libidn Reference Manual

0.6.7

Generated by Doxygen 1.4.7

Wed Sep 13 10:20:30 2006

Contents

Chapter 1

GNU Internationalized Domain Name Library

1.1 Introduction

GNU Libidn is an implementation of the Stringprep, Punycode and IDNA specifications defined by the IETF Internationalized Domain Names (IDN) working group, used for internationalized domain names. The package is available under the GNU Lesser General Public License.

The library contains a generic Stringprep implementation that does Unicode 3.2 NFKC normalization, mapping and prohibitation of characters, and bidirectional character handling. Profiles for Nameprep, i-SCSI, SASL and XMPP are included. Punycode and ASCII Compatible Encoding (ACE) via IDNA are supported. A mechanism to define Top-Level Domain (TLD) specific validation tables, and to compare strings against those tables, is included. Default tables for some TLDs are also included.

The Stringprep API consists of two main functions, one for converting data from the system's native representation into UTF-8, and one function to perform the Stringprep processing. Adding a new Stringprep profile for your application within the API is straightforward. The Punycode API consists of one encoding function and one decoding function. The IDNA API consists of the ToASCII and ToUnicode functions, as well as an high-level interface for converting entire domain names to and from the ACE encoded form. The TLD API consists of one set of functions to extract the TLD name from a domain string, one set of functions to locate the proper TLD table to use based on the TLD name, and core functions to validate a string against a TLD table, and some utility wrappers to perform all the steps in one call.

The library is used by, e.g., GNU SASL and Shishi to process user names and passwords. Libidn can be built into GNU Libc to enable a new system-wide getaddrinfo() flag for IDN processing.

Libidn is developed for the GNU/Linux system, but runs on over 20 Unix platforms (including Solaris, IRIX, AIX, and Tru64) and Windows. Libidn is written in C and (parts of) the API is accessible from C, C++, Emacs Lisp, Python and Java.

The project web page:

http://www.gnu.org/software/libidn/

The software archive:

ftp://alpha.gnu.org/pub/gnu/libidn/

For more information see:

http://www.ietf.org/html.charters/idn-charter.html

http://www.ietf.org/rfc/rfc3454.txt (stringprep specification)

http://www.ietf.org/rfc/rfc3490.txt (idna specification)

http://www.ietf.org/rfc/rfc3491.txt (nameprep specification)

http://www.ietf.org/rfc/rfc3492.txt (punycode specification)

http://www.ietf.org/internet-drafts/draft-ietf-ips-iscsi-string-prep-04.txt

http://www.ietf.org/internet-drafts/draft-ietf-krb-wg-utf8-profile-01.txt

http://www.ietf.org/internet-drafts/draft-ietf-sasl-anon-00.txt

http://www.ietf.org/internet-drafts/draft-ietf-sasl-saslprep-00.txt

http://www.ietf.org/internet-drafts/draft-ietf-xmpp-nodeprep-01.txt

http://www.ietf.org/internet-drafts/draft-ietf-xmpp-resourceprep-01.txt

Further information and paid contract development:

Simon Josefsson <simon@josefsson.org>

1.2 Examples

```
/* example.c --- Example code showing how to use stringprep().
 * Copyright (C) 2002, 2003, 2004  Simon Josefsson
 *
 * This file is part of GNU Libidn.
 *
 * GNU Libidn is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * GNU Libidn is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with GNU Libidn; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <locale.h>             /* setlocale() */
#include <stringprep.h>

/*
 * Compiling using libtool and pkg-config is recommended:
 *
 * $ libtool cc -o example example.c `pkg-config --cflags --libs libidn`
 * $ ./example
 * Input string encoded as 'ISO-8859-1': ł
 * Before locale2utf8 (length 2): aa 0a
 * Before stringprep (length 3): c2 aa 0a
 * After stringprep (length 2): 61 0a
 * $
 *
 */
```

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

  setlocale (LC_ALL, "");

  printf ("Input string encoded as `%s': ", stringprep_locale_charset ());
  fflush (stdout);
  fgets (buf, BUFSIZ, stdin);

  printf ("Before locale2utf8 (length %d): ", strlen (buf));
  for (i = 0; i < strlen (buf); i++)
    printf ("%02x ", buf[i] & 0xFF);
  printf ("\n");

  p = stringprep_locale_to_utf8 (buf);
  if (p)
    {
      strcpy (buf, p);
      free (p);
    }
  else
    printf ("Could not convert string to UTF-8, continuing anyway...\n");

  printf ("Before stringprep (length %d): ", strlen (buf));
  for (i = 0; i < strlen (buf); i++)
    printf ("%02x ", buf[i] & 0xFF);
  printf ("\n");

  rc = stringprep (buf, BUFSIZ, 0, stringprep_nameprep);
  if (rc != STRINGPREP_OK)
    printf ("Stringprep failed (%d): %s\n", rc, stringprep_strerror (rc));
  else
    {
      printf ("After stringprep (length %d): ", strlen (buf));
      for (i = 0; i < strlen (buf); i++)
        printf ("%02x ", buf[i] & 0xFF);
      printf ("\n");
    }

  return 0;
}


/* example3.c --- Example ToASCII() code showing how to use Libidn.
 * Copyright (C) 2002, 2003, 2004, 2006  Simon Josefsson
 *
 * This file is part of GNU Libidn.
 *
 * GNU Libidn is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * GNU Libidn is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with GNU Libidn; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA
 *
 */
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <locale.h>             /* setlocale() */
#include <stringprep.h>         /* stringprep_locale_charset() */
#include <idna.h>               /* idna_to_ascii_lz() */

/*
 * Compiling using libtool and pkg-config is recommended:
 *
 * $ libtool cc -o example3 example3.c `pkg-config --cflags --libs libidn`
 * $ ./example3
 * Input domain encoded as `ISO-8859-1': www.räksmörgåsł.example
 * Read string (length 23): 77 77 77 2e 72 e4 6b 73 6d f6 72 67 e5 73 aa 2e 65 78 61 6d 70 6c 65
 * ACE label (length 33): 'www.xn--rksmrgsa-0zap8p.example'
 * 77 77 77 2e 78 6e 2d 2d 72 6b 73 6d 72 67 73 61 2d 30 7a 61 70 38 70 2e 65 78 61 6d 70 6c 65
 * $
 *
 */

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);
  fgets (buf, BUFSIZ, stdin);
  buf[strlen (buf) - 1] = '\0';

  printf ("Read string (length %d): ", strlen (buf));
  for (i = 0; i < strlen (buf); i++)
    printf ("%02x ", buf[i] & 0xFF);
  printf ("\n");

  rc = idna_to_ascii_lz (buf, &p, 0);
  if (rc != IDNA_SUCCESS)
    {
      printf ("ToASCII() failed (%d): %s\n", rc, idna_strerror (rc));
      exit (1);
    }

  printf ("ACE label (length %d): '%s'\n", strlen (p), p);
  for (i = 0; i < strlen (p); i++)
    printf ("%02x ", p[i] & 0xFF);
  printf ("\n");

  free (p);

  return 0;
}


/* example4.c --- Example ToUnicode() code showing how to use Libidn.
 * Copyright (C) 2002, 2003, 2004, 2006  Simon Josefsson
 *
 * This file is part of GNU Libidn.
 *
 * GNU Libidn is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
```

```
 *
 * GNU Libidn is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with GNU Libidn; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <locale.h>             /* setlocale() */
#include <stringprep.h>         /* stringprep_locale_charset() */
#include <idna.h>               /* idna_to_unicode_lzlz() */

/*
 * Compiling using libtool and pkg-config is recommended:
 *
 * $ libtool cc -o example4 example4.c `pkg-config --cflags --libs libidn`
 * $ ./example4
 * Input domain encoded as `ISO-8859-1': www.xn--rksmrgsa-0zap8p.example
 * Read string (length 33): 77 77 77 2e 78 6e 2d 2d 72 6b 73 6d 72 67 73 61 2d 30 7a 61 70 38 70 2e 65 78
 * ACE label (length 23): 'www.räksmörgåsa.example'
 * 77 77 77 2e 72 e4 6b 73 6d f6 72 67 e5 73 61 2e 65 78 61 6d 70 6c 65
 * $
 *
 */

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);
  fgets (buf, BUFSIZ, stdin);
  buf[strlen (buf) - 1] = '\0';

  printf ("Read string (length %d): ", strlen (buf));
  for (i = 0; i < strlen (buf); i++)
    printf ("%02x ", buf[i] & 0xFF);
  printf ("\n");

  rc = idna_to_unicode_lzlz (buf, &p, 0);
  if (rc != IDNA_SUCCESS)
    {
      printf ("ToUnicode() failed (%d): %s\n", rc, idna_strerror (rc));
      exit (1);
    }

  printf ("ACE label (length %d): '%s'\n", strlen (p), p);
  for (i = 0; i < strlen (p); i++)
    printf ("%02x ", p[i] & 0xFF);
  printf ("\n");

  free (p);

  return 0;
```

```
}


/* example5.c --- Example TLD checking.
 * Copyright (C) 2004  Simon Josefsson
 *
 * This file is part of GNU Libidn.
 *
 * GNU Libidn is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * GNU Libidn is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with GNU Libidn; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* Get stringprep_locale_charset, etc. */
#include <stringprep.h>

/* Get idna_to_ascii_8z, etc. */
#include <idna.h>

/* Get tld_check_4z. */
#include <tld.h>

/*
 * Compiling using libtool and pkg-config is recommended:
 *
 * $ libtool cc -o example5 example5.c `pkg-config --cflags --libs libidn`
 * $ ./example5
 * Input domain encoded as `UTF-8': fooÃ§.no
 * Read string (length 8): 66 6f 6f c3 9f 2e 6e 6f
 * ToASCII string (length 8): fooss.no
 * ToUnicode string: U+0066 U+006f U+006f U+0073 U+0073 U+002e U+006e U+006f
 * Domain accepted by TLD check
 *
 * $ ./example5
 * Input domain encoded as `UTF-8': grâĆňâĆňn.no
 * Read string (length 12): 67 72 e2 82 ac e2 82 ac 6e 2e 6e 6f
 * ToASCII string (length 16): xn--grn-l50aa.no
 * ToUnicode string: U+0067 U+0072 U+20ac U+20ac U+006e U+002e U+006e U+006f
 * Domain rejected by TLD check, Unicode position 2
 *
 */

int
main (int argc, char *argv[])
{
  char buf[BUFSIZ];
  char *p;
  uint32_t *r;
  int rc;
  size_t errpos, i;

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

```
  fgets (buf, BUFSIZ, stdin);
  buf[strlen (buf) - 1] = '\0';

  printf ("Read string (length %d): ", strlen (buf));
  for (i = 0; i < strlen (buf); i++)
    printf ("%02x ", buf[i] & 0xFF);
  printf ("\n");

  p = stringprep_locale_to_utf8 (buf);
  if (p)
    {
      strcpy (buf, p);
      free (p);
    }
  else
    printf ("Could not convert string to UTF-8, continuing anyway...\n");

  rc = idna_to_ascii_8z (buf, &p, 0);
  if (rc != IDNA_SUCCESS)
    {
      printf ("idna_to_ascii_8z failed (%d): %s\n", rc, idna_strerror (rc));
      return 2;
    }

  printf ("ToASCII string (length %d): %s\n", strlen (p), p);

  rc = idna_to_unicode_8z4z (p, &r, 0);
  free (p);
  if (rc != IDNA_SUCCESS)
    {
      printf ("idna_to_unicode_8z4z failed (%d): %s\n",
              rc, idna_strerror (rc));
      return 2;
    }

  printf ("ToUnicode string: ");
  for (i = 0; r[i]; i++)
    printf ("U+%04x ", r[i]);
  printf ("\n");

  rc = tld_check_4z (r, &errpos, NULL);
  free (r);
  if (rc == TLD_INVALID)
    {
      printf ("Domain rejected by TLD check, Unicode position %d\n", errpos);
      return 1;
    }
  else if (rc != TLD_SUCCESS)
    {
      printf ("tld_check_4z() failed (%d): %s\n", rc, tld_strerror (rc));
      return 2;
    }

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

  return 0;
}
```

Chapter 2

libidn Data Structure Index

2.1 libidn Data Structures

Here are the data structures with brief descriptions:

Chapter 3

libidn File Index

3.1 libidn File List

Here is a list of all files with brief descriptions:

Chapter 4

libidn Data Structure Documentation

4.1 decomposition Struct Reference

`#include <gunidecomp.h>`

Data Fields

- gunichar ch
- guint16 canon_offset
- guint16 compat_offset

4.1.1 Detailed Description

Definition at line 1825 of file gunidecomp.h.

4.1.2 Field Documentation

4.1.2.1 guint16 decomposition::canon_offset

Definition at line 1828 of file gunidecomp.h.

4.1.2.2 gunichar decomposition::ch

Definition at line 1827 of file gunidecomp.h.

4.1.2.3 guint16 decomposition::compat_offset

Definition at line 1829 of file gunidecomp.h.

