

株式会社 ITS MORE

2020年4月設立

2020年5月18日 投稿者: YSATO@DELEGATE.ORG

goによるpngの生成

サーバで画像を生成して配信したい。まずは静止画で、アイコン、グラフ、QRコード等。高々100KB程度の画像を想定している。要するに、どのピクセルを何色にするか、だけのデータなのだから、簡単にできるに違いない。

PNG選定までの経緯

静止画ならpngが形式がシンプルで良かった記憶がある。機能的に大きな問題がなければPNGで行きたい気分。

静止画、フォーマットでぐぐると、[このブログ](#)が先頭にくる。比較対象としてあげられている JPEG, GIF, PNG, TIFF, BMP。生成する画像は写真では無いので、JPEGは必要無い。ただの一枚の静止画なのでTIFFである必要は無い。ブラウザで表示することが目的なので、やはりPNGかな。基本的には、非常に近距離で、小さな画像をやりとりするのが目的なので、圧縮の機能も必要ないというかOFFにできると良い。

[Wiki](#) をみても、PNGが適当と思う。PNGが出てきた経緯からして、それが普及した現在、GIFに戻る必要はないのでは。ひとつだけ気になるのはGIFアニメ。でも、アニメーションPNGというのがあるそうで、これも試してみたら、現在の主要5ブラウザで全てで表示できた。

全て問題なし。PNGが良いと思う。[WikiのPNGの項](#)を見たら、だいたいそう言うメリットがまとめられている。

実装言語としてのGo

それで仕様書は何だったろうとたぐると、大元は RFC 2083 だった。

Goでさくっと実装したい。仕様をざっと見て、共通ライブラリ的なもので、自作の線はないなと思うのは zlib。でもこれは go にパッケージがあるようなのでOK。CRC32も問題ない。そもそも可逆圧縮なんだし、フォーマットと圧縮は直交しているはずだ。

ん、さてよ、そもそもGo には PNG自体のライブラリがあるのではないか？やはり、[Package png](#) というのがあった。エンコーディングについては、これに任せれば良いらしい。その中で参照されているPNGの仕様は、[W3C](#)のものだった [<https://www.w3.org/TR/PNG/>]。2003年版とあるので、いい感じに枯れてるのじゃあるまいか。

PNGのフォーマット

Go で PNG で行こう。実際、もともとその一択だったと思う。準拠する仕様は W3C の 2003 年版すなわち、ISO/IEC 15948:2003 とする。

Portable Network Graphics (PNG) Specification (Second Edition)

Information technology — Computer graphics and image processing — Portable Network Graphics (PNG): Functional specification. ISO/IEC 15948:2003 (E)

W3C Recommendation 10 November 2003

<https://www.w3.org/TR/PNG/>

仕様書をざっと眺める。PDFにしても40~50ページ程度だから、コンパクトな規格だ。画像が入っているのが IETF RFCとは違うところだな（笑）。目次や Term & definitions の章がしっかりしているのは ISO の血の匂いがする。いろいろ書いてあるけど、とりあえず必要な情報って数ページ程度ではあるまいか？

やりたいことは PNG 形式を作る事、つまり 4.7 PNG datastream を作る事だ。ヘッダの後は、チャンクの連続だと書いてある。必須（クリティカル）のチャンクは、ヘッダ、パレット、データ、終端の4種類か。

The screenshot shows the W3C Recommendation page for PNG 4.7 PNG datastream. The page title is "4.7 PNG datastream". The main heading is "4.7.1 Chunks". The text states: "The PNG datastream consists of a PNG signature (see 5.2: [PNG signature](#)) followed by a sequence of chunks (see clause 11: [Chunk specifications](#)). Each chunk has a chunk type which specifies its function." Below this is "4.7.2 Chunk types". The text states: "There are 18 chunk types defined in this International Standard. Chunk types are four-byte sequences chosen so that they correspond to readable labels when interpreted in the ISO 646.IRV:1991 character set. The first four are termed critical chunks, which shall be understood and correctly interpreted according to the provisions of this International Standard. These are:" followed by a list:

- a. **IHDR**: image header, which is the first chunk in a PNG datastream.
- b. **PLTE**: palette table associated with indexed PNG images.
- c. **IDAT**: image data chunks.
- d. **IEND**: image trailer, which is the last chunk in a PNG datastream.

