Earth Prediction Innovation Center (EPIC) Release Process for Unified Forecast System (UFS) Applications

Version History:

- **LATEST*: Version 3 (Draft as of 12/16/22) Google Drive link**
- **Version 2 Google Drive link**
- **Version 1 Google Drive link**

*Note that the "latest" Google Doc contains the most recent updates. This page is updated periodically and may not reflect the latest changes.

Table of Contents:

I. INTRODUCTORY NOTE

Releases will be overseen by the Release Coordination Cross-Cutting Team (RC CCT) and executed by an application Release Team. The Release Team includes all individuals who perform tasks related to the release and who desire a role in planning and overseeing execution of the release. The team may include, but is not limited to, Earth Prediction Innovation Center (EPIC) Scrum Masters (SMs) and Product Owners (POs); Code, Configuration, Testing, and Data Managers for the various component repositories; subject matter experts; and stakeholders from various NOAA offices. All tasks for the release will be executed according to procedures established by the appropriate developers (e.g., Code, Configuration, Data, and Testing managers for component repositories).

II. RELEASE PLANNING PHASE

A. Release Meetings:

i. Schedule

Release meetings will occur weekly at a time most attendees are available. This may take place at the time of the usual Code Management meeting, if one exists, or at a separate time, based on mutual agreement by the Advanced User Support (AUS) Scrum Master, AUS Product Owner, Lead Code Manager (CM), and the Unified Forecast System (UFS) Application Lead. As releases and release planning become more frequent, these meetings may transition from a commitment for the duration of the release to an ongoing weekly commitment.

ii. Attendees

Attendees will include:

1. EPIC AUS Team members working on the release:
    a. AUS Product Owner
    b. AUS Scrum Master
    c. Other team members as needed
2. EPIC Platform Team members supporting the release:
    a. Platform Team Product Owner
    b. EPIC Configuration and Code Managers
    c. Data Subteam representative(s)
3. UFS and UFSR2O Application Leads
4. Representatives from component repositories
5. Leads/Members of the UFS Release Coordination Cross-Cutting Team (once formed)

The following team members are essential for release progress, and their availability should be prioritized when scheduling meetings:

1. AUS Product Owner
2. AUS Scrum Master
3. UFS Application Lead
4. Lead Code Manager for the application
5. CCPP Representative

iii. Roles & Responsibilities

The following is a summary of the roles and responsibilities of Release Team members. Additional roles and responsibilities may be added based on the demands of a particular release. The time commitment required will also vary based on the application and type of release.

a) AUS

AUS Scrum Master: Leads Release Team meetings once per week. Tracks & assesses progress; communicates status to the EPT.

AUS Product Owner: Coordinates the AUS Team technical contributions to the release. Additionally, runs Release Team meetings if the AUS Scrum Master is unable to do so.

AUS Team Members: Support development, testing, and documentation efforts for the release.

b) Platform

Code Managers: Enforce code management policies for the repository they manage. Provide guidance on the release plan, particularly the development and testing-related processes. Cut the release branch for the repository they manage.

Data Subteam: Stage data on the application's data bucket. If needed, create a new data bucket for the application.

c) ECE

Content Publishers: Add release-related information to the EPIC Community Portal (ECP) under the appropriate landing page; publish social media posts announcing the release; liaise with communications teams external to EPIC (e.g., the UFS website manager). This work will primarily take place in the final week before the release and will take approximately three capacity points.

d) Non-EPIC Roles

UFS Application Lead: A senior developer for the application who offers insight into the current state of the application and planned capabilities for the application. Attends the weekly Release Team meeting one hour per week.

UFS RC CCT: Provides guidance for release goals and timeline. Members attend an hour-long meeting once per month to discuss priorities for upcoming releases. Members may also attend the weekly application release team meetings for the repositories they are involved in.

Representatives from Component Repositories: Attend Release Team meetings one hour per week and serve as a liaison to their repository. Cut branches for component repositories at Code Slush if those repositories are not managed by EPIC CMs. Help prioritize and resolve critical issues that arise during release testing. Ensure that component repository documentation is updated ahead of a release.

- **NOTES:**
    - The workload for contributors to these repositories will vary depending on the state of the component code, the documentation, and the list of features that developers for the repository agree to add for the release. This list of features for the release can vary from zero, where EPIC takes a well-tested hash from the development branch, to an extensive feature list. In the case of an extensive list, these features typically correspond to work that the developers have already planned to complete within the release timeframe.
    - This workload estimate also assumes that adequate testing and documentation is completed as part of the usual development and pull request process. If not, there may be additional testing and documentation needs.
    - As EPIC familiarizes itself with new applications and functionality, representatives from component repositories may need to answer questions by email or via occasional short meetings so that EPIC is able to build, run, and contribute independently to various applications and components. This is an up-front cost, but it is necessary for knowledge transfer and will ultimately unburden representatives from these repositories in the long term as EPIC is able to share the work.

