TxPipe Docs

Rust

To integrate with Rust you need to make sure that your *trix* project has the `rust` bindings generation enabled in the config:

```
[[bindings]]
plugin = "rust"
output_dir = "./gen/rust"
```

With the binding generation enabled, run the following command from your CLI to trigger the code generator:

```
trix bindgen
```

If things went well, you should see the following new files:

```
my-protocol/
└── gen/
    └── rust/
        ├── lib.rs
        └── Cargo.toml
```

That new `lib.rs` file has the required types and functions to call your protocol from a Rust

code.

The `Cargo.toml` file is a standard file for Rust projects.

Now, we can use the generated bindings to build a transaction using our protocol.

First access to the `gen/rust` folder

```
cd gen/rust
```

We need to open `lib.rs` and update the TRP endpoint to point to the correct URL (unless you have it configured on trix.toml). The default value is `http://localhost:3000/trp`, but if you're using different port or endpoint, make sure to update it.

For the following example, we will use the main.tx3 file generated by trix.

Now, we can create a new file called `src/bin/test.rs` in the same folder (or any other folder) and add the following code:

```
use tx3_sdk::trp::args::ArgValue;

#[tokio::main]
async fn main() {
    let params = trix_example::TransferParams {
        sender:
ArgValue::from("addr_test1vpgcjapuwl7gfnzhzg6svtj0ph3gxu8kyuadudmf0kzsksqrfu
        receiver:
ArgValue::from("addr_test1vpry6n9s987fpnmjqcqt9un35t2rx5t66v4x06k9awdm5hqpma
        quantity: ArgValue::from(100000000),
    };

    match trix_example::PROTOCOL.transfer_tx(params).await {
        Ok(cbor) => {
            println!("{:?}", cbor);
        },
        Err(error) => {
            eprintln!("Error: {:?}", error);
        }
    }
}
```

NOTE: Replace `trix_example` with the name of your protocol.

To your Cargo.toml file, add the following dependencies:

```
tokio = { version = "1", features = ["full"] }
```

Finally, we can run the test file to build a transaction using our protocol:

```

cargo run --bin test
```

If everything went well, you should see a message like this:

```
TxEnvelope { tx:
"84a400d90102818258207e5d956d318264043baf8031e250c14b8703b69947ab2d3212d67
}
```