Target Tier Policy

Table of Contents

General

Rust provides three tiers of target support:

- Rust provides no guarantees about tier 3 targets; they exist in the codebase, but may or may not build.
- Rust's continuous integration checks that tier 2 targets will always build, but they may or may not pass tests.
- Rust's continuous integration checks that tier 1 targets will always build and pass tests.

Adding a new tier 3 target imposes minimal requirements; we focus primarily on avoiding disruption to other ongoing Rust development.

Tier 2 and tier 1 targets place work on Rust project developers as a whole, to avoid breaking the target. The broader Rust community may also feel more inclined to support higher-tier targets in their crates (though they are not obligated to do so). Thus, these tiers require commensurate and ongoing efforts from the maintainers of the target, to demonstrate value and to minimize any disruptions to ongoing Rust development.

This policy defines the requirements for accepting a proposed target at a given level of support.

Each tier builds on all the requirements from the previous tier, unless overridden by a stronger requirement. Targets at tier 2 and tier 1 may also provide *host tools* (such as `rustc` and `cargo`); each of those tiers includes a set of supplementary requirements that must be met if supplying host tools for the target. A target at tier 2 or tier 1 is not required to supply host tools, but if it does, it must meet the corresponding additional requirements for host tools.

The policy for each tier also documents the Rust governance teams that must approve the addition of any target at that tier. Those teams are responsible for reviewing and evaluating the target, based on these requirements and their own judgment. Those teams may apply additional requirements, including subjective requirements, such as to deal with issues not foreseen by this policy. (Such requirements may subsequently motivate additions to this policy.)

While these criteria attempt to document the policy, that policy still involves human judgment. Targets must fulfill the spirit of the requirements as well, as determined by the judgment of the approving teams. Reviewers and team members evaluating targets and target-specific patches should always use their own best judgment regarding the quality of work, and the suitability of a target for the Rust project. Neither this policy nor any decisions made regarding targets shall create any binding agreement or estoppel by any party.

Before filing an issue or pull request (PR) to introduce or promote a target, the target should already meet the corresponding tier requirements. This does not preclude an existing target's maintainers using issues (on the Rust repository or otherwise) to track requirements that have not yet been met, as appropriate; however, before officially proposing the introduction or promotion of a target, it should meet all of the necessary requirements. A target proposal must quote the corresponding requirements verbatim and respond to them as part of explaining how the target meets those requirements. (For the requirements that simply state that the target or the target developers must not do something, it suffices to acknowledge the requirement.)

For a list of all supported targets and their corresponding tiers ("tier 3", "tier 2", "tier 2 with host tools", "tier 1", or "tier 1 with host tools"), see platform support.

Several parts of this policy require providing target-specific documentation. Such documentation should typically appear in a subdirectory of the platform-support section of this rustc manual, with a link from the target's entry in platform support. Use TEMPLATE.md as a base, and see other documentation in that directory for examples.

Note that a target must have already received approval for the next lower tier, and spent a reasonable amount of time at that tier, before making a proposal for promotion to the next higher tier; this is true even if a target meets the requirements for several tiers at once. This policy leaves the precise interpretation of "reasonable amount of time" up to the approving teams; those teams may scale the amount of time required based on their confidence in the target and its demonstrated track record at its current tier. At a minimum, multiple stable releases of Rust should typically occur between promotions of a target.

The availability or tier of a target in stable Rust is not a hard stability guarantee about the future availability or tier of that target. Higher-level target tiers are an increasing commitment to the support of a target, and we will take that commitment and potential disruptions into account when evaluating the potential demotion or removal of a target that has been part of a stable release. The promotion or demotion of a target will not generally affect existing stable releases, only current development and future releases.

In this policy, the words "must" and "must not" specify absolute requirements that a target must meet to qualify for a tier. The words "should" and "should not" specify requirements that apply in almost all cases, but for which the approving

teams may grant an exception for good reason. The word "may" indicates something entirely optional, and does not indicate guidance or recommendations. This language is based on IETF RFC 2119.

Tier 3 target policy

At this tier, the Rust project provides no official support for a target, so we place minimal requirements on the introduction of targets.

A proposed new tier 3 target must be reviewed and approved by a member of the compiler team based on these requirements. The reviewer may choose to gauge broader compiler team consensus via a Major Change Proposal (MCP).

