XML Technologies
Unicode

Christian Grün
Database & Information Systems Group
Universität Konstanz

Text Encodings

- if an input stream is to be interpreted as text, the incoming bytes must be converted to characters
- figure on the right: predominant ASCII mapping (128 *code points*)
- in the past, the big variety of text encodings frequently caused surprises
- sample UTF-8 text parsed as CP-1252: `Hier gÃ¤be es gute GrÃ¼nde fÃ¼r die Nutzung von Unicode!`

Ctrl	Dec	Hex	Char	Code	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
^@	0	00		NUL	32	20		64	40	@	96	60	'
^A	1	01		SOH	33	21	!	65	41	A	97	61	a
^B	2	02		STX	34	22	"	66	42	B	98	62	b
^C	3	03		ETX	35	23	#	67	43	C	99	63	c
^D	4	04		EOT	36	24	$	68	44	D	100	64	d
^E	5	05		ENQ	37	25	%	69	45	E	101	65	e
^F	6	06		ACK	38	26	&	70	46	F	102	66	f
^G	7	07		BEL	39	27	'	71	47	G	103	67	g
^H	8	08		BS	40	28	(	72	48	H	104	68	h
^I	9	09		HT	41	29	)	73	49	I	105	69	i
^J	10	0A		LF	42	2A	*	74	4A	J	106	6A	j
^K	11	0B		VT	43	2B	+	75	4B	K	107	6B	k
^L	12	0C		FF	44	2C	,	76	4C	L	108	6C	l
^M	13	0D		CR	45	2D	-	77	4D	M	109	6D	m
^N	14	0E		SO	46	2E	.	78	4E	N	110	6E	n
^O	15	0F		SI	47	2F	/	79	4F	O	111	6F	o
^P	16	10		DLE	48	30	0	80	50	P	112	70	p
^Q	17	11		DC1	49	31	1	81	51	Q	113	71	q
^R	18	12		DC2	50	32	2	82	52	R	114	72	r
^S	19	13		DC3	51	33	3	83	53	S	115	73	s
^T	20	14		DC4	52	34	4	84	54	T	116	74	t
^U	21	15		NAK	53	35	5	85	55	U	117	75	u
^V	22	16		SYN	54	36	6	86	56	V	118	76	v
^W	23	17		ETB	55	37	7	87	57	W	119	77	w
^X	24	18		CAN	56	38	8	88	58	X	120	78	x
^Y	25	19		EM	57	39	9	89	59	Y	121	79	y
^Z	26	1A		SUB	58	3A	:	90	5A	Z	122	7A	z
^[	27	1B		ESC	59	3B	;	91	5B	[	123	7B	{
^\	28	1C		FS	60	3C	<	92	5C	\	124	7C	¦
^]	29	1D		GS	61	3D	=	93	5D	]	125	7D	}
^^	30	1E	▲	RS	62	3E	>	94	5E	^	126	7E	~
^-	31	1F	▼	US	63	3F	?	95	5F	_	127	7F	⌂

Source: `msdn.microsoft.com/en-us/library/60ecse8t(v=vs.80).aspx`

Unicode Initiative

- Unicode provides an encoding that embraces the characters of most languages of the world, and numerous additional symbols:
- range of the Unicode encoding: 0x0000-0x10FFFF (16 × 65536 characters):
  - codes that fit into the first 16 bits (denoted U+0000–U+FFFF) have been assigned to encode the most widely used languages and their characters (*Basic Multilingual Plane*, BMP).
  - codes U+00–U+7F have been assigned to match the ASCII encoding
- Unicode 6.0 will introduce numerous new decorative symbols:

  

- **UTF** (the *Unicode Transformation Format*) defines how a Unicode character is mapped into units of 32, 16 and 8 bits

Unicode 6.0

- for character nerds: buy decodeunicode (Hermann Schmidt Verlag, 2011)
- for the digital folks: check out Gerrit Imsieke's Unicode primer:

publishinggeekly.com/wp-content/uploads/2012/07/dr_strangechar.html

UTF: Unicode Transformation Format

UTF-32

- characters in the range U+0000–U+10FFFF are mapped to the corresponding 32-bit value 0x00000000–0x0010FFFF
- example transformation for the two characters `Hi`:
  `00000000 00000000 00000000 01001000`
  `00000000 00000000 00000000 01101001`
- example transformation for the two characters 日本:
  `00000000 00000000 01100101 11100101`
  `00000000 00000000 01100111 00101100`
- advantage: fixed-length encoding

✎ what may be disadvantages of this encoding?

