Multilingualism on the Web

Pascal Vaillant <vaillant@univ-paris13.fr>

Writing systems

Writing systems

- Mankind has been using speech for …
  as long as it deserves to be called human

  (definitory statement)

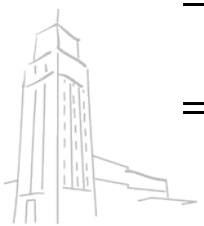
  e.g. 150 000 – 50 000 years (very approx)
- It has been using writing since it has become organized in urban societies

  e.g. 5 000 years BP (approx)

Membre fondateur de :

Writing systems

- Urban centres
  - ⇒ specialization of economic units
  - ⇒ currency
  - ⇒ a central authority to control and organize
  - ⇒ state and civil servants
  - ⇒ taxes
  - ⇒ accountancy
  - ⇒ **counting** and **writing**



Membre fondateur de :

Development of writing systems

- Highly probable origin: iconic (pictograms)
- Examples (from Chinese):

water: 水 (shuǐ)　　*field*: 田 (tián)

mountain: 山 (shān)　　*grass*: 艸 (cǎo)

fire: 火 (huǒ)　　*beast*: 豸 (zhì)

horse: 馬 (mǎo)　　*ox*: 牛 (niú)

Development of writing systems

- Combination → ideograms
- Example (from Chinese):

field: 田 (tián)

grass: 艸 (cǎo)

sprout: 苗 (miáo)

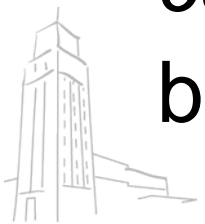
Development of writing systems

- Rebus → ideophonograms
- Example (from Chinese):

  ten thousands: 萬 (wàn) (orig. scorpion)

  sprout: 苗 (miáo)

  cat: 貓 (māo) (something pronounced a bit like “miáo”, but actually a beast)



Development of writing systems

- Adoption of a logographic writing system by other nations
- The same derivations (ideographic compounds, phonetic and semantic complements) continue to develop after the transfer
- But in addition, the phonetic component may have two different values

Development of writing systems

- Adoption of a logographic writing system by other nations :
  - Sumerian writing by the Akkadian ; later by the Hittite, the Assyrian, the Elamite and the Persian (cuneiform writing)
  - Egyptian « hieroglyphs » by ancient peoples in Sinai or Palestine, and by the Phoenician
  - Chinese by all surrounding nations (Vietnam, Korea, Japan)

Development of writing systems

- Adoption by other nations
- Examples (from Japanese):

water: 水 (mizu *or* sui (in compounds))

mountain: 山 (yama *or* san)

fire: 火 (hi *or* ka)

horse: 馬 (uma *or* ba)

Development of writing systems

- Trimming the list of phonograms
  → syllabic writing system
- Examples (from Japanese *katakana*):

加 (ka [mod. chin. jiā]) → カ (ka)

毛 (mô [mod. chin. máo]) → モ (mo)

久 (kyû [mod. chin. jiǔ]) → ク (ku)

Development of writing systems

- The original pictograms evolve in meaning (by combination or semantic drift), but also in shape
- This is often due to technical constraints : silk brush on paper (chinese *kǎishū*), iron point on soft clay (cuneiforms), reed pen on papyrus (egyptian demotic)
- Stylization, loss in iconicity

Development of writing systems

- Decomposing syllables in sounds

  → phonemic writing system
- Korean (hangul)

  대한민국
- Brahmic family (e.g. devanagari)

  नई दिल्ली

Development of writing systems

- Using the initial consonant of syllables
  → consonantic writing system
- Semitic scripts (Phoenican, Hebrew, Arab)

	(alf)	א (aleph)	ا (alif)
	(bet)	ב (beth)	ب (bā)
	(gaml)	ג (gimel)	ج (ǧīm)
	(delt)	ד (dalet)	د (dāl)

Membre fondateur de :

Development of writing systems

- It is not unlikely that the shapes of the semitic letters derive from an original syllabic rebus
- But it cannot be proven with certainty

	ʔalf	*ox*
	ʕain	*eye*

Development of writing systems

- Adding vocals to consonantic scripts

  → phonemic writing system (again)
- Greek, Latin, Cyrillic ...

  τὸ δὲ σύμπαν φῦλον, ὃ νῦν Γαλλικόν τε καὶ Γαλατικὸν καλοῦσιν, ἀρειμάνιόν ἐστι καὶ θυμικόν τε καὶ ταχὺ πρὸς μάχην, ἄλλως δὲ ἁπλοῦν καὶ οὐ κακόηθες.

Development of writing systems

- Adapting consonantic script to non-semitic languages → exaptation of some symbols
- Phoenician Greek/Latin


(ʔalf) A


(he) E

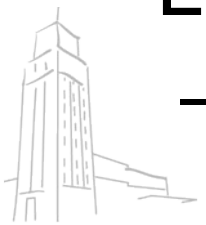

(ʕain) O

Actual writing systems

- Phonemic scripts
  - linear positioning of phonemes : alphabetic (latin, greek, cyrillic)
  - linear positioning of consonants, vowels noted as “diacritics” (can be above or below)
    - *compulsorily*: thai, devanagari …
    - *optionally*: hebrew, arab …
  - linear positioning of syllables : alpha-syllabic (hangul)

Actual writing systems

- Phonemic scripts
  - 1 graphic symbol = 1 phoneme
    (k – a – z – a – n)
- Syllabic scripts
  - 1 graphic symbol = 1 syllable
    (かざん / ka – za – n)
- Logographic scripts
  - 1 graphic symbol = 1 morpheme
    (火山 / Fire – Mountain)



Consequences on encoding

- Phonemic scripts: number of phonemes
  - counted in a few tens (typically 25 to 40)
- Syllabic scripts: number of syllables
  - likely closer to a hundred (71 for Japanese)
- Logographic scripts: number of morphemes
  - counted in thousands

  (standard dictionary of Chinese : 6500 characters to account for 50000 words or expressions)

Writing units

What is to be encoded

- A text, written in a human language, by use of a human writing systems, is a sequence of visual forms (*glyphs*) representing abstract graphic symbols (*graphemes*, or more commonly *characters*)
- A text is the production of a linear writing system (one dimension). Pictography in general (two-dimensional) is something else.

Membre fondateur de :
SORBONNE PARIS CITÉ
CAMPUS CONDORCET Paris-Aubervilliers

Written language *vs.* speech

- Written language obviously is not exactly the same thing as spoken language
- Much heated debate on the autonomy of scripture w.r.t. speech
- But, to the least, linearity on both sides allows at least partial transcription operations (reading / writing)
- Prosody generally left out (but see TEI ch 8)

Glyphs vs. Characters

- One character may yield several glyphs

Glyphs vs. Characters

- Generally speaking, characters are what you need to identify the words
- Glyph choice is decorative but also meaningful (by “connotation”: you do not use italics with the same function as roman)
- Identifying the character is generally considered to be part of *encoding*
- Choosing the glyph part of *typesetting*

Membre fondateur de :
SORBONNE PARIS CITÉ
CAMPUS CONDORCET Paris-Aubervilliers

Glyphs vs. Characters

- Decorate : chinese seal script

Glyphs vs. Characters

- Emphasize or quote
  - Japanese katakana
  - Latin or cyrillic italics

‘Look at your hands, and look at your mouth. What *is* that truck?’
‘*I* don’t know, Aunt.’
‘Well, *I* know. It’s jam, that’s what it is.’

Glyphs vs. Characters

- In some cases, the choice of the glyph is prescribed by rigid contextual rules (arabic)

Final	*Middle*	*Initial*	*Isolated*
ـه	ـهـ	هـ	ه
ـص	ـصـ	صـ	ص
ـغ	ـغـ	غـ	غ

Encoding

Encoding : basics

- Digital memory devices are based on small elementary cells carrying two possible values (0 or 1) : bits
- Encoding a text is representing its sequence of characters, without loss of information, onto a digital medium
- i.e. onto a sequence of bits (possibly grouped together)

Encoding : basics

- What is to be encoded is a sequence of *characters*, not *glyphs*
- First, correctly encode the character
- Then choose the character form
- In some cases it can be automatically done considering the context (arabic shapes)
- In other cases: the realm of typesetting

Encoding : basics

- To represent the *n* different symbols of an alphabet, you need an information block than may represent more than *n* different values
- One bit : 2 values
- 2 bits : 4 values (00, 01, 10, 11)
- 8 bits : 256 (2^8) different values
- 16 bits : 65536 (2^{16}) different values ...

