Playing with Proc Macro Magic

Tom Kunc

Derive Macros: Often Used, Very Often Made

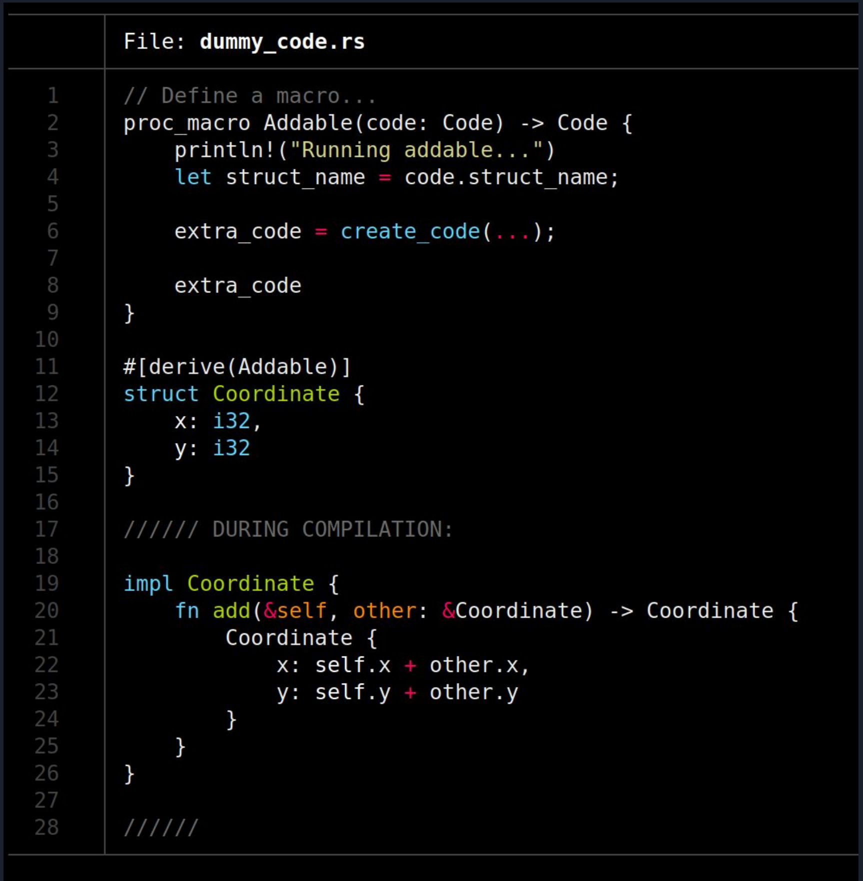
By the end of this talk, you will:

1. Hopefully know how derive macros really work
2. Be able to write your own!

If you have written proc macros before…

1. That’s pretty great, maybe talk about that with other people at the intermission
2. I’ll try to make this relatively interesting for the next 10 minutes

