ICM – Computer Science Major
M1 Cyber Physical and Social Systems
Course unit on Data Interoperability and Semantics

Tutorial 1

1) Bluetooth Mesh Model

The definitions below are extracts of a Bluetooth standard. Complete the tables.

Bluetooth Mesh Model specification v1.0.1 Section 3.1.6.1 Generic Battery Level *(modified)*
The Generic Battery Level state is a value ranging from 0 percent through 100 percent. The values must be divided by two (ex., 0x03 stands for 1.5 %). Values above 0xC8 (100 %) are forbidden.

	encoded	decoded
8-bit unsigned integer (UINT8)	0xA7 (hexa)	
UINT8	0b01010110 (binary)	

Bluetooth Mesh Model specification v1.0.1 Section 3.1.7.6 Local Altitude
The Local Altitude field determines the altitude of the device relative to the Generic Location Global Altitude. This is a 16-bit signed integer in decimeters. The valid range is from -32768 decimeters (0x8000) through 0 decimeters (0x0000) up to 32765 decimeters (0x7FFD).
The following formula can be used to decode data: $B2T(X) = -x_{w-1} \cdot 2^{w-1} + \sum_{i=0}^{w-2} x_i \cdot 2^i$

	encoded	decoded
INT16 - Big Endian (AB)	0x8001	
INT16 - Little Endian (BA)	0x8001	

2) IEEE 754

Decode the following hexadecimal strings using the Single precision IEEE 754 floating-point standard:

a) 0x3F000000
b) 0x44800000

Encode the following numbers using the Single precision IEEE 754 floating-point standard:

c) -3.0
d) 24.0

3) UTF-8

Convert the following strings to UTF-8:

a) cs
b) ¥
c) €
d) 😃

4) CBOR

Below is a document encoded using the Concise Binary Object Representation.
Write a JSON equivalent of this document.
Spaces and new lines in the representation are just there to help you.

82 A3 61 76 FA 3F800000 62 6363 63 455552 62 6373 63 E282AC
A3 61 76 FA 40F80000 62 6363 63 434E59 62 6373 62 C2A5

5) Base32 decoding

Below is the RFC 4648 Base32 Alphabet

Value	Encoding	Value	Encoding	Value	Encoding	Value	Encoding
0	A	9	J	18	S	27	3
1	B	10	K	19	T	28	4
2	C	11	L	20	U	29	5
3	D	12	M	21	V	30	6
4	E	13	N	22	W	31	7
5	F	14	O	23	X		
6	G	15	P	24	Y	(pad)	=
7	H	16	Q	25	Z		
8	I	17	R	26	2		

Decode the following string: `JVSWK5BAJJXWKICAGRYG2II=`

Sequence of values:

Sequence of 5 bits:

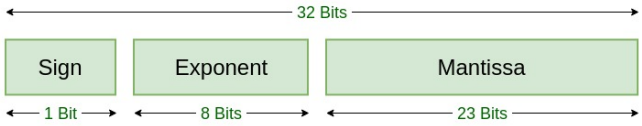
Decoded string:

6) CRLF

Different OSs represent ends of line differently.
Microsoft Windows uses CRLF, Unix uses only LF, Classic MacOS used CR, then LF since Mac OS X.

a) What is the full name of CR and LF?
b) What are their ASCII codes?
c) With what escape characters can these characters be included in a JSON string?
d) Why is it problematic that different OSs use different end-of-line characters ?

