TxPipe Docs

Types

This guide covers the data types available in Tx3 and how to use them.

Built-in Types

Integer (`Int`)

```
// Integer literals
123
-456
0
```

- Signed integer type
- Used for quantities, timestamps, etc.
- Supports basic arithmetic operations

Boolean (`Bool`)

```
// Boolean literals
true
false
```

- Logical values
- Used in conditions and flags
- Supports logical operations

Bytes (`Bytes`)

```
// Hex string literals
0x"deadbeef"
0x"1234"
```

- Raw byte data
- Used for addresses, hashes, etc.
- Represented as hex strings

String (`String`)

```
// String literals
"hello"
"world"
```

- Text data
- Used for names, messages, etc.
- UTF-8 encoded

Asset Types

Native Asset (Ada)

```
// ADA literals
Ada(1)  // 1 ADA
Lovelace(500000)   // 0.5 ADA
```

- Native blockchain currency
- Basic unit operations

Custom Assets

```
// Asset definition
asset MyToken = 0xABCDEF123."MYTOKEN";

// Asset literals
MyToken(100)
```

- User-defined tokens
- Requires policy ID and name
- Supports basic arithmetic

Custom Types

Records

```
// Record definition
type State {
    lock_until: Int,
    owner: Bytes,
    beneficiary: Bytes,
}

// Record construction
State {
    lock_until: 1234567890,
    owner: 0x"deadbeef",
    beneficiary: 0x"12345678",
}
```

- Named field collections
- Used for structured data
- Field access via dot notation

Variants

```
// Variant definition
type Result {
    Success: Int,
    Error: String,
}

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

- Tagged unions

- Used for alternative values
- Pattern matching support

Type Usage

In Parameters

```
tx transfer(
    amount: Int,
    recipient: Bytes,
    message: String
) {
    // ... transaction body
}
```

In Data Expressions

```
// Record construction
State {
    lock_until: 1234567890,
    owner: sender,
    beneficiary: receiver,
}

// Variant construction
Result::Success(42)
```

In Asset Expressions

```
// ADA amount
Ada(1000000)

// Custom token amount
MyToken(100)
```

Type Safety

Tx3 provides several type safety features:

1. **Static Type Checking**
   - Types are checked at compile time
   - No runtime type errors
   - Clear error messages
2. **Type Inference**
   - Types can be inferred from context
   - Different semantics for Data Expressions and Asset Expressions
   - Reduces type annotations
   - Maintains type safety
3. **Type Constraints**
   - Custom validation rules
   - Range checks
   - Format validation