2.5 Materials for MkDocs

To install Materials for MkDocs and the commonly used extensions for ITS projects, run the following command:

```
pip install mkdocs-material pymdown-extensions
```

D.3 Working with the Content

The content of ITS open-source documentation is generally written in Markdown, a lightweight and easy-to-use markup language that allows you to format text in a readable and visually appealing way.

Please read the "Frequently Used Markdown" section for details about how to use it in this project.

D.3.1 Default Document Structure

ITS open-source projects can cover a range of projects that have wildly different documentation needs. Each project is allowed to define its own structure, but unless otherwise specified **shall** use the structure defined in this document, which is intended for projects that result in a product that can be conceptualized as a single traditional document (e.g., a traditional standard).

Each major portion of the document **shall** be defined in a separate markdown file. Major portions are defined as:

- the title page, which shall be `index.md`;
- each top-level section of the front matter (e.g., Foreword, Introduction);
- each section in the body of the document; and
- each annex.

The document structure **shall** be reflected in the project's `mkdocs.yml` file under the `nav` section with all front matter located under a `Front Matter` heading.

```
  - Front Matter:
    - Title Page: index.md
    - Notices: notices.md
    - Acknowledgements: acknowledgements.md
    - Foreword: foreword.md
    - Introduction: introduction.md
  - 1 General: general.md
  - 2 Overview: overview.md
  - 3 Commenter Responsibilities: commenter-responsibilities.md
  - 4 Contributor Responsibilities: contributor-responsibilities.md
  - 5 Maintainer Responsibilities: maintainer-responsibilities.md
  - 6 WG Responsibilities: wg-responsibilities.md
  - A Code of Conduct: code-of-conduct.md
  - B Documentation Conventions: documentation-conventions.md
  - C Coding Conventions: code-quality.md
```

Note

When using the default configuration for ITS projects, this results in a left-hand left-hand navigation bar that shows the major portions of the document while the right-hand navigation shows the content of the currently opened section.

Note

Be sure to follow naming conventions. Notice that file names are not capitalized, and there are hyphens in place of spaces between words.

D.3.2 Structure of the Title Page File

The `index.md` file **shall** represent the title page of the document and **shall**:

- Start with a line containing a hashtag and nothing else
- Include code that suppresses unwanted markdownlint warnings
- Identify the status of the document
- Define the Document Identifier (e.g., NTCIP X8008)
- Define the Document Title (e.g., ITS Open-Source Process)
- Any other information required by the Standards Development Organization (SDO)

Example of a Title Page File

```
#
<!-- markdownlint-disable MD033 -->
<div style="text-align: center; font-style: italic; font-weight: bold;">
  A proposal to the NTCIP Joint Committee</div>
<div style="text-align: center; font-size: 1.5em; font-weight: bold;">
  NTCIP X8008
</div>
---
<div style="text-align: center; font-size: 1.5em; font-weight: bold;">
  National Transportation Communications ITS Protocol
</div>
<div style="text-align: center; font-size: 2em; font-weight: bold;">
  ITS Open-Source Process
</div>
<!-- markdownlint-enable MD033 -->
```

D.3.3 Structure of All Other Front Matter Files

Each file representing a major portion of the front matter, other than the title page, **shall** include a single level 1 heading that has the same title as defined in the `nav` section of the `mkdocs.yml` file and is the first line of the document

Example of Start of a Front Matter File

```
# Foreword
```

D.3.4 Structure of a Section File

Each file representing a section of the main body of the document **shall**:

- Start with code that sets the section counter for the body to the section number while suppressing unwanted markdownlint warnings
- Include a single level 1 heading that has the same title as defined in the `nav` section of the `mkdocs.yml` file and occurs immediately after the code defining the section number
- End each heading with `{.body}`

Example of Start of a Section File

```
<!-- markdownlint-enable require-heading-annex -->
<div markdown="1">
<style>
    :root { --section-number: 2; --section-style: decimal; }
</style>
```

Note

Rule MD033 of markdownlint does issues a warning about the use of HTML within markdown, but it is is necessary in this case to allow automated numbering to work properly.

