Title	COTS Product Policy, Process, and Operations
Policy ID	MAK-COTS-GUIDE-6
Revision Number	1.0
Effective Date	01/20/2025
Issuing Department	COTS
Link to Guidance Documentation	
Policy Owner	Jim Kogler, VP Products
Policy Owner Signature	James Kogler (Jan 21, 2025 13:00 EST)
Approved By	William Cole, CEO
Approved By Signature	William E. Cole (Jan 21, 2025 13:29 EST)

Revision Record:

Revision Number	Date Revised	Pages Affected	Change Note / Remarks
draft	11/15/2024–1/17/2025	All	Writing of the document
1.0	1/20/2025	All	Finalized the version

Contents

1. Purpose

This document serves as a guiding framework for the development and maintenance of MAK Technologies' commercial off-the-shelf (COTS) products, emphasizing the following objectives:

1. **Clarity and Alignment**: Establish a shared understanding among all stakeholders in the software development lifecycle.
2. **Customer Satisfaction**: Ensure a positive experience through collaborative engagement and reliable MAK COTS products.
3. **Cybersecurity Best Practices**: Integrate robust security measures into the product development process.

Guiding Principles:
MAK follows an Agile approach to software development, rooted in the Agile Manifesto, prioritizing:

- **People and Collaboration**: Valuing individuals and interactions above rigid processes and tools.
- **Delivering Value**: Preferring functional software over exhaustive design documentation.
- **Customer Engagement**: Emphasizing partnership and collaboration over strict contract terms.
- **Adaptability**: Responding swiftly to change rather than rigidly adhering to a fixed plan.

This approach ensures a dynamic, customer-focused, and secure development environment.

2. Scope

This document describes the methodology of MAK's COTS product development and support processes. This includes the process by which existing COTS products are expanded and maintained as well as all related support processes. The procedures referenced in this document are detailed in Confluence where they may be refined as better procedures are devised. The process described in this document should be followed and will be updated from time to time.

This document does not encapsulate the process for researching and introducing new products to the market.

3. Team Composition and Roles

These are the teams and roles in the MAK COTS Product Development team. While this table enumerates the formal roles in the COTS team, contributions about product direction come from many stakeholders at MAK including: the VP Sales, the CTO, the CEO, and the Marketing

team. While these people don't report to the VP Products, their input is critical to the success of MAK and therefore they are consulted throughout this process.

Role	Description
Product Manager (PM)	The PM is responsible for all product development prioritization; this role is filled by VP Products. They are responsible for ranking the roadmap, overseeing the development process, and signing off on releases when they are ready. The PM works with Sales and Marketing to coordinate pricing and marketing decisions. The PM represents the "Owner/Customer" role during the development phase. The PM may designate responsibility for some of these responsibilities to others on certain projects. The PM is responsible for ensuring that cybersecurity best practices are maintained throughout the product lifecycle.
Product Team Leads (TL)	Each product team has a TL who is responsible for product architecture and new feature execution. Regarding the product roadmap, the TL reports to the PM overseeing the product. Where the PM decides *what* to do, the TL decides *how* to do it. The TL is responsible for (in consultation with the PM) the order in which features are completed within a release. The TL is largely responsible for (also in consultation with the PM) maintenance releases, as well as deciding what bugs are fixed. The TL (or a designated engineer) holds regular stand-up and bug review meetings to keep releases on track. Team leads are responsible for maintaining cybersecurity best practices throughout the product lifecycle.
Software Development Engineers (engineer)	Engineers (also known as developers) are responsible for feature development and bug fixing. In addition, they provide product support to customers. By rotating engineers through these different roles, MAK ensures a well-rounded team of engineers capable of meeting customer needs. Engineers are encouraged to participate in industry groups, such as SISO.
Software Quality Assurance Personnel (SQA)	Software Quality Assurance personnel (SQA) are assigned to each product. SQA works with the product team and reports to the TL for that product. The goal is for SQA to be an embedded part of the development team and be intimately involved in the entire planning/development/support lifecycle of the product.
Art Director and Content Manager	The Art Director and Content Manager are responsible for the terrain and models that are used by COTS products.

Role	Description
Chief Architect	The Chief Architect is responsible for overall COTS product architecture. They work with the PM, TL, and engineers to address architectural issues, particularly those impacting multiple products. They ensure that the COTS products work efficiently in concert.
Documentation Lead (technical writer)	The technical writer is responsible for COTS product user documentation. They may assist with developer documentation, and they also work on other COTS documentation as assigned by their manager.

4. Release Paths

The goal of the COTS Product team is ultimately to produce new product releases. This section enumerates that process.

4.1 Formal Releases

Existing products are advanced following one of four paths, each with an associated release type.

Path / Release Type	Description
Major	Major releases are associated with major architectural changes and significant changes in product direction. A Major release defines a new "generation" of the product. Marketing considerations may factor into the decision of whether to call something a Major release. Major releases are developed following the same methodology as a Feature release.
Feature	Feature releases represent our typical releases that include significant new features as well as bug fixes. All functionality is subject to change in a Feature release.

Path / Release Type	Description
Maintenance	Maintenance releases provide bug fixes or no-impact changes. No-impact changes may include performance improvements, the addition of configuration parameters, improved documentation, and so on. Maintenance releases should not break source compatibility (but typically do not preserve binary compatibility). Maintenance releases should not change versions of prerequisite libraries, such as VR-Link and Qt, unless those prerequisites maintain source compatibility. Third-party libraries may be upgraded to address security issues or significant defects. Occasionally, and at the discretion of the PM, new features can be added to Maintenance releases. New functionality must be additive, such that existing customers are minimally impacted. Additionally, existing functionality can be changed if the existing functionality is deemed significantly broken or problematic. Such changes should be balanced carefully against the impact on customers by the PM and TLs.
Patch	Patch releases are special builds of our products that are released to one or more customers as part of the support process. A Patch release is typically built on the maintenance branch for that release and follows the rules for a maintenance release (such as there are no new features). A Patch release is sometimes referred to as a Letter release. The development methodology for a Patch release is the same as a Maintenance release except: • The technical writer has not updated the release notes. A Change Log is produced as described in section 6.3, The Patch Release Process (page 16). • Formal testing has not been conducted. The fixed defects have been verified, but no additional testing procedures are followed. • Patch releases may not be available on all platforms, whereas maintenance releases are always supported on all platforms. • They are intended as a stop-gap measure to help a customer who can't wait for the next release but needs more than only one file or new library. A Patch release can be used as a formal maintenance release in the specific situation where the changes were to the installer only or to licensing used in the installer; In these cases no other changes can be made to the libraries or header files which would break binary compatibility.

The methodology for Major and Feature releases is covered in section 5, Major and Feature Releases (page 10). The methodologies for Maintenance and Patch releases are covered in section 6, Maintenance and Patch (page 14). For information on the naming and numbering of releases, see section 9, Release Packaging (page 28).

4.2 Informal Releases

Informal releases include IPBs (internal product builds), pre-releases, and contract deliverables.

4.2.1 Internal Product Builds

An internal product build (IPB) is an informal way for MAK to distribute COTS products for review by target customers.

An IPB is …	Which means …
A periodic build of the COTS product suite in a compatible way	It is at minimum packages of VR-Forces, VR-Vantage, and VR-Engage that all work together. If one product, such as VR-Link, didn't change, then there is no update, and the previous version is used.
Labeled IPB#, where # increments	The number never resets. It always increments.
Fairly stable	New product features have been reviewed by the team and have been found to generally work. There has been a day or two of testing by QA with major issues resolved. It should be expected that there are many areas which have not been fully tested.
Not supported or maintained	Customers are welcome to report problems or send questions to product support, but this simply informs MAK of what should be addressed in a future release. There are no patches or extensive support otherwise.
Moderately documented	Documentation is as up to date as possible. The technical writer may not have rebuilt every document, and there are working release notes describing changes that are included in the IPB. User and developer documentation updates are in progress.
Periodic in nature	It occurs as needed, whether it is every other month, biannually, etc. Typically, once a date is announced, MAK tries to maintain that release date, allowing partners and key customers to coordinate.
Available with limited platform support	It is usually released only for Windows. Other platforms may be provided on a case-by-case basis.

An IPB is …	Which means …
Provided with download links	The delivery mechanism is the same as a product release, being posted on a host server and links shared with partners and key customers.
Preserved only for several months after the subsequent IPB release	Old IPB releases are not maintained and will be removed from the host server when no longer needed.

4.2.2 Pre-Releases

A pre-release is a top-of-tree software package build to key customers to evaluate before the formal release becomes generally available. Unlike an IPB, the pre-release is likely not all COTS products in the MAK ONE suite. Pre-Releases are minimally tested and may not include updated documentation.

4.2.3 Contract Deliverables

These are deliverables that are not part of the formal release process. The details of the delivery are defined by the contract.

5. Major and Feature Releases

MAK's development methodology and processes are used to produce (from conception through release) a Feature or Major release (collectively referred to as "Feature releases" here). Feature releases include new capabilities previously unsupported by the product. Feature releases are driven by feature requests which are maintained in a backlog.

All feature development shall adhere to the MAK Coding Standard (MAK-COTS-GUIDE-1).

A release is …	Which means …
Extensively tested	Specific release testing happens on the branch where no features are added.
Released to the public when it is ready	There is no fixed schedule for the release. A final call is made by the TL and the PM regarding the official announcement and placement on the server for distribution.
Announced publicly	The PM coordinates with the Marketing team to roll the release out to the public.

5.1 The Feature Roadmap and Backlog

All new work on a product occurs when a feature from the backlog is scheduled for development by adding it to a named release version in Jira (setting its Fix Version).

MAK maintains a feature backlog according to our tracking policy (see section 7, Feature Development and Tracking, on page 16), which comes from customer requests or perceived customer needs.

Customer …	Are sourced from …
Requests	• Customer visits • Customer requests through product support • Customer requests through sales
Perceived Needs	• Competitive analysis • Industry news and analysis • User testing (internal and external) through example use cases.

The backlog is maintained in Atlassian's Jira tracking system. All feature requests are added to Jira. Feature requests without an assigned Fix Version are considered to be in the backlog. Those that are assigned a Fix Version are on the roadmap for the indicated release. The features that are in either the roadmap or the backlog can change at any time, based on changing customer demand. Specifically, a new item could be added to the development schedule on short notice due to a high customer priority, replacing an item with a low customer priority that then returns to the backlog.

The PM makes all priority decisions regarding features, with advice and consent from the stakeholders mentioned in section 3, Team Composition and Roles (page 5).