A proposed target or target-specific patch that substantially changes code shared with other targets (not just target-specific code) must be reviewed and approved by the appropriate team for that shared code before acceptance.

- A tier 3 target must have a designated developer or developers (the "target maintainers") on record to be CCed when issues arise regarding the target. (The mechanism to track and CC such developers may evolve over time.)
- Targets must use naming consistent with any existing targets; for instance, a target for the same CPU or OS as an existing Rust target should use the same name for that CPU or OS. Targets should normally use the same names and naming conventions as used elsewhere in the broader ecosystem beyond Rust (such as in other toolchains), unless they have a very good reason to diverge. Changing the name of a target can be highly disruptive, especially once the target reaches a higher tier, so getting the name right is important even for a tier 3 target.
  - Target names should not introduce undue confusion or ambiguity unless absolutely necessary to maintain ecosystem compatibility. For example, if the name of the target makes people extremely likely to form incorrect beliefs about what it targets, the name should be changed or augmented to disambiguate it.
- Tier 3 targets may have unusual requirements to build or use, but must not create legal issues or impose onerous legal terms for the Rust project or for Rust developers or users.
  - The target must not introduce license incompatibilities.
  - Anything added to the Rust repository must be under the standard Rust license (`MIT OR Apache-2.0`).
  - The target must not cause the Rust tools or libraries built for any other host (even when supporting cross-compilation to the target) to depend on any new dependency less permissive than the Rust licensing policy. This applies whether the dependency is a Rust crate that would require adding new license exceptions (as specified by the `tidy` tool in the rust-lang/rust repository), or whether the dependency is a native library or binary. In other words, the introduction of the target must not cause a user installing or running a version of Rust

or the Rust tools to be subject to any new license requirements.
  - Compiling, linking, and emitting functional binaries, libraries, or other code for the target (whether hosted on the target itself or cross-compiling from another target) must not depend on proprietary (non-FOSS) libraries. Host tools built for the target itself may depend on the ordinary runtime libraries supplied by the platform and commonly used by other applications built for the target, but those libraries must not be required for code generation for the target; cross-compilation to the target must not require such libraries at all. For instance, `rustc` built for the target may depend on a common proprietary C runtime library or console output library, but must not depend on a proprietary code generation library or code optimization library. Rust's license permits such combinations, but the Rust project has no interest in maintaining such combinations within the scope of Rust itself, even at tier 3.
  - "onerous" here is an intentionally subjective term. At a minimum, "onerous" legal/licensing terms include but are *not* limited to: non-disclosure requirements, non-compete requirements, contributor license agreements (CLAs) or equivalent, "non-commercial"/"research-only"/etc terms, requirements conditional on the employer or employment of any particular Rust developers, revocable terms, any requirements that create liability for the Rust project or its developers or users, or any requirements that adversely affect the livelihood or prospects of the Rust project or its developers or users.
- Neither this policy nor any decisions made regarding targets shall create any binding agreement or estoppel by any party. If any member of an approving Rust team serves as one of the maintainers of a target, or has any legal or employment requirement (explicit or implicit) that might affect their decisions regarding a target, they must recuse themselves from any approval decisions regarding the target's tier status, though they may otherwise participate in discussions.
  - This requirement does not prevent part or all of this policy from being cited in an explicit contract or work agreement (e.g. to implement or maintain support for a target). This requirement exists to ensure that a developer or team responsible for reviewing and approving a target does not face any legal threats or obligations that would prevent them from freely exercising their judgment in such approval, even if such judgment involves subjective matters or goes beyond the letter of these requirements.
- Tier 3 targets should attempt to implement as much of the standard libraries as possible and appropriate (`core` for most targets, `alloc` for targets that can support dynamic memory allocation, `std` for targets with an operating system or equivalent layer of system-provided functionality), but may leave some code unimplemented (either unavailable or stubbed out as appropriate), whether because the target makes it impossible to implement or challenging to implement. The authors of pull requests are

not obligated to avoid calling any portions of the standard library on the basis of a tier 3 target not implementing those portions.