Rule MD041 of markdownlint indicates that the first line in a file should be a top-level heading, but our convention requires defining the section number first.

D.3.5 Structure of an Annex File

Each file representing an annex of the document **shall**:

- Start with code that sets the section counter for the annex to the numerical order of the annex (the script will transform this into an alphabetic letter)
- Include a single level 1 heading that has the same title as defined in the `nav` section of the `mkdocs.yml` file and occurs immediately after the code defining the section number
- End each heading with `{.annex}`

Example of Start of an Annex File

```
<!-- markdownlint-enable require-heading-annex -->
<div markdown="1">
<style>
    :root { --section-number: 2; --section-style: upper-alpha; }
</style>

# Example Annex {.annex}
```

D.3.6 Adding Definitions to the Glossary

If you add definitions to the project's glossary, ensure the definitions are added **alphabetically**.

D.3.7 Frequently Used Markdown

D.3.7.1 Headings

The hash (`#`) symbol at the start of a line denotes a heading (e.g., section, clause, subclause). There are six levels of headings, and the number of hash symbols indicates the heading level. The title of the heading should appear after the hash symbols and a space.

Example

```
### Heading 3

#### Heading 4
```

D.3.7.2 Text Formatting

- Make text bold by enclosing it with double asterisks (`**`).
- Make text italic by enclosing it with single underscores (`_`).
- Create inline code by wrapping text with backticks (`` ` ``).

Example

```
**This is a bold text.**

_This is an italic text._

This is an`inline code`.
```

This is a bold text.

This is an italic text.

This is an `inline code`.

D.3.7.3 Lists

- Create ordered lists using numbers followed by a period (`1.` , `2.` , etc.).
- Create unordered lists using hyphens (`-`).
- The line before a list must be blank and a list cannot be immediately preceded by a different list
- The style for the list is defined by the first list item

Example

```
1.  Item 1
2.  Item 2

New List

- Unordered Item 1
- Unordered Item 2
```

1. Item 1
2. Item 2

New List

- Unordered Item 1
- Unordered Item 2

Note

The numbering of numbered lists is automatic within markdown (i.e., when rendered, the list items are numbered sequentially from 1 regardless of what numbers are contained within the markdown file); however, it is good coding practice to maintain the correct numbering within the markdown file to prevent any confusion among contributors.

D.3.7.4 Links

Create links using square brackets (`[]`) for the link text and parentheses (`()`) for the URL.

Example

```
[NTCIP](https://ntcip.org)
```

NTCIP

D.3.7.5 Images

Embed images using an exclamation mark (`!`), followed by square brackets (`[]`) for the alt text, and parentheses (`()`) for the image URL. AN optional attribute field can be added to the end to specify the size.

Example

```
![NTCIP](_assets/images/NTCIP.jpg){ width=200px }
```

D.3.7.6 Block quotations

Create a blockquote using the greater-than symbol (`>`) or through Materials for MkDocs' admonition quote (`!!! quote`).

Example

```
> This is a blockquote.

!!! quote
    This is a Materials for MkDocs admonition quote.
```

> This is a blockquote.

Quote

This is a Materials for MkDocs admonition quote.

D.3.7.7 Code Blocks

Create code blocks using triple backticks (```) for fenced code blocks and specify a language next to the backticks before the fenced code block to highlight the syntax.

Example Example code

```
bash git pull
```

```
``` bash
bash git pull
```
```

D.3.7.8 Admonitions

Create callout out blocks for different purposes using the Materials for MkDocs admonitions feature by including three explanation points and the admonition type with the contained text indented by four spaces (`!!! note`)

Example

```
!!! note
    This is a note.

!!! tip
    This is a tip.
```

Note

This is a note.

Tip

This is a tip.