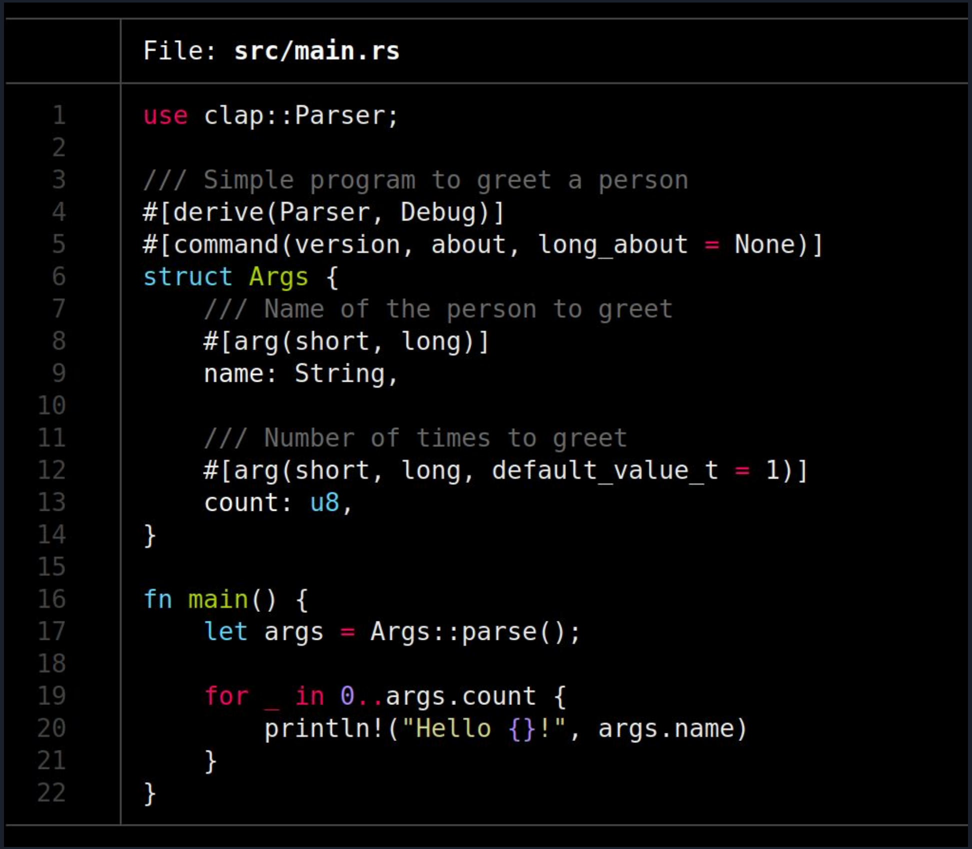
What are Derive Macros?



File: **dummy_code.rs**

```rust
// Define a macro...
proc_macro Addable(code: Code) -> Code {
    println!("Running addable...")
    let struct_name = code.struct_name;

    extra_code = create_code(...);

    extra_code
}

#[derive(Addable)]
struct Coordinate {
    x: i32,
    y: i32
}

////// DURING COMPILATION:

impl Coordinate {
    fn add(&self, other: &Coordinate) -> Coordinate {
        Coordinate {
            x: self.x + other.x,
            y: self.y + other.y
        }
    }
}

//////
```

