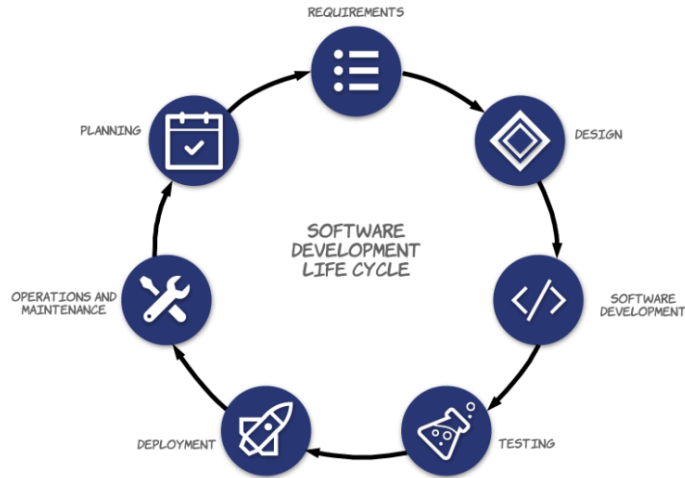




# **Earth Prediction Innovation Center (EPIC) Release Process for Unified Forecast System (UFS) Applications**

**Version 6**

**Jan 24, 2025**



## Change History

Date	Release	Changes
9/2/22	SRW App v2.0.0	Initial document generated to capture wisdom gained from the SRW App v2.0.0 release that could be applied to any UFS application release.
11/17/22	SRW App v2.1.0	Document updated to reflect information on procedures for minor releases; added roles & responsibilities section.
2/13/23	Land DA v1.0.0	Document updated to reflect new organization of EPIC teams. Added Land DA repository (land-offline_workflow) links and contact information. Added post-release actions and a checklist for the release process.
4/28/23	Land DA v1.0.0	Document updated following the Land DA v1.0.0 retrospective to better capture division of responsibilities and new EPIC teams. Updated links to point to new Confluence pages.
12/20/23	SRW App v2.2.0	Document updated following the SRW App v2.2.0 release to better capture division of responsibilities and new EPIC teams.
1/24/25	SRW App v3.0.0	Document updated ahead of the SRW App v3.0.0 release to better capture division of responsibilities, new release types, and new EPIC teams.

# Table of Contents:

<b>Earth Prediction Innovation Center (EPIC) Release Process for Unified Forecast System (UFS) Applications</b>	<b>1</b>
Change History	2
Table of Contents:	4
<b>I. INTRODUCTORY NOTE</b>	<b>6</b>
<b>II. INITIAL RELEASE PLANNING PHASE</b>	<b>6</b>
A. Release Meetings:	6
i. Schedule	6
ii. Attendees	6
iii. Announcements and (for feature-based releases) solicit community input	7
iv. Create a Release Planning Document	7
v. Establish Information Sharing and Communication Plan	7
vi. Delegate a Testing team and lead	7
vii. File Issues	8
viii. For Initial Releases:	8
ix. Update Release Planning Document with Release Plan	8
x. Submit and Distribute Release Plan	8
xi. Roles and Responsibilities	8
B. Scope	10
i. Feature Set	11
ii. Testing Plan	13
iii. Data Requirements	14
C. Timeline/Delivery Dates	14
D. Information Sharing & Communication	15
E. Initial Application Releases	16
<b>III. DESIGN &amp; DEVELOPMENT (10 weeks prior to release)</b>	<b>16</b>
A. Testing Plan	18
B. Data bucket	18
C. Release Meetings	18
D. Delays	18
<b>IV. CODE SLUSH (4 weeks prior to release)</b>	<b>18</b>
A. Major Release: Cut Release Branches (<3 days)	19
B. Minor Release	19
C. Update Externals in Applications Where manage_externals Is Used	19
D. Release-Related Pull Requests (PRs)	19
E. Subcomponent Documentation	20
F. Cut and Tag Branches	20
G. Data Update and testing	20

