

<b>Title:</b>	<b>Proposed Texts for Draft UTR-16 and an Informative Annex on UTF-EBCDIC to ISO/IEC 10646-1: 1983</b>
<b>Source:</b>	<b>V.S. UMaaheswaran, IBM National Language Technical Centre, Toronto</b>
<b>Status:</b>	<b>For consideration and approval and for forwarding to ISO/IEC JTC 1/SC 2/WG 2 by NCITS/L2 and UTC members</b>

UTF-EBCDIC was presented at UTC 77 and subsequently forwarded to SC 2/WG 2 for information and feedback as document NCITS/L2 98-257 Rev (SC2/WG2/N1848) to WG 2 meeting 35 in London, UK..

The information from that document has been prepared as Unicode Technical Report No. 16 for consideration by UTC. This document is attached for consideration by the UTC and progressing towards a Technical Report by the UTC.

It was stated at the London WG 2 meeting that the intent is to present the material to be processed as an informative annex to ISO/IEC 10646. Attached is the proposed text for an informative Annex to be processed as an Amendment to that standard. The material is written following the current text of UTF-8 (Amendment No. 2) very closely. (The attached document was prepared by Messrs. Alexis Cheng using a soft copy of Amendment No. 2 as the template and revised by V.S. Umamaheswaran of IBM NLTC.)

If accepted by UTC and NCITS/L2, it is requested that NCITS/L2 submit this document (revised as needed) to WG 2 for consideration at its Tokyo meeting in March 1999 towards processing it as an amendment to the standard.

Attachment 1: Draft UTR-1 - UTF-EBCDIC

Attachment 2: Draft text for new Annex to 10646-1 - UTF-EBCDIC

**DRAFT Unicode Technical Report #16**  
**UTF-EBCDIC**  
**EBCID-Friendly UCS Transformation Format**

<b>Revision</b>	2
<b>Author(s)</b>	V.S. UMAmaheswaran ( <a href="mailto:umavs@ca.ibm.com">umavs@ca.ibm.com</a> )
<b>Date</b>	January 22, 1999
<b>This Version</b>	<a href="http://www.unicode.org/unicode/reports/tr16-2/">http://www.unicode.org/unicode/reports/tr16-2/</a>
<b>Previous Version</b>	<a href="http://www.unicode.org/unicode/reports/tr16/">http://www.unicode.org/unicode/reports/tr16/</a>
<b>Latest Version</b>	<a href="http://www.unicode.org/unicode/reports/tr16-2/">http://www.unicode.org/unicode/reports/tr16-2/</a>

This report presents the specifications of *UTF-EBCDIC - EBCID-Friendly UCS Transformation Format*.

**Status of this document**

*This draft is published for public review. Previous version of this document has been considered by the Unicode Technical Committee, and it has had preliminary approval as a Draft Unicode Technical Report. The Unicode Technical Committee may approve, reject, or further amend this document before it becomes an approved Unicode Technical Report. This document does not, at this time, imply any endorsement by the Consortium's staff of member organizations. Please mail comments to [unicode@unicode.org](mailto:unicode@unicode.org).*

---

**Scope**

The term UTF-EBCDIC stands for EBCDIC-friendly UCS Transformation Format. EBCDIC, IBM's Extended Binary Coded Decimal Interchange Code, is one of the widely used 8-bit industry encodings. Detailed information on EBCDIC can be found in the IBM publication *IBM Character Data Representation Architecture, Reference and Registry*, SC09-2190-00, December 1996.

To address the use of Unicode character data in byte-oriented ASCII-based systems, the Unicode Standard (see *section A.2 of the Unicode Standard*) (also *ISO/IEC 10646 –1, Amendment no. 2*) has defined UTF-8. Use of UTF-8 permits existing ASCII-based systems that have hard-coded dependency on the encoding of the ASCII repertoire of characters to safely process the corresponding Unicode characters. There is a similar requirement to transform Unicode characters to a form that is safe for EBCDIC systems for the control characters and invariant characters.

This Technical Report defines the EBCDIC-friendly UCS transformation format, UTF-EBCDIC.

UTF-EBCDIC is not intended to be used in open interchange environments. Its usefulness is restricted to EBCDIC-constrained systems and networks only.

## Description

The UTF-EBCDIC encoding is derived from the Unicode values following a two step process:

- Step 1:** Conversion of the Unicode values to a variable length byte sequence called *l8-sequence* (intermediate 8-bit sequence) in this technical report by applying a modified UTF-8 transformation – UTF-8M, enabling the preservation of 64 control characters as single bytes.
- Step 2:** The bytes in the l8-sequence are then converted to the UTF-EBCDIC byte sequence by using a single-byte to single-byte reversible conversion.

These two steps are defined below.

## Definition

### Step 1: UTF-8M

The UTF-8M transformation definition is modeled after UTF-8 definition in the Unicode standard. UTF-8M transforms the Unicode values into l8-sequences. The Unicode characters U+0000 to U+001F (corresponding to the C0 control characters x00 to x1F of ASCII), U+0020 to U+007E (the ASCII repertoire), and U+007F (the ASCII 'DEL' control character) are represented as single bytes in the l8-sequence. In addition, U+0080 to U+009F (corresponding to the so-called C1 set of controls in ISO/IEC 6429) are also represented as single bytes (x80 to x9F). Thus the 65 characters corresponding to the 65 ISO/IEC 6429 control characters and the 95 ASCII graphic characters (the G0 set) are represented in the l8-sequence as single bytes. When these are converted to the UTF-EBCDIC form, the corresponding 65 EBCDIC control characters and 95 graphic characters are preserved as single bytes in the UTF-EBCDIC byte sequence.

Furthermore, the values x00...x9F (0...159) do not appear in any byte of an l8-sequence except as the direct representation of these C0, G0 and C1 values. Each Unicode value (non-surrogates) is represented in an l8-sequence by 1, 2, 3 or 4 bytes, depending on the Unicode value. Pairs of surrogates take either 4 bytes or 5 bytes, depending on the Unicode scalar values represented by the surrogate pairs.

