Text File Coding

Leonardo Araújo

UFSJ

Introduction to Text Encoding

- **Why Encoding Matters:**
  - Facilitates communication between computers and humans.
  - Ensures data integrity across different systems.
- **Overview:**
  - Evolution from simple to complex encoding systems.

Morse Code

- **History:** Developed by Samuel F.B. Morse in the early 1840s.
- **Mechanism:** Uses dots, dashes, and spaces for letters, numbers, and punctuation.
- **Usage:** Primarily telegraphy, but also in radio communication.

Figure 1: Morse Code

Jacquard

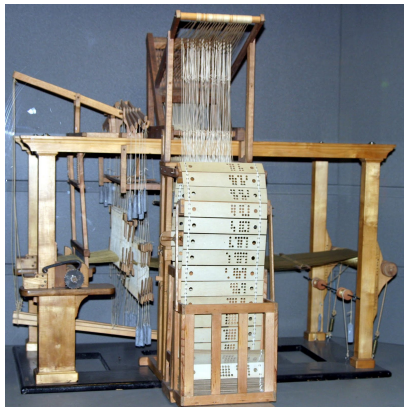- Joseph Marie Jacquard in Lyon in 1801.

Figure 2: Jacquard's loom.

Baudot and Murray Code

- **Baudot Code:**
  - Invented by Émile Baudot, 5-bit code for telegraphy.
  - Limited characters, used shift for numbers/letters.
- **Murray Code (ITA2):**
  - Extension of Baudot, improved by Donald Murray.
  - Added lower case, more symbols.

The International Telegraph Alphabet
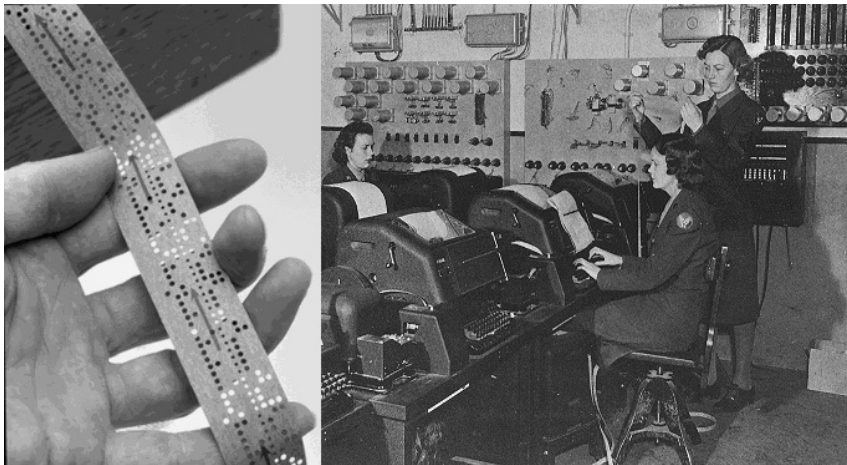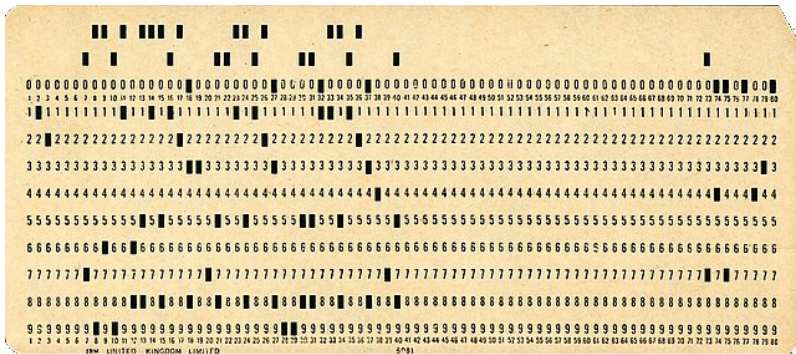
Figure 3: ITA2 Baudot-Murray code

Figure 4: Teletype and perforated strip.

EBCDIC (Extended Binary Coded Decimal Interchange Code)

- **History:** Developed by IBM for mainframe computers.
- **Characteristics:** Used in legacy systems.
- EBCD, a subset of EBCDIC.

Figure 5: 12-row/80-column IBM punched card and EBCD table.

ASCII and Extended ASCII

- **ASCII (American Standard Code for Information Interchange):**
  - 7-bit code, 128 characters including 33 non-printing control codes.
  - Standardized in 1963 (ANSI).
  - Backward Compatibility: Despite its age, ASCII remains widely used today for compatibility reasons.
- **Extended ASCII:**
  - 8-bit code, 256 characters, allowing for additional symbols and characters.

