Contributor Guide

- 1 How to contribute to Apache NiFi
- 2 Technologies
  - 2.1 Supported Versions
  - 2.2 Minimum Required Versions
- 3 Running NiFi in Debug mode
- 4 Where to Start?
  - 4.1 Finding Issues and Extending the Project
  - 4.2 Documentation Contributions
    - 4.2.1 System-level documentation
    - 4.2.2 Component-level documentation
- 5 Create a ticket for new bugs or features
- 6 Providing code or documentation contributions
  - 6.1 Configure your git client
    - 6.1.1 Ensure your git user name and email are configured
    - 6.1.2 Windows Specific configuration
  - 6.2 Clone a copy of the repository
    - 6.2.1 From the Apache Hosted Repository
    - 6.2.2 From the GitHub Mirror
    - 6.2.3 From your GitHub Fork
      - 6.2.3.1 Retrieval of upstream changes
  - 6.3 Checkout the 'main' or '1.x.0' branch
    - 6.3.1 The NiFi community uses a modified Gitflow development model. A summary:
  - 6.4 Perform your code changes
    - 6.4.1 Create a feature branch
  - 6.5 Test your changes
  - 6.6 Update Licensing Documentation
  - 6.7 Commit your changes
    - 6.7.1 List the current files that have changes
    - 6.7.2 Stage the file(s) to be committed with git add
    - 6.7.3 Perform the commit
  - 6.8 Keeping your feature branch current
    - 6.8.1 Update your local copy of main
    - 6.8.2 Perform the git rebase from your branch
  - 6.9 Supplying a contribution
    - 6.9.1 Supplying a contribution through patch and JIRA issue
    - 6.9.2 Supplying a contribution through a pull request (PR) via GitHub
      - 6.9.2.1 Push changes to your personal GitHub repository remote
      - 6.9.2.2 Opening a Pull Request (PR) to the NiFi project
- 7 Code Review Process
- 8 Additional notes on code contributions
  - 8.1 Rebasing and Squashing while under review
  - 8.2 Steps to merge/close pull requests with two main branches
    - 8.2.1 Fetch Config
  - 8.3 Code Style
    - 8.3.1 IntelliJ IDEA Users
    - 8.3.2 Eclipse Users
- 9 Contact us!

How to contribute to Apache NiFi

We are always excited to have contributions from the community - especially from new contributors!
We are interested in accepting contributions of code, as well as documentation and even artwork that can be applied as icons or styling to the application.

Technologies

The back end of Apache NiFi is written in Java. The web tier makes use of JAX-RS, and JavaScript is extensively used to provide a user interface. We depend on several third-party JavaScript libraries, including D3 and JQuery among others. We make use of Apache Maven for our builds and Git for our version control system.

Documentation is created in AsciiDoc.

Supported Versions

NiFi automated builds run on the latest updated versions of the following Java Long Term Support releases:
Minimum Required Versions

For each supported version of Java, NiFi requires the follow minimum update version:

- Java 17.0.6

NiFi automated builds run on the latest updated version of Apache Maven and require the following minimum version, which can be executed using the Maven Wrapper scripts included in the repository:

- Apache Maven 3.9.2

Running NiFi in Debug mode

While NiFi Mock library can greatly simplify your development and testing efforts, sometimes it is still necessary to be able to set breakpoints and debug your code within the running instance of NiFi. To do that NiFi needs to be started in Debug mode within your IDE. Please follow the instructions provided in the NiFi IDE Integration module.

Where to Start?

Finding Issues and Extending the Project

NiFi's JIRA page can be used to find tickets that need effort. The first step to getting involved is to create an account on Jira, then sign up for the Developer Mailing List and send an email to dev@nifi.apache.org requesting "Jira contributor access" and noting your Jira username. This will allow you to assign Jira tickets to yourself to indicate active development efforts.

Any tickets that are tagged as "beginner", or extension tickets (creating a new processor for interacting with a different system) are good places to start. Processors should be self-contained and not rely on other outside components (except for Controller Services), so they make for excellent starting points for new NiFi developers to get started. This exposes the developer to the NiFi API and is the most extensible part of the dataflow system. The NiFi Developer Guide provides details on how to get started with the NiFi API as well as best practices and common approaches to component design.