Membre fondateur de :
SORBONNE PARIS CITÉ
CAMPUS CONDORCET Paris-Aubervilliers

Encoding : basics

- With n bits, you can encode 2^n different codes:

00000000 ↦ 0
00000001 ↦ 1
00000010 ↦ 2
00000011 ↦ 3
00000100 ↦ 4
...
11111110 ↦ 254
11111111 ↦ 255

Hexadecimal notation :

0000: 0	1000: 8
0001: 1	1001: 9
0010: 2	1010: A
0011: 3	1011: B
0100: 4	1100: C
0101: 5	1101: D
0110: 6	1110: E
0111: 7	1111: F

Membre fondateur de :

Encoding : basics

- A character set Σ is a list of characters
- It may be ordered (A<Z, Ж<Щ, 丈 < 龜 ...)
- An encoding is a function $\mathbb{N} \rightarrow \Sigma$

  … i.e. a *numbered* list of characters
- Ex. ASCII : 65 (bin: 01000001; hex: 41) ↦ A

  66 (bin: 01000010; hex: 42) ↦ B

  ...

  90 (bin: 01011010; hex: 5A) ↦ Z

Encoding : basics

- The *code point* is the number

65 (bin: 01000001; hex: 41) ↦ A

66 (bin: 01000010; hex: 42) ↦ B

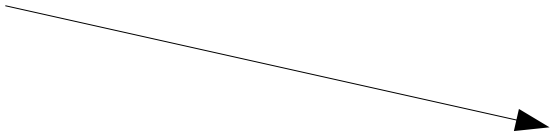
...

90 (bin: 01011010; hex: 5A) ↦ Z

Membre fondateur de :

Encoding : basics

- The *code point* is the number
- The *character* is what the number represents



65 (bin: 01000001; hex: 41) ↦ A

66 (bin: 01000010; hex: 42) ↦ B

...

90 (bin: 01011010; hex: 5A) ↦ Z

Encoding : basics

- The *code point* is the number
- The *character* is what the number represents
- It can also be a *“control character”*:

7 (bin: 00000111; hex: 07) ↦ <*bell*>

...

65 (bin: 01000001; hex: 41) ↦ A

66 (bin: 01000010; hex: 42) ↦ B

Membre fondateur de :

Encoding standards

A historical sketch

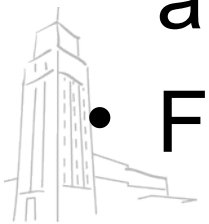
Membre fondateur de :

Encoding the latin alphabet

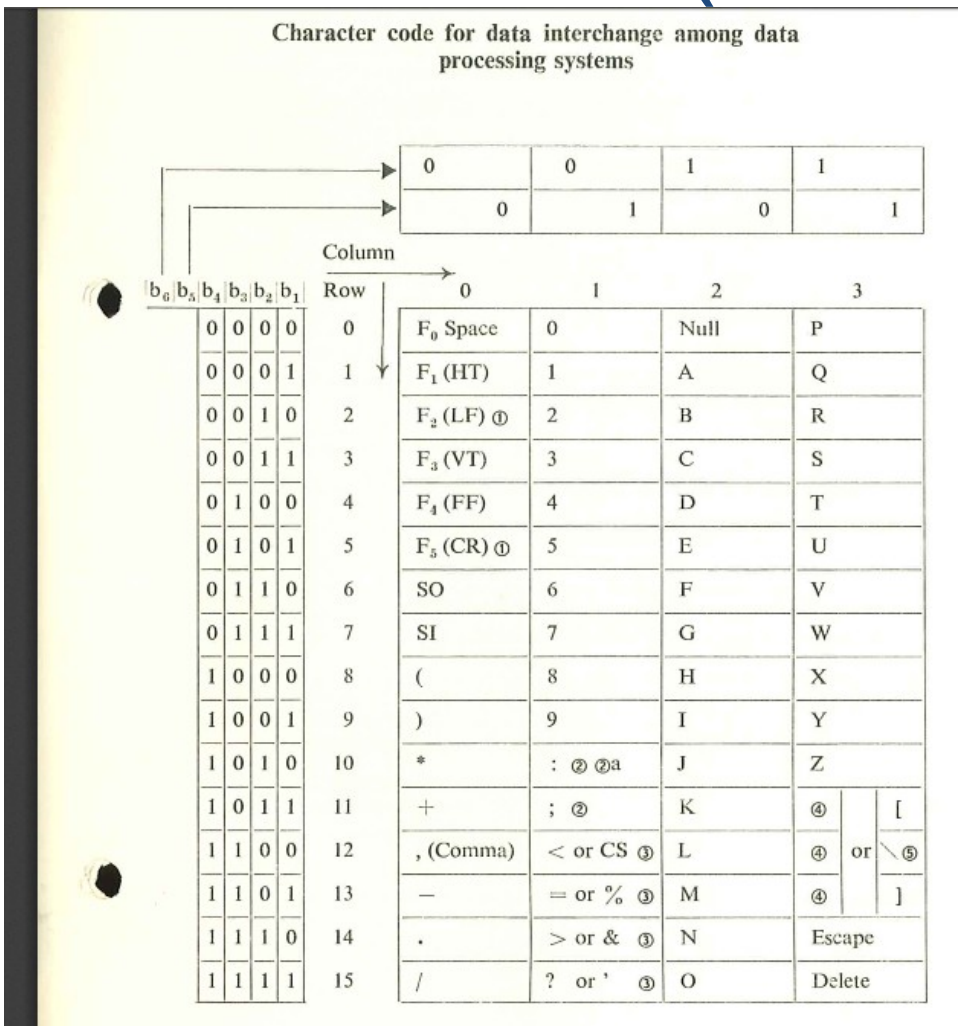
- Antecessor : the Baudot code (for TELEX)
- Latin alphabet proper: 26 characters. Additionally needed: figures (0,1,2 … 9), some essential typographic symbols (!&#'()"/:;?,.), some transmission codes (WRU, Space, Bell, CR, LF)
- 5 bits are not enough (32 code points) … but in 1930, size of data transmitted is an issue !
- Solution : **shifts** (to figures / to letters)
- Code 01010 means R when transmitting letters, or 4 when transmitting figures.

Encoding the latin alphabet

- First computers : in the USA, 1950s
- The English language only uses plain latin letters
- Size of data still an issue, but it is important to get rid of the *shifts* (drawbacks: contextuality of the meaning, big data loss if a transmission error occurs on shift code)
- First computer encoding standards : 6 bits



Ex : ECMA 1 (1963)



Character code for data interchange among data processing systems

						0	0	1	1
						0	1	0	1
b_6	b_5	b_4	b_3	b_2	b_1	Row ↓ Column →	0	1	2	3
0	0	0	0	0	0	0	F_0 Space	0	Null	P
0	0	0	0	1	1	1	F_1 (HT)	1	A	Q
0	0	0	1	0	2	2	F_2 (LF) ①	2	B	R
0	0	0	1	1	3	3	F_3 (VT)	3	C	S
0	0	1	0	0	4	4	F_4 (FF)	4	D	T
0	0	1	0	1	5	5	F_5 (CR) ①	5	E	U
0	0	1	1	0	6	6	SO	6	F	V
0	0	1	1	1	7	7	SI	7	G	W
0	1	0	0	0	8	8	(	8	H	X
0	1	0	0	1	9	9	)	9	I	Y
0	1	0	1	0	10	10	*	: ② ②a	J	Z
0	1	0	1	1	11	11	+	; ②	K	④ or [
0	1	1	0	0	12	12	, (Comma)	< or CS ③	L	④ or \ ⑤
0	1	1	0	1	13	13	–	= or % ③	M	④ or]
0	1	1	1	0	14	14	.	> or & ③	N	Escape
0	1	1	1	1	15	15	/	? or ' ③	O	Delete

Encoding the latin alphabet

- Six bits is just good enough for encoding programming instructions, not for actual texts (no case distinction, and a lot of typography is lacking)
- ANSI X32 standard ASCII (1963 – 1967)
- Encoding on 7 bits
- Allows case distinction, more typographic symbols, and a lot more control codes

Membre fondateur de :
SORBONNE PARIS CITÉ
CAMPUS CONDORCET Paris-Aubervilliers

Encoding the latin alphabet

- US-ASCII: 7 bits. Why 7?
- At the time, 8 bits processors were becoming the standard

  but: transmission length was still an issue

  and: transmission reliability was an issue
- Using 7 bits from 8 allows the use of a *parity check bit*
- Single-bit transmission errors are detected

Parity check

- Count the number of 1 in the 7 bits code
- If the sum is even, set the 8^{th} bit to 0
- If the sum is odd, set the 8^{th} bit to 1

  A = 65 = 1000001 encoded as **0**1000001

  C = 67 = 1000011 encoded as **1**1000011
- To remain undetected, a transmission error has to hit 2 bits in the sequence (probability squared)

ASCII: mother of all encodings

- ASCII, being an industry standard in the USA, is adopted by manufacturers and installed on all computers (except IBM mainframes, which use EBCDIC)
- Even other countries start using it: ISO-646
- It becomes the matrix for further encoding standards (KOI-8, JIS, ISO-8859-x)
- Until today, the 128 ASCII codepoints are the first 128 codepoints of Unicode

Encoding alphabets

- In the 1960s, composing and transmitting texts was still a marginal use for computers (no screens, bad printers)
- In the 1980s, as computers spread out as multipurpose devices, the non-English users felt the need to have their own writing signs available

Encoding alphabets

- In countries with only a few supplementary latin symbols, there were some national 7 bits variants of ASCII (ex. Germany DIN 66003, France AFNOR NF Z 62010).