The l8-sequence is a variable length encoding of Unicode characters using 8-bit sequences, where the high bits indicate which part of the sequence a byte belongs to. Table 1 shows how the bits in a Unicode value (or surrogate pair) are distributed among the bytes in the l8-sequence. The corresponding ranges of Unicode scalar values are also shown.

**Table 1 l8-Sequence Bit Distribution**

Scalar Value (hex)	Unicode Value	1st Byte	2nd Byte	3rd Byte	4th Byte	5th Byte
0 to 7F	00000000xxxxxx	0xxxxxxx				
80 to 9F	0000000100xxxx	100xxxxx				
A0 to 3FF	00000yyyxxxx	110yyyyy	101xxxxx			
400 to 3FFF	00zzzzzyyyxxxx	1110zzzz	101yyyyy	101xxxxx		
4000 to FFFF	wzzzzzyyyxxxx	1111000w	101zzzzz	101yyyyy	101xxxxx	
10000 to 3FFFF	110110uuuwzzzz +110111yyyyxxxx	11110ppw <sup>a</sup>	101zzzzz	101yyyyy	101xxxxx	
	x					
40000 to 10FFFF	110110uuuwzzzz +110111yyyyxxxx	1111100q <sup>b</sup>	101ppppw <sup>b</sup>	101zzzzz	101yyyyy	101xxxxx
	x					
where <sup>a</sup>	uuuu = 000pp -1, or					
<sup>b</sup>	uuuu = qpppp -1					
(to account for addition of 10000 <sub>16</sub> as in Section 3.7, <i>Surrogates</i> , in the Unicode Standard)						

When converting Unicode values to l8-sequences, always use the shortest form that can represent these

values. This preserves uniqueness of encoding. For example the Unicode value <0000000000000001> is encoded as <00000001>, not as <11000000 10100001>. The latter is an example of an unused I8-sequence. *Do not make use of these unused byte sequences for encoding any other information.*

When converting from I8-sequences to Unicode values, however, implementations do not need to check that the shortest encoding is being used, which simplifies the conversion algorithm.

## Characteristics of I8-sequence

Some of the important characteristics of I8-sequence are

- Unicode characters from U+0000 to U+007E (ASCII repertoire) map to single-byte I8-sequence values 00 to 7E (ASCII values 0 ... 127).
- Unicode characters from U+0080 to U+009F (C1 controls) map to single-byte I8-sequence values 80 to 9F (ISO-8 values 128 ... 159).
- ASCII values or ISO-8 values do *not* otherwise occur in an I8-sequence. This paves the way for transforming these into corresponding single-byte EBCDIC controls and graphics in the second step of UTF-EBCDIC transform.
- It is very simple and efficient to convert to and from Unicode text.
- The first byte indicates the number of bytes to follow in a multi-byte sequence. This allows for efficient forward parsing.
- It is efficient to find the start of a character string from an arbitrary location in a byte stream. You need to search at most five bytes (seven if full range of 31 bits of ISO/IEC 10646 is to be considered) backwards, and it is simple to recognize an initial byte. For example, after converting an UTF-EBCDIC byte back into I8-sequence, in C  
    isInitialByte = ( (byte & 0xE0) != 0xA0);  
The search can also be done directly on UTF-EBCDIC byte by utilizing a shadow vector (see Table 3 described later)
- The I8-sequence is reasonably compact in terms of number of bytes used for encoding.

## Step 2: Byte Conversion

The second step of UTF-EBCDIC transforms the I8-sequences using a reversible, unique one-to-one mapping into the byte sequences of UTF-EBCDIC. The 64 control characters (U+0000 to U+003F, U+0080 to U+009F), the ASCII 'DEL' character (U+007F), the 95 ASCII graphic characters (including the SPACE character) (U+0020 to U+007E) are mapped respecting EBCDIC conventions. The map preserves the invariance for a set of 81 graphic characters (known as the IBM Syntactic Graphic Character set), and maintains consistency with IBM MVS Open Systems Code page (CPGID 1047) for the variant characters from within the ASCII repertoire. The remaining 128 bytes of EBCDIC 8-bit structure are allocated to the leading bytes and trailing bytes of UTF-8M (from Table 1). Table 2 shows the maps between the I8-sequence bytes and UTF-EBCDIC bytes (both directions are shown).

The resulting UTF-EBCDIC bytes can be transparently processed in most EBCDIC systems. It also retains all the characteristics (see Characteristics of I8-sequence above) of I8-sequence mentioned earlier. Since EBCDIC encodings has only 81 variants and 15 invariants the choice of the above byte map has been made to accommodate the Open Systems environment for standardization purposes.





## Shadow flags

In order to assist in finding out if a byte in a UTF-EBCDIC sequence is a leading byte or a trailing byte, and how many bytes in the sequence corresponding to a Unicode character, instead of looking at the byte's bit combination (after converting into its corresponding I8-sequence), a shadow table – shown in Table 3 – containing the category of the byte can be utilized. The bytes having a value of '0' in the category table are control characters, '1' are single bytes, '9' are trailing bytes and '2'... '7' indicate the number of bytes in the sequence. Even though Table 1 shows I8-sequences of only up to 5 bytes (to

transform up to plane 16), the UTF-8M can contain up to 7 bytes to address all of the UCS-4 space (31-bits) in ISO/IEC 10646 standard (see Table 4).