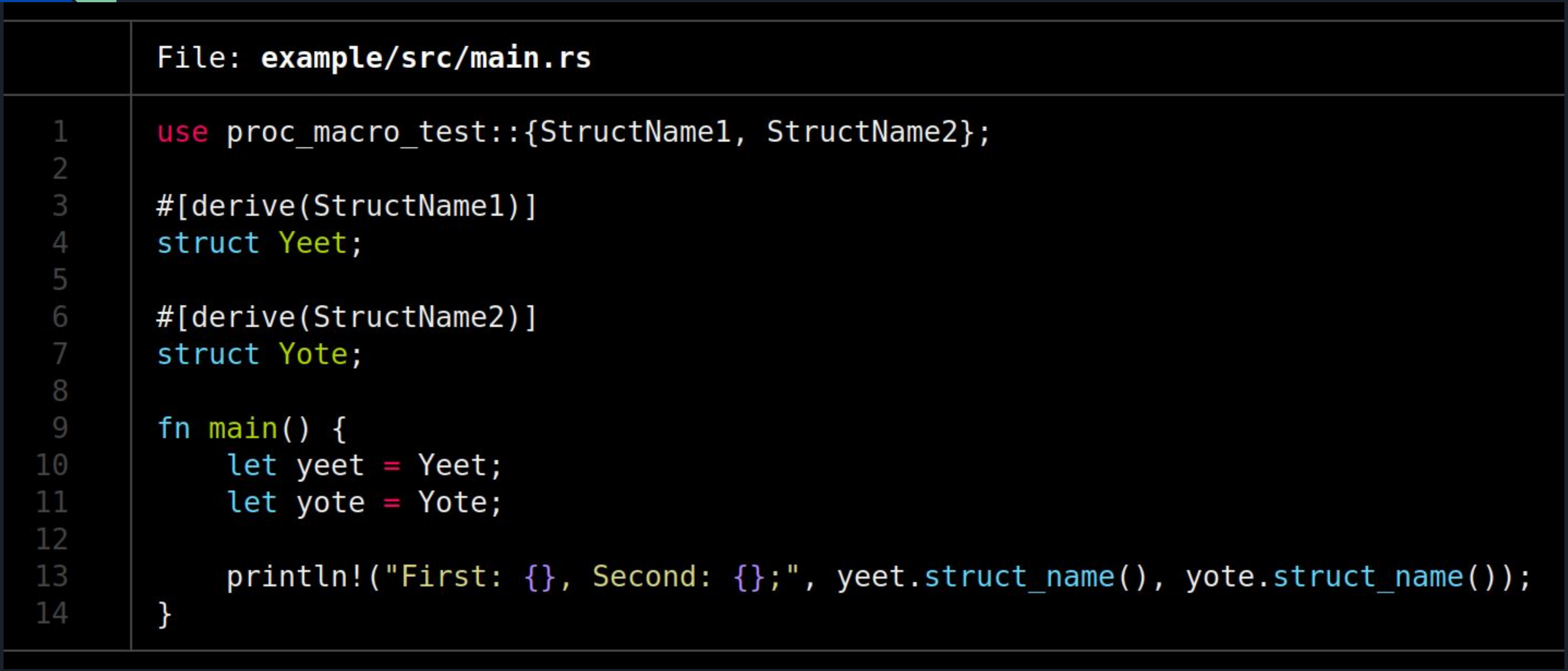
The Power Of Derive Macros



File: **src/main.rs**

```rust
use clap::Parser;

/// Simple program to greet a person
#[derive(Parser, Debug)]
#[command(version, about, long_about = None)]
struct Args {
    /// Name of the person to greet
    #[arg(short, long)]
    name: String,

    /// Number of times to greet
    #[arg(short, long, default_value_t = 1)]
    count: u8,
}

fn main() {
    let args = Args::parse();

    for _ in 0..args.count {
        println!("Hello {}!", args.name)
    }
}
```

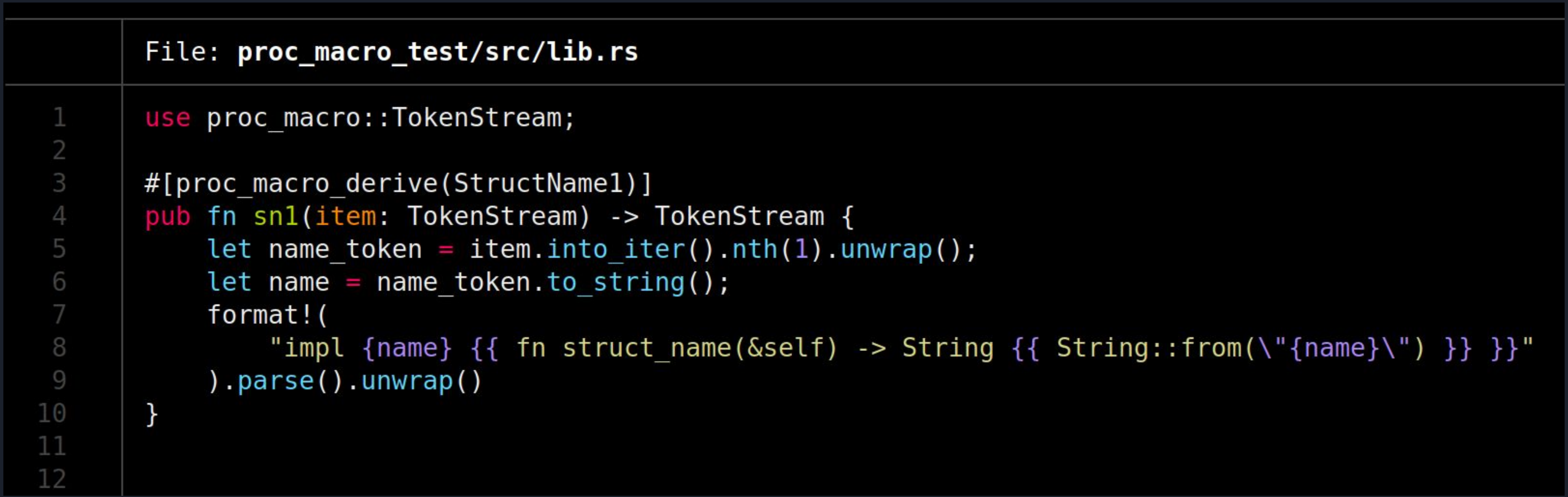
Let’s make one ourselves...