The documentation for this struct was generated from the following file:

- gunidecomp.h

4.2 Stringprep_profiles Struct Reference

`#include <stringprep.h>`

Data Fields

- const char * name
- const Stringprep_profile * tables

4.2.1 Detailed Description

Definition at line 95 of file stringprep.h.

4.2.2 Field Documentation

4.2.2.1 const char* Stringprep_profiles::name

Definition at line 97 of file stringprep.h.

Referenced by stringprep_profile().

4.2.2.2 const Stringprep_profile* Stringprep_profiles::tables

Definition at line 98 of file stringprep.h.

Referenced by stringprep_profile(), and tld_get_table().

The documentation for this struct was generated from the following file:

- stringprep.h

4.3 Stringprep_table Struct Reference

`#include <stringprep.h>`

Data Fields

- Stringprep_profile_steps operation
- Stringprep_profile_flags flags
- const Stringprep_table_element * table

4.3.1 Detailed Description

Definition at line 87 of file stringprep.h.

4.3.2 Field Documentation

4.3.2.1 Stringprep_profile_flags Stringprep_table::flags

Definition at line 90 of file stringprep.h.

4.3.2.2 Stringprep_profile_steps Stringprep_table::operation

Definition at line 89 of file stringprep.h.

Referenced by stringprep_4i().

4.3.2.3 const Stringprep_table_element* Stringprep_table::table

Definition at line 91 of file stringprep.h.

The documentation for this struct was generated from the following file:

- stringprep.h

4.4 Stringprep_table_element Struct Reference

`#include <stringprep.h>`

Data Fields

- uint32_t start
- uint32_t end
- uint32_t map [STRINGPREP_MAX_MAP_CHARS]

4.4.1 Detailed Description

Definition at line 79 of file stringprep.h.

4.4.2 Field Documentation

4.4.2.1 uint32_t Stringprep_table_element::end

Definition at line 82 of file stringprep.h.

4.4.2.2 uint32_t Stringprep_table_element::map[STRINGPREP_MAX_MAP_CHARS]

Definition at line 83 of file stringprep.h.

4.4.2.3 uint32_t Stringprep_table_element::start

Definition at line 81 of file stringprep.h.

The documentation for this struct was generated from the following file:

- stringprep.h

4.5 Tld_table Struct Reference

`#include <tld.h>`

Data Fields

- const char * name
- const char * version
- size_t nvalid
- const Tld_table_element * valid

4.5.1 Detailed Description

Definition at line 48 of file tld.h.

4.5.2 Field Documentation

4.5.2.1 const char* Tld_table::name

Definition at line 50 of file tld.h.

4.5.2.2 size_t Tld_table::nvalid

Definition at line 52 of file tld.h.

4.5.2.3 const Tld_table_element* Tld_table::valid

Definition at line 53 of file tld.h.

4.5.2.4 const char* Tld_table::version

Definition at line 51 of file tld.h.

The documentation for this struct was generated from the following file:

- tld.h

4.6 Tld_table_element Struct Reference

`#include <tld.h>`

Data Fields

- uint32_t start
- uint32_t end

4.6.1 Detailed Description

Definition at line 40 of file tld.h.

4.6.2 Field Documentation

4.6.2.1 uint32_t Tld_table_element::end

Definition at line 43 of file tld.h.

4.6.2.2 uint32_t Tld_table_element::start

Definition at line 42 of file tld.h.

The documentation for this struct was generated from the following file:

- tld.h

Chapter 5

libidn File Documentation

5.1 gunibreak.h File Reference

Defines

- #define G_UNICODE_DATA_VERSION "3.2"
- #define G_UNICODE_LAST_CHAR 0x10FFFF
- #define G_UNICODE_MAX_TABLE_INDEX 10000
- #define G_UNICODE_LAST_CHAR_PART1 0x2FAFF

5.1.1 Define Documentation

5.1.1.1 #define G_UNICODE_DATA_VERSION "3.2"

Definition at line 7 of file gunibreak.h.

5.1.1.2 #define G_UNICODE_LAST_CHAR 0x10FFFF

Definition at line 9 of file gunibreak.h.

5.1.1.3 #define G_UNICODE_LAST_CHAR_PART1 0x2FAFF

Definition at line 14 of file gunibreak.h.

5.1.1.4 #define G_UNICODE_MAX_TABLE_INDEX 10000

Definition at line 11 of file gunibreak.h.

5.2 gunicomp.h File Reference

Defines

- #define COMPOSE_FIRST_START 1
- #define COMPOSE_FIRST_SINGLE_START 147
- #define COMPOSE_SECOND_START 357
- #define COMPOSE_SECOND_SINGLE_START 388
- #define COMPOSE_TABLE_LAST 48

5.2.1 Define Documentation

5.2.1.1 #define COMPOSE_FIRST_SINGLE_START 147

Definition at line 5 of file gunicomp.h.

5.2.1.2 #define COMPOSE_FIRST_START 1

Definition at line 4 of file gunicomp.h.

5.2.1.3 #define COMPOSE_SECOND_SINGLE_START 388

Definition at line 7 of file gunicomp.h.

5.2.1.4 #define COMPOSE_SECOND_START 357

Definition at line 6 of file gunicomp.h.

5.2.1.5 #define COMPOSE_TABLE_LAST 48

Definition at line 9 of file gunicomp.h.

5.3 gunidecomp.h File Reference

Data Structures

- struct decomposition

Defines

- #define G_UNICODE_LAST_CHAR 0x10ffff
- #define G_UNICODE_MAX_TABLE_INDEX (0x110000 / 256)
- #define G_UNICODE_LAST_CHAR_PART1 0x2FAFF
- #define G_UNICODE_LAST_PAGE_PART1 762
- #define G_UNICODE_NOT_PRESENT_OFFSET 65535

5.3.1 Define Documentation

5.3.1.1 #define G_UNICODE_LAST_CHAR 0x10ffff

Definition at line 7 of file gunidecomp.h.

5.3.1.2 #define G_UNICODE_LAST_CHAR_PART1 0x2FAFF

Definition at line 11 of file gunidecomp.h.

5.3.1.3 #define G_UNICODE_LAST_PAGE_PART1 762

Definition at line 13 of file gunidecomp.h.

5.3.1.4 #define G_UNICODE_MAX_TABLE_INDEX (0x110000 / 256)

Definition at line 9 of file gunidecomp.h.

5.3.1.5 #define G_UNICODE_NOT_PRESENT_OFFSET 65535

Definition at line 15 of file gunidecomp.h.

5.4 iconvme.c File Reference

```
#include "iconvme.h"
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include "strdup.h"
```

Defines

- #define SIZE_MAX ((size_t) -1)

Functions

- char * iconv_string (const char *str, const char *from_codeset, const char *to_codeset)

5.4.1 Define Documentation

5.4.1.1 #define SIZE_MAX ((size_t) -1)

Definition at line 49 of file iconvme.c.

Referenced by iconv_string().

5.4.2 Function Documentation

5.4.2.1 char* iconv_string (const char * *str*, const char * *from_codeset*, const char * *to_codeset*)

Definition at line 60 of file iconvme.c.

References SIZE_MAX.

Referenced by stringprep_convert().

5.5 iconvme.h File Reference

Functions

- char * iconv_string (const char *string, const char *from_code, const char *to_code)

5.5.1 Function Documentation

5.5.1.1 char* iconv_string (const char * *string*, const char * *from_code*, const char * *to_code*)

Definition at line 60 of file iconvme.c.

References SIZE_MAX.

Referenced by stringprep_convert().

5.6 idn-free.c File Reference

```
#include "idn-free.h"
#include <stdlib.h>
```

Functions

- void idn_free (void *ptr)

5.6.1 Function Documentation

5.6.1.1 void idn_free (void * *ptr*)

Definition at line 30 of file idn-free.c.

5.7 idn-free.h File Reference

Functions

- void idn_free (void *ptr)

5.7.1 Function Documentation

5.7.1.1 void idn_free (void * *ptr*)

Definition at line 30 of file idn-free.c.

5.8 idn-int.h File Reference

```
#include "///usr/include/stdint.h"
#include <sys/types.h>
#include <limits.h>
#include <inttypes.h>
#include <sys/bitypes.h>
#include <stdio.h>
#include <time.h>
#include <wchar.h>
```

Defines

- #define _GL_JUST_INCLUDE_ABSOLUTE_INTTYPES_H
- #define _STDINT_MIN(signed, bits, zero) ((signed) ? (- ((zero) + 1) << ((bits) ? (bits) - 1 : 0)) : (zero))
- #define _STDINT_MAX(signed, bits, zero)
- #define int8_t signed char
- #define uint8_t unsigned char
- #define int16_t short int
- #define uint16_t unsigned short int
- #define int32_t int
- #define uint32_t unsigned int
- #define int64_t long long int
- #define uint64_t unsigned long long int
- #define _UINT8_T
- #define _UINT32_T
- #define _UINT64_T
- #define int_least8_t int8_t
- #define uint_least8_t uint8_t
- #define int_least16_t int16_t
- #define uint_least16_t uint16_t
- #define int_least32_t int32_t
- #define uint_least32_t uint32_t
- #define int_least64_t int64_t
- #define uint_least64_t uint64_t
- #define int_fast8_t long int
- #define uint_fast8_t unsigned int_fast8_t
- #define int_fast16_t long int
- #define uint_fast16_t unsigned int_fast16_t
- #define int_fast32_t long int
- #define uint_fast32_t unsigned int_fast32_t
- #define int_fast64_t int64_t
- #define uint_fast64_t uint64_t
- #define intptr_t long int
- #define uintptr_t unsigned long int
- #define intmax_t int64_t

- #define uintmax_t uint64_t
- #define INT8_MIN (~ INT8_MAX)
- #define INT8_MAX 127
- #define UINT8_MAX 255
- #define INT16_MIN (~ INT16_MAX)
- #define INT16_MAX 32767
- #define UINT16_MAX 65535
- #define INT32_MIN (~ INT32_MAX)
- #define INT32_MAX 2147483647
- #define UINT32_MAX 4294967295U
- #define INT64_MIN (~ INT64_MAX)
- #define INT64_MAX INTMAX_C (9223372036854775807)
- #define UINT64_MAX UINTMAX_C (18446744073709551615)
- #define INT_LEAST8_MIN INT8_MIN
- #define INT_LEAST8_MAX INT8_MAX
- #define UINT_LEAST8_MAX UINT8_MAX
- #define INT_LEAST16_MIN INT16_MIN
- #define INT_LEAST16_MAX INT16_MAX
- #define UINT_LEAST16_MAX UINT16_MAX
- #define INT_LEAST32_MIN INT32_MIN
- #define INT_LEAST32_MAX INT32_MAX
- #define UINT_LEAST32_MAX UINT32_MAX
- #define INT_LEAST64_MIN INT64_MIN
- #define INT_LEAST64_MAX INT64_MAX
- #define UINT_LEAST64_MAX UINT64_MAX
- #define INT_FAST8_MIN LONG_MIN
- #define INT_FAST8_MAX LONG_MAX
- #define UINT_FAST8_MAX ULONG_MAX
- #define INT_FAST16_MIN LONG_MIN
- #define INT_FAST16_MAX LONG_MAX
- #define UINT_FAST16_MAX ULONG_MAX
- #define INT_FAST32_MIN LONG_MIN
- #define INT_FAST32_MAX LONG_MAX
- #define UINT_FAST32_MAX ULONG_MAX
- #define INT_FAST64_MIN INT64_MIN
- #define INT_FAST64_MAX INT64_MAX
- #define UINT_FAST64_MAX UINT64_MAX
- #define INTPTR_MIN LONG_MIN
- #define INTPTR_MAX LONG_MAX
- #define UINTPTR_MAX ULONG_MAX
- #define INTMAX_MIN (~ INTMAX_MAX)
- #define INTMAX_MAX INT64_MAX
- #define UINTMAX_MAX UINT64_MAX
- #define PTRDIFF_MIN _STDINT_MIN (1, 32, 0)
- #define PTRDIFF_MAX _STDINT_MAX (1, 32, 0)
- #define SIG_ATOMIC_MIN
- #define SIG_ATOMIC_MAX
- #define SIZE_MAX _STDINT_MAX (0, 32, 0u)
- #define WCHAR_MIN _STDINT_MIN (1, 32, 0)
- #define WCHAR_MAX _STDINT_MAX (1, 32, 0)

- #define WINT_MIN _STDINT_MIN (0, 32, 0u)
- #define WINT_MAX _STDINT_MAX (0, 32, 0u)
- #define INT8_C(x) x
- #define UINT8_C(x) x
- #define INT16_C(x) x
- #define UINT16_C(x) x
- #define INT32_C(x) x
- #define UINT32_C(x) x ## U
- #define INT64_C(x) x##LL
- #define UINT64_C(x) x##ULL
- #define INTMAX_C(x) INT64_C(x)
- #define UINTMAX_C(x) UINT64_C(x)

5.8.1 Define Documentation

5.8.1.1 #define _GL_JUST_INCLUDE_ABSOLUTE_INTTYPES_H

Definition at line 65 of file idn-int.h.

