Open Source compliance: Technical must-knows for legal experts

Open Source Automation Development Lab (OSADL) eG

Binary encoding of strings (words, names etc.)

How are data stored in a computer?

- By principle, computers can only store
  - bits – a single storage cell that can be 0 or 1
  - a sequence of bits:
    - Bytes – a sequence of 8 bits (0 to 255)

How are data stored in a computer?

- By principle, computers can only store
  - bits – a single storage cell that can be 0 or 1
  - a sequence of bits:
    - Bytes – a sequence of 8 bits (0 to 255)
    - Short words - a sequence of 16 bits (0 to 65,545)

How are data stored in a computer?

- By principle, computers can only store
  - bits – a single storage cell that can be 0 or 1
  - a sequence of bits:
    - Bytes – a sequence of 8 bits (0 to 255)
    - Short words - a sequence of 16 bits (0 to 65,545)
    - Long words - a sequence of 32 bits (0 to 4,294,967,295)

How are data stored in a computer?

- By principle, computers can only store
  - bits – a single storage cell that can be 0 or 1
  - a sequence of bits:
    - Bytes – a sequence of 8 bits (0 to 255)
    - Short words - a sequence of 16 bits (0 to 65,545)
    - Long words - a sequence of 32 bits (0 to 4,294,967,295)

4 billion

How are data stored in a computer?

- By principle, computers can only store
  - bits – a single storage cell that can be 0 or 1
  - a sequence of bits:
    - Bytes – a sequence of 8 bits (0 to 255)
    - Short words - a sequence of 16 bits (0 to 65,545)
    - Long words - a sequence of 32 bits (0 to 4,294,967,295
    - Long long words - a sequence of 64 bits (0 to 18,446,744,073,709,551,615)

4 billion

How are data stored in a computer?

- By principle, computers can only store
  - bits – a single storage cell that can be 0 or 1
  - a sequence of bits:
    - Bytes – a sequence of 8 bits (0 to 255)
    - Short words – a sequence of 16 bits (0 to 65,545)
    - Long words – a sequence of 32 bits (0 to 4,294,967,295
    - Long long words - a sequence of 64 bits (0 to 18,446,744,073,709,551,615)

4 billion

18 quintillion

Binary numbers

- A byte (8 bits) is a generic storage unit. For example, the size of a file indicates how many bytes a file is long.
- The decimal value of a given byte is calculated by taking the number 2 to the power of the bit position:

$$0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0$$

$$0*2^{7} + 0*2^{6} + 0*2^{5} + 0*2^{4} + 0*2^{3} + 0*2^{2} + 0*2^{1} + 0*2^{0} = 0$$

$$0*128 + 0*64 + 0*32 + 0*16 + 0*8 + 0*4 + 0*2 + 0*1 = 0$$

Range of binary numbers

- A byte (8 bits) is a generic storage unit. For example, the size of a file indicates how many bytes a file is long.
- The decimal value of a given byte is calculated by taking the number 2 to the power of the bit position:

$$\begin{array}{cccccccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{array}$$

$$1*2^7 + 1*2^6 + 1*2^5 + 1*2^4 + 1*2^3 + 1*2^2 + 1*2^1 + 1*2^0 = 255$$

$$1*128 + 1*64 + 1*32 + 1*16 + 1*8 + 1*4 + 1*2 + 1*1 = 255$$

Convert a binary number to decimal

- A byte (8 bits) is a generic storage unit. For example, the size of a file indicates how many bytes a file is long.
- The decimal value of a given byte is calculated by taking the number 2 to the power of the bit position:

1 0 0 1 0 0 0 1

$$1*2^{7} + 0*2^{6} + 0*2^{5} + 1*2^{4} + 0*2^{3} + 0*2^{2} + 0*2^{1} + 1*2^{0} = 145$$

$$1*128 + 0*64 + 0*32 + 1*16 + 0*8 + 0*4 + 0*2 + 1*1 = 145$$

What are hexadecimal numbers?

- Since binary notation creates very long numbers that are difficult to read, they are combined into pairs of 4 bits.
- The maximum possible number of a 4-bit binary is 15. To cover the numbers 10 to 15 that are not available in the decimal system the letters A to F were appended after the 9.