We have additionally created filters that show various types of issues available to be worked:

- NiFi Unassigned Issues
- NiFi Unassigned Issues Tagged Beginner

Documentation Contributions

System-level documentation

System-level and overview documentation such as:

- Administration Guide,
- Developer's Guide,
- User's Guide,
- Overview,
  etc.

Are located in `<code checkout location>/nifi-docs/src/main/asciidoc` and can be changed by making changes to the ASCIIdoc files and following the code contribution process described in "Providing code or documentation contributions" section of this document.

Tools available to facilitate documentation generation are available at Editing AsciiDoc with Live Preview.

Component-level documentation

Component level documentation can be contributed to by making direct modifications to the source code of the relevant component. Generally the documentation of components is performed by adding the relevant details to the following annotations

```java
@CapabilityDescription( "The overall description of the component would go here..." )
```

and the description field of annotations such as:

```java
@DynamicProperty( description = "The description of the dynamic property would go here...")
```
and the description method of Component Properties and Relationships

```java
PropertyDescriptor.Builder()
    .name("Property Name")
    .description("The description of the Property would go here...")

Relationship.Builder()
    .name("name-of-the-relationship")
    .description("The description of the relationship would go here... ")
    .build();
```

When the documentation of a processor is particularly complex (e.g., may include very long descriptions or images), a contributor may choose to use NiFi's capability to automatically link additional documentation to a processor.

In this case the contributor may extend documentation by performing changes to 'additionalDetails.html'. As described in the Developer's Guide, this file should exist within a directory whose name is the fully-qualified name of the Processor, and this directory's parent should be named docs and exist in the 'resources' directory of a particular processor bundle.

As with other contributions, component level documentation should follow the process described in "Providing code or documentation contributions" section of this document.

Create a ticket for new bugs or features

Run into a bug or think there is something that would benefit the project? Regardless if you have the time to provide the fix or implementation, we encourage any such items to be filed as an issue at the Apache NiFi JIRA.

Providing code or documentation contributions

The source is hosted at https://git-wip-us.apache.org/repos/asf/nifi.git

Like all Apache projects, a mirror of the git repository is also located on GitHub at https://github.com/apache/nifi which provides ease in forking and generating pull requests (PRs).

Configure your git client

Ensure your git user name and email are configured

The following lines ensure your commits are appropriately annotated with your information

```bash
git config --global user.name "User Name"
git config --global user.email "User Name@example.com"
```
Windows Specific configuration

The following options provide handling of long file paths that can be troublesome as well as not using Windows style line returns.

```
git config --global core.longpaths true
git config --global core.autocrlf false
```

Clone a copy of the repository

**From the Apache Hosted Repository**

```
git clone https://git-wip-us.apache.org/repos/asf/nifi.git
```

**From the GitHub Mirror**

```
git clone https://github.com/apache/nifi.git
```

**From your GitHub Fork**

```
git clone git@github.com:<account name>/nifi.git
```

Retrieval of upstream changes

Additionally, it is beneficial to add a git remote for the mirror to allow the retrieval of upstream changes

```
git remote add upstream https://github.com/apache/nifi.git
```

Checkout the 'main' or '1.x.0' branch

```
git checkout -b main origin/main will create a local, tracking branch named main.
git checkout -b 0.x origin/1.x.0 will create a local, tracking branch named 1.x.0.
```

The main branch currently represents the next major release line (1.x.0).

**The NiFi community uses a modified Gitflow development model. A summary:**

- Use of a central repository
- Branch per feature similar to the Feature Branch Workflow
- Work is done locally and then pushed to the central repo
- 'main' branch contains the official release history. Code changes (not code formatting, administrative updates) require Review-Then-Commit (RTC) by another committer to get incorporated.

For a more in-depth guide, visit Gitflow Workflow by Atlassian as well as the original presentation of the development model by Vincent Driessen.

