Table of Contents

bytes

This package provides utilities for working with sequences of bytes, offering bo
th mutable (Bytes) and immutable (View) representations.

Creating Bytes

You can create Bytes from various sources including arrays, fixed arrays, and it
erators:

```

  test "bytes creation" {

    let arr = [b'h', b'e', b'l', b'l', b'o']
    let bytes1 = @bytes.from_array(arr)
    inspect(
      bytes1,
      content=(
        #|b"\x68\x65\x6c\x6c\x6f"
      ),
    )


    let fixed = FixedArray::make(3, b'a')
    let bytes2 = @bytes.of(fixed)
    inspect(
      bytes2,
      content=(
        #|b"\x61\x61\x61"
      ),
    )


    let empty = @bytes.default()
    inspect(
      empty,
      content=(
        #|b""
      ),
    )


    let iter_bytes = @bytes.from_iter(arr.iter())
    inspect(
      iter_bytes,
      content=(
        #|b"\x68\x65\x6c\x6c\x6f"
      ),
    )
  }
```

Converting Between Formats

Bytes can be converted to and from different formats:

```
1
2   test "bytes conversion" {
3     let original = [b'x', b'y', b'z']
4     let bytes = @bytes.from_array(original)
5
6
7     let array = bytes.to_array()
8     inspect(array, content="[b'\\x78', b'\\x79', b'\\x7A']")
9
10
11    let fixed = bytes.to_fixedarray()
12    inspect(fixed, content="[b'\\x78', b'\\x79', b'\\x7A']")
13
14
15    let collected = bytes.iter().to_array()
16    inspect(collected, content="[b'\\x78', b'\\x79', b'\\x7A']")
17  }
```

Working with Views

Views provide a way to work with portions of bytes and interpret them as various numeric types:

```
1
2   test "bytes view operations" {
3
4     let num_bytes = @bytes.from_array([0x12, 0x34, 0x56, 0x78])
5
6
7     let view = num_bytes[:]
8
9
10    inspect(view[0], content="b'\\x12'")
11
12
13    inspect(view.to_int_be(), content="305419896")
14
15
16    inspect(view.to_int_le(), content="2018915346")
17
18
19    let sub_view = view[1:3]
20    inspect(sub_view.length(), content="2")
21  }
```

Binary Data Interpretation

Views provide methods to interpret byte sequences as various numeric types in bo
th little-endian and big-endian formats:

```

  test "numeric interpretation" {

    let int64_bytes = @bytes.from_array([
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x42,
    ])
    let int64_view = int64_bytes[:]
    inspect(int64_view.to_int64_be(), content="66")
    inspect(int64_view.to_uint64_le(), content="4755801206503243776")
  }
```

Concatenation and Comparison

Bytes can be concatenated and compared:

```

  test "bytes operations" {
    let b1 = @bytes.from_array([b'a', b'b'])
    let b2 = @bytes.from_array([b'c', b'd'])


    let combined = b1 + b2
    inspect(
      combined,
      content=(
        #|b"\x61\x62\x63\x64"
      ),
    )


    let same = @bytes.from_array([b'a', b'b'])
    let different = @bytes.from_array([b'x', b'y'])
    inspect(b1 == same, content="true")
    inspect(b1 == different, content="false")
    inspect(b1 < b2, content="true")
  }
```