5.8.1.2 #define _STDINT_MAX(signed, bits, zero)

Value:

```
((signed) \
   ? ~ _STDINT_MIN (signed, bits, zero) \
   : ((((zero) + 1) << ((bits) ? (bits) - 1 : 0)) - 1) * 2 + 1)
```

Definition at line 101 of file idn-int.h.

5.8.1.3 #define _STDINT_MIN(signed, bits, zero) ((signed) ? (- ((zero) + 1) << ((bits) ? (bits) - 1 : 0)) : (zero))

Definition at line 98 of file idn-int.h.

5.8.1.4 #define _UINT32_T

Definition at line 141 of file idn-int.h.

5.8.1.5 #define _UINT64_T

Definition at line 142 of file idn-int.h.

5.8.1.6 #define _UINT8_T

Definition at line 140 of file idn-int.h.

5.8.1.7 #define INT16_C(x) x

Definition at line 418 of file idn-int.h.

5.8.1.8 #define INT16_MAX 32767

Definition at line 244 of file idn-int.h.

5.8.1.9 #define INT16_MIN (~ INT16_MAX)

Definition at line 243 of file idn-int.h.

5.8.1.10 #define int16_t short int

Definition at line 118 of file idn-int.h.

5.8.1.11 #define INT32_C(x) x

Definition at line 423 of file idn-int.h.

5.8.1.12 #define INT32_MAX 2147483647

Definition at line 251 of file idn-int.h.

5.8.1.13 #define INT32_MIN (~ INT32_MAX)

Definition at line 250 of file idn-int.h.

5.8.1.14 #define int32_t int

Definition at line 123 of file idn-int.h.

5.8.1.15 #define INT64_C(x) x##LL

Definition at line 435 of file idn-int.h.

5.8.1.16 #define INT64_MAX INTMAX_C (9223372036854775807)

Definition at line 259 of file idn-int.h.

5.8.1.17 #define INT64_MIN (~ INT64_MAX)

Definition at line 258 of file idn-int.h.

5.8.1.18 #define int64_t long long int

Definition at line 135 of file idn-int.h.

5.8.1.19 #define INT8_C(x) x

Definition at line 413 of file idn-int.h.

5.8.1.20 #define INT8_MAX 127

Definition at line 237 of file idn-int.h.

5.8.1.21 #define INT8_MIN (~ INT8_MAX)

Definition at line 236 of file idn-int.h.

5.8.1.22 #define int8_t signed char

Definition at line 113 of file idn-int.h.

5.8.1.23 #define INT_FAST16_MAX LONG_MAX

Definition at line 316 of file idn-int.h.

5.8.1.24 #define INT_FAST16_MIN LONG_MIN

Definition at line 315 of file idn-int.h.

5.8.1.25 #define int_fast16_t long int

Definition at line 190 of file idn-int.h.

5.8.1.26 #define INT_FAST32_MAX LONG_MAX

Definition at line 323 of file idn-int.h.

5.8.1.27 #define INT_FAST32_MIN LONG_MIN

Definition at line 322 of file idn-int.h.

5.8.1.28 #define int_fast32_t long int

Definition at line 192 of file idn-int.h.

5.8.1.29 #define INT_FAST64_MAX INT64_MAX

Definition at line 331 of file idn-int.h.

5.8.1.30 #define INT_FAST64_MIN INT64_MIN

Definition at line 330 of file idn-int.h.

5.8.1.31 #define int_fast64_t int64_t

Definition at line 195 of file idn-int.h.

5.8.1.32 #define INT_FAST8_MAX LONG_MAX

Definition at line 309 of file idn-int.h.

5.8.1.33 #define INT_FAST8_MIN LONG_MIN

Definition at line 308 of file idn-int.h.

5.8.1.34 #define int_fast8_t long int

Definition at line 188 of file idn-int.h.

5.8.1.35 #define INT_LEAST16_MAX INT16_MAX

Definition at line 280 of file idn-int.h.

5.8.1.36 #define INT_LEAST16_MIN INT16_MIN

Definition at line 279 of file idn-int.h.

5.8.1.37 #define int_least16_t int16_t

Definition at line 161 of file idn-int.h.

5.8.1.38 #define INT_LEAST32_MAX INT32_MAX

Definition at line 287 of file idn-int.h.

5.8.1.39 #define INT_LEAST32_MIN INT32_MIN

Definition at line 286 of file idn-int.h.

5.8.1.40 #define int_least32_t int32_t

Definition at line 163 of file idn-int.h.

5.8.1.41 #define INT_LEAST64_MAX INT64_MAX

Definition at line 295 of file idn-int.h.

5.8.1.42 #define INT_LEAST64_MIN INT64_MIN

Definition at line 294 of file idn-int.h.

5.8.1.43 #define int_least64_t int64_t

Definition at line 166 of file idn-int.h.

5.8.1.44 #define INT_LEAST8_MAX INT8_MAX

Definition at line 273 of file idn-int.h.

5.8.1.45 #define INT_LEAST8_MIN INT8_MIN

Definition at line 272 of file idn-int.h.

5.8.1.46 #define int_least8_t int8_t

Definition at line 159 of file idn-int.h.

5.8.1.47 #define INTMAX_C(x) INT64_C(x)

Definition at line 447 of file idn-int.h.

5.8.1.48 #define INTMAX_MAX INT64_MAX

Definition at line 351 of file idn-int.h.

5.8.1.49 #define INTMAX_MIN (~ INTMAX_MAX)

Definition at line 349 of file idn-int.h.

5.8.1.50 #define intmax_t int64_t

Definition at line 217 of file idn-int.h.

5.8.1.51 #define INTPTR_MAX LONG_MAX

Definition at line 341 of file idn-int.h.

5.8.1.52 #define INTPTR_MIN LONG_MIN

Definition at line 340 of file idn-int.h.

5.8.1.53 #define intptr_t long int

Definition at line 203 of file idn-int.h.

5.8.1.54 #define PTRDIFF_MAX _STDINT_MAX (1, 32, 0)

Definition at line 365 of file idn-int.h.

5.8.1.55 #define PTRDIFF_MIN _STDINT_MIN (1, 32, 0)

Definition at line 363 of file idn-int.h.

5.8.1.56 #define SIG_ATOMIC_MAX

Value:

```
_STDINT_MAX (1, 32, \
                0)
```

Definition at line 374 of file idn-int.h.

5.8.1.57 #define SIG_ATOMIC_MIN

Value:

```
_STDINT_MIN (1, 32, \
                0)
```

Definition at line 371 of file idn-int.h.

5.8.1.58 #define SIZE_MAX _STDINT_MAX (0, 32, 0u)

Definition at line 381 of file idn-int.h.

5.8.1.59 #define UINT16_C(x) x

Definition at line 419 of file idn-int.h.

5.8.1.60 #define UINT16_MAX 65535

Definition at line 245 of file idn-int.h.

5.8.1.61 #define uint16_t unsigned short int

Definition at line 119 of file idn-int.h.

5.8.1.62 #define UINT32_C(x) x ## U

Definition at line 424 of file idn-int.h.

5.8.1.63 #define UINT32_MAX 4294967295U

Definition at line 252 of file idn-int.h.

5.8.1.64 #define uint32_t unsigned int

Definition at line 124 of file idn-int.h.

Referenced by idna_to_ascii_4i(), idna_to_ascii_4z(), idna_to_ascii_8z(), idna_to_unicode_4z4z(), idna_to_unicode_8z4z(), idna_to_unicode_8z8z(), pr29_8z(), stringprep(), stringprep_4i(), stringprep_ucs4_nfkc_normalize(), tld_check_4t(), tld_check_4tz(), tld_check_4z(), tld_check_8z(), tld_get_4(), tld_get_4z(), and tld_get_z().

5.8.1.65 #define UINT64_C(x) x##ULL

Definition at line 436 of file idn-int.h.

5.8.1.66 #define UINT64_MAX UINTMAX_C (18446744073709551615)

Definition at line 260 of file idn-int.h.

5.8.1.67 #define uint64_t unsigned long long int

Definition at line 136 of file idn-int.h.

5.8.1.68 #define UINT8_C(x) x

Definition at line 414 of file idn-int.h.

5.8.1.69 #define UINT8_MAX 255

Definition at line 238 of file idn-int.h.

5.8.1.70 #define uint8_t unsigned char

Definition at line 114 of file idn-int.h.

5.8.1.71 #define UINT_FAST16_MAX ULONG_MAX

Definition at line 317 of file idn-int.h.

5.8.1.72 #define uint_fast16_t unsigned int_fast16_t

Definition at line 191 of file idn-int.h.

5.8.1.73 #define UINT_FAST32_MAX ULONG_MAX

Definition at line 324 of file idn-int.h.

5.8.1.74 #define uint_fast32_t unsigned int_fast32_t

Definition at line 193 of file idn-int.h.

5.8.1.75 #define UINT_FAST64_MAX UINT64_MAX

Definition at line 332 of file idn-int.h.

5.8.1.76 #define uint_fast64_t uint64_t

Definition at line 196 of file idn-int.h.

5.8.1.77 #define UINT_FAST8_MAX ULONG_MAX

Definition at line 310 of file idn-int.h.

5.8.1.78 #define uint_fast8_t unsigned int_fast8_t

Definition at line 189 of file idn-int.h.

5.8.1.79 #define UINT_LEAST16_MAX UINT16_MAX

Definition at line 281 of file idn-int.h.

5.8.1.80 #define uint_least16_t uint16_t

Definition at line 162 of file idn-int.h.

5.8.1.81 #define UINT_LEAST32_MAX UINT32_MAX

Definition at line 288 of file idn-int.h.

5.8.1.82 #define uint_least32_t uint32_t

Definition at line 164 of file idn-int.h.

5.8.1.83 #define UINT_LEAST64_MAX UINT64_MAX

Definition at line 296 of file idn-int.h.

5.8.1.84 #define uint_least64_t uint64_t

Definition at line 167 of file idn-int.h.

5.8.1.85 #define UINT_LEAST8_MAX UINT8_MAX

Definition at line 274 of file idn-int.h.

5.8.1.86 #define uint_least8_t uint8_t

Definition at line 160 of file idn-int.h.

5.8.1.87 #define UINTMAX_C(x) UINT64_C(x)

Definition at line 448 of file idn-int.h.

5.8.1.88 #define UINTMAX_MAX UINT64_MAX

Definition at line 352 of file idn-int.h.

5.8.1.89 #define uintmax_t uint64_t

Definition at line 218 of file idn-int.h.

5.8.1.90 #define UINTPTR_MAX ULONG_MAX

Definition at line 342 of file idn-int.h.

5.8.1.91 #define uintptr_t unsigned long int

Definition at line 204 of file idn-int.h.

5.8.1.92 #define WCHAR_MAX _STDINT_MAX (1, 32, 0)

Definition at line 388 of file idn-int.h.

5.8.1.93 #define WCHAR_MIN _STDINT_MIN (1, 32, 0)

Definition at line 386 of file idn-int.h.

5.8.1.94 #define WINT_MAX _STDINT_MAX (0, 32, 0u)

Definition at line 396 of file idn-int.h.

5.8.1.95 #define WINT_MIN _STDINT_MIN (0, 32, 0u)

Definition at line 394 of file idn-int.h.

5.9 idna.c File Reference

```
#include <stdlib.h>
#include <string.h>
#include <stringprep.h>
#include <punycode.h>
#include "idna.h"
```

Defines

- #define DOTP(c)

Functions

- int idna_to_ascii_4i (const uint32_t *in, size_t inlen, char *out, int flags)
- int idna_to_unicode_44i (const uint32_t *in, size_t inlen, uint32_t *out, size_t *outlen, int flags)
- int idna_to_ascii_4z (const uint32_t *input, char **output, int flags)
- int idna_to_ascii_8z (const char *input, char **output, int flags)
- int idna_to_ascii_lz (const char *input, char **output, int flags)
- int idna_to_unicode_4z4z (const uint32_t *input, uint32_t **output, int flags)
- int idna_to_unicode_8z4z (const char *input, uint32_t **output, int flags)
- int idna_to_unicode_8z8z (const char *input, char **output, int flags)
- int idna_to_unicode_8zlz (const char *input, char **output, int flags)
- int idna_to_unicode_lzlz (const char *input, char **output, int flags)

5.9.1 Define Documentation

5.9.1.1 #define DOTP(c)

Value:

```
((c) == 0x002E || (c) == 0x3002 ||       \
                  (c) == 0xFF0E || (c) == 0xFF61)
```

Definition at line 33 of file idna.c.

Referenced by idna_to_ascii_4z(), idna_to_unicode_4z4z(), and tld_get_4().

5.9.2 Function Documentation

5.9.2.1 int idna_to_ascii_4i (const uint32_t * *in*, size_t *inlen*, char * *out*, int *flags*)

idna_to_ascii_4i - convert Unicode domain name label to text

Parameters:

in input array with unicode code points.

inlen length of input array with unicode code points.

out output zero terminated string that must have room for at least 63 characters plus the terminating zero.

flags an Idna_flags value, e.g., IDNA_ALLOW_UNASSIGNED or IDNA_USE_STD3_ASCII_RULES.

The ToASCII operation takes a sequence of Unicode code points that make up one label and transforms it into a sequence of code points in the ASCII range (0..7F). 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 Idna_rc error code.