それぞれの定義は 11.2.2,3,4,5 にある。2ページ程度の仕様だ。

The screenshot shows the W3C Recommendation page for PNG 11.2 Critical chunks. The page title is "11.2 Critical chunks". The main heading is "11.2.1 General". The text states: "Critical chunks are those chunks that are absolutely required in order to successfully decode a PNG image from a PNG datastream. Extension chunks may be defined as critical chunks (see clause 14: [Editors and extensions](#)), though this practice is strongly discouraged." Below this is "11.2.2 IHDR Image header". The text states: "A valid PNG datastream shall begin with a PNG signature, immediately followed by an **IHDR** chunk, then one or more **IDAT** chunks, and shall end with an **IEND** chunk. Only one **IHDR** chunk and one **IEND** chunk are allowed in a PNG datastream." Below this is "The four-byte chunk type field contains the decimal values" followed by the text "73 72 68 82". Below this is "The **IHDR** chunk shall be the first chunk in the PNG datastream. It contains:" followed by a table:

| | |
|--------------------|---------|
| Width | 4 bytes |
| Height | 4 bytes |
| Bit depth | 1 byte |
| Colour type | 1 byte |
| Compression method | 1 byte |

だが、値を十進で表記するのやめて欲しい。ISO流なのか？なぜ「73 72 68 82」というオクテット列はASCIIコードとして解釈すると偶然ではなく「IHDR」という文字

列になります」と一言、言えないのだろうか？

まあいい。感じはわかった。いい感じだ。さくっと行けそうな気がする。

トライアル実装 (1)

さっそく何か作ってみよう。うーん、Cだと簡単に書けるのだが … とりあえずCで書こうか … つくり初めてみたが何か変だ。無意識に mkgif.c という名前でプログラムを作っていた (笑)

やはりGoで書こう。… … … こんな感じになった。

```
//
// 2020-0518-01 pnggen/0.0.1 (PNG generator) by ITS more :-)
// Reference: https://www.w3.org/TR/PNG/
//
package main
import (
    "os"
    "fmt"
)
func hton32b(s string, i int)(string){
    s += string(0xFF & (i >> 24))
    s += string(0xFF & (i >> 16))
    s += string(0xFF & (i >> 8))
    s += string(0xFF & (i))
    return s
}
func putNetInt32(file *os.File, length int){
    lengb := make([]byte, 4)
    lengs := string(lengb)
    lengs = ""
    lengs = hton32b(lengs,length)
    file.Write([]byte(lengs))
}
func main() {
    outfile := "x.png"
    fmt.Fprintf(os.Stderr, "-- pnggen/0.0.1 --\n")
    file, _ := os.Create(outfile)
    defer file.Close()

    PNGsignature := "\x89PNG\r\n\x1A\n" // 5.2 PNG signature
    file.Write([]byte(PNGsignature))
}
```

```
// 5.3 Chunk fields := Length, Chunk Type, Chunk Data, CRC
ckdbuf := make([]byte, 256) // 5.3 CHUNK DATA buffer
chunkdtop := string(ckdbuf)
chunkdtop = ""

// 11.2.2 IHDR Image header
IHDRtype := "IHDR" // IHDR chunk type
// IHDR chunk data
chunkd := hton32b(chunkdtop,64) // Width
chunkd = hton32b(chunkd,64) // Hight
chunkd += string(1) // Bit depth, 1 for monochrome
chunkd += string(0) // Color type, 0 for Grayscale
chunkd += string(0) // Compression method, 0 for deflate/i
chunkd += string(0) // Filter method, 0 for adaptive
chunkd += string(0) // Interlace method, 0 for no interlac
// IHDR CRC
// to be done

putNetInt32(file,len(chunkd)) // Length
file.Write([]byte(IHDRtype))
file.Write([]byte(chunkd))

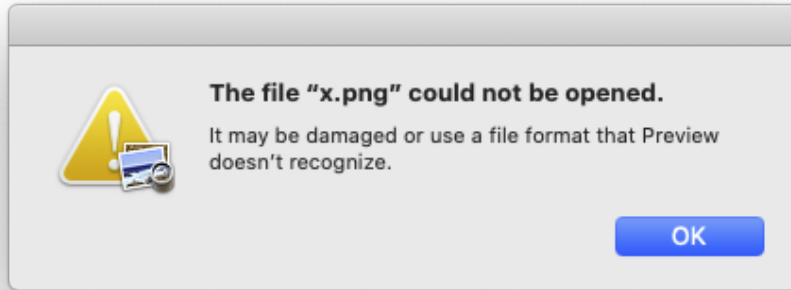
fmt.Fprintf(os.Stderr,"-- created %s (%d x %d)\n",outfile,
}
```