- The target must provide documentation for the Rust community explaining how to build for the target, using cross-compilation if possible. If the target supports running binaries, or running tests (even if they do not pass), the documentation must explain how to run such binaries or tests for the target, using emulation if possible or dedicated hardware if necessary.
- Tier 3 targets must not impose burden on the authors of pull requests, or other developers in the community, to maintain the target. In particular, do not post comments (automated or manual) on a PR that derail or suggest a block on the PR based on a tier 3 target. Do not send automated messages or notifications (via any medium, including via `@`) to a PR author or others involved with a PR regarding a tier 3 target, unless they have opted into such messages.
  - Backlinks such as those generated by the issue/PR tracker when linking to an issue or PR are not considered a violation of this policy, within reason. However, such messages (even on a separate repository) must not generate notifications to anyone involved with a PR who has not requested such notifications.
- Patches adding or updating tier 3 targets must not break any existing tier 2 or tier 1 target, and must not knowingly break another tier 3 target without approval of either the compiler team or the maintainers of the other tier 3 target.
  - In particular, this may come up when working on closely related targets, such as variations of the same architecture with different features. Avoid introducing unconditional uses of features that another variation of the target may not have; use conditional compilation or runtime detection, as appropriate, to let each target run code supported by that target.

If a tier 3 target stops meeting these requirements, or the target maintainers no longer have interest or time, or the target shows no signs of activity and has not built for some time, or removing the target would improve the quality of the Rust codebase, we may post a PR to remove it; any such PR will be CCed to the target maintainers (and potentially other people who have previously worked on the target), to check potential interest in improving the situation.

Tier 2 target policy

At this tier, the Rust project guarantees that a target builds, and will reject patches that fail to build on a target. Thus, we place requirements that ensure the target will not block forward progress of the Rust project.

A proposed new tier 2 target must be reviewed and approved by the compiler team based on these requirements. Such review and approval may occur via a Major Change Proposal (MCP).

In addition, the infrastructure team must approve the integration of the target into Continuous Integration (CI), and the tier 2 CI-related requirements. This review and approval may take place in a PR adding the target to CI, or simply by an infrastructure team member reporting the outcome of a team discussion.

- A tier 2 target must have value to people other than its maintainers. (It may still be a niche target, but it must not be exclusively useful for an inherently closed group.)
- A tier 2 target must have a designated team of developers (the "target maintainers") available to consult on target-specific build-breaking issues, or if necessary to develop target-specific language or library implementation details. This team must have at least 2 developers.
  - The target maintainers should not only fix target-specific issues, but should use any such issue as an opportunity to educate the Rust community about portability to their target, and enhance documentation of the target.
- The target must not place undue burden on Rust developers not specifically concerned with that target. Rust developers are expected to not gratuitously break a tier 2 target, but are not expected to become experts in every tier 2 target, and are not expected to provide target-specific implementations for every tier 2 target.
- The target must provide documentation for the Rust community explaining how to build for the target using cross-compilation, and explaining how to run tests for the target. If at all possible, this documentation should show how to run Rust programs and tests for the target using emulation, to allow anyone to do so. If the target cannot be feasibly emulated, the documentation should explain how to obtain and work with physical hardware, cloud systems, or equivalent.
- The target must document its baseline expectations for the features or versions of CPUs, operating systems, libraries, runtime environments, and similar.
- If introducing a new tier 2 or higher target that is identical to an existing Rust target except for the baseline expectations for the features or versions of CPUs, operating systems, libraries, runtime environments, and similar, then the proposed target must document to the satisfaction of the approving teams why the specific difference in baseline expectations provides sufficient value to justify a separate target.
  - Note that in some cases, based on the usage of existing targets within the Rust community, Rust developers or a target's maintainers may wish to modify the baseline expectations of a target, or split an existing target into multiple targets with different baseline expectations. A proposal to do so will be treated similarly to the analogous promotion, demotion, or removal of a target, according to this policy, with the same team approvals required.
    - For instance, if an OS version has become obsolete and unsupported, a target for that OS may raise its baseline expectations

for OS version (treated as though removing a target corresponding to the older versions), or a target for that OS may split out support for older OS versions into a lower-tier target (treated as though demoting a target corresponding to the older versions, and requiring justification for a new target at a lower tier for the older OS versions).

