

Fundamentals | Custom Errors

■ Custom Error

- ◆ Functions may fail in more than one way
- ◆ Useful to communicate the failure reason
- ◆ Error enumeration
 - Enumerations allow errors to be easily defined
 - Can *match* on the enumeration to handle specific error conditions

■ Error Requirements

- ◆ Implement the Debug trait
 - Displays error info in debug contexts
- ◆ Implement the Display trait
 - Displays error info in user contexts
- ◆ Implement the Error trait
 - Interop with code using dynamic errors

Manual Error Creation

```
#[derive(Debug)]
enum LockError {
    MechanicalError(i32),
    NetworkError,
    NotAuthorized,
}

use std::error::Error;
impl Error for LockError {}
```

Manual Error Creation

```
enum LockError {
    MechanicalError(i32),
    NetworkError,
    NotAuthorized,
}

use std::fmt;
impl fmt::Display for LockError {
    fn fmt(&self, f: &mut fmt::Formatter) -> fmt::Result {
        match self {
            Self::MechanicalError(code) => write!(f, "Mechanical Error: {}", code),
            Self::NetworkError => write!(f, "Network Error"),
            Self::NotAuthorized => write!(f, "Not Authorized"),
        }
    }
}
```

■ The ‘thiserror’ Crate

```
# Cargo.toml  
[dependencies]  
thiserror = "1.0"
```

■ The ‘thiserror’ Crate

```
use thiserror::Error;

#[derive(Debug, Error)]
enum LockError {
    #[error("Mechanical Error: {0}")]
    MechanicalError(i32),
    #[error("Network Error")]
    NetworkError,
    #[error("Not Authorized")]
    NotAuthorized,
}
```

Usage

```
fn lock_door() -> Result<(), LockError> {  
    // ... some code ...  
    Err(LockError::NetworkError)  
}
```

■ Error Conversion

```
use thiserror::Error;

#[derive(Debug, Error)]
enum NetworkError {
    #[error("Connection timed out")]
    Timeout,
    #[error("Unreachable")]
    Unreachable
}

enum LockError {
    #[error("Mechanical Error: {0}")]
    MechanicalError(i32, i32),
    #[error("Network Error")]
    Network(#[from] NetworkError),
    #[error("Not Authorized")]
    NotAuthorized,
}
```

■ Pro Tips: Do's

- ◆ Prefer to use error enumerations over strings
 - More concisely communicates the problem
 - Can be used with *match*
 - Strings are OK when prototyping, or if the problem domain isn't fully understood
 - ▶ Change to enumerations as soon as possible
- ◆ Keep errors specific
 - Limit error enumerations to:
 - ▶ Single modules
 - ▶ Single functions
- ◆ Try to use *match* as much as possible

■ More Pro Tips: Don'ts

- ◆ Don't put unrelated errors into a single enumeration
 - As the problem domain expands, the enumeration will become unwieldy
 - Changes to the enumeration will cascade across the entire codebase
 - Unclear which errors can be generated by a function

Recap

- ◆ Custom error enumerations communicate exactly what went wrong in a function
- ◆ Errors require three trait implementations
 - Debug (can be derived)
 - `std::error::Error` (empty *impl* ok)
 - Display (manual or crate)
- ◆ Use the ***thiserror*** crate to easily implement all required traits for errors
- ◆ Keep error enumerations module or function specific
 - Don't put too many variants in one error