Appendix A: Single precision IEEE 754 floating-point standard



$$-1^{s} \times 2^{(\exp-127)} \times 1.frac$$

As an example, the value for number 0xbfc00000 is -1.5

Appendix B: UTF-8 Encoding

UTF-8 encodes code points in one to four bytes, depending on the value of the code point. In the following table, the `x` characters are replaced by the bits of the code point:

Code point ↔ UTF-8 conversion					
First code point	**Last code point**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4**
U+0000	U+007F	`0xxxxxxx`			
U+0080	U+07FF	`110xxxxx`	`10xxxxxx`		
U+0800	U+FFFF	`1110xxxx`	`10xxxxxx`	`10xxxxxx`	
U+10000	U+10FFFF	`11110xxx`	`10xxxxxx`	`10xxxxxx`	`10xxxxxx`

The first 128 code points (ASCII) need one byte. The next 1,920 code points need two bytes to encode, which covers the remainder of almost all Latin-script alphabets, and also IPA extensions, Greek, Cyrillic, Coptic, Armenian, Hebrew, Arabic, Syriac, Thaana and N'Ko alphabets, as well as Combining Diacritical Marks. Three bytes are needed for the remaining 61,440 code points of the Basic Multilingual Plane (BMP), including most Chinese, Japanese and Korean characters. Four bytes are needed for the 1,048,576 code points in the other planes of Unicode, which include emoji (pictographic symbols), less common CJK characters, various historic scripts, and mathematical symbols.

A "character" can take more than 4 bytes because it is made of more than one code point. For instance a national flag character takes 8 bytes since it is "constructed from a pair of Unicode scalar values" both from outside the BMP.

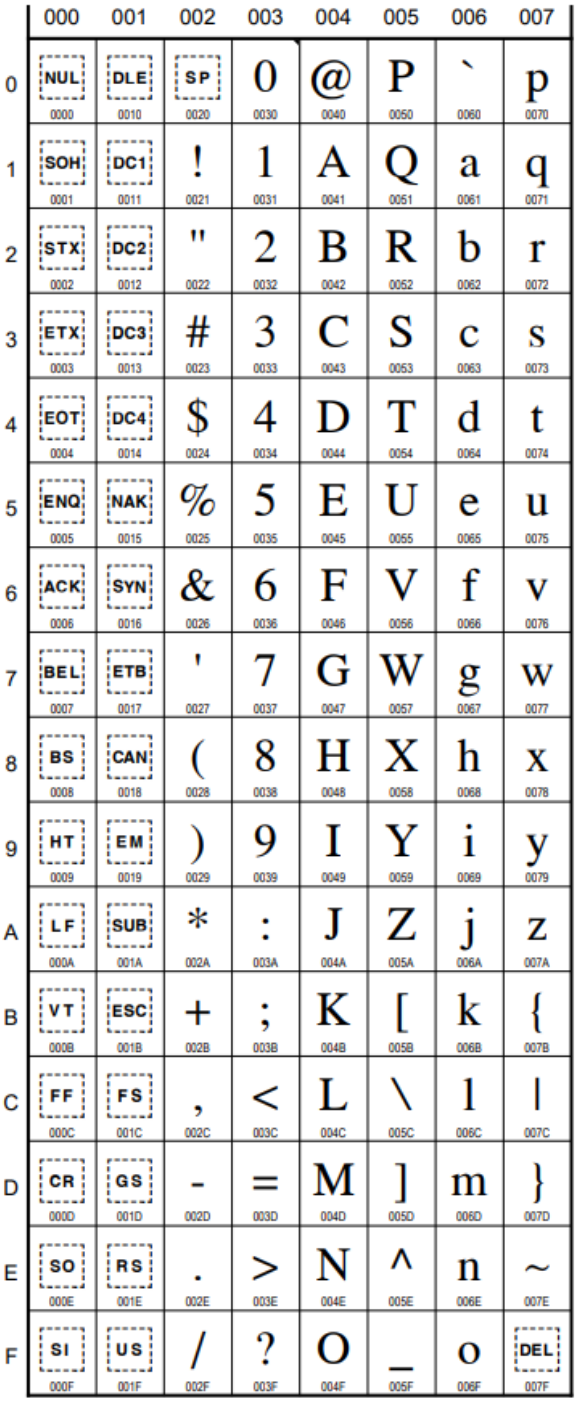
Encoding process

In these examples, character '|' indicate how bits from the code point are separated and distributed among the UTF-8 bytes. Additional bits added by the UTF-8 encoding process are underlined.

