File Formats

What is a File Format

- From Wikipedia:
  - A file format is a standard way that information is encoded for storage in a computer file. It specifies how bits are used to encode information in a digital storage medium. File formats may be either proprietary or free and may be either unpublished or open.

Aspects of Formats

- Proprietary or open?
- Text or Binary
- Single type of data or multiple types of data
- Fixed or extensible
- Independent from, but may be accompanied by a application programming interface (API)

Can make a huge difference in cost and performance

- Consider a single 10 digit integer
  - ~32 bits when represented in binary
  - 80 bits when represented as a text string
- Over a factor of two in performance and cost depending on representation.
- Cannot analyze data you cannot read!!

Types of Formats

- Raw
  - Direct representation of computer memory
- Text
  - Direct encoding of text characters
- Header file formats
  - Initial descriptive information about rest of file followed by data
- Chunked
  - File format constants one or more data “chunks” along with rule to describe and structure within chunk
- Directory
  - Hierarchal structuring of data

How do we know what format we have?

- File extension
  - Typically 3 characters: .txt, .jpg, .mov .mp3, ….
- Internal metadata
  - Header of Magic Number
  - Often at start of file, but can be elsewhere
- Attributes Managed by File System
  - OS X Uniform Type Identifiers (UTIs), OS/2 Extended Attributes, POSIX extended attributes
- External metadata
  - MIME types
  - Descriptor file
- File content based format identification

Text Files

- What is the format for a sequence of “text characters”
- Coding of characters:
  - What sequence of bits will be used to represent a text element?
  - ASCII: 1 byte character coding, Latin only

Code space & points

- Code space
  - A range of numerical values available for encoding characters
  - E.g., 0 to 10FFFF for Unicode, 0 to 7F for ASCII
- Code point
  - A value for a character in a code space