5.2 Development and Release Process

The development and release process for Feature releases occurs in four phases: planning, development, final test, and deployment.

5.2.1 Planning

Before any work on a release begins, a good deal of planning goes into the release. Some of this planning may be accomplished before the release is even scheduled, such as the development and prioritization of the backlog. (This is not to be confused with formal planning and documentation, as is done in a waterfall process.)

A target calendar date is maintained throughout the release. This date is subject to change and initially may be fairly loose, like "Summer 2023." As the release processes the date will firm up based on progress and market needs.

5.2.2 Feature Development

During the development phase, the engineers may work separately on several features, or all may collaborate on aspects of the same feature concurrently. The decision to allocate engineers to features is made solely at the discretion of the TL. If multiple engineers are assigned to a single feature, the TL shall appoint a lead engineer for the feature.

Potential features are initially discussed at a high level, then down-selected based on a number of factors. These factors include: the importance of the feature to existing and potential customers, the availability of engineers to carry out the work, time constraints, riskiness of the feature, and priority of the feature compared to other potential features and defects. If there are cross-product aspects to the feature that require work from multiple teams, cross-product team meetings are set up with the relevant engineers.

For each feature that is developed, the process is:

Development Step	Description
1. Discussion	Product team members meet with the PM and TL to discuss the feature.
2. Scope	Non-trivial features are developed in a separate development branch (feature branch). The TL, SQA, and the PM work with the lead engineer on the feature to define the scope, UI, and other aspects of the feature. For details, see section 7.1.1, Feature Design (page 18).
3. Incremental Development	The development is incremental, with SQA trying and testing the feature at the same time the PM and the technical writer are also trying it. At this point, issues found in the feature branch are raised using a method agreed on by the team. Issues that are tracked and resolved using that method, not entered into Jira. For more information, see section 7.1.2, Feature Development (page 1820).
4. Code Review	Once the engineer declares the feature complete, the engineer shall submit the code to the TL for code review. The TL may review it personally or assign it to another team member or otherwise appointed subgroup. For details, see section 7.1.37.1.3, Code Review (page 2020).
5. Feature Review	Once code review is complete, the PM may review the usability of the feature and approve the branch for merge back into the trunk.
6. Feature Merge	The feature branch is merged back into the main trunk. For details, see section 7.1.47.1.4, Feature Merge (page 2121).
7. QA Testing	SQA performs smoke tests after the merge. For more information, see section 12.3, QA Testing (page 3838).

5.2.3 Final Test Phase

When the complete feature set is developed and merged into the trunk, pending no other high priority customer requirements, the PM and the TL shall move the team into the final test phase. It is called “Final Test Phase” because—while testing occurs throughout the development process—comprehensive tests are performed before a final release.

Testing Step	Description
1. After Final Feature Merge	SQA retests newly merged features in the main development branch (or trunk). During this time, the TL and SQA form the test group and define areas for regression testing of the first Test Candidate. Written tests for features are stored in SpiraTest or an Excel spreadsheet and maintained by SQA.
2. Test Candidate	The TL, at their discretion, calls for a code freeze and creates the first Test Candidate (TC). There may be one or more TCs, depending on bugs found during regression testing. Regression tests are stored in SpiraTest or an Excel spreadhseet and maintained by SQA. A test candidate: • Is expected to be incomplete (for example, the docs may not be finished). • Will have a complete set of tests run on it by the whole test team. • May take days to weeks to fully test.
3. Release Candidate	Once SQA and the TL are satisfied that most of the major errors have been fixed in the TC, a Release Candidate (RC) is created and sent back to SQA for another regression testing pass. Again, there may be one or more RCs depending on what is uncovered through the testing process. A release candidate is: • Created after all tasks, including documentation, are finished. • Complete enough to be released to customers. A release candidate has the potential to become the release. It is considered complete and is indistinguishable from an actual release. • Tested for final packaging and documentation problems. • Scanned for viruses using appropriate scanning software.

Naming conventions for candidate packages are described in section 9.2, Package Names (page 30).

5.2.4 Feature Release Phase

When SQA and TL agree that a Release Candidate is ready for release, PM is notified of this status along with any caveats pertaining to this release. The PM and TL then start the release process:

Release Step	Description
1. Release Checklist	To obtain the approval of stakeholders, the product team works through the release checklist, which is submitted to stakeholders. All feedback is collected. The PM reviews the feedback and approves the release or suggests changes which then must be addressed. See the MAK Product Release Checklist appendix on page 51.
2. Further Fixing and Testing (Optional)	If the release is not approved, the process reverts back to the Final Test phase for further iteration.
3. Release Approval	Once the Feature release is approved, it is packaged for delivery, placed on Azure for download from the MAK ONE Downloads Page on the website, and a release announcement is made.

5.3 Release Retrospective

Release retrospectives are conducted to evaluate the development and release process, focusing on what worked well and identifying areas for improvement.

These retrospectives are scheduled if there is a Yes response marked on the "Does this release require a retrospective?" question in the Engineering section of the checklist. This decision is typically based on the complexity and risks of the release. They are strongly recommended for Major and Feature releases of VR-Vantage, VR-Forces, and VR-Engage, but may be conducted for other COTS products as needed.

During these sessions, the team reviews successes as well as challenges or inefficiencies encountered. The findings are recorded and applied to enhance the processes for future release cycles, fostering continuous improvement.

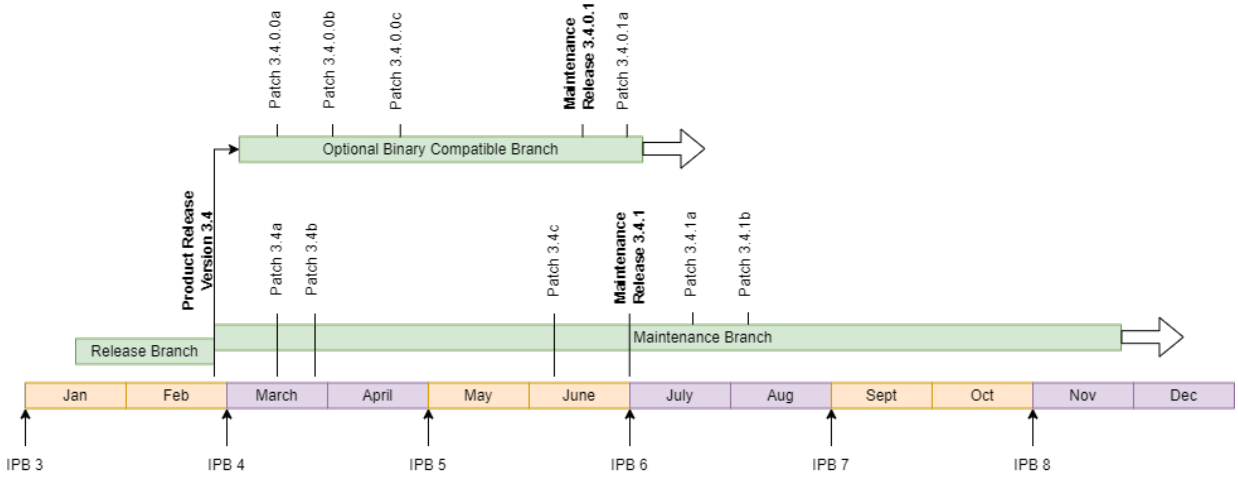
6. Maintenance and Patch Releases

MAK supports released software within the guidelines of our maintenance policy. See the MAK COTS Products Maintenance and Support Policy, found in SharePoint (Company/Documents/General/Document Templates/Maint & Support). To support this effort, MAK will occasionally deliver Maintenance and Patch releases of our software.

The decision to issue a Maintenance release is made by either the PM or TL based on careful consideration of a number of factors, including but not limited to:

- Severity of defects identified
- Quantity of defects identified and fixed since the previous Major or Feature release
- Time since the previous Feature or Maintenance release.
- Significance of a particular defect and impact on customers, as well as customer perception

The maintenance branch maintains a stable development branch for Maintenance and Patches release updates. It is not typically non-binary compatible, but changes must have only small impact on our customers. (For binary compatibility, see the Binary Compatible Builds appendix on page 55.)



6.1 The Defect Roadmap and Backlog

MAK maintains both a roadmap and a backlog of defects that consist of customer-reported defects, as well as defects found in the testing process but deferred (not fixed) in the Feature release process. This list of defects is maintained in Jira per our defect tracking policies (see section 8, Defect Tracking Policies, on page 24).

The defects roadmap are those defects that are scheduled to be fixed in a particular release. The backlog consists of defects that are known but not yet scheduled to be addressed in a release.

6.2 The Maintenance Release Process

6.2.1 Defect Fixing Methodology

Defects are fixed as they are identified, and Patches are released to customers based on customer requirements. For example, if the customer is happy with a workaround, no patch is released. If the customer cannot resolve the problem without a patch, however, then one is made. Individual engineers are empowered, as part of their product support responsibilities, to choose whether to produce a Patch release for a specific customer. This process does not necessarily occur only after a decision is made to deliver a Maintenance release. It is an ongoing process conducted daily by the development team as part of their normal support duties.

A Patch release is basically an intermediate maintenance release without the Final Test Phase. For details on delivering Patch releases, see section 6.3, The Patch Release Process (page 16). See section 14, Support (page 41) for more on product support.

6.2.2 Final Test Phase

On completion of bug fixing, SQA will be involved with basic testing to:

1. Verify the defects that were fixed.
2. Perform basic regression testing on the release and verify that significant other functionality hasn't broken.
3. Most importantly, SQA is tasked with ensuring the Maintenance release is of higher quality than the previous release.
4. Perform virus scans on all packages before they are released to customers.
5. Ensure all package naming and versioning rules are followed.

6.2.3 Maintenance Release Phase

When SQA and TL agree that a Release Candidate is ready for a Maintenance release, the PM is notified of this status along with any caveats pertaining to this release. The following steps are preformed:

Release Step	Description
1. Release Checklist	To achieve approval of the stakeholders, the product team works through the release checklist, which is submitted to the stakeholders for approval. All feedback is collected. The PM decides to release or not based on the feedback received. See the MAK Product Release Checklist appendix on page 51.
2. Further Fixing and Testing (Optional)	If the release is not approved, the process reverts back to the Final Test phase for another iteration. This happens after an agreement is made due to overcoming the objection(s) to release approval.
3. Release Approval	Once the Feature release is approved, it is packaged for delivery, placed on Azure for download from the MAK ONE Downloads Page on the website, and a release announcement is made if required.