**Table 2 UTF-8M string to / from UTF-EBCDIC string byte map**

From I8-sequence to UTF-EBCDIC																
↓ High nibble	Low nibble ⇒															
	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
0-	00	01	02	03	37	2D	2E	2F	16	05	25	0B	0C	0D	0E	0F
1-	10	11	12	13	3C	3D	32	26	18	19	3F	27	1C	1D	1E	1F
2-	40	5A	7F	7B	5B	6C	50	7D	4D	5D	5C	4E	6B	60	4B	61
3-	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	7A	5E	4C	7E	6E	6F
4-	7C	C1	C2	C3	C4	C5	C6	C7	C8	C9	D1	D2	D3	D4	D5	D6
5-	D7	D8	D9	E2	E3	E4	E5	E6	E7	E8	E9	AD	E0	BD	5F	6D
6-	79	81	82	83	84	85	86	87	88	89	91	92	93	94	95	96
7-	97	98	99	A2	A3	A4	A5	A6	A7	A8	A9	C0	4F	D0	A1	07
8-	20	21	22	23	24	15	06	17	28	29	2A	2B	2C	09	0A	1B
9-	30	31	1A	33	34	35	36	08	38	39	3A	3B	04	14	3E	FF
A-	80	8C	8D	8E	8F	90	9C	9D	9E	9F	A0	AC	AE	AF	BC	BE
B-	BF	CC	CD	CE	CF	DC	DD	DE	DF	EC	ED	EE	EF	FC	FD	FE
C-	42	43	44	45	46	47	48	49	52	53	54	55	56	57	58	59
D-	62	63	64	65	66	67	68	69	71	72	73	74	75	76	77	78
E-	8A	9A	AA	BA	CA	DA	EA	FA	8B	9B	AB	BB	CB	DB	EB	FB
F-	B2	B3	B4	B5	B6	B7	B8	B9	6A	70	B0	B1	41	51	4A	E1

	EBCDIC control character positions		Positions of EBCDIC-invariant character subset of the repertoire of IRV of ISO 646
	Positions of EBCDIC-variant character subset of the repertoire of IRV of ISO 646		Positions of characters outside the repertoire of IRV of ISO 646

From UTF-EBCDIC to I8-sequence																
↓ High nibble	Low nibble ⇒															
	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
0-	00	01	02	03	9C	09	86	7F	97	8D	8E	0B	0C	0D	0E	0F
1-	10	11	12	13	9D	85	08	87	18	19	92	8F	1C	1D	1E	1F
2-	80	81	82	83	84	0A	17	1B	88	89	8A	8B	8C	05	06	07
3-	90	91	16	93	94	95	96	04	98	99	9A	9B	14	15	9E	1A
4-	20	FC	C0	C1	C2	C3	C4	C5	C6	C7	FE	2E	3C	28	2B	7C
5-	26	FD	C8	C9	CA	CB	CC	CD	CE	CF	21	24	2A	29	3B	5E
6-	2D	2F	D0	D1	D2	D3	D4	D5	D6	D7	F8	2C	25	5F	3E	3F
7-	F9	D8	D9	DA	DB	DC	DD	DE	DF	60	3A	23	40	27	3D	22
8-	A0	61	62	63	64	65	66	67	68	69	E0	E8	A1	A2	A3	A4
9-	A5	6A	6B	6C	6D	6E	6F	70	71	72	E1	E9	A6	A7	A8	A9
A-	AA	7E	73	74	75	76	77	78	79	7A	E2	EA	AB	5B	AC	AD
B-	FA	FB	F0	F1	F2	F3	F4	F5	F6	F7	E3	EB	AE	5D	AF	B0
C-	7B	41	42	43	44	45	46	47	48	49	E4	EC	B1	B2	B3	B4
D-	7D	4A	4B	4C	4D	4E	4F	50	51	52	E5	ED	B5	B6	B7	B8
E-	5C	FF	53	54	55	56	57	58	59	5A	E6	EE	B9	BA	BB	BC
F-	30	31	32	33	34	35	36	37	38	39	E7	EF	BD	BE	BF	9F

**Table 3 Shadow flags associated with UTF-EBCDIC bytes**

<b>LEGEND:</b>	
<b>0</b> = Single-octet control characters	<b>1</b> = Single-octet invariant and variant graphic characters from IRV of ISO 646
<b>2</b> = Lead octet of a 2-octet string	<b>3</b> = Lead octet of a 3-octet string
<b>4</b> = Lead octet of a 4-octet string	<b>5</b> = Lead octet of a 5-octet string
<b>6</b> = Lead octet of a 6-octet string	<b>7</b> = Lead octet of a 7-octet string
<b>9</b> = A trailing octet of a multi-octet string	

	High nibble ↓				Low nibble ⇒											
	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
0-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4-	1	6	2	2	2	2	2	2	2	2	7	1	1	1	1	1
5-	1	6	2	2	2	2	2	2	2	2	1	1	1	1	1	1
6-	1	1	2	2	2	2	2	2	2	2	5	1	1	1	1	1
7-	5	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1
8-	9	1	1	1	1	1	1	1	1	1	3	3	9	9	9	9
9-	9	1	1	1	1	1	1	1	1	1	3	3	9	9	9	9
A-	9	1	1	1	1	1	1	1	1	1	3	3	9	1	9	9
B-	5	5	4	4	4	4	4	4	4	4	3	3	9	1	9	9
C-	1	1	1	1	1	1	1	1	1	1	3	3	9	9	9	9
D-	1	1	1	1	1	1	1	1	1	1	3	3	9	9	9	9
E-	1	7	1	1	1	1	1	1	1	1	3	3	9	9	9	9
F-	1	1	1	1	1	1	1	1	1	1	3	3	9	9	9	0

<b>0</b>	EBCDIC control character positions	<b>1</b>	EBCDIC-invariant subset of ASCII repertoire
<b>1</b>	EBCDIC-variant subset of ASCII repertoire	<b>2--9</b>	characters outside ASCII repertoire

## Special nature of UCS values FEFF, FFFE and FFFF

U+FFFE and U+FFFF are not used for character allocation in any plane of Unicode. U+FEFF (ZERO WIDTH NO-BREAK SPACE) is used as a signature for Unicode -- UCS-2 and UTF-16. U+FFFE may strongly suggest a byte-reversed Unicode string. U+FFFF is used to represent a numeric value that is guaranteed not to be a character, for uses such as the final value at the end of an index. UTF-8 also avoids the use of X'FF' and X'FE' as octets in its sequences. In I8-sequence, however, X'FE' and X'FF' may appear. The following paragraphs expand on which combinations of X'FF' and X'FE' may occur in an I8-sequence or UTF-EBCDIC sequence.