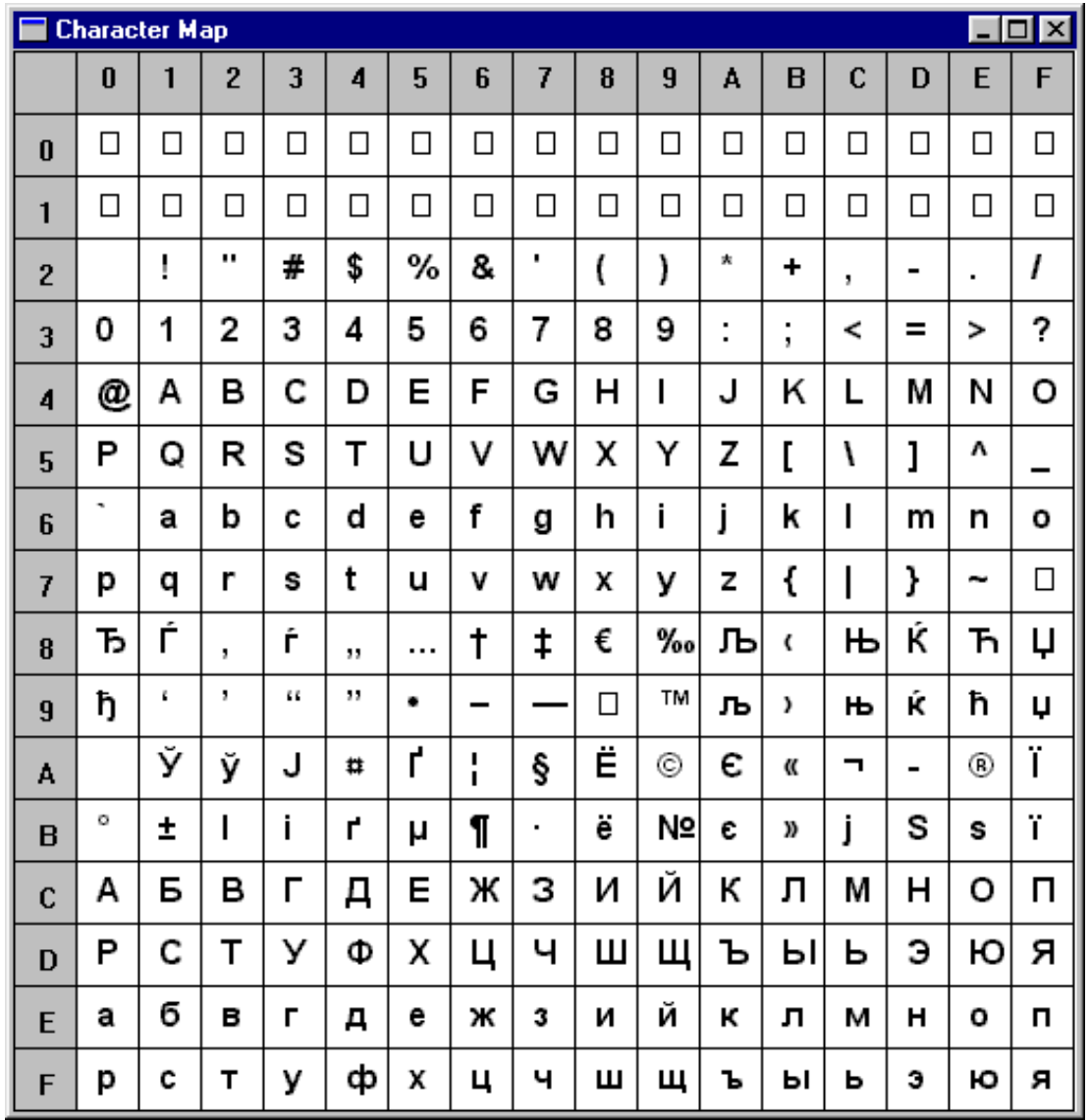
ASCII

Dec	Hex	Name	Char	Ctrl-char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	Null	NUL	CTRL-@	32	20	Space	64	40	@	96	60	`
1	1	Start of heading	SOH	CTRL-A	33	21	!	65	41	A	97	61	a
2	2	Start of text	STX	CTRL-B	34	22	"	66	42	B	98	62	b
3	3	End of text	ETX	CTRL-C	35	23	#	67	43	C	99	63	c
4	4	End of xmit	EOT	CTRL-D	36	24	$	68	44	D	100	64	d
5	5	Enquiry	ENQ	CTRL-E	37	25	%	69	45	E	101	65	e
6	6	Acknowledge	ACK	CTRL-F	38	26	&	70	46	F	102	66	f
7	7	Bell	BEL	CTRL-G	39	27	'	71	47	G	103	67	g
8	8	Backspace	BS	CTRL-H	40	28	(	72	48	H	104	68	h
9	9	Horizontal tab	HT	CTRL-I	41	29	)	73	49	I	105	69	i
10	0A	Line feed	LF	CTRL-J	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	VT	CTRL-K	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	FF	CTRL-L	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage feed	CR	CTRL-M	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	SO	CTRL-N	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	SI	CTRL-O	47	2F	/	79	4F	O	111	6F	o
16	10	Data line escape	DLE	CTRL-P	48	30	0	80	50	P	112	70	p
17	11	Device control 1	DC1	CTRL-Q	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	DC2	CTRL-R	50	32	2	82	52	R	114	72	r
19	13	Device control 3	DC3	CTRL-S	51	33	3	83	53	S	115	73	s
20	14	Device control 4	DC4	CTRL-T	52	34	4	84	54	T	116	74	t
21	15	Neg acknowledge	NAK	CTRL-U	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	SYN	CTRL-V	54	36	6	86	56	V	118	76	v
23	17	End of xmit block	ETB	CTRL-W	55	37	7	87	57	W	119	77	w
24	18	Cancel	CAN	CTRL-X	56	38	8	88	58	X	120	78	x
25	19	End of medium	EM	CTRL-Y	57	39	9	89	59	Y	121	79	y
26	1A	Substitute	SUB	CTRL-Z	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	ESC	CTRL-[	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	FS	CTRL-\	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	GS	CTRL-]	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	RS	CTRL-^	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	US	CTRL-_	63	3F	?	95	5F	_	127	7F	DEL

What about other languages…

- ANSI Standard --- use the 8th bit
  - Below 128, use ASCI
  - Use numbers from 128-255 for other language characters
  - Difference coding called a “code page”

Code Page 1251



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□
1	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	□
8	Ђ	Ѓ	‚	ѓ	„	…	†	‡	€	‰	Љ	‹	Њ	Ќ	Ћ	Џ
9	ђ	‘	’	“	”	•	–	—	□	™	љ	›	њ	ќ	ћ	џ
A		Ў	ў	Ј	¤	Ґ	¦	§	Ё	©	Є	«	¬	-	®	Ї
B	°	±	І	і	ґ	µ	¶	·	ё	№	є	»	ј	Ѕ	ѕ	ї
C	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
D	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
E	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
F	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я