Definition at line 70 of file idna.c.

References IDNA_ACE_PREFIX, IDNA_ALLOW_UNASSIGNED, IDNA_CONTAINS_ACE_PREFIX, IDNA_CONTAINS_MINUS, IDNA_CONTAINS_NON_LDH, IDNA_INVALID_LENGTH, IDNA_MALLOC_ERROR, IDNA_PUNYCODE_ERROR, IDNA_STRINGPREP_ERROR, IDNA_SUCCESS, IDNA_USE_STD3_ASCII_RULES, punycode_encode(), PUNYCODE_SUCCESS, stringprep_nameprep, stringprep_nameprep_no_unassigned, STRINGPREP_OK, STRINGPREP_TOO_SMALL_BUFFER, stringprep_ucs4_to_utf8(), stringprep_utf8_to_ucs4(), and uint32_t.

Referenced by idna_to_ascii_4z().

5.9.2.2 int idna_to_ascii_4z (const uint32_t * *input*, char ** *output*, int *flags*)

idna_to_ascii_4z - convert Unicode domain name label to text

Parameters:

input zero terminated input Unicode string.

output pointer to newly allocated output string.

flags an Idna_flags value, e.g., IDNA_ALLOW_UNASSIGNED or IDNA_USE_STD3_ASCII_RULES.

Convert UCS-4 domain name to ASCII string. The domain name may contain several labels, separated by dots. The output buffer must be deallocated by the caller.

Return value: Returns IDNA_SUCCESS on success, or error code.

Definition at line 455 of file idna.c.

References DOTP, IDNA_MALLOC_ERROR, IDNA_SUCCESS, idna_to_ascii_4i(), and uint32_t.

Referenced by idna_to_ascii_8z().

5.9.2.3 int idna_to_ascii_8z (const char * *input*, char ** *output*, int *flags*)

idna_to_ascii_8z - convert Unicode domain name label to text

Parameters:

input zero terminated input UTF-8 string.

output pointer to newly allocated output string.

flags an Idna_flags value, e.g., IDNA_ALLOW_UNASSIGNED or IDNA_USE_STD3_ASCII_-RULES.

Convert UTF-8 domain name to ASCII string. The domain name may contain several labels, separated by dots. The output buffer must be deallocated by the caller.

Return value: Returns IDNA_SUCCESS on success, or error code.

Definition at line 551 of file idna.c.

References IDNA_ICONV_ERROR, idna_to_ascii_4z(), stringprep_utf8_to_ucs4(), and uint32_t.

Referenced by idna_to_ascii_lz().

5.9.2.4 int idna_to_ascii_lz (const char * *input*, char ** *output*, int *flags*)

idna_to_ascii_lz - convert Unicode domain name label to text

Parameters:

input zero terminated input string encoded in the current locale's character set.

output pointer to newly allocated output string.

flags an Idna_flags value, e.g., IDNA_ALLOW_UNASSIGNED or IDNA_USE_STD3_ASCII_-RULES.

Convert domain name in the locale's encoding to ASCII string. The domain name may contain several labels, separated by dots. The output buffer must be deallocated by the caller.

Return value: Returns IDNA_SUCCESS on success, or error code.

Definition at line 584 of file idna.c.

References IDNA_ICONV_ERROR, idna_to_ascii_8z(), and stringprep_locale_to_utf8().

5.9.2.5 int idna_to_unicode_44i (const uint32_t * *in*, size_t *inlen*, uint32_t * *out*, size_t * *outlen*, int *flags*)

idna_to_unicode_44i - convert domain name label to Unicode

Parameters:

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 an Idna_flags value, 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 label and returns a sequence of Unicode code points. If the input sequence is a label in ACE form, then the result is an

equivalent internationalized label that is not in ACE form, otherwise the original sequence is returned unaltered.

ToUnicode never fails. If any step fails, then the original input sequence is returned immediately in that step.

The Punycode decoder can never output more code points than it inputs, but Nameprep can, and therefore ToUnicode can. Note that the number of octets needed to represent a sequence of code points depends on the particular character encoding used.

The inputs to ToUnicode are a sequence of code points, the AllowUnassigned flag, and the UseSTD3ASCIIRules flag. The output of ToUnicode is always a sequence of Unicode code points.

Return value: Returns Idna_rc error condition, but it must only be used for debugging purposes. The output buffer is always guaranteed to contain the correct data according to the specification (sans malloc induced errors). NB! This means that you normally ignore the return code from this function, as checking it means breaking the standard.

Definition at line 415 of file idna.c.

References IDNA_MALLOC_ERROR, IDNA_SUCCESS, and stringprep_ucs4_to_utf8().

Referenced by idna_to_unicode_4z4z().

5.9.2.6 int idna_to_unicode_4z4z (const uint32_t * *input*, uint32_t ** *output*, int *flags*)

idna_to_unicode_4z4z - convert domain name label to Unicode

Parameters:

input zero-terminated Unicode string.

output pointer to newly allocated output Unicode string.

flags an Idna_flags value, e.g., IDNA_ALLOW_UNASSIGNED or IDNA_USE_STD3_ASCII_RULES.

Convert possibly ACE encoded domain name in UCS-4 format into a UCS-4 string. The domain name may contain several labels, separated by dots. The output buffer must be deallocated by the caller.

Return value: Returns IDNA_SUCCESS on success, or error code.

Definition at line 615 of file idna.c.

References DOTP, IDNA_MALLOC_ERROR, IDNA_SUCCESS, idna_to_unicode_44i(), and uint32_t.

Referenced by idna_to_unicode_8z4z().

5.9.2.7 int idna_to_unicode_8z4z (const char * *input*, uint32_t ** *output*, int *flags*)

idna_to_unicode_8z4z - convert domain name label to Unicode

Parameters:

input zero-terminated UTF-8 string.

output pointer to newly allocated output Unicode string.

flags an Idna_flags value, e.g., IDNA_ALLOW_UNASSIGNED or IDNA_USE_STD3_ASCII_RULES.

Convert possibly ACE encoded domain name in UTF-8 format into a UCS-4 string. The domain name may contain several labels, separated by dots. The output buffer must be deallocated by the caller.

Return value: Returns IDNA_SUCCESS on success, or error code.

Definition at line 691 of file idna.c.

References IDNA_ICONV_ERROR, idna_to_unicode_4z4z(), stringprep_utf8_to_ucs4(), and uint32_t.

Referenced by idna_to_unicode_8z8z().

5.9.2.8 int idna_to_unicode_8z8z (const char * *input*, char ** *output*, int *flags*)

idna_to_unicode_8z8z - convert domain name label to Unicode

Parameters:

input zero-terminated UTF-8 string.

output pointer to newly allocated output UTF-8 string.

flags an Idna_flags value, e.g., IDNA_ALLOW_UNASSIGNED or IDNA_USE_STD3_ASCII_RULES.

Convert possibly ACE encoded domain name in UTF-8 format into a UTF-8 string. The domain name may contain several labels, separated by dots. The output buffer must be deallocated by the caller.

Return value: Returns IDNA_SUCCESS on success, or error code.

Definition at line 722 of file idna.c.

References IDNA_ICONV_ERROR, idna_to_unicode_8z4z(), stringprep_ucs4_to_utf8(), and uint32_t.

Referenced by idna_to_unicode_8zlz().

5.9.2.9 int idna_to_unicode_8zlz (const char * *input*, char ** *output*, int *flags*)

idna_to_unicode_8zlz - convert domain name label to Unicode

Parameters:

input zero-terminated UTF-8 string.

output pointer to newly allocated output string encoded in the current locale's character set.

flags an Idna_flags value, e.g., IDNA_ALLOW_UNASSIGNED or IDNA_USE_STD3_ASCII_RULES.

Convert possibly ACE encoded domain name in UTF-8 format into a string encoded in the current locale's character set. The domain name may contain several labels, separated by dots. The output buffer must be deallocated by the caller.

Return value: Returns IDNA_SUCCESS on success, or error code.

Definition at line 753 of file idna.c.

References IDNA_ICONV_ERROR, idna_to_unicode_8z8z(), and stringprep_utf8_to_locale().

Referenced by idna_to_unicode_lzlz().

5.9.2.10 int idna_to_unicode_lzlz (const char * *input*, char ** *output*, int *flags*)

idna_to_unicode_lzlz - convert domain name label to Unicode

Parameters:

input zero-terminated string encoded in the current locale's character set.

output pointer to newly allocated output string encoded in the current locale's character set.

flags an Idna_flags value, e.g., IDNA_ALLOW_UNASSIGNED or IDNA_USE_STD3_ASCII_RULES.

Convert possibly ACE encoded domain name in the locale's character set into a string encoded in the current locale's character set. The domain name may contain several labels, separated by dots. The output buffer must be deallocated by the caller.

Return value: Returns IDNA_SUCCESS on success, or error code.

Definition at line 785 of file idna.c.

References IDNA_ICONV_ERROR, idna_to_unicode_8zlz(), and stringprep_locale_to_utf8().

5.10 idna.h File Reference

```
#include <stddef.h>
#include <idn-int.h>
```

Defines

- #define IDNA_ACE_PREFIX "xn–"

Enumerations

- enum Idna_rc {

  IDNA_SUCCESS = 0, IDNA_STRINGPREP_ERROR = 1, IDNA_PUNYCODE_ERROR = 2, IDNA_CONTAINS_NON_LDH = 3,

  IDNA_CONTAINS_LDH = IDNA_CONTAINS_NON_LDH, IDNA_CONTAINS_MINUS = 4, IDNA_INVALID_LENGTH = 5, IDNA_NO_ACE_PREFIX = 6,

  IDNA_ROUNDTRIP_VERIFY_ERROR = 7, IDNA_CONTAINS_ACE_PREFIX = 8, IDNA_-ICONV_ERROR = 9, IDNA_MALLOC_ERROR = 201,

  IDNA_DLOPEN_ERROR = 202 }
- enum Idna_flags { IDNA_ALLOW_UNASSIGNED = 0x0001, IDNA_USE_STD3_ASCII_RULES = 0x0002 }

Functions

- const char * idna_strerror (Idna_rc rc)
- int idna_to_ascii_4i (const uint32_t *in, size_t inlen, char *out, int flags)
- int idna_to_unicode_44i (const uint32_t *in, size_t inlen, uint32_t *out, size_t *outlen, int flags)
- int idna_to_ascii_4z (const uint32_t *input, char **output, int flags)
- int idna_to_ascii_8z (const char *input, char **output, int flags)
- int idna_to_ascii_lz (const char *input, char **output, int flags)
- int idna_to_unicode_4z4z (const uint32_t *input, uint32_t **output, int flags)
- int idna_to_unicode_8z4z (const char *input, uint32_t **output, int flags)
- int idna_to_unicode_8z8z (const char *input, char **output, int flags)
- int idna_to_unicode_8zlz (const char *input, char **output, int flags)
- int idna_to_unicode_lzlz (const char *input, char **output, int flags)

5.10.1 Define Documentation

5.10.1.1 #define IDNA_ACE_PREFIX "xn–"

Definition at line 61 of file idna.h.

Referenced by idna_to_ascii_4i().

5.10.2 Enumeration Type Documentation

5.10.2.1 enum Idna_flags

Enumerator:

IDNA_ALLOW_UNASSIGNED
IDNA_USE_STD3_ASCII_RULES

Definition at line 54 of file idna.h.

5.10.2.2 enum Idna_rc

Enumerator:

IDNA_SUCCESS
IDNA_STRINGPREP_ERROR
IDNA_PUNYCODE_ERROR
IDNA_CONTAINS_NON_LDH
IDNA_CONTAINS_LDH
IDNA_CONTAINS_MINUS
IDNA_INVALID_LENGTH
IDNA_NO_ACE_PREFIX
IDNA_ROUNDTRIP_VERIFY_ERROR
IDNA_CONTAINS_ACE_PREFIX
IDNA_ICONV_ERROR
IDNA_MALLOC_ERROR
IDNA_DLOPEN_ERROR

Definition at line 34 of file idna.h.

5.10.3 Function Documentation

5.10.3.1 const char* idna_strerror (Idna_rc *rc*)

idna_strerror - return string describing idna error code

Parameters:

rc an Idna_rc return code.

Convert a return code integer to a text string. This string can be used to output a diagnostic message to the user.

IDNA_SUCCESS: Successful operation. This value is guaranteed to always be zero, the remaining ones are only guaranteed to hold non-zero values, for logical comparison purposes. IDNA_STRINGPREP_-ERROR: Error during string preparation. IDNA_PUNYCODE_ERROR: Error during punycode operation. IDNA_CONTAINS_NON_LDH: For IDNA_USE_STD3_ASCII_RULES, indicate that the string contains non-LDH ASCII characters. IDNA_CONTAINS_MINUS: For IDNA_USE_STD3_ASCII_RULES, indicate that the string contains a leading or trailing hyphen-minus (U+002D). IDNA_INVALID_LENGTH:

The final output string is not within the (inclusive) range 1 to 63 characters. IDNA_NO_ACE_PREFIX: The string does not contain the ACE prefix (for ToUnicode). IDNA_ROUNDTRIP_VERIFY_ERROR: The ToASCII operation on output string does not equal the input. IDNA_CONTAINS_ACE_PREFIX: The input contains the ACE prefix (for ToASCII). IDNA_ICONV_ERROR: Could not convert string in locale encoding. IDNA_MALLOC_ERROR: Could not allocate buffer (this is typically a fatal error). IDNA_DLOPEN_ERROR: Could not dlopen the libcidn DSO (only used internally in libc).

