Binary Data in Ruby

David Grayson
Las Vegas Ruby Meetup
2014-05-07

My applications of binary data

- Brawlsnapshots.com (2008)
  - Extract metadata from user-uploaded stage file
- redstone-bot2 (2012)
  - Minecraft bot written in Ruby
- Ruby ECDSA gem (2014)
  - Supports standard binary data formats

Outline

- Quick introduction to binary data
  - Bytes
  - ASCII
- Binary data in Ruby
  - Strings
  - Getting bytes and integers from a binary string
  - Bit fields
  - IO objects

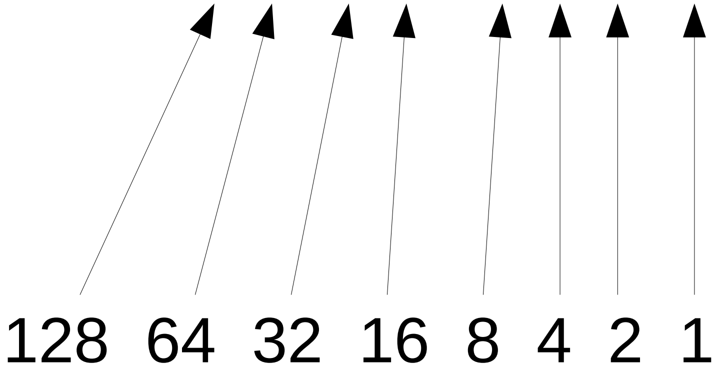
This talk focuses on reading information from binary data rather than writing binary data, but Ruby supports both!

Definition: Byte

1. A number between 0 and 255
2. A storage location that can hold such a number

A byte has 8 bits

```
Bits of byte 0:   0000 0000
Bits of byte 1:   0000 0001
Bits of byte 2:   0000 0010
Bits of byte 132: 1000 0100
```



128 64 32 16 8 4 2 1

List of all bytes in decimal

```
 0   16   32   48   64   80   96  112  128  144  160  176  192  208  224  240
 1   17   33   49   65   81   97  113  129  145  161  177  193  209  225  241
 2   18   34   50   66   82   98  114  130  146  162  178  194  210  226  242
 3   19   35   51   67   83   99  115  131  147  163  179  195  211  227  243
 4   20   36   52   68   84  100  116  132  148  164  180  196  212  228  244
 5   21   37   53   69   85  101  117  133  149  165  181  197  213  229  245
 6   22   38   54   70   86  102  118  134  150  166  182  198  214  230  246
 7   23   39   55   71   87  103  119  135  151  167  183  199  215  231  247
 8   24   40   56   72   88  104  120  136  152  168  184  200  216  232  248
 9   25   41   57   73   89  105  121  137  153  169  185  201  217  233  249
10   26   42   58   74   90  106  122  138  154  170  186  202  218  234  250
11   27   43   59   75   91  107  123  139  155  171  187  203  219  235  251
12   28   44   60   76   92  108  124  140  156  172  188  204  220  236  252
13   29   45   61   77   93  109  125  141  157  173  189  205  221  237  253
14   30   46   62   78   94  110  126  142  158  174  190  206  222  238  254
15   31   47   63   79   95  111  127  143  159  175  191  207  223  239  255
```

List of all bytes in hexadecimal

```
00  10  20  30  40  50  60  70  80  90  a0  b0  c0  d0  e0  f0
01  11  21  31  41  51  61  71  81  91  a1  b1  c1  d1  e1  f1
02  12  22  32  42  52  62  72  82  92  a2  b2  c2  d2  e2  f2
03  13  23  33  43  53  63  73  83  93  a3  b3  c3  d3  e3  f3
04  14  24  34  44  54  64  74  84  94  a4  b4  c4  d4  e4  f4
05  15  25  35  45  55  65  75  85  95  a5  b5  c5  d5  e5  f5
06  16  26  36  46  56  66  76  86  96  a6  b6  c6  d6  e6  f6
07  17  27  37  47  57  67  77  87  97  a7  b7  c7  d7  e7  f7
08  18  28  38  48  58  68  78  88  98  a8  b8  c8  d8  e8  f8
09  19  29  39  49  59  69  79  89  99  a9  b9  c9  d9  e9  f9
0a  1a  2a  3a  4a  5a  6a  7a  8a  9a  aa  ba  ca  da  ea  fa
0b  1b  2b  3b  4b  5b  6b  7b  8b  9b  ab  bb  cb  db  eb  fb
0c  1c  2c  3c  4c  5c  6c  7c  8c  9c  ac  bc  cc  dc  ec  fc
0d  1d  2d  3d  4d  5d  6d  7d  8d  9d  ad  bd  cd  dd  ed  fd
0e  1e  2e  3e  4e  5e  6e  7e  8e  9e  ae  be  ce  de  ee  fe
0f  1f  2f  3f  4f  5f  6f  7f  8f  9f  af  bf  cf  df  ef  ff
```

