Network Working Group
Internet Draft: Multi-Lingual String Format
Document: draft-ietf-acap-mlsf-01.txt
C. Newman
Innosoft
June 1997
Expires in six months

Multi-Lingual String Format (MLSF)

Status of this memo

Abstract

The IAB charset workshop [IAB-CHARSET] concluded that for human readable text there should always be a way to specify the natural language. Many protocols are designed with an attribute-value model (including RFC 822, HTTP, LDAP, SNMP, DHCP, and ACAP) which stores many small human readable text strings. The primary function of an attribute-value model is to simplify both extensibility and searchability. A solution is needed to provide language tags in these small human readable text strings, which does not interfere with these primary functions.

This specification defines MLSF (Multi-Lingual String Format) which applies another layer of encoding on top of UTF-8 [UTF-8] to permit

the addition of language tags anywhere within a text string. In addition, it defines an alternate form which can be used to include alternative representations of the same text in different character sets. MLSF has the property that UTF-8 is a proper subset of MLSF. This preserves the searchability requirement of the attribute-value model.

Appendix F of this document includes a brief discussion of the background behind MLSF and why some other potential solutions were rejected for this purpose.

1. Conventions used in this document

The key words "MUST", "MUST NOT", "SHOULD", "SHOULD NOT", and "MAY" in this document are to be interpreted as defined in "Key words for use in RFCs to Indicate Requirement Levels" [KEYWORDS].

2. MLSF simple form

MLSF uses "Tags for the Identification of Languages" [LANG-TAGS] as the basis for language identification.

Language tags are encoded by mapping them to upper-case, then adding hexadecimal A0 to each octet. The result is broken up into groups of five octets followed by a final group of five or fewer octets. Each group is prefixed by a UTF-8-style length count with the low bits set to 0. See Appendix D for sample source code to perform this conversion.

MLSF simple form is defined by the MLSF-SIMPLE rule in section 7. A quoted version of MLSF simple form is defined by the MLSF-SIMPLE-QUOTED rule.

Note that MLSF is not compatible with UTF-8. A program which uses MLSF MUST downconvert it to UTF-8 prior to using it in a context where UTF-8 is required. Sample code for this down conversion is included in Appendix B.

3. MLSF alternative form

A MLSF alternative form string may contain alternative representations of the same text in different primary languages. The octet with hexadecimal representation of FE is used to introduce a new alternative. This MUST be followed by a MLSF language tag for the primary language of the alternative.

The component of the MLSF string prior to the first FE octet is considered the "preferred" representation for the string. This is the version which will be displayed by MLSF clients which choose not to support alternative representations. The preferred representation MAY be prefixed by a MLSF language tag.

MLSF alternate form is defined by the MLSF-ALT rule in section 7. A quoted version of MLSF alternate form is defined by the MLSF-ALT-QUOTED rule.

Note that MLSF alternate form is not compatible with UTF-8. A program which uses MLSF MUST downconvert it to UTF-8 prior to using it in a context where UTF-8 is required. Sample code for this down conversion is included in Appendix B.

4. MLSF MIME character sets

The character set label "XXXX-simple" will be registered to indicate the use of MLSF simple form. The character set label "XXXX-alt" will be registered to indicate the use of MLSF alternate form.

MLSF may be used in conjunction with MIME header [MIME-HDR] encoding to permit language tagging and alternative representations in header fields. A work in progress [MIME-LANG] will propose a mechanism for language tagging in headers which is not dependent on the use of UTF-8.

For single language MIME body parts, the UTF-8 character set with an appropriate Content-Language [LANG-TAG] header SHOULD be used instead of MLSF. Text/enriched [ENRICHED] or HTML with language tags [HTML-I18N] are preferred to using MLSF for MIME bodies when possible.

5. Security Considerations

Multi-Lingual String Format is not believed to have any security considerations beyond those for simple US-ASCII strings. In particular, unfiltered display of certain US-ASCII control characters by a terminal emulator may result in modifying the behavior of the terminal emulator (e.g. by redefining function keys) such that security can be breached. Programs which display text to a potentially insecure terminal emulator channel are encouraged to remove control characters to avoid these problems.

