

WTF?

The Wobbly Transformation Format and the Windows File System

Nick Deguillaume: nick@riskpath.co.uk

<https://www.riskpath.co.uk/presentations/WTF-WindowsFileSystem.pdf>

25th January 2024

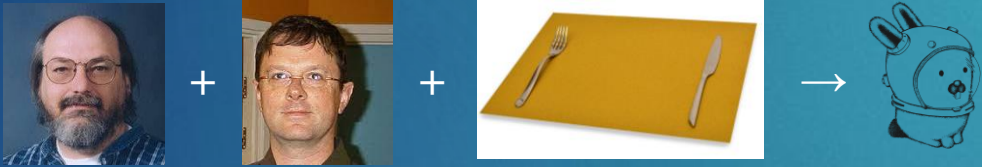

ACCU - Bristol

Character Encoding History



Historical

 JIS	 Shift_JIS	 EUC	 ASCII - 1963	 Extended ASCII	 GBK	 EUC-KR	 KPS 9566	 Scandinavian Encodings	 Morse Code
--	--	--	---	--	--	---	---	---	---

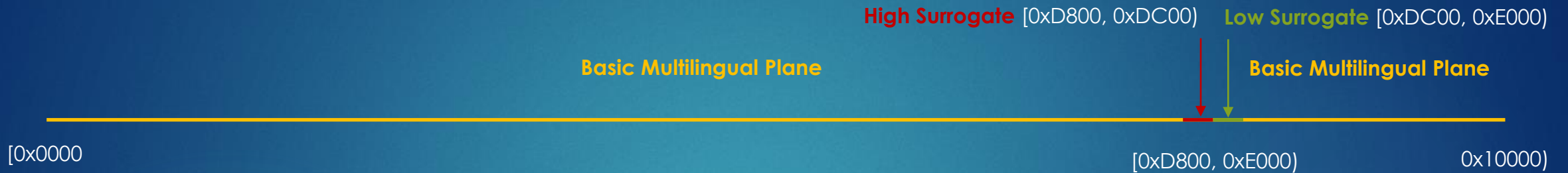
Unicode v1 1988 - 1996

<p>UTF-8 (1-6 bytes / codepoint. Encodes 2,147,483,648 codepoints)</p> 	<p>UTF-16 (2 bytes per codepoint. Encodes 65,536 codepoints)</p> 
---	--

Unicode v2+ 1996 onwards

<p>UTF-8 (1-4 bytes per codepoint. Encodes 1,112,064 codepoints)</p>  <p>UTF-8 won the battle of the internet with 98.1% of web pages encoded in UTF-8.</p>	<p>UTF-16 (2-4 bytes / codepoint. Encodes 1,112,064 codepoints)</p>  <p>Modern UTF-16 is not pretty. However, it will continue to exist due to the operating systems and vast amount of software employing it. However, even Microsoft now recommend to use UTF-8 where possible.</p>
--	--

UTF-16 and WTF-16



Legal UTF-16

- ▶ Any 16-bit value on the basic multilingual plane: $[0x0000, 0D800) \cup [0xE000, 0x10000)$
- ▶ Any 16-bit high surrogate: $[0xD800, 0xDC00)$ so long as it is followed by a low surrogate
- ▶ Any 16-bit low surrogate: $[0xDC00, 0xE000)$ so long as it is preceded by a high surrogate

WTF-16

- ▶ Any 16-bit value: $[0x0000, 0x10000)$. No restrictions apply

UTF/WTF-16 → UTF/WTF-8 Conversion

UTF-16 → UTF-32

- ▶ If a 16-bit value is on the basic multilingual plane, then the natural map applies. For example:

0xFFFF → 0x0000FFFF

- ▶ Otherwise, we take two 16-bit values: **H** (guaranteed to be a high surrogate) and **L** (guaranteed to be a low surrogate) and apply the formula:

$$(H - 0xD800) * 0x400 + (L - 0xDC00) + 0x10000$$

First High Surrogate

Scale Normalised High Surrogate by 1,024 = 2¹⁰

1st Low Surrogate

Add on 65,536 to bring you past the Basic Multilingual Plane

WTF-16 → WTF-32

- ▶ If a 16-bit value is on the basic multilingual plane or if together with the next 16-bit value, it forms a high surrogate / low surrogate pair then we perform the same operation as UTF-16 → UTF-32
- ▶ Otherwise, the natural map applies. For example:

