Oct 23, 2000
Dave Steck
Novell, Inc.

Proposed String Conversion Functions for LDAP C SDK

1. The Need for Conversion Functions

Page: 1
Directory data in the LDAPv3 C APIs is in UTF-8 format. Developers often write applications assuming the characters set is limited to ASCII characters. However, as internationalization efforts increase, these applications will fail and require difficult retrofitting to deal with UTF-8 characters correctly.

Part of the problem is that most platforms have no standard way to convert between Multi-byte, Unicode and UTF-8 strings. In cases where the platform does supply conversion routines, they may be specific to the platform.

It is the goal of this proposal to provide LDAP C developers standard cross-platform functionality to be able to convert easily between Mutli-byte, Unicode, and UTF-8 strings.

We would also like to expose the UTF-8 utility functions currently used internally in the OpenLDAP library; utf8_next, utf8_prev, utf8_strchr, etc.

In addition, the run-time libraries we deliver do not include the LDIF routines for converting data to or from base64 encoding. We would like to expose these to users of our LDAP SDK.

We propose adding or exposing functions for:

- Multi-byte ←→ Unicode conversion
- UTF-8 ←→ Unicode conversion
- Multi-byte ←→ UTF-8 conversion
- UTF-8 Utility functions
- Base64 encoding/decoding functions

2. Multi-byte ←→ Unicode Conversions

There are already 5 ANSI C functions dealing with these conversions. They aren't as powerful as the Windows or NetWare extended functions. In particular, they work only on the default code page as defined by the current locale.

These functions are simple and are available on any ANSI C platform. They don't allocate memory or return a pointer to the application when doing a conversion. But they can return the required size of the output buffer for a particular input string, allowing the application to allocate memory if necessary. The string versions of the functions return an error if an unmappable characters is encountered. Since single-character versions of the functions are also available, developers can implement other error handling strategies, such as using replacement characters, or converting to a special sequence allowing round trip conversion without losing characters.

The functions are prototyped in <stdlib.h>

Dave Steck, Novell, Inc.
The Leading Supplier of Net Services Software

mbtowc - Convert a single multi-byte character to a wide character.
wctomb - Convert a single wide character to a multi-byte character.
mbstowcs - Convert a multi-byte string to a wide character string.
wcstombs - Convert a wide character string to a multi-byte string.
mblen - Return the number of bytes in a multi-byte character.

While developers could use these routines directly, we propose adding wrapper functions to the SDK for a couple reasons:

1) So they are documented cleanly and consistently with the other conversion functions in the SDK.
2) It decouples the SDK from the system routines, allowing an implementer freedom to supply a different implementation.

We propose the functions be named with "lstr_" prefixes. While useful to an LDAP SDK, they're really not part of the LDAP protocol, so we hesitate giving them an "ldap_" prefix. The "lstr_" prefix clearly indicates the string manipulation nature of the routines, and groups them together in the documentation.

By basing the conversion routines on these ANSI standard functions, porting to other platforms becomes much easier.

Issue with wchar_t

These ANSI routines use wchar_t arguments. The size of wchar_t is 2 bytes on Windows and Netware, and 4 bytes on Unix. Unicode strings are often typed as "unsigned short *" in implementations. However, on Unix platforms, compiler constructs like L"xyz" and string functions like wcslen() would not work with strings of this type. Basing the functions on wchar_t works better for cross-platform code for these reasons, although developers must keep the 2-byte vs 4-byte issue in mind, not assuming one or the other.

2.1 lstr_mbtowc - Convert a single multi-byte character to a wide character.

```
int lstr_mbtowc( wchar_t *wchar, const char *mbchar, size_t count )

wchar       (OUT) Points to a wide character code to receive the
                  converted character.

mbchar      (IN)  Address of a sequence of bytes.

count       (IN)  The number of bytes of the mbchar argument to check.
                  This should normally be MB_CUR_MAX.

Return Value:
            If successful, the function returns the length in
            bytes of the multi-byte character.

            If mbchar is NULL or points to an empty string, or if count
            is zero, 0 is returned.

            If mbchar contains an invalid multi-byte character, -1 is
            returned.
```

2.2 lstr_wctomb - Convert a single wide character to a multi-byte character.

```
int lstr_wctomb( char *mbchar, wchar_t wchar )

mbchar      (OUT) Points to a byte array to receive the multi-byte
```

characters.

wchar (IN) The wide character to convert.

