CMS OSPO Outbound Review Checklist

Tier 4: Community Governance

Instructions

This is a review process to approve CMS-developed software to be released open source. If you would like your repository to be released, please complete the following steps.

Prerequisites

State the Benefits of Open Sourcing the Project

State the Risks of Open Sourcing the Project

Questions

Code Review

Code Analysis

Review Licensing

Review Commit History

Review Repository Hygiene

Review Project Metadata

Review Repository Details

Review OpenSSF Scorecard

Additional Notes & Questions

Sign off on risk acceptance of open-sourcing the software product

Flipping the Switch: Making the Repository Public

Prerequisites

Does your repository align with the requirements of a Tier 4 project? To verify:

- Review the flowchart or use `tier-determiner.py` provided in the README.md.
- Read more about Tier 4 and the overall CMS OSPO maturity model framework.

Results

Insert Review Here

State the Benefits of Open Sourcing the Project

- [ ] **Cost Savings**

  By making the project freely available, this reduces licensing and acquisition costs.

- [ ] **Ease of Repurposing**

  The open nature of the code allows users to modify and adapt the software to suit their specific needs, fostering customization and flexibility.

- [ ] **Minimization of Vendor Lock-in/Flexibility of Vendor Choice**

  Users are not tied to a single vendor, providing the freedom to choose between different vendors.

- [ ] **Enable Transparency**

  The source code is accessible and visible to anyone, promoting transparency in how the software functions, which helps build trust.

- [ ] **Enable extension/extensibility**

  Users can extend and enhance the functionality of the software by adding their own features.

- [ ] **Increase Interoperability**

  Planning in the open enables future compatibility and interoperability between different systems and software applications.

- [ ] **Facilitate Experimentation/Early Adoption**

  Working in the open encourages experimentation and early adoption of cutting-edge technologies, leading to faster innovation and improvement in software capabilities.

State the Risks of Open Sourcing the Project

- [ ] **Security Risks**

  Vulnerabilities may be exposed if the code is not thoroughly reviewed, potentially leading to security breaches or exploitation. (See: SECURITY.md) Does this project connect to any CMS-internal only systems? Does this project require authorization or authentication to operate? Does this project detail any non-public directories of CMS/HHS systems or people?

- [ ] **Financial Risks**

  Costs may arise from maintaining code, community engagement, addressing security concerns, subscription costs hardware costs, specialized tooling or infrastructure costs. Does this project require any ongoing financial costs or subscription fees? (e.g., Cloud Hosting, Specialized build systems, paid maintainers, paid libraries or dependencies.)

- [ ] **Privacy Risks**

  Does this project require access to non-public, non-synthetic PII, PHI, or other internal-only CMS systems containing such data or information?

Questions

- Does the code contain or touch any private information such as Personal Identifiable Information (PII) or Protected Health Information (PHI)?
    - Can it be removed? Is it absolutely needed to function? Can it be shipped with synthetic data instead?
- Does the code interface with any of CMS' internal-only systems (e.g. mainframes, JIRA instances, databases, etc…)?
- Does the repository contain any keys or credentials to access or authenticate with CMS' systems?
    - Can it be removed or is it needed?

If you answered "yes" to any of the above questions, your project may be 'sensitive' in nature and require a more thorough review before sharing publicly. Please reach out to opensource@cms.hhs.gov for guidance. If you answer yes to any of these questions above, it is best to NOT open source the product due to security reasons.

Results

Insert Review Here

Code Review

The existing codebase should be given a one time, top-to-bottom code quality and security vulnerability review by two (or more) engineers who have written production code within the past two years, in the languages used in the project. Engineers should review credential management practices with the development team to ensure that any keys, passwords, or other sensitive configurations are not checked into source code or in the git history.

The engineers can be federal government employees or trusted partners from outside the agency from other contracts, or from independent testing contracts. Their names, organizations, comments and approval/disapproval on the overall codebase should be tracked in this document.

