Interfaces

Modules and components

WebAssembly binaries may be **components** built according to the Component Model or **modules** built to the core WebAssembly specification.

As you begin writing a Wasm application using WASI APIs, one of your first decisions will be which type of binary you want to produce—a decision typically guided by your use-case and the runtime you wish to use. Check to see which WASI releases your runtime supports.

- **Components** can use WASI 0.2 and the Component Model for composability and interoperability, meaning that a WebAssembly component compiled from one language (Rust, for example) can communicate or be combined with a component compiled from another language (such as Go). WASI 0.2 sets the stage for the future of WASI.

- **Modules** can use APIs from WASI 0.1, an earlier stage of WASI's development. Since WASI 0.2 was released in February 2024, WASI 0.1 support is more widespread among Wasm runtimes, and it is widely used in production today.

WASI 0.2

Presentation

WASI 0.2 is the most recent WASI release. APIs designed for WASI 0.2 and the Component Model are defined with the **WebAssembly Interface Type (WIT)** Interface Description Language (IDL). WIT API definitions are made in `.wit` files which are composed into Wasm component binaries. The following interfaces are included in WASI P2:

API	Repository
Clocks	https://github.com/WebAssembly/wasi-clocks
Random	https://github.com/WebAssembly/wasi-random

API	Repository
Filesystem	https://github.com/WebAssembly/wasi-filesystem
Sockets	https://github.com/WebAssembly/wasi-sockets
CLI	https://github.com/WebAssembly/wasi-cli
HTTP	https://github.com/WebAssembly/wasi-http

You can explore the types and definitions for a given WASI 0.2 API in its WIT files. When you're ready to start using the API, you will typically generate bindings between the WIT definitions and the language you will be compiling to Wasm. For more information on WIT, see the WIT section of the Component Model documentation.

Versions

WASI 0.2 have a few subreleases, that are listed below.

Version	Changelog
0.2.2	This release includes new WIT features for `@deprecated` feature gates and is excersied in the wasi:http/proxy world. For more information, see component-model/WIT.md.
0.2.1	This release includes new WIT features for `@since` and `@unstable` feature gates. For more information, see component-model/WIT.md.
0.2.0	This version officially launched with the vote in the WASI Subgroup January 24th.

WASI 0.1

WASI P1 APIs were defined with WITX Interface Description Language (IDL), which was an iterative step toward WIT but bears notable differences, including that it was developed as a lower-level derivation of WebAssembly Text Format (a human-readable source format for

Wasm modules). Documentation for WASI 0.1 and WITX can be found in the `legacy` directory of the WASI GitHub repository, along with a complete list of 0.1 types and modules.

Proposals for the standard

All WASI APIs are **proposals** for standardization by the WASI Subgroup. The API proposals in WASI 0.1 and 0.2 met implementation and portability criteria for inclusion at the time of those releases. A proposal advances through the following stages as defined in the WASI Subgroup's Phase Process:

- **Phase 0 - Pre-proposal**: The pre-proposal phase serves as a way to share ideas. At this phase, the WASI subgroup has not yet decided that the pre-proposal is in scope for WASI, and there may be overlap between pre-proposals.
- **Phase 1 - Feature proposal**: In this phase, the proposal is added to the proposal list and a new fork of the spec repo is created.
- **Phase 2 - Feature description available**: During this phase, one or more implementations prototype the feature and a test suite is added.
- **Phase 3 - Implementation phase**: At this phase, project champions create releases following the conventions of semantic versioning (semver).
- **Phase 4 - Standardize the feature:** At this point, the feature is fully handed off to the Working Group, where edge cases are considered and only minor changes occur.
- **Phase 5 - The feature is standardized**: Once the Working Group reaches consensus that the feature is complete, editors perform final editorial tweaks and merge the feature into the main branch of the primary spec repo.

Proposals are first made to the **WASI Subgroup** of the WebAssembly Community Group. (See the WASI Subgroup's meeting schedule.)

All active WASI API proposals can be found on the WASI GitHub repository. See the Contributing to WASI page for information about submitting a new proposal.

Active Proposals

Phase 5 - The Feature is Standardized (WG)

API Proposal	Repository

Phase 4 - Standardize the Feature (WG)

API Proposal	Repository

Phase 3 - Implementation Phase (CG + WG)

API Proposal	Repository
I/O	https://github.com/WebAssembly/wasi-io
Clocks	https://github.com/WebAssembly/wasi-clocks
Random	https://github.com/WebAssembly/wasi-random
Filesystem	https://github.com/WebAssembly/wasi-filesystem
Sockets	https://github.com/WebAssembly/wasi-sockets
CLI	https://github.com/WebAssembly/wasi-cli
HTTP	https://github.com/WebAssembly/wasi-http

Phase 2 - Proposed Spec Text Available (CG + WG)

API Proposal	Repository
Machine Learning (wasi-nn)	https://github.com/WebAssembly/wasi-nn
Clocks: Timezone	https://github.com/WebAssembly/wasi-clocks

Phase 1 - Feature Proposal (CG)

API Proposal	Repository
Blob Store	https://github.com/WebAssembly/wasi-blob-store

API Proposal	Repository
Crypto	https://github.com/WebAssembly/wasi-crypto
Digital I/O	https://github.com/WebAssembly/wasi-digital-io
Distributed Lock Service	https://github.com/WebAssembly/wasi-distributed-lock-service
I2C	https://github.com/WebAssembly/wasi-i2c
Key-value Store	https://github.com/WebAssembly/wasi-kv-store
Logging	https://github.com/WebAssembly/wasi-logging
Messaging	https://github.com/WebAssembly/wasi-messaging
Observe	https://github.com/dylibso/wasi-observe
Parallel	https://github.com/WebAssembly/wasi-parallel
Pattern Match	https://github.com/WebAssembly/wasi-pattern-match
Runtime Config	https://github.com/WebAssembly/wasi-runtime-config
SPI	https://github.com/WebAssembly/wasi-spi
SQL	https://github.com/WebAssembly/wasi-sql
SQL Embed	https://github.com/WebAssembly/wasi-sql-embed
Threads	https://github.com/WebAssembly/wasi-native-threads
URL	https://github.com/WebAssembly/wasi-url
USB	https://github.com/WebAssembly/wasi-usb
WebGPU	https://github.com/WebAssembly/wasi-webgpu

Phase 0 - Pre-Proposal (CG)

Proposal	Repository
proxy-wasm/spec (will advance as multiple, smaller proposals)	https://github.com/proxy-wasm/spec

Versioning

Proposals remain in the 0.x semver range until they reach Phase 5 and are fully standardized. At that point, a 1.0 release should be made available.

For some APIs, it makes sense to add new features after the API itself has reached Phase 5. These feature additions should go through the same standardization process. Once they have reached Phase 5, the minor version number of the release should be incremented.

Some APIs may require backwards-incompatible changes over time. In these cases, proposals are allowed to increment the major version number *only if* the old API can be implmented in terms of the new API. As part of the new version, champions are expected to provide a tool that enables this backwards-compatibility. If that is not possible, then a new API proposal with a new name should be started. The original API can then be deprecated over time if it makes sense to do so.

Edit this page