ASCII code

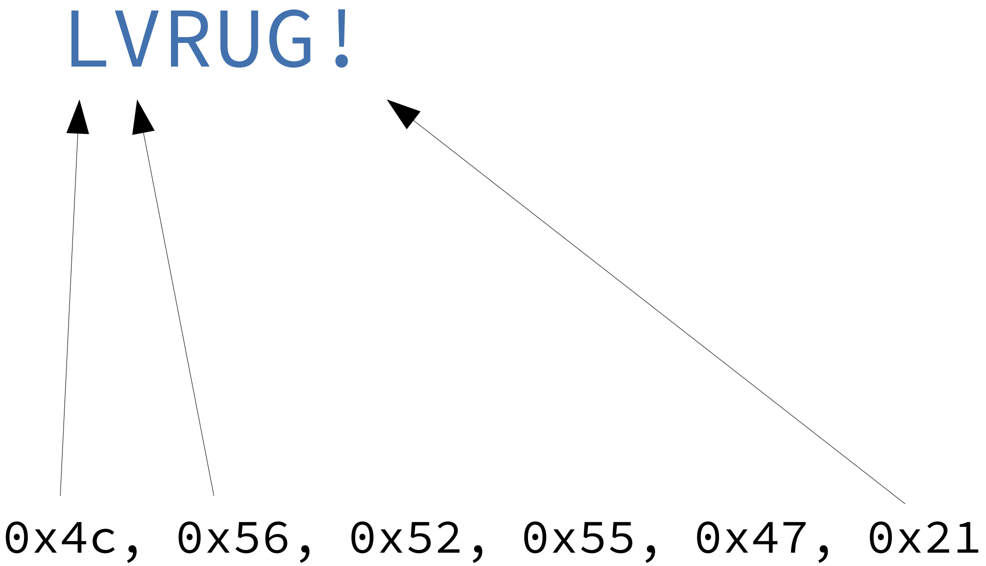
00		10		20		30	0	40	@	50	P	60	`	70	p
01		11		21	!	31	1	41	A	51	Q	61	a	71	q
02		12		22	\	32	2	42	B	52	R	62	b	72	r
03		13		23	#	33	3	43	C	53	S	63	c	73	s
04		14		24	$	34	4	44	D	54	T	64	d	74	t
05		15		25	%	35	5	45	E	55	U	65	e	75	u
06		16		26	&	36	6	46	F	56	V	66	f	76	v
07	\a	17		27	'	37	7	47	G	57	W	67	g	77	w
08	\b	18		28	(	38	8	48	H	58	X	68	h	78	x
09	\t	19		29	)	39	9	49	I	59	Y	69	i	79	y
0a	\n	1a		2a	*	3a	:	4a	J	5a	Z	6a	j	7a	z
0b	\v	1b	\e	2b	+	3b	;	4b	K	5b	[	6b	k	7b	{
0c	\f	1c		2c	,	3c	<	4c	L	5c	\	6c	l	7c	|
0d	\r	1d		2d	-	3d	=	4d	M	5d	]	6d	m	7d	}
0e		1e		2e	.	3e	>	4e	N	5e	^	6e	n	7e	~
0f		1f		2f	/	3f	?	4f	O	5f	_	6f	o	7f	

Not shown: ASCII characters without escape sequences in Ruby

Text files generally use ASCII

Text editor display:

Actual bytes in text file:



`LVRUG!`

```
0x4c, 0x56, 0x52, 0x55, 0x47, 0x21
```

But there are many other encodings, like UTF-8, UTF-16.