To provide independent review, the engineers should not have been involved in the development of the software product. This includes engineers who wrote part of the software or who directly provided technical direction and oversight in the creation of the software.

As part of the code review, engineers should reference modern listings of the most significant software security vulnerabilities. For instance, an acceptable description would be that the engineers showed how they used automated tools and manual review to check each item in OWASP's current 10 Most Critical Web Application Security Risks.

Results

Insert Review Here

Code Analysis

At least one automated tool for code analysis (such as static code analysis tools) has been run on the codebase to look for security vulnerabilities, and results have been appropriately acted upon. Even if all findings are eventually fixed, if the initial scans revealed significant, severe vulnerabilities (such as SQL injection vulnerabilities), this indicates that the software development team may not be adhering to the best practices required for open source public release.

Automated tooling for code analysis should be incorporated as a regularly scheduled part of the application development process. The development team should briefly document how frequently they commit to running these automated scanning tools, and who will be running the tests, interpreting, and acting upon the results.

Toolkit

Below is a list of suggested tools to run for code analysis:

Tool	Description	Link
Repo Linter	Lint repositories for common issues such as missing files,etc...	https://github.com/todogroup/repolinter
Gitleaks	Protect and discover secrets using Gitleaks	https://github.com/gitleaks/gitleaks Use and run gitleaks.yml provided in repository
git filter-repo	Entirely remove unwanted files / files with sensitive data from a repository's history	https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/removing-sensitive-data-from-a-repository
bulk_extractor	Check for secrets, URLs, emails, etc.	bulk_extractor Official Documentation OSPO Guide Documentation

Results

Insert Review Here

Review Licensing

Ensure that acceptable licensing is decided for the project. Most often, software released as open source by the federal government does so under the Creative Commons Zero 1.0 license.

Suggested licensing:

Public Domain

This project is in the public domain within the United States, and copyright and related rights in the work worldwide are waived through the CC0 1.0 Universal public domain dedication.

All contributions to this project will be released under the CC0 dedication. By submitting a pull request, you are agreeing to comply with this waiver of copyright interest.

If your project is not being dedicated to the public domain under CC0, due to being work for hire, or some other documented reason, then choosing another OSI approved license is the next best thing.

Results

Insert Review Here

Review Commit History

Review the history of commits to the version control system used, and whether the team prefers to clean (e.g., rebase) this history before releasing to the public.

If not rebasing, verify that:

1. There are no obscene or impolite remarks in comments or commit history
2. There are no sensitive internal URLs/IP Addresses in comments or commit history
3. There are no credential files such as Passwords, API/SSH/GPG keys checked into the repo.

Consider using the following tools to perform the tasks above:

Tool	Description	Link
gitleaks	Open source tool that detects and prevents secrets (passwords/api/ssh keys) checked-in to your git repo	https://github.com/gitleaks/gitleaks What is git leaks and how to use it? Use and run gitleaks.yml provided in repository
git filter-repo	Entirely remove unwanted files / files with sensitive data from a repository's history	https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/removing-sensitive-data-from-a-repository
bulk_extractor	Check for secrets, URLs, emails, etc.	bulk_extractor Official Documentation OSPO Guide Documentation

Results

Insert Review Here

Review Repository Hygiene

As part of our repository hygiene requirements, the project must include certain files and sections. Using repolinter will help you identify missing files and content that will need to be added to your repository before outbounding.

Running repolinter on your repository locally

1. Add repolinter.json to the root directory of your project

2. Run command:

```
repolinter lint
```

3. The result produces a list of file and section existence checks, indicating whether each requirement was met or not.

```
✔ license-file-exists: Found file (LICENSE.md)
✔ security-file-exists: Found file (SECURITY.md)
✔ readme-file-exists: Found file (README.md)
✔ contributing-file-exists: Found file (CONTRIBUTING.md)
✔ maintainers-file-exists: Found file (MAINTAINERS.md)
✖ codeowners-file-exists: Did not find a file matching the specified patterns ({docs/,.github/,}CODEOWNERS.md)
⚠ governance-file-exists: Did not find a file matching the specified patterns ({docs/,.github/,}GOVERNANCE.md)
✖ community-guidelines-file-exists: Did not find a file matching the specified patterns ({docs/,.github/,}COMM
✖ code-of-conduct-file-exists: Did not find a file matching the specified patterns ({docs/,.github/,}CODE_OF_C
```