0	→	0
1	→	1
2	→	2
3	→	3
4	→	4
5	→	5
6	→	6
7	→	7
8	→	8
9	→	9
10	→	A
11	→	B
12	→	C
13	→	D
14	→	E
15	→	F
16	→	10

Convert a binary number to decimal

- A byte (8 bits) is a generic storage unit. For example, the size of a file indicates how many bytes a file is long.
- The decimal value of a given byte is calculated by taking the number 2 to the power of the bit position:

1 0 0 1 0 0 0 1

$$1*2^7 + 0*2^6 + 0*2^5 + 1*2^4 + 0*2^3 + 0*2^2 + 0*2^1 + 1*2^0 = 145$$

$$1*128 + 0*64 + 0*32 + 1*16 + 0*8 + 0*4 + 0*2 + 1*1 = 145$$

COMPACT
OSADL
ONLINE
LECTURES

Convert a binary number to decimal

- A byte (8 bits) is a generic storage unit. For example, the size of a file indicates how many bytes a file is long.
- The decimal value of a given byte is calculated by taking the number 2 to the power of the bit position:

1	0	0	1	0	0	0	1	
$1*2^3$ +	$0*2^2$ +	$0*2^1$ +	$1* 2^0$	$0*2^3$+	$0*2^2$ +	$0*2^1$ +	$1*2^0$	= 145
1*8	+ 0*4	+0*2	+ 1*1	0*8 +	0*4 +	0*2 +	1* 1	= 145

Convert a binary number to hexadecimal

- A byte (8 bits) is a generic storage unit. For example, the size of a file indicates how many bytes a file is long.
- The decimal value of a given byte is calculated by taking the number 2 to the power of the bit position:

$$\begin{array}{cccc|cccc}
1 & 0 & 0 & 1 & 0 & 0 & 0 & 1
\end{array}$$

$$1*2^3 + 0*2^2 + 0*2^1 + 1*2^0 \quad \Big| \quad 0*2^3 + 0*2^2 + 0*2^1 + 1*2^0 = 145$$

$$\underbrace{1*8 + 0*4 + 0*2 + 1*1}_{9} \quad \Big| \quad \underbrace{0*8 + 0*4 + 0*2 + 1*1}_{1} = 145$$

$$= 145$$

Some examples of hexadecimal numbers

8-bit computer

Hex		Dec
00	→	0
10	→	16
1F	→	31
20	→	32
80	→	128
91	**→**	**145**
FF	→	255

16-bit computer

Hex		Dec
0000	→	0
0010	→	16
0100	→	256
01ff	→	511
0400	→	1024
1000	→	4096
FFFF	→	64535

Formatting a hexadecimal number

- To mark a number as hexadecimal a leading “0x” is often used. For example, the command line tool “printf” can be used to decode and encode hexadecimal numbers:

```
$ printf "%d\n" 0x91
145

$ printf "%d\n" 91
91

$ printf "0x%2x\n" 145
0x91
```

Why need strings to be encoded?

- Since original computers only had a storage length of 8 bits, and the code ranged from 0 to FF, an encoding scheme hat to be invented to store letters, numbers, punctuation, some mathematical symbols and special characters as 8-bit words.
- The result was the **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange (**ASCII**) that uses the first 127 numbers (00 - 7F).
- For example, the numbers “0” to “9” are encoded as 30 to 39, the upper-case letters “A” to “Z” from 41 to 5A and the lower-case letters “a” to “z” from 61 to 7A.

American Standard Code for Information Interchange (ASCII)

20	21	22	23	24	25	26	27	28	29	2a	2b	2c	2d	2e	2f
	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
30	31	32	33	34	35	36	37	38	39	3a	3b	3c	3d	3e	3f
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	41	42	43	44	45	46	47	48	49	4a	4b	4c	4d	4e	4f
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	51	52	53	54	55	56	57	58	59	5a	5b	5c	5d	5e	5f
P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
60	61	62	63	64	65	66	67	68	69	6a	6b	6c	6d	6e	6f
`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	71	72	73	74	75	76	77	78	79	7a	7b	7c	7d	7e	7f
p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	

Example: Detect ASCII strings in a binary program

- Binary programs may contain strings that are encoded as ASCII binaries, for example static license or copyright notices.
- The program *strings* can search text snippets with a given length in a binary file and convert them to printable ASCII sequences that may be filtered with *grep*.
- For example:

```
$ strings /bin/bash | grep GPL
```

Example: Detect ASCII strings in a binary program

- Binary programs may contain strings that are encoded as ASCII binaries, for example static license or copyright notices.
- The program *strings* can search text snippets with a given length in a binary file and convert them to printable ASCII sequences that may be filtered with *grep*.
- For example:

```
$ strings /bin/bash | grep GPL
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
```