

Crate | serde

■ Serialization / Deserialization

◆ Serialization

- Converts a data structure from memory into a flat representation
 - ▶ Can be saved to disk or transported across the network

◆ Deserialization

- Converts a serialized data structure to an in-memory one
 - ▶ Opposite of serialization

■ **serde**

- ◆ **S**erialization / **D**eserialization library
- ◆ Provides a ***derive*** macro to automatically allow serialization and deserialization
- ◆ `serde` only defines the serialization data *model*
 - Serialization formats must be mapped to the model
 - Formats are implemented in other crates
 - ▶ Multiple formats supported (JSON, Pickle, BSON, YAML, etc)

■ Cargo.toml

```
[dependencies]
serde = { version = "1.0", features = ["derive"] }
serde_json = "1.0"
```

■ Setup

```
use serde::{Deserialize, Serialize};

#[derive(Serialize, Deserialize, Debug)]
struct Form {
    email: String,
    name: String,
    age: usize,
}

let form = Form {
    email: "sample@example.com".to_string(),
    name: "Sample".to_string(),
    age: 25,
};
```

Serialize

```
let serialized = serde_json::to_string(&form)
    .expect("failed to serialize");

println!("{}", serialized);
```

```
{"email":"sample@example.com","name":"Sample","age":25}
```

■ Deserialize

```
let deserialized: Result<Form, _>;  
deserialized = serde_json::from_str(&serialized);  
  
println!("{:?}", deserialized);
```

```
Ok(Form { email: "sample@example.com", name: "Sample", age: 25 })
```

■ Recap

- ◆ Serialization is a way to export a data structure from Rust
 - Deserialization performs the opposite operation
- ◆ **serde** provides serialization & deserialization
 - Multiple formats supported through additional crates
- ◆ Include the *derive* feature to enable the Serialize and Deserialize macros
 - [*dependencies*]
serde = { **version** = "1.0", **features** = ["derive"] }