H. Demonstration	20
<b>V. TESTING &amp; COMPONENT DOCUMENTATION</b>	<b>20</b>
A. Testing	20
B. Documentation	20
i. Components Documentation	20
ii. Application Documentation:	21
C. Data Bucket	21
<b>VI. CODE FREEZE (3 weeks prior to release):</b>	<b>22</b>
A. Update manage_externals	22
B. Final Testing	22
C. Containers	22
D. Documentation	22
i. Documentation Testing	22
<b>VII. PRE-RELEASE (2 weeks prior to release)</b>	<b>23</b>
A. Release Publication Collaboration	23
i. SI Team	23
ii. US Team	23
iii. ECE Team	23
iv. NOAA Program Manager	24
VIII. PRE-RELEASE (1 week prior to release)	24
i. SI/CM Team	24
ii. US Team	25
iii. CM Team PO	25
IX. Content Freeze	25
<b>X. RELEASE</b>	<b>25</b>
i. ECE Team PO	25
ii. US Team Technical Writer	26
<b>XI. POST-RELEASE</b>	<b>26</b>
A. Merge Updates to Develop	26
B. Release Retrospective	26
<b>XII. EPIC RELEASE DELIVERABLES</b>	<b>27</b>
A. Release Plan	27
B. The Release	27
C. Report Demonstrating Code Reproducibility and Benchmarking in the Cloud and On-Premises HPC Resources	27
D. Sample Forecasts	27
<b>XIII. APPENDIX</b>	<b>27</b>
Release Process Checklist	27



## 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 (SMEs); 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).

The RC CCT Charter describes two types of releases: Full Application Releases and Capability Announcements. The feature-based releases and continuous releases referenced in this document are subsets of the Full Application Release.

**Please note:** Some links may only be available while on an approved VPN. For access issues, please contact the Office of the Chief Engineer: [Keven Blackman](#) or [Alex Burrows](#).

This document fulfills the requirements from PRS #2 in PWS 4.6.1:

Per 4.6.1, Update the UFS Release Coordination Vision and Release Coordination CCT Charter for review with UFS Steering Committee

Due - By the end of the third PI in the PoP following the milestones table specified in the PMP. - Online and MS Word Report via email; PowerPoint presentation to UFS-SC

PRS:

- Incorporate perspectives on of HTF and HSD frameworks from the EPIC team and key stakeholders

- **Update the EPIC release process documents to define feature vs. public releases**

## II. INITIAL 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 Software Integration (SI) Team Scrum Master, SI 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

The team or individual leading the release will determine release team members and set up release team meetings based on availability of key stakeholders and EPIC teams involved in the release. The team or individual leading the release will invite stakeholders to release meetings and communicate with component representatives and stakeholders regarding the level of participation required from those groups, including the expected communication and turnaround time.

Attendees may include:

1. EPIC SI Team members working on the release:
  - a. SI Product Owner
  - b. SI Scrum Master
  - c. Other team members as needed
2. EPIC Code Management Team members working on the release:
  - a. Code Management Team Scrum Master/Product Owner
  - b. Lead code manager for the application
  - c. Other EPIC configuration and code managers, as needed
3. Other EPIC team members supporting the release:
  - a. Data Subteam representative(s)
  - b. User Support (US) Team Lead and relevant team members
  - c. EPIC Community Engagement (ECE) Team Scrum Master/Product Owner
4. UFS and UFS-R2O Application Leads
5. Representatives from component repositories
6. Leads/Members of the UFS Release Coordination Cross-Cutting Team
7. Applicable stakeholders

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

1. SI Product Owner
2. SI Scrum Master
3. UFS Application Lead
4. Lead Code Manager for the application
5. Stakeholders
6. CCPP Representative (when a release contains the UFS Weather Model)

## iii. Release Notice

The EPIC team or individual leading the release must notify the following parties that a release is being planned and ensure that they are on release planning meeting invites: POs, SMs, and team leads for the SI, Code Management, ECE, and US teams.