0xDC00 → 0x0000DC00

UTF/WTF-32 → UTF/WTF-8

- ▶ View the 32-bit representation as binary. For example, we view the first low surrogate **0x0000DC00** as:

0b00000000 00000000 11011100 00000000 ← UTF-32

- ▶ Count the minimum number of bits required to represent the value. In this case 16 bits are required.
- ▶ Look up the number of UTF/WTF-8 bytes required in the following table:

# UTF/WTF-32 bits	# UTF/WTF-8 bytes	Byte 0	Byte 1	Byte 2	Byte 3
0 - 7	1	0xxxxxxx			
8 - 11	2	110xxxxx	10xxxxxx		
12 - 16	3	1110xxxx	10xxxxxx	10xxxxxx	
17 - 21	4	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

- ▶ We have 16 bits so three UTF/WTF8 bytes are required.
- ▶ Fill in the 'x' bits of the pattern provided in the table above starting from the right. This gives:

0b11011100 00000000 → 0b11101101 0b10110000 0b10000000

- ▶ Convert to hex if desired:

0b11101101 0b10110000 0b10000000 → 0xED 0xB0 0x80

WTF-8 String Representation

What do we want?

- ▶ Any codepoint that is UTF-8 representable to be represented in the same way as it is in UTF-8
- ▶ The lone surrogates should use the same calculation as is used to get from UTF-32 to UTF-8

When we convert a WTF-8 *string* to WTF-16 and then back again, we want to recover the same representation that we started with. Our conversion function is already set, so we must limit the space of WTF-8 strings in order for the bijection to apply.

To do this we must avoid:

- **Overlong Encodings**
- **Accidental Surrogate Pairs**

WTF/UTF-8 Overlong Encodings

Consider the codepoint 'A' with Unicode representation (U+0041).

The only correct UTF/WTF-8 representation of this is `0x41` or `0b01000001`.

However, a hacker could represent the same value using an overlong sequence:

One byte (correct): `0b01000001`

Two bytes (incorrect): `0b11000001 0b10000001`

Three bytes (incorrect): `0b11100000 0b10000001 0b10000001`

Four bytes (incorrect): `0b11110000 0b10000000 0b10000001 0b10000001`

The only permissible UTF/WTF-8 encoding is the shortest one.

WTF-8 - Accidental Surrogate Pair

7

- ▶ Consider the supposed WTF-8 hex sequence:

`ED A0 80 ED B0 80`

- ▶ `ED A0 80` and `ED B0 80` individually represent valid WTF-8 codepoints:

- `ED A0 80` represents the first high surrogate: `U+D800`
- `ED B0 80` represents the first low surrogate: `U+DC00`

Note that these are not individually valid UTF-8 codepoints.

- ▶ So in UTF/WTF-16, the sequence is:

`0xD800 0xDC00`

- ▶ This is actually the Unicode value:

$(0xD800 - 0xD800) * 0x400 + (0xDC00 - 0xDC00) + 0x10000 = U+10000$

- ▶ In binary `U+10000` is:

`0b00000001 00000000 00000000`

- ▶ Converting to UTF/WTF-8 gives:

`0b11110000 0b10010000 0b10000000 0b10000000`

- ▶ In hexadecimal this is: `F0 90 80 80`

- ▶ This is not the same sequence that we started with!

To avoid accidental surrogate pairs in WTF-8, we disallow a high surrogate sequence directly followed by a low surrogate sequence.

This fact needs to be considered in WTF-8 string concatenation.

Accidental surrogate pairs are impossible in UTF-8 because lone surrogates are illegal.