UTF-16

- characters are mapped to the following mapping scheme:
  `U+000000-U+00FFFF: ________ ________`
  `U+010000-U+10FFFF: 110110__ ________ 110111__ ________`
- for values ≤ 0x00FFFF, replace spaces with the character code
- otherwise, subtract 0x010000 and fill spaces with resulting 20-bit value
- example transformation for the two characters `Hi`:
  `00000000 01001000 00000000 01101001`
- example transformation for the two characters 日本:
  `01100101 11100101 01100111 00101100`
- advantage: takes less space than UTF-16

✎ do you think this can be further optimized?

UTF-8

- today's most prevalent encoding; popular because:
  - bytes (*octets*) are the basic processing unit of hardware
  - text-processing software today is built to deal with 8 bit character encodings
- mapping scheme:

```
U+000000-U+00007F: 0_______
U+000080-U+0007FF: 110_____ 10______
U+000800-U+00FFFF: 1110____ 10______ 10______
U+010000-U+10FFFF: 11110___ 10______ 10______ 10______
```

- example transformation for the two characters `Hi`:
  `01001000 01101001`
- example transformation for the two characters 日本:
  `11100110 10010111 10100101 11100110 10011100 10101100`

Advantages of UTF-8

- length of a byte sequence can be computed by counting the *leading 1-bits* (single-byte characters have a leading 0-bit)
- *character boundaries* are simple to detect (even when place at some arbitrary position in a UTF-8 byte stream)
- encoding does not affect (binary) *sort order*
- traditional 7-bit ASCII texts take *little space*
- text processing software which was originally developed to work with ASCII remains functional

✎ once again… are there any downsides around UTF-8?

XML: Text Declarations

- a well-formed XML document may start off with an optional header, the *text declaration*
- example: `<?xml version="1.0" encoding="UTF-8"?>`
- it includes the **XML version** and the **document encoding**
- defined XML versions so far: `1.0`; `1.1` (only few implementations)
- the encoding specifies how bytes in the input stream are to be interpreted
- most popular encodings:
  - `US-ASCII`: 128 characters; most-upper bit of all characters is always `0`
  - `ISO-8859-1`: character set with most popular Western characters
  - Unicode (`UTF-8`, `UTF-16`, `UTF-32`):
    provides the full range of characters; slowly rules out all other formats

BOM: Byte Order Marks

- some text editors and tools prefix text files with *byte order marks*
- BOMs are mainly used for Unicode formats; they indicate how characters are stored in a file:

Encoding	BOM	Character 0x3F (?)
UTF-8	EF BB BF	00111111
UTF-16 (BE)	FE FF	00000000 00111111
UTF-16 (LE)	FF FE	00111111 00000000

- if no BOM exists, encoding must be *guessed* by the application

Let's switch back to the XML context...

- the *text declaration* provides a generic way to specify the text encoding
- encoding is still guessed when reading the first bytes ✎ ...why?

Outlook: XQuery and Unicode

- all XQuery expressions and functions work with Unicode – encoding issues are history!
- functions exist to convert strings to code points, and vice versa:
```
string-to-codepoints('abc')
codepoints-to-string((97, 98, 99))
```
- BaseX provides additional functions to explicitly convert input and output to specific encodings:
```
docs.basex.org/wiki/File_Module
docs.basex.org/wiki/Conversion_Module
```
- the following sample query returns the hexadecimal byte representation of a specific string and encoding:
```
file:read-text('input.txt', 'ISO-8859-1') !
  convert:string-to-hex(., 'UTF-16LE')
```