6.3 The Patch Release Process

This is the process for creating a non-binary compatible Patch release for a customer. For information on binary compatible releases, see the Binary Compatible Builds appendix (page 55). Patch releases are built and delivered at the TLs discretion. The release checklist is not required.

6.3.1 Defect Fixing Methodology

Step	Description
1. Make fixes available on branch in SVN	Make the fixes on the appropriate branch(s) in SVN.
2. Make fixes available in trunk	Check the same fixes into the trunk if appropriate and not already there.

6.3.2 Prepare the Patch Build

Step	Description
1. Check version information in version.h (VR-Forces only)	Most patches are non-binary compatible, therefore that is the most typical procedure: • Check that the PLUGIN_COMPATIBLE_VERSION is set to "${VRF_VERSION_NUMBER}" so that it is automatically updated with every version. (As this is the most common option, it should already be set correctly.) If you are creating a binary-compatible patch, see the Binary Compatible Builds appendix (page 55).
2. Determine the version number	Determine what the version name and letter should be. See section 9.1.2, Letters, on page 28.
3. Check version information on NIGHTBUS	Make sure the version name/letter is correct on the NIGHTBUS page and the corresponding Doxygen build is set up before kicking off the build.
4. Set fix version in Jira	Ask the TL to add the new Fix Version in Jira. Once they create it, add that new Fix Version to all fixed bugs included in the patch.
5. Generate a build	Before starting any builds, ensure the version number is correct in the build configuration on NIGHTBUS . You only need to build the Patch release for the platforms that the customer needs. Each letter release will likely only have one or two platforms built.

6.3.3 Deliver the Patch

Step	Description
1. Upload the build to Azure	Upload finished build to Azure using the Cloud Upload button on NIGHTBUS.

Step	Description
2. Move the build to the appropriate folder on Azure	After the upload completes, move the folder to the ./vrforces/releasesAndPatches/<*version*>+ folder with the appropriate name (for example, ./vrforces/releasesAndPatches/5.1.1+). For detailed instructions, see Confluence. Patch releases are stored only on the Azure blob. Keep PDBs on the Azure blob, as well, in the same folder as the Patch release.
3. Create the Patch Release page on Confluence	Use the built-in feature to create a page for the Patch release in the product space under the Patch Releases page. For detailed instructions, see Confluence.
4. Create the change logs and put them in the Azure folder	Use the Release Notes button again to create the change log, which you will append to the branch's Word document and then save it as a PDF. For detailed instructions, see Confluence.

6.3.4 Set up the Branch to Build the Next Patch In Case One Is Needed

Step	Description
1. Create an SVN tag	Create an SVN tag for the Patch release.
2. Increment the version letter	Increment the version letter on NIGHTBUS to the next letter so that it will be ready for the next release.
3. Request release status change in Jira	Ask the TL to change the Patch release's status to Released.

7. Feature Development and Tracking

MAK uses Jira as its official feature tracking system. All feature requests, whether generated internally or externally, are recorded in Jira as a Feature type of ticket. Any MAK employee may create a feature ticket.

7.1 Feature Development

7.1.1 Feature Design

The scope of the feature is defined by the PM and TL, and then refined by the engineer assigned to work on the feature.

- Smaller, easy-to-describe features that can be defined in a couple of paragraphs, are put directly into Jira.
- Larger, more complex features have a Jira ticket that is linked to a design document. The TL determines whether this is done in a designated folder on SharePoint or the product's Confluence space. All design information for a feature must be referenced from the feature's Jira ticket.

Depending on the complexity of the feature, the engineer may solicit design reviews from other engineers and the TL, and possibly the PM.

Design Document Structure

The engineer should include the sections that are relevant to the feature they are designing.

Section	Required?	Description
Overview	✓	Brief overview of how the feature is intended to work.
Market Problem		Description of what problem this feature will solve for users.
Cybersecurity Consideration	✓	New features require thought of any cybersecurity concerns. Any concerns if know should be listed as well as potential mitigation. If there are no known concerns this should be noted.
Functional Specification		Short overview of the functions that the feature will perform.
Design Specification	✓	Details on how the engineer will implement the feature. It should be clear enough that another engineer can read this section and learn enough to implement the feature. The engineer determines the format necessary to communicate the design. It can involve diagrams, broad descriptions, detailed notes, and so on.
Test Case Specification		Recommended tests for the feature. Test cases for new features should ultimately be written in SpiraTest, implemented as automated tests run by the nightly build system (NIGHTBUS), or both. During the feature design, however, it can be useful to capture initial thoughts on what cases should be tested to ensure that the feature works correctly in the design document.
Jira Link	✓	Link to the Jira ticket(s) for this feature. If the design involves more than one Jira ticket, the best practice is to use an Epic in Jira to link all of the related features. You can then link to the Epic from this section. (It is also recommended to link from the Feature or Epic ticket to the design document so other people can read it.)
Documentation		Brief information on what updates must be made to the user and/or developer documentation.

Maintenance of Design Documents

Design documents are maintained only during the initial development of a feature. When the engineer commences work on the feature, the design may evolve and therefore the engineer may update the design document.

The design document is not documentation for the feature. It captures the original design and the rationale behind that design. In the future, the feature may change to meet new requirements or overcome unexpected challenges. There is no expectation that the design document will be updated; if necessary, the engineer assigned to implement the changes creates a new design document.

7.1.2 Feature Development

Features are typically developed in a feature branch on SVN (unless they are very minor). All work on the feature shall be done and checked into this branch until the feature is considered complete and ready to merge into the main development branch for the release. At the time it is created, the base for the feature branch should be the top of that development branch. Create the branch in the working directory for the product on SVN, for example, https://repo.mak.local/svn/<*product*>/working.

The naming convention is <*engineerusername*>-<*featureInBrief*>, for example, gwillikers-codeRefactor. If multiple engineers are expected to work in the branch, use the username for engineer considered the "lead" on the feature. That is, the person who ultimately can say when the feature is complete, at which point the feature branch can be archived and then deleted.

A build configuration for the given feature should be created on NIGHTBUS. It can be set up to build nightly or on demand. This enables testing by SQA and other engineers and also demos for stakeholders while the feature is in development.

The engineer should routinely merge the development branch into their working branch to minimize the risk of conflicts that could arise when they finally merge the working branch into the main branch.

For an overview of the workflow, see section 7, Feature Development and Tracking (page 16). To learn more about checking changes into the working branch, see section 10.3, Source Control (page 32).

7.1.3 Code Review

Code reviews are highly encouraged but not required. The code review may happen before the feature is merged into trunk. For complicated features the TL may request that multiple code reviews are conducted through the development process. Code reviewers are appointed by the TL.

The format of the review is at the discretion of the TL. For example, it may take the form of an interactive online meeting with screen sharing, or it may be done by posting the code for review in a tool such as Crucible.

7.1.4 Feature Merge

Once development of the feature is complete, the engineer coordinates with the TL and other engineers on the product team to determine when to merge the working branch into the main development branch.

Each feature branch should be merged separately, with enough time between merges to smoke test the main development branch after the merge. To reach this state, the feature must be functionally (or largely) complete and perform sufficiently well that it does not impede the progress of testing of the development branch as a whole. As a rule, the main development branch should always be in a buildable state.

After the merge, whenever possible, engineers and SQA should create automated tests and review the results to ensure the feature functions as designed. A test case should also be added to SpiraTest or the Excel spreadsheet. They shall test the feature after the merge and as part of final release testing. See section 12.2, QA Process (page 37) to learn about how SQA plans testing and 12.3, QA Testing (page 38) for the testing process.

7.1.5 Updating Major Product Dependencies

The MAK ONE applications—VR-Vantage, VR-Forces, and VR-Engage—are released as a collection of compatible builds, therefore their development cycles are synchronized for Major and Feature releases.

In the main development branch, the TL or an engineer designated by the TL upgrades the branch regularly (approximately once a week) to the latest build of the COTS product(s) it is built on. As each product advances through its own development cycle, for example, VR-Vantage goes up on VR-Link and DI-Guy, then VR-Forces goes up on VR-Vantage, and then VR-Engage goes up on VR-Forces.

7.2 Feature Tracking

7.2.1 Jira Feature Policies

All features will be tracked in Jira. The Jira Feature will contain at a minimum:

Rule	Description
Summary	A brief explanation of the feature.
Description	New features should contain a brief description of what the feature will accomplish. A use case is highly desired, as is the benefit to users.
Customers	A running list of all customers requesting the feature shall be maintained. This is done by associating the feature request with product support tickets or by adding the particulars in the feature ticket's comments (if the request did not come through support). Customer information will not appear anywhere else in the feature.

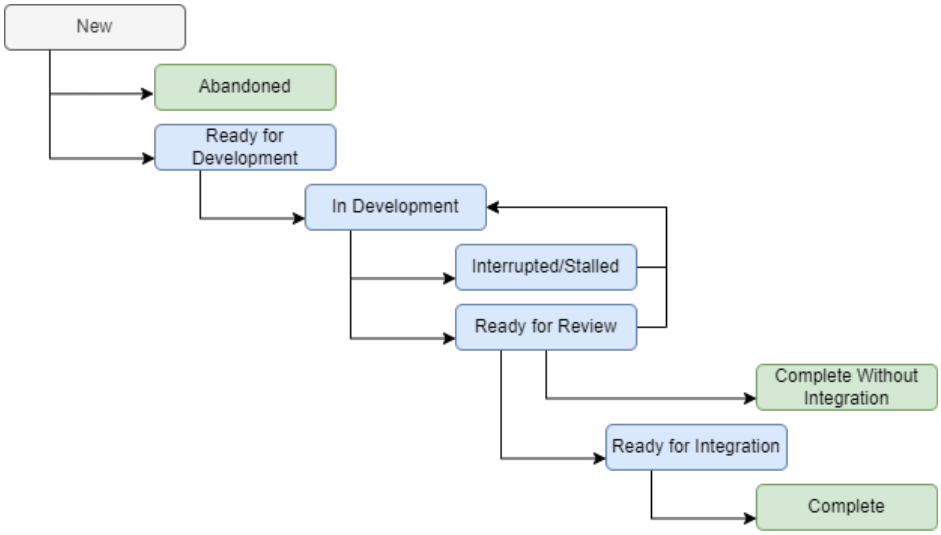
Rule	Description
Feature Tracking	Each feature is tracked in Jira by product.