Running repolinter on your repository via GitHub Actions

1. Add the tier-specific checks.yml to the .github directory of your project. The file includes a job that runs a repolinter called repolinter-checks.
2. Manually trigger the workflow.
3. The result produces an issue on the repository with a list of file and section existence checks, indicating whether each requirement was met or not.

Review Content

The project should include the following files and sections (link to templates):

- [ ] **LICENSE**

  License of your project, whether public domain (CC0) or other OSI-approved License. Using 'vanilla' license text will allow for GitHub to auto-label the license information on the repository landing page.

- [ ] **code.json**

  Contains metadata about the project, refer to Review Project Metadata

- [ ] **README.md**

  An essential guide that gives viewers a detailed description of your project

Section	Description	Included?
Project Description	This should be 1-3 sentence short description of the project that can be used as a 'one-liner' to describe the repo. A best practice is using this same language as the official 'description' on a GitHub repo landing page.	✓ ✖
About the Project	This should be a longer-form description of the project. It can include history, background, details, problem statements, links to design documents or other supporting materials, or any other information/context that a user or contributor might be interested in.	
Project Vision	This should be a forward-looking statement that outlines the desired future state or long-term goals of the project.	

Section	Description	Included?
Project Mission	This should be a statement that defines the purpose, scope, and specific objectives of the project.	
Agency Mission	Agency-led projects should include information about their agency mission. This should be taken directly from agency websites or wikis.	
Team Mission	Agency-led projects should include information about the team executing on the mission. This should be taken directly from internal team charters and functional statements.	
Core Team	This information helps with succession planning and provenance for security compliance and remediation. It helps future users and contributors understand where the code originated.	
Repository Structure	Including the repository structure helps viewers quickly understand the project layout. It is also helpful to include a 'table of contents" for your documentation, providing "bookmark" or "anchor" links to specific sections of your file to be referenced in tickets, docs, or other communication channels.	
Development and Software Delivery Lifecycle	Outlines the stages, practices, and tools used in developing, testing, and delivering the software.	
Local Development	Use step by step instructions to get from 'zero' to 'running code.'	
Coding Style and Linters	Best practices contributors should follow to reduce friction and improve readability, functionality, and quality of contributions.	
Branching Model	Branching models (such as git flow) are recommended as a best practice for keeping feature development history clear, and to help reinforce development best practices.	
Contributing	For projects that accept contributions, point towards the CONTRIBUTING.md file.	
Community	Point your contributors towards wherever your community exists (e.g. email lists, online discussion boards or channels, project backlogs and documentation, etc.).	
Community Guidelines	This section points to a CODE_OF_CONDUCT.md file or website providing information around acceptable conduct and reporting mechanisms and escalation strategies. It is better to have these processes defined before they are needed, so you can focus on support if/when there is an incident (e.g. Contributor-covenant.org).	
Governance	Make a short statement about how the project is governed (formally, or informally) and link to the GOVERNANCE.md file.	

Section	Description	Included?
Feedback	Direct users towards the channel where they're encouraged to provide feedback, typically a github.com/$REPO/issue/new URL.	
Glossary	Good candidate content includes any project-specific acronyms (esp applicable for Government projects) and any critical Subject Matter Expertise related vocabulary for people who are new to the domain your project is within.	
Policies	This section is to explicitly link to Federal policies and guidelines that are required or recommended for Federal projects to comply with, such as Accessibility (508), Interoperability, Anti-deficiency, Security, Licensing, and other policies that can vary between agencies and domains.	
Public Domain	A best practice is to list the LICENSE under which a project is released at the bottom of the README. In most cases for Federal repos, we default to Creative Commons Zero 1.0 International (world-wide public domain).	