The release must also be announced via GitHub in the ufs-community organizational space and in the repository from which the release will be issued. For feature-based releases, the announcement should solicit community input on features for the release. The US team is responsible for posting the release announcement on GitHub.

#### iv. Update Release Planning Document with Release Plan

As planning continues, the release plan will be updated to include:

1. Type of release (regular/continuous/capability announcement)
2. Target release schedule
3. Desired features including must-haves and nice-to-haves
4. List of supported platforms (including L3/L4 platforms)
5. Preliminary testing plan
6. Data dependencies
  - a. Determine “out-of-the-box” sample case for use in documentation
  - b. Determine which other data is required for the release

#### v. Submit and Distribute Release Plan

The release plan will be distributed to members of the application development team, the lead CM and/or code management team, and relevant stakeholders. The release plan will also be submitted as an EPIC deliverable if applicable.

#### vi. Roles and 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) SI Team

**SI Scrum Master:** Leads and schedules release team meetings once per week. Tracks and assesses progress; communicates status to the EPIC Program Team (EPT).

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

**SI Team Members:** Support development, testing, and documentation efforts for the release. Build a release container and stage it on all appropriate systems and in the S3 data bucket.

##### b) Code Management Team

**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.

c) **US Team**

**US Team Lead:** Coordinates documentation for the release. Drafts Zenodo citation, if required, and GitHub Release Announcement. Provides reviews and approvals for release-related publications.

**Data Subteam:** Advise on issues related to the application's data bucket. If needed, create a new data bucket for the application. Stage data on the application's data bucket and on Level 1 systems as needed.

d) **ECE Team**

**Content Publishers:** Add release-related information to the EPIC Community Portal (ECP) under the appropriate Get Code page; publish the news article and 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. All publications are reviewed by the key stakeholders before posting.

e) **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. Provide code and documentation reviews for the release team members. Provide graphics for the release announcement. During the release planning phase, the needs from these representatives should be discussed and documented, including expected communication and turnaround time.

- **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.

**NOTE: A comprehensive outline of what teams/team members are responsible for (depending on the type of release) is available in the [Release Checklist SOP](#) (requires Confluence access).**

## 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
Derecho (Intel) Gaea (C5) (Intel)	<i>None at this time</i>

Hera (Intel/GNU) Hercules (Intel/GNU) Jet (Intel) Orion (Intel) NOAA Cloud (Intel/GNU) Container (Linux Ubuntu w/Intel)	<b>Level 3 Platforms (Limited Test)</b>  Generic Linux Ubuntu (GNU/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. 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 or release tracking mechanism (if applicable).

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 repositories. Depending on the application being released, components may include, but are not limited to, the following:

#### 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](#)
  - (b) [land-offline\\_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, on the [ufs-community GitHub Discussions](#) page, and on the application-level GitHub Discussions page.

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 Code Management Team, SI Team, and (if applicable) 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 which 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:

- a) What functionality will be tested?
- b) Which tests will be run (fundamental, comprehensive, regression, unit, other)?
- c) On which platforms will tests be run?
- d) When/how often will tests be run?
- e) What constitutes a passing test?
- f) How will testing results be reported?
- g) What scientific testing will be performed (if any)?
- h) What manual tests might be necessary (if any)?
- i) How will containers be tested?
- j) How will supported graphics scripts be tested?
- k) What data is required to support the tests?
- l) Are there any concerns regarding the computational cost of testing or the testing resources available?

The testing team (in consultation with the lead application CM and SI PO) will be in charge of maintaining and updating a list of tests to run on the release code (e.g., regression tests, unit tests, fundamental and comprehensive tests). 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(s) 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 data dependencies, including a determination as to whether 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:

- A. Out-of-the-box case
- B. Supported regression or workflow end-to-end (WE2E) test cases
- C. Supported validation/verification (e.g., METplus)
- D. Supported graphics plotting (Natural Earth/Cartopy files)
- E. 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 **Brandon Selbig on the User Support 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 NOAA/EPIC leadership. The timeline should include:

- i. Reasonable goal dates for feature completion and testing (based on stakeholder/developer capacity)
- ii. Rules for revising due dates
  - a) When is an extension appropriate?
  - b) When is a descoping appropriate?
- iii. 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 SM and PO from the leading team will request a modification of the contract from EPIC's Raytheon and NOAA Program Managers.