File: **example/src/main.rs**

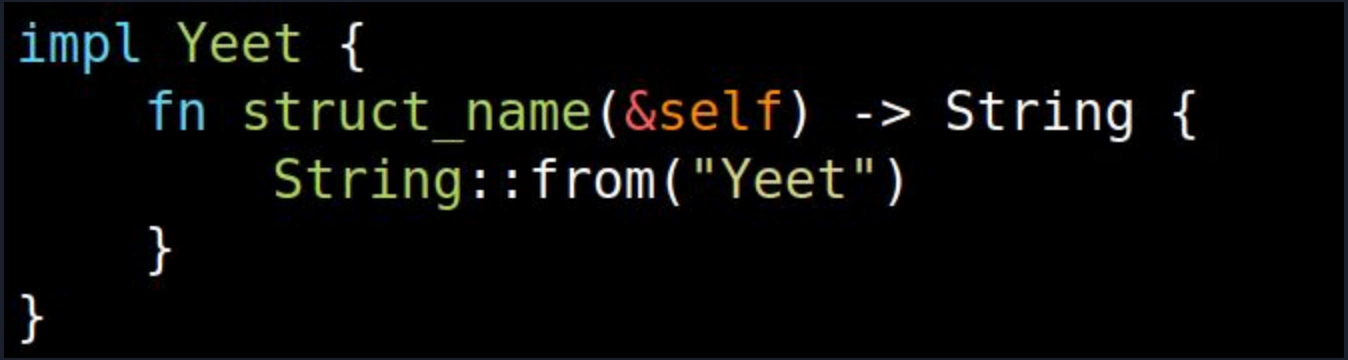
```rust
use proc_macro_test::{StructName1, StructName2};

#[derive(StructName1)]
struct Yeet;

#[derive(StructName2)]
struct Yote;

fn main() {
    let yeet = Yeet;
    let yote = Yote;

    println!("First: {}, Second: {};", yeet.struct_name(), yote.struct_name());
}
```

