(index.html)

Rust Forge

Rust Platform Support

The Rust compiler runs on, and compiles to, a great number of platforms, though not all platforms are equally supported. Rust's support levels are organized into three tiers, each with a different set of guarantees.

Platforms are identified by their "target triple" which is the string to inform the compiler what kind of output should be produced. The columns below indicate whether the corresponding component works on the specified platform.

Tier 1

Tier 1 platforms can be thought of as "guaranteed to work". Specifically they will each satisfy the following requirements:

- Official binary releases are provided for the platform.
- Automated testing is set up to run tests for the platform.
- Landing changes to the `rust-lang/rust` repository's master branch is gated on tests passing.
- Documentation for how to use and how to build the platform is available.

Target	std	rustc	cargo	notes
`i686-apple-darwin`	✓	✓	✓	32-bit OSX (10.7+, Lion+)
`i686-pc-windows-gnu`	✓	✓	✓	32-bit MinGW (Windows 7+)
`i686-pc-windows-msvc`	✓	✓	✓	32-bit MSVC (Windows 7+)
`i686-unknown-linux-gnu`	✓	✓	✓	32-bit Linux (2.6.18+)
`x86_64-apple-darwin`	✓	✓	✓	64-bit OSX (10.7+, Lion+)
`x86_64-pc-windows-gnu`	✓	✓	✓	64-bit MinGW (Windows 7+)
`x86_64-pc-windows-msvc`	✓	✓	✓	64-bit MSVC (Windows 7+)
`x86_64-unknown-linux-gnu`	✓	✓	✓	64-bit Linux (2.6.18+)

Tier 2

Tier 2 platforms can be thought of as "guaranteed to build". Automated tests are not run so it's not guaranteed to produce a working build, but platforms often work to quite a good degree and patches are always welcome! Specifically, these platforms are required to have each of the following:

- Official binary releases are provided for the platform.
- Automated building is set up, but may not be running tests.
- Landing changes to the `rust-lang/rust` repository's master branch is gated on platforms **building**. For some platforms only the standard library is compiled, but for others `rustc` and `cargo` are too.

Target	std	rustc	cargo	notes
`aarch64-apple-ios`	✓			ARM64 iOS
`aarch64-unknown-cloudabi`	✓			ARM64 CloudABI
`aarch64-linux-android`	✓			ARM64 Android
`aarch64-unknown-fuchsia`	✓			ARM64 Fuchsia
`aarch64-unknown-linux-gnu`	✓	✓	✓	ARM64 Linux
`aarch64-unknown-linux-musl`	✓			ARM64 Linux with MUSL
`arm-linux-androideabi`	✓			ARMv7 Android
`arm-unknown-linux-gnueabi`	✓	✓	✓	ARMv6 Linux
`arm-unknown-linux-gnueabihf`	✓	✓	✓	ARMv6 Linux, hardfloat
`arm-unknown-linux-musleabi`	✓			ARMv6 Linux with MUSL
`arm-unknown-linux-musleabihf`	✓			ARMv6 Linux, MUSL, hardfloat
`armv5te-unknown-linux-gnueabi`	✓			ARMv5TE Linux
`armv7-apple-ios`	✓			ARMv7 iOS, Cortex-a8
`armv7-linux-androideabi`	✓			ARMv7a Android
`armv7-unknown-cloudabi-eabihf`	✓			ARMv7 CloudABI, hardfloat
`armv7-unknown-linux-gnueabihf`	✓	✓	✓	ARMv7 Linux
`armv7-unknown-linux-musleabihf`	✓			ARMv7 Linux with MUSL
`armv7s-apple-ios`	✓			ARMv7 iOS, Cortex-a9
`asmjs-unknown-emscripten`	✓			asm.js via Emscripten
`i386-apple-ios`	✓			32-bit x86 iOS
`i586-pc-windows-msvc`	✓			32-bit Windows w/o SSE
`i586-unknown-linux-gnu`	✓			32-bit Linux w/o SSE
`i586-unknown-linux-musl`	✓			32-bit Linux w/o SSE, MUSL
`i686-linux-android`	✓			32-bit x86 Android
`i686-unknown-cloudabi`	✓			32-bit CloudABI
`i686-unknown-freebsd`	✓	✓	✓	32-bit FreeBSD
`i686-unknown-linux-musl`	✓			32-bit Linux with MUSL
`mips-unknown-linux-gnu`	✓	✓	✓	MIPS Linux
`mips-unknown-linux-musl`	✓			MIPS Linux with MUSL
`mips64-unknown-linux-gnuabi64`	✓	✓	✓	MIPS64 Linux, n64 ABI
`mips64el-unknown-linux-gnuabi64`	✓	✓	✓	MIPS64 (LE) Linux, n64 ABI
`mipsel-unknown-linux-gnu`	✓	✓	✓	MIPS (LE) Linux
`mipsel-unknown-linux-musl`	✓			MIPS (LE) Linux with MUSL
`powerpc-unknown-linux-gnu`	✓	✓	✓	PowerPC Linux
`powerpc64-unknown-linux-gnu`	✓	✓	✓	PPC64 Linux
`powerpc64le-unknown-linux-gnu`	✓	✓	✓	PPC64LE Linux
`s390x-unknown-linux-gnu`	✓	✓	✓	S390x Linux

