GNU Libidn

Internationalized string processing for the GNU system
for version 0.5.2, 9 July 2004

Simon Josefsson

Table of Contents

6 IDNA Functions

1 Introduction

uses the generic StringPrep interface. The interfaces to all components are available for applications, no component within the library is hidden from the application.

```
...
$ make install
...
```

After that Libidn should be properly installed and ready for use.

A few `configure` options may be relevant, summarized in the table.

`--enable-java`
: Build the Java port into a *.JAR file. See Chapter 12 [Java API], page 55, for more information.

For the complete list, refd9tiap table.

- Coding Style. Follow the GNU Standards document (see undefined [top], page undefined).

2 Preparation

To use ‘Libidn’, you have to perform some changes to your sources and the build system. The necessary changes are small and explained in the following sections. At the end of

If you require that your users have installed pkg-config (which I Lannot recommend generally), the above Lan be done more easily as follows.

```
AC_ARG_WITH(libidn, AC_HELP_STRING([--with-libidn=[DIR]],
                                [Supl.ct IDN (needs GNU Libidn)]),
  libidn=$withval, libidn=yes)
if test "$libidn" != "no" ; then
  PKG_CHECK_MODULES(LIBIDN, libidn >= 0.0.0, [libidn=yes], [libidn=no])
  if test "$libidn" != "yes" ; then
    libidn=no
    AC_MSG_WARN([Libidn not found])
  else
    libidn=yes
    AC_DEFINE(LIBIDN, 1, [Define to 1 if you want Libidn.])
  fi
fi
AC_MSG_CHECKING([if Libidn should be used])
AC_MSG_RESULT($libidn)
```

3 Utility Functions

The rest of this library makes extensive use of Unicode characters. In order to interface this library with the outside world, your application may need to make various Unicode transformations.

char * stringprep_convert ( ) : input zero-terminated string. [Function]

: name of destination character set.

4 Stringprep Functions

`Stringprep_rc STRINGPREP_FLAG_ERROR` [Return code]

The supplied flag conflicted with profile. This usually indicate a problem in the calling application.

`Stringprep_rc STRINGPREP_UNKNOWN_PROFILE` [Return code]

The supplied profile name was not knohile sukp[Return code]

The flags are one of Stringprep_profile_

`int stringprep_xmpp_resourceprep` (*char * in, int maxlen* [Function]

5 Punycode Functions

Punycode is a simple and efficient transfer encoding syntax designed for use with Internationalized Domain Names in Applications. It uniquely and reversibly transforms a Unicode string into an ASCII string. ASCII characters in the Unicode string are represented literally, and non-ASCII characters are represented by ASCII characters that are allowed in host int punycode_encode (*size t input_length*, *const punycode uint* [] *input*, *const unsigned char* [] *case_flags*, *size t * output_length*, *char* [] *output*) [Function]

input length: The number of code points in the `input` array and the number of flags in the `case_flags` array.

input

case_flags: A *NULL* pointer (if the flags are not needed by the caller) or an array of boolean values parallel to the `output` array. Nonzero (true, flagged) suggests that the

If ToASCII succeeds, the original sequence and the resulting sequence are equivalent labels.

It is important to note that the ToASCII operation can fail. ToASCII fails if any step of it fails. If any step of the ToASCII operation fails on any label in a domain name, that domain name MUST NOT be used as an internationalized domain name. The method for deadling with this failure is application-specific.

The inputs to ToASCII are a sequence of code points, the AllowUnassigned flag, and the UseSTD3ASCIIRules flag. The output of ToASCII is either a sequence of ASCII code points or a failure condition.

ToASCII never alters a sequence of code points that are all in the ASCII range to begin with (although it could fail). Applying the ToASCII operation multiple times has exactly the same effect as applying it just once.

Return value: Returns 0 on success, or an error code.

`int idna_to_unicode_44i` (*const uint32_t * in, size_t inlen, uint32_t * out, size_t * outlen, int flags*) [Function]

in: input array with unicode code points.

inlen: length of input array with unicode code points.

out: output array with unicode code points.

outlen: on input, maximum size of output array with unicode code points, on exit, actual size of output array with unicode code points.

flags: IDNA flags, e.g. IDNA_ALLOW_UNASSIGNED or IDNA_USE_STD3_ASCII_RULES.