- Tier 2 targets must not leave any significant portions of `core` or the standard library unimplemented or stubbed out, unless they cannot possibly be supported on the target.
  - The right approach to handling a missing feature from a target may depend on whether the target seems likely to develop the feature in the future. In some cases, a target may be co-developed along with Rust support, and Rust may gain new features on the target as that target gains the capabilities to support those features.
  - As an exception, a target identical to an existing tier 1 target except for lower baseline expectations for the OS, CPU, or similar, may propose to qualify as tier 2 (but not higher) without support for `std` if the target will primarily be used in `no_std` applications, to reduce the support burden for the standard library. In this case, evaluation of the proposed target's value will take this limitation into account.
- The code generation backend for the target should not have deficiencies that invalidate Rust safety properties, as evaluated by the Rust compiler team. (This requirement does not apply to arbitrary security enhancements or mitigations provided by code generation backends, only to those properties needed to ensure safe Rust code cannot cause undefined behavior or other unsoundness.) If this requirement does not hold, the target must clearly and prominently document any such limitations as part of the target's entry in the target tier list, and ideally also via a failing test in the testsuite. The Rust compiler team must be satisfied with the balance between these limitations and the difficulty of implementing the necessary features.
  - For example, if Rust relies on a specific code generation feature to ensure that safe code cannot overflow the stack, the code generation for the target should support that feature.
  - If the Rust compiler introduces new safety properties (such as via new capabilities of a compiler backend), the Rust compiler team will determine if they consider those new safety properties a best-effort improvement for specific targets, or a required property for all Rust targets. In the latter case, the compiler team may require the maintainers of existing targets to either implement and confirm support for the property or update the target tier list with documentation of the missing property.
- If the target supports C code, and the target has an interoperable calling convention for C code, the Rust target must support that C calling convention for the platform via `extern "C"`. The C calling convention does not need to be the default Rust calling convention for the target, however.
- The target must build reliably in CI, for all components that Rust's CI

considers mandatory.

- The approving teams may additionally require that a subset of tests pass in CI, such as enough to build a functional "hello world" program, `./x.py test --no-run`, or equivalent "smoke tests". In particular, this requirement may apply if the target builds host tools, or if the tests in question provide substantial value via early detection of critical problems.
- Building the target in CI must not take substantially longer than the current slowest target in CI, and should not substantially raise the maintenance burden of the CI infrastructure. This requirement is subjective, to be evaluated by the infrastructure team, and will take the community importance of the target into account.
- Tier 2 targets should, if at all possible, support cross-compiling. Tier 2 targets should not require using the target as the host for builds, even if the target supports host tools.
- In addition to the legal requirements for all targets (specified in the tier 3 requirements), because a tier 2 target typically involves the Rust project building and supplying various compiled binaries, incorporating the target and redistributing any resulting compiled binaries (e.g. built libraries, host tools if any) must not impose any onerous license requirements on any members of the Rust project, including infrastructure team members and those operating CI systems. This is a subjective requirement, to be evaluated by the approving teams.
  - As an exception to this, if the target's primary purpose is to build components for a Free and Open Source Software (FOSS) project licensed under "copyleft" terms (terms which require licensing other code under compatible FOSS terms), such as kernel modules or plugins, then the standard libraries for the target may potentially be subject to copyleft terms, as long as such terms are satisfied by Rust's existing practices of providing full corresponding source code. Note that anything added to the Rust repository itself must still use Rust's standard license terms.
- Tier 2 targets must not impose burden on the authors of pull requests, or other developers in the community, to ensure that tests pass for the target. In particular, do not post comments (automated or manual) on a PR that derail or suggest a block on the PR based on tests failing for the target. Do not send automated messages or notifications (via any medium, including via `@`) to a PR author or others involved with a PR regarding the PR breaking tests on a tier 2 target, unless they have opted into such messages.
  - Backlinks such as those generated by the issue/PR tracker when linking to an issue or PR are not considered a violation of this policy, within reason. However, such messages (even on a separate repository) must not generate notifications to anyone involved with a PR who has not requested such notifications.
- The target maintainers should regularly run the testsuite for the target, and should fix any test failures in a reasonably timely fashion.

- All requirements for tier 3 apply.