UTF-8 – UTF-16 Comparison

Feature	UTF-8	UTF-16
Encodes all Unicode points	✓	✓
Self-synchronising	✓	ish...
Better error detection	✓	X
Usually more compact	✓	X
More compact for some Asian languages	X	✓
Endianness agnostic	✓	X
Never contains eight zero bits in a row when a null is not present	✓	X
Has same ordering as Unicode under naïve byte comparison	✓	X
Identical ASCII representation	✓	X
If you are a Windows developer, using it will make Linux developers look down on you a little bit less	✓	X

CreateFileW Win API function

The CreateFileW function has many options. Some combinations are sensible. Some are not. The table below shows some sensible combinations.

Mode	Desired Access	Share Mode	Creation Disposition**	Flags and Attributes
Open existing file in read-only mode	GENERIC_READ	FILE_SHARE_READ	OPEN_EXISTING	FILE_ATTRIBUTE_NORMAL
Open in read/write mode with no truncation	GENERIC_READ GENERIC_WRITE DELETE	-	OPEN_EXISTING	FILE_ATTRIBUTE_NORMAL
Create new file that is deleted on handle close *	GENERIC_READ GENERIC_WRITE DELETE	-	CREATE_NEW	FILE_ATTRIBUTE_NORMAL
Open file or directory for information purposes	GENERIC_READ	FILE_SHARE_READ FILE_SHARE_WRITE	OPEN_EXISTING	FILE_FLAG_BACKUP_SEMANTICS

* If the third option is chosen, the file disposition should be set to delete using the Win API function: **SetFileInformationByHandle** with the **FileInformationClass** parameter set to **FileDispositionInfo**.

** Avoid OPEN_ALWAYS and CREATE_ALWAYS. They are not necessary and have been linked to vulnerabilities.

Handle-Based Win API Functions

10

Basic Input / Output Functions (Useful in creating streams)

Function	Used For
GetFileSizeEx	File size.
ReadFile	Read from file or other stream.
WriteFile	Write to file or other status.
SetFilePointerEx	Set file pointer / determine file pointer.
FlushFileBuffers	Flush file buffers.

Get / Set File Attributes

Function	Used For
SetFileTime	Set creation time, last access time, last write time.
SetFileInformationByHandle	Mark read-only, set delete disposition, reserve, resize, rename.
GetFileInformationByHandleEx	Determine read-only status, last write time, capacity. Determine location i.e. volume and file-id.
GetFinalPathNameByHandleW	Gives actual case of file path. Useful when interfacing with Linux where file paths are case sensitive.

More Win API Functions

11

Path-Based Win API Functions

Function	Used For
SHGetKnownFolderPath	Get known folder such as local app data.
ExpandEnvironmentStringsW	Substitute environment variables such as %LOCALAPPDATA%.
GetLongPathNameW	Gets long path from short path.
GetShortPathNameW	Gets short path.
WNetGetUniversalNameW	Universal name for a file on a network.
MoveFileExW	Useful in renaming a directory.
GetFileAttributesW	Useful in finding whether a file or directory is read-only.

Cryptographic Win API Functions

Function	Used For
CryptProtectData	Protect data so only the user who called this function can read it.
CryptUnprotectData	Decrypt data encrypted via CryptProtectData.
CryptProtectMemory	Protects memory in place. Useful for in-process encryption of passwords and keys.
CryptUnprotectMemory	Unprotect memory in place.

Plenty more at... <https://learn.microsoft.com/en-us/windows/win32/api/fileapi/>

Tips and Links

- ▶ Consider using the WIL library. It provides a modern C++ wrapper over the windows API functions.
- ▶ Consider using Natvis files to make debugging your custom types easier.
- ▶ When using third party tools consider Google. Their tools seem to be well tested on MSVC and Visual Studio. Personally, I use:
 - Abseil - Efficient containers (C++)
 - Google Test - Testing (C++)
 - Gumbo - Html parsing (C)
- ▶ <https://simonsapin.github.io/wtf-8/>
- ▶ <https://docs.microsoft.com/en-us/windows/win32/fileio/naming-a-file>
- ▶ <https://learn.microsoft.com/en-us/troubleshoot/windows-client/shell-experience/file-folder-name-whitespace-characters>
- ▶ <https://learn.microsoft.com/en-us/windows/win32/api/fileapi/>