6. Formal Grammar

This section defines the formal grammar for MLSF using Augmented BNF [ABNF] notation.

```
MLSF-ALT           = [[MLSF-LANG-TAG] MLSF-COMPONENT
                      *(MLSF-ALTERNATE MLSF-COMPONENT)]

MLSF-ALT-QUOTED    = <"> [[MLSF-LANG-TAG] MLSF-COMPONENT-Q
                      *(MLSF-ALTERNATE MLSF-COMPONENT-Q)] <">

MLSF-ALTERNATE     = %xFE MLSF-LANG-TAG

MLSF-COMPONENT     = UTF8-NON-NUL *([MLSF-LANG-TAG] UTF8-NON-NUL)

MLSF-COMPONENT-Q   = UTF8-QUOTED *([MLSF-LANG-TAG] UTF8-QUOTED)

MLSF-LANG-TAG      = *MLSF-LANG-5 (MLSF-LANG-1 / MLSF-LANG-2 /
                     MLSF-LANG-3 / MLSF-LANG-4 / MLSF-LANG-5)
                     ;; Encoded version of Language-Tag from RFC 1766
                     ;; characters converted to uppercase, with
                     ;; A0 added and broken into MLSF-LANG components

MLSF-LANG-CONT     = %xCD / %xE1..FA

MLSF-LANG-1        = %xC0 MLSF-LANG-CONT

MLSF-LANG-2        = %xE0 2MLSF-LANG-CONT

MLSF-LANG-3        = %xF0 3MLSF-LANG-CONT

MLSF-LANG-4        = %xF8 4MLSF-LANG-CONT

MLSF-LANG-5        = %xFC 5MLSF-LANG-CONT

MLSF-SIMPLE        = [[MLSF-LANG-TAG] MLSF-COMPONENT]

MLSF-SIMPLE-QUOTED = <"> [[MLSF-LANG-TAG] MLSF-COMPONENT-Q] <">

QUOTED             = "\" QUOTED-SPECIAL

QUOTED-SPECIAL     = "\" / <">

US-ASCII-SAFE      = %x01..09 / %x0B..0C / %x0E..21
                     / %x23..5B / %x5D..7F
                    ;; US-ASCII except QUOTED-SPECIALs, CR, LF, NUL

UTF8-NON-NUL       = UTF8-SAFE / CR / LF / QUOTED-SPECIAL
```

```
     UTF8-QUOTED        = UTF8-SAFE / QUOTED

     UTF8-SAFE          = US-ASCII-SAFE / UTF8-1 / UTF8-2 / UTF8-3
                          / UTF8-4 / UTF8-5

     UTF8-CONT          = %x80..BF

     UTF8-1             = %xC0..DF UTF8-CONT

     UTF8-2             = %xE0..EF 2UTF8-CONT

     UTF8-3             = %xF0..F7 3UTF8-CONT

     UTF8-4             = %xF8..FB 4UTF8-CONT

     UTF8-5             = %xFC..FD 5UTF8-CONT
```

7. References

[ABNF] Crocker, D., "Augmented BNF for Syntax Specifications: ABNF", Work in progress: draft-ietf-drums-abnf-xx.txt

[ENRICHED] Resnick, Walker, "The text/enriched MIME Content-type", RFC 1896, Qualcomm, InterCon, February 1996.

<ftp://ds.internic.net/rfc/rfc1896.txt>

[HTML-I18N] Yergeau, Nicol, Adams, Duerst, "Internationalization of the Hypertext Markup Language", RFC 2070, Alis Technologies, Electronic Book Technologies, Spyglass, University of Zurich, January 1997.

<ftp://ds.internic.net/rfc/rfc2070.txt>

[IAB-CHARSET] Weider, Preston, Simonsen, Alvestrand, Atkinson, Crispin, Svanberg, "The Report of the IAB Character Set Workshop held 29 February - 1 March, 1996", RFC 2130, April 1997.

