# BUFFERS

# WHAT ARE BUFFERS?

Buffers are node's way of dealing with binary data
They are essentially views to an allocated slab of memory in c++

# WHY DO THEY EXIST?

Initially node used UTF-8 encoded strings to represent binary
This was not performant

# CREATING BUFFERS

`Buffer` is a global construct

**01_create.js**

```
var buffer1 = new Buffer(1024);

console.log(buffer1.length);
console.log(buffer1);
```

# FILL

## 02_fill.js

```
var buffer = new Buffer(8);
console.log(buffer.length, buffer);

buffer.fill(0);
console.log(buffer.length, buffer);
```

# MORE WAYS TO CREATE BUFFERS

## 03_create_options.js

```
var array = ["a", 0xBA, 0xDF, 0x00, 0xD0, 255, 10];

var buffer1 = new Buffer(array);
console.log('buffer1', buffer1.length, buffer1);

// other encodings: ascii, utf16le, ucs2, base64, binary, hex
var buffer2 = new Buffer('hello world', 'utf8');
console.log('buffer2', buffer2.length, buffer2.toString());

var buffer3 = new Buffer('68656c6c6f20776f726c64', 'hex');
console.log('buffer3', buffer3.length, buffer3.toString());
```

# READING BUFFERS

# ARRAY ACCESS

## 04_array_read.js

```
var buffer = new Buffer([1, 2, 3, 4]);

console.log(buffer[1] + buffer[2]);
```

# TOSTRING

## 05_tostring.js

```
var parseBasicAuth = function(encodedString) {
  var buffer = new Buffer(encodedString, 'base64');
  var parts = buffer.toString('utf8').split(':');

  return {
    username: parts[0],
    password: parts[1]
  };
};

var credentials = parseBasicAuth('dXNlcjEyMzpzdXAzcnNlY3JldDc=');

console.log(credentials);
```

# READING VALUES

## 06_bitmap_header.js

```
var fs = require('fs');

fs.readFile(__dirname + '/support/2x16.bmp', function(err, buffer) {
  if (err) {
    throw err;
  }

  // See: http://en.wikipedia.org/wiki/BMP_file_format#Example_1
  if (buffer.toString('ascii', 0, 2) === 'BM') {

    console.log('width:', buffer.readInt32LE(0x12));
    console.log('height:', Math.abs(buffer.readInt32LE(0x16)));
    console.log('color depth:', buffer.readUInt16LE(0x1C));
  }
});
```

# WRITING BUFFERS

# WRITE

## 07_write_string.js

```
var buffer = new Buffer(100);

buffer.write('hello ');
buffer.write('world', 6);

console.log(buffer.toString('utf8', 0, 11));
```

# WRITING MULTIBYTE VALUES

Creating a windows bitmap file

**08_write_multibyte.js**

```
var fs = require('fs');
var width = 16;
var height = 16;
var pixelByteSize = width * height * 4;
var totalSize = pixelByteSize + 54;

/// create a buffer and populate it

fs.writeFile(__dirname + '/support/out.bmp', buffer, function(err) {
  if (err) throw err;
});
```

## **08_write_multibyte.js** continued

```
/// create a buffer and populate it
var buffer = new Buffer(totalSize);

// See: http://en.wikipedia.org/wiki/BMP_file_format#Example_1
buffer.fill(0);
buffer.write('BM');
buffer.writeUInt32LE(totalSize, 0x02);      // size of bitmap file
buffer.writeUInt32LE(54, 0x0A);             // pixel data offset
buffer.writeUInt32LE(40, 0x0E);             // size of header
buffer.writeUInt16LE(1, 0x1A);              // color planes
buffer.writeUInt32LE(32, 0x1C);             // bits per color
buffer.writeUInt32LE(pixelByteSize, 0x22);  // total pixel size
buffer.writeInt32LE(width, 0x12);           // width
buffer.writeInt32LE(height, 0x16);          // height

for (var i=54; i<totalSize; i+=4) {
  buffer.writeUInt32LE(0xFF0000FF, i); // write blue @ 100% opacity
}
```

# ARRAY ACCESS

## 09_array_write.js

```
var buffer = new Buffer(10);

buffer.fill(0);

for (var i=0; i<=9; i+=2) {
  buffer[i] = 0xff;
}

console.log(buffer);
```

# MANIPULATING BUFFERS

# CONCAT

## 10_concat.js

```
var buffer1 = new Buffer('hello');
var buffer2 = new Buffer('world');
var buffer3 = Buffer.concat([buffer1, buffer2]);

console.log(buffer3.toString());

buffer2.fill(0);

console.log(buffer3.toString());
```

# SLICE

## 11_slice.js

```
var buffer = new Buffer('some data');
var buffer2 = buffer.slice(0,1)

console.log(buffer2);

buffer[0] = 0xff;

console.log(buffer2);
```

# SUMMARY

- Buffers provide a means to manipulate binary data
- Useful for doing encoding
- Extremely useful for parsing binary structures
- Efficient (no converting back and forth to v8 strings)
- See the manual for a complete list of operations