Appendix F FSS-UTF

File System Safe UCS Transformation format

An informative annex to ISO10646-1 defines a UCS[5] Transformation Format called UTF-1. This format protects control bytes; specifically the null byte. That is, no byte in a UTF-1 string contains a null. This enables a UTF-1 string to be processed by operations that take special action on a null byte.

However, UTF-1 does not protect the ASCII slash character (/). This character is used as a special character in many file systems. Consequently the X/Open Company Ltd published a transformation format called File System Safe UCS Transformation Format (FSS_UTF). The format is published as an X/Open Preliminary Specification, Document Number:P316. The details are reproduced below.

This appendix is for information only: it does not form part of the Unicode Standard Version 1.1

F.1 Criteria for the Transformation Format

The FSS-UTF meets the following criteria:

1. It is compatible with historical file systems (which disallow the null byte and the ASCII slash character as a part of the file name).
2. It is compatible with existing programs. The existing model for multi-byte processing is that ASCII values do not occur in a single byte of a multi-byte encoding. An FSS-UTF representation of a non-ASCII character contains no ASCII code values. If the Unicode value is in the range [0x00, 0x7F] the transformation is in this range; otherwise, the transformed byte sequence does not contain any bytes in the range [0x00, 0x7F].
3. It is easy to convert from and to Unicode.
4. The first byte indicates the number of bytes to follow in a multi-byte sequence.
5. The FSS-UTF is not be extravagant in terms of number of bytes used for encoding.
6. It is be possible to find the start of a character efficiently starting from an arbitrary location in a byte stream.

F.2 Specification

The FSS-UTF encodes character values in the range [0, 0x7FFFFFFF][6] using multi-byte characters of lengths 1, 2, 3, 4, 5, and 6 bytes. For all encodings of more than one byte, the initial byte determines the number of bytes used by setting 1 in the equivalent number of high-order bytes. The next most significant bit is always 0. For example a 2-byte sequence starts with 110 and a 6-byte sequence starts with 1111110

The following table shows the format of the first byte of a character; the free bits available for coding the character are indicated by an x.

---

5. UCS is an abbreviation for the 10646 character set. Unicode is identical in code and repertoire with the 2 byte form, UCS-2.

6. Unicode only requires values up to FFFF and so only uses multi-byte characters of lengths up to 3, but for completeness the full ranges of the format are described.

Byte	Value	Bits Free
First of 2 bytes	110xxxxx	5
First of 3 bytes	1110xxxx	4
First of 4 bytes	11110xxx	3
First of 5 bytes	111110xx	2
First of 6 bytes	1111110x	1
All subsequent bytes	10xxxxxx	6

Therefore any byte that does not start with 10 is the start of a FSS-UTF character sequence. The figure below illustrates the FSS-UTF:

Bits	Hex Min	Hex Max	Byte Sequence in Binary
7	00000000	0000007f	0vvvvvvv
11	00000080	000007FF	110vvvvv 10vvvvvv
16	00000800	0000FFFF	1110vvvv 10vvvvvv 10vvvvvv
21	00010000	001FFFFF	11110vvv 10vvvvvv 10vvvvvv 10vvvvvv
26	00200000	03FFFFFF	111110vv 10vvvvvv 10vvvvvv 10vvvvvv 10vvvvvv
31	04000000	7FFFFFFF	1111110v 10vvvvvv 10vvvvvv 10vvvvvv 10vvvvvv 10vvvvvv

The Unicode value is just the concatenation of the v bits in the multibyte encoding. When there are multiple ways to encode a value, for example U+0000, only the shortest encoding is legal.

F.3 Sample Implementations

Below are sample implementations of the C standard wctomb() and mbtowc() functions which demonstrate the algorithms for converting from Unicode to the transformation format and converting from the transformation format to Unicode. The sample implementations include error checks, some of which may not be necessary for conformance:

```
typedef struct {
        int     cmask;
        int     cval;
        int     shift;
        long    lmask;
        long    lval;
}       Tab;

static Tab tab[] = {
        0x80,   0x00,   0*6,    0x7F,           0,              /* 1 byte sequence */
        0xE0,   0xC0,   1*6,    0x7FF,          0x80,           /* 2 byte sequence */
        0xF0,   0xE0,   2*6,    0xFFFF,         0x800,          /* 3 byte sequence */
        0xF8,   0xF0,   3*6,    0x1FFFFF,       0x10000,        /* 4 byte sequence */
        0xFC,   0xF8,   4*6,    0x3FFFFFF,      0x200000,       /* 5 byte sequence */
        0xFE,   0xFC,   5*6,    0x7FFFFFFF,     0x4000000,      /* 6 byte sequence */
        0,                                                      /* end of table */
};

int mbtowc ( wchar_t *p, char *s, size_t n ) {
        long l;
        int c0, c, nc;
        Tab *t;

        if ( s == 0 ) return 0;
        nc = 0;
        if ( n <= nc ) return -1;
        c0 = *s & 0xff;
        l = c0;
        for ( t = tab; t->cmask; t++) {
                nc++;
                if ( ( c0 & t->cmask ) == t->cval ) {
                        l &= t->lmask;
                        if ( l < t->lval ) return -1;
                        *p = l;
                        return nc;
```

```
                }
                if ( n <= nc )
                return -1;
                s++;
                c = ( *s ^ 0x80 ) & 0xFF;
                if ( c & 0xC0 ) return -1;
                l = ( l << 6 ) | c;
        }
        return -1;
}

int wctomb ( char *s, wchar_t wc ) {
        long l;
        int c, nc;
        Tab *t;

        if (s == 0 ) return 0;
        l = wc;
        nc = 0;
        for ( t=tab; t->cmask; t++ ) {
                nc++;
                if ( l <= t->lmask ) {
                        c = t->shift;
                        *s = t->cval | ( l >> c );
                        while ( c > 0 ) {
                                c -= 6;
                                s++;
                                *s = 0x80 | ( ( l >> c ) & 0x3F );
                        }
                        return nc;
                }
        }
        return -1;
}
```