Return value: Returns a pointer to a statically allocated string containing a description of the error with the return code .

Definition at line 65 of file strerror-idna.c.

References _, IDNA_CONTAINS_ACE_PREFIX, IDNA_CONTAINS_MINUS, IDNA_CONTAINS_NON_LDH, IDNA_DLOPEN_ERROR, IDNA_ICONV_ERROR, IDNA_INVALID_LENGTH, IDNA_MALLOC_ERROR, IDNA_NO_ACE_PREFIX, IDNA_PUNYCODE_ERROR, IDNA_ROUNDTRIP_VERIFY_ERROR, IDNA_STRINGPREP_ERROR, and IDNA_SUCCESS.

5.10.3.2 int idna_to_ascii_4i (const uint32_t * *in*, size_t *inlen*, char * *out*, int *flags*)

idna_to_ascii_4i - convert Unicode domain name label to text

Parameters:

in input array with unicode code points.

inlen length of input array with unicode code points.

out output zero terminated string that must have room for at least 63 characters plus the terminating zero.

flags an Idna_flags value, e.g., IDNA_ALLOW_UNASSIGNED or IDNA_USE_STD3_ASCII_RULES.

The ToASCII operation takes a sequence of Unicode code points that make up one label and transforms it into a sequence of code points in the ASCII range (0..7F). 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 Idna_rc error code.

Definition at line 70 of file idna.c.

References IDNA_ACE_PREFIX, IDNA_ALLOW_UNASSIGNED, IDNA_CONTAINS_ACE_PREFIX, IDNA_CONTAINS_MINUS, IDNA_CONTAINS_NON_LDH, IDNA_INVALID_LENGTH, IDNA_MALLOC_ERROR, IDNA_PUNYCODE_ERROR, IDNA_STRINGPREP_ERROR, IDNA_SUCCESS, IDNA_USE_STD3_ASCII_RULES, punycode_encode(), PUNYCODE_SUCCESS, stringprep_nameprep, stringprep_nameprep_no_unassigned, STRINGPREP_OK, STRINGPREP_TOO_SMALL_BUFFER, stringprep_ucs4_to_utf8(), stringprep_utf8_to_ucs4(), and uint32_t.

Referenced by idna_to_ascii_4z().

5.10.3.3 int idna_to_ascii_4z (const uint32_t * *input*, char ** *output*, int *flags*)

idna_to_ascii_4z - convert Unicode domain name label to text

Parameters:

input zero terminated input Unicode string.

output pointer to newly allocated output string.

flags an Idna_flags value, e.g., IDNA_ALLOW_UNASSIGNED or IDNA_USE_STD3_ASCII_RULES.

Convert UCS-4 domain name to ASCII string. The domain name may contain several labels, separated by dots. The output buffer must be deallocated by the caller.

Return value: Returns IDNA_SUCCESS on success, or error code.

Definition at line 455 of file idna.c.

References DOTP, IDNA_MALLOC_ERROR, IDNA_SUCCESS, idna_to_ascii_4i(), and uint32_t.

Referenced by idna_to_ascii_8z().

5.10.3.4 int idna_to_ascii_8z (const char * *input*, char ** *output*, int *flags*)

idna_to_ascii_8z - convert Unicode domain name label to text

Parameters:

input zero terminated input UTF-8 string.

output pointer to newly allocated output string.

flags an Idna_flags value, e.g., IDNA_ALLOW_UNASSIGNED or IDNA_USE_STD3_ASCII_RULES.

Convert UTF-8 domain name to ASCII string. The domain name may contain several labels, separated by dots. The output buffer must be deallocated by the caller.

Return value: Returns IDNA_SUCCESS on success, or error code.

Definition at line 551 of file idna.c.

References IDNA_ICONV_ERROR, idna_to_ascii_4z(), stringprep_utf8_to_ucs4(), and uint32_t.

Referenced by idna_to_ascii_lz().

5.10.3.5 int idna_to_ascii_lz (const char * *input*, char ** *output*, int *flags*)

idna_to_ascii_lz - convert Unicode domain name label to text

Parameters:

input zero terminated input string encoded in the current locale's character set.

output pointer to newly allocated output string.

flags an Idna_flags value, e.g., IDNA_ALLOW_UNASSIGNED or IDNA_USE_STD3_ASCII_RULES.

Convert domain name in the locale's encoding to ASCII string. The domain name may contain several labels, separated by dots. The output buffer must be deallocated by the caller.

Return value: Returns IDNA_SUCCESS on success, or error code.

Definition at line 584 of file idna.c.

References IDNA_ICONV_ERROR, idna_to_ascii_8z(), and stringprep_locale_to_utf8().

5.10.3.6 int idna_to_unicode_44i (const uint32_t * *in*, size_t *inlen*, uint32_t * *out*, size_t * *outlen*, int *flags*)

idna_to_unicode_44i - convert domain name label to Unicode

Parameters:

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 an Idna_flags value, 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 label and returns a sequence of Unicode code points. If the input sequence is a label in ACE form, then the result is an equivalent internationalized label that is not in ACE form, otherwise the original sequence is returned unaltered.

ToUnicode never fails. If any step fails, then the original input sequence is returned immediately in that step.

The Punycode decoder can never output more code points than it inputs, but Nameprep can, and therefore ToUnicode can. Note that the number of octets needed to represent a sequence of code points depends on the particular character encoding used.

The inputs to ToUnicode are a sequence of code points, the AllowUnassigned flag, and the UseSTD3ASCIIRules flag. The output of ToUnicode is always a sequence of Unicode code points.

Return value: Returns Idna_rc error condition, but it must only be used for debugging purposes. The output buffer is always guaranteed to contain the correct data according to the specification (sans malloc induced errors). NB! This means that you normally ignore the return code from this function, as checking it means breaking the standard.

Definition at line 415 of file idna.c.

References IDNA_MALLOC_ERROR, IDNA_SUCCESS, and stringprep_ucs4_to_utf8().

Referenced by idna_to_unicode_4z4z().

5.10.3.7 int idna_to_unicode_4z4z (const uint32_t * *input*, uint32_t ** *output*, int *flags*)

idna_to_unicode_4z4z - convert domain name label to Unicode

Parameters:

input zero-terminated Unicode string.

output pointer to newly allocated output Unicode string.

flags an Idna_flags value, e.g., IDNA_ALLOW_UNASSIGNED or IDNA_USE_STD3_ASCII_RULES.

Convert possibly ACE encoded domain name in UCS-4 format into a UCS-4 string. The domain name may contain several labels, separated by dots. The output buffer must be deallocated by the caller.

Return value: Returns IDNA_SUCCESS on success, or error code.

Definition at line 615 of file idna.c.

References DOTP, IDNA_MALLOC_ERROR, IDNA_SUCCESS, idna_to_unicode_44i(), and uint32_t.

Referenced by idna_to_unicode_8z4z().

5.10.3.8 int idna_to_unicode_8z4z (const char * *input*, uint32_t ** *output*, int *flags*)

idna_to_unicode_8z4z - convert domain name label to Unicode

Parameters:

input zero-terminated UTF-8 string.

output pointer to newly allocated output Unicode string.

flags an Idna_flags value, e.g., IDNA_ALLOW_UNASSIGNED or IDNA_USE_STD3_ASCII_RULES.

Convert possibly ACE encoded domain name in UTF-8 format into a UCS-4 string. The domain name may contain several labels, separated by dots. The output buffer must be deallocated by the caller.

Return value: Returns IDNA_SUCCESS on success, or error code.

Definition at line 691 of file idna.c.

References IDNA_ICONV_ERROR, idna_to_unicode_4z4z(), stringprep_utf8_to_ucs4(), and uint32_t.

Referenced by idna_to_unicode_8z8z().

5.10.3.9 int idna_to_unicode_8z8z (const char * *input*, char ** *output*, int *flags*)

idna_to_unicode_8z8z - convert domain name label to Unicode

Parameters:

input zero-terminated UTF-8 string.

output pointer to newly allocated output UTF-8 string.

flags an Idna_flags value, e.g., IDNA_ALLOW_UNASSIGNED or IDNA_USE_STD3_ASCII_RULES.

Convert possibly ACE encoded domain name in UTF-8 format into a UTF-8 string. The domain name may contain several labels, separated by dots. The output buffer must be deallocated by the caller.

Return value: Returns IDNA_SUCCESS on success, or error code.

Definition at line 722 of file idna.c.

References IDNA_ICONV_ERROR, idna_to_unicode_8z4z(), stringprep_ucs4_to_utf8(), and uint32_t.

Referenced by idna_to_unicode_8zlz().

5.10.3.10 int idna_to_unicode_8zlz (const char * *input*, char ** *output*, int *flags*)

idna_to_unicode_8zlz - convert domain name label to Unicode

Parameters:

input zero-terminated UTF-8 string.

output pointer to newly allocated output string encoded in the current locale's character set.

flags an Idna_flags value, e.g., IDNA_ALLOW_UNASSIGNED or IDNA_USE_STD3_ASCII_RULES.

Convert possibly ACE encoded domain name in UTF-8 format into a string encoded in the current locale's character set. The domain name may contain several labels, separated by dots. The output buffer must be deallocated by the caller.

Return value: Returns IDNA_SUCCESS on success, or error code.

Definition at line 753 of file idna.c.

References IDNA_ICONV_ERROR, idna_to_unicode_8z8z(), and stringprep_utf8_to_locale().

Referenced by idna_to_unicode_lzlz().

5.10.3.11 int idna_to_unicode_lzlz (const char * *input*, char ** *output*, int *flags*)

idna_to_unicode_lzlz - convert domain name label to Unicode

Parameters:

input zero-terminated string encoded in the current locale's character set.

output pointer to newly allocated output string encoded in the current locale's character set.

flags an Idna_flags value, e.g., IDNA_ALLOW_UNASSIGNED or IDNA_USE_STD3_ASCII_RULES.

Convert possibly ACE encoded domain name in the locale's character set into a string encoded in the current locale's character set. The domain name may contain several labels, separated by dots. The output buffer must be deallocated by the caller.

Return value: Returns IDNA_SUCCESS on success, or error code.

Definition at line 785 of file idna.c.

References IDNA_ICONV_ERROR, idna_to_unicode_8zlz(), and stringprep_locale_to_utf8().

5.11 nfkc.c File Reference

```
#include <stdlib.h>
#include <string.h>
#include "stringprep.h"
#include "gunidecomp.h"
#include "gunicomp.h"
```

Defines

- #define gboolean int
- #define gchar char
- #define guchar unsigned char
- #define glong long
- #define gint int
- #define guint unsigned int
- #define gushort unsigned short
- #define gint16 int16_t
- #define guint16 uint16_t
- #define gunichar uint32_t
- #define gsize size_t
- #define gssize ssize_t
- #define g_malloc malloc
- #define g_free free
- #define GError void
- #define g_set_error(a, b, c, d) ((void) 0)
- #define g_new(struct_type, n_structs) ((struct_type *) g_malloc (((gsize) sizeof (struct_type)) * ((gsize) (n_structs))))
- #define G_STMT_START do
- #define G_STMT_END while (0)
- #define g_return_val_if_fail(expr, val) G_STMT_START{ (void)0; }G_STMT_END
- #define G_N_ELEMENTS(arr) (sizeof (arr) / sizeof ((arr)[0]))
- #define TRUE 1
- #define FALSE 0
- #define UTF8_COMPUTE(Char, Mask, Len)
- #define UTF8_LENGTH(Char)
- #define UTF8_GET(Result, Chars, Count, Mask, Len)
- #define UNICODE_VALID(Char)
- #define g_utf8_next_char(p) (char *)((p) + g_utf8_skip[*(guchar *)(p)])
- #define CC_PART1(Page, Char)
- #define CC_PART2(Page, Char)
- #define COMBINING_CLASS(Char)
- #define SBase 0xAC00
- #define LBase 0x1100
- #define VBase 0x1161
- #define TBase 0x11A7
- #define LCount 19
- #define VCount 21

- #define TCount 28
- #define NCount (VCount ∗ TCount)
- #define SCount (LCount ∗ NCount)
- #define CI(Page, Char)
- #define COMPOSE_INDEX(Char) ((((Char) >> 8) > (COMPOSE_TABLE_LAST)) ? 0 : CI((Char) >> 8, (Char) & 0xff))

Enumerations

- enum GNormalizeMode {

  G_NORMALIZE_DEFAULT, G_NORMALIZE_NFD = G_NORMALIZE_DEFAULT, G_NORMALIZE_DEFAULT_COMPOSE, G_NORMALIZE_NFC = G_NORMALIZE_DEFAULT_COMPOSE,

  G_NORMALIZE_ALL, G_NORMALIZE_NFKD = G_NORMALIZE_ALL, G_NORMALIZE_ALL_COMPOSE, G_NORMALIZE_NFKC = G_NORMALIZE_ALL_COMPOSE }

Functions

- uint32_t stringprep_utf8_to_unichar (const char ∗p)
- int stringprep_unichar_to_utf8 (uint32_t c, char ∗outbuf)
- uint32_t ∗ stringprep_utf8_to_ucs4 (const char ∗str, ssize_t len, size_t ∗items_written)
- char ∗ stringprep_ucs4_to_utf8 (const uint32_t ∗str, ssize_t len, size_t ∗items_read, size_t ∗items_written)
- char ∗ stringprep_utf8_nfkc_normalize (const char ∗str, ssize_t len)
- uint32_t ∗ stringprep_ucs4_nfkc_normalize (uint32_t ∗str, ssize_t len)

5.11.1 Define Documentation

5.11.1.1 #define CC_PART1(Page, Char)

Value:

```
((combining_class_table_part1[Page] >= G_UNICODE_MAX_TABLE_INDEX) \
   ? (combining_class_table_part1[Page] - G_UNICODE_MAX_TABLE_INDEX) \
   : (cclass_data[combining_class_table_part1[Page]][Char]))
```

Definition at line 497 of file nfkc.c.