Code Page 1253

Codepage 1253 - Greece Windows

	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
0-	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	000A	000B	000C	000D	000E	000F
1-	0010	0011	0012	0013	0014	0015	0016	0017	0018	0019	001A	001B	001C	001D	001E	001F
2-	0020	! 0021	" 0022	# 0023	$ 0024	% 0025	& 0026	' 0027	(0028	) 0029	* 002A	+ 002B	, 002C	- 002D	. 002E	/ 002F
3-	0 0030	1 0031	2 0032	3 0033	4 0034	5 0035	6 0036	7 0037	8 0038	9 0039	: 003A	; 003B	< 003C	= 003D	> 003E	? 003F
4-	@ 0040	A 0041	B 0042	C 0043	D 0044	E 0045	F 0046	G 0047	H 0048	I 0049	J 004A	K 004B	L 004C	M 004D	N 004E	O 004F
5-	P 0050	Q 0051	R 0052	S 0053	T 0054	U 0055	V 0056	W 0057	X 0058	Y 0059	Z 005A	[005B	\ 005C	] 005D	^ 005E	_ 005F
6-	` 0060	a 0061	b 0062	c 0063	d 0064	e 0065	f 0066	g 0067	h 0068	i 0069	j 006A	k 006B	l 006C	m 006D	n 006E	o 006F
7-	p 0070	q 0071	r 0072	s 0073	t 0074	u 0075	v 0076	w 0077	x 0078	y 0079	z 007A	{ 007B	| 007C	} 007D	~ 007E	007F
8-	€ 20AC	0081	‚ 201A	*f* 0192	„ 201E	… 2026	† 2020	‡ 2021	0088	‰ 2030	008A	‹ 2039	008C	008D	008E	008F
9-	0090	‘ 2018	’ 2019	“ 201C	” 201D	• 2022	– 2013	— 2014	0098	™ 2122	009A	› 203A	009C	009D	009E	009F
A-	00A0	΅ 0385	Ά 0386	£ 00A3	¤ 00A4	¥ 00A5	¦ 00A6	§ 00A7	¨ 00A8	© 00A9	ª 00AA	« 00AB	¬ 00AC	- 00AD	® 00AE	― 2015
B-	° 00B0	± 00B1	² 00B2	³ 00B3	΄ 0384	µ 00B5	¶ 00B6	· 00B7	Έ 0388	Ή 0389	Ί 038A	» 00BB	Ό 038C	½ 00BD	Ύ 038E	Ώ 038F
C-	ΐ 0390	Α 0391	Β 0392	Γ 0393	Δ 0394	Ε 0395	Ζ 0396	Η 0397	Θ 0398	Ι 0399	Κ 039A	Λ 039B	Μ 039C	Ν 039D	Ξ 039E	Ο 039F
D-	Π 03A0	Ρ 03A1		Σ 03A3	Τ 03A4	Υ 03A5	Φ 03A6	Χ 03A7	Ψ 03A8	Ω 03A9	Ϊ 03AA	Ϋ 03AB	ά 03AC	έ 03AD	ή 03AE	ί 03AF
E-	ΰ 03B0	α 03B1	β 03B2	γ 03B3	δ 03B4	ε 03B5	ζ 03B6	η 03B7	ϑ 03B8	ι 03B9	κ 03BA	λ 03BB	μ 03BC	ν 03BD	ξ 03BE	ο 03BF
F-	π 03C0	ρ 03C1	ς 03C2	σ 03C3	τ 03C4	υ 03C5	φ 03C6	χ 03C7	ψ 03C8	ω 03C9	ϊ 03CA	ϋ 03CB	ό 03CC	ύ 03CD	ώ 03CE	

What about?

- Asian languages that don’t fit into 256 characters?
- Multi-language text
- Ancient languages (hieroglyphics), emoji, …??

Unicode

- Abstract character set
  - Capitol “A”, Ω, ….
  - Separate character from glyph
- Defining what is the abstract character space can be tricky
  - German ss vs ß
  - Letters that change shape at end of a word are different letters in Hebrew, but the same letters in Arabic

Unicode

- Define mapping of abstract character into 20 bit “code point”
  - 1,114,112 code points in the range 0 to 10FFFF
- Unicode code point indicated by U+ hexidecimal number
- Example:
  - “e” U+0065 (LATIN SMALL LETTER E)
  - “é” U+0065 U+0301
  - “é” U+00E9 (LATIN SMALL LETTER E WITH ACUTE).

Unicode

- Code space is divided into 17 planes
- Each plan = contiguous 2^{16} code points
- Recall that code points range from 0 to 10FFFF