1. The Unicode code point for the euro sign € is U+20AC.
2. As this code point lies between U+0800 and U+FFFF, this will take three bytes to encode.
3. Hexadecimal `20AC` is binary `0010 | 0000 10|10 1100`. The two leading zeros are added because a three-byte encoding needs exactly sixteen bits from the code point.
4. Because the encoding will be three bytes long, its leading byte starts with three 1s, then a 0 (`1110...`)
5. The four most significant bits of the code point are stored in the remaining low order four bits of this byte (`11100010`), leaving 12 bits of the code point yet to be encoded (`...0000 10|10 1100`).
6. All continuation bytes contain exactly six bits from the code point. So the next six bits of the code point are stored in the low order six bits of the next byte, and `10` is stored in the high order two bits to mark it as a continuation byte (so `10000010`).
7. Finally the last six bits of the code point are stored in the low order six bits of the final byte, and again `10` is stored in the high order two bits (`10101100`).

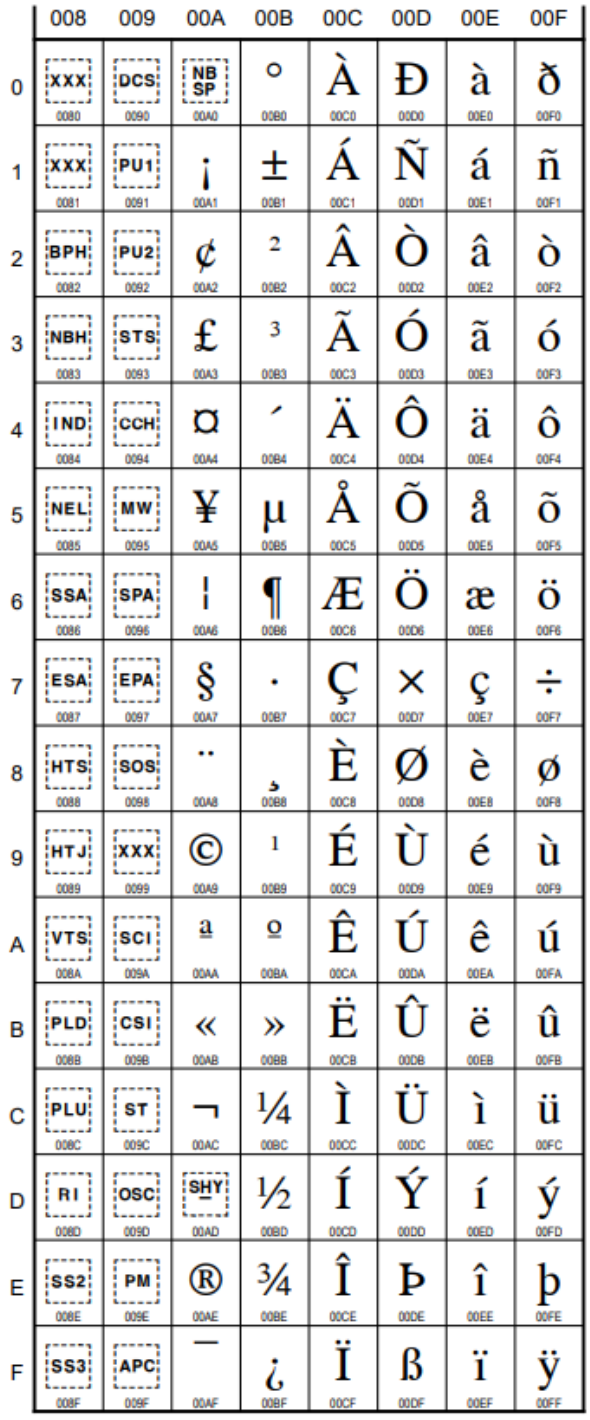
The three bytes `11100010 10000010 10101100` can be more concisely written in hexadecimal, as `E2 82 AC`.