7.2.2 Feature Workflow

Jira features use the following workflow:

Workflow Phase	Status	Description	SQA Responsibility
New	New	It's an idea or a rough requirement	
Abandoned	Done	This is a complete state where the feature will never be looked at again and not done.	
Ready for Development	In Progress	It's a flushed-out requirement that is imminent. Someone will start work on it soon.	SQA works with team leadership to understand what that feature goals are.
In Development	In Progress	Someone is working on this.	SQA attends meetings and works with engineers to understand progress of development. At an appropriate time, QA begins playing with the development branch. Issues are tracked outside of Jira.
Interrupted/Stalled	In Progress	Someone started working on this but stopped due to some priority. This is slipping into the *distant* future. It is probably in a branch and the branch isn't going anywhere for a long time. The feature returns to the backlog. This step is basically the same as the New phase and, if it's restarted, then archaeology will be required.	

Workflow Phase	Status	Description	SQA Responsibility
Ready for Review	In Progress	Awaiting a code review and/or testing by SQA, TL, or PM. This step doesn't *require* that someone perform a code review. The team could simply accept the feature and move it to the Ready for Integration step.	SQA provides input during the review process.
Ready for Integration	In Progress	Ready to integrate into the trunk. Can be skipped if there is no integration required because it has already gone into the trunk.	SQA updates any relevant test plans.
Complete Without Integration	Done	This feature is 100% done but it has not been integrated into a final MAK product. This could be art or a plug-in created for a customer that doesn't make it into the final product. If someone wants to integrate it into the product, they must create a new feature for it. There is no way out of Complete Without Integration.	
Feature Completed	Done	The feature is 100% done.	SQA verifies review process and starts reporting any bugs in Jira.



8. Defect Tracking

MAK uses Jira as its official defect tracking system. All bugs, whether found internally or externally, are recorded in Jira as a Defect type of ticket unless otherwise noted below. Defects may be logged by anyone at MAK.

There are different approaches for tracking defects:

Defect Type	Tracking Method
Bugs	Defects found in the software after features were merged into the development trunk, including after a release. These can be found internally or by customers. They are tracked in Jira as defects for the product.
Known Problems	Significant defects are recorded as a defect in the Jira system and therefore issued a defect number. They are also listed in the Confluence customer-facing knowledge base. Some problems which may not be considered defects will be tracked as Known Problems in Confluence but not in Jira. See section 8.1.1, Documenting Known Problems, below.)
Development Branch Bugs	Defects found during the iterative development process (before branch merge) are not placed in Jira but instead tracked using a method agreed on by the team.

As needed, the TL holds a bug list review meeting. The team reviews the tickets currently scheduled for the release they are working on as well as the backlog of Jira defect tickets. With input from the team, the TL prioritizes and assigns tickets to particular releases. Team members

can share any knowledge they have about the issues raised in the tickets and weigh in on factors such as degree of difficulty to implement the fix. SQA provides details on bugs and can offer opinions on severity.

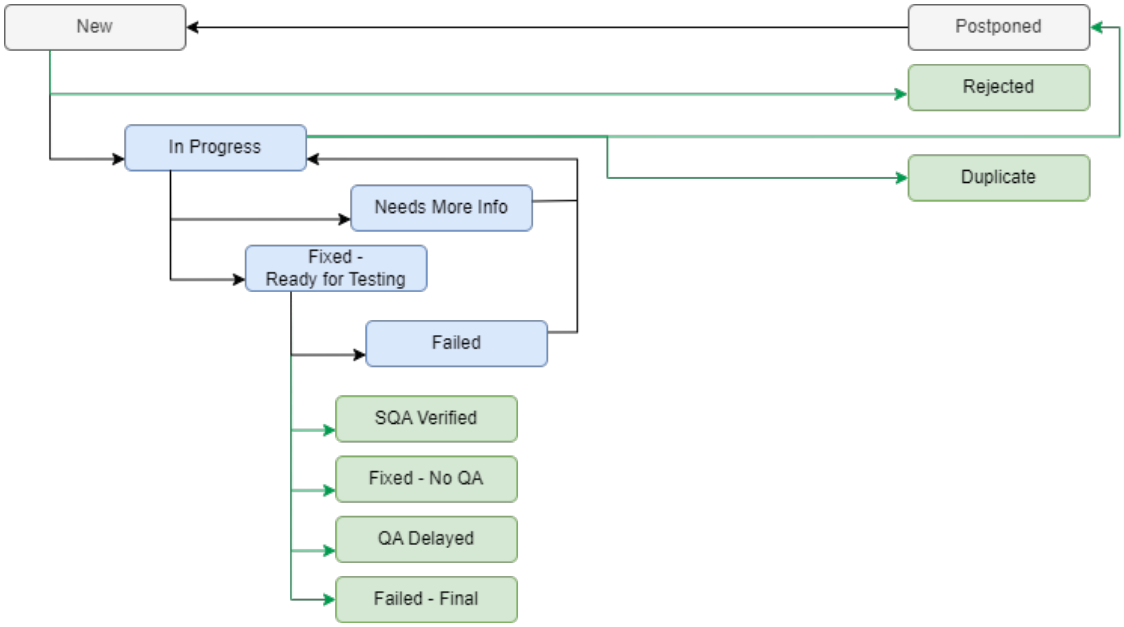
8.1 Jira Defect Tracking Data

Users should complete the follow fields for defects entered into Jira.

Parameter	Description
Product Identification	Defects are assigned to a product on creation by creating the Jira ticket within that project. Defects must always be assigned to a product.
Summary	The summary (title) of the defect should briefly describe the nature of the problem. It will be exposed to customers in release notes and therefore the wording should be clear and appropriate. **It must not include customer information**.
Description	The description of the defect provides more information about the nature of the problem.
Replication Procedure	The steps necessary to see the defect occur. Preferably, the reporter should include the expected behavior as well as the problematic behavior.
Fix Version	• Defects found during the final test phase are assigned to the release that is being tested. • Defects reported by customers are not necessarily assigned to a release. • A release associated with a defect indicates the formal release the defect is fixed in, or targeted to be fixed in, not the release the defect was found in.
Affects Version / Version Found In	This is where the reporter found the defect: • Version Found In can be a formal release, an internal product build (IPB), or a nightly build. • The Affects Version must be a formal release that has been rolled out to customers. Select from the existing list. The Affects Version will never match the Fix Version.
Fixing a Defect	When a software engineer starts to fix a release, the defect must be: • Assigned to the engineer • Assigned to the next applicable release per the TL or the PM
Customer Information	Customer contact information for all known customers waiting for this defect shall be tracked. This is done by associating the defect with product support tickets or by adding the particulars in the defect's comments (if the issue did not come through support). Customer information shall not appear anywhere else in the defect.

8.2 Defect Workflow

In the diagram, the arrows show the workflow for COTS product defects.



These are the workflow phases for COTS product defects:

When in this workflow phase …	It has the status …	And it indicates that …
New	New	This is a reported bug.
In Progress	In Progress	Someone is actively fixing this bug
Postponed	New	This bug is such a low priority that we will likely never do to it, but we don't want to abandon it in case someone needs to do some archaeology.
Duplicate	Done	We are closing this because it's addressed in another defect that is linked to this one.
Rejected	Done	This is not a bug, either because the behavior is expected or the problem cannot be reproduced, or we don't plan to ever fix it. In the latter case, someone should probably document it as a known issue in the customer-facing knowledge base, or least in the internal one for engineers to know about.
Needs More Info	In Progress	No one can fix this because there isn't enough information to know what's wrong. Someone must answer engineer's questions before the fix can proceed.
Fixed - Ready for Testing	In Progress	Engineer has fixed it.

When in this workflow phase …	It has the status …	And it indicates that …
Failed	In Progress	It was marked as fixed, but someone tested it and it was still broken.
Fixed - No QA	Done	We fixed it but have not tested it and may not test it in the future.
QA Delayed	Done	We fixed it, but don't have time for QA to verify it yet. It should be verified before a release.
SQA Verified	Done	We fixed it and someone (who didn't fix it) verified that it was fixed.
Failed - Final	Done	This bug was verified after the IPB as failing. It will never leave this state. A new bug will be created and treated like the issue was just found and fixed in the next IPB or release.

8.3 Documenting Known Problems

For defects that are known to exist in a release that customers are using, the PM or TL determines whether they should be documented in the knowledge base. If so, they task someone to add a topic to the customer-facing knowledge base. This can be a member of the product team or the technical writer. The article shall follow the established template which contains the following information:

If the issue …	Then the article shall …
Has a workaround	Provide the details of the workaround.
Will be fixed	Note that the feature will be fixed in an unspecified future release and include the relevant Jira ticket numbers without links. Once a release that includes the fix (verified by SQA) has been released, the article should be updated with that version. If there are security implications, they must be noted specifically as a security issue.
Will not be fixed	Describe the problem along with any workaround information. There shall be no implication that MAK will work on a fix.
Involves security issues in third-party libraries	Describe the issue and the library information. Indicate that it is a security issue.

9. Release Packaging

9.1 Version Numbering

9.1.1 Numbers

COTS products use three digits to denote a version:

A.*B*.*C*

where *A* is the Major release, *B* is the Feature release, and *C* is the Maintenance release. The *C* is omitted for Major and Feature releases (that is, the number is in *A*.*B* format). For more about each of these release types, see section 4, Release Paths (page 7).

Release Type	Description
Major	Major releases change the major release number (*A*), for example, 2.0 and 3.0.
Feature	Feature releases change the feature release number (*B*), for example, 2.1 and 3.3.
Maintenance	Maintenance releases change the maintenance release number (*C*). For example, 2.1.1 is the Maintenance release for Feature release 2.1, and 3.0.1 is the Maintenance release for Major release 3.0. If the Maintenance release is fully binary compatible with the previous release, the text BinaryComp and a letter are appended, for example, 4.5-BinaryComp-a.

9.1.2 Letters

Letters are appended to the version to indicate a patch built for a specific customer. They are a complete package with a full installer. For more about these types of releases, see section 4, Release Paths (page 7).

The letters are lowercase and are placed immediately after the version number with no intervening punctuation.

Example patch version number:

✓ 3.4a

Unacceptable patch version numbers:

× 3.4.a
× 3.6-a

9.1.3 Internal Product Builds

When an IPB is delivered, it is identified as IPB# where # increments.

Example IPB version number:

- ✓ IPB12

9.1.4 Test Candidates

A test candidate is something that is close to a Major, Feature, or Maintenance release candidate, but is known not to be releasable. The details of what constitutes a test candidate are covered in section 5.2.3, Final Test Phase, for Major and Feature releases and section 6.2.2, Final Test Phase, for Maintenance releases.

Test candidate numbers start at 1.

Example test candidate version number:

- ✓ 4.0-tc1

Unacceptable test candidate version numbers:

- × 4.0tc1
- × 4.0.tc1

Test candidates may have internal strings to indicate that they are test candidates. Such text must be removed before creating a release candidate.