$\Rightarrow$Total code points = 17 * 2^{16} or 1,114,112 code points

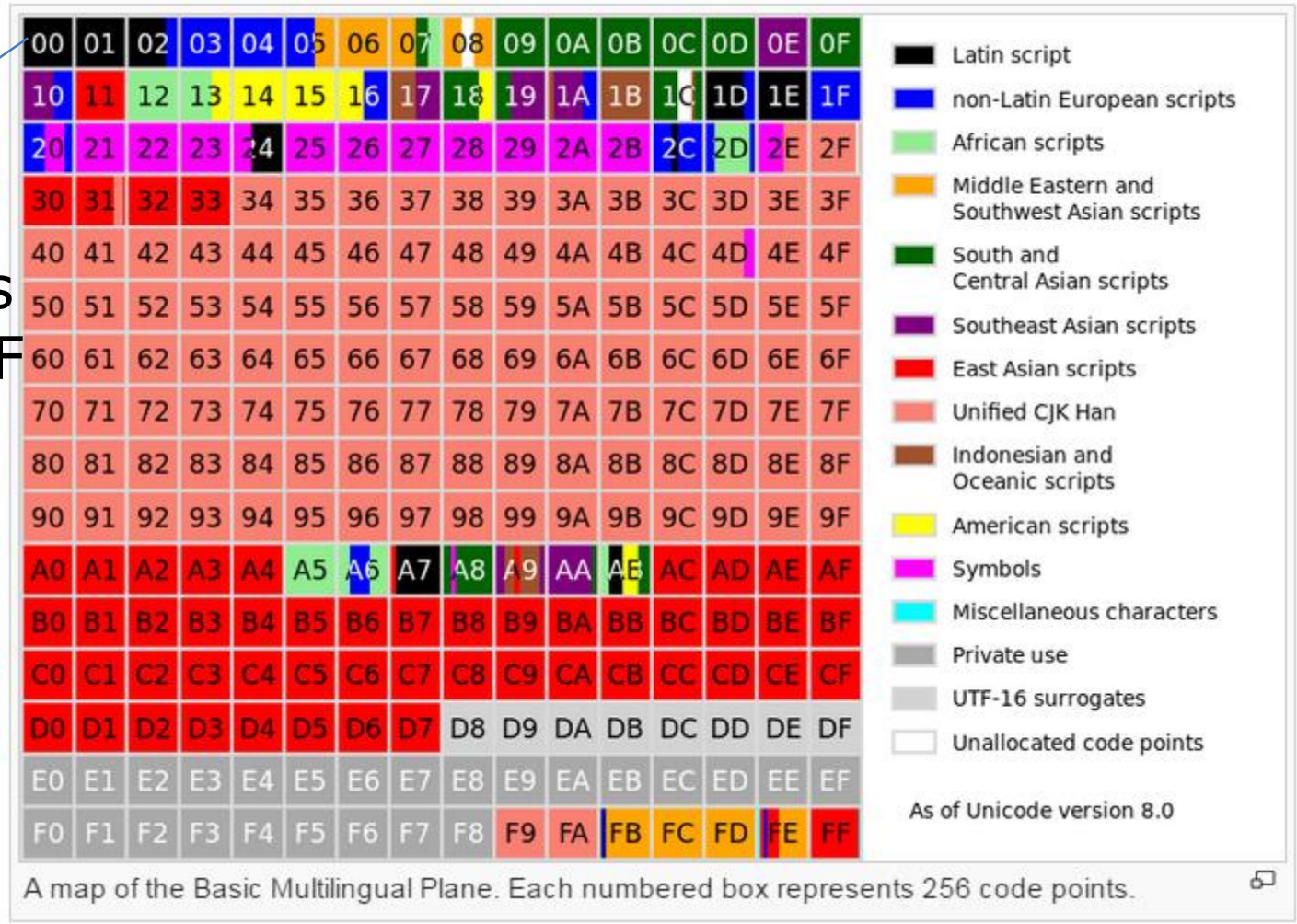
Note 2^{16} = 65,536

Planes in Unicode

V·T·E	Unicode planes and used code point ranges											[hide]
Basic		Supplementary										
Plane 0		Plane 1		Plane 2		Planes 3–13		Plane 14		Planes 15–16		
0000–FFFF		10000–1FFFF		20000–2FFFF		30000–DFFFF		E0000–EFFFF		F0000–10FFFF		
Basic Multilingual Plane		Supplementary Multilingual Plane		Supplementary Ideographic Plane		*unassigned*		Supplementary Special-purpose Plane		Supplementary Private Use Area		
BMP		SMP		SIP		—		SSP		S PUA A/B		
0000–0FFF	8000–8FFF	10000–10FFF		20000–20FFF	28000–28FFF			E0000–E0FFF		*15: PUA-A* F0000–FFFFF		
1000–1FFF	9000–9FFF	11000–11FFF		21000–21FFF	29000–29FFF							
2000–2FFF	A000–AFFF	12000–12FFF		22000–22FFF	2A000–2AFFF							
3000–3FFF	B000–BFFF	13000–13FFF	1B000–1BFFF	23000–23FFF	2B000–2BFFF					*16: PUA-B* 100000–10FFFF		
4000–4FFF	C000–CFFF	14000–14FFF		24000–24FFF	2C000–2CFFF							
5000–5FFF	D000–DFFF		1D000–1DFFF	25000–25FFF								
6000–6FFF	E000–EFFF	16000–16FFF	1E000–1EFFF	26000–26FFF								
7000–7FFF	F000–FFFF		1F000–1FFFF	27000–27FFF	2F000–2FFFF							

Plane 0: BMP (Basic Multilingual Plane)

Block 00
Represents
0000~00FF



Each block represents 256 code points

Hello

- U+0048 U+0065 U+006C U+006C U+006F.

Unicode Encoding

- How to map a code point into bits?
  - Number of bits needed will depend on codepoint values
- Would be nice to have backward compatibility with ANSI
- What about code points that require 2 or 3 bytes?