- [ ] **COMMUNITY.md**

  Outlines the community members, roles, responsibilities, and guidelines for participating in the project.

Section	Description	Included
Table of Project Members	Who are the points of contact in your project who are responsible/accountable for the project? This can often be an engineering or design manager or leader, who may or may not be the primary maintainers of the project.	✅ ❌
Responsibilities & Roles	Defines the key members of the project team, their responsibilities, and how they help steer the project's growth and success. the	
Maintainers List	Who are the project maintainers? List out @USERNAMES where possible so they can be tagged in issues/PRs directly.	
Approvers List	Who are the project approvers? List out @USERNAMES where possible so they can be tagged in issues/PRs directly.	
Reviewers List	Who are the project reviewers? List out @USERNAMES where possible so they can be tagged in issues/PRs directly.	
Contributors	Who has contributed to the project? Highlight and credit the users who have contributed to the repository.	
Alumni	Who are the past maintainers or contributors who previously played significant roles in this project who are no longer actively involved? Consider including their roles and dates for context.	
Principles	This section communicates to prospective contributors and users what the values of your community are. The examples provided in the template were established by the Justice40 project at USDS.	

Section	Description	Included
Community Guidelines	This section communicates specific norms and guidelines for how to participate and contribute positively to your community. The more explicit you can be about behaviors you'd like to encourage or discourage, the less friction new contributors will experience in onboarding and operating within your project.	
Acknowledgements	This section recognizes previous work and best practices established by the other members of the federal open source community such as USDS, GSA, 18F, and the Justice40 Project.	

- [ ] **CONTRIBUTING.md**

  Provides guidance on how users can run your project and make contributions to it

Section	Description	Included
How to Contribute	Basic instructions about where to send patches, check out source code, and get development support.	✔ ✖
Getting Started	Includes installation steps, prerequisites for installation, and instructions for working with the source code.	
Team Specific Guidelines	This section helps contributors understand any team structure in the project (formal or informal). Encouraged to point towards the COMMUNITY.md file for further details.	
Building dependencies	This step is often skipped, so don't forget to include the steps needed to install on your platform. If your project can be multi-platform, this is an excellent place for first time contributors to send patches.	
Building the Project	Be sure to include build scripts and instructions, not just the source code itself!	
Workflow and Branching	If your project has a preferred workflow or branching structure, mention it here. We recommend 'git flow' as a good default.	
Testing Conventions	Discuss where tests can be found, how they are run, and what kind of tests/coverage strategy and goals the project has.	
Coding Style and Linters	HIGHLY ENCOURAGED. Specific tools will vary between different languages/frameworks (e.g. Black for Python, eslint for JavaScript, etc.).	
Writing Issues	Make a brief statement about where to file issues, and conventions for doing so. Link to ISSUE_TEMPLATE.md file.	
Writing Pull Requests	Make a brief statement about where to file pull/merge requests, and conventions for doing so. Link to PULL_REQUEST_TEMPLATE.md file.	
Reviewing Pull Requests	Make a brief statement about how pull requests are reviewed, and who is doing the reviewing. Linking to COMMUNITY.md can help.	

Section	Description	Included
Shipping Releases	What cadence does your project ship new releases? (e.g. one-time, ad-hoc, periodically, upon merge of new patches) Who does so? Using the release-guidelines-template.md can assist with developing and documenting the process.	
Documentation Updates	Where is the documentation hosted? How is it updated? Who updates it?	
Policies	This section is here to explicitly link to Federal policies and guidelines that are required or recommended for Federal projects to comply with, such as Accessibility (508), Interoperability, Anti-deficiency, Security, Licensing, and other policies that can vary between agencies and domains.	
Public Domain	This section is to explicitly link to Federal policies and guidelines that are required or recommended for Federal projects to comply with, such as Accessibility (508), Interoperability, Anti-deficiency, Security, Licensing, and other policies that can vary between agencies and domains.	