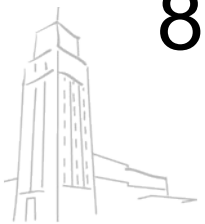
ASCII	@	[	\	]	{	|	}	~
DIN 66003	§	Ä	Ö	Ü	ä	ö	ü	ß
NF Z62010	à	°	ç	§	é	ù	è	¨

- A big impediment since the ASCII characters were used in other contexts

Encoding alphabets

- In the 1980s, the use of the parity bit is less relevant
- Most manufacturers get to the conclusion that they should use that bit, which opens 128 brand new code points for encoding
- The solution to the encoding problem:

  8-bit extensions of ASCII



Encoding cyrillic

- Soviet standard for a 8-bit extension of ASCII as soon as 1974 : the GOST 19678-74 standard, widely known as KOI-8 *(Код для Обмена Информации)*
- Designed so that cyrillic letters are assigned codepoints on the second plane that correspond to “similar” latin letters
- If the 1st bit is stripped, text still partly legible
- In use until it became replaced by Unicode

Encoding cyrillic : KOI-8 RU

Съешь же ещё этих мягких французских булок, да выпей чаю.

s_E[X VE E]# \TIH MQGKIH FRANCUZSKIH BULOK, DA WYPEJ ^A@.

Encoding latin diacritics

- Many languages make use of latin letters with distinctive marks to fit their phonemic system (ä,ö,ü,é,è,ç,ã,ñ,å,ø,č,š,ę,ă,ğ …)
- Those marks are called *diacritics*
- Before the standardization of 8-bit encodings, text processing programs used combinations of ASCII letters and non-spacing diacritics
- Ex : TeX (1982) : `\'{e}, \`{a}, \^{\i}, \c{c}`

Membre fondateur de :
SORBONNE PARIS CITÉ
CAMPUS CONDORCET Paris-Aubervilliers

Encoding latin diacritics

- But : a latin letter with a diacritic generally represents a phoneme: it logically is considered a character in its own right
- Various national 8-bit extensions of ASCII are defined in the 1980s

Encoding latin diacritics

- In the late 1970s and 1980s, operating system manufacturers set their own proprietary standards for ASCII extensions
- There are many different systems in competition with one another
- Example for Western European languages: *OEM CP 437, OEM CP 850, Mac Roman, HP Roman 8, Windows CP 1250, DEC MCS, LaTeX T1 encoding (Cork) ...*

Membre fondateur de :
SORBONNE PARIS CITÉ
CAMPUS CONDORCET Paris-Aubervilliers

Encoding latin diacritics

- As the need for exchanging document increases (with internet), the jungle of proprietary format becomes unbearable
- A new set of standards arises in 1986 : the ISO-8859
- For western european languages, the standard is ISO-8859-1 (Latin-1)
- It is mostly based on DEC MCS, and Windows CP-1252 is also quite similar

Membre fondateur de :
SORBONNE PARIS CITÉ
CAMPUS CONDORCET Paris-Aubervilliers

Encoding alphabets

- ISO-8859 principles :
  - 15 different substandards (ISO-8859-*x*)
  - 1 character = 1 byte = 8 bits
  - the first half (1st bit = 0) is always ASCII
  - the second half (1st bit = 1) extends to the needs of other languages
  - like in the first half, the first 32 code points of the second half (8/0 to 9/15) are assigned to control codes, not to characters

Membre fondateur de :

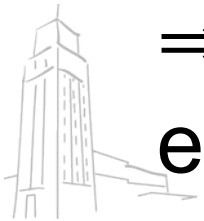
Encoding alphabets

- ISO-8859 principles :
- Doubling the ASCII code space: 128 new code points

  ⇒ enough to fit plenty of new characters for German and French *or* for Czech and Polish *or* for Turkish *or* for Greek ...

  ... but not for all of them at the same time !

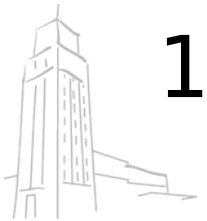
  ⇒ ISO-8859-*x* standards are mutually exclusive



Encoding alphabets

- ISO-8859-*x* standards are mutually exclusive; examples:

	ISO-8859-1	ISO-8859-2
11100000 :	à	ŕ
11101000 :	è	č
11101111 :	ï	ď
11111000 :	ø	ř



Membre fondateur de :
CAMPUS CONDORCET Paris-Aubervilliers

Encoding alphabets

- What happens when you look at a text in ISO-8859-2 through an ISO-8859-1 agent?

  Situace na východě Ukrajiny bude podle Kyjeva vyřešena do 48 hodin, a to diplomaticky, nebo silou. Separatisté na východě kontrolují část úřadů. Ruské ministerstvo zahraničí se mezitím dušuje, že ruská vojska u ukrajinských hranic nepředstavují hrozbu, Moskva prý invazi nechystá.

Encoding alphabets

- What happens when you look at a text in ISO-8859-2 through an ISO-8859-1 agent?

  Situace na východì Ukrajiny bude podle Kyjeva vyøe¹ena do 48 hodin, a to diplomaticky, nebo silou. Separatisté na východì kontrolují èást úøadù. Ruské ministerstvo zahranièí se mezitím du¹uje, Že ruská vojska u ukrajinských hranic nepøedstavují hrozbu, Moskva prý invazi nechystá.

Encoding alphabets

- ISO-8859-1 : Western European Latin
- ISO-8859-2 : Eastern European Latin
- ISO-8859-5 : Cyrillic (never beat KOI-8)
- ISO-8859-6 : Arabic (not usable alone)
- ISO-8859-7 : Modern Greek
- ISO-8859-8 : Hebrew (not usable alone)
- ISO-8859-9 : like ISO-8859-1 + Turkish
- ISO-8859-15 : like ISO-8859-1 + €

Membre fondateur de :
GORBONNE PARIS CITÉ
CAMPUS CONDORCET Paris-Aubervilliers

Encoding Japanese

- Like in the alphabetic world, the first thing to try is a 8-bit extension of ASCII
- JIS X 0201:
  - First half = ASCII
  - Second half = basic katakana (leaves 64 blank code points)

⇒ impossible to fully write Japanese

but: possible to spell Japanese (approximately)

第１面(JISラテン文字)

	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	¥	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	‾
F	SI	US	/	?	O	_	o	DEL

第２面(半角カナ文字)

	0	1	2	3	4	5	6	7
0				ー	タ	ミ		
1			。	ア	チ	ム		
2			「	イ	ツ	メ		
3			」	ウ	テ	モ		
4			、	エ	ト	ヤ		
5			・	オ	ナ	ユ		
6			ヲ	カ	ニ	ヨ		
7			ァ	キ	ヌ	ラ		
8			ィ	ク	ネ	リ		
9			ゥ	ケ	ノ	ル		
A			ェ	コ	ハ	レ		
B			ォ	サ	ヒ	ロ		
C			ャ	シ	フ	ワ		
D			ュ	ス	ヘ	ン		
E			ョ	セ	ホ	゛		
F			ッ	ソ	マ	゜		

IUT de Bobigny
1, rue de Chablis — 93017 Bobigny cedex
www.iut-bobigny.univ-paris13.fr

Encoding Japanese

- To write Japanese (with *kanji*), tousands of characters are needed
- They do not fit into 8 bits, but fit into 16 bits
- Shift-JIS : implements JIS X 0208 character set
- Total 6879 characters (incl. 6355 kanji)
- compatible with JIS X 0201; the free zones in JIS X 0201 are used as first bytes for characters encoded in two bytes
- The second byte can only be interpreted in the context of the first byte

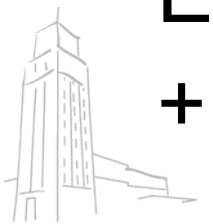
Membre fondateur de :
SORBONNE PARIS CITÉ
CAMPUS CONDORCET Paris-Aubervilliers

Encoding Japanese

- Extended repertoire: JIS X 0212
- brings 6067 new characters (5801 kanji)
- Two ways to combine it with ASCII :
  - ISO-2022 : fixed size encoding (2 bytes), compatible with JIS X 0201, uses *shifts* triggered by escape sequences
  - EUC-JP : compatible with ASCII but not with JIS X 201. Variable width encoding (1 to 3 bytes)

Encoding Chinese

- To write Chinese: in 1984, Taiwan defines a two-byte standard compatible with ASCII
- Named Big5 (5 big industrial companies)
- First byte beginning with a 1 (so that the system understands it is leaving ASCII)
- Arbitrary second byte
- Encodes more than 11000 standard *hànzì* + possible proprietary extensions



Encoding Chinese

- The Big5 standard, originated in Taiwan, has been used up from the 1980s up to now (ebbing away in favour of Unicode)
- in Taiwan, Hong Kong, Macao
- and everywhere where overseas Chinese communities use the traditional character shapes