Definition: text

- A sequence of characters

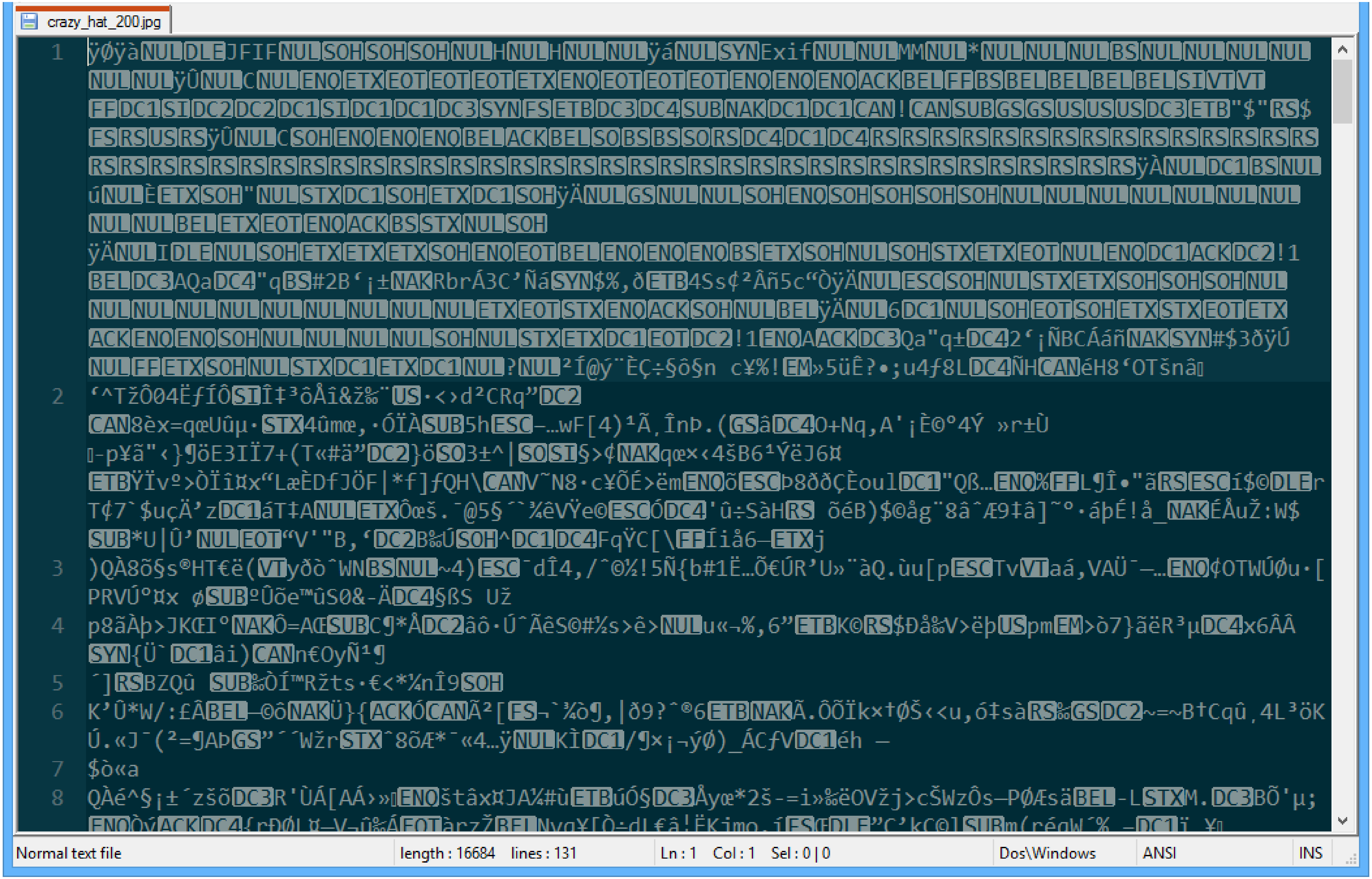
Definition: binary data

- A sequence of bytes that is not text

General properties of binary data

