

# filetypes

operating systems & open source group

feb 12 2026

carston wiebe

# what are some filetypes?

## audio

- mp3
- flac
- wav
- m4a
- ogg
- opus

## image

- png
- jpeg
- avif
- heic
- svg
- webp
- gif

## video

- mp4
- mov

## document

- pdf
- html
- docx
- pptx
- xlsx
- gdoc
- gslides
- gsheet
- epub
- mobi

## container

- mkv
- zip
- tar
- riff

## program

- jar
- exe
- apk
- apkm
- apks

## text

- txt
- md
- adoc

## code

- c
- java
- ...

## data

- json

## ???

- xml

# common types

## riff

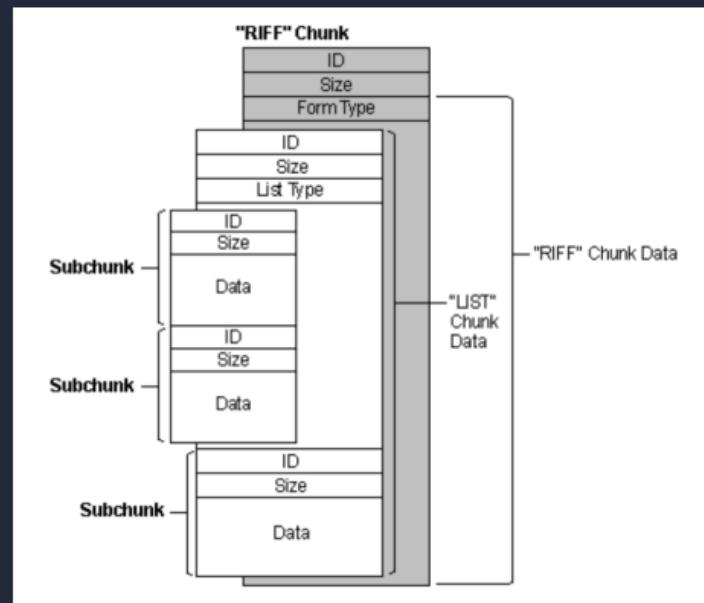
- resource interchange file format
- full of "chunks" of data (in bits)
- each chunk header contains the chunk's type and size

## xml

- full of "tags" which contain data (strings)
- tags open and close themselves and have attributes
- `<html><head></head><body></body></html>`

## zip

- compresses data using one of many algorithms
- primarily **deflate** (also FILETYPES : MD : FEB 11 2026 pngs!)



# how does your computer know a file's type?

- file extension

- easily fooled

- guess

- based on the file contents
  - not always accurate or possible

- type signatures

- also called magic numbers
  - opening x bits of the file

# inside a pdf

```
%PDF-1.7
1 %<9f><96>¤
2 6 0 obj
3 <</Filter /FlateDecode/Length 395>>
4 stream
5 xÚ<95><93>ÍjÃ0^LCi~
6 ¿ÀRI<96>ü^Ac<87>Ã^NÝ@<85><8c>^MÆ^Ne¬=u<90>îýaRêt; ]^[[ä8<96>¬^?¤_^Tô ä^Nö
F^"\üçþu^N<9a>$½ùôÐ<9b>; ·Ø`^C~÷£'Pb<8e>ÁO^_^N;g^Q<82>â^C±? | ^W÷}nÚ°µ<8e>@<8e>?Ùþ
e^ðØ°§¶÷aÉ^<99>=<82>@4Èuÿd0^_5µ^Ny<8c>
7 K¤2ÅJ^0v<99>, ^W <9a>^U^N,z-I
8 ..¤^X}»w<8b>ÓÛrð^ô^D¾Ý°÷{^@<8d>^CþÔ¹Ö<99>^A$éÔ¾<84>°<8f>^0^_¾}¶R-<8d><8b>^X,i<9
8>=^QÝ<82>a6jj<9d>Á ^X^RéÇo"!^Pä!%BÊ^S^N&<96>R<93>3<84><84>c^Nxå 4^Ytå#^Kc#^ë^
F^@<82>ñ" | bðÛy8ií^0¥<9e>^[[ää(ÖiÃ;ää>ÓÛÉ^_@æn<8c>^HØë'¾^ûlÐÄ8^C½(ÄBÖ{<8c><98
>¢<89>f^NÈçÐ{^-^L<97>|W^@^FÊ BÇ
9 ^D^U^F^Nk¾^F^0<9a>^VßÖ<83>³A^Sä^L<8c>^\<>84>
10 ^[ ^L<95>¢d ^X')Ã^D<86>ie^ ^H<83>k^GÉ|þ<91>
11 <82> K@+~=<8c>_F^R^R)
12 endstream
13 endobj
```

# inside a png

```
<89>PNG^M  
1 ^Z  
2 ^@^@^@^@^MIHDR^@^@^E^P^@^@^BJ^H^B^@^@^@^@^@^@^@^  
>iÝw|\W<99>7ðçÍ<9d>r§w<8d>È“÷nY“å<9a>ë  
ÜP^C<9b><84><92>p^`‘,÷"Ûê“zo#iz<9f>yÿ,  
ÏaRÍÅ^D^@^@^@^@^@^@^@^@^@^<97>^S~t^@^@^@^@^@^@^  
R^@^@^@^@^@^@^@^x'ó^FÄÑ/1Å^L^@^@^@^@^@^@^@^  
^@Ä<81><84>^Y^@^@^@^@^@^@^N$Ì^@^@^@^@^  
Ãá<81>á@óm<83>½<83>Ój¥$7ÓP]<96>@<94>s¢  
%T»!úv<9e>`ÐiööÙ o^M^N½2>6·y_oØR¤ÖÍ^_c  
gÆG?uÚD2^Mò~ºv]<9d>)í“Ö¬~ï<9d>^?Ý<9d>í  
Ä{7iT^Ä^[_Q^_¬<98>eµ^|\W@7þfÍúü@äE~^?Ü<§  
_ÍÍ_+7_#<99>ïG<81>åÖd<99>XÁ»^C^@^@^@^@  
<ë<8b>æžÖé÷^_^_æË2<91>[]@ÈT“^X<86>)E  
#¢2½ñ<87>^[_|èð~W(4÷^V^G<86>ú^Cñ<8e>^  
8b>Û^]spKz<98>e<89>hÍ`^?¾<8b>^@^@^@^@  
<88>@æöø¤<9c>8æZW ðÖ!óÑGíOMûdU^MÇ  
3 Ku<86>Û^MÆÝ<93><97>^M¢^N:ìÑð?<94><95>
```