Return Value:
If successful, the function returns the number of bytes in the converted multi-byte character.

If the wide character is the null character, the function returns 1 and a null is written to *mbchar*.

If *mbchar* is NULL, 0 is returned.

If the wide character cannot be mapped to the local code page, the function returns –1.

2.3 lstr_mbstowcs - Convert a multi-byte string to a wide character string.

```
size_t mbstowcs( wchar_t *wcstr, const char *mbstr, size_t count)
```

wcstr (OUT) Points to the array of wide chars to receive the converted string. May be NULL.

mbstr (IN) The null-terminated string of multi-byte characters to be converted.

count (IN) The number of multi-byte characters to convert, or equivalently, the size of the output buffer in wide characters.

Return Value:
If successful, the function returns the number of wide characters written to *wcstr*, excluding the null termination character, if any.

If *wcstr* is NULL, the function returns the number of wide characters required to contain the converted string, excluding the null termination character.

If an invalid multi-byte sequence is encountered, the function returns –1.

The output string will be null terminated if there is space for it in the output buffer.

2.4 lstr_wcstombs - Convert a wide character string to a multi-byte string.

```
size_t wcstombs( char *mbstr, const wchar_t *wcstr, size_t count)
```

mbstr (OUT) Points to the byte array to receive the converted multi-byte string.

wcstr (IN) The null-terminated wide character string to convert.

count (IN) The size of the output buffer in bytes.

```
Return Value:
            If successful, the function returns the number of bytes
            written to mbstr, excluding the null termination
            character, if any.

            If mbstr is NULL, the function returns the number of bytes
            required to contain the converted string, excluding the
            null termination character.

            If the function encounters a wide character that cannot
            be mapped to a multi-byte sequence, the function returns –
      1.

The output string will be null terminated if there is space for it in
the output buffer.
```

2.5 lstr_mblen - Return the number of bytes in a multi-byte character.

```
int      mblen( const char *mbchar, size_t count )

mbchar      (IN)  Points to the multi-byte character sequence.

count       (IN)  The number of bytes in mbchar to check.  This should
                  normally be set to MB_CUR_MAX.
Return Value:
            If successful, the function returns the number of bytes in
            the multi-byte character (1 or 2).

            If mbchar is NULL or points to an empty string, or if count
            is 0, the function returns 0.

            If mbchar contains an invalid multi-byte characterm, the
            function returns –1.
```

3. UTF-8 ←→ Unicode conversions

The following new conversion routines will be added, following the pattern of the Multi-byte to Unicode routines.

int lstr_utf8towc - Convert a single UTF-8 encoded character to a wide character.
int lstr_wctoutf8 - Convert a single wide character to a UTF-8 sequence.
int lstr_utf8stowcs - Convert a UTF-8 string to a wide character string.
int lstr_wcstoutf8s - Convert a wide character string to a UTF-8 string.

3.1 lstr_utf8towc - Convert a single UTF-8 encoded character to a wide character.

```
int lstr_utf8towc ( wchar_t *wchar, const char *utf8char )

wchar       (OUT) Points to a wide character code to receive the
                  converted character.

utf8char    (IN)  Address of the UTF8 sequence of bytes.

Return Value:
```

If successful, the function returns the length in bytes of the UTF-8 input character.

If *utf8char* is NULL or points to an empty string, the function returns 1 and a NULL is written to *wchar*.

If *utf8char* contains an invalid UTF-8 sequence -1 is returned.

3.2 lstr_wctoutf8 - Convert a single wide character to a UTF-8 sequence.

```
int lstr_wctoutf8 ( char *utf8char, wchar_t wchar )
```

utf8char (OUT) Points to a byte array to receive the converted UTF-8 string.

wchar (IN) The wide character to convert.

Return Value:

If successful, the function returns the length in bytes of the converted UTF-8 output character.