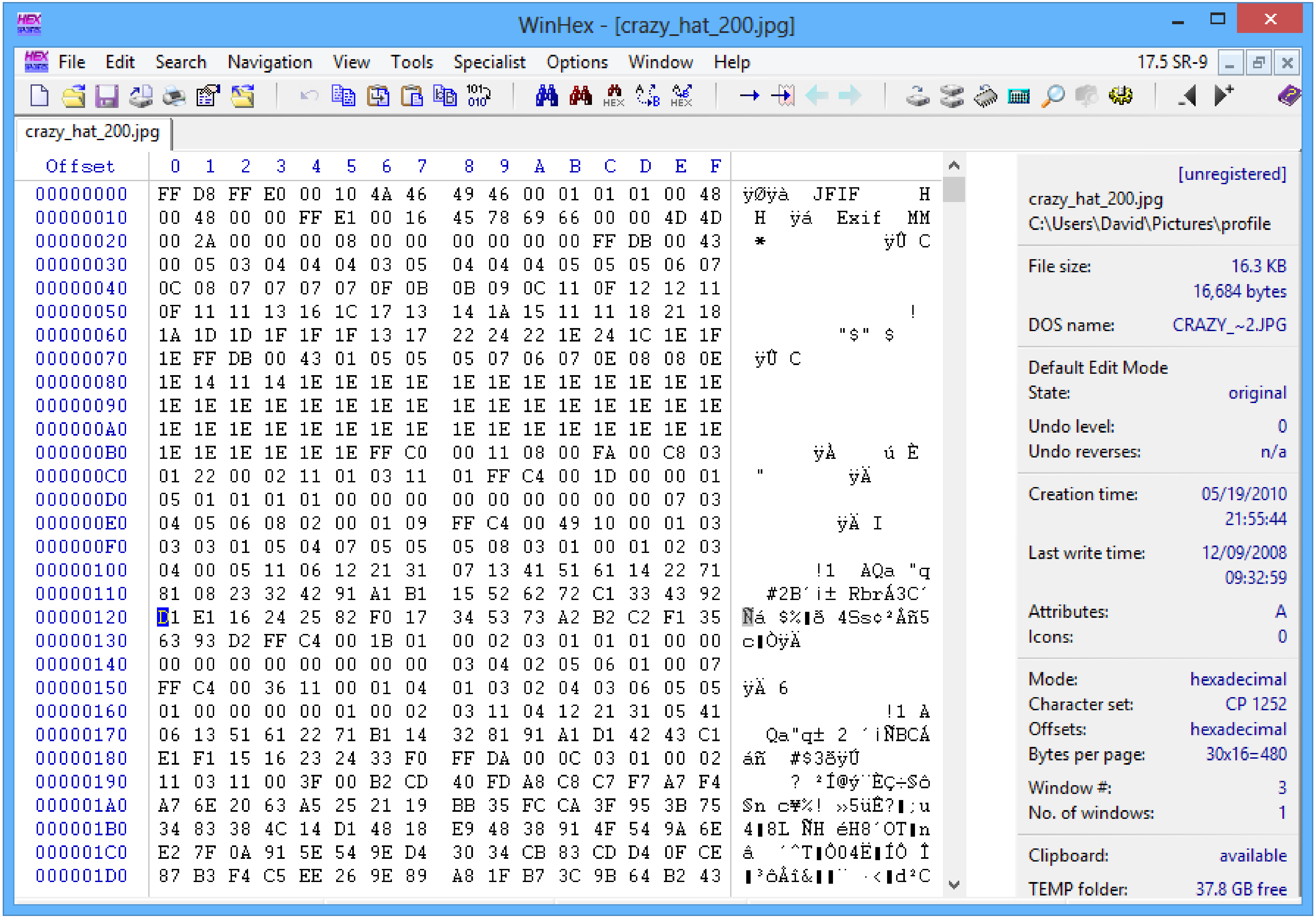
- It has bytes that are not ASCII characters, like 0x80–0xFF (128 to 255)
- It stores data more compactly than text.
- It looks like junk in a text editor.

