

# An Experimental iRODS Client in Rust

—

By Phillip Davis

- MsCS, App State '24
- Interned with iRODS  
Summer '22 and '23
- Currently freelancing



1. Overview
2. Quick tour
3. Housekeeping /  
Moving forward

# Why Rust?

- Memory safety ( $\approx$  data safety)
  - Reliable concurrency.
  - Performance.
  - Rich type system.
  - Absurdly powerful macro system.
  - Because I like it!
-

# What's the deal with this client?

- Async everywhere
    - Only runs under Tokio
  - Easy to change protocol encoding.
  - Designed with longer-running connections in mind.
  - Write your rules in Rust.
  - (Probably not cross-platform)
-

```

~ 20 use irods_client::connection::Account;
+ 19 use irods_client::reexports::tokio;
+ 18 use irods_client::{connection::authenticate::NativeAu
17
16 #[rule(
15     body = "writeLine('stdout', '*greeting1, *greetin
14     output = "ExecRuleOut"
13 )]
12 #[derive(Debug)]
11 pub struct VeryAdvancedHelloWorldRule {
10     pub greeting1: String,
9     pub greeting2: String,
8 }
7
6 fn main() {
~ 5     let conn = /* Spin up a connection */
+ 4
+ 3     let rule = VeryAdvancedHelloWorldRule {
+ 2         greeting1: "Hello".to_string(),
+ 1         greeting2: "World".to_string(),
+ 21     };
+ 1
+ 2     rule.execute(&mut conn).unwrap();
+ 3 }

```

```

impl<T, C, A> Manager for IrodsManager<T, C, A>
where
    T: ProtocolEncoding + Send + Sync,
    C: Connect<T> + Send + Sync + 'static,
    C::Transport: Send + Sync + 'static,
    A: Authenticate<T, C::Transport> + Send + Sync + 'static,
{
    type Type = Connection<T, C::Transport>;
    type Error = IrodsError;

    async fn create(&self) -> Result<Self::Type, Self::Error> {
        self.connector
            .connect(self.account.clone())
            .and_then(|unauth_conn| self.authenticator.authenticate(unauth_conn))
            .await
    }

    async fn recycle(
        &self,
        conn: &mut Self::Type,
        metrics: &deadpool::managed::Metrics,
    ) -> RecycleResult<Self::Error> {
        Ok(())
    }
}

```

```
3 pub(crate) async fn read_standard_header<T>(&mut self) -> Result<StandardHeader, IrodsError>
4 where
5     T: ProtocolEncoding,
6 {
7     self.read_to_header_buf(4).await?;
8     let header_len = u32::from_be_bytes(self.header_buf[..4].try_into().unwrap()) as usize;
9     self.read_to_header_buf(header_len).await?;
10
11     Ok(T::decode(&self.header_buf[..header_len])?)
12 }
13
14 pub(crate) async fn read_msg<T, M>(&mut self, len: usize) -> Result<M, IrodsError>
15 where
16     T: ProtocolEncoding,
17     M: Deserializable,
18 {
19     self.read_to_msg_buf(len).await?;
20     Ok(T::decode(&self.msg_buf[..len])?)
21 }
```