9.1.5 Release Candidates

A release candidate is something that could be released. The details of what constitutes a release candidate are covered in section 5.2.3, Final Test Phase, for Major and Feature releases and section 6.2.2, Final Test Phase, for Maintenance releases.

Release candidate numbers start at one.

Example release candidate version number:

- ✓ 4.0-rc1

Unacceptable release candidate version numbers:

- × 4.0rc1
- × 4.0.rc1

Release Candidates can be marked as such in the packaging, but all internal versions should be final for the release. For example, the About box must read "4.0" not "4.0-rc1".

9.1.6 Top-of-Tree Snapshots

A top-of-tree build is a snapshot of the main trunk. These snapshots should be given to customers only when (1) they need a feature that we've implemented since the most recent numbered release when (2) we are not in a position to generate an official Maintenance release.

Top-of-tree snapshots should take the version of the upcoming release that we are working towards, prefixed by "TOT*YYYY-MM-DD*" where *YYYY* is the four-digit year, *MM* is the two-digit month, and *DD* is the day. Use leading zeroes for numbers that are not two digits. For example, the month of January is 01.

Example top-of-tree snapshot package name:

- ✓ vrf5.2-TOT20240816

Unacceptable top-of-tree snapshot package names:

- × vrf20240816tot5.2
- × vrf20240816tot-5.2
- × vrf5.2-TOT2024816

9.2 Package Names

9.2.1 Package Names for Released Products

When packaging a product for distribution, adhere to the following conventions:

<productNameInCamelCase><version>-<platform>-<compiler bits>-<YYYYMMDD>-<installer description>

Example product package names:

- ✓ vrLink5.9.1-linux64-rhe8-20240920.tar.gz
- ✓ vrLink5.9.1-win64-vc14-20240923.exe
- ✓ vrVantage3.1.1-linux64-rh8-20240924.tar.gz
- ✓ vrVantage3.1.1-win64-vc15-20240924.exe
- ✓ vrVantage3.1.1-win64-vc15-20240924.pdbs.exe
- ✓ vrVantage3.1.1-win64-vc15-20240924.docu.exe

Example data package names:

- ✓ makData18-data-Core-20240926.Core.MAK-targz
- ✓ makData18-data-Terrain-20240926.Terrain.1Of3.MAK-targz

These are examples of unacceptable package names:

- × vrlink-3.9.1-win32-msvc++7.1-setup-bcv0.exe
- × vrlink3.9.1-msvc++7.1-win32-setup-4dfg.exe

9.2.2 Package Names for Non-Released Products (Nightly Builds)

<productNameInCamelCase><branch>-<platform>-<os(where applicable)>-<compiler (where applicable)>-<date>.

where:

- <*branch*> = The SVN branch this was built against. It should be "TOT" for trunk.
- <*date*> = in format YYYYMMDD

9.2.3 Product Names

Do not use uppercase for abbreviations. Standard COTS product names include:

- makLogger
- makRti
- diguy
- vrlink
- vrengage
- vrvantage
- vrforces
- vrexchange
- sensorfx
- radarfx

9.2.4 Component Names

These are OS annotations added to packaging to indicate platform information.

Platform	Compiler
win32	msvc++6.0 - compatible with 32-bit libraries/binaries, runs on win32 OS and win64 OS
	msvc++7.1
	msvc++8.0
win64	msvc++8.0 - compatible with 64-bit libraries/binaries, runs on win64 OS only
	msvc++12.0
	msvc++14.0
	msvc++15.0
	msvc++17.0
win32-win64 or win	msvc++9.0 - contains both win32 and win64 configurations (see above)
	msvc++10.0
	msvc++11.0
redhatlinux8.0	gcc3.2
redhatlinux9.0	gcc3.2.2

redhatentws3	gcc3.2.3
redhatentws4	gcc3.4.4
redhatentws5	gcc4.1.1
redhatentws6	gcc 4.4
redhatentws7	gcc 4.8
redhatentws8	gcc 8.3
redhatentws9	gcc 11.1
fedoracore3	gcc3.4.4
irix6	n32
solaris8	suncxx5

10. Source Control and Routine Builds

MAK Products are maintained in a source control system and built regularly using a nightly build and test process.

10.1 Source Control

MAK uses Subversion (SVN) as a version and source control system for proprietary source code. While several historical systems are maintained for historical reasons, nothing should be added to these repositories, and no other repository system shall be created. The SVN repository is organized in repositories by COTS products and projects, where each individual code base is stored in a separate and distinct directory.

In each individual repository, there are four subdirectories, each with different purposes.

Subdirectory	Description
trunk	The *trunk* directory holds the mainline development repository of the product. Major and Feature releases are typically created from the trunk of the product. Other branches, such as maintenance branches or feature development branches, are branched from the top of the relevant development branch and later merged into the trunk or branch to incorporate bug fixes and new features.
branches	When a Major or Feature release is created from the trunk, a branch is made and saved into the branches directory for future Maintenance and Patch releases. Each branch in the *branches* directory should be named to reflect the release version.
working	The *working* directory is for feature development or bug fixing branches that would be interruptive to develop directly in the trunk. Working branches are temporary and can be discarded after the changes have been merged into the trunk or branch.

Subdirectory	Description
tags	Tags are snapshots of either the trunk or a maintenance branch at the time of a release. Each formal release (Major, Feature, Maintenance, or Patch) should be tagged in SVN. Tags allow engineers to quickly access the code exactly as it was for a release. Tags are not branches; once a tag has been made, it should not be changed.

The TL decides when to create branches based on the product's current development. Major and Feature releases typically originate from the trunk, but release branches may be created before a product release to allow ongoing development for future releases in the trunk.

Some projects, such as internal libraries, demos, and one-off efforts, may not have tags or branches for every release but should always maintain a trunk.

SVN commit messages must be concise. Commits tied to specific defects or features must include the related Jira defect or feature number for easier tracking. For syntax fixes, build corrections, code cleanup, or other unrelated changes, commit messages should briefly explain the purpose.

Commit descriptions should be clear and informative for stakeholders like engineers and SQA. A well-written SVN log is crucial for supporting the product post-release, as engineers often rely on these descriptions when working on features or troubleshooting issues.

10.2 The Nightly Build System (NIGHTBUS)

NIGHTBUS is MAK's internal nightly build system. It is a framework for automatic building, packaging, and testing of COTS products or other third-party libraries used by MAK Software. The NIGHTBUS interface is a web page that allows users to create, execute, suspend, archive, and delete build configuration. It is also a place where MAK employees can access and download builds.

The NIGHTBUS interface is separated by project, where different projects align with different repositories in the MAK SVN code base. Each project page can contain multiple build configurations organized by a "tag", which typically matches the SVN branch name of the project.

Each project page should have a fixed "tot" tag, which stands for "top of tree" or "top of trunk." The build configurations under the "tot" tag compile that project's "trunk" repository across the various supported platforms.

Other tags and build configurations should be created for maintenance branches. These build configurations are used to produce Maintenance and Patch releases. Build configurations for currently supported releases should remain active as long as the product is supported. Once a

given release is no longer supported, the accompanying build configurations should be suspended and archived.

Create a descriptive tag group and build configuration for feature development branches, especially when the feature branch introduces new files, new dependencies, or changes packaging, as described in section 7.1.2, Feature Development (page 20). This allows engineers to resolve any compile, linker, or packaging errors across the various supported platforms before merging the changes into the main development branch or trunk. Build configurations for working branches should be suspended, archived, and deleted once the changes have been merged and verified in the development branch or trunk.

10.3 Product Builds on Azure

Throughout the development cycle, nightly builds of the development branch are posted to Azure. These include development snapshots that SQA uses to test features and bug fixes, as well as test and release candidates.

All product builds of significance should be uploaded via the NIGHTBUS Upload interface to Azure for storage. The uploaded build is stored according to the following rules on the productinstallers Azure Blob:

Type of build	**Location in the Azure productinstallers blob**	**NIGHTBUS Upload?**
Development snapshots of the trunk	/<*product*>/<*product*>.tot	✓
Development snapshots of working branches	/<*product*>/Branches	✓
Product releases and patches for customers	/<*product*>/releasesAndPatches	

The NIGHTBUS script for uploading builds automatically creates a folder in either the Branches or <*product*>.tot directory, depending on whether the build is based on a branch or trunk in SVN. The name of the folder for the build follows the convention:

YYYY.MM.DD SVN *######*

where:

- *YYYY.MM.DD* is the date when the build was created
- *######* is the “Last Changed” SVN revision number in the build that was uploaded.

Example paths on Azure:

- Branch build:
  vrEngage/Branches/gwillikers-codeRefactor /2024.11.17 SVN 234567/
- Trunk build:
  vrEngage/vrEngage.tot/2024.11.17 SVN 234567/

Working branch builds and trunk builds are posted as needed by the TL, SQA, and engineers throughout the development cycle.

11. Security Assurance

COTS products are specifically designed to operate on government systems where cybersecurity is a top priority. To minimize cyber risks to MAK and its customers, the following policies have been adopted. These policies align with industry best practices and draw from multiple authoritative sources.

Authority	Source
Cybersecurity & Infrastructure Security Agency (CISA)	• Known Exploited Vulnerabilities Catalog https://www.cisa.gov/known-exploited-vulnerabilities-catalog • Product Security Bad Practices | CISA https://www.cisa.gov/resources-tools/resources/product-security-bad-practices
UK Government	Secure by Design Principles - UK Government Security – Beta https://www.security.gov.uk/policy-and-guidance/secure-by-design/principles

11.1 Third Party/Open-Source Library Management

11.1.1 Maintaining a Comprehensive List

MAK will maintain a comprehensive list of all third-party libraries called the Third-Party/Open-Source List (3POSL). Information maintained will include library names, versions, and intellectual property rights statements. This list will be reviewed every release to ensure that all version updates are recorded.

11.1.2 Availability for Customers

The 3PSOL shall be published in the MAK Product Support Knowledge Base and accessible to all customers for inspection. The list can be found on the Product Support Knowledge Base: Third-Party Open Source List

11.2 Vulnerability Disclosure Policy

Knowledge base articles disclosing known software vulnerabilities will be reviewed and updated quarterly or as soon as possible upon change of status. If any exploits are discovered or

reported in third-party versions used by COTS products, details of the exploits will be published in the MAK Product Support Knowledge Base. All vulnerabilities listed in the Knowledge Base will be clearly identified as known software vulnerabilities. The list can be found on the Product Support Knowledge Base: Known Vulnerabilities.

