Rust Package Project Structure

Reusable Cargo crate with proper module organization, docs, and crates.io publishing.

#rust #package #library #crate #cargo

PNG PDF Copy Prompt

Project Directory

```
mypackage/
    Cargo.toml  Crate manifest
    Cargo.lock
    LICENSE
    README.md
    .gitignore
    rust-toolchain.toml  Pin Rust version
    src/
        lib.rs  Crate root, pub…
        core.rs  Main functional…
        types.rs  Public types an…
        error.rs  Error types wit…
        internal/  Private impleme…
            mod.rs
            parser.rs
            utils.rs
    tests/  Integration tes…
        integration_tests.rs
    examples/  Runnable exampl…
        basic.rs  cargo run --exa…
    benches/  Benchmarks
        benchmarks.rs
```

Why This Structure?

Rust's module system enforces visibility at compile time. The `lib.rs` file is your crate root and defines the public API. Private implementation lives in submodules. This structure follows Cargo conventions for `examples/`, `tests/`, and `benches/`.

Key Directories

`src/lib.rs` - Crate root, declares modules and re-exports public API

`src/types.rs` - Public structs, enums, and traits

`src/internal/` - Private modules, not exposed via pub

`examples/` - Runnable examples for documentation

Public API in lib.rs

```
// src/lib.rs
//! MyPackage - A brief description.
//!
//! # Examples
//! ```
//! use mypackage::process;
//! let result = process("input").unwrap();
//! ```

mod core;
mod error;
mod internal;
mod types;

pub use core::process;
pub use error::Error;
pub use types::{Config, Result};
```

Getting Started

1. `cargo new mypackage --lib`
2. Define public API in `src/lib.rs`
3. Add documentation with `///` and `//!`
4. `cargo test`
5. `cargo publish`

When To Use This

- Building a reusable Rust library
- Publishing to crates.io
- Code shared across multiple binaries
- Internal company crates
- SDK for an API

Trade-offs

`Module boilerplate` - mod.rs or module_name.rs for each module

`Visibility rules` - pub(crate) vs pub vs private takes learning

`Semver strictness` - Breaking changes require major version bump

Best Practices

- Re-export public API from lib.rs
- Use `pub(crate)` for internal-only items
- Document all public items with `///`
- Add crate-level docs with `//!` in lib.rs
- Include examples that compile (doc tests)

Naming Conventions

`Crate name` - lowercase with hyphens (my-package)

`Modules` - snake_case (internal/parser.rs)

`Types` - PascalCase (Config, Error, Result)

Testing Strategy

`Unit tests` - In-module #[cfg(test)] blocks

`Integration tests` - tests/ directory tests public API only

`Doc tests` - Examples in /// docs are compiled and run