B. Scope

During the initial release meeting(s), the Release Team will outline the scope of the release. At a minimum, this will include the type of release (major, minor, or bug-fix) and a discussion of the supported platforms. Although the nature of a release may be unclear in practice, in principle, the following definitions hold, and the Release Team should determine which definition fits the release description most closely:

- ***Bug-fix release:*** Composed of bug-fixes since the previous release.
- ***Minor release:*** Includes updates and/or incremental improvements to existing capabilities. May also include bug fixes.
- ***Major release:*** Involves the addition of significant *new* capabilities, in addition to more minor updates and bug fixes.

Unless otherwise noted, the standard set of supported platforms will be:

Supported Platform & Compiler Pairings	
Level 1 Platforms	***Level 2 Platforms***
Cheyenne (Intel/GNU)	*None at this time*
Gaea (Intel)	***Level 3 Platforms (Limited Test)***
Hera (Intel)	Generic Linux Ubuntu (GNU/Intel)
Jet (Intel)	Generic Mac (GNU/Intel)
Orion (Intel)	
NOAA Cloud (Intel/GNU)	
Container (Linux Ubuntu w/Intel)	

The Release Team will determine whether any exceptions must be made for supported Level 1 & 2 platforms. Then, the team will select or update specific operating system and compiler combinations for supported Level 3 systems and decide whether to add any Level 4 systems.

It is EPIC's charter to improve community involvement with the UFS via public releases of UFS applications, and those releases must coincide with clearly defined milestones in application development. With guidance from the RC CCT and the UFS Steering Committee (UFS-SC), the Release Team will therefore identify which features will be included in the release and which UFS application component repositories and supporting systems will be involved (e.g., WM, CCPP, UFS_UTILS, etc.). The Release Team will also identify dependencies involving other EPIC teams and solicit feedback, participation, and open communication from the beginning of the release planning efforts. The Release Team will then establish an initial timeline for completion of new features. Additionally, the team will outline a preliminary testing plan and data requirements.

i. Feature Set

The ***desired feature set*** for the release will include a breakdown of features that must be in the release ("essential features") and features that the team would like to add to the release if possible ("desirable features"). The list of desirable features can be pared down in case of a descoping of the release. The scope of each feature, reason(s) for including it in the release, and reasons for classifying it as essential versus desirable will be noted in the release plan. Any features that are not already complete during the initial release planning phase should have an issue associated with them. If one does not already exist it should be filed. These issues should be added to the release's GitHub project for tracking.

When elaborating the desired feature set, the Release Team will consider each application component and designate a point of contact (POC) and back-up POC for each component repository. These POCs must be code managers for their respective repository.

(a) Components

1. UFS Weather Model
    a. FV3 Framework (fv3atm)
    b. FV3 Dynamical Core (atmos_cubed_sphere)
    c. Subcomponent Models & Couplers: MOM6, CICE6, CMEPS, CDEPS, GOCART, AQM, HYCOM, WW3
    d. CMakeModules
2. UFS_UTILS
3. CCPP
    a. Framework
    b. Physics
        i. Earth System Radiation Submodule
4. Stochastic Physics
5. Unified Post Processor (UPP)
6. Workflow
    a. Global Workflow
7. Software Stack
    a. HPC-Stack
    b. Spack-Stack
8. METplus Verification
9. Other (e.g., GSI, CRTM)

In collaboration with the UFS Weather Model (WM) code management group, the Release Team will inquire about upcoming releases to the software stack that could affect the release (particularly updates to ESMF and FMS). The Release Team will also determine whether any previously supported capabilities will be deprecated with the new release. If so, the Release Team will delegate a team member to post a deprecation announcement on the repository's wiki page, in the ufs-community GitHub Discussions page, and on the application-level GitHub Discussions page. Additionally, AUS will send the deprecation announcement to the ECE team to post on EPIC's social media platforms.

The Release Team will use release branches, rather than tags, for major releases. These branches will follow the naming convention "*release/public-v#.#.#*". An initial release should be labeled "*release/public-v1.0.0*". Bug-fix releases will increment the right-most digit, minor releases increment the center digit, and major releases increment the first digit. The Release Team must identify repositories that will require a release branch. This decision should be based on the likelihood of needing hot-fixes or release-specific commits to the repository. The Team should also plan to use a well-tested hash of the WM as the basis for the WM release branch.