Encoding Chinese

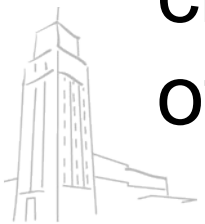
- In People's Republic of China, the writing system went through two simplification reforms in 1956 and 1964 (aimed at increasing the literacy rate)

書 車 鐘 幾 麼 讓 龜

书 车 钟 几 么 让 龟

Encoding Chinese

- PRC standard since 1980 : GB 2312
- 6763 simplified characters
- Fixed, two-bytes encoding
- Compatible with ASCII within the EUC-CN encoding schema
- A byte beginning with a 0 is an ASCII character, a byte beginning with a 1 is one of the two bytes of a GB 2312 character



Membre fondateur de :

Encoding standards in 1990

- Many different encoding systems in use
- For alphabetic scripts: a more or less dominant standard (ISO-8859-*x*) but many proprietary standards still around
- Not able to represent Hebrew and Arab right-to-left directionality, or Arab character contextuality
- Many fixed-width (1 byte) character encoding systems, mutually incompatible

Encoding standards in 1990

- Many different encoding systems in use
- For asiatic scripts: a jungle of national character sets (GB 2312, Big 5, JIS X 0208, JIS X 0212, KS X 1001)
- Different physical encoding schemes in competition for every character set: ISO 2022, EUC, Shift-JIS ..
- Fixed or variable width, *shifts* using escape sequences, stateful encoding …

Membre fondateur de :
SORBONNE PARIS CITÉ
CAMPUS CONDORCET Paris-Aubervilliers

Encoding standards in 1990

- Many different encoding systems in use

  ⇒ Nightmare!
- Only ASCII as common ground

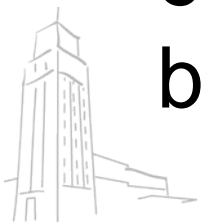
Encoding standards

The Unicode standards

Membre fondateur de :

Unicode

- Unicode is a **universal encoding system**
- It defines a **universal character set** for all languages in the world (past or present)
- It encompasses all former available character sets
- It comes along with guidelines to ensure correct coding of directionality, word breaking, line breaking, and collation



Unicode character set

- The character set proper is built in cooperation with the ISO 10646 standard
- Natively, code points have 32 bits
- Code space is divided in 65536 “planes” of 65536 characters (16 bits)
- In practice, only the first three “planes” have been alloted, and only the first one *(Basic Multi-lingual Plane* or *Plane 0)* is in common use

Membre fondateur de :

Unicode character set

- Plane 0 *(Basic Multilingual Plane)* allotted for writing systems currently in use
- Plane 1 *(Supplementary Multilingual Plane)* allotted for ancient writings (e.g. Egyptian hieroglyphs, Sumerian cuneiforms ...)
- Plane 2 *(Supplementary Ideographic Plane)* allotted for ancient or rare Asian characters
- Planes 15 and 16 reserved for private use

Unicode character set

- Natively, code points have 32 bits
- In practice, only the first three “planes” of 65536 characters (16 bits) have been alloted, and only the first one *(Basic Multi-lingual Plane* or *Plane 0)* is in common use
- It is envisioned that Planes 3 to 13 could be made available to code point allotment in the future
- Last code point: 00 10 FF FF (21 bits)

Unicode character set

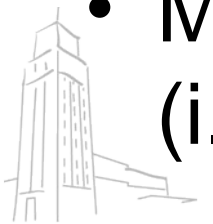
⇒ 5×65536 = 327 680 currently reserved

- on more than 2 billion (2^{31}-1) code points
- 11×65536 = 720 896 available on planes 3-13
- knowing that all writing systems currently in common use by living people fit within the BMP

⇒ No risk that Unicode will run out of coding space, *ever*

Basic Multilingual Plane

- Code points on 2 bytes
- Native encoding : UCS-2 (*Universal Character Set* on 2 bytes encoding)
- Derived encoding : UTF-16, to allow shifts to other planes (if it is necessary to be able to use characters outside the BMP)
- Most of the time, UTF-16 = UCS-2 code (i.e. a plain unsigned integer number)



Membre fondateur de :
SORBONNE PARIS CITÉ
CAMPUS CONDORCET Paris-Aubervilliers

Unicode BMP and former standards

- The first 256 code points are ISO-8859-1

  ⇒ The first 128 code points are ASCII
- The ISO-8859-*x* series follow in sequence
- Characters with common characteristics are grouped together (for example, right-to-left scripts appear contiguously)
- Alignment on 128-code-point boundaries

Unicode BMP and former standards, Asian scripts

- A huge unification endeavour to bring together all 漢字 used by Traditional Chinese, Simplified Chinese, Japanese and Literate Corean *(CJK Unified Ideographs)*
- Code points from 4E00 to 9FCC (uses up 1/3 of the BMP space)
- Traditional order (Radical + Stroke count)

Unicode BMP and former standards, Asian scripts

- Order of 漢字 in classical dictionaries: first sorted by Kangxi radical (214 of them)

  … then by the number of strokes needed to complete the character

  Example : 法 (*fǎ* : the law, France)

Radical 85 (water) + 5 more strokes

氵 去

Unicode BMP and former standards, Asian scripts

Correspondances between the *CJK Unified Ideographs* of Unicode, all former standards (JIS, GB, Big5), classical dictionaries, various latin translitterations of Mandarin (pinyin, Wade-Giles) or other Chinese dialects (Cantonese) or pronounciation in other languages (Japanese “on” reading) are stored in the *UNIHAN Database*

Membre fondateur de :

Unicode character set

- There is every character you may need in the Unicode character set
- All you need is the correct font to display it
- Unicode TTF now available from all foundries
- General “fallback” fonts (sometimes ugly, but work anywhere) : SimSun, Arial Unicode MS, Lucida Sans Unicode

Grapheme clusters

- As seen before, an encoding schema should encode characters, not glyphs
- However, some clusters of characters are represented by a single visual form
  - Latin st, fi: ﬆ, ﬁ (fancy, visual enhancements)
  - Arabic lam-alif: ﻻ (compulsory), lam-ǧim: ﳉ
  - Greek *alpha* with rough breathing, tone and subscript iota: ᾇ
  - Devanagari ka: क्‍ु

Grapheme clusters

- Those visual shapes are not clearly separable in the linear (syntagmatic) axis, but they are not writing characters in their own rights, they are graphical ligatures
- You expect to find the word “find” by looking for a substring [f,i,n,d] in a search engine, even when it’s written “find”
- Unicode considers them not characters but “grapheme clusters”

Grapheme clusters

- Unlike a genuine character like “ş”, a grapheme cluster like “ſt” should be internally represented by two different code points, not one single code point
- It is the job of the visual rendering agent to have the ligature displayed
- However, Unicode includes some single code points for such ligatures as a legacy from older standards

Unicode transformation formats

- Native Unicode code points are 31 bits integers (UCS-4)
- But all of the characters actually used for living languages fit in 2 bytes (BMP)
- Handling four bytes words is fine for modern processors (**wchar_t** in C and C++)
- But when storing them on a hard disk or sending them over a cable, it is a waste

Unicode transformation formats

- If you mainly have data in English, and if you store each of your characters as 4 bytes, you fill up ¾ of your hard disk with zeroes
- If you mainly have data in Chinese, you still fill up ½ of your hard disk with zeroes
- That's what the UTF were made for (Unicode Transformation Format, or *encoding scheme*)

Unicode UTF-16

- Unicode originally planned to be a fixed-width, two-bytes encoding
- When more planes were added, it was necessary to find a way to encode characters outside the BMP
- Solution : use two-byte sequences with no assigned meaning in the BMP, and use their spare bits
- This is called the *surrogates* mechanism

Membre fondateur de :
CAMPUS CONDORCET Paris-Aubervilliers

Unicode UTF-16

- The code point range D800-DFFF is forbidden (contains no characters)
- A character in the BMP (0000-D7FF or E000-FFFF) is represented untransformed
- A character beyond the BMP may be represented by a sequence of 20 bits
  (NB. no code points after 10FFFF)
- It is represented by a sequence of four bytes called a *surrogate pair*

Membre fondateur de :
SORBONNE PARIS CITÉ
CAMPUS CONDORCET Paris-Aubervilliers

Unicode UTF-16

- A character beyond the BMP is in the range 10000-10FFFF (in hexadecimal)
- that is, in binary :

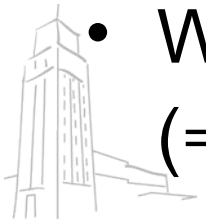
  1 xxxx xxxx xxxx xxxx - 10000 xxxx xxxx xxxx xxxx
- the bits above the lower-order 16 bits go from 1 to 10000
- subtract 1, you get something between 0 and 1111 instead (⇒ 4 bits only)

Unicode UTF-16

- So : uuuuu xxxxxx yyyyyyyyyy (5+6+10 bits) becomes vvvv xxxxxx yyyyyyyyyy (4+6+10 bits), where vvvv = uuuuu - 1
- High (lead) surrogate: D800 + vvvvxxxxxx
- Low (trail) surrogate: DC00 + yyyyyyyyyy
  The Unicode character in UTF-16 becomes
  110110*vv vvxxxxxx* 110111*yy yyyyyyyy*
- A two-byte sequence beginning with 110110 necessarily is a HS, with 110111 a LS