A simple Macro

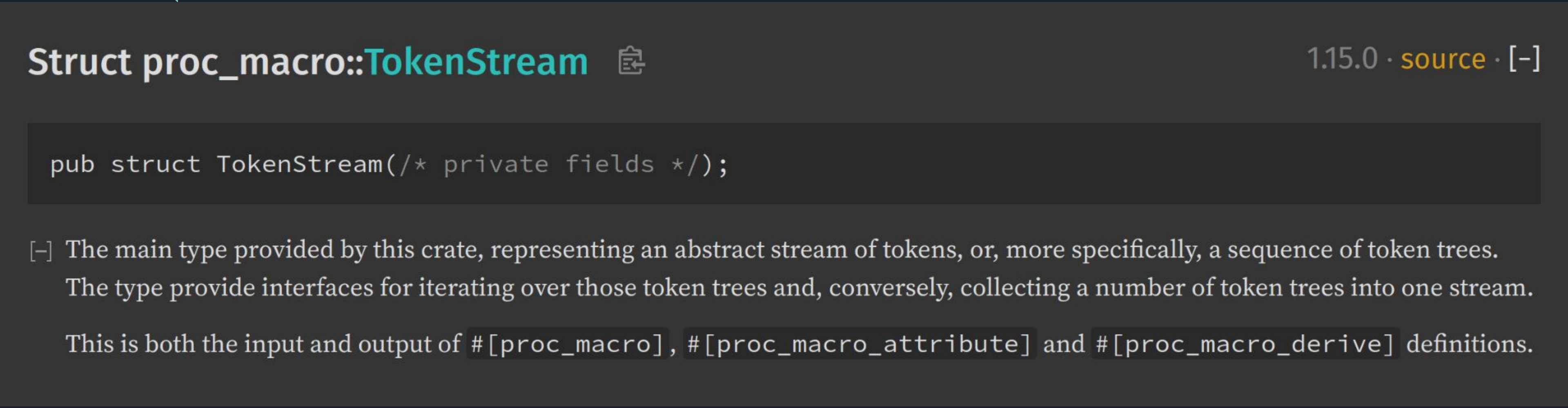


File: **proc_macro_test/src/lib.rs**

```rust
use proc_macro::TokenStream;

#[proc_macro_derive(StructName1)]
pub fn sn1(item: TokenStream) -> TokenStream {
    let name_token = item.into_iter().nth(1).unwrap();
    let name = name_token.to_string();
    format!(
        "impl {name} {{ fn struct_name(&self) -> String {{ String::from(\"{name}\") }} }}"
    ).parse().unwrap()
}


```



```rust
impl Yeet {
    fn struct_name(&self) -> String {
        String::from("Yeet")
    }
}
```



Struct proc_macro::TokenStream

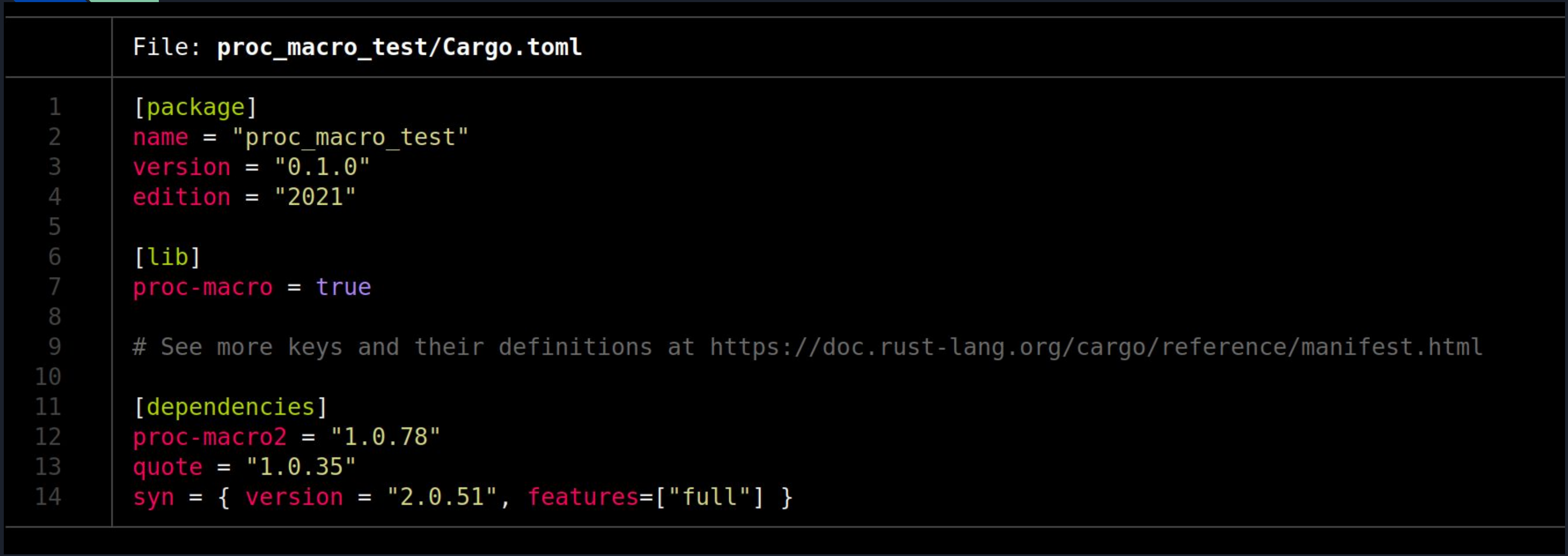
1.15.0 · source · [−]

```
pub struct TokenStream(/* private fields */);
```

[−] The main type provided by this crate, representing an abstract stream of tokens, or, more specifically, a sequence of token trees. The type provide interfaces for iterating over those token trees and, conversely, collecting a number of token trees into one stream.