Materials for MkDocs supports the following standard admonitions:

- abstract
- bug
- danger
- example
- failure
- info
- note
- question
- quote
- success
- tip
- warning

D.3.8 Markdown Tips

- Preview your Markdown locally to ensure proper formatting before submitting your contribution.
- Keep your Markdown content organized, and use headings to structure your sections.
- There should be exactly one `heading 1` within each file.
- Use code blocks to highlight code snippets or configuration examples.
- See the official Markdown Guide for more information about Markdown.
- See the Materials for MkDocs Guide for more information about Materials for MkDocs.

July 17, 2025

Annex D Coding Conventions

D.1 Python Coding Conventions

Each contributor **shall** adhere to style guidelines defined in Python Enhancement Proposals (PEP) 8 – Style Guide for Python Code.

Highlights of PEP 8

- Imports should be at the top of the file
- Imports should be grouped into three sections with a blank line between each: (1) standard library imports, (2) thirdd party library imports, and (3) local imports
- Function and variable names should be in lowercase_with_underscores
- Class names should be in UpperCamelCase
- Constants should be in ALL_CAPS_WITH_UNDERSCORES
- Do not use tabs; use four spaces for each indentation level
- Limit lines to 79 characters; or 72 characters for long comments
- Separate top-level functions and class definitions with two blank lines
- Inside functions, use one blank line to separate significant logical sections

Each contributor **should** use a linter to automatically enforce the PEP 8 rules.

Example

Pylince

July 17, 2025

Annex D Examples for Material for MkDocs

Warning

Interactive features have limitations when the site is rendered as a PDF.

D.1 Call-out Blocks

D.1.1 Code blocks with syntax highlighting

Code blocks allow a user to define a block of text that is called out to appear as computer code in a specified language. Material for MkDocs includes an extension to support a wide range of syntax highlighters (i.e., to colorize keywords) and also allows custom-defined syntax highlighters for user-defined languages. Code blocks start with three tick marks followed by a space and then an indication of the syntax highlighter to be used. The block of code is indented with four spaces and the block ends with another three tick mark code.

C++ Example C++ Example Code ASN.1 Example ASN.1 Example Code

```
for(i = 0; i < max; i++) {
    // sample loop code
}

    ``` c++
    for(i = 0; i < max; i++) {
        // sample loop code
    }
    ```

SEQUENCE OF {
    item1 INTEGER (0..255),
    item2 OCTET STRING
}

    ``` asn1
    SEQUENCE OF {
        item1 INTEGER (0..255),
        item2 OCTET STRING
    }
    ```
```

D.1.2 Admonitions

Material for MkDocs supports creating call-out boxes for notes, examples, questions, information, etc. It calls these boxes "admonitions". They are represented in a similar way to

code blocks but start with three exclamation points (!) followed by a space and then the type of admonition.

Note Example Code for Note Warning Example Code for Warning

Note

Material for MkDocs allows users to define their own admonition types as well.

```
!!! note
    Material for MkDocs allows users to define their own admonition types as well.
```

Warning

The type of admonition defines the color and icon used in the banner of the box.

```
!!! warning
    The type of admonition defines the color and icon used in the banner of the box.
```

D.1.3 Collapsible

Material for MkDocs also allows call-out boxes to be collapsible by using question marks instead of the exclamation points.

Note Example Code for Note

Note

Material for MkDocs allows users to define their own admonition types as well.

```
```

Note

```
Material for MkDocs allows users to define their own admonition types as well.
```

```

Warning Example Code for Warning

Warning

The type of admonition defines the color and icon used in the banner of the box.

```

Warning

```
The type of admonition defines the color and icon used in the banner of the box.
```

```

Note

When rendered to a PDF, a collapsible box is always shown expanded, but includes a downward arrow (v) in the title bar.

D.1.4 Content Tabs

As shown in these examples, boxes can also have multiple tabs. This is achieved by using three equal signs (=).
```

Example

Tab 1 Tab 2 Code

Example

This is an example using tabs.