If *wchar* is NULL, the function returns 1 and a NULL is written to *utf8char*.

If *wchar* cannot be converted to a UTF-8 character, the function returns –1.

3.3 lstr_utf8stowcs - Convert a UTF-8 string to a wide character string.

```
int lstr_utf8stowcs (wchar_t *wcstr, const char *utf8str, size_t count)
```

wcstr (OUT) Points to a wide char buffer to receive the converted wide char string.

utf8str (IN) Address of the null-terminated UTF-8 string to convert.

count (IN) The number of UTF-8 characters to convert, or equivalently, the size of the output buffer in wide characters.

Return Value:

If successful, the function returns the number of wide characters written to *wcstr*, excluding the null termination character, if any.

If *wcstr* is NULL, the function returns the number of wide characters required to contain the converted string, excluding the null termination character.

If an invalid UTF-8 sequence is encountered, the function returns –1.

The output string will be null terminated if there is space for it in the output buffer.

3.4 int lstr_wcstoutf8s - Convert a wide character string to a UTF-8 string.

```
int wcstoutf8s ( char *utf8str, const wchar_t *wcstr, size_t count )

utf8str          (OUT) Points to a byte array to receive the converted
                 UTF-8 string.

wcstr       (IN) Address of the null-terminated wide char string to
                 convert.

count       (IN) The size of the output buffer in bytes.

Return Value:
            If successful, the function returns the number of bytes
            written to utf8str, excluding the null termination
            character, if any.

            If utf8str is NULL, the function returns the number of
            bytes required to contain the converted string, excluding
            the null termination character.

            If the function encounters a wide character that cannot
            be mapped to a UTF-8 sequence, the function returns –1.

The output string will be null terminated if there is space for it in
the output buffer.
```

4. Multi-byte ←→ UTF-8 Conversions

The following new routines will be added, following the same pattern. These are the functions that we believe most LDAP C programmers would use, and they can be built upon the previously defined functions. These functions will be implemented by converting the string from Multibyte-to-Wide, then from Wide-to-UTF8, or vice versa.

lstr_mbtoutf8 - Convert a multi-byte character to a UTF-8 character.
lstr_utf8tomb - Convert a UTF-8 character to a multi-byte character.
lstr_mbstoutf8s - Convert a multi-byte string to a UTF-8 string.
lstr_utf8stombs - Convert a UTF-8 string to a multi-byte string.

4.1 lstr_mbtoutf8 - Convert a multi-byte character to a UTF-8 character.

```
int lstr_mbtoutf8 ( char *utf8char, const char *mbchar, size_t count )

utf8char    (OUT) Points to a byte buffer to receive the
                  converted UTF-8 character.

mbchar      (IN)  Address of a sequence of bytes.

count       (IN)  The number of bytes of the mbchar argument to check.
                  This should normally be MB_CUR_MAX.

Return Value:
```

If successful, the function returns the length in bytes of the UTF-8 output character.

If *mbchar* is NULL or points to an empty string, the function returns 1 and a null byte is written to *utf8char*.

If *count* is zero, 0 is returned and nothing is written to *utf8char*.

If *mbchar* contains an invalid multi-byte character, -1 is returned.

4.2 lstr_utf8tomb - Convert a UTF-8 character to a multi-byte character.

```
int lstr_utf8tomb ( char *mbchar, const char *utf8char )

mbchar      (OUT) Points to a byte buffer to receive the converted
                  multi-byte character.

utf8char    (IN)  Address of the UTF-8 character sequence.

Return Value:
```

If successful, the function returns the length in bytes of the multi-byte output character.

If *utf8char* is NULL or points to an empty string, the function returns 1 and a null byte is written to *mbchar*.

If *utf8char* contains an invalid UTF-8 sequence, -1 is returned.

4.3 lstr_mbstoutf8s - Convert a multi-byte string to a UTF-8 string.

```
int lstr_mbstoutf8s (char *utf8char, const char *mbchar, size_t count)

utf8char    (OUT) Points to a byte buffer to receive the
                  converted UTF-8 string

mbchar      (IN)  Address of the null-terminated multi-byte input
                  string.

count       (IN)  The number of bytes of the mbchar argument to check.
                  This should normally be MB_CUR_MAX.

Return Value:
```

If successful, the function returns the length in bytes of the UTF-8 output string, excluding the null terminator, if present.