For minor releases, the Release Team will cut a release branch for the application-level repository, but it will primarily use tags/hashes (not branches) for subcomponent repositories. The release hashes will correspond to the hashes used in the application development branch when the application release branch is cut. These hashes are typically close to the top of each subcomponent's development branch, which will allow EPIC to provide the latest feature updates to the public in any minor release while minimizing the overhead required when cutting and maintaining release branches. Since minor releases largely draw on well-tested code and simply update already-existing capabilities, there is less risk that post-release bug fixes will be required.

ii. Testing Plan

After determining the desired feature set for the release, the Release Team will draft a ***preliminary testing plan*** and delegate a testing team in charge of testing plan development/elaboration over the course of the release. The testing team should include members from the EPIC Platform Team and from the Developmental Testbed Center (DTC). At least one person on this team should have experience with CI/CD and automated testing in Jenkins. If any testing team members do not have access to Jenkins and believe they will need it for their duties, they should request access at this point.

The preliminary testing plan will delineate a minimum set of tests to run and indicate what data is required to support them. The testing plan may evolve over the course of the release but will provide initial direction for the release. As new capabilities are added to the release, the testing team will need to expand the list of tests to cover new functionality. The plan should address the following questions:

- What functionality will be tested?
- Which tests will be run (fundamental, comprehensive, regression, unit, other)?
- On which platforms will tests be run?
- When/how often will tests be run?
- What constitutes a passing test?
- How will testing results be reported?
- What scientific testing will be performed (if any)?

- What manual tests might be necessary (if any)?
- How will containers be tested?
- How will supported graphics scripts be tested?
- What data is required to support the tests?
- Are there any concerns regarding the computational cost of testing or the testing resources available?

The testing team (in consultation with the AUS Product Owner) will be in charge of maintaining and updating a list of fundamental and comprehensive tests to run on the release code. They will also be responsible for selecting the subset of tests that will be supported for the release.

Testing will proceed more smoothly if regular testing of the development branch has already been implemented. Such testing would include a standard set of tests that is run at defined intervals. This regular development branch testing would ensure that when the release branches are cut, the testing team is starting with well-tested code and can focus on testing only new/supported features on the release branch. However, where regular testing is not already implemented on the development branch, the testing team will need to assess whether/how to implement a set of tests to accomplish this purpose. Alternatively, they will need to account for additional testing and bug fix time in the Testing Phase. This should be captured in the release timeline. Science tests will be the responsibility of the application or component teams, not the Release Team.

The Release Team will identify an ***"out-of-the-box" case*** (often based on an established test(s)) to use throughout the documentation, particularly in the Quick Start Guide. Additional cases may be identified to form the basis for a Tutorial chapter. The tutorial would explain to users how to run the application, change elements of the configuration, rerun, examine the output, create graphics, and compare the results.

iii. Data Requirements:

The Release Team will determine the data dependencies for the release with input from the application developers and EPIC's Data Subteam. This will include a determination of whether Graduate Student Tests (GSTs), case files with initial and boundary conditions, or other relevant data are available. The Release Team will communicate these dependencies to the Data Subteam and to the other developers within the release plan. The Release Team will also delegate team members to install data in standardized directories on all Level 1 platforms. The data dependencies should include the collection of files necessary to run the:

- Out-of-the-box case
- Supported regression or end-to-end (E2E) test cases
- Supported validation/verification (e.g., METplus)
- Supported graphics plotting (Natural Earth/Cartopy files)
- Additional tutorial case(s) (if included)

If the application does not yet have a data bucket and landing page, the Release Team will notify EPIC's Data Subteam contact (currently Sylvia Chin on the Platform Team) that one will be required and create a plan for adding the bucket.

C. Timeline/Delivery Dates

The Release Team will outline a timeline and delivery dates for the release. If the desired feature list is unlikely to be completed by a contractual deadline for a release, the scope of the release will need to be revised and/or an extension discussed with EPIC/NOAA leadership. The timeline should include:

- Reasonable goal dates for feature completion and testing (based on stakeholder/developer capacity)
- Rules for revising due dates
    - When is an extension appropriate?
    - When is a descoping appropriate?
- When is final component documentation due? (No later than one week after the code freeze.)

The release dates will be established according to the readiness of the code and in coordination with the UFS RC CCT. If a release is tied to a contractual deliverable, and the release needs to be pushed back to the next contract year due to delays in essential functionality, the AUS Scrum Master and Product Owner will request a modification of the contract from EPIC's Raytheon and NOAA Program Managers.

D. Information Sharing & Communication:

The Release Team will delegate a person to create a GitHub Project and a Google Drive Folder on the Shared RI&S Drive for sharing information related to the release.

The GitHub Project will be used to track tasks, issues, and PRs relevant to the release. All issues and PRs related to the release should be linked to the GitHub Project. Tickets should include a title, status, assignee, and due date. The due date field and other non-default fields will need to be added to the basic board. Other fields, including "reviewer" may be assigned if relevant. For a basic tutorial on GitHub Projects, see here.

A Google Drive folder will hold any documents relevant to the release and will be open for viewing/commenting to anyone with a NOAA email. The tasks in the project can and should link to these documents if relevant. Documents may include the Release Plan, once developed, as well as the release meeting notes, a list/spreadsheet of proposed tests (including WE2E tests, unit tests, regression tests) and physics suites, contact information, etc. Edit permissions will be granted to team members on a case-by-case basis.