5.11.1.2 #define CC_PART2(Page, Char)

Value:

```
((combining_class_table_part2[Page] >= G_UNICODE_MAX_TABLE_INDEX) \
   ? (combining_class_table_part2[Page] - G_UNICODE_MAX_TABLE_INDEX) \
   : (cclass_data[combining_class_table_part2[Page]][Char]))
```

Definition at line 502 of file nfkc.c.

5.11.1.3 #define CI(Page, Char)

Value:

```
((compose_table[Page] >= G_UNICODE_MAX_TABLE_INDEX) \
   ? (compose_table[Page] - G_UNICODE_MAX_TABLE_INDEX) \
   : (compose_data[compose_table[Page]][Char]))
```

Definition at line 679 of file nfkc.c.

5.11.1.4 #define COMBINING_CLASS(Char)

Value:

```
(((Char) <= G_UNICODE_LAST_CHAR_PART1) \
   ? CC_PART1 ((Char) >> 8, (Char) & 0xff) \
   : (((Char) >= 0xe0000 && (Char) <= G_UNICODE_LAST_CHAR) \
      ? CC_PART2 (((Char) - 0xe0000) >> 8, (Char) & 0xff) \
      : 0))
```

Definition at line 507 of file nfkc.c.

5.11.1.5 #define COMPOSE_INDEX(Char) ((((Char) >> 8) > (COMPOSE_TABLE_LAST)) ? 0 : CI((Char) >> 8, (Char) & 0xff))

Definition at line 684 of file nfkc.c.

5.11.1.6 #define FALSE 0

Definition at line 72 of file nfkc.c.

5.11.1.7 #define g_free free

Definition at line 52 of file nfkc.c.

5.11.1.8 #define g_malloc malloc

Definition at line 51 of file nfkc.c.

5.11.1.9 #define G_N_ELEMENTS(arr) (sizeof (arr) / sizeof ((arr)[0]))

Definition at line 70 of file nfkc.c.

5.11.1.10 #define g_new(struct_type, n_structs) ((struct_type *) g_malloc (((gsize) sizeof (struct_type)) * ((gsize) (n_structs))))

Definition at line 55 of file nfkc.c.

5.11.1.11 #define g_return_val_if_fail(expr, val) G_STMT_START{ (void)0; }G_STMT_END

Definition at line 69 of file nfkc.c.

5.11.1.12 #define g_set_error(a, b, c, d) ((void) 0)

Definition at line 54 of file nfkc.c.

5.11.1.13 #define G_STMT_END while (0)

Definition at line 66 of file nfkc.c.

5.11.1.14 #define G_STMT_START do

Definition at line 65 of file nfkc.c.

5.11.1.15 #define g_utf8_next_char(p) (char *)((p) + g_utf8_skip[*(guchar *)(p)])

Definition at line 174 of file nfkc.c.

5.11.1.16 #define gboolean int

Definition at line 39 of file nfkc.c.

5.11.1.17 #define gchar char

Definition at line 40 of file nfkc.c.

5.11.1.18 #define GError void

Definition at line 53 of file nfkc.c.

5.11.1.19 #define gint int

Definition at line 43 of file nfkc.c.

5.11.1.20 #define gint16 int16_t

Definition at line 46 of file nfkc.c.

5.11.1.21 #define glong long

Definition at line 42 of file nfkc.c.

Referenced by stringprep_ucs4_to_utf8(), and stringprep_utf8_to_ucs4().

5.11.1.22 #define gsize size_t

Definition at line 49 of file nfkc.c.

5.11.1.23 #define gssize ssize_t

Definition at line 50 of file nfkc.c.

5.11.1.24 #define guchar unsigned char

Definition at line 41 of file nfkc.c.

5.11.1.25 #define guint unsigned int

Definition at line 44 of file nfkc.c.

5.11.1.26 #define guint16 uint16_t

Definition at line 47 of file nfkc.c.

5.11.1.27 #define gunichar uint32_t

Definition at line 48 of file nfkc.c.

5.11.1.28 #define gushort unsigned short

Definition at line 45 of file nfkc.c.

5.11.1.29 #define LBase 0x1100

Definition at line 516 of file nfkc.c.

5.11.1.30 #define LCount 19

Definition at line 519 of file nfkc.c.

5.11.1.31 #define NCount (VCount * TCount)

Definition at line 522 of file nfkc.c.

5.11.1.32 #define SBase 0xAC00

Definition at line 515 of file nfkc.c.

5.11.1.33 #define SCount (LCount * NCount)

Definition at line 523 of file nfkc.c.

5.11.1.34 #define TBase 0x11A7

Definition at line 518 of file nfkc.c.

5.11.1.35 #define TCount 28

Definition at line 521 of file nfkc.c.

5.11.1.36 #define TRUE 1

Definition at line 71 of file nfkc.c.

5.11.1.37 #define UNICODE_VALID(Char)

Value:

```
((Char) < 0x110000 &&                           \
   (((Char) & 0xFFFFF800) != 0xD800) &&         \
   ((Char) < 0xFDD0 || (Char) > 0xFDEF) &&      \
   ((Char) & 0xFFFE) != 0xFFFE)
```

Definition at line 146 of file nfkc.c.

5.11.1.38 #define UTF8_COMPUTE(Char, Mask, Len)

Definition at line 91 of file nfkc.c.

5.11.1.39 #define UTF8_GET(Result, Chars, Count, Mask, Len)

Value:

```
(Result) = (Chars)[0] & (Mask);                 \
  for ((Count) = 1; (Count) < (Len); ++(Count))         \
    {                                                   \
      if (((Chars)[(Count)] & 0xc0) != 0x80)            \
        {                                               \
          (Result) = -1;                                \
          break;                                        \
        }                                               \
      (Result) <<= 6;                                   \
      (Result) |= ((Chars)[(Count)] & 0x3f);            \
    }
```

Definition at line 133 of file nfkc.c.

5.11.1.40 #define UTF8_LENGTH(Char)

Value:

```
((Char) < 0x80 ? 1 :                          \
   ((Char) < 0x800 ? 2 :                      \
    ((Char) < 0x10000 ? 3 :                   \
     ((Char) < 0x200000 ? 4 :                 \
      ((Char) < 0x4000000 ? 5 : 6)))))
```

Definition at line 125 of file nfkc.c.

5.11.1.41 #define VBase 0x1161

Definition at line 517 of file nfkc.c.

5.11.1.42 #define VCount 21

Definition at line 520 of file nfkc.c.

5.11.2 Enumeration Type Documentation

5.11.2.1 enum GNormalizeMode

Enumerator:

G_NORMALIZE_DEFAULT

G_NORMALIZE_NFD

G_NORMALIZE_DEFAULT_COMPOSE

G_NORMALIZE_NFC

G_NORMALIZE_ALL

G_NORMALIZE_NFKD

G_NORMALIZE_ALL_COMPOSE

G_NORMALIZE_NFKC

Definition at line 76 of file nfkc.c.

5.11.3 Function Documentation

5.11.3.1 uint32_t* stringprep_ucs4_nfkc_normalize (uint32_t * *str*, ssize_t *len*)

stringprep_ucs4_nfkc_normalize - normalize Unicode string

Parameters:

str a Unicode string.

len length of array, or -1 if is nul-terminated.

Converts UCS4 string into UTF-8 and runs stringprep_utf8_nfkc_normalize().

Return value: a newly allocated Unicode string, that is the NFKC normalized form of .

Definition at line 1048 of file nfkc.c.

References G_NORMALIZE_NFKC, stringprep_ucs4_to_utf8(), and uint32_t.

Referenced by stringprep_4i().

5.11.3.2 char* stringprep_ucs4_to_utf8 (const uint32_t * *str*, ssize_t *len*, size_t * *items_read*, size_t * *items_written*)

stringprep_ucs4_to_utf8 - convert UCS-4 string to UTF-8

Parameters:

str a UCS-4 encoded string

len the maximum length of to use. If < 0, then the string is terminated with a 0 character.

items_read location to store number of characters read read, or NULL.

items_written location to store number of bytes written or NULL. The value here stored does not include the trailing 0 byte.

Convert a string from a 32-bit fixed width representation as UCS-4. to UTF-8. The result will be terminated with a 0 byte.

Return value: a pointer to a newly allocated UTF-8 string. This value must be freed with free(). If an error occurs, NULL will be returned and set.

Definition at line 1001 of file nfkc.c.

References glong.

Referenced by idna_to_ascii_4i(), idna_to_unicode_44i(), idna_to_unicode_8z8z(), stringprep(), and stringprep_ucs4_nfkc_normalize().

5.11.3.3 int stringprep_unichar_to_utf8 (uint32_t *c*, char * *outbuf*)

stringprep_unichar_to_utf8 - convert Unicode code point to UTF-8

Parameters:

c a ISO10646 character code

outbuf output buffer, must have at least 6 bytes of space. If NULL, the length will be computed and returned and nothing will be written to .

Converts a single character to UTF-8.

Return value: number of bytes written.

Definition at line 956 of file nfkc.c.

5.11.3.4 char* stringprep_utf8_nfkc_normalize (const char * *str*, ssize_t *len*)

stringprep_utf8_nfkc_normalize - normalize Unicode string

Parameters:

str a UTF-8 encoded string.

len length of , in bytes, or -1 if is nul-terminated.

Converts a string into canonical form, standardizing such issues as whether a character with an accent is represented as a base character and combining accent or as a single precomposed character.

The normalization mode is NFKC (ALL COMPOSE). It standardizes differences that do not affect the text content, such as the above-mentioned accent representation. It standardizes the "compatibility" characters in Unicode, such as SUPERSCRIPT THREE to the standard forms (in this case DIGIT THREE). Formatting information may be lost but for most text operations such characters should be considered the same. It returns a result with composed forms rather than a maximally decomposed form.

Return value: a newly allocated string, that is the NFKC normalized form of .

Definition at line 1031 of file nfkc.c.

References G_NORMALIZE_NFKC.

5.11.3.5 uint32_t* stringprep_utf8_to_ucs4 (const char * *str*, ssize_t *len*, size_t * *items_written*)

stringprep_utf8_to_ucs4 - convert UTF-8 string to UCS-4

Parameters:

str a UTF-8 encoded string

len the maximum length of to use. If < 0, then the string is nul-terminated.

items_written location to store the number of characters in the result, or NULL.

Convert a string from UTF-8 to a 32-bit fixed width representation as UCS-4, assuming valid UTF-8 input. This function does no error checking on the input.

Return value: a pointer to a newly allocated UCS-4 string. This value must be freed with free().

Definition at line 977 of file nfkc.c.

References glong.

Referenced by idna_to_ascii_4i(), idna_to_ascii_8z(), idna_to_unicode_8z4z(), pr29_8z(), stringprep(), and tld_check_8z().

5.11.3.6 uint32_t stringprep_utf8_to_unichar (const char * *p*)

stringprep_utf8_to_unichar - convert UTF-8 to Unicode code point

Parameters:

p a pointer to Unicode character encoded as UTF-8

Converts a sequence of bytes encoded as UTF-8 to a Unicode character. If `does` not point to a valid UTF-8 encoded character, results are undefined.

Return value: the resulting character.

Definition at line 939 of file nfkc.c.

5.12 pr29.c File Reference

```
#include "pr29.h"
#include <stringprep.h>
```

Functions

- int pr29_4 (const uint32_t *in, size_t len)
- int pr29_4z (const uint32_t *in)
- int pr29_8z (const char *in)

5.12.1 Function Documentation

5.12.1.1 int pr29_4 (const uint32_t * *in*, size_t *len*)

pr29_4 - check if input trigger Unicode normalization bugs

Parameters:

in input array with unicode code points.

len length of input array with unicode code points.

Check the input to see if it may be normalized into different strings by different NFKC implementations, due to an anomaly in the NFKC specifications.

Return value: Returns the Pr29_rc value PR29_SUCCESS on success, and PR29_PROBLEM if the input sequence is a "problem sequence" (i.e., may be normalized into different strings by different implementations).