- X'FE' X'FF' and X'FF' X'FF' in the UTF-8M sequence:***

The X'FE' and X'FF' are lead octets of seven-byte I8-sequence (assuming values from all the planes of UCS-4). They will be surrounded (in a properly formed UTF-8M transformed string) by a value less than X'C0'. None of X'FF' X'FF', X'FE' X'FF', and X'FF' X'FE' sequences should appear in a properly formed I8-sequence. The I8-sequence to UTF-EBCDIC byte mappings are: X'FE' to X'4A', and X'FF' to X'E1'.

- X'FE' X'FF' and X'FF' X'FF' in the UTF-EBCDIC sequence:***

The values X'FE' and X'FF' are generated in a UTF-EBCDIC byte sequence by converting an I8-sequence using X'BF' to X'FE' and X'9F' to X'FF' (from Table 2).

X'BF' is the last element of the set of trailing bytes possible in an I8-sequence and must be preceded by a lead byte and zero or more trailing bytes (all within the range X'A0' to X'FF'). An X'9F' cannot precede it in a properly formed I8-sequence, and hence the sequence X'FE' X'FF' should not appear in a UTF-EBCDIC. However, it is conceivable to have an X'BF' (a trailing byte) followed by X'9F' (APC in ISO/IEC 6429) followed by some APC parameter string bytes – and therefore a sequence of X'FE' X'FF' may appear in a UTF-EBCDIC sequence.

The X'9F' is assigned to the control character -- Application Program Command (APC) -- in ISO-8 C1. According to ISO/IEC 6429, APC is followed by a parameter string using bit combinations from 0/8 to 0/13 (X'08' to X'0D') and 2/0 to 7/14 (X'20' to X'7E') and terminated by the control function String Terminator (ST) (coded at X'9C' in C1). So the sequence X'FF' X'FF' (equivalent of two APC controls without intervening parameters or ST-s) also should not appear in an E-string.

## Signature

The signature character U+FEFF (*zero width no-break space*) of Unicode transforms into the byte sequence X'B3 FE DE FE' in UTF-EBCDIC. The byte-swapped signature (suggesting a "little-endian" Unicode string) U+FFFE transforms into the UTF-EBCDIC sequence X'B3 FE FE FD'. When this sequence is displayed (erroneously) using a single-byte EBCDIC code page, it can be visualized as different character strings. In Latin-1 EBCDIC code pages 1047 (and coincidentally also in Latin-1 code pages 500 and 37), they appear as "•ÙùÙ" (middle dot, U-acute, u-acute, U-acute) and "•ÙÙÙ" (middle dot, U-acute, U-acute, U-grave) respectively. It can appear differently with other single-byte EBCDIC code pages.

## Normalization

Dealing with a variable number of bytes may not be possible or desirable in some processing situations (even though proper handling of Unicode text strings will require the ability to correctly deal with combining sequences). Normalization into a form with a fixed number of bits is needed for such cases. It would be always desirable to revert to the original 16-bit form or the corresponding 32-bit form as a normalization to fixed-width data. However, this would be possible only if processing is tolerant to native Unicode encoding. If transparency to EBCDIC invariance and controls is needed also in the normalized form, then Unicode cannot be directly used for normalization. It can be seen from Table 1 that the last code position in the BMP – U+FFFF – of Unicode requires a four-bytes in the I8-sequence and in the corresponding UTF-EBCDIC sequence. A 32-bit integer can be used for normalization of up to four-byte UTF-EBCDIC sequences.

The maximum Unicode scalar value that a four-byte I8-sequence or UTF-EBCDIC sequence can represent is:

B'11110111 10111111 10111111 10111111' (X'3FFFF')

corresponding to end of plane 3 in group 0. Using UTF-16 to represent planes 1 to 16, the surrogate characters in the BMP can be used. By treating the surrogate characters as any other BMP characters, up to plane 16 can be encoded using the 16-bit form, and hence can be contained within the 32-bit normalized form of UTF-EBCDIC. Care has to be taken to correctly process the corresponding UTF-EBCDIC sequence corresponding to the surrogate pairs, similar to dealing with combination sequences. When it is desirable to convert valid surrogate pairs into corresponding Unicode scalar value and then apply UTF-EBCDIC, only up to plane 3 can be contained within the 32-bit normalized value. For all values beyond group 0, plane 3 of UCS, the UTF-EBCDIC will contain more than four octets. The normalization for these cases will need 64 bits (assuming nothing between 32 and 64 bits is practical).

## Where to use UTF-EBCDIC

UTF-EBCDIC is intended to be used inside EBCDIC systems or in closed networks where there is a dependency on EBCDIC hard-coding assumptions. It is not meant to be used for open interchange among heterogeneous platforms using different data encodings. Due to specific requirements for ASCII encoding for line endings in some Internet protocols, UTF-EBCDIC is unsuitable for use over the Internet using such protocols. UTF-8 or UTF-16 forms should be used in open interchange.

### A comparison of UTF-EBCDIC and UTF-8

UTF-EBCDIC is a byte-mapped version of I8-sequence. The bit patterns of UTF-EBCDIC bytes and UTF-8 therefore are different. A comparison of the bit patterns of UTF-EBCDIC is not so meaningful. However, the I8-sequence and UTF-8 sequence can be compared to understand the salient differences between the two. UTF-8M being derived from UTF-8 retains all of its salient features. A comparative summary of the basic characteristics of I8-sequence and UTF-8 sequence is shown in the Table 4. Note that Table 4 shows the entire range 31 bits in the transformation, whereas Table 1 includes only the BMP and up to plane 16 using surrogate pairs.