The Release Team will communicate via email, Slack, and GitHub, as appropriate. When the release plan has been developed, it will be shared via email with members of the application development team, the lead CM and/or code management team, and relevant stakeholders. The email should communicate tentative dates for the phases of the release and due dates for important line items. Recipients will have a week to suggest alterations and offer feedback.

At least six weeks prior to the release date, the AUS Team will submit the release plan, approved by the Release Team, to the deliverable review process. At least four weeks prior to the release, the release plan will be submitted to the EPIC Program Team (EPT) as an EPIC contractual deliverable (see EPIC RELEASE DELIVERABLES). The release plan will include the release timeline, desired feature set, and supported platforms and compilers.

III. DESIGN & DEVELOPMENT

During the Design & Development phase, release-related commits should, when possible, be prioritized for review. Application code managers, with support from the Release Team, may maintain a Commit Queue for release-related commits either on the GitHub wiki (similar to the UFS Weather Model C ommit Queue) and/or in GitHub Projects.

The Release Team needs to communicate with the code managers of the application and components to ensure that no **major** changes beyond the agreed-upon features come into the development branch during the Design & Development period (or that if they do, they are mutually agreed upon and well-documented). In cases where such additions to the development branch are unavoidable and/or delaying the change is undesirable, the Release Team will evaluate how to proceed. The team may choose to incorporate the change into the release, cut the release branch early, or cherry-pick commits from the development branch into the release branch at a later date. If the change affects a Weather Model submodule, all reasonable effort should be made to avoid cutting the release branch early.

If a component release branch *is* cut early, there must be a designated POC for the affected repository. This POC will notify the AUS Scrum Master when changes that are put into the development branch need to be added to the release branch. In general, there should be regular communication between the POC and AUS, and AUS will need to test the components thoroughly before incorporating them into the release to ensure that all components work together as expected.

When possible, developers should give the EPIC Release Team notice when a major PR is being prepared (such as the switch from env files to modulefiles or the addition of the Python workflow during the SRW App v2.0.0 release). This will give the Release Team more time to evaluate the situation and settle on a path forward.

As the code slush date approaches, the Release Team will collaborate with code managers to designate a person from each component repository to cut the release branch. At this point, the Release Team must also determine who will be in charge of approvals for the release branch (e.g., EPIC-AUS, the UFS code managers, or both) and how many approvals will be required on a release branch PR.

A. Testing Plan

As development continues, the testing team will be responsible for updating the testing plan. This involves updating the list of tests that will ultimately be run and the list of tests provided to users and supported for the release. These tests may include any combination of workflow end-to-end (WE2E) tests, unit tests, regression tests, and science tests. Additional testing categories may be added if necessary. During this phase, the testing team will design Jenkins recipes that can be used in the testing phase of the release. All of EPIC's release testing will be done in Jenkins, unless an exception is noted in the release plan. However, other automated testing (e.g., via GitHub actions) may continue as usual per the repository's policies. A testing team delegate will report updates to the testing plan at weekly meetings.

B. Release meetings

All release meetings will be led by the EPIC AUS Scrum Master and should address the following issues:

- Kanban Board updates: add tasks, report/update status of tasks
    - Commit Queue updates: assign reviewers and due dates for newly opened PRs
    - Update release documents (e.g., list of commits that will need to go back into develop, list of tests)
- Are there any major PRs in progress not already anticipated for the release?
- Any upcoming updates to the software stack?
- Testing Plan Updates

IV. CODE SLUSH

Code Slush refers to code that is in a semi-frozen state. The code slush date is the target date by which all desired features for the release should be merged. No major changes/new features should come into the code base after this date. However, the code may require minor modifications/tweaks to fix bugs or minor problems with the code that are caught during testing. For the code slush to occur, all "essential features" must be in a fairly complete state. The AUS Product Owner and Scrum Master collectively make the call whether to extend the deadline for the Code Slush. Any "desirable features" (aka non-essential features) not finished when the Code Slush occurs will be excluded from the release. Every effort should be made to keep the time between the Code Slush and Code Freeze as short as possible.

A. Major Release: Cut Release Branches (<3 days)

Once the Release Team verifies that all features have been merged, and the component repositories are ready, the AUS Scrum Master will email the code managers designated to cut the release branches for the component repositories. This ensures that if key individuals were absent at the weekly meeting, they will still be notified to cut the release branch. The release branches will be cut according to the code management procedures for each repository's development branch. The release branches should follow the standard naming convention for releases. Representatives of each component repository must notify the AUS Scrum Master or their delegate when the release branch has been cut.

B. Minor Release

Once the Release Team verifies that all features have been merged, and the component repositories are ready, the AUS Scrum Master will notify the code manager for the application-level repository to cut the application release branch. The subcomponent repositories currently tagged in the development branch will be used as the subcomponent release hashes.