This is both the input and output of `#[proc_macro]`, `#[proc_macro_attribute]` and `#[proc_macro_derive]` definitions.

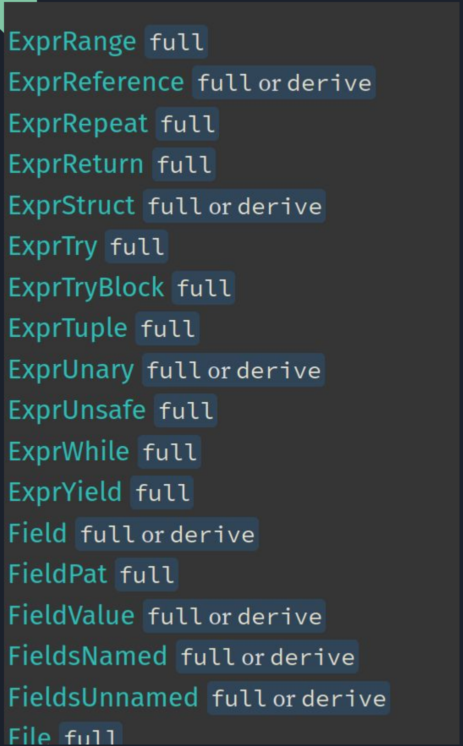
The standard proc-macro library



File: **proc_macro_test/Cargo.toml**

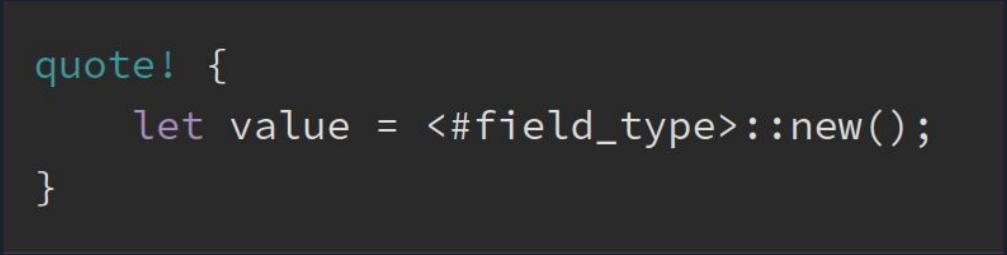
```toml
[package]
name = "proc_macro_test"
version = "0.1.0"
edition = "2021"

[lib]
proc-macro = true

# See more keys and their definitions at https://doc.rust-lang.org/cargo/reference/manifest.html

[dependencies]
proc-macro2 = "1.0.78"
quote = "1.0.35"
syn = { version = "2.0.51", features=["full"] }
```