Text editors



```
ÿØÿàNULDLEJFIFNULSOHSOHSOHNULHNULHNULNULÿáNULSYNExifNULNULMMNUL*NULNULNULBSNULNULNULNUL
NULNULÿÛNULCNULENQETXEOTEOTEOTETXENQEOTEOTEOTENQENQENQACKBELFFBSBELBELBELBELSIVTVT
FFDC1SIDC2DC2DC1SIDC1DC1DC3SYNFSETBDC3DC4SUBNAKDC1DC1CAN!CANSUBGSGSUSUSUSDC3ETB"$"RS$
FSRSUSRSÿÛNULCSOHENQENQENQBELACKBELSOBSBSSORSDC4DC1DC4RSRSRSRSRSRSRSRSRSRSRSRSRSRSRS
RSRSRSRSRSRSRSRSRSRSRSRSRSRSRSRSRSRSRSRSRSRSRSRSRSRSRSRSRSRSRSRSRSRSRSÿÀNULDC1BSNUL
úNULÈETXSOH"NULSTXDC1SOHETXDC1SOHÿÄNULGSNULNULSOHENQSOHSOHSOHSOHNULNULNULNULNULNULNUL
NULNULBELETXEOTENQACKBSSTXNULSOH
ÿÄNULIDLENULSOHETXETXETXSOHENQEOTBELENQENQENQBSETXSOHNULSOHSTXETXEOTNULENQDC1ACKDC2!1
BELDC3AQaDC4"qBS#2B‘¡±NAKRbrÁ3C’ÑáSYN$%,ðETB4Ss¢²Âñ5c“ÒÿÄNULESCSOHNULSTXETXSOHSOHSOHNUL
NULNULNULNULNULNULNULNULNULETXEOTSTXENQACKSOHNULBELÿÄNUL6DC1NULSOHEOTSOHETXSTXEOTETX
ACKENQENQSOHNULNULNULNULSOHNULSTXETXDC1EOTDC2!1ENQAACKDC3Qa"q±DC42‘¡ÑBCÁáñNAKSYN#$3ðÿÚ
NULFFETXSOHNULSTXDC1ETXDC1NUL?NUL²Í@ý¨ÈÇ÷§ô§n c¥%!EM»5üÊ?•;u4ƒ8LDC4ÑHCANéH8‘OTšnâ
‘^TžÔ04ËƒÍÔSIÎ‡³ôÅî&ž‰¨US·<›d²CRq”DC2
CAN8èx=qœUûµ·STX4ûmœ,·ÓÏÀSUB5hESC–…wF[4)¹Ã¸ÎnÞ.(GSâDC40+Nq,A'¡È©°4Ý »r±Ù
-p¥ã"‹}¶öE3IÏ7+(T«#ä”DC2}öSO3±^|SOSI§>¢NAKqœ×‹4šB6¹ŸëJ6¤
ETBŸÏvº>ÒÏî¤x“LæÈDfJÖF|*f]ƒQH\CANV˜N8·c¥ÕÉ>ëmENQõESCÞ8ððÇÈoulDC1"Qß…ENQ%FFL¶Î•"ãRSESCí$©DLEr
T¢7`$uçÄ’zDC1áT‡ANULETXÔœš.¯@5§``¾êVŸe©ESCÓDC4'û÷SàHRS õéB)$©åg¨8â^Æ9‡â]˜°·áþÉ!å_NAKÉÅuŽ:W$
SUB*U|Û’NULEOT“V'"B,‘DC2B‰ÚSOH^DC1DC4FqŸC[\FFÍiå6–ETXj
)QÀ8õ§s®HT€ë(VTyðò^WNBSNUL~4)ESC¯dÎ4,/^©½!5Ñ{b#1Ë…Õ€ÚR’U»¨àQ.ùu[pESCTvVTaá,VAÜ¯—…ENQ¢OTWÚØu·[
PRVÚº¤x øSUBºÛõe™ûS0&-ÄDC4§ßS Už
p8ãÀþ>JKŒIºNAKÔ=AŒSUBC¶*ÅDC2âô·Ú^ÃêS©#½s>ê>NULu«¬%,6”ETBK©RS$Ðå‰V>ëþUSpmEM>ò7}ãëR³µDC4x6ÂÂ
SYN{Ü`DC1âi)CANn€OyÑ¹¶
´]RSBZQû SUB‰ÒÍ™Ržts·€<*¼nÎ9SOH
K’Û*W/:£ÂBEL–©ôNAKÜ}{ACKÓCANÃ²[FS¬`¾ò¶,|ð9?^®6ETBNAKÃ.ÔÕÏk×†ØŠ‹<u,ó‡sàRS‰GSDC2~=~B†Cqû¸4L³öK
Ú.«J¯(²=¶AÞGS”´´WžrSTX^8õÆ*¯«4…ÿNULKÌDC1/¶×¡¬ýØ)_ÁCƒVDC1éh –
$ò«a
QÀé^§¡±´zšõDC3R'ÙÁ[AÁ›»ENQštâx¤JA¼#ùETBúÓ§DC3Åyœ*2š-=i»‰ëOVžj>cŠWzÔs—PØÆsäBEL-LSTXM.DC3BÕ'µ;
ENQÒýACKDC4{rÐØL¤—V-û‰ÁEOTàrzŽBELNvq¥[Ò÷dL€â!ËKimo.íFSŒDLE”C’kC©]SUBm(réqW´% –DC1ï ¥
```