Subcomponent code managers will label the hash used in the application release with a release tag. Tags for an application release should follow the naming convention "ufs-app-v#.#.#", where "app" refers to the abbreviation for the application (e.g., srw, mrw, s2s, hafs). However, this convention may be amended at a later date as UFS code managers standardize naming conventions across UFS repositories.

C. Update Externals in Applications Where *manage_externals* Is Used

When the release branch is cut at the application level, files (e.g., *Externals.cfg*) that pull in external repositories should be altered to point to the HEAD of each external component release branch rather than to static hashes on those release branches. This will ensure that testing always occurs using the most up-to-date code contributions from component release branches. There may be exceptions to this practice based on code management practices for the application or model being released. In such cases, a policy for updating the externals' hashes with each PR should be discussed if it does not already exist. Immediately prior to the release, *Externals.cfg* should again be modified to point back to static hashes. This will ensure that users are always pulling the same stable release code.

D. Release-Related Pull Requests (PRs)

Starting with the Code Slush, when a PR is applicable to both the development branch and the release branch, it should be placed in the release branch first with a release label and a title that indicates that it belongs in the release (e.g., *[release]: <Title>*). Developers must keep track of any PRs merged directly into the release branch that will need to be added back into the development branch. Since few PRs should be required after the Code Slush, this list should be extremely short. However, the Release Team will distribute a Google Doc/Spreadsheet or other tool to track the limited number of PRs that need to be tested and merged into the development branch. Reminders to update this list will be issued at release meetings.

E. Subcomponent Documentation

The Technical Writer will email those in charge of subcomponent documentation with updated deadlines for completing the subcomponent documentation.

V. TESTING & COMPONENT DOCUMENTATION

A. Testing

The testing team will follow the testing and reporting plan outlined in the Planning phase and updated during the Design & Development phase. At a minimum, the team must test:

- On all supported platforms
- Using Jenkins on Level 1 platforms
- Container & non-container options
- Graphics (if supported)

B. Documentation

i. Components Documentation

The Technical Writer will send an email reminder to those in charge of subcomponent documentation with updated deadlines for the subcomponent release documentation. The email will request that relevant parties send a final documentation link(s) to the AUS Technical Writer within a week of the planned code freeze date. If component repositories have information or documentation that is application-specific (rather than component-specific), it should be included in the application-level documentation. Individuals in charge of documentation are encouraged to communicate with the Technical Writer to provide such information before the code freeze date.

ii. Application Documentation:

The Technical Writer will update documentation in the release branch to reflect information for the release. This includes the following changes:

- Update git clone command to reflect the release branch (not develop)
- Update data location paths in the documentation to reflect the release folder (e.g., v2p0)
- Update version numbers for physics, components (e.g., CCPP v6.0.0, WM v3.0.0)
- Update images (if necessary)
- Add alt text to images
- Complete first-pass edits on all chapters

Any code changes in the workflow or application-level repositories that require documentation updates must be communicated to the EPIC Technical Writer by the code freeze date and/or included in a PR. Developers are encouraged to update the .rst files themselves until the code freeze date. After the code freeze date, the Technical Writer should be notified in order to edit and/or review any documentation updates. This will prevent broken links and formatting errors on the ReadTheDocs site.

C. Data Bucket

If the app does not yet have a data bucket landing page, the Technical Writer is responsible for drafting this and forwarding it to the Platform Team member in charge of data buckets. If a page already exists, it should be reviewed and updated if necessary.

An AUS team member will be assigned to place tar files containing current release data and, if necessary, fix file and plotting data onto Orion. Then the Platform Team contact (currently Sylvia Chin) must be notified of where the data is staged on Orion once it is ready for inclusion in a data bucket. The Platform Team will inform the Release Team when the data is staged in the bucket.

VI. CODE FREEZE (2-3 weeks prior to release):

The code freeze occurs when the code is finished, and no known issues remain after initial testing. At this point, only documentation changes are permitted. No changes affecting the functionality of the application can be pushed because final platform testing has begun to ensure readiness for a release. If critical bugs are found during this final testing period, the AUS Product Owner will decide whether to revert the code freeze and push the changes or release the application after documenting known bugs (and, if applicable, their workarounds). If the code freeze is reverted, any final testing already completed will need to be redone, unless the changes made *cannot* affect testing on certain platforms (such as containers). If the bug fix takes more than one week, the final release date will need to be postponed.

A. Final Testing

The Release Team will designate team members (preferably from the testing team) to test the code on each Level 1 platform via Jenkins and record results in a Drive file/spreadsheet.

B. Documentation

Component documentation should be sent to the EPIC Technical Writer no later than one week after the code freeze date.

