TxPipe Docs

Transactions

Transactions are the core of blockchain interactions, and in Tx3, they're defined as templates that generate complete, valid blockchain transactions based on parameters.

What Makes Tx3 Different

Unlike most blockchain development approaches where transactions are constructed programmatically through multiple API calls, Tx3 uses a declarative approach where:

- The entire transaction structure is visible in one place
- Input selection is constraint-based rather than explicit
- Transaction balancing is handled automatically
- Asset operations are type-checked at compile time
- Parameters provide customization without sacrificing readability

Overview

Transaction templates are the core building blocks of Tx3 programs. They define patterns for constructing valid transactions by specifying:

- Required inputs
- Expected outputs
- Validation conditions
- Parameter bindings

Transaction Template Structure

A Tx3 transaction template has this general form

```
tx name(param1: Type1, param2: Type2) {
    // Optional reference block
    reference name {...}

    // Input blocks
    input name { ... }
    input name2* { ... }

    // Optional collateral block
    collateral { ... }

    // Output blocks
    output { ... }

    // Optional mint/burn blocks
    mint { ... }
    burn { ... }

    // Optional chain-specific blocks
    cardano::stake_delegation { ... }
}
```

Reference block

Reference blocks define what is known as the *reference inputs* of a transaction. This means we can reference a set of UTxOs that will be read by a validator without consuming it.

```
reference name  {
  ref: DataExpr // Required: Reference to the specific UTxO
}
```

Input Blocks

Input blocks define the UTxOs that must be consumed by the transaction.

```
input name {
    from: Party,           // Required: who owns the UTxO
    min_amount: AssetExpr, // Optional: minimum value required
    datum_is: Type,        // Optional: required datum type
    ref: DataExpr,         // Optional: reference to specific UTxO
    redeemer: DataExpr,    // Optional: redeemer data
}
```

The `from`, `min_amount`, `datum_is` and `ref` serve as criteria for selecting inputs during the resolution phase.

The `redeemer` field is a data expressions that will be passed to the validator that controls the consumption of this particular input.

Examples

```
// Basic input
input source {
    from: Sender,
    min_amount: Ada(1000000),
}

// Input with datum
input locked {
    from: TimeLock,
    datum_is: State,
    redeemer: UnlockData { timestamp },
}

// Specific UTxO
input specific {
    ref: 0xABCDEF1234#0,
    from: Owner,
}
```

Output Blocks

Output blocks define the UTxOs that will be created:

```
output name? {
    to: DataExpr,      // Required: recipient address
    amount: AssetExpr, // Required: value to send
    datum: DataExpr,   // Optional: datum to attach
}
```

Examples

```
// Basic output
output {
    to: Receiver,
    amount: Ada(1000000),
}

// Named output with datum
output locked {
    to: TimeLock,
    amount: Ada(1000000),
    datum: State {
        lock_until: until,
        owner: Owner,
        beneficiary: Beneficiary,
    }
}
```

Mint/Burn Blocks

Mint and burn blocks define token operations:

```
mint {
    amount: AssetExpr,  // Required: amount to mint
    redeemer: DataExpr, // Required: minting policy data
}

burn {
    amount: AssetExpr,  // Required: amount to burn
    redeemer: DataExpr, // Required: burning policy data
}
```

Examples

```
// Minting tokens
mint {
    amount: MyToken(100),
    redeemer: MintData { quantity: 100 },
}

// Burning tokens
burn {
    amount: MyToken(50),
    redeemer: BurnData { quantity: 50 },
}
```

Chain-Specific Blocks

Chain-specific blocks allow for blockchain-specific features:

```
cardano {
    // Cardano-specific fields
    collateral: InputBlock,
    certificates: [Certificate],
    withdrawals: [(StakeCredential, Int)],
}
```

Parameter Binding

Templates can take parameters that are bound at runtime:

```
tx transfer(
    amount: Int,
    recipient: Bytes,
    message: String
) {
    input source {
        from: Sender,
        min_amount: Ada(amount),
    }

    output {
        to: recipient,
        amount: Ada(amount),
        datum: Message { text: message },
    }
}
```

Common Patterns

Simple Transfer

```
tx transfer(amount: Int) {
    input source {
        from: Sender,
        min_amount: Ada(amount) + fees,
    }

    output {
        to: Receiver,
        amount: Ada(amount),
    }

    output {
        to: Sender,
        amount: source - Ada(amount) - fees,
    }
}
```

Time-Locked Transaction

```
tx lock(until: Int) {
    input source {
        from: Owner,
        min_amount: Ada(amount) + fees,
    }

    output locked {
        to: TimeLock,
        amount: Ada(amount),
        datum: State {
            lock_until: until,
            owner: Owner,
            beneficiary: Beneficiary,
        }
    }

    output {
        to: Owner,
        amount: source - Ada(amount) - fees,
    }
}
```

Multi-Asset Transfer