Encoding (of code points)

- Code unit: the smallest unit (comprising a number of bits) used to construct an encoding for a code point
  - Code unit for UTF-8: 8-bit
  - UTF-16:16-bit

- UTF (Unicode Transformation Format) encoding
  - E.g., UTF-8 and UTF-16

Variable-length encoding

- Characters encoded using codes of different length
- In Unicode, a code point may be represented using multiple code units
  - E.g., 1-4 in UTF-8, 1-2 in UTF-16

UTF-8

- Encoding scheme for Unicode code space
- Code unit = 8 bits
- Variable length
  - Code point may be represented using 1-4 code units

UTF-8 Design

- This shows the original design
  - Current: only up to U+10FFFF code points used
  - So no 5-byte/6-byte sequences

Bits of code point	First code point	Last code point	Bytes in sequence	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
7	U+0000	U+007F	1	0xxxxxxx					
11	U+0080	U+07FF	2	110xxxxx	10xxxxxx				
16	U+0800	U+FFFF	3	1110xxxx	10xxxxxx	10xxxxxx			
21	U+10000	U+1FFFFF	4	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx		
26	U+200000	U+3FFFFFF	5	111110xx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	
31	U+4000000	U+7FFFFFFF	6	1111110x	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx

UTF-8 Features

- Backward compatibility
  - One byte for ASCII, leading bit of byte is zero
- Clear distinction btw single- vs. multi-byte characters
  - Single-byte/multi-byte: start with 0/1 respectively
- Multiple length
  - a leading byte starts with 2 or more 1's, followed by a 0, e.g., '110', '1110', etc.
  - One or more continuation bytes all start with '10'

UTF-8 Features

- Clear indication of code sequence length
  - By # of 1's in leading byte (for multi-byte)
- Self-synchronization
  - Can find start of characters by backing up at most 3 bytes (5 in original design)

Example

- Encode ‘€’ using UTF-8
- Code point = U+20AC
- Need 3 bytes in UTF-8

Character		Binary code point	Binary UTF-8	Hexadecimal UTF-8
$	U+0024	0100100	00100100	24
¢	U+00A2	00010100010	11000010 10100010	C2 A2
€	U+20AC	0010000010101100	11100010 10000010 10101100	E2 82 AC
𐍈	U+10348	00001000000110100 1000	11110000 10010000 10001101 10001000	F0 90 8D 88

UTF-16

- Code unit = 16 bits
- Variable-length encoding
  - Code point = one/two code units
- Not compatible with ASCII

UTF-16

- Plane 0: encoded using one code unit: 16 bit
- Rest: two code units

V·T·E	Unicode planes and used code point ranges											[hide]
Basic		Supplementary										
Plane 0		Plane 1		Plane 2		Planes 3–13		Plane 14		Planes 15–16		
0000–FFFF		10000–1FFFF		20000–2FFFF		30000–DFFFF		E0000–EFFFF		F0000–10FFFF		
Basic Multilingual Plane		Supplementary Multilingual Plane		Supplementary Ideographic Plane		*unassigned*		Supplementary Special-purpose Plane		Supplementary Private Use Area		
BMP		SMP		SIP		—		SSP		S PUA A/B		
0000–0FFF	8000–8FFF	10000–10FFF		20000–20FFF	28000–28FFF			E0000–E0FFF		*15: PUA-A* F0000–FFFFF		
1000–1FFF	9000–9FFF	11000–11FFF		21000–21FFF	29000–29FFF							
2000–2FFF	A000–AFFF	12000–12FFF		22000–22FFF	2A000–2AFFF							
3000–3FFF	B000–BFFF	13000–13FFF	1B000–1BFFF	23000–23FFF	2B000–2BFFF					*16: PUA-B* 100000–10FFFF		
4000–4FFF	C000–CFFF	14000–14FFF		24000–24FFF	2C000–2CFFF							
5000–5FFF	D000–DFFF		1D000–1DFFF	25000–25FFF								
6000–6FFF	E000–EFFF	16000–16FFF	1E000–1EFFF	26000–26FFF								
7000–7FFF	F000–FFFF		1F000–1FFFF	27000–27FFF	2F000–2FFFF							

UTF-16

- U+0000 to U+D7FF and U+E000 to U+FFFF
  - Represent directly as 16 bit number
- U+D800 to U+DFFF
  - Left open by Unicode standard
- U+010000 to U+10FFFF
  - Subtract 0x10000 to make 20 bit number (0 .. 0x0FFFFF)
  - Code into 2 16 bit numbers
    - Add 0xD800 to top 10 bits (0xD800 .. 0xDBFF)
    - Add 0xDC00 to bottom 10 bits (0xDC00 .. 0xDFFF)