#### **D. Information Sharing & Communication**

If applicable (depending on the type of release), 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, or GitHub, as appropriate. When the release plan has been developed, it will be shared with the lead code manager, members of the repository 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 SI 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 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. Subsequent changes to the timeline, supported platforms, or major features will need to be communicated to EPIC leadership if they occur after submission of the release plan.

#### **E. Initial Application Releases**

For the first release of an application (v1.0.0), additional steps must be taken during the release planning phase to adequately scope the release and prepare the timeline.

Before the timeline is determined, release readiness should be assessed. Considerations for release readiness include maturity of the build system, test maturity and stability, documentation, and portability. To assess release readiness, the Application Lead or designee should demonstrate for the release team how to build and run the code base and the tests. The release team can then outline the work needed, if any, to mature the application to a publicly releasable state. A plan should then be communicated to the Application Lead and all relevant stakeholders for agreement. If this shifts the release schedule, the findings should be reported to the EPT for a determination on schedule relief.

### III. DESIGN & DEVELOPMENT<sup>1</sup> (~10 weeks prior to release)

The Design & Development phase begins once the release team approves the Release Plan (generally one week after the Release Plan is shared). Features will be tracked in a GitHub project named [APP-NAME]-Release-v#. #.#. Issues will be treated as tasks and added to the project. Each issue will be assigned to a specific person. When a PR addressing the issue is opened, it will be linked to the appropriate issue, added to the GitHub Project, and reviewers will be assigned. For tasks pertaining to repositories that are not under the [ufs-community](#) organization, a task/note can be created, and any pertinent issues or PRs can be linked in the description. Assignees can update the status of tasks on the project as they complete them. The release team will also update the board and track progress via its weekly release meetings.

	Title	Assignees	Status	Reviewers	Due Date
1	Final documentation edits	gspetro-NOAA	Done		06/22/2022
2	Sample Item	gspetro-NOAA	Todo		12/31/2022

*Table View*

Column	Count	Task	Due Date
Todo	1	Draft Sample Item	12/31/2022
In Progress	0		
Done	1	Draft Final documentation edits	06/22/2022

*Kanban View*

<sup>1</sup> This is essentially a Feature Freeze, such that any new findings during coding and testing must feed back to here for impact assessment.

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 [Commit 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 efforts 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 lead application CM if/when action is required to incorporate the functionality into the application's release branch. In general, there should be regular communication between the POC and the Code Management Team, and the release testing team 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 code managers, 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 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 (if applicable). **For feature-based releases,** 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. Data bucket

A technical writer on the US team will finalize the text for the data bucket landing page. Brandon Selbig will request a new data bucket (if applicable).

## C. Release Meetings

All release meetings will be led by the SM for the team leading the release 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?
- Are there any upcoming updates to the software stack?
- Testing Plan Updates

## D. Delays

If a feature seems unlikely to be completed by the projected release date, the SM or PO from the leading team will communicate this to RTX management and include stakeholder recommendations (e.g., to descope a particular feature or delay release).

## IV. CODE SLUSH (4 weeks prior to release)

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 complete state. **The SM and PO for the lead team** collectively make the call on 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 lead application CM will contact the CMs designated to cut the release branches for the component repositories, if applicable. 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 lead application CM 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 lead CM for the application-level repository will cut the application release branch. The subcomponent repositories currently tagged in the development branch will be used as the subcomponent release hashes.

Subcomponent CMs 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).

## **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 the 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>*). The lead CM, with the assistance of the development team, 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. Data Update and Testing**

Depending on the type of release, either the SI Team PO or the lead CM for the application will update data paths in machine files (if applicable).

## **F. Demonstration**

Approximately four weeks prior to release, either the SI or CM Team PO will demonstrate the release to component representatives and stakeholders during a pre-release meeting or external meeting (optional).

