perplexity

give me a example about creating my own error in rust that derive from std err

To create your own error type in Rust that derives from `std::error::Error`, you can define a custom error type using an enum or a struct and then implement the `std::error::Error` trait for it [1] [2].

Here's an example using an enum:

```
use std::fmt;
use std::error::Error;

#[derive(Debug)]
enum CustomError {
    NoBananas,
    TooManyBananas,
}

impl fmt::Display for CustomError {
    fn fmt(&self, f: &mut fmt::Formatter) -> fmt::Result {
        write!(f, "error: {:?}", self)
    }
}

impl Error for CustomError {}

fn main() {
    fn produce_error() -> Result<(), CustomError> {
        Err(CustomError::NoBananas)
    }

    match produce_error() {
        Err(e) => eprintln!("{}", e),
        _ => println!("No error"),
    }
}
```

In this example:

- A custom error type `CustomError` is defined as an enum with two variants: `NoBananas` and `TooManyBananas` [1].
- The `fmt::Display` trait is implemented to provide a way to display the error message. This is important for presenting nice error messages to the user [1] [2].

- The `Error` trait from the `std::error` module is implemented for `CustomError`. This signifies that `CustomError` is an error type[1].

Here's an example using a struct:

```
use std::fmt;
use std::error::Error;

#[derive(Debug)]
struct AppError {
    code: usize,
    message: String,
}

impl fmt::Display for AppError {
    fn fmt(&self, f: &mut fmt::Formatter) -> fmt::Result {
        let err_msg = match self.code {
            404 => "Sorry, Can not find the Page!",
            _ => "Sorry, something is wrong! Please Try Again!",
        };
        write!(f, "{}", err_msg)
    }
}

impl Error for AppError {}

fn produce_error() -> Result<(), AppError> {
    Err(AppError {
        code: 404,
        message: String::from("Page not found"),
    })
}

fn main() {
    match produce_error() {
        Err(e) => eprintln!("{}", e),
        _ => println!("No error"),
    }
}
```

In this example:

- A custom error type `AppError` is defined as a struct with fields for an error code and a message[3].
- The `fmt::Display` trait is implemented to provide a human-readable error message[1] [3].
- The `Error` trait from the `std::error` module is implemented for `AppError`[1].

You can also implement the `From` trait to convert errors from other types into your custom error type[3]:

```
use std::fs::File;
use std::io;
use std::error::Error;
```

```
use std::fmt;

#[derive(Debug)]
struct AppError {
    kind: String,
    message: String,
}

impl fmt::Display for AppError {
    fn fmt(&self, f: &mut fmt::Formatter) -> fmt::Result {
        write!(f, "{}", self.message)
    }
}

impl Error for AppError {}

impl From<io::Error> for AppError {
    fn from(error: io::Error) -> Self {
        AppError {
            kind: String::from("io"),
            message: error.to_string(),
        }
    }
}

fn main() -> Result<(), AppError> {
    let _file = File::open("nonexistent_file.txt")?;
    Ok(())
}
```

Here, the `From<io::Error>` trait is implemented for `AppError`, allowing easy conversion of `io::Error` instances into `AppError` instances[3].

⁂

1. https://www.halcyon.hr/posts/error-handling-in-rust/
2. https://doc.rust-lang.org/rust-by-example/error/multiple_error_types/define_error_type.html
3. https://learning-rust.github.io/docs/custom-error-types/