Examples

- Encoding code points in BMP is easy

Character		Binary code point	Binary UTF-16	UTF-16 hex code units	UTF-16BE hex bytes	UTF-16LE hex bytes
$	U+0024	0000 0000 0010 0100	0000 0000 0010 0100	0024	00 24	24 00
€	U+20AC	0010 0000 1010 1100	0010 0000 1010 1100	20AC	20 AC	AC 20
𐐷	U+10437	0001 0000 0100 0011 0111	1101 1000 0000 0001 1101 1100 0011 0111	D801 DC37	D8 01 DC 37	01 D8 37 DC
𤭢	U+24B62	0010 0100 1011 0110 0010	1101 1000 0101 0010 1101 1111 0110 0010	D852 DF62	D8 52 DF 62	52 D8 62 DF

Byte Order

- What order to we place the bytes in 16 bits?
  - A B C D
- Big Endian: A B C D
- Little Endian: B A D C
- Unicode recommends to prepend a **Byte Order Mark** (BOM) to the string, representing the character U+FEFF.
  - FE, FF, the encoding is UTF-16BE.
  - FF, FE, it is UTF-16LE.

Gulliver’s Travels

Besides, our Histories of six thousand Moons make no mention of any other Regions, than the two great Empires of Lilliput and Blefuscu. Which two mighty Powers have, as I was going to tell you, been engaged in a most obstinate War for six and thirty Moons past.

It began upon the following Occasion. It is allowed on all Hands, that the primitive way of breaking Eggs, before we eat them, was upon the larger End: But his present Majesty's Grand-father, while he was a Boy, going to eat an Egg, and breaking it according to the ancient Practice, happened to cut one of his Fingers. Whereupon the Emperor his Father published an Edict, commanding all his Subjects, upon great Penaltys, to break the smaller End of their Eggs.

The People so highly resented this Law, that our Histories tell us there have been six Rebellions raised on that account; wherein one Emperor lost his Life, and another his Crown. These civil Commotions were constantly fomented by the Monarchs of Blefuscu; and when they were quelled, the Exiles always fled for Refuge to that Empire. It is computed, that eleven thousand Persons have, at several times, suffered Death, rather than submit to break their Eggs at the smaller End.

Many hundred large Volumes have been published upon this Controversy: But the books of the Big-Endians have been long forbidden, and the whole Party rendered incapable by Law of holding Employments. During the Course of these Troubles, the Emperors of Blefuscu did frequently expostulate by their Ambassadors, accusing us of making a Schism in Religion, by offending against a fundamental Doctrine of our great Prophet Lustrog, in the fifty-fourth Chapter of the Brundrecal (which is their Alcoran.) This, however, is thought to be a meer Strain upon the Text: For the Words are these: That all true Believers shall break their Eggs at the convenient End: and which is the convenient End, seems, in my humble Opinion, to be left to every Man's Conscience, or at least in the power of the Chief Magistrate to determine.

Now the Big-Endian Exiles have found so much Credit in the Emperor of Blefuscu's Court, and so much private Assistance and Encouragement from their Party here at home, that a bloody War has been carried on between the two Empires for six and thirty Moons with various Success; during which time we have lost forty Capital Ships, and a much greater number of smaller Vessels, together with thirty thousand of our best Seamen and Soldiers; and the Damage received by the Enemy is reckon'd to be somewhat greater than Ours. However, they have now equipped a numerous Fleet, and are just preparing to make a Descent upon us; and his Imperial Majesty, placing great Confidence in your Valour and Strength, has commanded me to lay this Account of his affairs before you.

"Example" in different encodings (UTF-16 with BOM):

```
ASCII:      45 78 61 6d 70 6c 65
UTF-16BE:   FE FF 00 45 00 78 00 61 00 6d 00 70 00 6c 00 65
UTF-16LE:   FF FE 45 00 78 00 61 00 6d 00 70 00 6c 00 65 00
```

Unicode Text Encoding Examples

Character	Code Point	UTF-16	UTF-8
a	U+0061	0061	61
ä	U+00E4	00E4	C3 A0
σ	U+03C3	03C3	CF 83
א	U+05D0	05D0	D7 90
٣	U+0663	0663	D9 A3
力	U+30AB	30AB	E3 82 AB
退	U+9000	9000	E9 80 80
𠀁	U+21BC1	D846 DFC1	F0 A1 AF 81

If you don’t know the encoding, you cannot use the data !!

Emoji: U+1F36D LOLLIPOP, U+1F36E CUSTARD, U+1F36F HONEY POT, and U+1F370 SHORTCAKE

A type writer


TELETYPE
DATA DYNAMICS

What about new line (end of line)?

- LF (Line feed, '\n', 0x0A, 10 decimal)
- CR (Carriage return, '\r', 0x0D, 13 in decimal)
- Different systems represent newline differently
  - Window: \r\n
  - Unix based: \n
  - Old Mac: \r

Unicode

Unicode standard defines a number of characters that conforming applications should recognize as line terminators:

- LF: Line Feed, U+000A
- VT: Vertical Tab, U+000B
- FF: Form Feed, U+000C
- CR: Carriage Return, U+000D
- CR+LF: CR (U+000D) followed by LF (U+000A)
- NEL: Next Line, U+0085
- LS: Line Separator, U+2028
- PS: Paragraph Separator, U+2029

Comma Separated Values

- Model
  - Plain text file using a character set
  - Set of records (one per line)
  - Separated into fields by reserved character
    - Often a comma
- every record has the same sequence of fields.