Hex editors



```
Offset     0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
00000000  FF D8 FF E0 00 10 4A 46  49 46 00 01 01 01 00 48  ÿØÿà  JFIF     H
00000010  00 48 00 00 FF E1 00 16  45 78 69 66 00 00 4D 4D   H  ÿá  Exif  MM
00000020  00 2A 00 00 00 08 00 00  00 00 00 00 FF DB 00 43   *          ÿÛ C
00000030  00 05 03 04 04 04 03 05  04 04 04 05 05 05 06 07
00000040  0C 08 07 07 07 07 0F 0B  0B 09 0C 11 0F 12 12 11
00000050  0F 11 11 13 16 1C 17 13  14 1A 15 11 11 18 21 18                ! 
00000060  1A 1D 1D 1F 1F 1F 13 17  22 24 22 1E 24 1C 1E 1F          "$" $   
00000070  1E FF DB 00 43 01 05 05  05 07 06 07 0E 08 08 0E   ÿÛ C
00000080  1E 14 11 14 1E 1E 1E 1E  1E 1E 1E 1E 1E 1E 1E 1E
00000090  1E 1E 1E 1E 1E 1E 1E 1E  1E 1E 1E 1E 1E 1E 1E 1E
000000A0  1E 1E 1E 1E 1E 1E 1E 1E  1E 1E 1E 1E 1E 1E 1E 1E
000000B0  1E 1E 1E 1E 1E 1E FF C0  00 11 08 00 FA 00 C8 03        ÿÀ    ú È 
000000C0  01 22 00 02 11 01 03 11  01 FF C4 00 1D 00 00 01   "       ÿÄ
000000D0  05 01 01 01 01 00 00 00  00 00 00 00 00 00 07 03
000000E0  04 05 06 08 02 00 01 09  FF C4 00 49 10 00 01 03          ÿÄ I
000000F0  03 03 01 05 04 07 05 05  05 08 03 01 00 01 02 03
00000100  04 00 05 11 06 12 21 31  07 13 41 51 61 14 22 71        !1  AQa "q
00000110  81 08 23 32 42 91 A1 B1  15 52 62 72 C1 33 43 92    #2B´¡± RbrÁ3C´
00000120  D1 E1 16 24 25 82 F0 17  34 53 73 A2 B2 C2 F1 35  Ñá $%▮ð 4Ss¢²Âñ5
00000130  63 93 D2 FF C4 00 1B 01  00 02 03 01 01 01 00 00  c▮ÒÿÄ
00000140  00 00 00 00 00 00 00 00  03 04 02 05 06 01 00 07
00000150  FF C4 00 36 11 00 01 04  01 03 02 04 03 06 05 05  ÿÄ 6
00000160  01 00 00 00 00 01 00 02  03 11 04 12 21 31 05 41              !1 A
00000170  06 13 51 61 22 71 B1 14  32 81 91 A1 D1 42 43 C1    Qa"q± 2 ´¡ÑBCÁ
00000180  E1 F1 15 16 23 24 33 F0  FF DA 00 0C 03 01 00 02  áñ  #$3ðÿÚ
00000190  11 03 11 00 3F 00 B2 CD  40 FD A8 C8 C7 F7 A7 F4      ? ²Í@ý¨ÈÇ÷§ô
000001A0  A7 6E 20 63 A5 25 21 19  BB 35 FC CA 3F 95 3B 75  §n c¥%! »5üÊ?▮;u
000001B0  34 83 38 4C 14 D1 48 18  E9 48 38 91 4F 54 9A 6E  4▮8L ÑH éH8´OT▮n
000001C0  E2 7F 0A 91 5E 54 9E D4  30 34 CB 83 CD D4 0F CE  â  ´^T▮Ô04Ë▮ÍÔ Î
000001D0  87 B3 F4 C5 EE 26 9E 89  A8 1F B7 3C 9B 64 B2 43  ▮³ôÅî&▮▮¨ ·<▮d²C
```

[unregistered]
crazy_hat_200.jpg
C:\Users\David\Pictures\profile

File size:	16.3 KB 16,684 bytes
DOS name:	CRAZY_~2.JPG
Default Edit Mode	
State:	original
Undo level:	0
Undo reverses:	n/a
Creation time:	05/19/2010 21:55:44
Last write time:	12/09/2008 09:32:59
Attributes:	A
Icons:	0
Mode:	hexadecimal
Character set:	CP 1252
Offsets:	hexadecimal
Bytes per page:	30x16=480
Window #:	3
No. of windows:	1
Clipboard:	available
TEMP folder:	37.8 GB free

Recap of binary data

- A byte is a number between 0 and 255.
- A byte can be written as 2 hex digits.
- A byte can be written as 8 bits.
- ASCII is a popular mapping between *bytes* and *characters*.