プログラムから出力したPNGファイルの中身を確認する。

```
% go run pnggen.go
-- pnggen/0.0.1 --

-- created x.png (64 x 64)
% file x.png
x.png: PNG image data, 64 x 64, 1-bit grayscale, non-interlaced
% od -c -t d1 x.png
00000000  211  P  N  G  \r  \n 032  \n  \0  \0  \0  \r  I  H
          -119 80 78 71 13 10 26 10  0  0  0 13 73 72
00000020  \0  \0  \0  @  \0  \0  \0  @ 001  \0  \0  \0  \0
          0  0  0 64  0  0  0 64  1  0  0  0  0
```

とりあえず file コマンドは、これが 64 x 64 のPNGだと認識してくれている。CRCはまだつけてないけど、チェックしてないんだな。では Preview で開いてみよう。

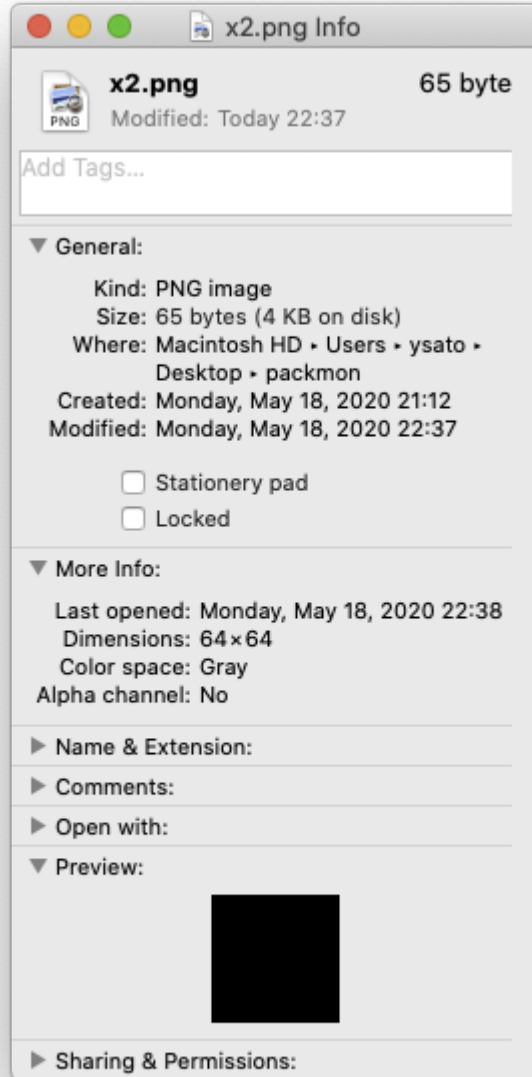


おっしゃるとおり、ダメージを受けております。ブラウザは何と言うだろう？



Firefoxは怒った。が、他のブラウザたちはノールックでスルーパス。わりといい加減なんだな。

どうやら必須エリアはそろってないと怒られるようなので、空のパレットと空のイメージデータを追加して、IENDで終端してみる。で、プレビューで見ると…



できたー！苦節6時間（笑）半分以上は、Go言語の string 型との闘いだった。まだよくわからない。そもそもなぜ、printf / scanf でナマの整数値の読み書き、特にネットワークバイトオーダーとの変換ができないのか不思議だ。なぜたかが hton / ntoh するために、面倒なことをしなければならないのか？？