Here is no CSV standard

- What about:
  - Values with commas in them
  - Values with new lines in them
  - Character coding?
  - Whitespace
  - Header or not?

RFC 4180: MIME type “text/csv”

- DOS-style lines that end with (CR/LF) character
  - optional for the last line
- An optional header record
  - How do we know if there is a header?
- Each record "should" contain the same number of comma-separated fields
- Whitespace is preserved
- Any field *may* be quoted (with double quotes).
- Fields containing a line-break, double-quote, and/or commas *should* be quoted.
- A (double) quote character in a field *must* be represented by two (double) quote characters.

Example

Name	Favorate	Cost	Notes
carl@isi.edu	Pliny the elder	> 8	Hoppy, hoppy, hoppy Beer advocate “best beer”
Flintstone, Fred	Sculpin	3	It’s a good beer
August Busch	Budvar	$2	SVĚTLÝ LEŽÁK

carl@isi.edu,Pliny the elder,> 8,"Hoppy, hoppy, hoppy"
,,,Beer advocate ôbest beerö
"Flintstone, Fred",Sculpin,3,ItÆs a good beer
August Busch,Budvar,$2 ,SV_TL LEÄ┴K

Example

Name	Favorate	Cost	Notes
carl@isi.edu	Pliny the elder	> 8	Hoppy, hoppy, hoppy Beer advocate “best beer”
Flintstone, Fred	Sculpin	3	It’s a good beer
August Busch	Budvar	$2	SVĚTLÝ LEŽÁK

In textedit:

carl@isi.edu,Pliny the elder,> 8,"Hoppy, hoppy, hoppy"
,,,Beer advocate ôbest beerö
"Flintstone, Fred",Sculpin,3,ItÆs a good beer
August Busch,Budvar,$2 ,SV_TL▌ LEÄ┴K

In a “smarter” editor:

carl@isi.edu,Pliny the elder,> 8,"Hoppy, hoppy, hoppy"
,,,Beer advocate \223best beer\224
"Flintstone, Fred",Sculpin,3,It\222s a good beer
August Busch,Budvar,$2 ,SV_TL\335 LE\216\301K

File formats with a File Header

- Identification bytes (magic number)
- Header checksum
- Version number
- Offset to data

Portable Gray Map

- Header (fields separated by whitespace (blank, TAB, CR, LF).
  - magic number identifying the file type: "P5".
  - A width, formatted as ASCII characters in decimal.
  - A height, again in ASCII decimal.
  - maximum gray value again in ASCII decimal.
- A raster
  - Height rows, width columns
  - 0 black, max gray is white
  - one byte if Val < less than 256, 2 bytes if > 256, big endian

PGM Example

```
P2
# feep.pgm
24 7
15
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  3  3  3  3  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0 15  0
0  3  3  3  0  0  0  7  7  7  0  0  0 11 11 11  0  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0  0  0
0  3  0  0  0  0  0  7  7  7  7  0  0 11  0  0  0  0  0 15  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

0x31
0x35
0x20

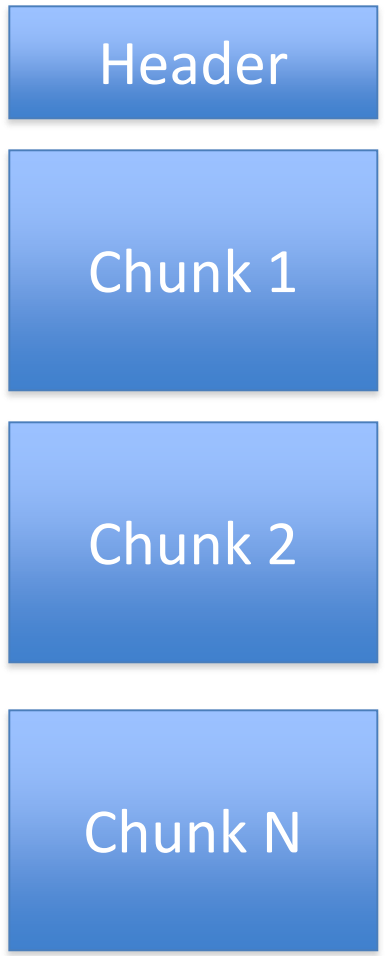
PGM Example (Binary)