Perform your code changes

**Create a feature branch**

Typically, there is a 1-to-1 mapping of a branch to a backing ticket as specified in JIRA.

Create a local branch that relates the associated JIRA issue with the branch. Such an example would be:

```
git checkout -b nifi-359 main
```

This provides instant traceability to the supporting issue and provides a means of linking discussion.

Test your changes
For code changes, ensure that the full suite of tests is executed via `mvn -Pcontrib-check clean install` at the root nifi folder. Please write or update unit tests to verify your changes.

For documentation related changes, ensure that format looks appropriate for the output in which it is rendered.

**Update Licensing Documentation**

Did your change introduce new dependencies? Are these dependencies licensed in a way that is compatible for inclusion under ASF 2.0? Did you change or remove any existing dependency? Did you update the LICENSE and NOTICE files?

Each new source or configuration file added must have the ASF 2.0 license at the beginning of the file. It should be easy enough to copy one from another location.

There are a few places where licensing matters when you’re making software changes. There are global LICENSE and NOTICE files in the nifi-assembly module which rolls up the contents of each NIFI nar LICENSE and NOTICE. The general guidelines for maintaining these are as follows:

- If you've added dependencies, you need to ensure that the dependencies are licensed in a way that is compatible for inclusion under ASF 2.0. These are generally defined here.
- If you've changed a dependency at all (e.g. add, remove, change version), you must verify that the LICENSE and NOTICE from that dependency are contained the nar you're changing (if applicable) and in nifi-assembly. Be mindful to check if the dependency you're adding already exists in the LICENSE and NOTICE files you're editing. If it does, Consider the following guidelines:
  - LICENSE: If the dependency version already exists here and the licenses are identical, you may simply add the additional version (e.g. "The binary distribution of this product bundles Google Protocol Buffers Java 2.5.0 and 3.3.1"). If the licenses are different, list them separately.
  - NOTICE: If the dependency version already exists here and the notices are the same, there's no change needed. If the notices are different you need to determine how different they are. If one is a superset of the other, you may simply replace the subset with the superset. If they're vastly different from each other, you need to list them separately.

Other general guidelines as they relate to licensing in NIFI:

- Whether some binary dependencies NOTICE calls out a transitive binary dependency it might or might not have is not relevant. What is relevant is which transitive dependencies, no matter how many levels deep it comes in, we pull into our nars or convenience binaries. They must all be accounted for properly if we're including them. See the Apache Licensing Guide for more information on this.
- All binary dependencies bundled in a given nar must be documented in its LICENSE and NOTICE. On top of that, the nifi-assembly should cover those dependencies as well.
- All source dependencies taken as-in or modified must be documented in the nifi-assembly LICENSE and NOTICE.

**Commit your changes**

**List the current files that have changes**

$ git status
On branch nifi-359
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)
modified: nifi-docs/src/main/asciidoc/contributor-guide.adoc

no changes added to commit (use "git add" and/or "git commit -a")

**Stage the file(s) to be committed with git add**

$ git add nifi-docs/src/main/asciidoc/contributor-guide.adoc

$ git status
On branch nifi-359
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)
modified: nifi-docs/src/main/asciidoc/contributor-guide.adoc

In the case of multiple files, it is also possible to stage all tracked files through `git add`.

**Perform the commit**

In the interest of providing traceability, it is helpful to lead off your commit with the associated JIRA issue number (case-sensitive, NIFI in uppercase) and a summary of the change this commit is providing. Such an example is:

$ git commit -m "NIFI-359 Adding a guide for procedure and best practices in contributing to the project."

The issue being listed first provides easy access to the supporting ticket and ensures it is not truncated from the varied means in which commits will be viewed.
Keeping your feature branch current

If you are working on a branch over an extended period of time, it helps to keep your code current with the main branch to ensure your changes are applied as anticipated and to facilitate the merging process.

This is best accomplished through the git rebase functionality. There are many ways in which git rebase can be utilized in this context of branching and rebasing. Typically, the command to do so is performed from your feature branch. Continuing on with the sample of NIFI-359, we will show one way of accomplishing this task.