- [ ] **SECURITY.md**

  Details security policies and procedures

- [ ] **CODE_OF_CONDUCT.md**

  Defines the code of conduct for contributors

- [ ] **GOVERNANCE.md**

  Outlines project governance structure and processes

- [ ] **repolinter.json**

  Lints repository for missing files and sections above

Communications Style Guide

As you are writing content about the repository, consider the following when describing the project, the agency, and how work is done:

- **Focus on solutions, not problems** *Rather than describing the negatives and shortcomings of an existing project, process, or group, focus on highlighting the benefits, outcomes, and positive impacts of the solution. Be constructive and accentuate the positive.*
- **Use Plain Language** *Be clear and concise when explaining concepts. Avoid jargon unless it is necessary and well-defined. Use active voice to be clearer and more engaging.*
- **Focus on the audience** *Write for a broad audience, ensuring that developers, stakeholders, end-users, and viewers of all skill levels can understand and engage with the content.*

Please refer to the style guides below for additional tips and guidance:

- CMS Resources
  - Guidelines for Effective Writing
  - Tone of Voice section in Branding Guide
  - 2020 Plain Writing Report
- 18F Style Guide
- UK Content Style Guide

Results

Insert Review Here

Review Project Metadata

As part of the SHARE IT Act, Federal Source Code Policy, and the agency's software inventory tracking initiatives, each repository must contain a code.json file, storing metadata on your project.

For more information on code.json, please review the gov-codejson documentation repository.

Creating code.json on your repository

Using form site

Users can fill out a web form that creates a code.json file to be uploaded to a project's source code repository: https://dsacms.github.io/codejson-generator.

Using automated-codejson-generator

The automated-codejson-generator is a GitHub Action that automatically generates and maintains code.json files for federal open source repositories. It ensures schema consistency and automates various metadata calculations.

Using the repo-scaffolder cookiecutter CLI

1. In the `.github` directory, run the command:

```
cookiecutter . –directory=codejson
```

2. Answer various questions about your project.

3. A code.json file will be generated with your responses.

As you continue development in this repository, it is important to keep this file up-to-date. Our automated-codejson-generator can assist with updating this file.

Results

Insert Review Here

Review Repository Details

The GitHub repository homepage features a concise description of the project, a list of relevant topic tags, and general information about the repository to provide a comprehensive overview for users and contributors.

About Section:

- [ ] Description

  1-2 sentences describing the project

- [ ] Website

  Link to project's website

- [ ] Topics

  Tags for project discoverability. Helpful topics to classify a repository include the repository's intended purpose, subject area, community, or language.

Include in Home Page:

- [ ] Releases
- [ ] Packages
- [ ] Deployments

Results

Insert Review Here

Review OpenSSF Scorecard

Checks	Description & Condition	Risk	Min	Score
Dangerous-Workflow	Does the project avoid dangerous coding patterns in GitHub Actions? (e.g. Untrusted Code Checkout, Script Injection with Untrusted Context Variables)	Critical	10	✅ ❌
Dependency-Update-Tool	Does the project use tools to help update its dependencies e.g. Dependabot, RenovateBot?	High	10	
Token-Permissions	Does the project declare GitHub workflow tokens as read only?	High	9	
Branch-Protection	Does the project use Branch Protection?	High	6	
Code-Review	Does the project require code review before code is merged?	High	10	
Binary-Artifacts	Is the project free of checked-in binaries?	High	10	
Maintained	Is the project maintained?	High	10	