# **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 (e.g., RDHPCS, NOAA Cloud)
- Using Jenkins on Level 1 platforms
- Container & non-container options
- Graphics (if supported)

## **B. Documentation**

### **i. Components Documentation**

The technical writer will send a reminder to those in charge of subcomponent documentation with updated deadlines for the subcomponent release documentation. The reminder will request that relevant parties send a final documentation link(s) to the 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:

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

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 application does not yet have a data bucket landing page, the technical writer is responsible for drafting this and forwarding it to the User Support team member in charge of data buckets. If a page already exists, it should be reviewed and updated if necessary.

A **release** 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 **US** Team contact (currently **Brandon Selbig**) must be notified of where the data is staged on Orion once it is ready for inclusion in a data bucket. The **US** Team will inform the release team when the data is staged in the bucket.

## VI. CODE FREEZE (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 **PO for the team leading the release** 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. Update *manage\_externals*

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. 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.

## C. Containers

A member of the SI team will create and test containers.

## D. 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. At this time, the technical writer will also create a **Zenodo** citation for the UFS application following [these instructions](#).

### i. Documentation Testing

The technical writer and other release team members will conduct manual testing of the release documentation, particularly the Quick Start 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  $\langle \rangle$ ). 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.

## VII. PRE-RELEASE (2 weeks 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. Release Publication Collaboration

#### i. SI Team

Two weeks prior to the release, the leading team will notify the ECE SM/PO, SI SM/PO, CM SM/PO, US team lead, and the UFS website manager of the finalized release date. The SI Team will follow the [Release Publication Checklist SOP](#) and the ECE Team [Website Content SOP](#) to provide all release announcement content to be posted. The leading team will also generate a news article incorporating information about the application, details of features/updates in the release, and a paragraph citing release contributors.

#### ii. US Team

A technical writer will prepare a GitHub Announcement for the release, which will be sent to the NOAA Program Manager (currently Maoyi Huang) for approval. A technical writer will also provide

content updates for the ECP according to the ECE Team's [Website Content SOP](#). A technical writer will provide reviews for the news article, Get Code page, social media posts, and any other updates (e.g., technical FAQs) as part of the usual content approval process leading up to the release. A technical writer will also update the Zenodo citation with the final content and release date.

A technical writer will propose a list of [EPIC Community Portal](#) updates (including updates to content and links on the application-specific "Get Code" page and the "Get Support" and "Technical FAQ" pages) via the ECP Page Update Request form by clicking on PI# Content Release - Home > "ECP Page Update Request" from the [ECE Community Engagement Content](#) page.

- SI or CM Team provides graphics
- Get Code page content updated or created
- User Support updates provided
- Technical FAQ updates provided

### iii. ECE Team

The ECE Team will follow the ECE section of the [Release Publication Checklist SOP](#) to post all of the content and to notify all appropriate parties.

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, hands-on workshops at AGU and UIFCW). 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/training-resources/>. Application-specific materials, such as documentation, are linked on the application-specific Get Code page on the ECP (e.g., <https://epic.noaa.gov/get-code/short-range-weather/> for the SRW Application).

### iv. NOAA Program Manager

The NOAA Program Manager, Maoyi Huang, will share news articles with key stakeholders (as identified by EPT) and get their concurrence. Include a "response required by [DATE]" in the communication. Please note that news articles are not required for minor continuous releases.

## VIII. PRE-RELEASE (1 week prior to release)

### i. SI/CM Team

One week prior to release, the SI/CM Team (depending on the type of release) will Create a new article page by clicking on [Community Engagement Content](#) > PI# Content Release - Home > "News Article Form" from the [ECE Community Engagement Content](#) page (where PI# refers to the

current PI, e.g., PI10 Content Release - Home). Reach out to the ECE PO (Zach Shrader) if you have any questions.

