# Crate core

Since 1.6.0 ·

Settings Help Summary

## The Rust Core Library

The Rust Core Library is the dependency-free[1] foundation of The Rust Standard Library. It is the portable glue between the language and its libraries, defining the intrinsic and primitive building blocks of all Rust code. It links to no upstream libraries, no system libraries, and no libc.

The core library is *minimal*: it isn't even aware of heap allocation, nor does it provide concurrency or I/O. These things require platform integration, and this library is platform-agnostic.

## How to use the core library

Please note that all of these details are currently not considered stable.

This library is built on the assumption of a few existing symbols:

- `memcpy`, `memmove`, `memset`, `memcmp`, `bcmp`, `strlen` - These are core memory routines which are generated by Rust codegen backends. Additionally, this library can make explicit calls to `strlen`. Their signatures are the same as found in C, but there are extra assumptions about their semantics: For `memcpy`, `memmove`, `memset`, `memcmp`, and `bcmp`, if the `n` parameter is 0, the function is assumed to not be UB, even if the pointers are NULL or dangling. (Note that making extra assumptions about these functions is common among compilers: clang and GCC do the same.) These functions are often provided by the system libc, but can also be provided by the compiler-builtins crate. Note that the library does not guarantee that it will always make these assumptions, so Rust user code directly calling the C functions should follow the C specification! The advice for Rust user code is to call the functions provided by this library instead (such as `ptr::copy`).
- Panic handler - This function takes one argument, a `&panic::PanicInfo`. It is up to consumers of this core library to define this panic function; it is only required to never return. You should mark your implementation using `#[panic_handler]`.
- `rust_eh_personality` - is used by the failure mechanisms of the compiler. This is often mapped to GCC's personality function, but crates which do not trigger a panic can be assured that this function is never called. The `lang` attribute is called `eh_personality`.

---

1. Strictly speaking, there are some symbols which are needed but they aren't always necessary. ↩

## Primitive Types

| | |
|---|---|
| array | A fixed-size array, denoted `[T; N]`, for the element type, `T`, and the non-negative compile-time constant size, `N`. |
| bool | The boolean type. |
| char | A character type. |
| f32 | A 32-bit floating-point type (specifically, the "binary32" type defined in IEEE 754-2008). |
| f64 | A 64-bit floating-point type (specifically, the "binary64" type defined in IEEE 754-2008). |
| fn | Function pointers, like `fn(usize) -> bool`. |
| i8 | The 8-bit signed integer type. |
| i16 | The 16-bit signed integer type. |
| i32 | The 32-bit signed integer type. |
| i64 | The 64-bit signed integer type. |
| i128 | The 128-bit signed integer type. |
| isize | The pointer-sized signed integer type. |
| pointer | Raw, unsafe pointers, `*const T`, and `*mut T`. |
| reference | References, `&T` and `&mut T`. |
| slice | A dynamically-sized view into a contiguous sequence, `[T]`. |
| str | String slices. |
| tuple | A finite heterogeneous sequence, `(T, U, ..)`. |
| u8 | The 8-bit unsigned integer type. |
| u16 | The 16-bit unsigned integer type. |
| u32 | The 32-bit unsigned integer type. |
| u64 | The 64-bit unsigned integer type. |
| u128 | The 128-bit unsigned integer type. |
| unit | The `()` type, also called "unit". |
| usize | The pointer-sized unsigned integer type. |
| f16 Experimental | A 16-bit floating-point type (specifically, the "binary16" type defined in IEEE 754-2008). |
| f128 Experimental | A 128-bit floating-point type (specifically, the "binary128" type defined in IEEE 754-2008). |
| never Experimental | The `!` type, also called "never". |

## Modules

| | |
|---|---|
| alloc | Memory allocation APIs |
| any | Utilities for dynamic typing or type reflection. |
| arch | SIMD and vendor intrinsics module. |
| array | Utilities for the array primitive type. |

| | |
|---|---|
| ascii | Operations on ASCII strings and characters. |
| borrow | Utilities for working with borrowed data. |
| cell | Shareable mutable containers. |
| char | Utilities for the `char` primitive type. |
| clone | The `Clone` trait for types that cannot be 'implicitly copied'. |
| cmp | Utilities for comparing and ordering values. |
| convert | Traits for conversions between types. |
| default | The `Default` trait for types with a default value. |
| error | Interfaces for working with Errors. |
| f32 | Constants for the `f32` single-precision floating point type. |
| f64 | Constants for the `f64` double-precision floating point type. |
| ffi | Platform-specific types, as defined by C. |
| fmt | Utilities for formatting and printing strings. |
| future | Asynchronous basic functionality. |
| hash | Generic hashing support. |
| hint | Hints to compiler that affects how code should be emitted or optimized. |
| i8 Deprecation planned | Redundant constants module for the `i8` primitive type. |
| i16 Deprecation planned | Redundant constants module for the `i16` primitive type. |
| i32 Deprecation planned | Redundant constants module for the `i32` primitive type. |
| i64 Deprecation planned | Redundant constants module for the `i64` primitive type. |
| i128 Deprecation planned | Redundant constants module for the `i128` primitive type. |
| isize Deprecation planned | Redundant constants module for the `isize` primitive type. |
| iter | Composable external iteration. |
| marker | Primitive traits and types representing basic properties of types. |
| mem | Basic functions for dealing with memory. |
| net | Networking primitives for IP communication. |
| num | Numeric traits and functions for the built-in numeric types. |
| ops | Overloadable operators. |
| option | Optional values. |
| panic | Panic support in the standard library. |
| pin | Types that pin data to a location in memory. |
| prelude | The core prelude |
| primitive | This module reexports the primitive types to allow usage that is not possibly shadowed by other declared types. |
| ptr | Manually manage memory through raw pointers. |
| result | Error handling with the `Result` type. |
| slice | Slice management and manipulation. |
| str | String manipulation. |
| sync | Synchronization primitives |
| task | Types and Traits for working with asynchronous tasks. |
| time | Temporal quantification. |