If *mbchar* is NULL or points to an empty string, the function returns 1 and a null byte is written to *utf8char*.

If *count* is zero, 0 is returned and nothing is written to *utf8char*.

If *mbchar* contains an invalid multi-byte character, -1 is returned.

4.4 int lstr_utf8stombs - Convert a UTF-8 string to a multi-byte string.

```
int lstr_utf8stombs ( char *mbstr, const char *utf8str, size_t count )

mbstr       (OUT) Points to a byte buffer to receive the
                  converted multi-byte

utf8str           (IN)  Address of the null-terminated UTF-8 string to
                  convert.

count       (IN)  The size of the output buffer in bytes.

Return Value:
            If successful, the function returns the number of bytes
            written to mbstr, excluding the null termination
            character, if any.

            If mbstr is NULL, the function returns the number of bytes
            required to contain the converted string, excluding the
            null termination character.

            If an invalid UTF-8 character is encountered, the
            function returns –1.

The output string will be null terminated if there is space for it in
the output buffer.
```

5. UTF8 Utility functions

OpenLDAP recently added several utility functions for dealing with UTF-8 strings. The functions were named with the "ldap_" prefix. We would suggest naming these with a different prefix, as discussed above, and exposing them. The optimization macros would also be exposed.

Functions:

lstr_utf8_charlen(p) - Returns the byte length of this UTF-8 character.
lstr_utf8_chars(p) - Returns the # of chars (not bytes) in a null-terminated UTF-8 string.
lstr_utf8_next(p) - Returns the address of the next UTF-8 character.
lstr_utf8_prev(p) - Returns the address of the previous UTF-8 character.
lstr_utf8_copy(d,s) - Copies one character from src to dest.

Macros. Avoids a function call if it's an ASCII character.

LSTR_UTF8_ISASCII(c) - Returns 1 if c < 0x80.
LSTR_UTF8_CHARLEN(p)
LSTR_UTF8_NEXT(p)
LSTR_UTF8_PREV(p)
LSTR_UTF8_INCR(p)
LSTR_UTF8_DECR(p)
LSTR_UTF8_COPY(d,s)

String functions that deal with single byte characters, modified to work for UTF-8 multiple-byte characters. Could be useful. Probably expose these.

lstr_utf8_strchr
lstr_utf8_strspn
lstr_utf8_strcspn
lstr_utf8_strpbrk
lstr_utf8_strtok (We would suggest changing name to lstr_utf8_strtok_r)

6. Binary ← → Base64 conversions

We would like to expose the routines in the current LDIF library to our developers in the form of a dynamic library as opposed to a static library. This requires minimal or no changes to the existing OpenLDAP source.

The existing routines to expose are:

ldif_read_record() - Reads the next record from a FILE stream into a buffer, with newline characters. Buffer memory is allocated by the library.

ldif_getline() - Get the next "line" from a buffer with newline characters. Continuation lines are combined into one big line.

ldif_parse_line() - Parses a big line consisting of attribute: value into its components. If the value is base64 encoded (indicated by a double colon), the base64 value is decoded.

ldif_is_not_printable() - Returns TRUE if the input string must be base64 encoded in an LDIF file.

ldif_put() - Puts a line consisting of an attribute and value to a buffer in LDIF format. A "type" argument controls how the encoding is done. The buffer is allocated by the library.

ldif_sput() - Same as ldif_put() except the buffer is passed in by the application and is assumed to be large enough to contain the data.

6.1 Freeing memory.

If the LDIF functions are moved to the main LDAP library, memory allocated by LDIF functions could be freed with ldap_memfree(). However, if the functions were ever moved to a separate dynamic library, some platforms would require they be freed by a function in that library. To keep the options open, we propose adding this routine. Memory allocated by any "ldif_" routines should be freed with this one.

```
void ldif_memfree(void *p)
```