Target	std	rustc	cargo	notes
`sparc64-unknown-linux-gnu`	✓			SPARC Linux
`sparcv9-sun-solaris`	✓			SPARC Solaris 10/11, illumos
`wasm32-unknown-unknown`	✓			WebAssembly
`wasm32-unknown-emscripten`	✓			WebAssembly via Emscripten
`x86_64-apple-ios`	✓			64-bit x86 iOS
`x86_64-linux-android`	✓			64-bit x86 Android
`x86_64-rumprun-netbsd`	✓			64-bit NetBSD Rump Kernel
`x86_64-sun-solaris`	✓			64-bit Solaris 10/11, illumos
`x86_64-unknown-cloudabi`	✓			64-bit CloudABI
`x86_64-unknown-freebsd`	✓	✓	✓	64-bit FreeBSD
`x86_64-unknown-fuchsia`	✓			64-bit Fuchsia
`x86_64-unknown-linux-gnux32`	✓			64-bit Linux
`x86_64-unknown-linux-musl`	✓			64-bit Linux with MUSL
`x86_64-unknown-netbsd`	✓	✓	✓	NetBSD/amd64
`x86_64-unknown-redox`	✓			Redox OS

Tier 3

Tier 3 platforms are those which the Rust codebase has support for, but which are not built or tested automatically, and may not work. Official builds are not available.

Target	std	rustc	cargo	notes
`i686-pc-windows-msvc` (XP)	✓			Windows XP support
`i686-unknown-haiku`	✓			32-bit Haiku
`i686-unknown-netbsd`	✓			NetBSD/i386 with SSE2
`le32-unknown-nacl`	✓			PNaCl sandbox
`mips-unknown-linux-uclibc`	✓			MIPS Linux with uClibc
`mipsel-unknown-linux-uclibc`	✓			MIPS (LE) Linux with uClibc
`msp430-none-elf`	*			16-bit MSP430 microcontrollers
`sparc64-unknown-netbsd`	✓	✓		NetBSD/sparc64
`thumbv6m-none-eabi`	*			Bare Cortex-M0, M0+, M1
`thumbv7em-none-eabi`	*			Bare Cortex-M4, M7
`thumbv7em-none-eabihf`	*			Bare Cortex-M4F, M7F, FPU, hardfloat
`thumbv7m-none-eabi`	*			Bare Cortex-M3
`x86_64-pc-windows-msvc` (XP)	✓			Windows XP support
`x86_64-unknown-bitrig`	✓	✓		64-bit Bitrig
`x86_64-unknown-dragonfly`	✓	✓		64-bit DragonFlyBSD
`x86_64-unknown-haiku`	✓			64-bit Haiku
`x86_64-unknown-openbsd`	✓	✓		64-bit OpenBSD
NVPTX (https://github.com/japaric/nvptx#targets)	**			`--emit=asm` generates PTX code that runs on NVIDIA GPUs

* These are bare-metal microcontroller targets that only have access to the core library, not std.

** There's backend support for these targets but no target built into `rustc` (yet). You'll have to write your own target specification file (see the links in the table). These targets only support the core library.

But those aren’t the only platforms Rust can compile to! Those are the ones with built-in target definitions and/or standard library support. When linking only to the core library, Rust can also target “bare metal” in the x86, ARM, MIPS, and PowerPC families, though it may require defining custom target specifications to do so. All such scenarios are tier 3.