| | |
|---|---|
| u8 Deprecation planned | Redundant constants module for the `u8` primitive type. |
| u16 Deprecation planned | Redundant constants module for the `u16` primitive type. |
| u32 Deprecation planned | Redundant constants module for the `u32` primitive type. |
| u64 Deprecation planned | Redundant constants module for the `u64` primitive type. |
| u128 Deprecation planned | Redundant constants module for the `u128` primitive type. |
| usize Deprecation planned | Redundant constants module for the `usize` primitive type. |
| assert_matches Experimental | Unstable module containing the unstable `assert_matches` macro. |
| async_iter Experimental | Composable asynchronous iteration. |
| autodiff Experimental | Unstable module containing the unstable `autodiff` macro. |
| bstr Experimental | The `ByteStr` type and trait implementations. |
| contracts Experimental | Unstable module containing the unstable contracts lang items and attribute macros. |
| f16 Experimental | Constants for the `f16` half-precision floating point type. |
| f128 Experimental | Constants for the `f128` quadruple-precision floating point type. |
| intrinsics Experimental | Compiler intrinsics. |
| io Experimental | Traits, helpers, and type definitions for core I/O functionality. |
| panicking Experimental | Panic support for core |
| pat Experimental | Helper module for exporting the `pattern_type` macro |
| random Experimental | Random value generation. |
| range Experimental | Experimental replacement range types |
| simd Experimental | Portable SIMD module. |
| ub_checks Experimental | Provides the `assert_unsafe_precondition` macro as well as some utility functions that cover common preconditions. |
| unicode Experimental | |
| unsafe_binder Experimental | Operators used to turn types into unsafe binders and back. |

## Macros

| | |
|---|---|
| assert | Asserts that a boolean expression is `true` at runtime. |
| assert_eq | Asserts that two expressions are equal to each other (using `PartialEq`). |
| assert_ne | Asserts that two expressions are not equal to each other (using `PartialEq`). |
| cfg | Evaluates boolean combinations of configuration flags at compile-time. |
| column | Expands to the column number at which it was invoked. |
| compile_error | Causes compilation to fail with the given error message when encountered. |
| concat | Concatenates literals into a static string slice. |
| debug_assert | Asserts that a boolean expression is `true` at runtime. |
| debug_assert_eq | Asserts that two expressions are equal to each other. |

| | |
|---|---|
| debug_assert_ne | Asserts that two expressions are not equal to each other. |
| env | Inspects an environment variable at compile time. |
| file | Expands to the file name in which it was invoked. |
| format_args | Constructs parameters for the other string-formatting macros. |
| include | Parses a file as an expression or an item according to the context. |
| include_bytes | Includes a file as a reference to a byte array. |
| include_str | Includes a UTF-8 encoded file as a string. |
| line | Expands to the line number on which it was invoked. |
| matches | Returns whether the given expression matches the provided pattern. |
| module_path | Expands to a string that represents the current module path. |
| option_env | Optionally inspects an environment variable at compile time. |
| panic | Panics the current thread. |
| stringify | Stringifies its arguments. |
| todo | Indicates unfinished code. |
| try Deprecated | Unwraps a result or propagates its error. |
| unimplemented | Indicates unimplemented code by panicking with a message of "not implemented". |
| unreachable | Indicates unreachable code. |
| write | Writes formatted data into a buffer. |
| writeln | Writes formatted data into a buffer, with a newline appended. |
| assert_unsafe_precondition Experimental | Checks that the preconditions of an unsafe function are followed. |
| cfg_match Experimental | A macro for defining `#[cfg]` match-like statements. |
| concat_bytes Experimental | Concatenates literals into a byte slice. |
| concat_idents Experimental | Concatenates identifiers into one identifier. |
| const_format_args Experimental | Same as `format_args`, but can be used in some const contexts. |
| format_args_nl Experimental | Same as `format_args`, but adds a newline in the end. |
| log_syntax Experimental | Prints passed tokens into the standard output. |
| pattern_type Experimental | Creates a pattern type. |
| trace_macros Experimental | Enables or disables tracing functionality used for debugging other macros. |