Definition at line 1237 of file pr29.c.

References PR29_PROBLEM, and PR29_SUCCESS.

Referenced by pr29_4z().

5.12.1.2 int pr29_4z (const uint32_t * *in*)

pr29_4z - check if input trigger Unicode normalization bugs

Parameters:

in zero terminated array of Unicode code points.

Check the input to see if it may be normalized into different strings by different NFKC implementations, due to an anomaly in the NFKC specifications.

Return value: Returns the Pr29_rc value PR29_SUCCESS on success, and PR29_PROBLEM if the input sequence is a "problem sequence" (i.e., may be normalized into different strings by different implementations).

Definition at line 1278 of file pr29.c.

References pr29_4().

Referenced by pr29_8z().

5.12.1.3 int pr29_8z (const char * *in*)

pr29_8z - check if input trigger Unicode normalization bugs

Parameters:

in zero terminated input UTF-8 string.

Check the input to see if it may be normalized into different strings by different NFKC implementations, due to an anomaly in the NFKC specifications.

Return value: Returns the Pr29_rc value PR29_SUCCESS on success, and PR29_PROBLEM if the input sequence is a "problem sequence" (i.e., may be normalized into different strings by different implementations), or PR29_STRINGPREP_ERROR if there was a problem converting the string from UTF-8 to UCS-4.

Definition at line 1303 of file pr29.c.

References pr29_4z(), PR29_STRINGPREP_ERROR, stringprep_utf8_to_ucs4(), and uint32_t.

5.13 pr29.h File Reference

```
#include <stdlib.h>
#include <idn-int.h>
```

Enumerations

- enum Pr29_rc { PR29_SUCCESS = 0, PR29_PROBLEM = 1, PR29_STRINGPREP_ERROR = 2 }

Functions

- const char * pr29_strerror (Pr29_rc rc)
- int pr29_4 (const uint32_t *in, size_t len)
- int pr29_4z (const uint32_t *in)
- int pr29_8z (const char *in)

5.13.1 Enumeration Type Documentation

5.13.1.1 enum Pr29_rc

Enumerator:

PR29_SUCCESS

PR29_PROBLEM

PR29_STRINGPREP_ERROR

Definition at line 37 of file pr29.h.

5.13.2 Function Documentation

5.13.2.1 int pr29_4 (const uint32_t * *in*, size_t *len*)

pr29_4 - check if input trigger Unicode normalization bugs

Parameters:

in input array with unicode code points.

len length of input array with unicode code points.

Check the input to see if it may be normalized into different strings by different NFKC implementations, due to an anomaly in the NFKC specifications.

Return value: Returns the Pr29_rc value PR29_SUCCESS on success, and PR29_PROBLEM if the input sequence is a "problem sequence" (i.e., may be normalized into different strings by different implementations).

Definition at line 1237 of file pr29.c.

References PR29_PROBLEM, and PR29_SUCCESS.

Referenced by pr29_4z().

5.13.2.2 int pr29_4z (const uint32_t * *in*)

pr29_4z - check if input trigger Unicode normalization bugs

Parameters:

in zero terminated array of Unicode code points.

Check the input to see if it may be normalized into different strings by different NFKC implementations, due to an anomaly in the NFKC specifications.

Return value: Returns the Pr29_rc value PR29_SUCCESS on success, and PR29_PROBLEM if the input sequence is a "problem sequence" (i.e., may be normalized into different strings by different implementations).

Definition at line 1278 of file pr29.c.

References pr29_4().

Referenced by pr29_8z().

5.13.2.3 int pr29_8z (const char * *in*)

pr29_8z - check if input trigger Unicode normalization bugs

Parameters:

in zero terminated input UTF-8 string.

Check the input to see if it may be normalized into different strings by different NFKC implementations, due to an anomaly in the NFKC specifications.

Return value: Returns the Pr29_rc value PR29_SUCCESS on success, and PR29_PROBLEM if the input sequence is a "problem sequence" (i.e., may be normalized into different strings by different implementations), or PR29_STRINGPREP_ERROR if there was a problem converting the string from UTF-8 to UCS-4.

Definition at line 1303 of file pr29.c.

References pr29_4z(), PR29_STRINGPREP_ERROR, stringprep_utf8_to_ucs4(), and uint32_t.

5.13.2.4 const char* pr29_strerror (Pr29_rc *rc*)

pr29_strerror - return string describing pr29 error code

Parameters:

rc an Pr29_rc return code.

Convert a return code integer to a text string. This string can be used to output a diagnostic message to the user.

PR29_SUCCESS: Successful operation. This value is guaranteed to always be zero, the remaining ones are only guaranteed to hold non-zero values, for logical comparison purposes. PR29_PROBLEM: A problem sequence was encountered. PR29_STRINGPREP_ERROR: The character set conversion failed (only for pr29_8() and pr29_8z()).

Return value: Returns a pointer to a statically allocated string containing a description of the error with the return code .

Definition at line 49 of file strerror-pr29.c.

References _, PR29_PROBLEM, PR29_STRINGPREP_ERROR, and PR29_SUCCESS.

5.14 profiles.c File Reference

```
#include "stringprep.h"
```

Variables

- const Stringprep_profiles stringprep_profiles [ ]
- const Stringprep_profile stringprep_nameprep [ ]
- const Stringprep_profile stringprep_kerberos5 [ ]
- const Stringprep_table_element stringprep_xmpp_nodeprep_prohibit [ ]
- const Stringprep_profile stringprep_xmpp_nodeprep [ ]
- const Stringprep_profile stringprep_xmpp_resourceprep [ ]
- const Stringprep_profile stringprep_plain [ ]
- const Stringprep_profile stringprep_trace [ ]
- const Stringprep_table_element stringprep_iscsi_prohibit [ ]
- const Stringprep_profile stringprep_iscsi [ ]
- const Stringprep_table_element stringprep_saslprep_space_map [ ]
- const Stringprep_profile stringprep_saslprep [ ]

5.14.1 Variable Documentation

5.14.1.1 const Stringprep_profile stringprep_iscsi[ ]

Initial value:

```
{
  {STRINGPREP_MAP_TABLE, 0, stringprep_rfc3454_B_1},
  {STRINGPREP_MAP_TABLE, 0, stringprep_rfc3454_B_2},
  {STRINGPREP_NFKC, 0, 0},
  {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_1_1},
  {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_1_2},
  {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_2_1},
  {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_2_2},
  {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_3},
  {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_4},
  {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_5},
  {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_6},
  {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_7},
  {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_8},
  {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_9},
  {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_iscsi_prohibit},
  {STRINGPREP_BIDI, 0, 0},
  {STRINGPREP_BIDI_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_8},
  {STRINGPREP_BIDI_RAL_TABLE, ~STRINGPREP_NO_BIDI, stringprep_rfc3454_D_1},
  {STRINGPREP_BIDI_L_TABLE, ~STRINGPREP_NO_BIDI, stringprep_rfc3454_D_2},
  {STRINGPREP_UNASSIGNED_TABLE, ~STRINGPREP_NO_UNASSIGNED,
   stringprep_rfc3454_A_1},
  {0}
}
```

Definition at line 246 of file profiles.c.

5.14.1.2 const Stringprep_table_element stringprep_iscsi_prohibit[ ]

Definition at line 176 of file profiles.c.

5.14.1.3 const Stringprep_profile stringprep_kerberos5[ ]

Initial value:

```
 {

  {STRINGPREP_MAP_TABLE, 0, stringprep_rfc3454_B_1},
  {STRINGPREP_MAP_TABLE, 0, stringprep_rfc3454_B_3},
  {STRINGPREP_NFKC, 0, 0},
  {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_1_2},
  {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_2_2},
  {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_3},
  {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_4},
  {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_5},
  {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_6},
  {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_7},
  {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_8},
  {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_9},
  {STRINGPREP_BIDI, 0, 0},
  {STRINGPREP_BIDI_PROHIBIT_TABLE, ~STRINGPREP_NO_BIDI,
   stringprep_rfc3454_C_8},
  {STRINGPREP_BIDI_RAL_TABLE, 0, stringprep_rfc3454_D_1},
  {STRINGPREP_BIDI_L_TABLE, 0, stringprep_rfc3454_D_2},
  {STRINGPREP_UNASSIGNED_TABLE, ~STRINGPREP_NO_UNASSIGNED,
   stringprep_rfc3454_A_1},
  {0}
}
```

Definition at line 60 of file profiles.c.

5.14.1.4 const Stringprep_profile stringprep_nameprep[ ]

Initial value:

```
 {
  {STRINGPREP_MAP_TABLE, 0, stringprep_rfc3454_B_1},
  {STRINGPREP_MAP_TABLE, 0, stringprep_rfc3454_B_2},
  {STRINGPREP_NFKC, 0, 0},
  {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_1_2},
  {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_2_2},
  {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_3},
  {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_4},
  {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_5},
  {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_6},
  {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_7},
  {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_8},
  {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_9},
  {STRINGPREP_BIDI, 0, 0},
  {STRINGPREP_BIDI_PROHIBIT_TABLE, ~STRINGPREP_NO_BIDI,
   stringprep_rfc3454_C_8},
  {STRINGPREP_BIDI_RAL_TABLE, 0, stringprep_rfc3454_D_1},
  {STRINGPREP_BIDI_L_TABLE, 0, stringprep_rfc3454_D_2},
  {STRINGPREP_UNASSIGNED_TABLE, ~STRINGPREP_NO_UNASSIGNED,
   stringprep_rfc3454_A_1},
  {0}
}
```

Definition at line 37 of file profiles.c.

5.14.1.5 const Stringprep_profile stringprep_plain[ ]

Initial value:

```
{
 {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_2_1},
 {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_2_2},
 {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_3},
 {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_4},
 {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_5},
 {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_6},
 {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_8},
 {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_9},
 {STRINGPREP_BIDI, 0, 0},
 {STRINGPREP_BIDI_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_8},
 {STRINGPREP_BIDI_RAL_TABLE, ~STRINGPREP_NO_BIDI, stringprep_rfc3454_D_1},
 {STRINGPREP_BIDI_L_TABLE, ~STRINGPREP_NO_BIDI, stringprep_rfc3454_D_2},
 {0}
}
```

Definition at line 144 of file profiles.c.

5.14.1.6 const Stringprep_profiles stringprep_profiles[ ]

Initial value:

```
{
 {"Nameprep", stringprep_nameprep},
 {"KRBprep", stringprep_kerberos5},
 {"Nodeprep", stringprep_xmpp_nodeprep},
 {"Resourceprep", stringprep_xmpp_resourceprep},
 {"plain", stringprep_plain},
 {"trace", stringprep_trace},
 {"SASLprep", stringprep_saslprep},
 {"ISCSIprep", stringprep_iscsi},
 {"iSCSI", stringprep_iscsi},
 {NULL, NULL}
}
```

Definition at line 24 of file profiles.c.

Referenced by stringprep_profile().

5.14.1.7 const Stringprep_profile stringprep_saslprep[ ]

Initial value:

```
{
 {STRINGPREP_MAP_TABLE, 0, stringprep_saslprep_space_map},
 {STRINGPREP_MAP_TABLE, 0, stringprep_rfc3454_B_1},
 {STRINGPREP_NFKC, 0, 0},
 {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_1_2},
 {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_2_1},
 {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_2_2},
 {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_3},
 {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_4},
 {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_5},
 {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_6},
 {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_7},
 {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_8},
 {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_9},
 {STRINGPREP_BIDI, 0, 0},
 {STRINGPREP_BIDI_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_8},
 {STRINGPREP_BIDI_RAL_TABLE, ~STRINGPREP_NO_BIDI, stringprep_rfc3454_D_1},
 {STRINGPREP_BIDI_L_TABLE, ~STRINGPREP_NO_BIDI, stringprep_rfc3454_D_2},
```

```
  {STRINGPREP_UNASSIGNED_TABLE, ~STRINGPREP_NO_UNASSIGNED,
   stringprep_rfc3454_A_1},
  {0}
}
```

Definition at line 292 of file profiles.c.

5.14.1.8 const Stringprep_table_element stringprep_saslprep_space_map[ ]

Initial value:

```
 {
  {0x0000A0, 0, {0x0020}},
  {0x001680, 0, {0x0020}},
  {0x002000, 0, {0x0020}},
  {0x002001, 0, {0x0020}},
  {0x002002, 0, {0x0020}},
  {0x002003, 0, {0x0020}},
  {0x002004, 0, {0x0020}},
  {0x002005, 0, {0x0020}},
  {0x002006, 0, {0x0020}},
  {0x002007, 0, {0x0020}},
  {0x002008, 0, {0x0020}},
  {0x002009, 0, {0x0020}},
  {0x00200A, 0, {0x0020}},
  {0x00200B, 0, {0x0020}},
  {0x00202F, 0, {0x0020}},
  {0x00205F, 0, {0x0020}},
  {0x003000, 0, {0x0020}},
  {0}
}
```

Definition at line 271 of file profiles.c.

