

TCW_#_α-0»:¥TRUNK¥MULTI-IO¥TEXT¥BOM-1.rtf
G+2013-05-03-5.zz.h03:33:

Terry Carl Walker
1739 East Palm Lane
Phoenix, AZ 85006-1930
United States
1-480-929-9628
waxymat@aztecfreenet.org

BOM-1 Proposal

BOM-1 Proposal

Page 1 of 3

BOM-1 Proposal

There is a problem with the current Byte Order Mark or BOM. To illustrate this, the following table shows the byte values of the BOM in hexadecimal in every format.

BOM-0 = @.0`feff		
UTF-7	@.2f`7b`3f	/{?
UTF-8	@.ef`bb`bf	
UTF-16-BE	@.fe`ff	
UTF-16-LE	@.ff`fe	
UTF-32-BE	@.00`00`fe`ff	
UTF-32-LE	@.ff`fe`00`00	

The UTF-16-LE BOM, when followed by the ASCII character null (@.00), becomes identical to the byte pattern for the UTF-32-LE BOM. If such an occurrence happens in UTF-16-LE, the file will be incorrectly interpreted as UTF-32-LE. The proposal is to have an optional replacement BOM available on a code point other than on the Base Multilingual Plane (BMP, plane zero).

As pointed out in the Unicode standards document, the sub-plane value of @.feff, when the bytes are swapped, produces a value in the sub-plane that is an invalid code point because it is one of the two highest possible sub-plane values -- which is prohibited. The two different byte values mean that the order of the bytes can be determined. Together, this makes it possible to detect when the bytes are swapped. The result, when combined with the different word length formats and the extreme unlikeliness of having this particular value point as the first text code point, makes this a good choice for a BOM in plane zero. Unfortunately, the current BOM refers to a code point that is actually valid as a text value. If the replacement BOM has the same sub-plane value as the current BOM but in a different plane and with a code point that is declared as invalid, then there should be no confusion when it is encountered that it is a BOM and that it clearly identifies the exact format of the text. Because it is a format-only code point, this alternate BOM should only occur once (if at all) in any file -- as the first code point of the file. If it is present, it should not be interpreted as part of the text.

The original BOM in plane zero will hereby be called BOM-0 -- in order to distinguish it from the alternate BOM. The sub-plane value of the BOM is valid as a text code point in planes zero, fifteen, and sixteen. The last two planes are entirely reserved for user-defined code points and are therefore not available for use as possible BOMs. This leaves planes one through fourteen. If we use the lowest available plane (plane one) so as to reserve the higher planes for future assigned code points, then the alternate BOM can be referred to as BOM-1. The resulting values for the alternate BOM are therefore as follows:

BOM-1 = @.1`feff			
UTF-7	@.3c`4f`7b`3f	<O{?	
UTF-8	@.f0`9f`bb`bf		
UTF-16-BE	@.d8`3f`de`ff	?	
UTF-16-LE	@.3f`d8`ff`de	?	
UTF-32-BE	@.00`01`fe`ff		
UTF-32-LE	@.ff`fe`01`00		

With the BOM-1 values taking up four bytes in all formats and the values being different for both BOMs in every coding scheme (except for the UTF-16-LE BOM-0 possibly matching either UTF-32-LE BOM), it is recommended that new files start with a BOM-1. Files may therefore start with a BOM-0, a BOM-1, or neither. Newer programs will recognize a BOM-0 or a BOM-1 as a formatting code point and not as a text code point. Older programs will, unfortunately, see the BOM-1 as one or two unrecognized code points (depending on the format and the age of the program) and may display them as error values. As usual, no BOM should be in the middle of the file -- a rule that should cause any program that concatenates files to remove the BOMs from the beginning of all files (except the first) in the resultant combined file. Also, a file that is split should only duplicate the BOM (if any) from the original file into each new file.