Development Process

Contribution Standards

New Features or Major Updates

Before you start coding…

… please make sure there is a JIRA issue that corresponds to your contribution. This is a general rule that the MXNet community follows for all code contributions, including bug fixes, improvements, or new features, with an exception for trivial hot fixes. If you would like to fix a bug that you found or if you would like to add a new feature or improvement to MXNet, please follow the [File a bug report or Propose an improvement or a new feature](http://mxnet.io/community/index.html) guidelines to open an issue in [MXNet's JIRA](http://issues.apache.org/jira/browse/MXNet) before starting with the implementation.

If the description of a JIRA issue indicates that its resolution will touch sensible parts of the code base, be sufficiently complex, or add significant amounts of new code, the MXNet community might request a design document (most contributions should not require a design document). The purpose of this document is to ensure that the overall approach to address the issue is sensible and agreed upon by the community. JIRA issues that require a design document are tagged with the requires-design-doc label. The label can be attached by any community member who feels that a design document is necessary. A good description helps to decide whether a JIRA issue requires a design document or not. The design document must be added or attached to or link from the JIRA issue and cover the following aspects:

- Overview of the general approach
- List of API changes (changed interfaces, new and deprecated configuration parameters, changed behavior, …)
- Main components and classes to be touched
- Known limitations of the proposed approach

A design document can be added by anybody, including the reporter of the issue or the person working on it.

Please check with community (by emailing the dev list) if a design document is required before starting to code on a major feature. A design document must be accepted by the community with lazy consensus before a contribution will be added to MXNet's code base.

Core Library

- Follow the Google C++ Style Guide for C++ code.
- Use `doxygen` to document all of the interface code.
- Use RAII to manage resources, including smart pointers like shared_ptr and unique_ptr as well as allocating in constructors and deallocating in destructors. Avoid explicit calls to new and delete when possible. Use make_shared and make_unique instead.
- To reproduce the linter checks, use `make lint`. (You must `pip install pylint cpplint` before you try this.)

Python Package

- Always add `docstring` to the new functions in `numpydoc` format.
- To reproduce the linter checks, type `make lint`.

R Package

Code Style

- Most of the C++ code in the R package relies heavily on Rcpp.
- We follow the Google C++ Style Guide for C++ code. This allows us to maintain consistency with the rest of the project. It also allows us to check style automatically with a linter.
- To check the code style, type the following command at the root folder:

```
make rcpplint
```

- If necessary, disable the linter warning on certain lines with `// NOLINT(*)` comments.

Auto-Generated API

- Many MXNet APIs are exposed dynamically from Rcpp.
- mxnet_generated.R is the auto-generated API and documents for these functions.
- Remake the file by typing the following command at root folder:

```
make rcppexport
```

- Use this command only when there is an update to dynamic functions.

API Document

The document is generated using roxygen2. To remake the documents in the root folder, use the following command: `bash make roxygen.`

R Markdown Vignettes

R Markdown vignettes are located in R-package/vignettes. These R Markdown files aren't compiled. We host the compiled version on doc/R-package.

To add a new R Markdown vignettes:

- Add the original R Markdown file to `R-package/vignettes`
- Modify `doc/R-package/Makefile` to add the Markdown files to be built.
- Clone the dmlc/web-data repo to the `doc` folder.
- Type the following command for the `doc/R-package` package:

```
make the-markdown-to-make.md
```

- This generates the markdown and the figures and places them into `doc/web-data/mxnet/knitr`.
- Modify the `doc/R-package/index.md` to point to the generated markdown.
- Add the generated figure to the `dmlc/web-data` repo.
- If you have already cloned the repo to doc, use `git add`.
- Create a pull request for both the markdown and `dmlc/web-data`.
- You can also build the document locally with the following command: `doc`

```
make html
```

New APIs

Make sure to add documentation with any code you contribute. Follow these standards:

API Documentation

- Document are created with Sphinx and recommonmark.
- Follow numpy doc standards. With the following caveats:
    - *If an API is implemented in Python or has a wrapper defined, the documentation and the examples reside where the function is defined in `.py` file in python/mxnet folder. Same goes for other languages.*
    - If the API is dynamically generated from the MXNet backend, the documentation is in the C++ code(.cc file) where the operator is registered in describe method of the `NNVM_REGISTER_OP`. The file and line number for the function is usually printed with the API documentation on mxnet.io.
    - *A clear and concise description of the function and its behavior.* List and describe each parameter with the valid input values, whether it is required or optional, and the default value if the parameter is optional.
    - *Add an example to help the user understand the API better. If the example can be language-neutral or is conceptual, add it in the C++ documentation.*
    - *Make sure your example works by running a Python version of the example.* If a concrete and simple language-specific example can further clarify the API and the API arguments, add the example in language-specific files.

Refer to these examples for guidance: Embedding , ROIPooling , Reshape.

Test Cases

- All of the tests can be found in /tests.
- We use `nose` for Python test cases, and `gtest` for C++ unit tests.

Examples

- Use cases and examples in /examples.

- Many tutorials that are in Jupyter notebook format are in /docs/tutorials.
- If you write a blog post or tutorial about or using MXNet, please tell us by creating an issue in JIRA. We regularly feature high-quality contributed content from the community.

Testing and Rendering

- Make sure not to break any coding standards. Run

```
make lint
```

- You can build documents locally to proof them.

Guidelines for Reviewers/Committers

1. As a reviewer you have the responsibility to ensure contributions meet the quality bar. This means you should try and ensure:
    a. Contributions are inline with the overall code architecture.
    b. Contributions are not lowering the overall code maintainability i.e. documentation and unit tests are not missing.
    c. If you don’t understand the logic/flow now, chances are it won’t automatically get better in the future. So feel free to push back if code /logic is hard to follow. Ask to include comments clarifying implementation details whenever necessary.
2. When providing feedback or when seeking clarification, be clear. Don’t assume the PR owner knows exactly what is in your mind.
3. If you decide to close a PR without merging, get an ack from the PR owner that that is indeed the right next step.
4. It is NOT recommended for a committer to merge pull requests that the committer authored. Instead the committer MUST get at least one approval from another committer to merge his/her pull request.
5. When you update a pull request with upstream, you MUST use rebase to ensure that the pull request is easy to review by the community. See the how-to link here: https://mxnet.incubator.apache.org/community/contribute.html

Deleting Comments on GitHub

GitHub issues and pull requests allow a user with "Write" permissions to delete another's comment. The Apache Way yearns for openness in all situations, deleting a comment is rarely the right thing to do.

If a comment does need deleting, most likely because it does not adhere to our Code of Conduct, then the PMC as a whole should discuss and vote on the deletion. If a comment needs to be deleted quickly, then either the PMC Chair (or while in the Incubator a Mentor) should make that decision.

Related Articles

- Development Guide