TxPipe Docs

Data Expressions

This guide covers how to work with expressions in Tx3.

Overview

Expressions in Tx3 are used to:

- Compute values
- Access properties
- Perform operations
- Construct data

Data Expressions

Literals

```
// Integer literals
123
-456
0

// Boolean literals
true
false

// String literals
"hello"
"world"

// Bytes literals
0xDEADBEEF
0x1234
```

Constructors

```
// Record construction
State {
    field1: value1,
    field2: value2,
}

// Variant construction
Result::Success(42)
Result::Error("failed")
```

Binary Operations

```
// Arithmetic
a + b
a - b

// Comparison
a == b
a != b
a < b
a > b
a <= b
a >= b

// Logical
a && b
a || b
!a
```

Property Access

```
// Record field access
record.field

// Variant field access
variant.field
```

Asset Expressions

Asset Constructors

```
// ADA constructor
Ada(1000000)

// Custom asset constructor
MyToken(100)

// NFT constructor
AnyAsset(policy_id, asset_name, 1)
```

Asset Operations

```
// Addition
asset1 + asset2

// Subtraction
asset1 - asset2

// Property access
asset.amount
```

Common Patterns

Value Computation

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

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

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

State Updates

```
tx update_state(new_value: Int) {
    input current {
        from: Contract,
        datum_is: State,
    }

    output {
        to: Contract,
        amount: current.amount,
        datum: State {
            version: current.version + 1,
            value: new_value,
            timestamp: current.timestamp,
        }
    }
}
```

Conditional Logic

```
tx conditional_transfer(
    amount: Int,
    should_lock: Bool
) {
    input source {
        from: Sender,
        min_amount: Ada(amount),
    }

    output {
        to: should_lock ? TimeLock : Receiver,
        amount: Ada(amount),
        datum: should_lock ? LockData { amount } : None,
    }
}
```

Expression Evaluation

Order of Operations

1. Parentheses
2. Property access
3. Unary operations
4. Binary operations
5. Constructors

Examples

```
// Complex expression
(a + b) * (c - d)

// Property access with operation
record.field + value

// Nested construction
State {
    value: (a + b) * c,
    timestamp: current.timestamp + 1,
}
```

Best Practices

1. **Expression Clarity**
    - Use parentheses for clarity
    - Break complex expressions
    - Document assumptions
2. **Type Safety**
    - Check operand types
    - Handle edge cases
    - Validate results
3. **Performance**
    - Minimize computation
    - Cache repeated values
    - Optimize expressions
4. **Error Prevention**

- Check for null/undefined
- Validate ranges
- Handle edge cases

Common Use Cases

Fee Calculation

```
tx transfer_with_fee(
    amount: Int,
    fee_rate: Int
) {
    input source {
        from: Sender,
        min_amount: Ada(amount + (amount * fee_rate / 100)),
    }

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

    output {
        to: FeeCollector,
        amount: Ada(amount * fee_rate / 100),
    }
}
```

State Transitions

```
tx transition(
    new_state: State
) {
    input current {
        from: Contract,
        datum_is: State,
    }

    output {
        to: Contract,
        amount: current.amount,
        datum: State {
            version: current.version + 1,
            state: new_state,
            timestamp: current.timestamp + 1,
        }
    }
}
```

Asset Management

```
tx manage_assets(
    amounts: [Int]
) {
    input source {
        from: Manager,
        min_amount: amounts.fold(Ada(0), |acc, x| acc + Ada(x)),
    }

    output {
        to: Pool,
        amount: amounts.fold(Ada(0), |acc, x| acc + Ada(x)),
    }
}
```