When a vulnerability is discovered, an email will be sent to a cyber vulnerability email list maintained by the MAK Marketing team. Customers can register for the Known Vulnerability mailing list on the MAK website.

Before every release, the Known Exploited Vulnerabilities Catalog will be consulted to ensure no known exploits are included in COTS products. Where a resolution is possible—either by upgrading a library or otherwise addressing the vulnerability—the engineering team will endeavor to implement the fix before the software is released. If the issue cannot be resolved, it will be reported in the Product Support Knowledge Base.

Knowledge base articles disclosing known software vulnerabilities will be reviewed and updated quarterly or as soon as possible when the status changes.

11.3 Design for Safety

Cybersecurity shall be considered during all phases of the software design process. The following guidance is provided to enumerate best design practice:

- Establish the project's risk appetite and maintain an assessment of cyber security risks to build protections appropriate to the evolving threat landscape.
- Implement digital services and update legacy components to allow for easier integration of new security controls in response to changes in business requirements, cyber threats and vulnerabilities.
- Use only the capabilities, software, data and hardware components necessary for a service to mitigate cyber security risks while achieving its intended use.

11.3.1 Coding Standard Security Rules

The following rules shall be followed for all phases of software development.

- No user-provided SQL strings shall be allowed to be input. Specifically, at no point should a user be able to enter an SQL statement into any dialog or input field on any COTS product.
- No user-provided operating system command shall be allowed to be input. Specifically, at no point should a user be allowed to enter an operating system command into any dialog or input field in any COTS product GUI.

11.4 Memory-Safe Languages

Where appropriate and feasible, and also given business considerations, memory-safe languages shall be used. COTS products are primarily built on a legacy C++ codebase, which is extensive and essential for performance computing. When possible and suitable, new products will be developed using memory-safe languages.

12. Quality Assurance

The Software Quality Assurance (SQA) team at MAK is responsible for planning and executing product testing. All COTS products follow a product testing schedule to ensure every product undergoes thorough testing before release. SQA is also tasked with continuously improving general development, testing, and release processes, including daily workflows, to deliver product quality that meets customer requirements and expectations. Embedded within the software development teams, SQA leads the analysis of existing procedures, including test case development, test execution, and documentation improvements.

12.1 QA Tasks

Within a rapid development environment, SQA:

- Maintains tools for product testing.
- Collaborates with TLs and stakeholders.
- Tests partially complete product code before merged into trunk.
- Develops and updates test sets and test cases.
- Works with the product team to execute testing.
- Reports defects, helps with triage, and retests fixes.
- Ensures Quality Control before release through testing and review.
- Reviews product documentation, user guides, and release notes.
- Drives process improvements and ensures high product quality.

12.2 QA Process Artifacts

12.2.1 Planned Test Coverage

Planned Test Coverage serves as the roadmap for QA testers during a release. Completing all planned test coverage is required for release. While the coverage may evolve slightly during the project, its early and clear definition as the release goal is crucial. Progress toward this goal is measurable, e.g., "We're 50% through our plan." Testing targets all platforms, guided by discussions with stakeholders.

12.2.2 Test Matrix

The Test Matrix is stored in SpiraTest or a spreadsheet and lists Test Sets and Test Cases. SQA updates it to track progress accurately. It includes PASS/FAIL status, BUG numbers (cross-referenced to Jira), the Test Case tested, the test run date, and any comments.

12.2.3 Bug Tracking

Defects are reported following the procedures in section 8, Defect Tracking Policies, on page 24.

12.3 QA Testing

12.3.1 QA Setup

Preparation is key for SQA before a test cycle, requiring significant time and effort. The test strategy, approach, priorities, and test cases must be defined and ready, with development personnel and stakeholders accessible as needed.

A test setup can range from a single PC or application to a collection of software and hardware devices. SQA often requires both a main "Test Bed" for planned testing and a separate "Debug Test Bed" for replicating issues. This allows testing to continue while investigating bugs on a different system alongside an engineer.

12.3.2 Testing During Feature Development

Features and defects assigned to the release are tested throughout the development cycle. The SQA shall routinely check the product's Jira status for completed features (those with the Ready for Review status) and fixed bugs (those with the Fixed - Ready for Testing status). SQA then tests those features and bug fixes.

12.3.3 Testing During Release

Testing continues throughout the Major and Feature Release process where engineers and others from the product team may be assigned to assist with testing.

12.3.4 Fixed Defect Verification

For fixed bugs, SQA changes the ticket's status as specified in section 8.2, Defect Workflow (page 26). If the defect is repeated, SQA follows up with the engineer who implemented the fix, providing more information as needed. The cycle continues until the fix is verified.

Sometimes a feature or bug fix cannot be verified before the release, such as a lack of time, engineer availability, or reprioritization of work for the release. In those circumstances, SQA and the TL work together to decide how to proceed.

13. Technical Documentation

The technical writer is responsible for updating the documentation for Major, Feature, and Maintenance releases to reflect the feature and defect changes to the experience of using the product.

The user and developer documentation adds value to the COTS products because they help customers understand, use, integrate, and extend the software more effectively. MAK values the documentation and products are not considered complete until the documentation is updated for the release.

13.1 Standards for COTS Documentation

The procedures for writing and maintaining COTS product documentation are found in the Tech Doc space on Confluence.

13.1.1 Style

All style decisions are determined by the Documentation Lead. They are informed primarily by the Microsoft Style Guide. Common usage—and especially any deviation from the Microsoft Style Guide—shall be documented in the Tech Doc space on Confluence.

13.1.2 Documentation Outputs

Most of the user documentation is provided to customers as PDF files. The primary documentation for most products (such as the user guide) is also delivered as an integrated HTML5 help system. Developer documentation is in HTML format and built using Doxygen.

13.2 Tracking Documentation Work

13.2.1 Product Features and Fixed Defects

There are two custom fields in Jira that we use to track whether a ticket requires changes to the core user documentation or should be included in the Release Notes: Documentation Update Rqmt and Release Notes Rqmt. For information on how we use the statuses, refer to the Tech Doc space on Confluence

13.2.2 Improvements to Documentation

If a Jira ticket is submitted to improve some aspect of the user documentation, it receives the label Documentation. If it is a correction or improvement to the documentation resulting from product support, then the ticket shall be included in the release notes.

Jira tickets for changes to developer documentation shall have the label DevGuide. The technical writer is typically not responsible for those changes, but they are available to assist engineers.

13.3 User Documentation

User documentation describes how to use the software as it is delivered. It may also mention developer practices but, in such cases, the user document should point the reader to the developer documentation. The primary audience is end users.

13.3.1 User Guide

The user guide for a product explains how to use the product, particularly from a GUI perspective. The user guide is delivered as a PDF and also as HTML5.

13.3.2 First Experience Guides and Tutorials

Some products may need a way to walk customers through features. The first experience guides are one of the ways to do this, like a “getting started” guide.

Some features might deserve a tutorial approach; that is, something that walks the user through an example of how to accomplish a complex task.

Either of these tutorials may be a candidate for a video, which is then posted to MAK TV and linked from the website’s Video Tutorials page. The technical writer works with Sales and Marketing to determine areas of interest and creates the videos when there is time available.

13.4 Developer Guide and Class Documentation

The TL and engineers are largely responsible for the developer guide and class documentation that accompany the product’s SDK. They may request that technical writer and review the contents in the developer guide and class documentation for clarity and style.

When extensive work is required for existing or new developer guide or class documentation, the TL may determine that the product team should meet to discuss it.

13.5 Release Documentation

All defects fixed in a Major, Feature, and Maintenance release shall be enumerated with the defect number, a summary and a brief description of the problem and/or resolution in the Release Notes. Defects addressed in a Patch release are enumerated in the Change Log, as described in section 6.3.3, Deliver the Patch (page 17).

The only exceptions to this policy are:

- Defects related to licensing, which have no effect on the customer but where the known existence of said defect would reveal confidential information.
- Defects caused by feature development or bug fixing after a formal release and resolved before the next one.

13.5.1 Capabilities Document

The Capabilities document is a technical brochure for the product. There is also a page on the website with content that matches that document.

The technical writer is responsible for updating that document and the web page with new and changed product functionality. They shall update it for Major and Feature releases.

14. Support

MAK places high value on supporting our customers. We tell them that we are like the "engineer down the hall". This section describes how we achieve our superior level of product support.

14.1 Logging and Tracking Product Support

Support tickets are tracked using Jira Service Management. Tracking product support is very important. If customers write to us outside of Jira, encourage them to use the support system. It ensures that we will provide them with the assistance that they need and not lose track of their request.

14.2 Assignment and Initial Response

Remember these points that make our product support excellent:

Principle	Description
We Respond Promptly	We promise customers a response within one business day, excluding US holidays. A response isn't an answer; it can be as little as a statement that says, "We are looking into it. I will get back to you next in *X units of time*."
Tickets Are Assigned to a Product	Everyone is responsible for looking at unassigned tickets. Please check the "Unassigned Issues" queue in Jira and assign tickets to the correct project if you know them.
Follow Product Team's Policy for Assigning Tickets	Product teams should follow their own policies regarding how tickets are assigned. For example, some teams have a sweep person who can assign tickets while others require everyone to run through the box and pick the tickets they know how to solve. Speak with your TL; be sure your team has a policy to make sure all open tickets are assigned within a single business day. The default is not to assign tickets to a specific person.
Update Fields	Make a conscious effort ensure data fields are filled in. Product names and versions should be filled in. The time you spend on the ticket should be logged.
Check the Title/Subject	Verify the title (Subject) of the ticket is clear. Update the subject to reflect a clear succinct description of the issue if needed.

Principle	Description
Tickets Are Not Left in Limbo	If someone goes on vacation with outstanding tickets, they should work with the TL to arrange for someone to continue working with the customers as needed.

14.3 Triaging Tickets

If …	Then …
The ticket has been assigned	Make sure the customer information is filled in. See section 14.10, Customer and Organization Information (page 46).
There are multiple questions in a single ticket	If the questions are not closely related, and specifically if they cross between multiple products or different areas of expertise on the same product, respond using the canned response for Multiple Questions. Notify your TL that you have done this, and close the ticket. The user will need to repost the questions as two tickets.
Multiple products are involved	Used the canned response for multiple products/questions and then close the ticket.
The customer writes multiple emails opening multiple tickets for one issue	Please write to the customer and explain how their actions make it hard to answer their problem; they usually stop doing it. You can then either close all other tickets with a mark to show that they are duplicates. (If you can, associate each ticket with the ones that will be worked on.)
The ticket is Spam	Mark it as Junk. If it's otherwise irrelevant but comes from someone who could potentially be a customer, assign yourself and mark it as resolved. Please remember, Junk is only for tickets that are completely irrelevant to our business.