Binary data in Ruby

- Typically stored in a String

```
str = "\x12\x34\xFE"
```

Fix your string's encoding!

```
"\x12\x34\xfe".force_encoding('BINARY')
```

```
# coding: ASCII-8BIT
"\x12\x34\xfe"
```

first line of file

```
def some_public_method(str)
  str = str.dup.force_encoding('BINARY')
  # ...
end
```

Integer literals in Ruby

- Three ways to write the same number:

```
181        # decimal
0xb5       # hex
0b10110101 # binary
```

Inspecting a string's bytes

```
str = "\x0d\x0e\x00\x40"

str.bytes.to_a    # => [13, 14, 0, 64]

str.inspect       # => "\r\x0E\x00@"

str.hex_inspect   # => "\x0d\x0e\x00\x40"
```

```
class String
  def hex_inspect
    '"' + each_byte.map { |b| '\x%02x' % b }.join + '"'
  end
end
```

https://gist.github.com/DavidEGrayson/10093790

Getting a single byte from a string

```
str = "\x0d\x0e\x00\x40"

byte1 = str[1].ord  # => 14
```

Unpacking integers from a string

- Main tool: String#unpack
- Big endian vs. little endian

```
# two 8-bit unsigned integers
"\x34\x12".unpack('cc')  # => [0x34, 0x12]

# one 16-bit unsigned int, little-endian
"\x34\x12".unpack('S')   # => [0x1234]

# one 16-bit unsigned int, big-endian
"\x34\x12".unpack('n')   # => [0x3412]
```

http://ruby-doc.org/core/String.html#method-i-unpack

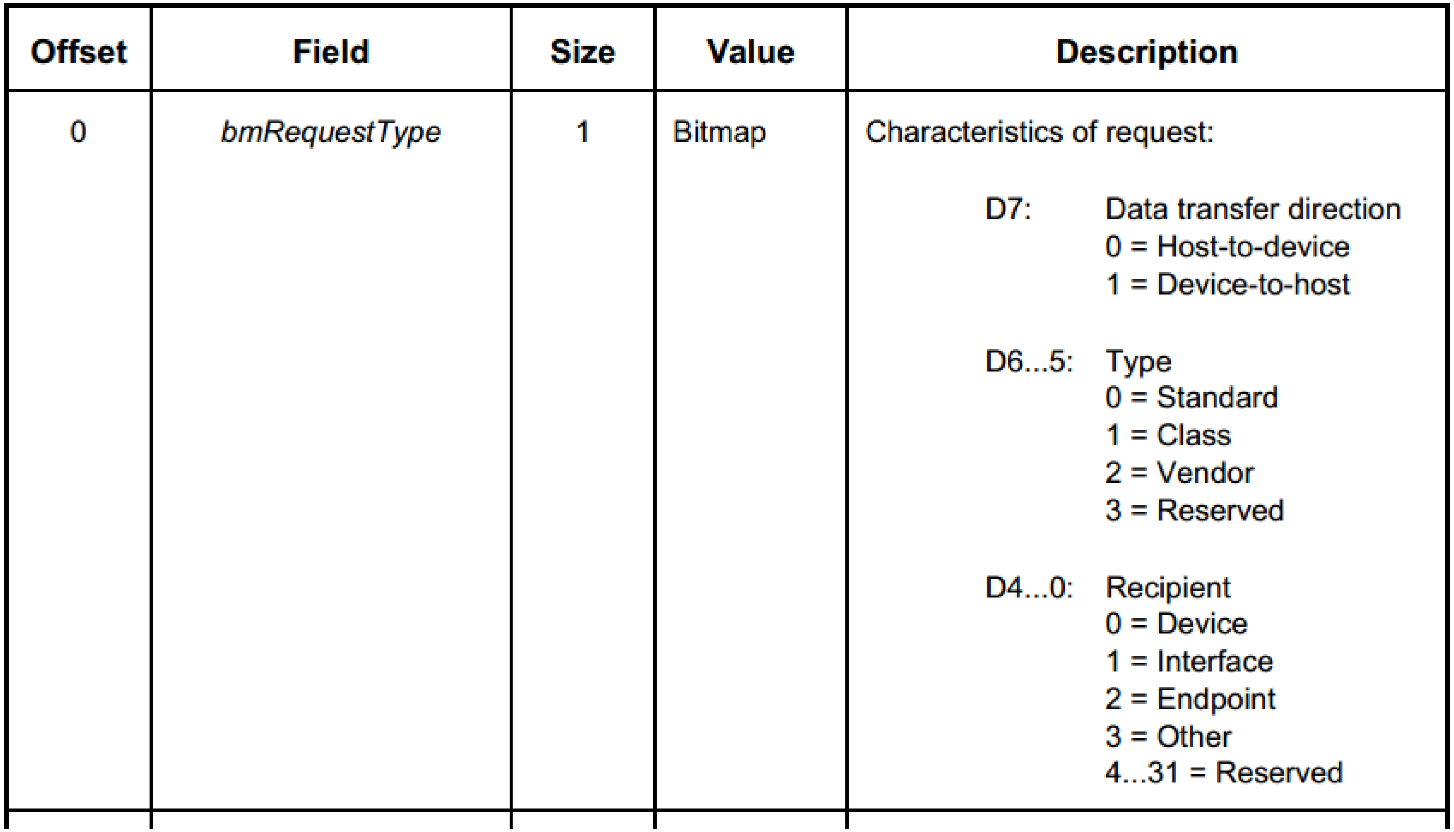
Converting integers to hex

```
'm = %02x' % [14]     # => "m = 0e"
```