Beginning with the code freeze, any new application-level PRs updating documentation must be communicated to the EPIC Technical Writer for review. Each chapter must undergo a final review/edit. This should begin with the least changeable chapters first (e.g., Rocoto Info, Glossary) in case any last-minute bug fixes result in changes to the more variable chapters (e.g., Quickstart, Build/Run, or Testing). At this time, the Technical Writer will also create a **Zenodo** citation for the UFS application following these instructions. The Technical Writer will:

- Update the directory structure information in the Introduction
- Update all links to each component's release documentation. (To update links, search each chapter of the docs using *Ctrl+F http* rather than component-specific terms.)
- Resolve/remove all comments from the documentation.
- Update the Introduction chapter and the README.md file with the correct citation information, including the new DOI and tentative release date.

If there are any documentation-related submodules, they need to be updated to reflect the HEAD of the *release* branch of their repository. In general, use of Intersphinx mapping should be preferred to submodules when the submodule is only required for documentation.

i. Documentation Testing

The Technical Writer and other Release Team members will conduct manual testing of the release documentation, particularly the Quickstart and Build /Run chapters. For this testing, team members must copy-paste commands into the command line, only making alterations where indicated (e.g., to code paths within <>). This will ensure that there are no typos. The documentation should be tested on a handful of Level 1 platforms (both cloud and non-cloud) and in containers (both cloud and non-cloud); it does not need to be tested on every system.

ii. Other

The Technical Writer will also verify the following:

- Does the git clone command reflect the release branch?
- Are all data paths updated?
    - *Ctrl + F UFS_SRW_App* and/or "*develop*"
- Are version numbers for physics and components updated (e.g., CCPP v6.0.0, WM v3.0.0)?
- Have images been updated and added to git? (This should be verified on the authoritative repository's ReadTheDocs (RTD) website, not a local copy, to ensure that images are rendering properly.)
- Is there alt text for each image?

VII. PRE-RELEASE (1 week prior to release)

When all testing is complete and documentation is finished, the Release Team will set a final release date and coordinate the final details of the release. The Technical Writer will update the release date on Zenodo, in the Introduction section of the documentation, and in the README.

A. Merge Updates to Develop

All team members involved in the release should review the release document that lists PRs made directly to the release branch (if any) to ensure that their own PRs are included on that list. The Release Team will delegate a person (or people) to merge those PRs (or pertinent portions of them) back into the development branch. This will act as a final review to catch any code errors or typos in the documentation.

At this time, if the application uses the *manage_externals* tool, the lead CM will switch *Externals.cfg* to point at static component hashes rather than at the head of a component release branch. This step will only be required in applications that use the *manage_externals* utility, and it will only be relevant to components that use a release branch.

B. AUS/Release Team and ECE Team Collaboration

i. AUS Team

One week prior to the release, AUS will notify the ECE Team and the UFS website manager of the projected release date.

The AUS team will follow the ECE team Website Content SOP to provide a release announcement of approximately two paragraphs and will generate and select graphics to include with the release posts and announcements. AUS will also propose a list of EPIC Community Portal updates. This will include updates to content and links on the application-specific "Get Code" landing page. Additionally, AUS will provide a short release summary of 280 characters or less that can be used for social media posts. Per the Website Content SOP, this will be posted on a Confluence page. In addition to the normal approvals, the Confluence page must also be approved by the EPT PM.

AUS will also provide the ECE Team with a release description and links (e.g., to documentation) for the ECE Team to forward to the UFS Team. The UFS Team will use the information to update their How to Access Code? page. If desired, the Release Team can also provide an updated version of the UFS Application page (e.g., here for the SRW).

The AUS Technical Writer will update the wiki on the application's GitHub repository.

ii. ECE Team

The ECE Team will update the News page and Application page (under “Get Code”) on the EPIC Community Portal using the release announcement and recommended updates provided by AUS. ECE will provide guidance for any additional changes required to the release summary to make it appropriate for social media. They will add links back to the news article and/or to the application landing page once those have been generated/updated. They will also post (or delegate an AUS team member to post) a release announcement under “General” in the GitHub Discussions for the *ufs-community* page and for the specific repository (e.g., here for SRW).

The ECE Team will reach out to the UFS Community, NOAA’s communications team, and Raytheon to convey the information from the News article and landing page. These offices will use the information provided to update their own websites as needed. The ECE Team should verify that the release announcement will be posted on the UFS Community Portal under *What’s New?* and on the UFS Community Forums (until deprecated). They should also verify that the UFS team will update the How to Access Code? page and (if applicable) the application page with the information provided by AUS. They should also request that the UFS update their website banner with the new release information.

NOTE: In the course of its regular responsibilities, the ECE Team develops and conducts practical training sessions serving the broader community (e.g., AMS short courses, workshops at AMS and AGU). It also produces training materials, including online tutorials, instructional videos, and training presentation slides, many of which are publicly available. In general, these training materials use the latest application release. Sample training materials can be viewed on the EPIC Community Portal (ECP) at https://epic.noaa.gov/tutorials/. Application-specific materials, such as documentation, are linked on the application-specific ECP page (e.g., https://epic.noaa.gov/get-code/short-range-weather/ for the SRW Application).