Syn



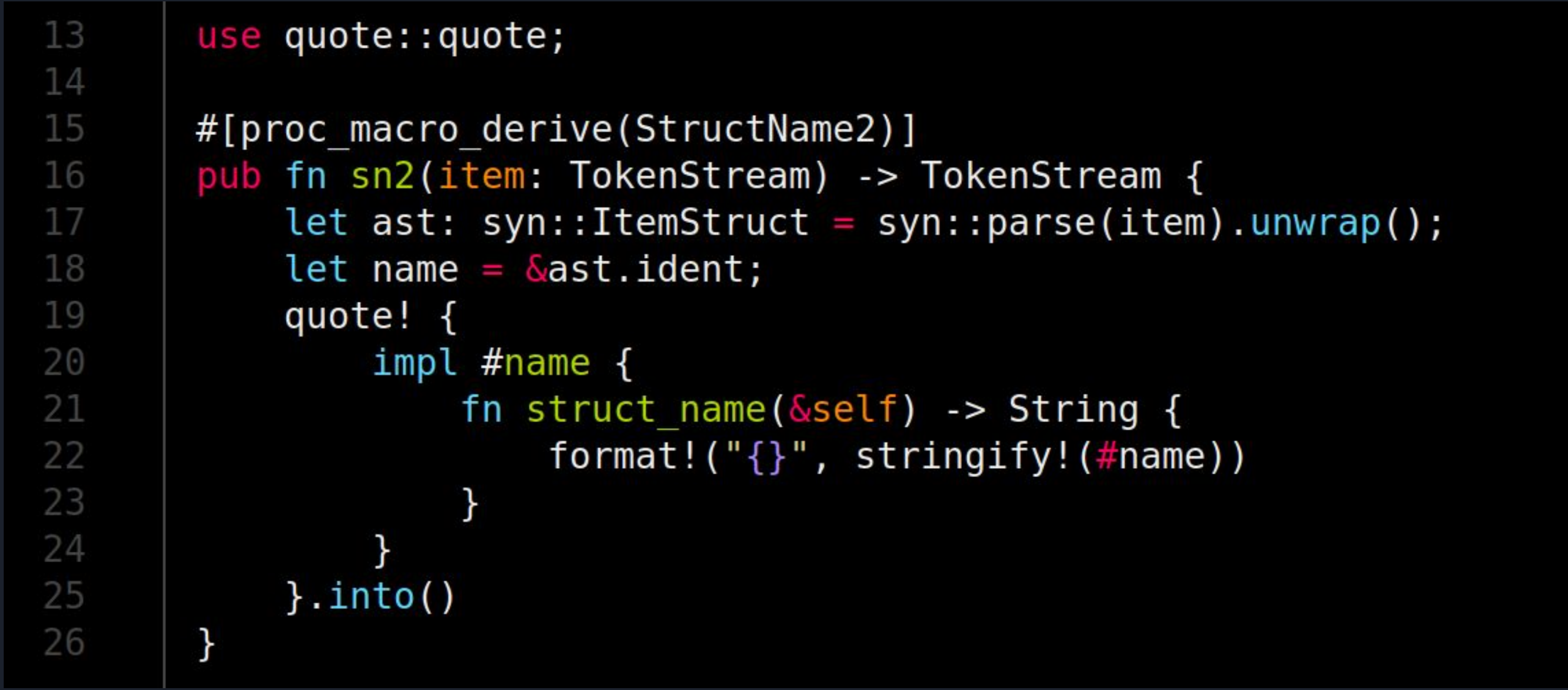
ExprRange	full
ExprReference	full or derive
ExprRepeat	full
ExprReturn	full
ExprStruct	full or derive
ExprTry	full
ExprTryBlock	full
ExprTuple	full
ExprUnary	full or derive
ExprUnsafe	full
ExprWhile	full
ExprYield	full
Field	full or derive
FieldPat	full
FieldValue	full or derive
FieldsNamed	full or derive
FieldsUnnamed	full or derive
File	full