Bit fields

- Bits and groups of bits inside a byte can have individual meaning

Table 9-2. Format of Setup Data



Offset	Field	Size	Value	Description
0	*bmRequestType*	1	Bitmap	Characteristics of request: D7: Data transfer direction 0 = Host-to-device 1 = Device-to-host D6...5: Type 0 = Standard 1 = Class 2 = Vendor 3 = Reserved D4...0: Recipient 0 = Device 1 = Interface 2 = Endpoint 3 = Other 4...31 = Reserved

Getting bit fields from an integer

- Binary operators:
  - << bitwise shift left
  - >> bitwise shift right
  - & bitwise and
  - | bitwise or
- Example: extract a 2-bit bitfield in bits 5-6

```
x = 0b11000000      # same as 0xC0
(x >> 5) & 0b11     # => 0b10
```

Read binary data from a file

```
f = File.open('foo.dat', 'rb')

f.read(2)  # => 2-byte string
f.read(10) # => 10-byte string

f.close
```

More methods you can use on files & sockets: http://www.ruby-doc.org/core/IO.html

Converting a String to an IO object

```
require 'stringio'
io = StringIO.new "\x0d\x0e\x00\x40"

io.read(2)  # => "\x0d\x0e"
io.read(1)  # => "\x00"
io.read(1)  # => "\x40"
io.read(1)  # => nil
```

Reading variable-length binary data

```
"\x02\x00\x12\x34\xAA\xBB"

length = io.read(2).unpack('S')[0]
array = length.times.map do
  io.read(2).unpack('S')[0]
end
```

Extending or refining the IO class allows clearer code!

```
length = io.read_uint16
array = length.times.map do
  io.read_uint16
end
```

Example
IO extension

```ruby
module DataReader
  def read_uint8
    read(1).ord
  end

  def read_int8
    read(1).unpack('c')[0]
  end

  def read_int16
    read(2).unpack('s')[0]
  end

  def read_uint16
    read(2).unpack('S')[0]
  end

  def read_uint16_array
    length = read_uint16
    length.times.map { read_uint16 }
  end
end

io_object.extend DataReader
```

Example: reading Minecraft Entity Properties packet (simplified)

```ruby
def receive_data(stream)
  @eid = stream.read_int
  property_count = stream.read_int
  @properties = property_count.times.map do
    key = stream.read_string
    value = stream.read_double
    [key, value]
  end
end
```

Creating binary data

- Array#pack
- String#concat (or just adding strings)
- IO#write
- Be careful to open files in binary mode:
  - `File.open(name, 'wb')`

End