ともあれ、最小のPNGファイルは、65バイトであることがわかった。

```
% go run pnggen.go
-- pnggen/0.0.2 (PNG Generator) --
CRC=82124C73
CRC=4BA88955
CRC=35AF061E
CRC=AE426082
```

```
-- created x2.png (64 x 64)
```

```
% od -c x2.png
00000000 211 P N G \r \n 032 \n \0 \0 \0 \r I H
00000020 \0 \0 \0 @ \0 \0 \0 @ 001 \0 \0 \0 \0 202 0
00000040 s \0 \0 \0 \0 P L T E K 250 211 U \0
00000060 \0 I D A T 5 257 006 036 \0 \0 \0 \0 256
0000100 202
```

あれ？IENDが出てないや。出力し忘れた。Previewのチェックも中途半端だな。出力しよう。ということで、最小のPNGファイルは、69バイトであった。

```
% od -c x2.png
00000000 211 P N G \r \n 032 \n \0 \0 \0 \r I H
00000020 \0 \0 \0 @ \0 \0 \0 @ 001 \0 \0 \0 \0 202 0
00000040 s \0 \0 \0 \0 P L T E K 250 211 U \0
00000060 \0 I D A T 5 257 006 036 \0 \0 \0 \0 I
0000100 D 256 B ` 202
0000105
% od -t x1 x2.png
00000000 89 50 4e 47 0d 0a 1a 0a 00 00 00 0d 49 48
00000020 00 00 00 40 00 00 00 40 01 00 00 00 00 82
00000040 73 00 00 00 00 50 4c 54 45 4b a8 89 55 00
00000060 00 49 44 41 54 35 af 06 1e 00 00 00 00 49
0000100 44 ae 42 60 82
0000105
```

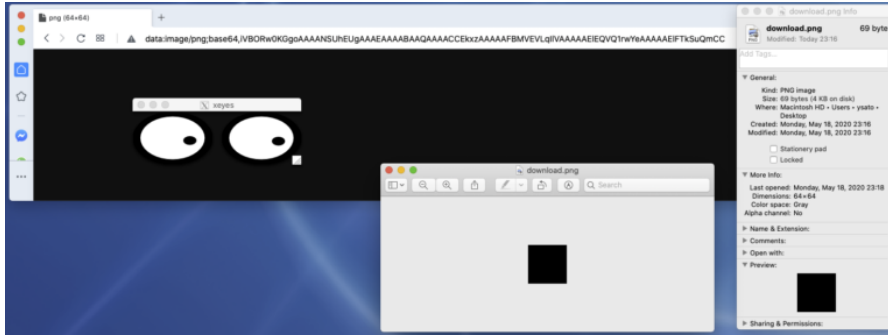
data URI にしてみよう。まずはbase64にする。

```
% delegated -FenMime -b x2.png
iVBORw0KGgoAAAANSUhEUgAAAEAAAABAAQAAACCEkxzAAAAAFBMVEVLqI1VAAAAAE
rwYeAAAAAE1FTkSuQmCC
```

data URI はこれで良いはずだ。

```
data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAEAAAABAAQAAACCEkxz
rwYeAAAAAE1FTkSuQmCC
```