<ftp://ds.internic.net/rfc/rfc2130.txt>

[IMAP4] Crispin, "Internet Message Access Protocol - Version 4rev1", RFC 2060, University of Washington, December 1996.

<ftp://ds.internic.net/rfc/rfc2060.txt>

[KEYWORDS] Bradner, "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, Harvard University, March 1997.

<ftp://ds.internic.net/rfc/rfc2119.txt>

[LANG-TAGS] Alvestrand, H., "Tags for the Identification of Languages", RFC 1766.

<ftp://ds.internic.net/rfc/rfc1766.txt>

[MIME-HDR] Moore, "MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text", RFC 2047, University of Tennessee, November 1996.

<ftp://ds.internic.net/rfc/rfc2047.txt>

[MIME-IMB] Freed, Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, Innosoft, First Virtual, November 1996.

<ftp://ds.internic.net/rfc/rfc2045.txt>

[MIME-LANG] Freed, Moore, "MIME Parameter Value and Encoded Words: Character Sets, Language, and Continuations", work in progress, March 1997.

[UTF8] Yergeau, F. "UTF-8, a transformation format of Unicode and ISO 10646", RFC 2044, Alis Technologies, October 1996.

<ftp://ds.internic.net/rfc/rfc2044.txt>

8. Acknowledgements

Special thanks to Mark Crispin for the idea of using unused UTF-8 codes for this purpose.   Thanks are also due to participants of the ACAP WG mailing list who helped review this proposal.

9. Author's Address

Chris Newman
Innosoft International, Inc.
1050 East Garvey Ave. South
West Covina, CA 91790 USA

Email: chris.newman@innosoft.com

Appendix A. Client advice

A simple UTF-8 client is likely to find the source code in Appendix B useful. A simple Latin-1 based client is likely to find the source code in Appendix C useful.

A more sophisticated client will allow the user to select a preferred language and use something like the source code in Appendix E to find the best alternative in an MLSF string. Such clients should also be aware that sometimes the client's preferred language is misconfigured, and the user may wish to have the last few messages repeated after they have changed languages. For this reason, such a client may wish to cache the last few MLSF strings displayed to the user.

