Rustdoc for everyone!

Jeremiah Peschka
facility9.com
@peschkaj
Slides: https://github.com/pdxrust/resources/

Brief introduction

Awesome facts about me:

- Former sandwich artist
- Recovering system administrator
- Polyglot developer (Rust, C#, Ruby, JavaScript, whatever)
- Some kind of student at PSU
- Licensed ham - KG7TXV

Documentation basics

Why document?

Document for yourself?

Help other people use your code?

Make it easier for people to maintain your code?

Make life easier

“What is this code doing?”

“Why did I write it this way?”

“How can I use libXYZ?”

“Where should I fix bug A in libXYZ?”

How do we document?

What’s a comment?

Just in case you forgot…

Single line comments:

```
// Look at this awesome code!
```

Block comments:

```
/* ZOMG
   I can write lots of stuff
   And keep pressing enter
 */
```

About block comments

Everything has a block comment variant.

All of my examples will be of the single line variety.

Single line comments are what’s used throughout the compiler’s code base and are the most common in the Rust community.

Comments document thoughts

This is helpful. Please use comments.

But comments don’t produce documentation by default.

We need better comments!

Documentation Comments

These make the docs happen!

```
/// It buzzes the fizz
fn fizzbuzz()
```

```
/** It buzzes the fizz */
fn fizzbuzz()
```

AKA outer doc comments

These go outside of the thing you’re documenting.

These are typically used for functions, structs, and anything else inside a library/module.

Module Comments

Modules and libraries also deserve comments.

It’s difficult to put a comment *outside* of a module or library.
That’s like putting a comment nowhere.

```
//! I am a module comment
```

```
/*! I am also a module comment */
```

Only use these for crate and module level documentation.

Module Comments - Example

```
//! # The Rust Standard Library
//!
//! The Rust Standard Library provides
//! the essential runtime functionality
//! for building portable Rust software.
```

Hiding Boilerplate

Some samples need a lot of set up.

It may detract from your documentation, or may be beyond the scope of what a user needs to know.

We can hide code from finished docs by using a # in our source.

Hiding Boilerplate

```
/// Documents a thing
///
/// ```rust
/// # let magic_number = 12;
///
/// println!("{}", magic_number);
/// ```
```

The basics, again

Three kinds of comments.