Conduct a final review of the news article, to include:

- Application leads, including stakeholders
- Technical Writer
- SI or CM Team PO (depending whether regular/continuous release)
- RTX Mgmt (Farida)
- EPIC PM (Maoyi – will reach out to NOAA and community partners to ensure that the list of contributors is complete)
- SI Team PO (Cameron) or CM Team PO (Jong)
- Include a "Ready for Publication" check box with a publication date to check off once the last person has approved.

Trim the release announcement or news article for social media posts (280 characters).

Create a new Social Media Update request by clicking on *PI# Content Release - Home* > "Social Media Update Form" from the [ECE Community Engagement Content](#) page.

Generate and select graphics to include with the news article, release announcement, and (if applicable) landing page.

## ii. US Team

A technical writer will provide a GitHub release announcement of approximately two paragraphs (a pared-down version of the news article that will link to the news article).

## iii. CM Team PO

The CM Team PO will generate a report demonstrating code reproducibility on the Tier 1 on-premises HPC resources and NOAA Cloud, if required. Ideally, this is done sooner than one week prior to the release.

## IX. Content Freeze

All content (e.g., code changes, testing, documentation, publicity/website/SM) must be complete and approved by 9am ET the day prior to the release. Once completion is confirmed w/the POs of each Agile team, notify all RTX parties of the planned release date for the following day:

- EPIX RTX RM
- EPIC RTX Product Manager

- ECE SM/PO
- SI SM/PO
- CM SM/PO
- US Team Lead

## **X. RELEASE**

### **A. ECE Team PO**

The following actions will be completed upon release:

1. Publish the updated Get Code page on the [EPIC Community Portal](#), along with any other proposed updates (e.g., to links on the Get Support page).
2. Publish the news article on the EPIC Community Portal and notify the US Team Lead that the article has been posted.
3. Publish social media (SM) announcements (IG / X).
4. All available ECE Team members will perform a QA check on changes live and visible to external visitors. At a minimum, the ECE PO will perform QA.
5. Notify EPIC Program Management that all release content has been posted.
6. If applicable, reach out to the UFS Community, NOAA, 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/desired. For contact information, please refer to the [Release Checklist SOP](#).
7. If applicable, request that the UFS website update content on their website to reflect the release (POC: Alex Alder).
8. If applicable, request that the WPO website update the “What’s Happening Now > UPDATES and EVENTS” portion of their website to reflect the release (POC: Kiana Briscoe).

### **B. US Team Technical Writer**

1. Publish the Zenodo citation and update the ECE Team so that they can proceed with content publication/publicity.
2. Update wiki on the application’s GitHub repository.
3. Post release announcement (linking to the news article) on:
  - a. The *ufs-community* GitHub Discussion page (under “Announcements”)
  - b. The specific repository’s GitHub Discussions Page (e.g., [here](#) for SRW)
  - c. Notify the ECE PO that the GitHub Announcement has been posted.

## **XI. POST-RELEASE**

### **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 have been 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.

### **B. 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 SM of the EPIC team leading the release.

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).

## **XII. EPIC RELEASE DELIVERABLES**

All release-related EPIC deliverables not already submitted to the [EPIC Deliverable Workflow](#) for approval should be submitted within one week of the release.

### **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 news article for the release 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](#) on the day of the release for approval by the end of the week.

### 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](#) on the day of the release for approval by the end of the week.

## XIII. Capability Announcements

Capability announcements loosely follow the same procedures common to feature-based and continuous releases. However, there are fewer formal requirements. Most tasks typically performed by the SI or CM team SM or PO are instead performed by whichever team member is leading/coordinating the development work. Regular release meetings, a GitHub project, and a formal release plan are not required. Because the capability announcement simply highlights new capabilities in the development branch, there is no need to cut a release branch, tag a release, or create a Zenodo citation. In general, teams have broad discretion to omit steps that are not relevant or to add steps if doing so is necessary or prudent. The [SOP Release Checklist](#) (Confluence access required) details the relevant steps for most capability announcements.

## XIV. APPENDIX

### Release Process Checklist

See the [Release Process Checklist](#) on Confluence.