Unicode UTF-16

- Problem with UTF-16: byte order in memory
- Motorola processors are *big endian* (high byte first), Intel are *little endian* (low byte first); dumping raw memory data to disk → ?
- The solution : *Byte-Order Mark*.
- Put the code point “FEFF” (deprecated meaning : *zero-width nbr space*) in front of every document
- When reading, see if it appears as FEFF or FFFE (⇒ *big endian* and *little endian*, respectively)



Membre fondateur de :
SORBONNE PARIS CITÉ
CAMPUS CONDORCET Paris-Aubervilliers

Unicode UTF-16

- Unicode UTF-16 is the closest to the original idea of Unicode
- It still allows for the possibility to go beyond the BMP
- It is the native encoding system of Windows XP (theoretically) and further versions, and of the NTFS file system
- It is the internal representation of characters in the Java language

Membre fondateur de :
SORBONNE PARIS CITÉ
CAMPUS CONDORCET Paris-Aubervilliers

Unicode UTF-8

- With UTF-16: if the data you have to work with mainly uses alphabetic scripts, you still fill up ½ of your hard disk space with zeroes
- But you do not want to go back to ISO-8859 because you want support for different latin variants, cyrillic, greek, and the occasional Chinese character …
- The solution is called UTF-8

Unicode UTF-8

- The idea behind UTF-8 : variable width character encoding
- ASCII characters (0xxxxxxx) should be encoded as single bytes, transparently, just as plain 7-bit ASCII
- If you are a British or Northern American user, you efficiently use your hard disk just as if it were plain ASCII, without losing the benefit of multilinguality

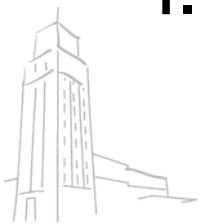
Membre fondateur de :

Unicode UTF-8

- ASCII characters (0xxxxxxx) should be encoded as single bytes, transparently, just as plain 7-bit ASCII
- When you go above 007F, use two bytes : the first one starting with 110, the second one with 10.
- That leaves 11 bits for meaningful encoding
  110xxxxx 10xxxxxx

Unicode UTF-8

- When 11 bits are not enough, go up to three bytes: the first one starting with 1110, the second and third ones with 10.
- That leaves 16 bits for meaningful encoding:
  1110xxxx 10xxxxxx 10xxxxxx
- You can get to code point 2^{16}-1 (65535)
  i.e. cover the BMP



Unicode UTF-8

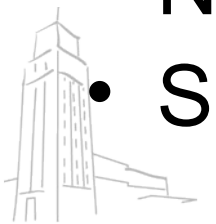
- If you need to go beyond the BMP, get up to four bytes:

  11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

  That leaves 21 bits for encoding
- You don’t need more!
- (last codepoint : 0FFFFF)

Unicode UTF-8

- With UTF-8:
- You still have the whole Unicode
- Latin texts encoded with very little overhead compared to fixed 8-bit encodings
- A character may be represented by an ordered sequence of 1 to 4 bytes
- No byte order problem (bytes are ordered)
- Sort order is preserved



Unicode UTF-8

- Unlike *shift*-based encoding systems, you always know how to interpret one byte in the middle of a byte flow :
  - If it starts with 0, it is a plain ASCII character
  - If it starts with 110, it is the first byte of a sequence of two bytes encoding a character
  - If it starts with 1110 or 11110: same principle
  - If it starts with 10, it is a trailing byte of a sequence encoding a character ; go back to at most 3 characters above to find the beginning

Unicode UTF-8 and UTF-16

- How are the following characters encoded in ISO-8859-1, UTF-16 and UTF-8 ?

  è: (like in French *père*) 00E8

  ü: (like in German *müde*) 00FC

  č: (like in Czech *hořčica*) 010D

  ж: (like in Russian *жадный*) 0436

  水 : (like in Chinese 風水) 6C34

   : (laughing emoticon) 1F604

Unicode UTF-8 and UTF-16

- How are the following characters encoded in ISO-8859-1, UTF-16 and UTF-8 ?

è:	00E8	E8	00 E8	C3 A8
ü:	00FC	FC	00 FC	C3 BC
č:	010D	*	01 0D	C4 8D
ж:	0436	*	04 36	D0 B6
水:	6C34	*	6C 34	E6 B0 B4
😄:	1F604	*	D8 3D DE 04	F0 9F 98 84
		ISO-8859-1	**UTF-16**	**UTF-8**

Choice of encoding

- When should I choose ISO-8859-1 ?

  → when you need a very compact representation of characters, and you are absolutely sure that you never ever will need anything else than English, German or French (best keep it internal)
- When should I choose UTF-8 ?

  → when you want Unicode, and you're using mostly the latin alphabet
- When should I choose UTF-16 ?

  → when you are Chinese or Japanese (save 33%)

Unicode standard annexes

- Unicode also offers :
  - A bidirectional algorithm
  - A line-breaking algorithm
  - A collation algorithm
  - Normalization forms
  - Text segmentation
  - Definition of character sequences
- They are meant for use with plain text files : in **documents**, **markup** provides alternate ways of achieving the same functions

Unicode annexes (references)

- Unicode bidirectional algorithm

  `http://www.unicode.org/reports/tr9/`
- Unicode normalization forms

  `http://www.unicode.org/reports/tr15/`
- Unicode collation algorithm

  `http://www.unicode.org/reports/tr10/`

Unicode annexes (references)

- Unicode text segmentation scheme

  `http://www.unicode.org/reports/tr29/`
- Unicode line-breaking algorithm

  `http://www.unicode.org/reports/tr14/`
- Unicode named character sequences

  `http://www.unicode.org/reports/tr34/`

Transfer encoding

- There is a further possible layer of encoding
- Independent of the content encoding, a computer process may chose to re-encode any sequence of bytes in “ASCII-clean” form
- This has to be done whenever there is a risk that higher order characters may be “re-interpreted” on the way
- We know that ASCII is the only thing that gets through anywhere

Membre fondateur de :

Common transfer encodings

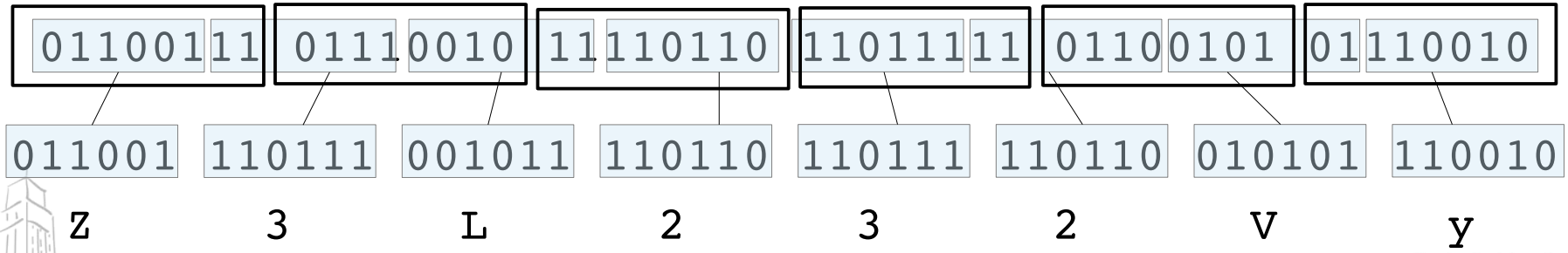
- **8bit** : no transformation ; the 8 bits are transferred as they are.
- Ex. if input is in ISO-8859-1 :

  “größer” (67 72 F6 DF 65 72)

  → “größer” (67 72 F6 DF 65 72)
- Quite simple (no transfer encoding layer)
- OK if you trust the transmission chain to be 8bit clean (no bit stripping, no re-encoding)

Common transfer encodings

- **base64** : 8bits → 6 bits (25% overhead)
- 6-bits sequences are coded by 64 plain ASCII characters (26 capital letters, 26 lowercase letters, 10 figures, “+”, and “/”)
- Ex : “größer” (67 72 F6 DF 65 72):



- “größer” → “Z3L232Vy”

Common transfer encodings

- **quoted-printable** : high bytes (beginning with 1) and controls → 3 bytes (“=” + hex)
- The character “=” itself → “=3D”
- More readable if the byte flow consists mostly of latin text (ASCII chars left unmodified)
- Low overhead if mostly latin text
- Ex : “größer” (67 72 F6 DF 65 72):
- “größer” → “gr=F6=DFer”

Common transfer encodings

- **“percent encoding”** : similar to quoted-printable, but the escape character is “%”
- Ex : “größer” (67 72 F6 DF 65 72):
- “größer” → “gr%F6%DFer”
- Mainly used when strings sent to servers by GET have to pass through URLs