Appendix B. Sample code to convert to UTF-8

Here is sample C source code to convert from MLSF to UTF-8.

```
#include <stdio.h>
#include <ctype.h>

/* a UTF8 lookup table */
#define BAD 0x80
#define SEP 0x40
#define EXT 0x20
static unsigned char utlen[256] = {
        /* 0x00 */ BAD,   1,   1,   1,   1,   1,   1,   1,
        /* 0x08 */   1,   1,   1,   1,   1,   1,   1,   1,
        /* 0x10 */   1,   1,   1,   1,   1,   1,   1,   1,
        /* 0x18 */   1,   1,   1,   1,   1,   1,   1,   1,
        /* 0x20 */   1,   1,   1,   1,   1,   1,   1,   1,
        /* 0x28 */   1,   1,   1,   1,   1,   1,   1,   1,
        /* 0x30 */   1,   1,   1,   1,   1,   1,   1,   1,
        /* 0x38 */   1,   1,   1,   1,   1,   1,   1,   1,
        /* 0x40 */   1,   1,   1,   1,   1,   1,   1,   1,
        /* 0x48 */   1,   1,   1,   1,   1,   1,   1,   1,
        /* 0x50 */   1,   1,   1,   1,   1,   1,   1,   1,
        /* 0x58 */   1,   1,   1,   1,   1,   1,   1,   1,
        /* 0x60 */   1,   1,   1,   1,   1,   1,   1,   1,
        /* 0x68 */   1,   1,   1,   1,   1,   1,   1,   1,
        /* 0x70 */   1,   1,   1,   1,   1,   1,   1,   1,
        /* 0x78 */   1,   1,   1,   1,   1,   1,   1,   1,
        /* 0x80 */ EXT, EXT, EXT, EXT, EXT, EXT, EXT, EXT,
        /* 0x88 */ EXT, EXT, EXT, EXT, EXT, EXT, EXT, EXT,
        /* 0x90 */ EXT, EXT, EXT, EXT, EXT, EXT, EXT, EXT,
        /* 0x98 */ EXT, EXT, EXT, EXT, EXT, EXT, EXT, EXT,
        /* 0xA0 */ EXT, EXT, EXT, EXT, EXT, EXT, EXT, EXT,
        /* 0xA8 */ EXT, EXT, EXT, EXT, EXT, EXT, EXT, EXT,
        /* 0xB0 */ EXT, EXT, EXT, EXT, EXT, EXT, EXT, EXT,
        /* 0xB8 */ EXT, EXT, EXT, EXT, EXT, EXT, EXT, EXT,
        /* 0xC0 */   2,   2,   2,   2,   2,   2,   2,   2,
        /* 0xC8 */   2,   2,   2,   2,   2,   2,   2,   2,
        /* 0xD0 */   2,   2,   2,   2,   2,   2,   2,   2,
        /* 0xD8 */   2,   2,   2,   2,   2,   2,   2,   2,
        /* 0xE0 */   3,   3,   3,   3,   3,   3,   3,   3,
        /* 0xE8 */   3,   3,   3,   3,   3,   3,   3,   3,
        /* 0xF0 */   4,   4,   4,   4,   4,   4,   4,   4,
        /* 0xF8 */   5,   5,   5,   5,   6,   6, SEP, BAD
};
```

```
/* Down conversion from NUL terminated MLSF string to UTF-8.
 *  this strips the language tags and only keeps the preferred
 *  representation.
 * It returns the length of the final string.
 * The destination string will not be longer than the source string.
 *  dst and src may be the same for in-place conversion.
 */
int MLSFtoUTF8(unsigned char *dst, unsigned char *src)
{
    unsigned char *start = dst;
    int len;

    for (;;) {
        len = utlen[*src];
        if (len > 6) break;
        /* skip language tags */
        if (len > 1 && src[1] > 0xC0U) {
            while (len && *src != '\0') {
                ++src;
                --len;
            }
            continue;
        }
        /* copy UTF8 character */
        while (len && *src != '\0') {
            *dst = *src;
            ++dst;
            ++src;
            --len;
        }
    }
    *dst = '\0';

    return (dst - start);
}
```

Appendix C. Sample code to convert to Latin-1

```
/* Down conversion from NUL terminated MLSF string to 8859-1
 * The destination string will not be longer than the source string.
 *  fillc is used to fill untranslatable characters,
 *  if fillc is NUL, untranslatable characters are ignored.
 * returns 0 if source only contained latin-1, returns -1 otherwise.
 */
int MLSFtoLatin1(unsigned char *dst, unsigned char *src, int fillc)
{
    int len, result = 0;

    for (;;) {
        len = utlen[*src];
        /* copy US-ASCII */
        if (len == 1) {
            *dst = *src;
            ++dst;
            ++src;
            continue;
        }
        /* stop at illegal character or end of string */
        if (len > 6) break;
        /* skip non-latin1 glyphs and language tags */
        if (*src > 0xC3U || src[1] > 0xC0U) {
            if (src[1] <= 0xC0U) {
                /* non-latin1 glyph found */
                result = -1;
                if (fillc) {
                    *dst = fillc;
                    ++dst;
                }
            }
            while (len && *src != '\0') {
                ++src;
                --len;
            }
            continue;
        }
        /* copy latin 1 character */
        *dst = ((src[0] & 0x03) << 6) | (src[1] & 0x3F);
        ++dst;
        src += 2;
    }
    *dst = '\0';

    return (result);
}
```