Appendix C: Some Unicode code points



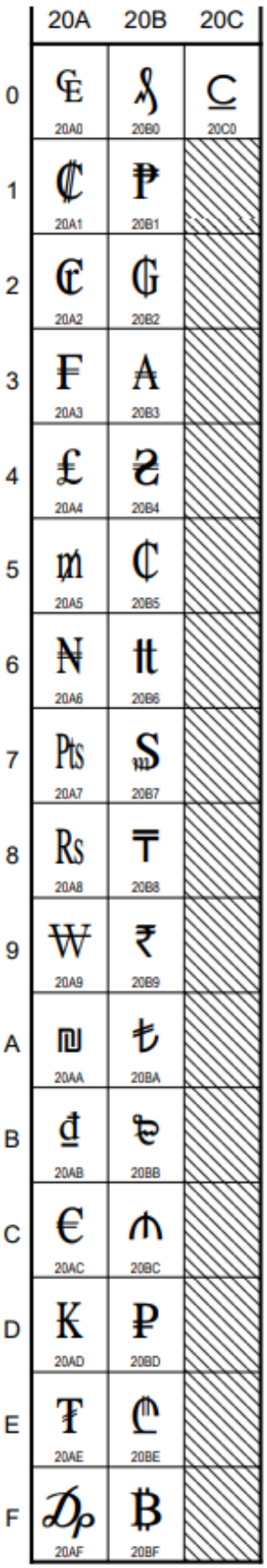
	000	001	002	003	004	005	006	007
0	NUL 0000	DLE 0010	SP 0020	0 0030	@ 0040	P 0050	` 0060	p 0070
1	SOH 0001	DC1 0011	! 0021	1 0031	A 0041	Q 0051	a 0061	q 0071
2	STX 0002	DC2 0012	" 0022	2 0032	B 0042	R 0052	b 0062	r 0072
3	ETX 0003	DC3 0013	# 0023	3 0033	C 0043	S 0053	c 0063	s 0073
4	EOT 0004	DC4 0014	$ 0024	4 0034	D 0044	T 0054	d 0064	t 0074
5	ENQ 0005	NAK 0015	% 0025	5 0035	E 0045	U 0055	e 0065	u 0075
6	ACK 0006	SYN 0016	& 0026	6 0036	F 0046	V 0056	f 0066	v 0076
7	BEL 0007	ETB 0017	' 0027	7 0037	G 0047	W 0057	g 0067	w 0077
8	BS 0008	CAN 0018	(0028	8 0038	H 0048	X 0058	h 0068	x 0078
9	HT 0009	EM 0019	) 0029	9 0039	I 0049	Y 0059	i 0069	y 0079
A	LF 000A	SUB 001A	* 002A	: 003A	J 004A	Z 005A	j 006A	z 007A
B	VT 000B	ESC 001B	+ 002B	; 003B	K 004B	[005B	k 006B	{ 007B
C	FF 000C	FS 001C	, 002C	< 003C	L 004C	\ 005C	l 006C	| 007C
D	CR 000D	GS 001D	- 002D	= 003D	M 004D	] 005D	m 006D	} 007D
E	SO 000E	RS 001E	. 002E	> 003E	N 004E	^ 005E	n 006E	~ 007E
F	SI 000F	US 001F	/ 002F	? 003F	O 004F	_ 005F	o 006F	DEL 007F