Update your local copy of main

$ git checkout main
Switched to branch 'main'

$ git fetch upstream
remote: Counting objects: 52, done.
remote: Compressing objects: 100% (11/11), done.
remote: Total 16 (delta 7), reused 0 (delta 0)
Unpacking objects: 100% (16/16), done.
From https://git-wip-us.apache.org/repos/asf/nifi
 a49a03d..8d745c2 main -> upstream/main

$ git merge upstream/main
Updating a49a03d..8d745c2
Fast-forward
1 file changed, 10 insertions(+), 17 deletions(-)

Perform the git rebase from your branch

$ git checkout nifi-359
Switched to branch 'nifi-359'

$ git rebase main
First, rewinding head to replay your work on top of it...
Applying: NIFI-359: Removing the output to System.out and providing some simple checks to ensure extraction of attributes is consistent with the sample file used in the test.

Supplying a contribution

There are currently two paths from which we review and accept contributions, ASF JIRA and GitHub Pull Requests (PRs), regardless of whether your contribution is source code or documentation (note that submitting your contribution through a pull request via GitHub is the preferred option as it makes the reviews much easier).

Supplying a contribution through patch and JIRA issue

Assuming you are still on the branch you want to make a patch from (NIFI-359) and that you've just committed the change and it is a single commit then you can run the following to generate a patch file

`git format-patch nifi-359~1 -o </desired/path/for/patch>/nifi-359.patch`

Now you can go to JIRA for ticket NIFI-359 and select 'more -> attach files' and upload the patch file. Then select 'Submit Patch'.

TODO: Once auto-build verification comes into play this guidance will likely change.

Supplying a contribution through a pull request (PR) via GitHub

Push changes to your personal, GitHub repository remote

Typically, this remote is listed as origin. To confirm, perform a git remote -v from within your cloned repository directory.

$ git remote -v
origin git@github.com:apiri/nifi.git (push)
origin git@github.com:apiri/nifi.git (fetch)
upstream https://git-wip-us.apache.org/repos/asf/nifi.git (fetch)

Once the appropriate remote (remote name) is determined, it is possible to push the branch with your changes to GitHub through:

$ git push <remote name> <branch name>
As an example, based on the context of this guide, this would take the form of:

$ git push origin nifi-359

This will update your fork of NiFi and can be verified from your GitHub personal project page:

Opening a Pull Request (PR) to the NiFi project

From your personal fork, a Pull Request can be opened simply by selecting the Pull Request link highlighted below.
This will allow specification of the your branch which has the changes, and will open a PR for review by the committers of the project.
While we request that the PR begin with a single, squashed commit as the *initial commit*, if you continue to modify the code, please continue to make commits against the same branch. GitHub handles this process well and by pushing to the same remote branch you opened the PR from, your commits will show automatically. This way reviewers can track changes in accordance with their feedback cleanly. Please avoid using `squash` or `--force` when making these changes, as it will overwrite the remote branch and reduce the capability to track these changes. See additional note about Rebasings and Squashing while under review below for more information.

**Code Review Process**
Apache NIFI has a **Review-Then-Commit (RTC)** philosophy for handling all contributions. Reviewers first ensure that the code applies and builds appropriately to the build, passes the code standards as established by the Maven profile "contrib-check." From here, code is evaluated to ensure best practices within the NiFi framework are applied and, where applicable, that the user experience of interfacing with the contribution is consistent and any changes are backwards compatible. This process may be iterative but works toward a final product that is then merged into the codebase.

While only committers can actively promote contributions into the repository, feedback on issues, regardless of committer status, is appreciated and valued in the review process.

If you are interested in facilitating the review process, a listing of all code contributions with a patch are available via a JIRA filter, **NIFI Patch Available**.