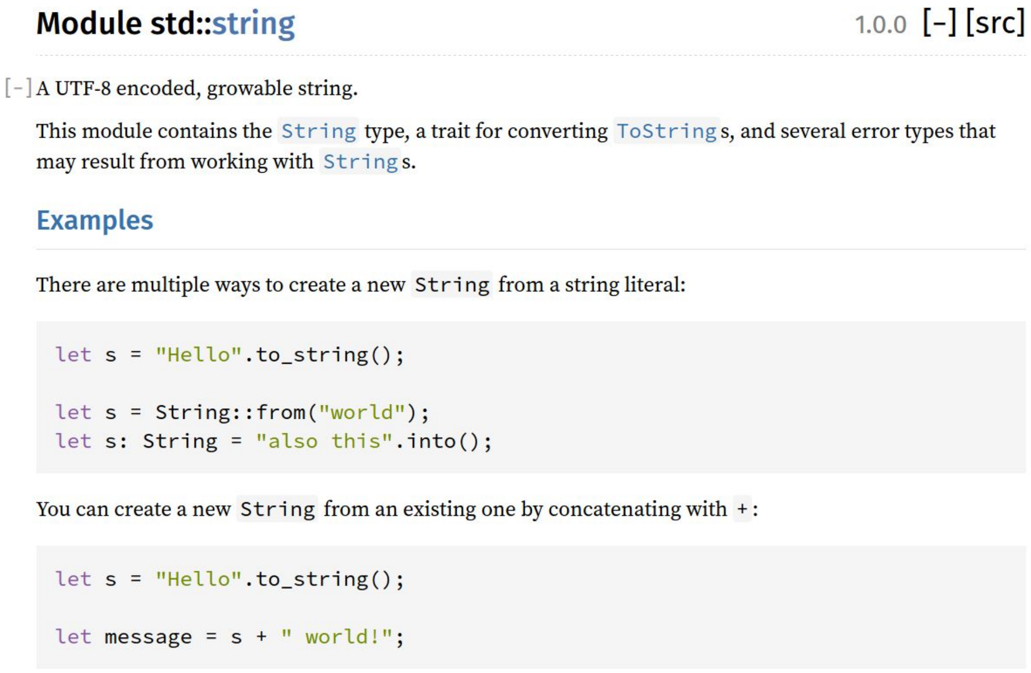
Regular comments using //

Outer doc comments ///
These are for functions, structs, etc

Inner doc comments //!
These go inside the thing you’re documenting:
e.g. modules, crates

Formatting Documentation

Rust’s docs are attractive!



Module std::string

1.0.0 [−] [src]

[−] A UTF-8 encoded, growable string.

This module contains the `String` type, a trait for converting `ToString`s, and several error types that may result from working with `String`s.

Examples

There are multiple ways to create a new `String` from a string literal:

```
let s = "Hello".to_string();

let s = String::from("world");
let s: String = "also this".into();
```

You can create a new `String` from an existing one by concatenating with `+`:

```
let s = "Hello".to_string();

let message = s + " world!";
```

Format with Markdown

It’s simple, easy, and most of us know it.

- Rustdoc currently uses hoedown
- Hoedown uses the CommonMark variant of Markdown

All your favorite Markdown works.

Documentation & Code

We can embed code

Rustdoc assumes code blocks are Rust.

If you need to specify a different language just like you would on GitHub:

~~~
```php
$no_idea->about(php);
```
~~~

Rustdoc tests examples

Rust code in doc comments is executed during `cargo test`.

With this feature, we have to keep our samples up to date or tests will fail.

This only works for library crates.

Skipping tests: bad code

Let’s say we want to provide an example that may not work.

- Bad code
- Logic that requires external resources

~~~
```rust,ignore
lolwut::this_wont_work();
```
~~~

Skipping tests: infinite code

Some code runs forever:
Services/daemons. Infinite loops.

We still want the code to compile during a `rust doc` run.

This guarantees that the example could work.

~~~
```rust,no_run
loop_forever();
```
~~~

Passing tests without panic

Sometimes you need to demonstrate code that panics.

This should be compiled.

And it should panic during cargo test.

~~~
```rust,should_panic
assert!(false);
```
~~~

What Goes In Docs?

Writing documentation is hard

It can be difficult to determine what should go into docs.

Do I need examples or just an explanation?

How should the text be structured?

What sections should I have?

RFC 505

This provides general guidelines for documentation.

Also see RFC 1574
And RFC 1687
(which is still proposed, but whatever)

Overall Structure

Module level documentation should provide a high level summary of the contents of the module.

Each type should provide its own complete documentation.

It’s OK to duplicate information between the two levels.

Formatting

Make the first line in your comment a single sentence summary of your code.

Keep it short - it’s used throughout Rustdoc’s output.

Common Sections

Use a top level markdown heading (#) for these

- Examples
- Panics
- Errors
- Safety
- Aborts
- Undefined Behaviors

Examples

Keep inline examples short and sweet.

If you need to create a complex example, there’s an examples folder just for that!

Public APIs

If someone can call a function, document it.

You should probably provide a simple example for it, too.

Keep The Reader In Mind

Who normally reads docs?

Someone who is:

- New to a library.
- Frustrated and trying to solve a problem.
- Searching for ways to solve complex problems.

Think about the audience

Documentation can meet the needs of all three examples.

- The summary page should be the most basic.
- API documentation can be the most complex.

What should the reader know to work with your library?

- You don’t have to teach them.
- But you can point them in the right direction.

Getting Involved

How can you make docs better?

Write your own docs!

Participate in rust-docs

- We meet every Wednesday in #rust-docs (irc.mozilla.org) at 1PM Pacific
- Join Rust Doc Days!
  There's a call for suggestions at
  https://users.rust-lang.org/t/call-for-proposals-for-next-rust-doc-days-crates/6685

Good docs take time. Start now.

https://github.com/pdxrust/resources