Basic Latin Range: U+0000 – U+007F

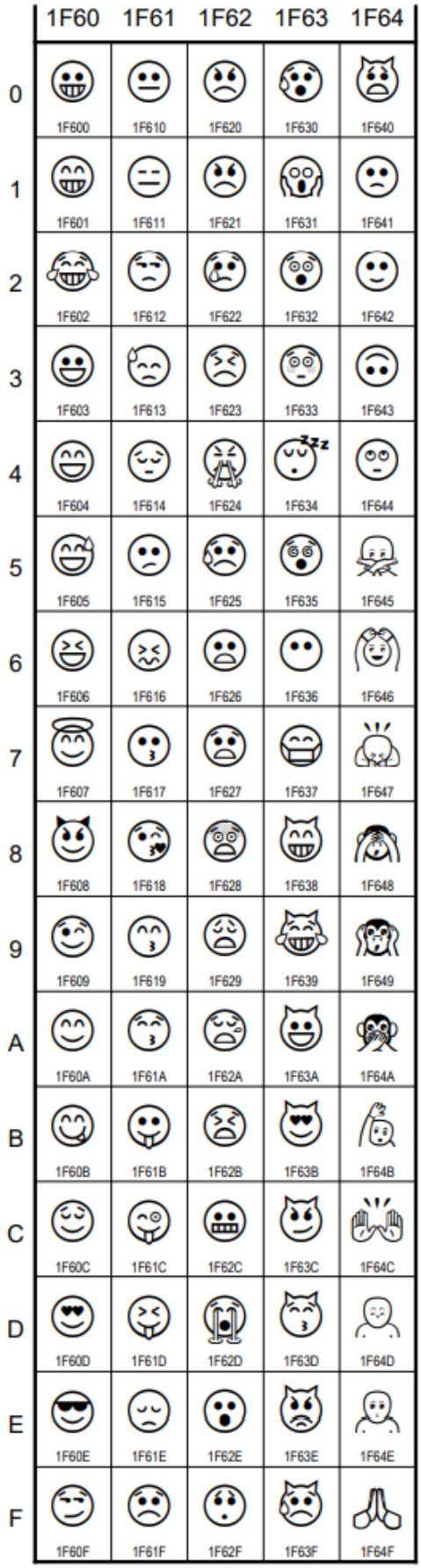


	008	009	00A	00B	00C	00D	00E	00F
0	XXX 0080	DCS 0090	NB SP 00A0	° 00B0	À 00C0	Ð 00D0	à 00E0	ð 00F0
1	XXX 0081	PU1 0091	¡ 00A1	± 00B1	Á 00C1	Ñ 00D1	á 00E1	ñ 00F1
2	BPH 0082	PU2 0092	¢ 00A2	² 00B2	Â 00C2	Ò 00D2	â 00E2	ò 00F2
3	NBH 0083	STS 0093	£ 00A3	³ 00B3	Ã 00C3	Ó 00D3	ã 00E3	ó 00F3
4	IND 0084	CCH 0094	¤ 00A4	´ 00B4	Ä 00C4	Ô 00D4	ä 00E4	ô 00F4
5	NEL 0085	MW 0095	¥ 00A5	µ 00B5	Å 00C5	Õ 00D5	å 00E5	õ 00F5
6	SSA 0086	SPA 0096	¦ 00A6	¶ 00B6	Æ 00C6	Ö 00D6	æ 00E6	ö 00F6
7	ESA 0087	EPA 0097	§ 00A7	· 00B7	Ç 00C7	× 00D7	ç 00E7	÷ 00F7
8	HTS 0088	SOS 0098	¨ 00A8	¸ 00B8	È 00C8	Ø 00D8	è 00E8	ø 00F8
9	HTJ 0089	XXX 0099	© 00A9	¹ 00B9	É 00C9	Ù 00D9	é 00E9	ù 00F9
A	VTS 008A	SCI 009A	ª 00AA	º 00BA	Ê 00CA	Ú 00DA	ê 00EA	ú 00FA
B	PLD 008B	CSI 009B	« 00AB	» 00BB	Ë 00CB	Û 00DB	ë 00EB	û 00FB
C	PLU 008C	ST 009C	¬ 00AC	¼ 00BC	Ì 00CC	Ü 00DC	ì 00EC	ü 00FC
D	RI 008D	OSC 009D	SHY 00AD	½ 00BD	Í 00CD	Ý 00DD	í 00ED	ý 00FD
E	SS2 008E	PM 009E	® 00AE	¾ 00BE	Î 00CE	Þ 00DE	î 00EE	þ 00FE
F	SS3 008F	APC 009F	¯ 00AF	¿ 00BF	Ï 00CF	ß 00DF	ï 00EF	ÿ 00FF