```
P4
# feep.pgm
24 7
15
000000000000000000000000
033330077770 0AAAA00FFFF0
030000070000 0A0000F00F0
033300077700 0AAA000FFFF0
030000070000 0A0000F0000
030000077770 0AAAA00F0000
000000000000000000000000
```

24x7 bytes

Portable Network Graphics (PNG)

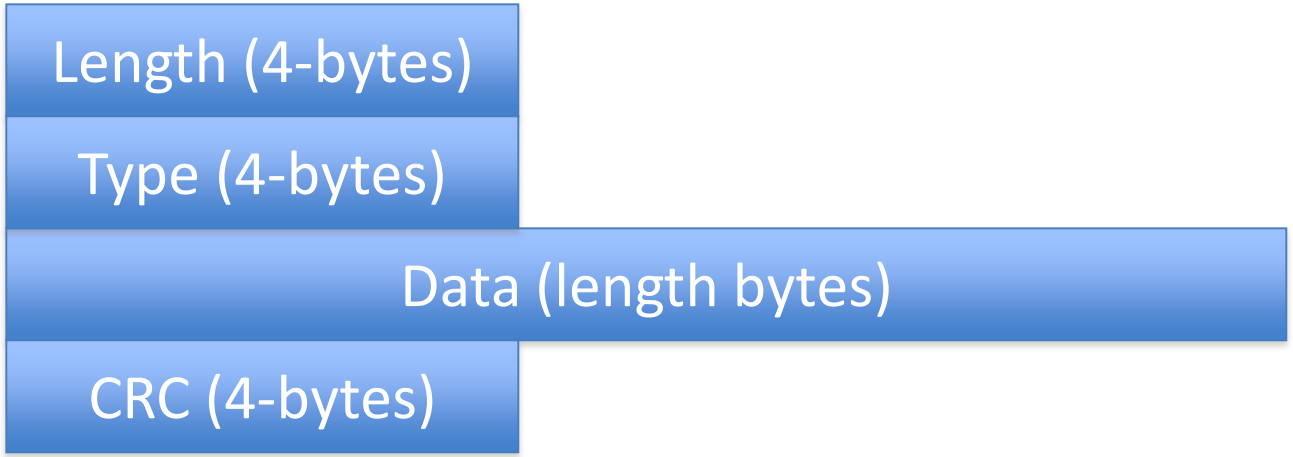
- Chunk based file format
- Header
- Two types of chunks
  - Critical (must be able to decode)
  - Ancillary (not required to decode)



PNG File Header

Bytes	Purpose
89	Has the high bit set to detect transmission systems that do not support 8 bit data and to reduce the chance that a text file is mistakenly interpreted as a PNG, or vice versa.
50 4E 47	ASCI letters PNG
0D 0A	CR/LF to detect conversion of the data
1A	End of file byte that stops display of file under DOS
0A	LF to detect unix coding

PNG Chunk Format



Critical Chunks

- IHDR
  - must be the first chunk; it contains the image's width, height, color type and bit depth.
- PLTE
  - contains the palette; list of colors.
- IDAT
  - contains the image, which may be split among multiple IDAT chunks.
- IEND marks the image end.

IHDR

- The IHDR chunk must appear FIRST. It contains:
  - Width: 4 bytes
  - Height: 4 bytes
  - Bit depth: 1 byte
  - Color type: 1 byte
  - Compression method: 1 byte
  - Filter method: 1 byte
  - Interlace method: 1 byte

Previous Example in PNG

- Width: 24
- Height: 7
- Bit depth: 4
- Color type: 0
- Compression method: 0
- Filter method: 0
- Interlace method: 0

MP3

- Store audio coded with specific compression
  - Like PNG in that manner
- Structure
  - File consists of a set of chunks (called frames).
  - Each frame has a header, followed by frame data
    - Bitrate may change between frames
    - Coding allows data to span frames
  - Descriptive metadata may be at beginning or end of file
    - Use ID3 tagging

ID3 Tagging

- “Trick” MP3 into having metadata
  - Ignored by older players
- Defines a chunk based format for extensible

ID3v2 Format

Header Format:
ID3v2/file identifier "ID3”
ID3v2 version 0x03 00
ID3v2 flags abc00000
ID3v2 size 4 * 0xxxxxxx

Frame Format:
Frame ID xx xx xx xx
Size xx xx xx xx
Flags xx xx
Frame Data.....

Example ID3 Frames

- **TCOM**
  - The 'Composer(s)' frame is intended for the name of the composer(s). They are seperated with the "/" character.
- **TFLT**
  - The 'File type' frame indicates which type of audio this tag defines, eg. MPEG/1, MPEG/2, MPEG/3
- **TIT1**
  - The 'Content group description' frame is used if the sound belongs to a larger category of sounds/music. For example, classical music is often sorted in different musical sections (e.g. "Piano Concerto", "Weather - Hurricane").
- **TIT2**
  - The 'Title/Songname/Content description' frame is the actual name of the piece (e.g. "Adagio", "Hurricane Donna").
- **TIT3**
  - The 'Subtitle/Description refinement' frame is used for information directly related to the contents title (e.g. "Op. 16" or "Performed live at Wembley").

ZIP Files

- Archive format
  - Single container for multiple files
  - Support for lossless compression
  - Easy modification/update

ZIP File Structure