**Table 4 Comparison of I8-Sequence with UTF-8 Generated Byte Sequence**

	I8-sequence	UTF-8-sequence	Remarks
No. of bytes in transformed sequence	Scalar Values (hex)	Scalar Values (hex)	
1	00 to 9F	00 to 7F	C0, G0 and C1 in I8-sequence C0 and G0 in UTF-8
2	A0 to 3FF	80 to 7FF	To middle of BMP in I8-sequence To end of BMP in UTF-8
3	400 to 7FFF	800 to FFFF	
4	8000 to 3 FFFF	1 0000 to 1F FFFF	To end of plane 3 in I8-sequence To end of plane 16 in UTF-8
5	4 0000 to 3F FFFF	20 0000 to 3FF FFFF	To end of plane 16 in I8-sequence To end of UCS in UTF-8
6	40 0000 to 3FF FFFF	400 0000 to 7FFF FFFF	
7	400 0000 to 7FFF FFFF	Not used	To end of UCS in I8-sequence
<b>Trailing Bytes</b>	32 values - X'A0' -- X'BF' B'101vvvv' 5 v-bits per byte	64 values - X'80' -- X'BF' B'10vvvvv' 6 v-bits per byte	I8-sequence trailing byte has only five information bits per trailing byte, compared to 6 in UTF-8
<b>Lead Bytes for:</b>	<b>Hex</b>	<b>Hex</b>	
2-Byte sequence	C0 -- DF	C0 -- DF	Same in both
3-Byte sequence	E0 -- EF	E0 -- EF	Same in both
4-Byte sequence	F0 -- F7	F0 -- F7	Same in both
5-Byte sequence	F8 -- FB	F8 -- FB	Same in both
6-Byte sequence	FC and FD	FC and FD	Same in both
7-Byte sequence	FE and FF	Not used	Only used in UTF-8M



## Bibliography

- The Unicode Standard Version 2.0:** The Unicode Consortium ISBN 0-201-48345-9, Addison Wesley Developers Press, July 1996.
- CDRA:** IBM - Character Data Representation Architecture - Reference and Registry, SC09-2190-00, December 1996.
- ISO/IEC 10646-1: 1993(E):** Information Processing - Universal Coded Character Set (UCS):Part 1, Basic Multilingual Plane
- Amendment 1 to ISO/IEC 10646-1:** Transformation Format for 16 Planes of Group 00 (UTF-16); 1996
- Amendment 2 to ISO/IEC 10646-1:** Transformation Format 8 (UTF-8)
- ISO/IEC 646:** Information Processing - 7-Bit Coded Character Set for Information Interchange
- ISO/IEC 4873:** Information Processing - 8-Bit Code for Information Interchange - Structure and Rules for implementation
- ISO/IEC 6429:** Information Processing - 7-Bit and 8-Bit Coded Character Sets - Control Functions for Coded Character Sets
- ISO/IEC 8859-xx:** Information Processing - 8-Bit Single-Byte Coded Graphic Character Sets (several parts)
- SHARE Report SSD No. 366:** ASCII and EBCDIC Character Set and Code Issues in Systems Application Architecture, The ASCII/EBCDIC Character Set Task Force. Edited by Edwin Hart, The Johns Hopkins University, Applied Physics Laboratory, Laurel, Maryland, USA; published by Share Inc., 111 East Wacker Drive, Chicago, Illinois, USA 60601; June 1989

## ANNEX - Intellectual Property Related

Transcript of Letter  
regarding Disclosure of IBM Technology - EF-UTF  
(Hard copy is on file with the Chair of UTC and the Chair of NCITS/L2)  
Transcribed on 1998-07-11

---

International Business Machines Corporation

IBM LOGO  
Route 100  
Somers, NY 10589

June 2, 1998

The Chair, Unicode Technical Committee

Subject: Disclosure of IBM Technology - EBCDIC-Friendly UCS Transformation Format (EF-UTF)

The attached document entitled "EBCDIC-Friendly UCS Transformation Format (EF-UTF)" contains IBM technology that has been filed for application for Canadian Patent. However, IBM believes that the technology could be beneficial to the EBCDIC community at large; allowing the community to derive the enormous benefits provided by UCS (ISO/IEC 10646 and Unicode).

This letter is to inform you that IBM is pleased to make the attached documentation, and the associated technology that has been filed for patent, freely available to anyone concerned towards making the transformation format as part of the UCS standards.

Sincerely

SIGNED

Elizabeth G. Nichols  
Director of National Language Support  
and Information Development

EGN:ghs  
Attachment

---

(Note: The term EF-UTF has been changed to UTF-EBCDIC at the suggestion of UTC meeting 78 -- V.S. Umamaheswaran)

## Changes from previous revisions

This is the second version of this technical report. It has been completely re-written based on input received from some UTC reviewers to make it more suitable as a Unicode technical report than a tutorial document.

## Copyright

*Copyright (C)1999 Unicode, Inc.. All Rights Reserved. The Unicode Consortium makes no expressed or implied warranty of any kind, and assumes no liability for errors or omissions. No liability is assumed for incidental and consequential damages in connection with or arising out of the use of the information or programs contained or accompanying this technical report.*

**PROPOSED TEXT FOR AMENDMENT ON UTF-EBCDIC**

Add the following new annex:

**Annex ?**

(informative)

**EBCDIC-Friendly UCS Transformation Format (UTF-EBCDIC)**

UTF-EBCDIC is an alternative coded representation form for all of the characters of the UCS. It can be used to transmit text data through communication systems which assume that individual octets in the range 00 to 3F and FF have a definition according to the IBM EBCDIC standard. It is not intended to be used in open interchange, where one of the other normative forms defined in this standard should be used.

The number of octets in the UTF-EBCDIC coded representation of the characters of the UCS ranges from one to seven; the value of the first octet indicates the number of octets in that coded representation.