14.4 Customers' Questions

These are MAK's standard guidelines for answering support questions:

Rule	Guideline
Verify that the person asking the question is a customer	If the customer information is incomplete (with company and details), ask the customer to better identify themselves before offering support. If that does not work, talk to the TL. They may in turn talk to the PM and/or Sales for further guidance. If the customer does not identify themselves, please put the ticket in a hold position until the customer is identified.

Rule	Guideline
Check whether the customer is international or domestic	Regarding distributors and international customers: Our international customers (outside North America) are supposed to channel their questions through the distributors. The distributors then turn around and ask us for help if they need it. In an ideal world, it would work like that. In general, if you know the customer is from country X, you should politely and firmly encourage them to work through the reseller. Next, answer their question and CC the reseller if possible. We prefer to err on the side of too much support rather than too little. There are several international customers we support directly; please contact your TL for a list.
Verify whether the product version is still supported	Some customers may ask about products we no longer support. If necessary, speak with the TL about whether to proceed and, if so, how. In general, if you know the answer, you should provide it. If you can provide an answer based on the current version – please do so. If there is significant work to resolve or even understand the issue in the obsolete version, talk to the TL.
Do not say no	Never say "no" to a customer without discussing with your TL. Telling customers that we cannot help them will happen from time to time for many reasons. Just make sure you communicate the issue with your TL before ending the ticket.

14.5 Working with Sales

The sales team is not expected to track issues in Jira. - Try not to assign tickets to the sales team if they haven't asked you to. Simply shepherd the process along if they need to be involved.

If …	Then …
A Sales person says they will handle the ticket	Find out if they plan to handle it in Jira or if you need to close the ticket.
The customer is using a very old version, may not have a license, or is doing something "odd"	Bring it to their sales rep's attention. If you don't know who the sales rep is, talk to the TL or other engineers; someone will know to whom the customer's account belongs.
The customer requests a price or a download	Reach out to the Sales team using Slack to ask what they want you to do with the ticket. Once you know, they may ask you to close the ticket or assign it to them.

14.6 Answering Questions and Fixing Bugs

When a ticket is assigned to you, you must work to resolve it. Please remember, our maintenance policy says we aren't required to fix anything for anyone but, if we don't help the customer get their job done, they won't be a customer for long. Your goal in support is to help the customer get their job done. To do that, you can:

- Explain how to solve the problem with the product.
- Explain a workaround.
- Issue a Patch release or a code snippet.
- Other: talk to your TL. The primary goal is to be reasonable.

While working on support tickets:

If …	Then …
You spend time on support, track that time in Jira and your timesheet	Track the time you spend interacting with the customer and working to answer this question. This is time you spend on the support ticket itself, not on any associated Jira feature or defect tickets. (Tip: The total hours you log to your product's Support code for the day should closely match the total time you entered on the support ticket(s) you worked on. On your time sheet, also be sure that you are logging time spent providing support as Support and not Development.)
You can reproduce the problem	Open a Defect ticket in Jira linked to the support ticket. Give the customer the Jira ID for tracking. If you are going to work on it for the customer, please track the time you spend fixing the issue on the Defect ticket.
You cannot reproduce the problem	Work with the customer to see if they have a replication procedure. Let the customer know what you have done to repeat it. Feel free to escalate the issue to the TL. Make it clear without the ability to repeat it, we're unlikely to fix it. You can make a bug if you want, but it's not necessary if there is nothing actionable on it.
You would like a Logger tape of the problem but the customer does not have a license	When a Logger tape of the issue would help but the customer doesn't have a license, speak to the TL. If the customer is able to make a recording of the problem and send it to you, we can provide them with a temporary MAK Data Logger license. The TL will work with Sales.
The support ticket is a feature request	If the customer asks for new functionality, make a Feature ticket in Jira and give the customer the feature ID. Try to figure out the urgency of the feature request and include that information in the feature ticket.

If …	Then …
The issue reported indicates a cyber security problem that impacts customer security	Immediately contact the TL and the PM. The issue will be documented and added to our Known Issues and tagged as a security bulletin.
The customer changes topics	Request that they submit a new ticket to address the new topic. There is a canned response in Jira for this situation. You are allowed to close the ticket once the appropriate response is issued.

14.7 Support Ticket Status

The support project uses the following workflow:

Set the status to …	When …
Open	The ticket is first received.
Assigned	You are working on the ticket.
Waiting for Customer	You ask the customer a question. The ticket's status changes back to Assigned when the customer responds. Tickets in the Waiting for Customer status should close automatically after two weeks.
Reopened	The customer responds to a ticket that was previously closed.
Done	You sent the customer a plausible answer. Tickets automatically reopen if the customer writes back. You won't miss it. (Many customers will not send an email back saying, "Thanks! That worked!" They will simply move on with their task.)
Duplicate	The ticket duplicates another one. You should also link it as a duplicate to the ticket it duplicates.
Junk	The ticket is Spam.
Canceled	No further work should be done on the ticket.

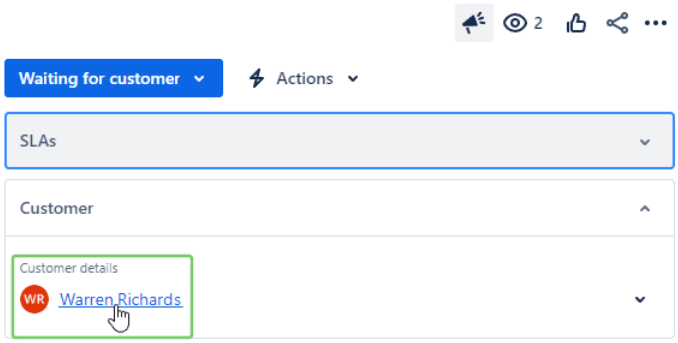
14.8 Closing Tickets

Close tickets aggressively to keep the queue clean. As stated in 14.7, Support Ticket Status, set the ticket to Done when you are reasonably sure you sent the customer a plausible answer. Tickets automatically reopen if there is activity after you close it. While there are certainly reasons to keep a ticket around for a long time (such as you promised someone a build), these should be extremely rare. In these cases, make sure the ticket subject reflects what is holding up closing it.

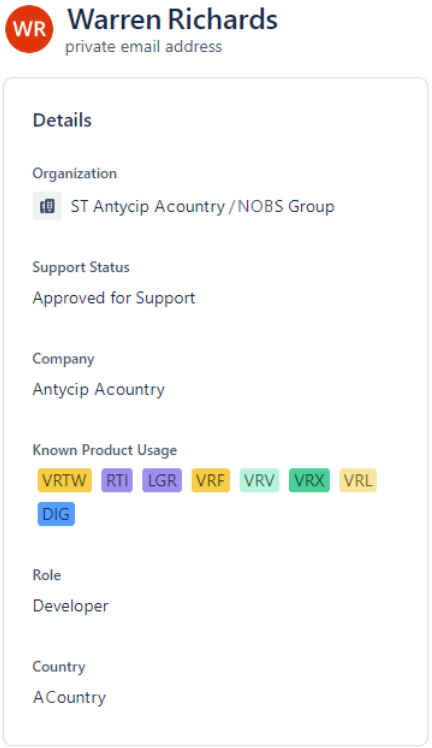
14.9 Customer and Organization Information

14.9.1 Customer Information

We track customer information. To access the customer details, click the name in the support ticket. If the information is incomplete, please work to add it.



Field	Description
Organization	See 13.9.2. Organizations.
Support Status	The statuses are defined in 14.10.2, Customer Statuses, below.
Company	The name of the company that the customer works for.
Known Product Usage	The tags of the products that the customer is known to use.
Role	The role that the customer performs at their company.
Country	The country where the customer is located.



14.9.2 Customer Statuses

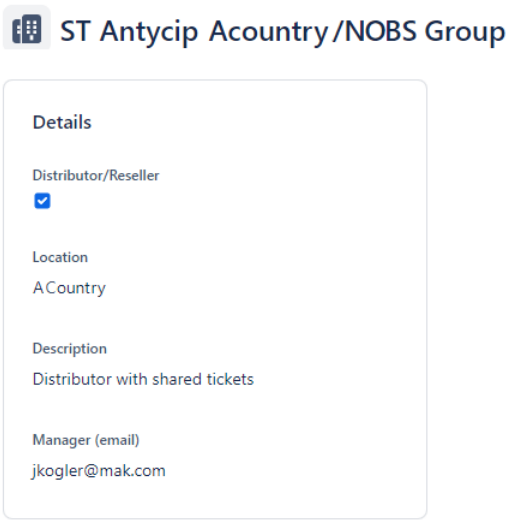
Status	Description
Approved for Support	You can provide support to the customer.
Evaluation	The customer is evaluating the product using a limited license.
MAK Contractor	This is an external person who works for MAK.
MAK Employee	This is an internal person who works for MAK.
MAK Partner	This is a person who works for a partner of MAK.
MAK Distributor	This is a person who works for a distributor of MAK COTS products.
On-Hold - Problem	The person is not currently approved for support.

14.9.3 Organizations

The Organization field in Jira is not used for customers nor companies. It defines a group of users who want to share tickets. It could be people involved in one program inside a company, or a group of people from multiple companies involved in a program.

Field	Description
Distributor/Reseller	Indicates whether the organization is an authorized distributor of COTS software products.

Field	Description
Location	The place where the organization is located. It can be a country or simply a region.
Description	Describe the group, explaining why we have the organization.
Manager (email)	The person responsible for approving members to the organization. This could be a customer or an internal MAK P.O.C. For resellers, it is jkogler@mak.com. Think of this person as the “owner” of the organization.



15. VR-TheWorld Orders and Production

VR-TheWorld releases are coordinated between MAK and our partner company Pelican, who develops the underlying software of the VR-TheWorld server. A VR-TheWorld server is shipped on a hard drive, which contains the VR-TheWorld software, a collection of opensource terrain data, and documentation.