USASCII code chart

$b_7 b_6 b_5$ →					$0\,0\,0$	$0\,0\,1$	$0\,1\,0$	$0\,1\,1$	$1\,0\,0$	$1\,0\,1$	$1\,1\,0$	$1\,1\,1$
b_4	b_3	b_2	b_1	Column / Row	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l		
1	1	0	1	13	CR	GS	−	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	—	o	DEL

Figure 6: ASCII Table

Character	Binary (Uppercase)	Binary (Lowercase)	Character
A	01000001	01100001	a
B	01000010	01100010	b
C	01000011	01100011	c
. . .	. . .	. . .	. . .
Z	01011010	01111010	z
1	00110001	00100001	!
2	00110010	01000000	"
3	00110011	00100011	#
4	00110100	00100100	$
. . .	. . .	. . .	. . .

ASCII Art

```
 ------------------------
|.----------------------.|
||         _____        ||
||      .;;;;;;;;;.       ||
||     /;;;;;;;;;;;\      ||
||    /;/`      `-;;;;; . .||
||    |;|__   __  \;;;|   ||
||.-.|;| e`/e`   |;;;|   ||
||   |;|  |      |;;;|'--||
||   |;|  '-     |;;;|   ||
||   |;;\ --'  /|;;;|   ||
||   |;;;;;---'\|;;;|   ||
||   |;;;;|     |;;;|   ||
||   |;;.-'     |;;;|   ||
||'--|/`        |;;;|--.||
||;;;;    .    ;;;;-\;;||
||;;;;;-.;_   /.-;;;;;||
||;;;;;;;;;;;;;;;;;;;;;||
'------------------------'
```

BTC Genesis Block

Figure 7: The message embedded by Satoshi Nakamoto in Bitcoin's first block (the Genesis Block). The message reads, "The Times 03/Jan/2009 Chancellor on brink of second bailout for banks," which was a headline from The Times newspaper on that date.

Base64

Base64 is a binary-to-text encoding scheme that represents binary data in an ASCII string format. Each Base64 digit represents exactly 6 bits of data, providing a way to encode binary data as text.

Base64 is used in:

- email attachments,
- embedding binary data in XML, JSON, or HTML, and
- data exchange in APIs.

```
$ base64 /tmp/tux.png
```
iVBORw0KGgoAAAANSUhEUgAAADIAAAA7CAYAAAA5MNl5AAAAxHpUWHRSYXcgcHJvZmlsZSB0eXBl
IGV4aWYAAHjabVBbDsMgDPvnFDsCiVMIx6GPSbvBjr8AaVW2WcINceSahOP9eoZHA5MEWbKmklI0
SJHC1QqNA7UzRencAXaN5n7A6gJbC21yXDX5/Nmny2B8qlXLzUg3F9F9ZZZKOL++mXkidAStXp3o7Jd
kbtAblDHs2Iqmu9PWI84Q8cJjUTn2D/3bNvbF/sPmA8QojGgIwDakYBqAhszkg0SxGpB7p3TzBby
b08nwgcmkllHdJ9h5QAAAYRpQ0NQSUNDIHByb2ZpbGUAAHicfZE9SMNAHMVfU0WRiogdpDhkqLrY
RUUcaxWKUCHUCq06mFz6BU0akhQXR8G14ODHYtXBxVlXB1dBEPwAcXZwUnSREv+XFFrEeHDcj3f3
HnfvAKFRYZrVFQc03TbTyYSYza2KPa8IIYIBJbjCMoM8uYYk6QUfMfXPQJ8vYvxLP9zf45+Nw8xICAS
x5lh2sQbxDObtsF5nzjMIsr1KfE48YdDFiR+5rnj8xrnossAzw2YmPU8cJhaLLaxMMuzGvE0cVTV
dMoXsh6rnLc4a5Uaa92TvzCU11eWuU5zBEksYgkSRCiooYwkbMRo1UmxkKb9hI8/4volcinkKoOR
YwFVaJBdP/gf/07WKkxNekmhBND94jgfo0DPLtCs0873seM0T4DgM3C1t/3VBjD7SXq9r9UWPgIFt
4OK6rS17wOUOOMPxkyKbsSkGaQgQqEAvJ/RN+WAoVugb83rrbWP0wcwgQ12lboCDQ2CsSNnrPu/u7ezt
3z0t/n4AkJZysiZoae0AAA14aVRYdFhNTDpjb20uYWRvYmUuWG1wAAAAAAA8P3hwWWNrZXQgYmVn
aW49I+/7vyIgWQ9Ilc1TTBNcENNlaGlIenJlU3pNVG6a2M5ZCCI/Pgo8eDp4bXBtZXRhIHhtbG5zOn4z
Ong9ImFkYe2JlOm5zOm1lldGGEvIiB4OnhtptcHRrSJYTVAgQ29yZaSAQLjQUMClFeGCl2MiI+CiAb8cmRm
OlJERiB4bWxuczpyZGY9Imh0dHA6Ly93d3cudWzMub3JnLzE5OTkvMDIvMjItcmRmLXN5bnRheC1u

Figure 8: Encoding of Tux image into base64.