Appendix D. Sample code for encoding/decoding language tags

```
/* encode a language tag
 *  the destination must have a size of least (counting terminating NUL):
 *        (6 * strlen(src) + 9) / 5
 *  returns the length of the destination.
 */
int MLSFlangencode(unsigned char *dst, unsigned char *src)
{
    static unsigned char prefix[] = { 0xC0, 0xE0, 0xF0, 0xF8, 0xFC };
    unsigned char *start = dst;
    int len;                   /* source length */
    int complen;               /* component length */
    int i;

    for (len = strlen(src); len > 0; len -= complen) {
        /* find maximal component length */
        complen = len;
        if (len >= 5) {
            complen = 5;
        }
        /* look up component prefix */
        *dst = prefix[complen - 1];
        ++dst;
        /* copy and map characters in component */
        for (i = 0; i < complen; ++i) {
            *dst = (islower(*src) ? toupper(*src) : *src) + 0xA0U;
            ++dst;
            ++src;
        }
    }
    *dst = '\0';

    return (dst - start);
}
```

```
/* decode a language tag
 *  the destination will not be longer than the source
 *  dst and src may be the same for in-place conversion
 * returns the length of the destination
 */
int MLSFlangdecode(unsigned char *dst, unsigned char *src)
{
    unsigned char *start = dst;
    int complen;

    while (src[0] >= 0xC0U && src[1] > 0xC0U) {
        for (complen = utlen[*src++]; complen > 1; --complen) {
            *dst = *src - 0xA0U;
            ++dst;
            ++src;
        }
    }
    *dst = '\0';

    return (dst - start);
}
```

Appendix E. Sample code for selecting the "best" alternative

```
/* select the "best" language match from an MLSF string
 *  assume input language tag has been converted to upper case
 *  assume language tags in string won't exceed 256 characters
 *  "best" is calculated by matching RFC 1766 language tag components
 * returns a pointer to the start of best matching component
 */
unsigned char *MLSFselect(unsigned char *str, unsigned char *tag)
{
    unsigned char ltag[256];
    unsigned char *best, *match1, *match2;
    int bestlen, mlen;

    /* start with match on preferred alternative */
    best = str;
    bestlen = 0;

    /* skip test if no language tag */
    if (tag != NULL && *tag != '\0') {
        do {
            /* get language tag for this component */
            MLSFlangdecode(ltag, str);
```

```
            /* calculate match length of language tags */
            match1 = ltag;
            match2 = tag;
            mlen = 0;
            while (*match1 != '\0' && *match1 == *match2) {
                ++match1, ++match2;
                /* save length of partial match */
                if (*match2 == '-'
                    && (*match1 == '-' || *match1 == '\0')) {
                    mlen = match1 - ltag;
                }
            }

            /* finish on exact match */
            if (*match2 == '\0'
                && (*match1 == '-' || *match1 == '\0')) {
                best = str;
                break;
            }

            /* remember best match */
            if (mlen > bestlen) {
                best = str;
                bestlen = mlen;
            }

            /* skip to next MLSF component */
            while (*str != '\0' && *str++ != 0xFEU)
                ;
        } while (*str != '\0');
    }

    return (best);
}
```

Appendix F. Background and Alternate Solutions

MLSF was designed to deal with language tagging in the context of the ACAP protocol, but is believed to be useful in other contexts. Specific scenarios cited during discussion were human names in address books, system administrator alert error messages, and error messages which include identifiers potentially in a different language from the client's preferred error message language. Since ACAP is an arbitrary attribute-value protocol, it is impossible to imaging all possible scenarios in advance, so a general purpose mechanism was needed.

There have been several attempts to solve language tagging in

attribute value protocols. RFC 822 poses a particularly troublesome scenario, since headers must be 7-bit. The MIME solution to label character sets [MIME-HDR] and languages [MIME-LANG] in headers is thus a necessary evil. The result of this is to make header searching services such as those provided by IMAP [IMAP4] massively more complex. If 8-bit headers were permitted a solution like MLSF would have been far simpler and more efficient.

Another approach taken is demonstrated by the current vCard, iCalendar, and LDAPv3 proposals (all works in progress). These proposals overload the attribute namespace to provide language tagging and creates a concept roughly described as attributes of the attribute. The result of this is that clients have to deal with a multiple attribute response to a query where each attribute may have multiple values. The additional complexity this adds to client processing was deemed unacceptable for ACAP where client simplicity was an important design goal.

Another possible approach is the use of a markup language such as text/enriched [ENRICHED]. While this is certainly a suitable language tagging solution for large text objects such as MIME bodies, it is unsuitable for the attribute-value model where searching is a primary function.