VIII. RELEASE

On the morning of the release, the ECE team will publish social media posts announcing the release. ECE will also post the release announcement and/or news article provided by the AUS Scrum Master on the EPIC website. Similarly, the UFS will post updates to their website to reflect the new release. The AUS Technical Writer will update the main repository’s wiki page with a release announcement, links to new documentation, and any other pertinent information.

All release-related EPIC deliverables not already submitted to the EPIC Deliverable Workflow for approval should be submitted on the day of the release (see EPIC RELEASE DELIVERABLES).

IX. POST-RELEASE

A. Release Retrospective

Following the release, a release retrospective will be scheduled with the Release Team and any stakeholders who want to offer input. This meeting will be scheduled and facilitated by the AUS Scrum Master.

The goal of the retrospective is to identify what went well during the release process and what could be improved in the future. The output of the retrospective is retrospective notes and (if applicable) updates to the Release Process document. If changes to the release process are significant, a new version of the Release Process document will be created.

Topics that require further discussion and decisions should be taken to the Release Coordination Cross-Cutting Team (RC CCT).

B. Gather Release Feedback

TBD following discussion by the RC CCT.

X. EPIC RELEASE DELIVERABLES

A. Release Plan

The release plan is formulated during the Release Planning phase and is approved by the Release Team. The release plan includes the release timeline, desired feature set, and supported platforms and compilers. The release plan must be approved and submitted no later than four weeks prior to the target release date.

B. The Release

There will be an EPIC deliverable for the release. The evidence provided can be a link to the release branch or tag in the authoritative repository and a link to the updated, versioned documentation. A link to the release announcement should also be submitted. This deliverable should be submitted to the EPIC Deliverable Workflow for approval on the day of the release.

C. Report Demonstrating Code Reproducibility and Benchmarking in the Cloud and On-Premises HPC Resources

If cloud platforms are part of the supported platforms for the release, a report demonstrating code reproducibility and benchmarking in the cloud and on-premises HPC resources will need to be submitted**.** This report is designed to show that the released code delivers the same results when tested on the three cloud providers and on one on-premises HPC resource, proving that results across platforms are reproducible. This deliverable should be submitted in the form of a document or set of documents. See the SRW 2.1.0 deliverable for an example of supporting documentation. This should be submitted to the EPIC Deliverable Workflow for approval on the day of the release.

D. Sample Forecasts

If there are sample forecasts already provided for the application being released, these cases should be run with the new release. These versioned, updated sample forecasts should be made available to the community with the new release. The links to the new version of the sample forecasts should be provided as the evidence for the deliverable. This deliverable should be submitted to the EPIC Deliverable Workflow for approval on the day of the release.

XI. APPENDIX

A. Contact List for UFS Component Repositories

Repository Link	Contact Person(s)	Email
ufs-weather-model	Ratko Vasic (SRW)	ratko.vasic@noaa.gov
	Denise Worthen	denise.worthen@noaa.gov
	Jong Kim (Lead CM)	jong.kim@noaa.gov
	Brian Curtis	brian.curtis@noaa.gov
FV3	Jun Wang	jun.wang@noaa.gov
	Dusan Jovic	dusan.jovic@noaa.gov
atmos_cubed_sphere	Lauren Chilutti	lauren.chilutti@noaa.gov
	Rusty Benson	rusty.benson@noaa.gov
CMakeModules		
UFS_UTILS	Jeff Beck	jeff.beck@noaa.gov
	George Gayno	george.gayno@noaa.gov
	Larissa Reames	larissa.reames@noaa.gov
CCPP-physics	Ligia Bernardet	ligia.bernardet@noaa.gov
	Lulin Xue	xuel@ucar.edu
	Grant Firl	grant.firl@noaa.gov
	Dustin Swales	dswales@ucar.edu
CCPP-framework	Mike Kavulich	kavulich@ucar.edu
	Ligia Bernardet	ligia.bernardet@noaa.gov
	Lulin Xue	xuel@ucar.edu
stochastic_physics	Phil Pegion	philip.pegion@noaa.gov
UPP	Jong Kim	jong.kim@noaa.gov
	Kate Fossell	fossell@ucar.edu
	Tracy Hertneky	hertneky@ucar.edu
regional_workflow	Gerard Ketefian	gerard.ketefian@noaa.gov
global-workflow	Kate Friedman	kate.friedman@noaa.gov
	Walter Kolczynski	walter.kolczynski@noaa.gov
HAFS	Bin Liu	bin.liu@noaa.gov
	Zhan Zhang	zhan.zhang@noaa.gov
	Kathryn Newman	knewman@ucar.edu
	Linlin Pan	Linin.Pan@noaa.gov
HPC-Stack	Alex Richert	alexander.richert@noaa.gov
spack-stack	Dom Heinzeller	dom.heinzeller@noaa.gov
METplus Verification	Tara Jensen	jensen@ucar.edu
EMC_verif-global	Tara Jensen	jensen@ucar.edu
	Mallory Row	mallory.row@noaa.gov
Rocoto	Christopher Harrop	christopher.w.harrop@noaa.gov
AQM	Jianping Huang	jianping.huang@noaa.gov
	Brian Curtis	brian.curtis@noaa.gov