A view of the filter is shown below:

<table>
<thead>
<tr>
<th>T</th>
<th>Key</th>
<th>Summary</th>
<th>Assignee</th>
<th>Reporter</th>
<th>P</th>
<th>Status</th>
<th>Resolution</th>
<th>Created</th>
<th>Updated</th>
<th>Due</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NIFI-11938</td>
<td>Create a Processor to consume slack message events</td>
<td>Mark Payne</td>
<td>Mark Payne</td>
<td></td>
<td>PATCH AVAILABLE</td>
<td>Unresolved</td>
<td>Aug 10, 2023</td>
<td>Aug 10, 2023</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NIFI-11912</td>
<td>Add to Standard OAuth2 Access Token controller service the possibility to choose a Proxy Configuration Service</td>
<td>Nandor Abonyi</td>
<td>Andrea Molteni</td>
<td></td>
<td>PATCH AVAILABLE</td>
<td>Unresolved</td>
<td>Aug 04, 2023</td>
<td>Aug 15, 2023</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NIFI-11877</td>
<td>Add a comments field for UpdateAttribute Rules</td>
<td>Mark Bean</td>
<td>Mark Bean</td>
<td></td>
<td>PATCH AVAILABLE</td>
<td>Unresolved</td>
<td>Jul 30, 2023</td>
<td>Aug 17, 2023</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NIFI-11874</td>
<td>Reorganize UI presentation of Process Group configuration</td>
<td>Mark Bean</td>
<td>Mark Bean</td>
<td></td>
<td>PATCH AVAILABLE</td>
<td>Unresolved</td>
<td>Jul 29, 2023</td>
<td>Aug 10, 2023</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NIFI-11857</td>
<td>CLI - recursively change version of processors</td>
<td>Pierre Villard</td>
<td>Pierre Villard</td>
<td></td>
<td>PATCH AVAILABLE</td>
<td>Unresolved</td>
<td>Jul 25, 2023</td>
<td>Jul 25, 2023</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NIFI-11852</td>
<td>CLI - connect two process groups</td>
<td>Pierre Villard</td>
<td>Pierre Villard</td>
<td></td>
<td>PATCH AVAILABLE</td>
<td>Unresolved</td>
<td>Jul 24, 2023</td>
<td>Jul 25, 2023</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NIFI-11779</td>
<td>Refactor Groovy tests in nifi-elasticsearch-restapi-processors to Java (and JUnit 5)</td>
<td>Daniel Stiegli...</td>
<td>Daniel Stiegli...</td>
<td></td>
<td>PATCH AVAILABLE</td>
<td>Unresolved</td>
<td>Jul 03, 2023</td>
<td>Aug 11, 2023</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NIFI-11740</td>
<td>Add SnowflakeConfigurationService</td>
<td>Peter Turcsanyi</td>
<td>Peter Turcsanyi</td>
<td></td>
<td>PATCH AVAILABLE</td>
<td>Unresolved</td>
<td>Jun 22, 2023</td>
<td>Jun 26, 2023</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NIFI-11739</td>
<td>Add ability to ignore missing fields in Puliccegb</td>
<td>Matt Burgess</td>
<td>Matt Burgess</td>
<td></td>
<td>PATCH AVAILABLE</td>
<td>Unresolved</td>
<td>Jun 22, 2023</td>
<td>Jul 25, 2023</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NIFI-11609</td>
<td>Support Request-Response pattern in MQTT Processors</td>
<td>Nandor Abonyi</td>
<td>Nandor Abonyi</td>
<td></td>
<td>PATCH AVAILABLE</td>
<td>Unresolved</td>
<td>May 28, 2023</td>
<td>Jul 28, 2023</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NIFI-11595</td>
<td>ServiceProvider.replace() cannot create the initial state</td>
<td>Peter Turcsanyi</td>
<td>Peter Turcsanyi</td>
<td></td>
<td>PATCH AVAILABLE</td>
<td>Unresolved</td>
<td>May 24, 2023</td>
<td>Jun 15, 2023</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NIFI-11518</td>
<td>Upgrade to Jetty 10</td>
<td>David Handermann</td>
<td>David Handerman</td>
<td></td>
<td>PATCH AVAILABLE</td>
<td>Unresolved</td>
<td>May 02, 2023</td>
<td>Aug 17, 2023</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NIFI-11480</td>
<td>PutElasticsearchRecord should have an option to output _bulk api response errors as flowfile attributes</td>
<td>Chris Sampson</td>
<td>Chris Sampson</td>
<td></td>
<td>PATCH AVAILABLE</td>
<td>Unresolved</td>
<td>Apr 24, 2023</td>
<td>Aug 18, 2023</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NIFI-11303</td>
<td>Create direct go-to option from Provenance lineage</td>
<td>Reynaldo Rea</td>
<td>Mark Bean</td>
<td></td>
<td>PATCH AVAILABLE</td>
<td>Unresolved</td>
<td>Mar 20, 2023</td>
<td>Aug 14, 2023</td>
<td></td>
</tr>
</tbody>
</table>
Additional notes on code contributions