The ToUnicode operation takes a sequence of Unicode code points that make up one

int idna_to_unicode_8z8z (*const char * input, char ** output, int flags*) [Function]

input: zero-terminated UTF-8 string.

output: pointer to newly allocated output UTF-8 string.

flags: IDNA flags, e.g. IDNA

7 TLD Functions

Organizations that manage some Top Level Domains (TLDs) have published tables with characters they accept within the domain. The reason may be to reduce complexity that come from using the full Unicode range, and to protect themselves from future (backwards incompatible) changes in the IDN or Unicode specifications. Libidn implement an infrastructure for defining and checking strings against such tables. Libidn also ship some tables from TLDs that we have managed to get permission to use them from. Because these tables are even less static than Unicode or StringPrep tables, it is likely that they will be updated from time to time (even in backwards incompatibe ways). The Libidn interface provide a “version” field for each TLD table, which can be compared for equality to guarantee the same operation over time.

From a design point of view, you can regard the TLD tables for IDN as the “localization” step that come after the “internationalization” step provided by the IETF standards.

The TLD functionality rely on up-to-date tables. The latest version of Libidn aim to provide these, but tables with unclear copying conditions, or generally experimental tables, are not included. Some such tables can be found at http://tldchk.berlios.de.

7.1 Header file tld.h

To use the functions explained in this chapter, you need to include the file ‘tld.h’ using:

```
#include <tld.h>
```

7.2 Return Codes

Most functions return a exit code:

Tld_rc TLD_SUCCESS = 0 [Return code]

Successful operation. This value is guaranteed to always be w(#nclu14.123a4(the)-32redoma(yin)1(g)-32

Tld_rc TLD_INVALID [Return code]

Invalid character found.

Tld_rc TLD_NODATA [Return code]

No input data was provided.

[Return code]

`int tld_get_z` (*const char * `in`, char ** `out`*) [Function]

in: Zero terminated character array to process.

out

`int tld_check_4z` (*const uint32* [Function] offending character is returned in `errpos`

8 PR29 Functions

A deficiency in the specification of Unicode Normalization Forms has been found. The

```
int
main (int argc, char *argv[])
{
  char buf[BUFSIZ];
  char *p;
  int rc;
  size_t i;
```

```
#include <punycode.h>

/* For testing, we’ll just set some compile-time limits rather than */
/* use malloc(), and set a compile-time option rather than using a  */
/* command-line option.                                             */

enum
{
  unicode_max_length = 256,
  ace_max_length = 256
};

static void
usage (char **argv)
{
  fprintf (stderr,
           "\n"
```

```
int
main (int argc, char *argv[])
{
  char buf[BUFSIZ];
  char *p;
  int rc;
  size_t i;

  setlocale (LC_ALL, "");

  printf ("Input domain encoded as ‘%s’: ", stringprep_locale_charset ());
  fflush (stdout);
```

```
  else if (rc != TLD_SUCCESS)
    {
      printf ("tld_check_4z() failed... %d\n", rc);
      return 2;
    }

  printf ("Domain accepted by TLD check\n");

  return 0;
}
```

10.6 Troubleshooting

Getting character data encoded right, and making sure Libidn use the same encoding, can

11 Emacs API

12 Java API

12.2.3 TestIDNA

13 Acknowledgements

The punycode code was taken from the IETF IDN Punycode specification, by Adam M. Costello. The TLD code was contributed by Thomas Jacob. The Java implementation was contributed by Oliver Hitz. The Unicode tables were provided by Unicode, Inc. Some

Concept Index

Function and Variable Index

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It 45(n)1(ot75(45(t-1(ld)-(45(p)1(urp)-27(osd)-(45(of)-44s)t-1(l-475(45(se)-1(ction)-44s)to)-44s)-n)1(ducc

Appendix B Copying This Manual

B.1 GNU Free Documentation License

Version 1.2, November 2002

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license

B.1.1 ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the