**?1 Features of UTF-EBCDIC**

- Control functions in positions 00000000 to 0000001F, and the DELETE character in position 0000007F, are represented in the IBM EBCDIC control function positions of 00 to 3F, and FF.
- Characters in the 7-bit ASCII (ISO/IEC 646:1991-IRV) repertoire--00000020 to 0000007E--are transformed to identical characters in the IBM EBCDIC code page 1047 standard.
- The first octet in the UTF-EBCDIC coded representation of any character can be directly identified when a CC-data-element is examined, one octet at a time, starting from an arbitrary location. With a simple octet look up, an associated octet indicates the number of continuing octets (if any) in the multi-octet sequence that constitutes the coded representation of that character.

**?2 Specification of UTF-EBCDIC**

In the UTF-EBCDIC coded representation form each character from this International Standard shall have a coded representation that comprises a sequence of octets of length 1, 2, 3, 4, 5, 6, or 7 octets.

The UTF-EBCDIC transformation consists of two steps:

1. The first step transforms the UCS-4 coded representation of the character into an ISO-8-compatible string form (l8-string).
2. The second step maps each octet of the l8-string to a corresponding octet of the UTF-EBCDIC coded representation of the character.

**?3 Transformation of UCS-4 form to l8-string form**

In the l8-string coded representation form each character from this International Standard shall have a coded representation that comprises a sequences of octets of length 1, 2, 3, 4, 5, 6, or 7 octets.

For all sequences of one octet, its value will be between 00 and 9F.

For all sequences of more than one octet, the number of ONE bits in the first octet, starting from the most significant bit position, shall indicate the number of octets in the sequence. The next most significant bit shall be a ZERO bit, except when there are seven ONE bits in the first octet starting from the most significant bit position, in such situation the next most significant bit can be either a ZERO or ONE bit.

NOTE 1 - For example, the first octet of a 2-octet sequence has bits 110 in the most significant positions, and the first octet of a 6-octet sequence has bits 1111110 in the most significant positions.

All of the octets, other than the first in a sequence, are known as continuing octets. The three most significant bits of a continuing octet shall be a ONE bit followed by a ZERO bit followed by a ONE bit.

The remaining bit positions in the octets of the sequence shall be "free bit positions" that are used to distinguish between the characters of this International Standard. These free bit positions shall be used, in order of increasing significance, for the bits of the UCS-4 coded representation of the character, starting from its least significant bit. Some of the high-order ZERO bits of the UCS-4 representation shall be omitted, as specified below.

Table 1 below shows the format of the octets of a coded character in I8-string form. Each free bit position available for distinguishing between the characters is indicated by an x. Each entry in the column "Maximum UCS-4 value" indicates the upper end of the range of coded representations from UCS-4 that may be represented in a I8-string sequence having the length indicated in the "I8-string octet usage" column.

Table 1 shows that, in a CC-data-element conforming to I8-string, the range of values for each octet indicates its usage as follows:

00 to 9F first and only octet of a sequence;  
A0 to BF continuing octet of a multi-octet

sequence;

C5 to DF first octet of a two-octet sequence.  
E1 to EF first octet of a three-octet sequence.  
F0 to F7 first octet of a four-octet sequence.  
F8 to FB first octet of a five-octet sequence.  
FC and FD first octet of a six-octet sequence.  
FE and FF first octet of a seven-octet sequence.

The mapping between UCS-4 and I8-string shall be as shown in ?.5; the reverse mapping is shown in ?.6.

NOTE 2 - Examples of UCS-4 coded representations and the corresponding I8-string coded representations are shown in Tables 2 and 3 below.

Table 2 shows the UCS-4 and the I8-string coded representations, in binary notation, for a selection of code positions from the UCS.

Table 3 shows the UCS-4 and the I8-string coded representations, in hexadecimal notation, for the same selection of code positions from the UCS.

**Table 1 - Format of octets in an I8-string sequence**

I8-string octet usage	Format (binary)	No. of free bits	Maximum UCS-4 value
1st of 1	xxxxxxx	8	0000 009F
1st of 2	110xxxx	5	0000 03FF
1st of 3	1110xxx	4	0000 3FFF
1st of 4	11110xx	3	0003 FFFF
1st of 5	111110xx	2	003F FFFF
1st of 6	1111110x	1	03FF FFFF
1st of 7	1111111x	1	7FFF FFFF
continuing (2nd .. 7th )	101xxxx	5	

**Table 2 - Examples in binary notation**

Four-octet form - UCS-4	I8-string form
00000000 00000000 00000000 00000001;	00000001;
00000000 00000000 00000000 10011111;	10011111;
00000000 00000000 00000000 10100000;	11000101; 10100000;
00000000 00000000 00000011 11111111;	11011111; 10111111;
00000000 00000000 00001100 00000000;	11100001; 10100000; 10100000;
00000000 00000000 00111111 11111111;	11101111; 10111111; 10111111;
00000000 00000000 01000000 00000000;	11110000; 10110000; 10100000; 10100000;
00000000 00000011 11111111 11111111;	11110111; 10111111; 10111111; 10111111;
00000000 00001100 00000000 00000000;	11111000; 10101000; 10100000; 10100000; 10100000;
00000000 00111111 11111111 11111111;	11111011; 10111111; 10111111; 10111111; 10111111;
00000000 01000000 00000000 00000000;	11111100; 10100100; 10100000; 10100000; 10100000; 10100000;
00000011 11111111 11111111 11111111;	11111101; 10111111; 10111111; 10111111; 10111111; 10111111;
00001100 00000000 00000000 00000000;	11111110; 10100010; 10100000; 10100000; 10100000; 10100000; 10100000;
01111111 11111111 11111111 11111111;	11111111; 10111111; 10111111; 10111111; 10111111; 10111111; 10111111;

**Table 3 - Examples in hexadecimal notation**

UCS-4 form	I8-string form
0000 0001;	01;
0000 009F;	9F;
0000 00A0;	C5; A0;
0000 03FF;	DF; BF;
0000 0400;	E1; A0; A0;
0000 3FFF;	EF; BF; BF;
0000 4000;	F0; B0; A0; A0;
0003 FFFF;	F7; BF; BF; BF;
0004 0000;	F8; A8; A0; A0; A0;
003F FFFF;	FB; BF; BF; BF; BF;
0040 0000;	FC; A4; A0; A0; A0; A0;
03FF FFFF;	FD; BF; BF; BF; BF; BF;
0400 0000;	FE; A2; A0; A0; A0; A0; A0;
7FFF FFFF;	FF; BF; BF; BF; BF; BF; BF;