15.1 Server Orders

Step	Description
1. Sales Submits the Order through Product Support	When a customer requests a VR-TheWorld server, the Sales team submits a product support ticket that contains the following information: • The VRTW Customer • Task Type: Server Sale, Docker Software Sale, Internal Use, Evaluation, Replacement, Docker Update • The Distributor • End User • Invoice Number • CRM Opportunity • Support Ticket Number • Quantity • Due Date • Description: Delivery Address • Data Drive OS: Linux or Windows
2. Build Engineer Creates Jira Ticket	The Build Engineer creates a VRTW Hardware ticket to track the drive creation process, using the information supplied in the product support ticket. They associate the VRTW Hardware ticket with the support ticket through Jira.
3. Build Engineer Supplies the Server	The Build Engineer then creates a new VR-TheWorld drive or packages one of the spare VR-TheWorld drives.
4. Office Manager Receives Server for Shipping	The Build Engineer takes the finished drive along with the shipment address and drops it off with the Office Manager. If the Office Manager is not available, the Build Engineer works with someone else to ensure that the package is prepared for shipping.
5. Build Engineer Updates Jira Ticket	The Build Engineer marks the VRTW Hardware ticket as “Ready to Ship” once the package has been prepared for shipping. They also mark the related Product Support ticket to “Done”, which notifies the Salesperson that the shipment has been made.
6. Office Manager Ships Server	The Office Manager hands the drive off to our shipment company.

16. Definitions

Term	Definition
archaeology	Research into engineering work that was previously planned or completed and the context is now needed to inform future decisions.
COTS	Commercial Off-The-Shelf
GUI	Graphical User Interface. Used interchangeably with UI.
IPB	Internal Product Build. See section 4.2.1, Internal Product Builds.
Jira	The tool used by MAK to track features and defects. It is provided and maintained by Atlassian.
MAK	MAK is short for MAK Technologies, Inc.
MAK ONE	MAK COTS products are in one of two categories: applications and infrastructure. • The MAK ONE applications include VR-Vantage products such as VR-Vantage, VR-Vantage with SensorFX, VR-Forces, VR-Engage, and VR-Engage with SensorFX. • The MAK ONE infrastructure products are VR-Link, the MAK RTI, VR-Exchange, the MAK Data Logger, MAK WebLVC Server, VR-TheWorld, and the DI-Guy SDK.
MOPI	The MAK ONE Product Installer (MOPI) is used by MAK employees and distributors of MAK ONE products. It can also be used to generate links that the employee, reseller, or distributor can then send to a customer.
NIGHTBUS	NIGHTBUS is the nightly build system. See section 9.3, The Nightly Build System (Nightbus).
PM	VP Products. See section 3, Team Composition and Roles.
RC	Release Candidate. See section 5.2.3, Final Test Phase.
Regression Test	Regression testing validates that features work correctly after code changes. The goal is to ensure that the product performs at least as well as it did before those changes.
Smoke Test	Smoke testing is conducted before more rigorous testing. The goal is to verify that feature(s) work as expected.
SQA	Software Quality Assurance engineer. See section 3, Team Composition and Roles.
SVN	Subversion, which is the source and version control system for proprietary source code.
TC	Test Candidate. See section 5.2.3, Final Test Phase.
TL	Product Team Lead. See section 3, Team Composition and Roles.
UI	User Interface. Used interchangeably with GUI.

A. MAK Product Release Checklists

A.1 Major, Feature, and Maintenance Releases

This is the MAK Product Release Checklist, which is used to ensure that all stakeholders are aware of the scope of a Major, Feature, or Maintenance release.

Product Name			
Product Version		Who Filled in This Form?	

Engineering			
Are there new features?	Y/N		
Were there API changes?	Y/N		
Were there configuration file changes?	Y/N		
Were there platform support changes?	Y/N	If so, what are they?	
Does any other release depend on this release?	Y/N	If so, which ones?	
Please list any significant changes:			
Does this release require a retrospective?	Y/N	If yes, note when it is scheduled:	

Quality Assurance			
Was this release reviewed by a QA engineer?	Y/N	Who:	
Was a QA engineer involved in testing throughout development?	Y/N	Who:	
On a scale of 1–10, how risky is this release? (with 1 being not risk and 10 being very risky)		Describe the risks and concerns:	
Are there automated tests for this product?	Y/N	If so, did they pass?	Y/N
Was packaging tested on all platforms?	Y/N		
Was this package and product scanned for viruses?	Y/N	Who:	
Was licensing tested on all platforms?	Y/N		

Technical Documentation			
Is the user documentation up to date?	Y/N	If not, why?	
Is the developer documentation up to date?	Y/N	If not, why?	
Is the Capabilities / Technical Brochure up to date?	Y/N	If not, why?	
Are the web pages up to date?	Y/N	If not, why?	
Is the knowledge base up to date?	Y/N	If not, why?	

Security			
Has the Third-Party/Open Source List been updated?	Y/N	If not, why?	
Has the Known Exploited Vulnerabilities Catalog been consulted? (See Known Exploited Vulnerabilities Catalog | CISA.)	Y/N	If not, why?	
Has the product been run through a virus scanner?	Y/N	If not, why?	
Are there any other cybersecurity concerns with this release?	Y/N	If not, why?	

Marketing and Sales			
Is the product **brochure** up to date?	Y/N	If not, why?	
Has the sales team been educated on changes to the product?	Y/N	If not, why?	
Is the product **web page** up to date?	Y/N	If not, why?	
Is the product **video** up to date?	Y/N	If not, why?	
Is there any other launch activity planned?	Y/N	Please describe:	

B. Version Change Checklist for Product Releases

Update	**Release Type**			
	Major	**Minor**	**Maintenance**	**Patch**
version.h and VrfDependencyVersions (VRF Only)	✓	✓	✓	✓

Update	Release Type			
	Major	Minor	Maintenance	Patch
License Maintenance Date	✓	✓	✓	
License Cancel Key	✓	✓	✓	
Toolkit Installation Key (where applicable)	✓	✓		
Lua Doc Version Number (VRF Only)	✓	✓	✓	
SMS Checksum Files (VRF Only)	✓	✓	✓	

B.1 (VR-Forces only) version.h and VrfDependencyVersions

Update for each version change.

The version.h header file is generated from ./include/vrfutil/version.h.in and some settings in VrfDependencyVersions.cmake.

In the VrfDependencyVersions.cmake file:

Set the Variables:	Notes
VRF_MAJOR_VERSION, VRF_MINOR_VERSION, and VRF_MAINTENANCE_VERSION	Used to set a few different string names in version.h, which are usually the same but used for various purposes.
VRF_MESSAGE_MAJOR_VERSION and VRF_MESSAGE_MINOR_VERSION	The combination of these must be unique for each version (major, minor, maintenance, and IPB). These are what determine if a sim will try to decode the messages to it, for example.
VRF_RELEASE_NUM (Linux only)	Updated for each major, minor, and maintenance version. Not for IPBs.

(Note that VRF_MINOR_VERSION refers to the feature version (*B*). See section 9.1, Version Numbering.)

In the version.h.in file:

Set the Variable:	Notes
PLUGIN_COMPATIBLE_VERSION	Usually, you simply set this to "${VRF_VERSION_NUMBER}", which pulls the current version number from the build. However, if you are creating a binary-compatible branch or build, this should be explicitly set to the version it is compatible with.

B.2 License Maintenance Date

Update for each major, minor, and maintenance version.

This is a flexLM code that should encode a date close to the actual release of the product.

User licenses allow for running any version of VR-Forces with a maintenance date up to a specified date (usually 1 year post purchase). That date in their license is compared to the encoded date in VRF.

To update:

1. Edit the file ./include/nonDistributed/licenseConstants.h
2. Find the date you want to encode in the format <*year*>.<*day-number*>
   - day-number must be three digits (add a leading zero if needed).
3. Use the vendorKey.exe program to encode the date:
   - Get vendorKey from SVN: https://repo/svn/vrlink/trunk/vendorKey
4. Put encoded date into the licenseConstants.h, along with comments to know what the encoded value is.

B.3 License Cancel Key

Update for each major, minor, and maintenance version.

This is the special environment variable that allows site-license users to cancel licensing.

It is maintained in ./include/nonDistributed/licenseConstants.h

An arbitrary string is needed that is unique for the product version, so we use "vrforces" followed by the version number spelled out in English as the string, for example: "vrforcesfourten"

It is generated by passing this string into vendorKey.exe (see above).

Update the Product Licensing page on Confluence with the new string.

B.4 Toolkit Installation Key

Update for each major and minor version.

This is the toolkit installation key required by the installer (Windows only) to install the header files, libs, etc.

It is maintained in ..\releaseEngineering\vrf-release.iss with the define MAKLicenseDecodeString

This define must be a string unique to the product and version, so we use "VR-Forces" followed by the version number. (e.g. "VR-Forces4.10")

To update:

1. Set the new string.
2. Run gen.exe (in releaseEngineering\vrforces directory) and pass this string as the "product" using -p (quotes required around the string)
   - Pass a count of 10 or some number.
3. All generated keys are valid, but pick one that doesn't look too similar to a recently used key (just to avoid confusion), and note the new key here: VR-Forces
4. Check in the changed file.

B.5 (VR-Forces Only) Lua Doc Version Number

Update for each major, minor, and maintenance version.

Version number displayed in the Lua Doc is unfortunately not integrated to get the version from version.h at this time. It has to be updated manually.

In file: ..\releaseEngineering\luadoc\modifyHtmlOutput.pl

Edit the two locations that the version number shows to the new version number.

B.6 (VR-Forces Only) SMS Checksum Files

Update for each major, minor, and maintenance version.

Each of the major SMSs has a list of checked in smsChecksum files for old versions. Check each SMS directory and ensure checksums for the previous releases and IPBs are present.

If they are not present:

1. From an install of the previous version, go to the SMS directories.
2. Copy the smsChecksum<*version*>.cksm files from each SMS that matches the installed version.
3. Paste these files into the matching directory (for example, .\data\simulationModelSets\EntityLevel) in your development directory and then check them in.

C. Binary Compatible Builds

Some products maintain a binary compatible branch to release binary compatible patches and releases. This is optional. It is designed to help customers who have many existing plug-ins by fixing issues without the need to rebuild those plug-ins.

C.1 Binary versus Non-Binary Compatible

For each release, the TL may need to ensure that there are two branches in SVN: the “regular” non-binary compatible branch and possibly a binary compatible one.

C.2 Binary Compatible Package Versions

For a binary-compatible release (uncommon):

- Make sure all fixes from the binary-compatible branch are merged to the non-binary-compatible branch.
- (VRF Specific) Check that the PLUGIN_COMPATIBLE_VERSION in ./include/vrfutil/version.h (or version.h.in in newer versions) is set to the base version from which the binary-compatible build was branched.

C.3 Merging Two Branches

The “regular” branch should include all fixes on the binary compatible branch, plus the non-binary compatible fixes made to this branch.

To reduce the need for engineers to continually check in to three places, merge all fixes from the binary-compatible branch to the regular branch only when we are doing a new release from the regular branch.