- eight byte signature
- 0x89504e470d0a1a0a

0x89 : 0b10001001, can't be 7-bit

0x504e47 : ascii for png

0x0d0a : dos-style line ending

0x1a : dos-style end-of-file

0x0a : unix-style line ending

# files that lie to you

## how

- file extensions that don't match the file signature
- file extensions that do match, but take advantage of how a parser works

## why

- malicious intent
- completely normal reasons

## examples

- arbitrarily rename text files
- html vs xml
- svg vs xml
- wav vs riff
- jar vs zip
- gdoc vs json
- docx vs xml
- docx vs zip
- pptx vs xml
- pptx vs zip
- opf vs xml
- opf vs zip
- epub vs xml
- epub vs zip
- mobi vs xml
- mobi vs zip
- azw vs xml
- azw vs zip
- kf8 vs zip

# **everything is a zip file of xml**

- turns out making your file format is hard
- using an existing one is easy
- but, you don't want your files getting read by the wrong software
- the practice goes back to at least 1991 w/ wav
- most filetypes are like this

# real filetypes



- from matroska (pronounced matryoshka, the word for russian nesting doll)
- released 2002
- container for unlimited audio, video, images, and subtitles
- commonly used as output from video/audio creation software, and you then strip out what you need
- honorary zip file of xml, but for extensible binary meta language
- ebml is xml but binary, with opening/closing tags, created for mkvs
- arbitrary-length "tags" in binary with clever trick



# real filetypes

## mp3

- lossy compression format that commonly achieves 75%--95% reduction in size
- revolutionized music distribution
- designed by the moving picture experts group, developed largely in germany by the fraunhofer society
- released 1991
- objectively very cool

# real filetypes

## mp3 file structure

ID3v2x Metadata
MP3 Header
MP3 Data
MP3 Header
MP3 Data
+++ Repeated +++
MP3 Header
MP3 Data
MP3 Header
MP3 Data

## mp3 encoding

1. divide the audio into small, overlapping pieces
1. convert each piece into a sum of cos functions
1. perform the fourier transform on each piece
1. remove sounds humans can't hear
1. encode each piece according to the bitrate
1. format each piece into an mp3 header/data block

# math stuff

## ■ laplace

$$\mathcal{L}\{1\} = \frac{1}{s}$$

$$\mathcal{L}\{t\} = \frac{1}{s^2}$$

$$\mathcal{L}\{e^{at}\} = \frac{1}{s-a}$$

$$\mathcal{L}\left\{\frac{df(t)}{dt}\right\} = sF(s)$$

## ■ fourier

$$\mathcal{F}\{\cos(at)\} = \pi\delta_a + \pi\delta_{-a}$$

## ■ dct

- discrete cosine transform
  - type i : dct
  - type ii : idct
  - type iii : dst
  - **type iv : mdct**

## ■ fft

- fast fourier transform
- equivalent to discrete fourier transform
- mathematically the same, faster, and more accurate

# real filetypes



## png

- lossless compression format
- pronounced "ping" apparently
- created to be an improved version of gif (pronounced "jif" apparently)
- designed by the portable network graphics development group
- released 1996



# real filetypes

## ■ png file structure

- type signature: `0x89504e470d0a1a0a`
- made up of "chunks", like mp3s and riffs
- but png's headers are more fun
  - 4 bytes for size
  - 4 bytes for type
  - x bytes for data
  - 4 bytes for checksum

## ■ png headers

- chunk types are given as ascii words
  - `IHDR`
  - `PLTE`
  - `IDAT`
  - `IEND`
  - `bKGD`
  - `gAMA`
  - `eXIf`
- the case of each letter gives additional info
  - 1st = critical
  - 2nd = public
  - 3rd = reserved
  - 4th = dependent

# real filetypes

## ■ png compression

- uses `deflate`
  - `deflate` really likes zeros
- first uses one of a number of prediction methods
  - that number is 1
- called method 0
  - 1. do nothing
  - 1. `next = prev`
  - 1. `next = upper`
  - 1. `next = prev upper`
  - 1. `next = closest to prev + upper - prev upper`
- intended to get everything close to 0 for `deflates` benefit
- basically, take the derivative of the image



# closing

- most filetypes are fake
- basically just different ways of telling the computer to stop reading
- the real ones are normally for media
  - and use scary math like derivatives and transforms

in many ways they are very simple --- computers have to read them really fast, and complication costs a lot

**fin.**

any questions?

# activity

## ■ write a little parser

don't worry it's not that hard. trust

## ■ riff

```
id    : 4 bytes
size : 4 bytes (little-endian)
data  : size bytes
pad   : 1 or 0 bytes (total must be even)
```

if id is RIFF or LIST then the first 4 bytes of data will be a name, and the rest of data is more chunks