Latin-1 Supplement: U+0080 – U+00FF



Currency Symbols U+20A0 – U+20C0



Emoji symbols U+1F600 – U+1F64F

Appendix D: RFC 8949 - Concise Binary Object Representation (CBOR)

Concise Binary Object Representation (CBOR) is a binary data serialization format loosely based on JSON authored by C. Bormann. Like JSON it allows the transmission of data objects that contain name–value pairs, but in a more concise manner. This increases processing and transfer speeds at the cost of human readability. It is defined in IETF RFC 8949

Specification of the CBOR encoding
CBOR encoded data is seen as a stream of data items. Each data item consists of a header byte containing a 3-bit type and 5-bit short count. This is followed by an optional extended count (if the short count is in the range 24–27), and an optional payload.

For types 0, 1, and 7, there is no payload; the count is the value. For types 2 (byte string) and 3 (text string), the count is the length of the payload. For types 4 (array) and 5 (map), the count is the number of items (pairs) in the payload.

CBOR cheatsheet

Major type	Description	Binary	Shorthand
0	an unsigned integer	000_xxxxx	unsigned(#)
1	a negative integer	001_xxxxx	negative(#-1)
2	a byte string	010_xxxxx	bytes(n)
3	a text string (UTF-8)	011_xxxxx	text(n)
4	an array of data items	100_xxxxx	array(n)
5	a map of pairs of data items	101_xxxxx	map(n)
7	Floating-point numbers and values with no content	111_xxxxx	simple(v)

[1] *For negative(#-1), # represents the negative value minus 1. For example, negative(4) represents a value of -5.*
[5] *For map(n), n represents number of key/value pairs.*

Unsigned integer

Number	Hex
0 .. 15	00 .. 0F
16 .. 23	10 .. 17
24 .. 255	18 18 .. 18 FF
256 .. 65535	19 0100 .. 19 FFFF

Negative integer

Number	Hex
-1 .. -16	20 .. 2F
-17 .. -24	30 .. 37
-25 .. -256	38 18 .. 38 FF
-257 .. -65536	39 0100 .. 39 FFFF

Array

Length	*Hex*
0 .. 15	*80 .. 8F*
16 .. 23	*90 .. 97*
24 .. 255	*98 18 .. 98 FF*
256 .. 65535	*99 0100 .. 99 FFFF*
Indefinite	*9F*

Length represents number of key/value pairs.

Map

Length	*Hex*
0 .. 15	*A0 .. AF*
16 .. 23	*B0 .. B7*
24 .. 255	*B8 18 .. B8 FF*
256 .. 65535	*B9 0100 .. B9 FFFF*
Indefinite	*BF*

Length represents number of key/value pairs.

Floating point numbers and values with no content

5-Bit Value Semantics

20	false
21	true
22	null
23	undefined
25	IEEE 754 Half-Precision Float (16 bits follow)
26	IEEE 754 Single-Precision Float (32 bits follow)
27	IEEE 754 Double-Precision Float (64 bits follow)
31	"break" stop code for indefinite-length items

Examples:

As JSON:	As CBOR:
{	A2 # map(2)
"t":2,	61 # text(1)
"h":null	74 # "t"
}	02 # unsigned(2)
	61 # text(1)
	68 # "h"
	F6 # primitive(22)