Checks	Description & Condition	Risk	Min	Score
Vulnerabilities	Does the project have unfixed vulnerabilities? Uses the OSV service.	High	8	
Fuzzing	Does the project use fuzzing tools, e.g. OSS-Fuzz? (For projects that accept user input)	Medium	10	
Static Code Analysis Tools	Does the project use static code analysis tools, e.g. CodeQL, LGTM, SonarCloud?	Medium	10	
Pinned-Dependencies	Does the project declare and pin dependencies? (For stadiums + federations)	Medium	8	
Security Policy	Does the project contain a security policy?	Medium	10	
CII-Best-Practices	Does the project have a CII Best Practices Badge? Project has an OpenSSF Best Practices Badge (For stadiums + federations)	Low	5	
Contributors	Does the project have contributors from multiple organizations?	Low	6	
License	Does the project declare a license?	Low	9	
CI-Tests	Does the project run tests in CI, e.g. GitHub Actions, Prow?	Low	10	
Packaging	Does the project build and publish official packages from CI/CD, e.g. GitHub Publishing? (For projects that are packages)	Medium	10	
Signed-Releases	Does the project cryptographically sign releases?	High	8 If under TLD: 10	

Results

Overall Score:

Insert review here

Additional Notes & Questions

Insert any notes or questions here

Sign off on risk acceptance of open-sourcing the software product

After reviewing the materials prepared by the team that is working to open source the product, the business owner signs off on a risk acceptance for open-sourcing the software product.

Requesting sign off from key people on this request.

Reviewer Organization	Reviewer Names	Reviewer's Recommendation
Code Reviewer 's Reccommendation	CODE REVIEWER 1 CODE REVIEWER 2 CODE REVIEWER 3	[Approved/Needs Approval] [Approved/Needs Approval] [Approved/Needs Approval]
ISSO	ISSO REVIEWER	[Approved/Needs Approval]
ISG Technical Approval	ISG REVIEWER	[Approved/Needs Approval]
Business Owner(s)	BUSINESS OWNER 1 BUSINESS OWNER 2	[Approved/Needs Approval] [Approved/Needs Approval]

Flipping the Switch: Making the Repository Public

Once the repository has passed outbound review, we are ready to "flip the switch" and officially make it public. Once made public, there are a couple of actions that need to be taken:

Repository Actions

Please enable the following features to enhance repository security and maintain code quality:

- [ ] **Dependabot Alerts**

  A GitHub Feature. Get notified when one of your dependencies has a vulnerability

- [ ] **Secret Scanning Alerts**

  A GitHub Feature. Get notified when a secret is pushed to this repository. Ideally set this up to run after each new commit is pushed to the Repository.

- [ ] **Branch Protections**

  Ensures the integrity of important branches by preventing unauthorized actions like force pushes and requiring pull request reviews with specific checks before merging. Dev and main should be protected branches in the repository.

- [ ] **Git Branching**

  After making the repository public, make sure there is a coherent git branching plan in place. For example: agree to merge feature related pull requests into dev but merge bug fixes into main instead of dev first.

- [ ] **Add Repolinter GH Action to CI**

  For ongoing adherence to repository hygiene standards, integrate the repolinter GitHub Action into your CI pipeline. This addition enhances your workflow by automatically enforcing repository cleanliness standards.

- [ ] **Optional: DCO (Developer Certificate of Origin)**

Requires all commit messages to contain the Signed-off-by line with an email address that matches the commit author. The Developer Certificate of Origin (DCO) is a lightweight way for contributors to certify that they wrote or otherwise have the right to submit the code they are contributing to the project. The GitHub app to enforce DCO can be found here .

Communications & Rollout

Share the good news with communities both inside and outside CMS!

- [ ] **Draft a launch announcement** Be sure to include the following information:
- Repo Description
    - Repo URL
    - Authoring Team Email Contact
    - Authoring Team URL
    - Authoring Team Slack Channel
    - Call to action (File issues, contribute PRs)
- [ ] **Post launch announcement to CMS slack channel**
    - #cms-opensource
    - #cms-api-community
    - #cms-data-community
    - #cms-engineering-community
    - #ai-community
- [ ] **Send a launch announcement email**
- [ ] **Add launch announcement to a Confluence Wiki Page**

Tracking

Add your project to our inventories.

- [ ] **Add to https://github.com/dsacms/open projects inventory**
- [ ] **Add code.json to repository and sent over a pull request to code.gov**