Quote

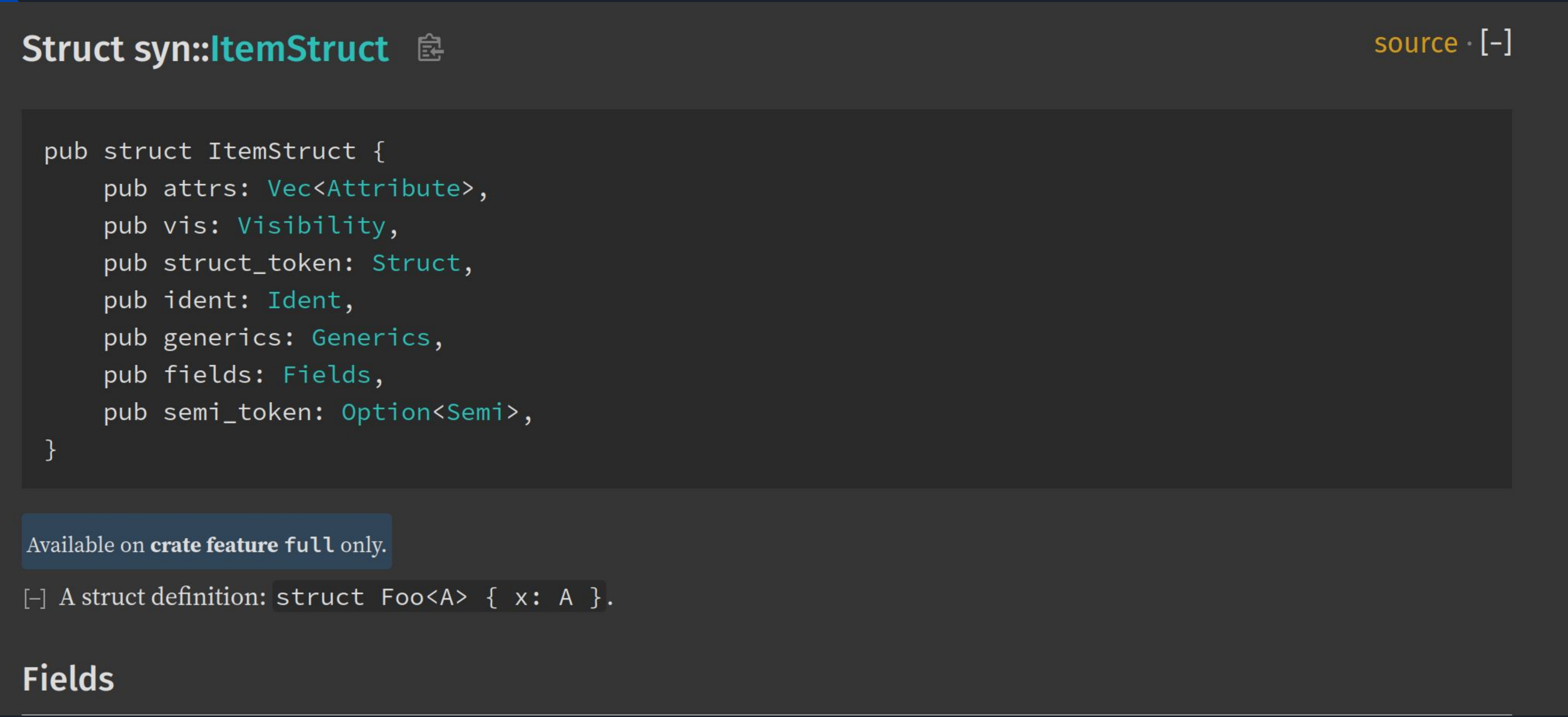


```
quote! {
    let value = <#field_type>::new();
}
```

A less simple Macro



```rust
use quote::quote;

#[proc_macro_derive(StructName2)]
pub fn sn2(item: TokenStream) -> TokenStream {
    let ast: syn::ItemStruct = syn::parse(item).unwrap();
    let name = &ast.ident;
    quote! {
        impl #name {
            fn struct_name(&self) -> String {
                format!("{}", stringify!(#name))
            }
        }
    }.into()
}
```



Struct syn::ItemStruct

```
pub struct ItemStruct {
    pub attrs: Vec<Attribute>,
    pub vis: Visibility,
    pub struct_token: Struct,
    pub ident: Ident,
    pub generics: Generics,
    pub fields: Fields,
    pub semi_token: Option<Semi>,
}
```

Available on **crate feature `full`** only.

[–] A struct definition: `struct Foo<A> { x: A }`.

Fields

Problems & Solutions

- Incredibly easily hackable : https://github.com/dtolnay/watt
- Slow to compile : ^
- Difficult to write : Macro Rules
- Difficult to learn: Hopefully this talk!
- A lack of hygiene : Caution!

Further Reading

- https://github.com/dtolnay/proc-macro-workshop
- https://github.com/tfpk/rustau-talk-2024
-