RoseStacker tasks

Task guides and troubleshooting for RoseStacker. Check the installed release before changing configuration or data.

- Measure RoseStacker changes with stack statistics instead of labels
- Check whether custom dropped items should merge into one stack
- Verify spawner-stack quantity before changing farm output
- Test stacked-block removal before allowing it in a public build
- Give staff RoseStacker inspection access without cleanup powers
- Explain a RoseStacker placeholder count that differs from nearby mobs

Measure RoseStacker changes with stack statistics instead of labels

A larger stack label does not by itself prove that fewer server entities are active. Compare the plugin statistics with a controlled test population before tuning a whole farm.

Check the installed content

RoseStacker stores behaviour for stacked mobs, dropped items, spawners and blocks. A stack label is not a reliable inventory of every underlying entity or item. Test one feature at a time with expendable objects before changing rules that affect existing farms.

Work through the change

1. Record the installed RoseStacker version and the type of stacking under review. Save the relevant configuration and keep other farms unchanged during the measurement window.
2. In game, use the documented `/rs stats` command to record the available stack statistics. Note the time and whether the test chunks are loaded so later results are comparable.
3. Introduce a small known number of expendable test entities or items on staging. Observe merging and record statistics again, using the same loaded area and player position.
4. Change one relevant stacking rule, then repeat the same test population. Compare actual server timing evidence as well as the plugin counters before calling the result a performance improvement.

Verify the result

The counters should explain how the known population is represented after stacking. The test should distinguish a display change from a reduction in active entity processing.

If the result differs

If counts jump unexpectedly, inspect chunk loading and other nearby farms before changing limits. If labels differ but statistics do not, investigate presentation settings separately from stack formation.

Keep a recovery path

Restore the previous rule if the comparison is inconclusive. Remove only the deliberately created test population through normal supported interactions; avoid global clear commands that could destroy other players mobs or items.

References and related help

Check the maintainer reference: RoseStacker documentation, RoseStacker documentation.

Before changing live data, read how to keep a completed recovery copy. For startup errors, use the log collection guide.

Check whether custom dropped items should merge into one stack

Items with similar names can carry different metadata and should not automatically be treated as interchangeable. Test custom-item stacking with a small labelled comparison set.

Check the installed content

RoseStacker stores behaviour for stacked mobs, dropped items, spawners and blocks. A stack label is not a reliable inventory of every underlying entity or item. Test one feature at a time with expendable objects before changing rules that affect existing farms.

Work through the change

1. Record the custom item provider and the RoseStacker version. Back up the staging inventory and the item-stacking configuration before changing compatibility rules.
2. Prepare ordinary identical items, two copies of one custom item and a visibly similar item with different legitimate metadata. Keep the quantities small and record each starting count.
3. Drop each comparison set separately in a clear staging area. Observe whether it merges, then pick it up normally and inspect its identity and total quantity.
4. Repeat after one documented item-stacking adjustment only if the installed reference supports it. Check pickup into a nearly full inventory and compare the preserved custom behaviour after collection.

Verify the result

Compatible copies should retain their total quantity and custom identity, while genuinely different items should remain distinguishable. A merged display must not erase information needed by the

custom-item plugin.

If the result differs

If custom items lose behaviour even without merging, inspect their own provider first. If only one metadata combination fails, retain that exact disposable pair for a compatibility report rather than weakening every comparison rule.

Keep a recovery path

Restore the previous stacking configuration and staging inventory checkpoint if identity or quantity changes. Keep valuable player items out of the test and do not use item-clear commands to tidy a live area.

References and related help

Check the maintainer reference: RoseStacker documentation, RoseStacker documentation.

Before changing live data, read how to keep a completed recovery copy. For startup errors, use the log collection guide.

Verify spawner-stack quantity before changing farm output

A spawner display and its represented quantity need to be understood before adjusting farm yield. Test one known stack rather than changing every spawner multiplier at once.

Check the installed content

RoseStacker stores behaviour for stacked mobs, dropped items, spawners and blocks. A stack label is not a reliable inventory of every underlying entity or item. Test one feature at a time with expendable objects before changing rules that affect existing farms.

Work through the change

1. Record the RoseStacker version, spawner type and known starting quantity. Save the relevant configuration and a small staging-world checkpoint containing the test location.
2. Create a disposable test setup through the supported normal placement workflow. Record the represented quantity using the plugin display or supported inspection method for your release.
3. Observe several spawning cycles with ordinary activation conditions satisfied. Keep entity caps, nearby mobs and player distance comparable, and count actual spawned output.
4. Test the documented normal add or remove interaction with a small quantity. Recheck represented spawners and output before making any farm-yield adjustment.

Verify the result

The setup should account for its starting spawner quantity and maintain it through normal interactions. Output measurements should be consistent enough to separate activation limits from stacking behaviour.

If the result differs

If output stops, inspect Minecraft activation conditions and nearby entity limits before increasing stack output. If quantity differs, examine placement and break permissions or integrations rather than compensating with another spawner.

Keep a recovery path

Restore the staging-world checkpoint and previous configuration if quantity becomes uncertain. Do not break production spawner stacks or issue replacement items during diagnosis; both can complicate accounting and recovery.