**?4 Notation**

1. All numbers are in hexadecimal notation, except for the decimal numbers used in the power-of operation (see 5 below).
2. Boundaries of code elements are indicated with semicolons; these are single-octet boundaries within the I8-string coded representations, and four-octet boundaries within UCS-4 coded representations.
3. The symbol "%" indicates the modulo operation, e.g.:  $x \% y = x \text{ modulo } y$
4. The symbol "/" indicates the integer division operation, e.g.:  $7 / 3 = 2$
5. Superscripting indicates the power-of operation, e.g.:  $2^3 = 8$
6. Precedence is: power-of operation > integer division > modulo operation > integer multiplication > integer addition.  
e.g.:  $x / y^z \% w = ((x / (y^z)) \% w)$

**?5 Mapping from UCS-4 form to I8-string form**

Table 4 defines in mathematical notation the mapping from the UCS-4 coded representation form to the I8-string coded representation form.

In the left column (UCS-4) the notation  $x$  indicates the four-octet coded representation of a single character of the UCS. In the right column (I8-string)  $x$  indicates the corresponding integer value.

NOTE 3 - Values of  $x$  in the range 0000D800 .. 0000DFFF are reserved for the UTF-16 form and do not occur in UCS-4. The values 0000FFFE and 0000FFFF also do not occur (see clause 8). The mappings of these code positions in I8-string are undefined.

NOTE 4 - The algorithm for converting from UCS-4 to I8-string can be summarised as follows.

For each coded character in UCS-4 the length of octet sequence in I8-string is determined by the entry in the right column of Table 1. The bits in the UCS-4 coded representation, starting from the least significant bit, are then distributed across the free bit positions in order of increasing significance until no more free bit positions are available.

**Table 4 - Mapping from UCS-4 to I8-string**

Range of values in UCS-4	Sequence of octets in I8-string
$x = 0000\ 0000 \dots 0000\ 009F;$	$x;$
$x = 0000\ 00A0 \dots 0000\ 03FF;$	$C0 + x / 2^5;$ $A0 + x \% 2^5;$
$x = 0000\ 0400 \dots 0000\ 3FFF;$	$E0 + x / 2^{10};$ $A0 + x / 2^5 \% 2^5;$ $A0 + x \% 2^5;$
$x = 0000\ 4000 \dots 0003\ FFFF;$ (see Note 3)	$F0 + x / 2^{15};$ $A0 + x / 2^{10} \% 2^5;$ $A0 + x / 2^5 \% 2^5;$ $A0 + x \% 2^5;$
$x = 0004\ 0000 \dots 003F\ FFFF;$	$F8 + x / 2^{20};$ $A0 + x / 2^{15} \% 2^5;$ $A0 + x / 2^{10} \% 2^5;$ $A0 + x / 2^5 \% 2^5;$ $A0 + x \% 2^5;$
$x = 0040\ 0000 \dots 03FF\ FFFF;$	$FC + x / 2^{25};$ $A0 + x / 2^{20} \% 2^5;$ $A0 + x / 2^{15} \% 2^5;$ $A0 + x / 2^{10} \% 2^5;$ $A0 + x / 2^5 \% 2^5;$ $A0 + x \% 2^5;$
$x = 0400\ 0000 \dots 7FFF\ FFFF;$	$FE + x / 2^{30};$ $A0 + x / 2^{25} \% 2^5;$ $A0 + x / 2^{20} \% 2^5;$ $A0 + x / 2^{15} \% 2^5;$ $A0 + x / 2^{10} \% 2^5;$ $A0 + x / 2^5 \% 2^5;$ $A0 + x \% 2^5;$

**?6 Mapping from I8-string form to UCS-4 form**

Table 5 defines in mathematical notation the mapping from the I8-string coded representation form to the UCS-4 coded representation form.

In the left column (I8-string) the following notations apply:

- $z$  is the first octet of a sequence. Its value determines the number of continuing octets in the sequence.
- $y$  is the 2nd octet in the sequence.
- $x$  is the 3rd octet in the sequence.
- $w$  is the 4th octet in the sequence.
- $v$  is the 5th octet in the sequence.
- $u$  is the 6th octet in the sequence.
- $t$  is the 7th octet in the sequence.

The ranges of values applicable to these octets are shown in ?.3 above, following Table 1.

NOTE 5- The algorithm for converting from I8-string to UCS-4 can be summarised as follows.

For each coded character in I8-string the bits in the free bit positions are concatenated as a bit-string. The bits from this string, in increasing order of significance, are then distributed across the bit positions of a four-octet sequence, starting from the least significant bit position. The remaining bit positions of that sequence are filled with ZERO bits.

## ?.7 Mapping from I8-string form to UTF-EBCDIC form

In the UTF-EBCDIC coded representation form each character from this International Standard shall have a coded representation that comprises a sequences of octets of length 1, 2, 3, 4, 5, 6, or 7 octets.

Table 6 defines the mapping from an I8-string octet to the corresponding UTF-EBCDIC octet.

## ?.8 Mapping from UTF-EBCDIC form to I8-string form

In the I8-string coded representation form each character from this International Standard shall have a coded representation that comprises a sequences of octets of length 1, 2, 3, 4, 5, 6, or 7 octets.

Table 7 defines the mapping from an UTF-EBCDIC octet to the corresponding I8-string octet.

## ?.9 Identification of UTF-EBCDIC ???

The differences in the EBCDIC and ISO-2022 conventions are such that it is undesirable to be able to invoke an the UTF-EBCDIC encoding from an ISO-2022 or vice versa.

Identification of UTF-EBCDIC should be done by use of a higher-level protocol and is undefined in this standard.