ASCII Smuggling

ASCII smuggling is a technique that leverages Unicode characters, which are invisible in user interfaces but can be interpreted by large language models (LLMs), to embed hidden instructions or data within text. This method allows attackers to manipulate AI responses or exfiltrate sensitive information without the user's awareness, by embedding these hidden Unicode tags within clickable hyperlinks or documents shared in chats.
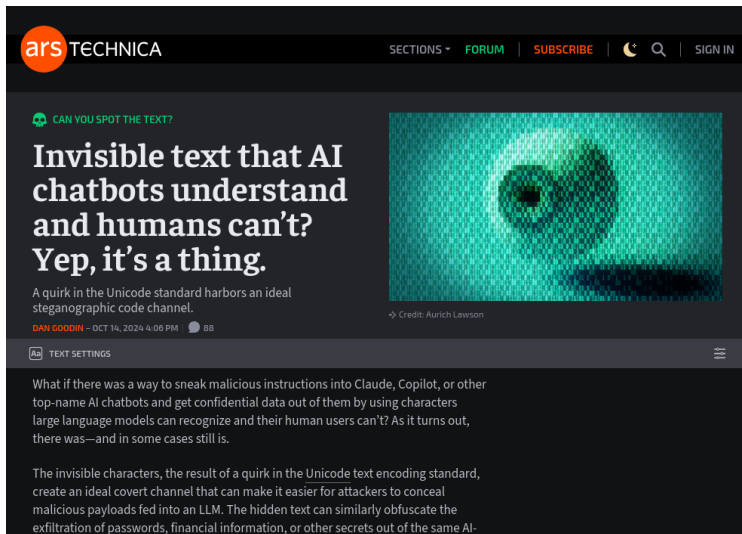
Figure 9: Ars Technica

ASCII Smuggler

Convert ASCII text to Unicode Tags which are invisible in most UI elements.
Check if a text has hidden Unicode Tags embedded with Decode.

Can you spot the text?
Invisible text that AI chatbots understand and humans can't? Yep, it's a thing.

A quirk in the Unicode standard harbors an ideal steganographic code channel.

Encode **Decode**

Advanced Options

Can you spot the ░Easter Egg░ text?
Invisible text that AI chatbots understand and humans can't? Yep, it's a thing.

A quirk in the Unicode standard harbors an ideal steganographic code channel.

Hidden Unicode Tags discovered.

Clear

Figure 10: https://embracethered.com/blog/ascii-smuggler.html

ISO/IEC Standards

- **ISO/IEC 8859:**
  - Series for 8-bit character encoding supporting multiple languages.
  - ISO-8859-1 (Western Europe), also known as ISO Latin 1.
    - The first 128 characters are identical to ASCII.
    - 0x00 to 1F and 0x80 to 0x9F (hex) used for C0 and C1 control codes.
    - C0 set was originally defined in ISO 646 (ASCII) (e.g., Start of Heading, Start of Text, End of Text, End of Transmission, . . .).
    - C1 are additional control codes (e.g., Padding Character, High Octet Preset, Break Permitted Here, No Break Here, . . .).
- **ISO/IEC 10646:**
  - Universal character set (UCS) for multilingual text.

Figure 11: ISO-8859-1 code page

Windows Code Pages

- **Overview:**
  - Multiple code pages for different regions and languages.
- **Examples:**
  - CP1252 (Western Europe), CP932 (Japan)
- **Issues:**
  - Inconsistencies across different systems.

Unicode

- **Unicode:**
  - Universal character set covering all scripts, supporting over 143,000 characters.
  - It assigns a unique number (called a "code point") to each character, regardless of platform, program, or language.
  - 1.112.064 valid code points within the codespace.
  - As of Unicode 16.0, released in September 2024, 299,056 (27%) of these code points are allocated, 155,063 (14%) have been assigned characters, 137,468 (12%) are reserved for private use, 2,048 are used to enable the mechanism of surrogates, and 66 are designated as noncharacters, leaving the remaining 815,056 (73%) unallocated.
  - Unicode has different encoding forms: UTF-8, UTF-16, and UTF-32.