A tier 2 target may be demoted or removed if it no longer meets these requirements. Any proposal for demotion or removal will be CCed to the target maintainers, and will be communicated widely to the Rust community before being dropped from a stable release. (The amount of time between such communication and the next stable release may depend on the nature and severity of the failed requirement, the timing of its discovery, whether the target has been part of a stable release yet, and whether the demotion or removal can be a planned and scheduled action.)

In some circumstances, especially if the target maintainers do not respond in a timely fashion, Rust teams may land pull requests that temporarily disable some targets in the nightly compiler, in order to implement a feature not yet supported by those targets. (As an example, this happened when introducing the 128-bit types `u128` and `i128`.) Such a pull request will include notification and coordination with the maintainers of such targets, and will ideally happen towards the beginning of a new development cycle to give maintainers time to update their targets. The maintainers of such targets will then be expected to implement the corresponding target-specific support in order to re-enable the target. If the maintainers of such targets cannot provide such support in time for the next stable release, this may result in demoting or removing the targets.

Tier 2 with host tools

Some tier 2 targets may additionally have binaries built to run on them as a host (such as `rustc` and `cargo`). This allows the target to be used as a development platform, not just a compilation target.

A proposed new tier 2 target with host tools must be reviewed and approved by the compiler team based on these requirements. Such review and approval may occur via a Major Change Proposal (MCP).

In addition, the infrastructure team must approve the integration of the target's host tools into Continuous Integration (CI), and the CI-related requirements for host tools. This review and approval may take place in a PR adding the target's host tools to CI, or simply by an infrastructure team member reporting the outcome of a team discussion.

- Depending on the target, its capabilities, its performance, and the likelihood of use for any given tool, the host tools provided for a tier 2 target may include only `rustc` and `cargo`, or may include additional tools such as `clippy` and `rustfmt`.
- Approval of host tools will take into account the additional time required to build the host tools, and the substantial additional storage required for the host tools.
- The host tools must have direct value to people other than the target's maintainers. (It may still be a niche target, but the host tools must not be

exclusively useful for an inherently closed group.) This requirement will be evaluated independently from the corresponding tier 2 requirement.
    - The requirement to provide "direct value" means that it does not suffice to argue that having host tools will help the target's maintainers more easily provide the target to others. The tools themselves must provide value to others.
- There must be a reasonable expectation that the host tools will be used, for purposes other than to prove that they can be used.
- The host tools must build and run reliably in CI (for all components that Rust's CI considers mandatory), though they may or may not pass tests.
- Building host tools for the target must not take substantially longer than building host tools for other targets, and should not substantially raise the maintenance burden of the CI infrastructure.
- The host tools must provide a substantively similar experience as on other targets, subject to reasonable target limitations.
    - Adding a substantively different interface to an existing tool, or a target-specific interface to the functionality of an existing tool, requires design and implementation approval (e.g. RFC/MCP) from the appropriate approving teams for that tool.
        * Such an interface should have a design that could potentially work for other targets with similar properties.
        * This should happen separately from the review and approval of the target, to simplify the target review and approval processes, and to simplify the review and approval processes for the proposed new interface.
    - By way of example, a target that runs within a sandbox may need to modify the handling of files, tool invocation, and similar to meet the expectations and conventions of the sandbox, but must not introduce a separate "sandboxed compilation" interface separate from the CLI interface without going through the normal approval process for such an interface. Such an interface should take into account potential other targets with similar sandboxes.
- If the host tools for the platform would normally be expected to be signed or equivalent (e.g. if running unsigned binaries or similar involves a "developer mode" or an additional prompt), it must be possible for the Rust project's automated builds to apply the appropriate signature process, without any manual intervention by either Rust developers, target maintainers, or a third party. This process must meet the approval of the infrastructure team.
    - This process may require one-time or semi-regular manual steps by the infrastructure team, such as registration or renewal of a signing key. Any such manual process must meet the approval of the infrastructure team.
    - This process may require the execution of a legal agreement with the signature provider. Such a legal agreement may be revocable, and may potentially require a nominal fee, but must not be otherwise

onerous. Any such legal agreement must meet the approval of the infrastructure team. (The infrastructure team is not expected or required to sign binding legal agreements on behalf of the Rust project; this review and approval exists to ensure no terms are onerous or cause problems for infrastructure, especially if such terms may impose requirements or obligations on people who have access to target-specific infrastructure.)
  - Changes to this process, or to any legal agreements involved, may cause a target to stop meeting this requirement.
  - This process involved must be available under substantially similar non-onerous terms to the general public. Making it available exclusively to the Rust project does not suffice.
  - This requirement exists to ensure that Rust builds, including nightly builds, can meet the necessary requirements to allow users to smoothly run the host tools.
- Providing host tools does not exempt a target from requirements to support cross-compilation if at all possible.
- All requirements for tier 2 apply.