Transfer encoding

- Transfer encoding is agnostic as to which character set encoding is used by the interpreting program
- It just transcodes bytes (not characters)
  (ex : `https://www.google.de/?q=gr%C3%B6%C3%9Fer`)
- (C3 B6 is the Unicode UTF-8 encoding of “ö”
  C3 9F is the Unicode UTF-8 encoding of “ß”)

Levels of encoding

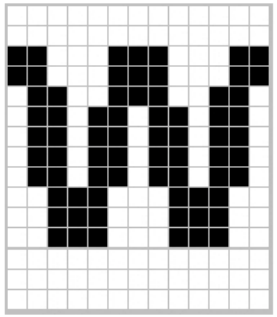
- **Encoding** means three different things at three different levels :
- *Character encoding* (in the character set) :
  ö = code point #246 (hex F6)
- *Encoding scheme* (UTF-16 or UTF-8) :
  ö (11110110) → C3 B6 (**110**00011 **10**110110)
- *Content transfer encoding* :
  C3 B6 → =C3=B6 (in “quoted-printable”)

From texts do documents

« Rich » or « Enriched » text

- The human reader is familiar with seeing visual formatting along with text
- A text is not a mere sequence of characters : it also involves typesetting
  - Paragraph alignment (left, center, justified …)
  - Font family (Times, Helvetica, Courier ...)
  - Font size (normal, small, big)
  - Font style (roman, italic)
  - Font weight (normal, bold)

« Rich » or « Enriched » text

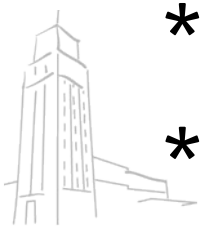
- Until the 1980s there was no middle stand between sending an image (“raster”) file and sending a sequence of characters
  - Image file: visual structure, but very stupidly encoded; huge waste of bytes:  (256 bits)
  - Text as a sequence of character codes: no visual structure: “w” (8 bits)

Membre fondateur de :
SORBONNE PARIS CITÉ
CAMPUS CONDORCET Paris-Aubervilliers

« Rich » or « Enriched » text

- Users of specific word processing programs could exchange proprietary files
- E-mail and usenet users had spontaneously developed a form of ASCII compatible “poor man’s” markup :

```
_underlined_
*italic*
**bold**
```



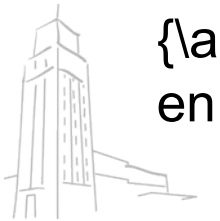
« Rich » or « Enriched » text

- 1984 : Adobe issues the Postscript format, designed for printers
- allows to represent text and visual structure (programmed instructions)
- files may be sent from computer to computer
- still very heavy, if not compressed

« Rich » or « Enriched » text

- 1987 : Microsoft issues the RTF format (Rich Text Format), a readable export format for text processing documents
- Typesetting instructions are embedded in curly brackets

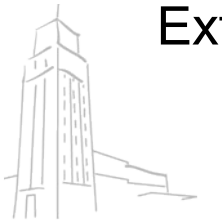
{\af0 \ltrch s} {\af0 \ltrch \rquote } {\af0 \ltrch appuyer sur les m\'eames indices. Cela \'e9vite en outre l} {\af0 \ltrch \rquote } {\af0 \ltrch \'e9cueil de l}{\af0 \ltrch \rquote } {\af0 \ltrch \'e9valuation } {\ai\af0 \ltrch\i a priori} {\af0 \ltrch , qui consiste \'e0 trier les lecteurs en deux groupes, }



« Rich » or « Enriched » text

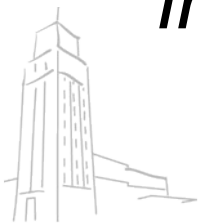
- 1993 : RFC 1523 defines a MIME standard for « enriched text » — inspired by the contemporary development of HTML (mimetype : text/enriched ; typical file extension : ETF)
- Markup is enclosed in angle brackets :

  <bold>MIME</bold> : acronym for <italic>Multipurpose Internet Mail Extensions</italic>.



« Rich » or « Enriched » text

- But all of this still only brings visual typesetting information
- It does not say much about the logical structure of a text
- It remains superficial with respect to the growing need to handle *structured information*



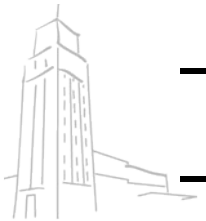
Annotating document structure

- A *text* is a sequence of characters
- A *document* is an ordered structure

  (e.g. a book is a sequence of chapters, every chapter being a sequence of paragraphs)
- For a document, encoding characters is not enough: you have to encode the structure

Markup languages

- A first step towards storing the structure of documents is SGML (*Standard Generalized Markup Language*, adopted ISO standard 8879:1986)
- The main idea : define structure, content, and layout separately
  - *Structure* is defined in a DTD
  - *Content* is given in an SGML file
  - *Layout* is defined elsewhere (user-agent)



Markup languages

- Offspring of SGML : HTML (developed at CERN by Tim Berners-Lee and Robert Cailliau, 1990)
- HTML = a specific SGML DTD
  - adapted to technical documentation
  - defining some structure (<h1>, <h2>, etc.), and some visual formatting (<b>, <i>, etc.)
  - adding hyperlinks (<a href="...">)
  - designed along with the HTTP protocol

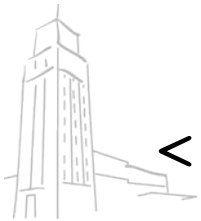
XML

- 1996 : W3C (World Wide Web consortium) working group : Bray, Paoli and Sperberg-McQueen
- All-purpose generalization of HTML
- Keeps the new ideas in HTML (hypertext, network portability)
- goes back to the versatility of SGML (with the concept of DTD)

XML Markup System

- XML (eXtensible Markup Language) is a meta-language, offering a general frame for encoding any (tree-like) document structure

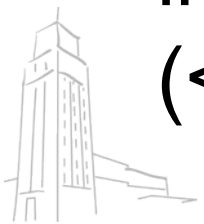
```
<book>
  <title>Der Golem</title>
  <chapter>
    <p>...</p>
    <p>...</p>
  </chapter>
</book>
```



Membre fondateur de :
CAMPUS CONDORCET Paris-Aubervilliers

XML Markup System

- In XML, markup is mixed with the content of the text, in the same file
- Marks are conventionally enclosed in <>
- What is inside an opening-closing pair is an *element* (...)
- An element can be assigned additional information by the way of *attributes* (...)



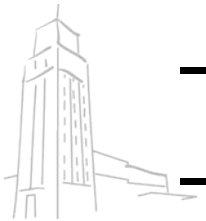
Membre fondateur de :

XML Markup System

- General rules for well-formedness:
  - No crossovers (<a>.........)
  - No unclosed elements (<p>...)
  - one single root element per document
- Linked to the tree-like structure of any document

Validity w.r.t. document type

- XML itself does not specify the structure of all possible documents
- XML is only a general frame for storing or exchanging tagged and structured data (a meta-language)
- The structure of a specific family of documents is defined by:
  - a *DTD* (Document Type Definition)
  - or an *XML Schema*

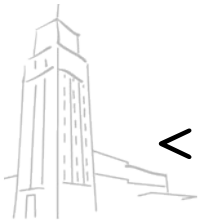


Membre fondateur de :

Validity w.r.t. document type

- Example families of documents:
- HTML (specified in XML by XHTML1.0)

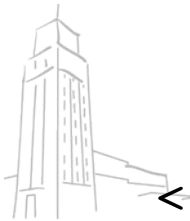
```
<html>
  <head><title>My web page</title></head>
  <body>
    <h1>Welcome to my web page</h1>
    <p>Hello, world!</p>
    <p><a href="b.html">Click me!</a></p>
  </body>
</html>
```



Validity w.r.t. document type

- Example families of documents:
- XDOC

```
<document>
  <properties>
    <title>My document</title>
    <author>Pascal Vaillant</author>
  </properties>
  <body>
    <section name="Important matter">
      <p>Hello, world!</p>
    </section>
  </body>
</document>
```



Validity w.r.t. document type

- A document is *well-formed* when it does not violate the basic rules which define what an XML document is (i.e. basically, a tree)
- It is *valid* (with respect to a given document type) when its structure conforms to the structure defined in a DTD or XML Schema
- (example: a HTML document contains a <html> element, which contains a <head> and a <body>, etc.)

Membre fondateur de :

XML and script encoding

- XML has been released in 1998, after Unicode (1991), so XML is Unicode-aware
- Since XML has been designed for information interchange, it prefers UTF-8 (no need to transmit two bytes when you need one)
- So is XHTML1.0 (HTML 4 rewritten in XML), and further versions of HTML

Representing text documents

Text Encoding Initiative

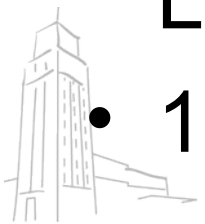
- In the 1980s, emerging need to store and process humanities data as digital text

  (use of the computer tool by scholars in literary studies, history, linguistics)
- Need to represent textual structure and textual units
- But : no common standard

  ⇒ A jungle of different specific uses

Text Encoding Initiative

- 1987 : meeting at Vassar College
- Foundation of the *Text Encoding Initiative*
- Sponsored by the Association for Computers and the Humanities (ACH), the Association for Computational Linguistics (ACL), and the Association for Literary and Linguistic Computing (ALLC)
- 1988 : v1 of the TEI Guidelines



The TEI Guidelines

- Ensure compatibility between character, text, and document encoding standards
- Define a common standard for referring to electronic texts and storing metadata
- Define some common basic text structures (chapters and sections, notes, references)
- Define standards for some more specific uses (literary texts, historical documents, dictionaries, transcriptions of speech ...)