UTF

- **UTF-8:**
  - Variable-length encoding, backward compatible with ASCII, byte-order independent.
- **UTF-16:**
  - Variable-length encoding (2 or 4 bytes per character).
  - Latin and most commonly used CJK[1] characters are encoded in 2 bytes.
- **UTF-32**:
  - Fixed-length encoding (4 bytes per character).

---

[1]Chinese, Japanese, and Korean.

Number of bytes	First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
1	U+0000	U+007F	0xxxxxxx					
2	U+0080	U+07FF	110xxxxx	10xxxxxx				
3	U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx			
4	U+10000	U+1FFFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx		
5	U+200000	U+3FFFFFF	111110xx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	
6	U+4000000	U+7FFFFFFF	1111110x	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx

Figure 12: UTF-8 Structure
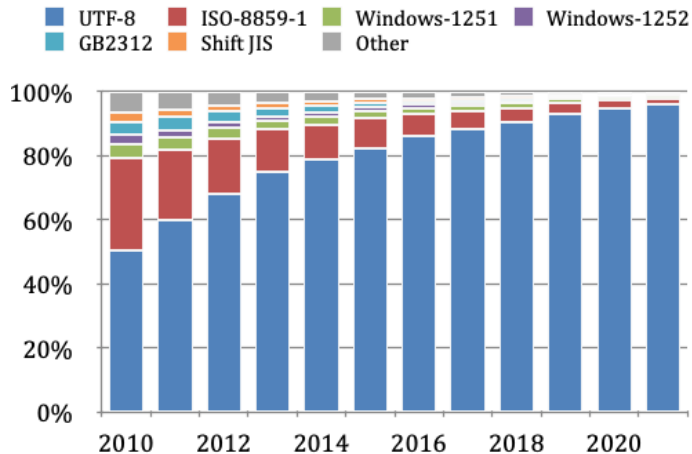
UTF-8 takes over

Figure 13: Declared character set for the 10 million most popular websites since 2010

Endianness

- **Big Endian vs. Little Endian:**
  - Byte order in memory representation.
  - Impacts how multi-byte characters are read.

Figure 14: Endianness

- **Example:**
    - UTF-16 and UTF-32 can be big or little endian.
    - Byte Order Mark (BOM) to indicate the endianness being used.
    - The BOM is the code point U+FEFF (BOM, ZWNBSP[2]).
        - Big-endian (UTF-16BE): FE FF
        - Little-endian (UTF-16LE): FF FE
        - Big-endian (UTF-32BE): 00 00 FE FF
        - Little-endian (UTF-32LE): FF FE 00 00
        - BOM in UTF-8: EF BB BF, serves more as a signature to indicate that the file is encoded in UTF-8 rather than specifying byte order.

---

[2] zero width no-break space

Text File Formats

- **.txt:** Usually ASCII or UTF-8.
- **.csv:** Can use various encodings; important for data exchange.
- **.json:** UTF-8 by default, supports Unicode.
- **.log**: Log files for recording events, errors, and system activities.

Markup Files

- **TeX:**
    - **Usage:** Typesetting language for high-quality typography.
    - **Encoding:** Often UTF-8, but can be sensitive to non-ASCII characters without proper preamble setup.
- **XML (eXtensible Markup Language):**
    - **Purpose:** Used for structured data storage and transmission.
    - **Encoding:** Declared in XML declaration, typically UTF-8 or UTF-16. Encoding declaration is crucial for correct parsing.
- **HTML (HyperText Markup Language):**
    - **Usage:** Standard markup language for documents designed to be displayed in a web browser.
    - **Encoding:** Default is often UTF-8, but can be specified with the `charset` attribute in the `<meta>` tag. Incorrect encoding can lead to garbled text.
- **Markdown:**
    - **Purpose:** Lightweight markup language for formatting text.
    - **Encoding:** Generally UTF-8, supports rich text with simple syntax.

Linux Tools for Text Encoding

- **iconv:** Converts text from one encoding to another.
    - Example: `iconv -f ISO-8859-1 -t UTF-8 input.txt > output.txt`
- **file:** Identifies file types and encodings.
    - Example: `file -i example.txt`
- **uconv:** More advanced conversion with Unicode support.
    - Example: `uconv -f UTF-8 -t UTF-16 input.txt -o output.txt`
- **dos2unix / unix2dos:** Converts between Windows and Unix line endings.
    - Example: `dos2unix file.txt` (converts CRLF to LF)
    - Example: `unix2dos file.txt` (converts LF to CRLF)
- **Use Cases:**
    - Data migration, cleaning, and internationalization.

Conclusion

- **Key Points:**
  - Text encoding has evolved from simple to complex systems.
  - Unicode provides a universal solution for global text representation.
- **Future:**
  - Continued development in encoding standards to support new scripts and symbols.