A target may be promoted directly from tier 3 to tier 2 with host tools if it meets all the necessary requirements, but doing so may introduce substantial additional complexity. If in doubt, the target should qualify for tier 2 without host tools first.

Tier 1 target policy

At this tier, the Rust project guarantees that a target builds and passes all tests, and will reject patches that fail to build or pass the testsuite on a target. We hold tier 1 targets to our highest standard of requirements.

A proposed new tier 1 target must be reviewed and approved by the compiler team based on these requirements. In addition, the release team must approve the viability and value of supporting the target. For a tier 1 target, this will typically take place via a full RFC proposing the target, to be jointly reviewed and approved by the compiler team and release team.

In addition, the infrastructure team must approve the integration of the target into Continuous Integration (CI), and the tier 1 CI-related requirements. This review and approval may take place in a PR adding the target to CI, by an infrastructure team member reporting the outcome of a team discussion, or by including the infrastructure team in the RFC proposing the target.

- Tier 1 targets must have substantial, widespread interest within the developer community, and must serve the ongoing needs of multiple production users of Rust across multiple organizations or projects. These requirements are subjective, and determined by consensus of the approving teams. A tier 1 target may be demoted or removed if it becomes obsolete or no longer meets this requirement.

- The target maintainer team must include at least 3 developers.
- The target must build and pass tests reliably in CI, for all components that Rust's CI considers mandatory.
  - The target must not disable an excessive number of tests or pieces of tests in the testsuite in order to do so. This is a subjective requirement.
  - If the target does not have host tools support, or if the target has low performance, the infrastructure team may choose to have CI cross-compile the testsuite from another platform, and then run the compiled tests either natively or via accurate emulation. However, the approving teams may take such performance considerations into account when determining the viability of the target or of its host tools.
- The target must provide as much of the Rust standard library as is feasible and appropriate to provide. For instance, if the target can support dynamic memory allocation, it must provide an implementation of `alloc` and the associated data structures.
- Building the target and running the testsuite for the target must not take substantially longer than other targets, and should not substantially raise the maintenance burden of the CI infrastructure.
  - In particular, if building the target takes a reasonable amount of time, but the target cannot run the testsuite in a timely fashion due to low performance of either native code or accurate emulation, that alone may prevent the target from qualifying as tier 1.
- If running the testsuite requires additional infrastructure (such as physical systems running the target), the target maintainers must arrange to provide such resources to the Rust project, to the satisfaction and approval of the Rust infrastructure team.
  - Such resources may be provided via cloud systems, via emulation, or via physical hardware.
  - If the target requires the use of emulation to meet any of the tier requirements, the approving teams for those requirements must have high confidence in the accuracy of the emulation, such that discrepancies between emulation and native operation that affect test results will constitute a high-priority bug in either the emulation or the implementation of the target.
  - If it is not possible to run the target via emulation, these resources must additionally be sufficient for the Rust infrastructure team to make them available for access by Rust team members, for the purposes of development and testing. (Note that the responsibility for doing target-specific development to keep the target well maintained remains with the target maintainers. This requirement ensures that it is possible for other Rust developers to test the target, but does not obligate other Rust developers to make target-specific fixes.)
  - Resources provided for CI and similar infrastructure must be available for continuous exclusive use by the Rust project. Resources provided for access by Rust team members for development and testing must be

available on an exclusive basis when in use, but need not be available on a continuous basis when not in use.

- Tier 1 targets must not have a hard requirement for signed, verified, or otherwise "approved" binaries. Developers must be able to build, run, and test binaries for the target on systems they control, or provide such binaries for others to run. (Doing so may require enabling some appropriate "developer mode" on such systems, but must not require the payment of any additional fee or other consideration, or agreement to any onerous legal agreements.)
  - The Rust project may decide to supply appropriately signed binaries if doing so provides a smoother experience for developers using the target, and a tier 2 target with host tools already requires providing appropriate mechanisms that enable our infrastructure to provide such signed binaries. However, this additional tier 1 requirement ensures that Rust developers can develop and test Rust software for the target (including Rust itself), and that development or testing for the target is not limited.
- All requirements for tier 2 apply.