Rebasing and Squashing while under review

Although you may be asked to rebase or squash your contribution as part of the review process, don't feel the need to do so speculatively. The committer working on merging the contribution may prefer to do these types of operations as part of the merge process, and the history of your patch or pull request may aid in the review process.

Steps to merge/close pull requests with two main branches

As NiFi now has a 1.x (main) and 1.x.y (support) branches, pull requests (PR) may be applied to both, but typically will only be targeted at main. Here is a step-by-step guide for committers to ensure this occurs for all PRs.

1. Check out the latest main
   a. $ git checkout main
   b. $ git pull upstream main

2. Check out the PR (example #327). This will be in detached-HEAD state. (Note: You may need to edit the .git/config file to add the fetch lines below)
   a. $ git checkout github/pr/327

2. Create a branch for the PR
   a. $ git checkout -b pr327

2. Apply the changes and sign off. This could be through a commit --amend, rebase, etc.
   a. $ git commit --amend --s
   b. Edit the commit file to contain "This closes #327."
   c. $ git log
   d. Copy commit id of last commit

2. Switch back to the main branch
   a. $ git checkout main

2. Merge the changes. You can use cherry-pick, merge, etc.
   a. $ git merge

2. (Optional) Ensure the commit was applied successfully
   a. $ git log

2. Push to the Apache repository (main branch)
   a. $ git push apache main
2. (Optional) Switch to the support branch - the following steps are only required if the change is specifically being targeted for a patch release of an existing build (e.g. a bug fix to create 1.15.3 from the 1.15.2 build)
   a. $ git checkout -t support/1.x.y

2. (Optional) Check the status of the branch
   a. $ git log

2. (Optional) Apply the changes from the PR branch
   a. $ git cherry-pick <commit id>

2. (Optional) Push to the Apache repository (support branch)
   a. $ git push apache support/nifi-1.x.y

Fetch Config
To ensure you are able to pull the PR directly, add the following lines to your .git/config file.

```
[remote "github"]
url = git@github.com:apache/nifi.git
fetch = +refs/heads/*:refs/remotes/github/*
fetch = +refs/pull/*/head:refs/remotes/github/pr/*
```

Code Style
The maven-checkstyle-plugin enforces minimum code style requirements.

The checkstyle.xml configuration in the project root directory contains the current set of style rules.

IntelliJ IDEA Users
1. Download Checkstyle-IDEA plugin
2. Install plugin by loading the downloaded zip file from Preferences - Plugins - Install plugin from disk...
3. After restarting IntelliJ, configure Checkstyle plugin to use NiFi Checkstyle configuration file from Preferences - Other Settings - Checkstyle
4. Add checkstyle.xml in Configuration File and activate the configuration

Eclipse Users
The Eclipse Checkstyle Plugin can be configured with the project checkstyle.xml to evaluate conformance with project coding standards.

Contact us!
The developer mailing list (dev@nifi.apache.org) is monitored pretty closely, and we tend to respond quickly. If you have a question, don't hesitate to shoot us an e-mail - we're here to help! Unfortunately, though, e-mails can get lost in the shuffle, so if you do send an e-mail and don't get a response within a day or two, it's our fault - don't worry about bothering us. Just ping the mailing list again.