Note

The code tab only shows the first two tabs to avoid recursive code.

```
=== "Tab 1"
    !!! example
        This is an example using tabs.

=== "Tab 2"
    !!! note
        The code tab only shows the first two tabs to avoid recursive code.
```

Tip

When rendered to a PDF, the tabs are shown across the top but the content of each tab is displayed in order (with no real distinction between the content of each tab).

D.2 Annotations

If there is a preference to have comments appear by the user clicking and seeing a tooltip, Material for MkDocs also supports annotations

Sample annotation Code for annotation

Clicking on this (1) icon will show more text

1. More text

```
Clicking on this (1) icon will show more text
{ .annotate }

1. More text
```

Warning

When rendered to a PDF, the annotation is rendered largely as the markdown text, minus the `{ .annotate }` line.

D.3 Footnotes

Footnotes are similar to annotations but place the additional information at the bottom of the page rather than as a tooltip that appears.

Sample footnote Code for annotation

Clicking on the superscripts[1] will jump to the footnote[2]

```
Clicking on this[^1] icon will show more text[^2]

[^1]: Short note on one line
[^2]:
    Long footnotes must start on the following line and be indented by four
    spaces. Clicking on the icon at the end of the footnote will cause the
    display to jump back to the location of the footnote in the text.
```

Tip

When rendered to a PDF, the footnote is rendered at the end of the file (e.g., section) where the footnote appears.

D.4 Abbreviations / Glossary

Tooltips can also be used to display term definitions or meanings of abbreviations (abbr). For one-off usage, the file simply includes a line (typically at the end) the indicates the term in square brackets preceded by an asterisk and followed by a colon space and the definition. The line defining the term is not rendered, but the term being defined (e.g., abbr) will be underlined wherever it occurs in the document and hovering over any instance of the term will reveal its definition in a tooltip. By using the auto_append feature, all term definitions can be moved to a separate file and applied to all pages within the project.

Code for defining abbr mkdocs.yml

```
*[abbr]: abbreviation
```

```
markdown_extensions:
  - pymdownx.snippets:
      auto_append:
        - includes/abbreviations.md
```

Warning

When rendered to a PDF, the information in the tooltip is not included in the document.

D.5 Paragraph attributes

The Attribute Lists extension allows to add HTML attributes and CSS classes to almost every Markdown inline- and block-level element with a special syntax.

For example, this document marks all headings with the .annex class. This applies the annex style from the extra.css file so that the heading is preceded with a section number that starts with a letter.

Example

Example heading Code for example

D.5.1 My heading

```
### My heading {.annex}
```

D.6 Sortable tables

Standard markdown supports tables; Material for MkDocs allows for extending this feature to allow for sortable tables with some edits to the mkdocs.yml file and a javascript.

Sortable table mkdocs.yml docs/javascripts/tablesort.js Sortable table code

Group	Title
WG 1	Architecture
WG 3	ITS geographic data
WG 5	Fee and toll collection
WG 7	General fleet management and commercial/freight
WG 8	Public transport/emergency
WG 9	Integrated transport information, management and control
WG 10	Traveller information systems
WG 14	Driving automation and active safety systems
WG 16	Communications
WG 17	Nomadic Devices in ITS Systems
WG 18	Cooperative systems
WG 19	Mobility integration
WG 20	Big Data and Artificial Intelligence supporting ITS

```
extra_javascript:
  - <https://unpkg.com/tablesort@5.3.0/dist/tablesort.min.js>
  - javascripts/tablesort.js
```

```
document$.subscribe(function() {
  var tables = document.querySelectorAll("article table:not([class])")
  tables.forEach(function(table) {
    new Tablesort(table)
  })
})
```

```
|Group | Title                 |
|:----:|:----------------------|
|WG 1  | Architecture          |
|WG 3  | ITS geographic data   |
|WG 5  | Fee and toll collection |
|WG 7  | General fleet management and commercial/freight|
|WG 8  | Public transport/emergency |
|WG 9|Integrated transport information, management and control|
|WG 10|Traveller information systems|
|WG 14 | Driving automation and active safety systems|
|WG 16 | Communications |
|WG 17 | Nomadic Devices in ITS Systems |
|WG 18 | Cooperative systems |
|WG 19 | Mobility integration |
|WG 20 | Big Data and Artificial Intelligence supporting ITS |
```

> **Warning**
>
> When rendered to a PDF, the ability to sort table rows is lost.