MOM6	Jiande Wang	jiande.wang@noaa.gov
CICE6	Denise Worthen	denise.worthen@noaa.gov
CMEPS	Denise Worthen	denise.worthen@noaa.gov
CDEPS	Bin Li	bin.li@noaa.gov
GOCART	Li Pan	li.pan@noaa.gov
WW3	Ali Abdolali Jessica Meixner	ali.abdolali@noaa.gov jessica.meixner@noaa.gov
HYCOM	Daniel Rosen Bin Liu	daniel.rosen@noaa.gov bin.liu@noaa.gov
Data Bucket/Platform	Sylvia Chin	sylvia.chin@noaa.gov
Jenkins CI/CD	Jesse McFarland	jesse.mcfarland@noaa.gov

B. Release Process Checklist

i. Release Planning

- [ ] Identify Release Team members
- [ ] Schedule recurring Release Team meetings
- [ ] Create the GitHub project
- [ ] Create release planning document
- [ ] Determine and document target release schedule
- [ ] Determine and document desired features, including must-haves and nice-to-haves
- [ ] File issues for all desired features and add to the GitHub project
- [ ] Determine and document target platforms
- [ ] Create and document preliminary testing plan
- [ ] Determine and document data dependencies
- [ ] Distribute release plan via email to members of the application development team, the lead CM and/or code management team, and relevant stakeholders
- [ ] Submit the release plan as an EPIC deliverable

ii. Design & Development

- [ ] Update the testing plan
- [ ] Determine and document final feature set

iii. Code Slush

- [ ] Cut release branches
- [ ] Update externals in applications where *manage_externals* is used
- [ ] Email those in charge of subcomponent documentation with updated deadlines for completing the subcomponent documentation

iv. Testing & Component Documentation

- [ ] Test on all supported platforms
- [ ] Test using Jenkins on Level 1 platforms
- [ ] Test container & non-container options
- [ ] Test graphics (if supported)
- [ ] Update the git clone command in the application documentation to reflect the release branch
- [ ] Update the data location paths in the application documentation to reflect the release folder (e.g., v2p0)
- [ ] Update version numbers for physics, components (e.g., CCPP v6.0.0, WM v3.0.0) in the application documentation
- [ ] Update application documentation images (if necessary)
- [ ]

- [ ] Add alt text to images
- [ ] Complete first-pass edits on all documentation chapters
- [ ] Create or update the data bucket landing page
- [ ] Stage release data and any fix file data on Orion
- [ ] Put the release data from Orion into the data bucket

v. Code Freeze

- [ ] Create a spreadsheet to capture final testing results
- [ ] Designate Release Team members to perform final testing on each platform
- [ ] Record final testing results in the spreadsheet
- [ ] All component documentation is updated and sent to the technical writer
- [ ] Create the Zenodo citation
- [ ] Perform manual testing of the documentation
- [ ] Technical Writer finalizes documentation updates, including:
  - [ ] Update the directory structure information in the Introduction
  - [ ] Update all links to each component's release branch documentation
  - [ ] Resolve/remove all comments from the documentation
  - [ ] Update the Introduction chapter and the README.md file with the correct citation information, including the new DOI and tentative release date.
- [ ] Update sample forecasts (if applicable) and documentation, and add versioned sample forecast data to the data bucket

vi. Pre-release

- [ ] Determine final release date
- [ ] Update the release date in Zenodo
- [ ] Update the Introduction section of the documentation with the final release date
- [ ] Update the README with the final release date
- [ ] In applications where *manage_externals* is used, switch *Externals.cfg* to point at static component hashes
- [ ] Notify the ECE team and UFS Website Manager of the projected release date
- [ ] Add announcements to Confluence, per the Website Content SOP
- [ ] Update the News page and Application page (under "Get Code") on the EPIC Community Portal using the release announcement and recommended updates
- [ ] Create the report demonstrating code reproducibility and benchmarking in the cloud and on on-premises HPC resources

vii. Release

- [ ] Post all release announcements
- [ ] Update the main repository's wiki page with a release announcement, links to new documentation, and any other pertinent information
- [ ] Submit all release-related EPIC deliverables for review and approval

viii. Post-Release

- [ ] Schedule retrospective
- [ ] Facilitate retrospective and capture notes
- [ ] Update release process documentation where applicable
- [ ] Deliver a list of topics needing further discussion to the RC CCT