References and related help

Check the maintainer reference: RoseStacker documentation, RoseStacker documentation.

Before changing live data, read how to keep a completed recovery copy. For startup errors, use the log collection guide.

Test stacked-block removal before allowing it in a public build

A stacked block can represent more than one ordinary placed block. Check the documented interaction and item return on staging before players use large stacks in valuable builds.

Check the installed content

RoseStacker stores behaviour for stacked mobs, dropped items, spawners and blocks. A stack label is not a reliable inventory of every underlying entity or item. Test one feature at a time with expendable objects before changing rules that affect existing farms.

Work through the change

1. Identify the stacked-block feature and block type supported by your installed release. Save its configuration and a small staging-world backup before placing a disposable test stack.
2. Record the starting block count and create the stack through the normal supported workflow. Keep its location clear of unrelated blocks and protection rules for the first comparison.
3. Use the documented ordinary removal interaction with a normal account. Count what remains represented and what returns to the inventory after each action.
4. Repeat inside a test protected region and with a nearly full inventory. Compare ordinary break behaviour with the protected case instead of granting broad bypass permissions to make it work.

Verify the result

Each permitted action should account for the intended number of blocks without losing or duplicating the represented quantity. Denied actions should leave both the stack and inventory unchanged.

If the result differs

If only protected regions differ, inspect the integration and permission decision. If the unprotected baseline fails, preserve the exact small setup and version details before changing unrelated entity or spawner settings.

Keep a recovery path

Restore the staging-world checkpoint if the test count becomes uncertain. Keep the feature out of public valuable builds until accounting is clear, and avoid global stack-clearing commands as a cleanup method.

References and related help

Check the maintainer reference: RoseStacker documentation, RoseStacker documentation.

Before changing live data, read how to keep a completed recovery copy. For startup errors, use the log collection guide.

Give staff RoseStacker inspection access without cleanup powers

Staff may need to inspect stack state without gaining destructive cleanup commands. Review the exact command permissions and test the restricted role with an ordinary account.

Check the installed content

RoseStacker stores behaviour for stacked mobs, dropped items, spawners and blocks. A stack label is not a reliable inventory of every underlying entity or item. Test one feature at a time with expendable objects before changing rules that affect existing farms.

Work through the change

1. Open the command and permission reference for the installed RoseStacker build. List the inspection actions staff require, including statistics and any documented stack inspection tool.
2. Compare those nodes with commands that remove stacks or alter stored content. Save the permission group before making a narrowly scoped staging change.
3. Grant only the documented inspection access to a temporary group. Test its information commands and, where appropriate, the stack tool on expendable objects in a staging world.
4. Attempt a restricted administrative action using its harmless permission-check path or a disposable staging target. Confirm normal players outside the role cannot obtain inspection tools or administrative information.

Verify the result

The role should read the required stack state while destructive actions remain denied. Inspection should not change represented quantity or modify player-owned objects.

If the result differs

If the role still has cleanup powers, inspect inherited groups and operator status. If inspection fails, compare the exact installed command node rather than adding a plugin-wide wildcard.

Keep a recovery path

Restore the saved permission group if access is broader than intended. Remove disposable tools and temporary grants after the trial, and record the final narrow nodes for future staff onboarding.

References and related help

Check the maintainer reference: RoseStacker documentation.

Before changing live data, read how to keep a completed recovery copy. For startup errors, use the log collection guide.

Explain a RoseStacker placeholder count that differs from nearby mobs

A scoreboard placeholder may report a plugin-wide value rather than the visible entities around one player. Identify its scope before treating a different count as missing mobs.

Check the installed content

RoseStacker stores behaviour for stacked mobs, dropped items, spawners and blocks. A stack label is not a reliable inventory of every underlying entity or item. Test one feature at a time with expendable objects before changing rules that affect existing farms.

Work through the change

1. Record the exact PlaceholderAPI placeholder and the scoreboard or menu that renders it. Consult the installed RoseStacker placeholder reference for its meaning and dependencies.
2. Compare the value with documented RoseStacker statistics at the same timestamp. Note loaded worlds and chunks, because a local visual count may cover a much smaller area.
3. On staging, add one known disposable stack and refresh the display through its supported workflow. Observe which counters change and whether the displayed value is cached.
4. Test the placeholder directly through the documented PlaceholderAPI inspection route for your installed release, then compare the scoreboard result. This separates provider output from formatting or caching.

Verify the result

The value should match its documented scope and update when the corresponding underlying statistic changes. The display should label that scope clearly rather than implying a local head

count.

If the result differs

If direct expansion is correct but the scoreboard is stale, inspect its update interval and cache. If the placeholder remains literal, check expansion availability and syntax before editing stack data.

Keep a recovery path

Restore the previous display expression if the new label or placeholder is incorrect. Keep the actual stack configuration and stored entities untouched; this investigation concerns a reported metric rather than missing world content.

References and related help

Check the maintainer reference: RoseStacker documentation, RoseStacker documentation.

Before changing live data, read how to keep a completed recovery copy. For startup errors, use the log collection guide.