OperaのURL窓に食わせて表示、ダウンロード。Preview で表示。問題なし。みんなで記念撮影しよう。



ミニマムPNGファイルできました。

あれ？そーいや zlib かけてないんだけど、通ったじゃん。自動判別してるのかな？
あ、圧縮するのはデータだけなのかな。とりあえず現状を保存。

Pnggen/0.0.2 (PNG Generator) ソースコード

```
//
// 2020-0518-02 pnggen/0.0.2 (PNG generator) by ITS more :-)
// Reference: https://www.w3.org/TR/PNG/
//
package main
import (
    "os"
    "fmt"
    "hash/crc32"
)
func myid() string {
    return "pnggen/0.0.2 (PNG Generator)"
}
func hton32b(i uint32)(string){
    buf := make([]byte, 4)
    buf[0] = byte(0xFF & (i >> 24))
    buf[1] = byte(0xFF & (i >> 16))
    buf[2] = byte(0xFF & (i >> 8))
    buf[3] = byte(0xFF & (i))
    return string(buf)
}
func putNetInt32(file *os.File, i32 uint32){
    intb := make([]byte, 4)
    ints := string(intb)
    ints = ""
    ints = hton32b(i32)
    file.Write([]byte(ints))
}
func main() {
    outfile := "x2.png"
    genw := 64;
    genh := 64;
    fmt.Fprintf(os.Stderr, "-- %s --\n", myid())
    file, _ := os.Create(outfile)
    defer file.Close()

    PNGsignature := "\x89PNG\r\n\x1A\n" // 5.2 PNG signature
```

```

file.Write([]byte(PNGsignature))

// 5.3 Chunk fields := Length, (Chunk Type, Chunk Data), CRC
ckdbuf := make([]byte, 256) // 5.3 CHUNK DATA buffer
chunkd := string(ckdbuf)

//----- IHDR
// 11.2.2 IHDR Image header
// https://www.w3.org/TR/PNG/#1IHDR
IHDRtype := "IHDR" // IHDR chunk type
chunkd = IHDRtype;
chunkd += hton32b(64) // Width
chunkd += hton32b(64) // Hight
chunkd += string(1) // Bit depth, 1 for monochrome
chunkd += string(0) // Color type, 0 for Grayscale
chunkd += string(0) // Compression method, 0 for deflate/inflate
chunkd += string(0) // Filter method, 0 for adaptive
chunkd += string(0) // Interlace method, 0 for no interlace

// 5.3 CRC - for chunk type and chunk data
crc := crc32.ChecksumIEEE([]byte(chunkd))
fmt.Fprintf(os.Stderr, "CRC=%X\n", crc)
putNetInt32(file, uint32(len(chunkd)-4)) // chunk data length, excluding type filed
file.Write([]byte(chunkd))
putNetInt32(file, crc)

//----- PLTE
// 11.2.2.3 PLTE Pallete
// https://www.w3.org/TR/PNG/#1PLTE
PLTEtype := "PLTE"
chunkd = PLTEtype;
// empty pallete

crc = crc32.ChecksumIEEE([]byte(chunkd))
fmt.Fprintf(os.Stderr, "CRC=%X\n", crc)
putNetInt32(file, uint32(len(chunkd)-4)) // chunk data length, excluding type filed
file.Write([]byte(chunkd))
putNetInt32(file, crc)

//----- IDAT
// 11.2.2.4 IDAT Image data
IDATtype := "IDAT"
chunkd = IDATtype;
// empty data

crc = crc32.ChecksumIEEE([]byte(chunkd))
fmt.Fprintf(os.Stderr, "CRC=%X\n", crc)
putNetInt32(file, uint32(len(chunkd)-4)) // chunk data length, excluding type filed
file.Write([]byte(chunkd))
putNetInt32(file, crc)

//----- IEND
// 11.2.2.4 IEND Image trailer
IENDtype := "IEND"
chunkd = IENDtype;

crc = crc32.ChecksumIEEE([]byte(chunkd))
fmt.Fprintf(os.Stderr, "CRC=%X\n", crc)

```

```
putNetInt32(file,0) // chunk data length, excluding type filed
file.Write([]byte(chunkd))
putNetInt32(file,crc)

fmt.Fprintf(os.Stderr,"-- created %s (%d x %d)\n",outfile,genw,genh)
}
```

2020-0518 SatoxITS

