



Products

# Problems with rust lifetime specifier

Log in Sign up

Ask Question

Asked 3 years, 1 month ago

Active 3 years, 1 month ago

Viewed 143 times



3



Consider the following Program:

```
fn func(source: &str, state: & Vec<&str>) {
    println!("{}", source);
    println!("{}", state[0]);
}
fn step<a>(source: &a str, state: &mut Vec<&a str>) {
    state.push(&source[4..10]);
    func(source, state);
    state.push(&source[4..10]);
}
fn main() {
    let source = "abcdefghijklmnopqrstuvwxy";
    {
        let mut state = Vec:::<&str>::new();
        step(source, &mut state);
        step(source, &mut state);
        step(source, &mut state);
    }
}
```

This compiles and runs without errors. I can totally understand why the lifetime specifier 'a' is needed here, because otherwise you could run:

```
fn main() {
    let mut state = Vec:::<&str>::new();
    {
        let source = "abcdefghijklmnopqrstuvwxy";
        step(source, &mut state);
        step(source, &mut state);
        step(source, &mut state);
    }
    println!("{}", state[0]);
}
```

which would result in undefined output. My problem is that I want to pack both arguments of "func" into a structure, but I can't make this work:

```
struct MyStruct<a, b> {
    pub source: &a str,
    pub state: &b mut Vec<&b str>,
}
fn func(arg: MyStruct) {
    println!("{}", arg.source);
    println!("{}", arg.state[0]);
}
fn step<a>(source: &a str,
    state: &mut Vec<&a str>) {
    state.push(&source[4..10]);
    let s = MyStruct {source: source, state: state};
    func(s);
    state.push(&source[4..10]);
}
fn main() {
    let source = "abcdefghijklmnopqrstuvwxy";
    {
        let mut state = Vec:::<&str>::new();
        step(source, &mut state);
        step(source, &mut state);
        step(source, &mut state);
    }
}
```

The code above does not compile with the error message:

```
107|     state: &mut Vec<&a str>) {
    |         ---- consider changing the type of `state` to `&a mut std::vec::Vec<&a str>`
108|     state.push(&source[4..10]);
109|     let s = MyStruct {source: source, state: state};
    |                   ~~~~~ lifetime `a` required
```

If I change the state of state to &a mut Vec<&a str>, the build failes with the following message:

```
117|     step(source, &mut state);
    |         ---- first mutable borrow occurs here
118|     step(source, &mut state);
    |         ~~~~~ second mutable borrow occurs here
119|     step(source, &mut state);
120| }
    | - first borrow ends here
```

Your privacy

By clicking "Accept all cookies", you agree Stack Exchange can store cookies on your device and disclose information in accordance with our [Cookie Policy](#).

Accept all cookies Customize settings

I would be thankful if anyone could tell me how to make this work.

Note: In this example program, "source" is a String with static lifetime. In the "real" program "source" does not have a static lifetime, but the content of a file that the program reads during runtime. "source" lives as long or longer than "state"

[rust](#) [lifetime](#)

[Share](#)  
[Improve this question](#)  
[Follow](#)

asked Nov 24 '18 at 22:00

  
[Volker Weißmann](#)  
544 ● 5 ● 21

[Add a comment](#)

1 Answer

[Active](#) [Oldest](#) [Votes](#)



2



The lifetime situation in your function parameters is this:

```
fn step<'a>(source: &'a str,
           state: &mut Vec<&'a str>)
```

Let's name the reference that doesn't have an explicit name on the lifetime:

```
fn step<'a, 'x>(source: &'a str,
               state: &'x mut Vec<&'a str>)
```

There, now all the references have explicit lifetimes and we can see that the configuration in the struct does not correspond to how they relate to each other in the function parameters.

If your struct should follow the same relations we want this:

```
struct MyStruct<'a: 'x, 'x> {
    pub source: &'a str,
    pub state: &'x mut Vec<&'a str>,
}
```

And since this is a type definition, we need to add that relation that was implicit in the function that 'a: 'x "a outlives 'x"

Keep in mind you also need to scope the borrow that the `s` variable of type `MyStruct` holds -- it borrows the state mutably (mutably means exclusively). So you need to insert scopes around the line with `let s = ..` and `f(s)` so that `s`'s scope ends before the next step call:

```
{
    let s = MyStruct {source: source, state: state};
    func(s);
}
```

[Share](#)  
[Improve this answer](#)  
[Follow](#)

answered Nov 24 '18 at 23:00

  
[bluss](#)  
10.3k ● 40 ● 45



[Add a comment](#)

Your Answer



Post Your Answer

By clicking "Post Your Answer", you agree to our [terms of service](#), [privacy policy](#) and [cookie policy](#)

Not the answer you're looking for? Browse other questions tagged [rust](#) [lifetime](#) or ask your own question.

-  Sequencing your DNA with a USB dongle and open source code
-  Don't push that button: Exploring the software that flies SpaceX rockets and...

#### Featured on Meta

-  Providing a JavaScript API for userscripts
-  Congratulations to the 59 sites that just left Beta

#### Related

[What is the lifetime of a static variable in a C++ function?](#)

[Why doesn't println! work in Rust unit tests?](#)

[Syntax of Rust lifetime specifier](#)

[type parameter for function vs struct \(lifetime issue\)](#)







[Why are explicit lifetimes needed in Rust?](#)

[Borrow errors for multiple borrows](#)

[Getting around Rust ownership problems when using state machine pattern](#)

[Lifetime constraints to model scoped garbage collection](#)

#### Hot Network Questions

-  EU ETS: if within-EU flight emissions are limited to climate goals, is there still a reason not to fly as long as you can afford it?
  -  Who (or what) created the atropal?
  -  Does anyone have a DG Business BASIC manual?
  -  Will these simple 2 convex lens arrangement telescope see the moon clearly?
  -  Have there been no deaths due to omicron in Africa?
- [more hot questions](#)
-  [Question feed](#)

#### STACK OVERFLOW

[Questions](#)  
[Jobs](#)  
[Developer Jobs Directory](#)  
[Salary Calculator](#)  
[Help](#)  
[Mobile](#)

#### PRODUCTS

[Teams](#)  
[Talent](#)  
[Advertising](#)  
[Enterprise](#)

#### COMPANY

[About](#)  
[Press](#)  
[Work Here](#)  
[Legal](#)  
[Privacy Policy](#)  
[Terms of Service](#)  
[Contact Us](#)  
[Cookie Settings](#)  
[Cookie Policy](#)

#### STACK EXCHANGE NETWORK

[Technology](#)  
[Culture & recreation](#)  
[Life & arts](#)  
[Science](#)  
[Professional](#)  
[Business](#)  
[API](#)  
[Data](#)

[Blog](#)  
[Facebook](#)  
[Twitter](#)  
[LinkedIn](#)  
[Instagram](#)