Membre fondateur de :

The TEI Guidelines

- Originally SGML + existing ISO charsets
- Now XML + Unicode (since P4, 2002)
- Current version TEI P5 :

  http://www.tei-c.org/Guidelines/P5/

Marking the encoding

XML and script encoding

- Every XML document should begin with a XML declaration specifying encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
```

(UTF-8 being the default if not specified)

How do I know the encoding?

- To interpret a sequence of bytes and to display it, I need to know the encoding
- How do I know the encoding of a document?
  - either it is written in the document itself
  - or it is sent by a computer process to another as side information
  - (or both)

How do I know the encoding?

- A general framework to specify the layers of encoding (character set + transfer encoding) : MIME
- MIME : Multipurpose Internet Mail Extensions
- RFC1341 (IETF), 1992
- RFC1521 et RFC1522 (IETF), 1993
- Defines attributes : MIME-Version, Content-Type, Content-Transfer-Encoding, Content-ID, Content-Description
- Originally designed for use as e-mail headers

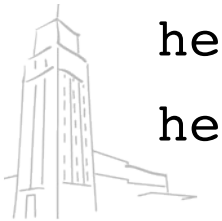
How do I know the encoding?

- The HTTP protocol provides a way for the server to send the encoding information to the client in the HTTP header:

```
Content-Type: text/plain; charset=UTF-8
Content-Transfer-Encoding: 8bit
```

- A PHP-based web site may choose to send this information in the header :

```
header("Content-Type: text/plain; charset=UTF-8");
header("Content-Transfer-Encoding: 8bit");
```



How do I know the encoding?

- The author of a HTML document may include “meta” tags in the document, to give the HTTP server process a hint about what information to send in the HTTP header:

```
<meta http-equiv="Content-Type"
content="text/html; charset=UTF-8"/>
```

```
<meta http-equiv="Content-Transfer-
Encoding" content="quoted-printable"/>
```

- Lower priority than "true" HTTP headers

How do I know the encoding?

- In HTML5 there is a new standard tag, shorter to type :

```
<meta charset="UTF-8"/>
```

- Recognized by all web browsers which are HTML5 compliant (even if put in the header of a HTML4 file!)

How do I know the encoding?

- If the content is plain text (no markup), the only way to know the encoding is side information

  … or trying to guess !
- If there is a BOM (Byte Order Mark: FEFF) at the beginning of the file ⇒ UTF-16
- Sometimes the BOM is present and has been re-encoded in UTF-8: EF BB BF (“ï»¿” when looked through as ISO-8859-1)

How do I know the encoding?

⇒ A **XHTML** document may possibly carry the encoding information up to three ways

- A *Byte-Order Mark* at the beginning (bytes FF FE) signals that it's UTF-16
- This can also be specified in the XML declaration
- And can be said again in a HTML meta tag
- Those pieces of information had better agree !

How do I know the encoding?

- How should a small PHP script be designed to write « À très bientôt ! Grüße ! » in a small XMTML 1.0 file, encoded in UTF-8, with the maximum safety in making the encoding information redundant ?
  - as a fake UTF-8 BOM
  - in sending the HTTP header
  - in the XML header
  - in the HTML header ?

Membre fondateur de :
GORBONNE PARIS CITÉ
CAMPUS CONDORCET Paris-Aubervilliers

8-bit clean version

```
<?php
header("Content-Type: text/html; charset=UTF-8");
header("Content-Transfer-Encoding: 8bit");
echo "ï»¿<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n";
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
xml:lang="fr" lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
</head>
<body><p>Ã bientÃ´t ! GrÃ¼Ãe !</p></body></html>
```

Paranoid version

```
<?php
header("Content-Type: text/html; charset=UTF-8");
header("Content-Transfer-Encoding: 8bit");
echo "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n";
?>
<!DOCTYPE html "(bla, bla, bla ...)">
<html xmlns="http://www.w3.org/1999/xhtml"
xml:lang="fr" lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
</head>
<body><p>&#xC0; Bient&#xF4;t ! Gr&#xFC;&#xDF;e!
</p></body></html>
```

Paranoid version

```
<?php
header("Content-Type: text/html; charset=UTF-8");
header("Content-Transfer-Encoding: 8bit");
echo "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n";
?>
<!DOCTYPE html "(bla, bla, bla ...)">
<html xmlns="http://www.w3.org/1999/xhtml"
xml:lang="fr" lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
</head>
<body><p>&#xC0; Bient&#xF4;t ! Gr&#xFC;&#xDF;e!
</p></body></html>
```

Never trust anyone !

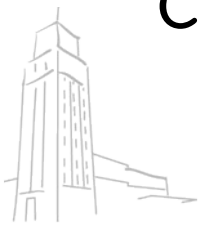
How do I know the encoding?

- In DBMS servers, there is a way to specify the internal encoding (here MySQL ex.):

```
CREATE TABLE (name VARCHAR(32)
    CHARACTER SET utf8
    COLLATE utf8_general_ci,
  PRIMARY KEY (name));
```

- At the database level:

```
CREATE DATABASE somedatabase
  DEFAULT CHARACTER SET utf8
  DEFAULT COLLATE utf8_general_ci
```



How do I know the encoding?

- To see the available character sets:

```
SHOW CHARACTER SET;
```

- To see the available collations:

```
SHOW COLLATION;
```

How do I know the encoding?

To set the character encoding used between the HTTP server and the DBMS (here in PHP, with the MySQLi extension):

```
$lien_bd=new mysqli('localhost',$id,
$pwd,'somedatabase');
$lien_bd->set_charset("utf8");
```

How do I know the encoding?

- What happens when the server and the client have not exchanged information about character encoding?
  - **In the case where the server sends UTF-8 and the client configuration is ISO-8859 ?**
  - Same question with ISO-8859 and UTF-8 ?
  - Same question with ISO-8859 and UTF-16 ?

“déjà” = 64 E9 6A E0 : dÃ©jÃ

“größer” = 67 72 F6 DF 65 72 : grÃ¶Ã er

How do I know the encoding?

- What happens when the server and the client have not exchanged information about character encoding?
  - In the case where the server sends UTF-8 and the client configuration is ISO-8859 ?
  - **Same question with ISO-8859 and UTF-8 ?**
  - Same question with ISO-8859 and UTF-16 ?

“déjà” = 64 E9 6A E0 : d��

“größer” = 67 72 F6 DF 65 72 : gr�er

How do I know the encoding?

- What happens when the server and the client have not exchanged information about character encoding?
  - In the case where the server sends UTF-8 and the client configuration is ISO-8859 ?
  - Same question with ISO-8859 and UTF-8 ?
  - **Same question with ISO-8859 and UTF-16 ?**

"déjà" = 64 E9 6A E0 : 摮櫰

"größer" = 67 72 F6 DF 65 72 : 杲㊀敲

How do I know the encoding?

- What happens when the DBMS has sent text in UTF-8, which has been interpreted as ISO-8859-1 by the Apache server ...

  and then the Apache server has re-encoded it in UTF-8 (trying to be helpful) ...

  and then the client browser thinks it is ISO-8859-1?

  “größer” = 67 72 F6 DF 65 72

  → 67 72 C3 B6 C3 9F 65 72

  → 67 72 C3 83 C2 B6 C3 83 C2 9F 65 72

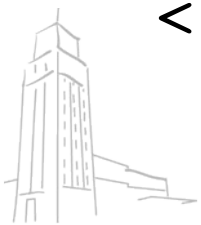
  “grÃ�Â¶Ã�Âùer”

Marking the language

How do I know the language?

- XML provides an attribute applicable to all elements in all namespaces : *lang*
- Specified as xml:lang for the namespace
- Example:

```
<book xml:lang="de">
  <title>Der Golem</title>
  ...
</book>
```



What's in a language tag?

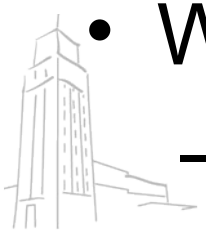
- This is specified by BCP-47 and RFC-5646
  - 1. Language code itself
    - ISO-639-1: de, fr, en, pt
    - ISO-639-3: deu, fra, nds, sxu, pcd
  - 2. An optional country code:
    - en-GB, en-US
  - 3. An optional script code:
    - sr-Cyrl, sr-Latn
  - 4. An optional variety code

What's in a language tag?

- **The language code**
- Defined by the ISO-639 series of standards
- ISO-639-1 and 2 : *Library of Congress*
  - ISO-639-1 : major languages of the world
  - ISO-639-2 : other written languages
- ISO-639-3 : SIL International
  - ISO-639-3 : comprehensive list of known languages (incl. macrolanguages)

What's in a language tag?

- **The country code**
- When useful to distinguish regional varieties
- Country code selected in the ISO-3166 list
  - Examples: en-GB, en-US, de-DE, de-AT
- If it is meaningful, a *world area* code may be used instead of the country code
- World area codes: UN M49 area codes
  - Examples: en-029, es-005, es-419



What's in a language tag?

- **The script code**
- To note the script (writing system) used
- Standard tags listed in ISO 15924
- Examples:
  - tr-Arab: Turkish written in Arabic script
  - tr-Latn: Turkish written in Latin script
  - zh-Hant: Chinese traditional characters
  - zh-Hans: Chinese simplified characters

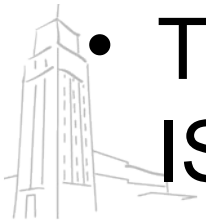
What's in a language tag?

- **The variant code**
- May denote a dialectal variety
  - hy-arevela: Eastern Armenian
- May denote a variant orthography
  - de-1901: Traditional German orthography
  - de-1996: New German orthography
- With the script code: a variant translitteration:
  - zh-Latn-pinyin: Chinese romanized with pinyin
  - zh-Latn-wadegile: Chinese romanized with Wade-Giles

Standard for language tags?

- The language codes proper (primary language subtags) have their standard: ISO-639

  (for Unicode applications : ISO-639-1 preferred over ISO-639-3 when available)
- The country codes have their standard: ISO-3166
- The script codes have their standard: ISO-15924

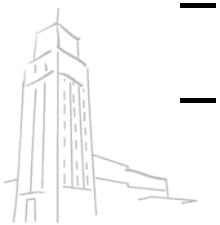


Standard for language tags?

- The variant codes are agreed upon by academic and industry active players
- The actual list of agreed-upon codes is the *IANA Language Subtag Registry*

http://www.iana.org/assignments/language-subtag-registry

- 7 types of entries :
  - language, extlang, script, variant, region
  - grandfathered, redundant



Content Negociation

Building a multilingual website

- **Website #0**: is written in one language, does not even bother to tell the world in which language it is written: ***BAD***
- **Website #1**: is written in one language, but tells the world in which character set it is encoded and in which language it is written: ***OK*** (at least we're doing what we can...)

Membre fondateur de :

Building a multilingual website

- **Website #2**: has at least part of its contents in different languages, every page with proper markup and links to other versions: ***Good***.
- **Website #3**: able to display contents in different languages, and to automatically adjust to the client's character set and language preferences : ***Excellent !***

Membre fondateur de :
SORBONNE PARIS CITÉ
CAMPUS CONDORCET Paris-Aubervilliers

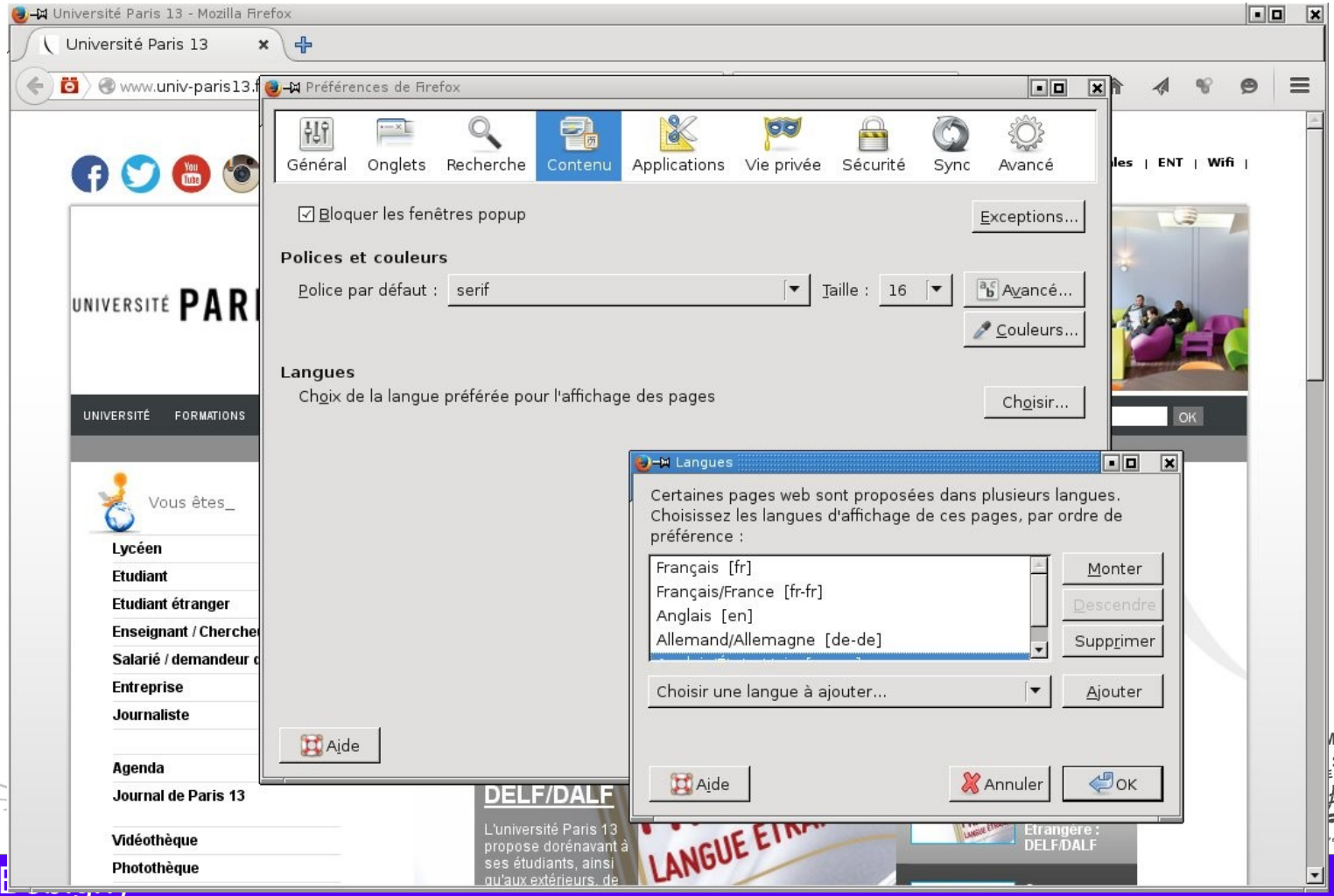
Content Negociation

- *Content Negotiation* presupposes that one single URI may give access to different contents, varying along some parameters (language, charset, encoding, size, display media …)
- *Content Negotiation* implies a mechanism by which client programs may signal what they accept, and server programs may select what they send

Membre fondateur de :
SORBONNE PARIS CITÉ
CAMPUS CONDORCET Paris-Aubervilliers

Content Negociation

On most web browsers, users may define language preferences:



Content Negociation

- User preferences are transmitted from the client to the server using the header attribute *Accept-Language* (defined in HTTP/1.1 by RFC 2616)

  Accept-Language: fr, en, en-US, de-de;q=0.8

- Reference :

  `http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html`

- The method also works for *Accept-Charset* and *Accept-Encoding*

Content Negociation

- User preferences are transmitted from the client to the server using the header attribute *Accept-Language* (defined in HTTP/1.1 by RFC 2616)

  Accept-Language: fr, en, en-US, de-de;q=0.8

- Reference :

  `http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html`

- The method also works for *Accept-Charset* and *Accept-Encoding*

Transparent Content Negotiation

- In *Transparent Content Negociation*, the server may send a list of available choices, and the client pick its best choice

  ⇒ no need to send long *Accept-** headers
- two-sided adaptation between client and server
- TCN defined in RFC 2295 (1998)

  `http://www.ietf.org/rfc/rfc2295.txt`

Membre fondateur de :

Transparent Content Negotiation

- The client sends a GET request with an additional *Negociate: trans* header (*Accept-** headers are optional)

Transparent Content Negotiation

- The server sends a list of alternatives
- In the HTTP headers :

```
HTTP/1.1 300 Multiple Choices
TCN: list
Alternates: {"fr_page" 1 {type text/html}
{language fr}}, {"en_page" 0.9 {type
text/html} {language en}}
```

- There must also be some HTML content in the response, in case the client does not support TCN

Transparent Content Negotiation

- The client then picks up its preferred variant
- And then the server sends a response with the HTTP header :

  TCN: choice

- A choice response should also contain a list of alternatives
- The HTTP headers should contain a *Vary:* * line

Transparent Content Negotiation

- Finally, the HTML header must also contain the information about the alternate versions:

```
<link rel="alternate" href="fr_page"
hreflang="fr"/>
<link rel="alternate" href="en_page"
hreflang="en"/>
<link rel="alternate" href="fr_page"
hreflang="x-default"/>
```