A tier 1 target may be demoted if it no longer meets these requirements but still meets the requirements for a lower tier. Any proposal for demotion of a tier 1 target requires a full RFC process, with approval by the compiler and release teams. Any such proposal will be communicated widely to the Rust community, both when initially proposed and before being dropped from a stable release. A tier 1 target is highly unlikely to be directly removed without first being demoted to tier 2 or tier 3. (The amount of time between such communication and the next stable release may depend on the nature and severity of the failed requirement, the timing of its discovery, whether the target has been part of a stable release yet, and whether the demotion or removal can be a planned and scheduled action.)

Raising the baseline expectations of a tier 1 target (such as the minimum CPU features or OS version required) requires the approval of the compiler and release teams, and should be widely communicated as well, but does not necessarily require a full RFC.

Tier 1 with host tools

Some tier 1 targets may additionally have binaries built to run on them as a host (such as `rustc` and `cargo`). This allows the target to be used as a development platform, not just a compilation target.

A proposed new tier 1 target with host tools must be reviewed and approved by the compiler team based on these requirements. In addition, the release team must approve the viability and value of supporting host tools for the target. For a tier 1 target, this will typically take place via a full RFC proposing the target, to be jointly reviewed and approved by the compiler team and release team.

In addition, the infrastructure team must approve the integration of the target's host tools into Continuous Integration (CI), and the CI-related requirements for host tools. This review and approval may take place in a PR adding the target's host tools to CI, by an infrastructure team member reporting the outcome of a team discussion, or by including the infrastructure team in the RFC proposing the target.

- Tier 1 targets with host tools should typically include all of the additional tools such as `clippy` and `rustfmt`, unless there is a target-specific reason why a tool cannot possibly make sense for the target.
  - Unlike with tier 2, for tier 1 we will not exclude specific tools on the sole basis of them being less likely to be used; rather, we'll take that into account when considering whether the target should be at tier 1 with host tools. In general, on any tier 1 target with host tools, people should be able to expect to find and install all the same components that they would for any other tier 1 target with host tools.
- Approval of host tools will take into account the additional time required to build the host tools, and the substantial additional storage required for the host tools.
- Host tools for the target must have substantial, widespread interest within the developer community, and must serve the ongoing needs of multiple production users of Rust across multiple organizations or projects. These requirements are subjective, and determined by consensus of the approving teams. This requirement will be evaluated independently from the corresponding tier 1 requirement; it is possible for a target to have sufficient interest for cross-compilation, but not have sufficient interest for native compilation. The host tools may be dropped if they no longer meet this requirement, even if the target otherwise qualifies as tier 1.
- The host tools must build, run, and pass tests reliably in CI, for all components that Rust's CI considers mandatory.
  - The target must not disable an excessive number of tests or pieces of tests in the testsuite in order to do so. This is a subjective requirement.
- Building the host tools and running the testsuite for the host tools must not take substantially longer than other targets, and should not substantially raise the maintenance burden of the CI infrastructure.
  - In particular, if building the target's host tools takes a reasonable amount of time, but the target cannot run the testsuite in a timely fashion due to low performance of either native code or accurate emulation, that alone may prevent the target from qualifying as tier 1 with host tools.
- Providing host tools does not exempt a target from requirements to support cross-compilation if at all possible.
- All requirements for tier 2 targets with host tools apply.
- All requirements for tier 1 apply.

A target seeking promotion to tier 1 with host tools should typically either be tier 2 with host tools or tier 1 without host tools, to reduce the number of

requirements to simultaneously review and approve.

In addition to the general process for demoting a tier 1 target, a tier 1 target with host tools may be demoted (including having its host tools dropped, or being demoted to tier 2 with host tools) if it no longer meets these requirements but still meets the requirements for a lower tier. Any proposal for demotion of a tier 1 target (with or without host tools) requires a full RFC process, with approval by the compiler and release teams. Any such proposal will be communicated widely to the Rust community, both when initially proposed and before being dropped from a stable release.