D.7 Figures

D.8 Generic Figures

Figures are can be included on a page by using the code format `![Example Title](example-file.svg)`. To ensure that the figure is centered, it can be wrapped in a `figure` element (e.g., `<figure>![Example Title](example-file.svg)</figure>`).

D.8.1 Mermaid diagrams

Material for MkDocs supports Mermaid diagrams. Mermaid allows for relatively simple text-based statements to define diagrams that follow well-defined rules, such as UML diagrams, block diagrams, etc.

Example

Sequence diagram Sequence diagram code

```
%%{init: { 'sequence': { 'mirrorActors': false } }}%%
sequenceDiagram
  participant Proposer
  participant Committee
  participant WG as Working Group
  participant Maintainer
  participant Repo as Open-Source Project Repository

  Proposer ->> Committee: Propose project
  Committee ->> WG: Establish WG
  Committee ->> Maintainer: Assign maintainer
  Maintainer ->> Repo: Establish public repository
  Maintainer ->> Repo: Upload initial baseline
  Maintainer ->> WG: Suggest project plan
  WG -->> Maintainer: feedback
  Maintainer ->> Repo: Post project plan
  Maintainer ->> Repo: Create appropriate branches for work
```

Figure D-1 Example sequence diagram

````
    ```mermaid
    %%{init: { 'sequence': { 'mirrorActors': false } }}%%
    sequenceDiagram
      participant Proposer
      participant Committee
      participant WG as Working Group
      participant Maintainer
      participant Repo as Open-Source Project Repository

      Proposer ->> Committee: Propose project
      Committee ->> WG: Establish WG
      Committee ->> Maintainer: Assign maintainer
      Maintainer ->> Repo: Establish public repository
      Maintainer ->> Repo: Upload initial baseline
      Maintainer ->> WG: Suggest project plan
      WG -->> Maintainer: feedback
      Maintainer ->> Repo: Post project plan
      Maintainer ->> Repo: Create appropriate branches for work
    ```

    <figure><figcaption class="annex">Example sequence diagram</figcaption></figure>
````

> **Warning**
>
> With the current PDF generator, Mermaid diagrams need to be converted to SVGprior to including in the file.

D.8.2 Figure Caption

Figures should be followed with a caption. Captions can be added with the following code: `<figure><figcaption>Sample caption</figcaption></figure>`. The `figcaption` element will ensure that the figure is preceded with the correct section-figure number (e.g., `Figure 3-6:`).

D.8.3 Referencing Figure Numbers

The next figure number can be included in the text preceding the figure with the following code: `<next-fig/>`.

D.9 Additional features

D.10 Search

The search feature adds a search field into the page header. Include by including the following in your mkdocs.yml file.

```
plugins:
  - search
```

D.11 Comment System

Material for MkDocs allows to easily add the third-party comment system of your choice to the footer of any page by using theme extension.

D.12 Fields for information from GitHub

When using the `mkdocs-git-revision-date-localized` plugin, users can show the release number on the index page using the field `{{ release_number }}`.

D.12.1 Version history

Material for MkDocs has a powerful versioning system that allows a site to maintain a history of all released versions of a document.

D.12.2 Last edit date for each page

The `mkdocs-git-revision-date-localized-plugin` for Material for MkDocs. An example of this appears at the bottom of this page and is enabled by ensuring the git-revision-date-localized feature is listed in the plugins section of your mkdocs.yml file.

---

1. Short note on one line ↩
2. Long footnotes must start on the following line and be indented by four spaces. Clicking on the icon at the end of the footnote will cause the display to jump back to the location of the footnote in the text. ↩

July 23, 2025