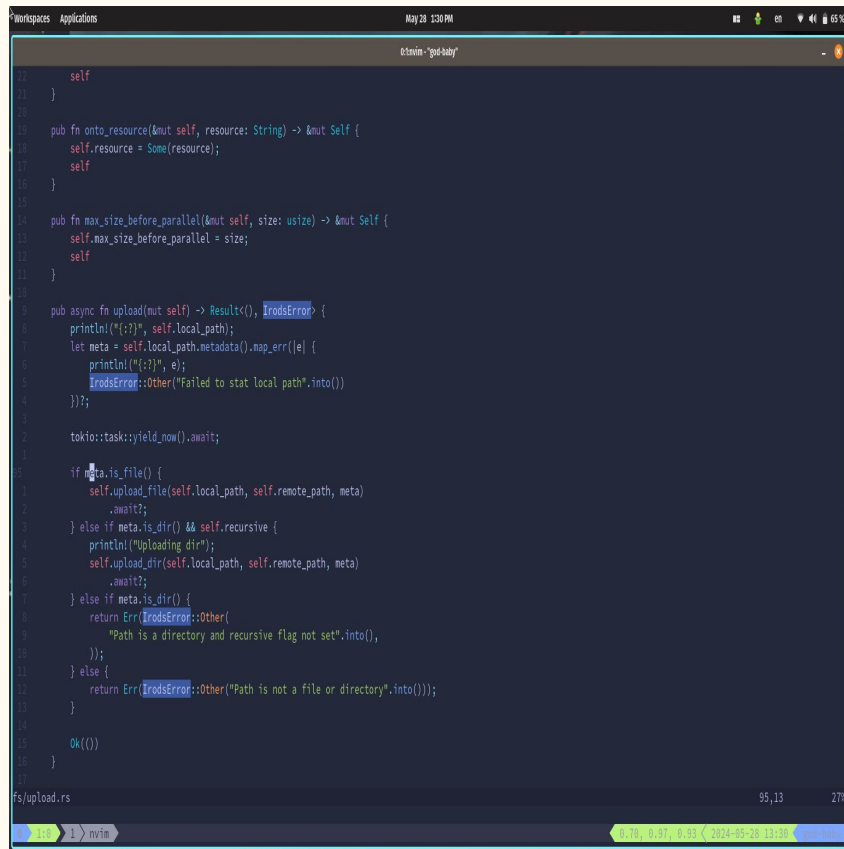
# Caveats / Future Directions





# The Bad: or, The Path to Production-Readiness

1. Well-defined error-handling strategy.
2. Comprehensive integration tests.
3. PR in flux with quick-xml.
4. More features (e.g., password-change algorithm)
5. Etc., etc.



```
22     self
21   }
20
19   pub fn onto_resource(&mut self, resource: String) -> &mut Self {
18     self.resource = Some(resource);
17     self
16   }
15
14   pub fn max_size_before_parallel(&mut self, size: usize) -> &mut Self {
13     self.max_size_before_parallel = size;
12     self
11   }
10
9   pub async fn upload(&mut self) -> Result<(), IrodsError> {
8     println!("{}", self.local_path);
7     let meta = self.local_path.metadata().map_err(|e| {
6       println!("{}", e);
5       IrodsError::Other("Failed to stat local path".into())
4     });
3
2     tokio::task::yield_now().await;
1
0   if meta.is_file() {
1     self.upload_file(self.local_path, self.remote_path, meta)
2       .await;
3   } else if meta.is_dir() && self.recursive {
4     println!("Uploading dir");
5     self.upload_dir(self.local_path, self.remote_path, meta)
6       .await;
7   } else if meta.is_dir() {
8     return Err(IrodsError::Other(
9       "Path is a directory and recursive flag not set".into(),
10    ));
11   } else {
12     return Err(IrodsError::Other("Path is not a file or directory".into()));
13   }
14   }
15   Ok(())
16 }
17 }
```

fs/upload.rs 95,13 274

# The Good: or, Nice-to-haves

1. “Statically” checked rule syntax (via compile-time execution)
2. Statically sized connection buffers might allow arena allocation (inspired by `battlesnake_game_types` crate)

```
33 mod victor_determinable;
34 mod you_determinable;
35
36 pub use eval::EvaluateMode;
37
38 /// A compact board representation that is significantly faster for simulation than
39 /// `battlesnake_game_types::wire_representation::Game`.
40 #[derive(Debug, Copy, Clone, PartialEq, Eq, Hash)]
41 pub struct CellBoard<
42     T: CN,
43     DimensionsType: Dimensions,
44     const BOARD_SIZE: usize,
45     const MAX_SNAKES: usize,
46 > {
47     hazard_damage: u8,
48     cells: [Cell<T>; BOARD_SIZE],
49     healths: [u8; MAX_SNAKES],
50     heads: [CellIndex<T>; MAX_SNAKES],
51     lengths: [u16; MAX_SNAKES],
52     dimensions: DimensionsType,
53 }
54
55 #[allow(dead_code)]
56 fn get_snake_id(
57     snake: &crate::wire_representation::BattleSnake,
```

# Join me!

- Recently open-sourced the project. It lives here:

<https://tinyurl.com/irods-rust-github>

- Issue #7 is a (subject-to-change) list of issues that are important to resolve before issuing an initial release.

- Anyone who wants an iRODS Rust client is welcome to contribute. Right now it's just me.

Thank you.