## ?.10 Incorrect sequences of octets: Interpretation by receiving devices

According to ?.3 an octet in the range 00 .. 9F or C5 .. DF is the first octet of an I8-string sequence, and is followed by the appropriate number (from 0 to 6) of continuing octets in the range A0 .. BF. Furthermore, octets whose value is C0, C1, C2, C3, C4, or E0 are not used; thus they are invalid in I8-string (and UTF-EBCDIC).

If a CC-data-element includes either:

- a first octet that is not immediately followed by the correct number of continuing octets, or
- one or more continuing octets that are not required to complete a sequence of first and continuing octets, or
- an invalid octet,

then according to ?.3 such a sequence of octets is not in conformance with the requirements of I8-string. It is known as a malformed I8-string sequence.

If a receiving device that has adopted the UTF-EBCDIC form receives a sequence of octets such that after the mapping done according to ?.8 yields a malformed I8-string sequence, because of error conditions either:

- in an originating device, or
- in the interchange between an originating and a receiving device, or
- in the receiving device itself,

then it shall interpret that malformed sequence in the same way that it interprets a character that is outside the adopted subset that has been identified for the device (see 2.3c).

**Table 5 - Mapping from I8-string to UCS-4**

Sequence of octets in I8-string	Four-octet sequences in UCS-4
z = 00..9F;	z;
z = C5..DF; y;	$(z-C0)*2^5 + (y-A0);$
z = E1..EF; y; x;	$(z-E0)*2^{10} + (y-A0)*2^5 + (x-A0);$
z = F0..F7; y; x; w;	$(z-F0)*2^{15} + (y-A0)*2^{10} + (x-A0)*2^5 + (w-A0);$
z = F8..FB; y; x; w; v;	$(z-F8)*2^{20} + (y-A0)*2^{15} + (x-A0)*2^{10} + (w-A0)*2^5 + (v-A0);$
z = FC, FD; y; x; w; v; u;	$(z-FC)*2^{25} + (y-A0)*2^{20} + (x-A0)*2^{15} + (w-A0)*2^{10} + (v-A0)*2^5 + (u-A0);$
z = FE, FF; y; x; w; v; u; t;	$(z-FE)*2^{30} + (y-A0)*2^{25} + (x-A0)*2^{20} + (w-A0)*2^{15} + (v-A0)*2^{10} + (u-A0)*2^5 + (t-A0);$

**Table 6 - Mapping from I8-string octet to UTF-EBCDIC octet**

	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
0-	00	01	02	03	37	2D	2E	2F	16	05	25	0B	0C	0D	0E	0F
1-	10	11	12	13	3C	3D	32	26	18	19	3F	27	1C	1D	1E	1F
2-	40	5A	7F	7B	5B	6C	50	7D	4D	5D	5C	4E	6B	60	4B	61
3-	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	7A	5E	4C	7E	6E	6F
4-	7C	C1	C2	C3	C4	C5	C6	C7	C8	C9	D1	D2	D3	D4	D5	D6
5-	D7	D8	D9	E2	E3	E4	E5	E6	E7	E8	E9	AD	E0	BD	5F	6D
6-	79	81	82	83	84	85	86	87	88	89	91	92	93	94	95	96
7-	97	98	99	A2	A3	A4	A5	A6	A7	A8	A9	C0	4F	D0	A1	07
8-	20	21	22	23	24	15	06	17	28	29	2A	2B	2C	09	0A	1B
9-	30	31	1A	33	34	35	36	08	38	39	3A	3B	04	14	3E	FF
A-	80	8C	8D	8E	8F	90	9C	9D	9E	9F	A0	AC	AE	AF	BC	BE
B-	BF	CC	CD	CE	CF	DC	DD	DE	DF	EC	ED	EE	EF	FC	FD	FE
C-	42	43	44	45	46	47	48	49	52	53	54	55	56	57	58	59
D-	62	63	64	65	66	67	68	69	71	72	73	74	75	76	77	78
E-	8A	9A	AA	BA	CA	DA	EA	FA	8B	9B	AB	BB	CB	DB	EB	FB
F-	B2	B3	B4	B5	B6	B7	B8	B9	6A	70	B0	B1	41	51	4A	E1

**Table 7 - Mapping from UTF-EBCDIC octet to I8-string octet**

	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
0-	00	01	02	03	9C	09	86	7F	97	8D	8E	0B	0C	0D	0E	0F
1-	10	11	12	13	9D	85	08	87	18	19	92	8F	1C	1D	1E	1F
2-	80	81	82	83	84	0A	17	1B	88	89	8A	8B	8C	05	06	07
3-	90	91	16	93	94	95	96	04	98	99	9A	9B	14	15	9E	1A
4-	20	FC	C0	C1	C2	C3	C4	C5	C6	C7	FE	2E	3C	28	2B	7C
5-	26	FD	C8	C9	CA	CB	CC	CD	CE	CF	21	24	2A	29	3B	5E
6-	2D	2F	D0	D1	D2	D3	D4	D5	D6	D7	F8	2C	25	5F	3E	3F
7-	F9	D8	D9	DA	DB	DC	DD	DE	DF	60	3A	23	40	27	2D	22
8-	A0	61	62	63	64	65	66	67	68	69	EO	E8	A1	A2	A3	A4
9-	A5	6A	6B	6C	6D	6E	6F	70	71	72	E1	E9	A6	A7	A8	A9
A-	AA	7E	73	74	75	76	77	78	79	7A	E2	EA	AB	5B	AC	AD
B-	FA	FB	F0	F1	F2	F3	F4	F5	F6	F7	E3	EB	AE	5D	AF	B0
C-	7B	41	42	43	44	45	46	47	48	49	E4	EC	B1	B2	B3	B4
D-	7D	4A	4B	4C	4D	4E	4F	50	51	52	E5	ED	B5	B6	B7	B8
E-	5C	FF	53	54	55	56	57	58	59	5A	E6	EE	B9	BA	BB	BC
F-	30	31	32	33	34	35	36	37	38	39	E7	EF	BD	BE	BF	9F