5.14.1.9 const Stringprep_profile stringprep_trace[ ]

Initial value:

```
 {
  {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_2_1},
  {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_2_2},
  {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_3},
  {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_4},
  {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_5},
  {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_6},
  {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_8},
  {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_9},
  {STRINGPREP_BIDI, 0, 0},
  {STRINGPREP_BIDI_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_8},
  {STRINGPREP_BIDI_RAL_TABLE, ~STRINGPREP_NO_BIDI, stringprep_rfc3454_D_1},
  {STRINGPREP_BIDI_L_TABLE, ~STRINGPREP_NO_BIDI, stringprep_rfc3454_D_2},
  {0}
}
```

Definition at line 160 of file profiles.c.

5.14.1.10 const Stringprep_profile stringprep_xmpp_nodeprep[ ]

Initial value:

```
{
 {STRINGPREP_MAP_TABLE, 0, stringprep_rfc3454_B_1},
 {STRINGPREP_MAP_TABLE, 0, stringprep_rfc3454_B_2},
 {STRINGPREP_NFKC, 0, 0},
 {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_1_1},
 {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_1_2},
 {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_2_1},
 {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_2_2},
 {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_3},
 {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_4},
 {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_5},
 {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_6},
 {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_7},
 {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_8},
 {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_9},
 {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_xmpp_nodeprep_prohibit},
 {STRINGPREP_BIDI, 0, 0},
 {STRINGPREP_BIDI_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_8},
 {STRINGPREP_BIDI_RAL_TABLE, 0, stringprep_rfc3454_D_1},
 {STRINGPREP_BIDI_L_TABLE, 0, stringprep_rfc3454_D_2},
 {STRINGPREP_UNASSIGNED_TABLE, ~STRINGPREP_NO_UNASSIGNED,
  stringprep_rfc3454_A_1},
 {0}
}
```

Definition at line 97 of file profiles.c.

5.14.1.11 const Stringprep_table_element stringprep_xmpp_nodeprep_prohibit[ ]

Initial value:

```
{
 {0x000022},
 {0x000026},
 {0x000027},
 {0x00002F},
 {0x00003A},
 {0x00003C},
 {0x00003E},
 {0x000040},
 {0}
}
```

Definition at line 85 of file profiles.c.

5.14.1.12 const Stringprep_profile stringprep_xmpp_resourceprep[ ]

Initial value:

```
{
 {STRINGPREP_MAP_TABLE, 0, stringprep_rfc3454_B_1},
 {STRINGPREP_NFKC, 0, 0},
 {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_1_2},
 {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_2_1},
 {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_2_2},
 {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_3},
 {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_4},
 {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_5},
 {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_6},
 {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_7},
 {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_8},
```

```
  {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_9},
  {STRINGPREP_BIDI, 0, 0},
  {STRINGPREP_BIDI_PROHIBIT_TABLE, 0, stringprep_rfc3454_C_8},
  {STRINGPREP_BIDI_RAL_TABLE, ~STRINGPREP_NO_BIDI, stringprep_rfc3454_D_1},
  {STRINGPREP_BIDI_L_TABLE, ~STRINGPREP_NO_BIDI, stringprep_rfc3454_D_2},
  {STRINGPREP_UNASSIGNED_TABLE, ~STRINGPREP_NO_UNASSIGNED,
   stringprep_rfc3454_A_1},
  {0}
}
```

Definition at line 122 of file profiles.c.

5.15 punycode.c File Reference

```
#include <string.h>
#include "punycode.h"
```

Defines

- #define basic(cp) ((punycode_uint)(cp) < 0x80)
- #define delim(cp) ((cp) == delimiter)
- #define flagged(bcp) ((punycode_uint)(bcp) - 65 < 26)

Enumerations

- enum {

  base = 36, tmin = 1, tmax = 26, skew = 38,

  damp = 700, initial_bias = 72, initial_n = 0x80, delimiter = 0x2D }

Functions

- int punycode_encode (size_t input_length, const punycode_uint input[ ], const unsigned char case_flags[ ], size_t *output_length, char output[ ])
- int punycode_decode (size_t input_length, const char input[ ], size_t *output_length, punycode_uint output[ ], unsigned char case_flags[ ])

5.15.1 Define Documentation

5.15.1.1 #define basic(cp) ((punycode_uint)(cp) < 0x80)

Definition at line 74 of file punycode.c.

Referenced by punycode_decode(), and punycode_encode().

5.15.1.2 #define delim(cp) ((cp) == delimiter)

Definition at line 77 of file punycode.c.

Referenced by punycode_decode().

5.15.1.3 #define flagged(bcp) ((punycode_uint)(bcp) - 65 < 26)

Definition at line 108 of file punycode.c.

Referenced by punycode_decode().

5.15.2 Enumeration Type Documentation

5.15.2.1 anonymous enum

Enumerator:

base

tmin

tmax

skew

damp

initial_bias

initial_n

delimiter

Definition at line 68 of file punycode.c.

5.15.3 Function Documentation

5.15.3.1 int punycode_decode (size_t *input_length*, const char *input*[ ], size_t * *output_length*, punycode_uint *output*[ ], unsigned char *case_flags*[ ])

punycode_decode - decode Punycode to Unicode

Parameters:

input_length The number of ASCII code points in the array.

input An array of ASCII code points (0..7F).

output_length The caller passes in the maximum number of code points that it can receive into the array (which is also the maximum number of flags that it can receive into the array, if is not a NULL pointer). On successful return it will contain the number of code points actually output (which is also the number of flags actually output, if case_flags is not a null pointer). The decoder will never need to output more code points than the number of ASCII code points in the input, because of the way the encoding is defined. The number of code points output cannot exceed the maximum possible value of a punycode_uint, even if the supplied is greater than that.

output An array of code points like the input argument of punycode_encode() (see above).

case_flags A NULL pointer (if the flags are not needed by the caller) or an array of boolean values parallel to the array. Nonzero (true, flagged) suggests that the corresponding Unicode character be forced to uppercase by the caller (if possible), and zero (false, unflagged) suggests that it be forced to lowercase (if possible). ASCII code points (0..7F) are output already in the proper case, but their flags will be set appropriately so that applying the flags would be harmless.

Converts Punycode to a sequence of code points (presumed to be Unicode code points).

Return value: The return value can be any of the Punycode_status values defined above. If not PUNYCODE_SUCCESS, then , , and might contain garbage.

Definition at line 337 of file punycode.c.

References base, basic, delim, flagged, initial_bias, initial_n, punycode_bad_input, punycode_big_output, punycode_overflow, punycode_success, tmax, and tmin.

5.15.3.2 int punycode_encode (size_t *input_length*, const punycode_uint *input*[ ], const unsigned char *case_flags*[ ], size_t * *output_length*, char *output*[ ])

punycode_encode - encode Unicode to Punycode

Parameters:

input_length The number of code points in the array and the number of flags in the array.

input An array of code points. They are presumed to be Unicode code points, but that is not strictly REQUIRED. The array contains code points, not code units. UTF-16 uses code units D800 through DFFF to refer to code points 10000..10FFFF. The code points D800..DFFF do not occur in any valid Unicode string. The code points that can occur in Unicode strings (0..D7FF and E000..10FFFF) are also called Unicode scalar values.

case_flags A NULL pointer or an array of boolean values parallel to the array. Nonzero (true, flagged) suggests that the corresponding Unicode character be forced to uppercase after being decoded (if possible), and zero (false, unflagged) suggests that it be forced to lowercase (if possible). ASCII code points (0..7F) are encoded literally, except that ASCII letters are forced to uppercase or lowercase according to the corresponding case flags. If is a NULL pointer then ASCII letters are left as they are, and other code points are treated as unflagged.

output_length The caller passes in the maximum number of ASCII code points that it can receive. On successful return it will contain the number of ASCII code points actually output.

output An array of ASCII code points. It is *not* null-terminated; it will contain zeros if and only if the contains zeros. (Of course the caller can leave room for a terminator and add one if needed.)

Converts a sequence of code points (presumed to be Unicode code points) to Punycode.

Return value: The return value can be any of the Punycode_status values defined above except PUNYCODE_BAD_INPUT. If not PUNYCODE_SUCCESS, then and might contain garbage.

Definition at line 188 of file punycode.c.

References base, basic, delimiter, initial_bias, initial_n, punycode_big_output, punycode_overflow, punycode_success, tmax, and tmin.

Referenced by idna_to_ascii_4i().

5.16 punycode.h File Reference

```
#include <stddef.h>
#include <idn-int.h>
```

Typedefs

- typedef uint32_t punycode_uint

Enumerations

- enum punycode_status { punycode_success = 0, punycode_bad_input = 1, punycode_big_output = 2, punycode_overflow = 3 }
- enum Punycode_status { PUNYCODE_SUCCESS = punycode_success, PUNYCODE_BAD_INPUT = punycode_bad_input, PUNYCODE_BIG_OUTPUT = punycode_big_output, PUNYCODE_OVERFLOW = punycode_overflow }

Functions

- const char * punycode_strerror (Punycode_status rc)
- int punycode_encode (size_t input_length, const punycode_uint input[ ], const unsigned char case_flags[ ], size_t *output_length, char output[ ])
- int punycode_decode (size_t input_length, const char input[ ], size_t *output_length, punycode_uint output[ ], unsigned char case_flags[ ])

5.16.1 Typedef Documentation

5.16.1.1 typedef uint32_t punycode_uint

Definition at line 94 of file punycode.h.

5.16.2 Enumeration Type Documentation

5.16.2.1 enum Punycode_status

Enumerator:

PUNYCODE_SUCCESS

PUNYCODE_BAD_INPUT

PUNYCODE_BIG_OUTPUT

PUNYCODE_OVERFLOW

Definition at line 81 of file punycode.h.

5.16.2.2 enum punycode_status

Enumerator:

punycode_success

punycode_bad_input

punycode_big_output

punycode_overflow

Definition at line 73 of file punycode.h.

5.16.3 Function Documentation

5.16.3.1 int punycode_decode (size_t *input_length*, const char *input*[ ], size_t * *output_length*, punycode_uint *output*[ ], unsigned char *case_flags*[ ])

punycode_decode - decode Punycode to Unicode

Parameters:

input_length The number of ASCII code points in the array.

input An array of ASCII code points (0..7F).

output_length The caller passes in the maximum number of code points that it can receive into the array (which is also the maximum number of flags that it can receive into the array, if is not a NULL pointer). On successful return it will contain the number of code points actually output (which is also the number of flags actually output, if case_flags is not a null pointer). The decoder will never need to output more code points than the number of ASCII code points in the input, because of the way the encoding is defined. The number of code points output cannot exceed the maximum possible value of a punycode_uint, even if the supplied is greater than that.

output An array of code points like the input argument of punycode_encode() (see above).

case_flags A NULL pointer (if the flags are not needed by the caller) or an array of boolean values parallel to the array. Nonzero (true, flagged) suggests that the corresponding Unicode character be forced to uppercase by the caller (if possible), and zero (false, unflagged) suggests that it be forced to lowercase (if possible). ASCII code points (0..7F) are output already in the proper case, but their flags will be set appropriately so that applying the flags would be harmless.

Converts Punycode to a sequence of code points (presumed to be Unicode code points).

Return value: The return value can be any of the Punycode_status values defined above. If not PUNYCODE_SUCCESS, then , , and might contain garbage.

Definition at line 337 of file punycode.c.

References base, basic, delim, flagged, initial_bias, initial_n, punycode_bad_input, punycode_big_output, punycode_overflow, punycode_success, tmax, and tmin.

5.16.3.2 int punycode_encode (size_t *input_length*, const punycode_uint *input*[ ], const unsigned char *case_flags*[ ], size_t * *output_length*, char *output*[ ])

punycode_encode - encode Unicode to Punycode

Parameters:

input_length The number of code points in the array and the number of flags in the array.

input An array of code points. They are presumed to be Unicode code points, but that is not strictly REQUIRED. The array contains code points, not code units. UTF-16 uses code units D800 through DFFF to refer to code points 10000..10FFFF. The code points D800..DFFF do not occur in any valid Unicode string. The code points that can occur in Unicode strings (0..D7FF and E000..10FFFF) are also called Unicode scalar values.

case_flags A NULL pointer or an array of boolean values parallel to the array. Nonzero (true, flagged) suggests that the corresponding Unicode character be forced to uppercase after being decoded (if possible), and zero (false, unflagged) suggests that it be forced to lowercase (if possible). ASCII code points (0..7F) are encoded literally, except that ASCII letters are forced to uppercase or lowercase according to the corresponding case flags. If is a NULL pointer then ASCII letters are left as they are, and other code points are treated as unflagged.

output_length The caller passes in the maximum number of ASCII code points that it can receive. On successful return it will contain the number of ASCII code points actually output.

output An array of ASCII code points. It is ∗not∗ null-terminated; it will contain zeros if and only if the contains zeros. (Of course the caller can leave room for a terminator and add one if needed.)

Converts a sequence of code points (presumed to be Unicode code points) to Punycode.

Return value: The return value can be any of the Punycode_status values defined above except PUNYCODE_BAD_INPUT. If not PUNYCODE_SUCCESS, then and might contain garbage.

Definition at line 188 of file punycode.c.

References base, basic, delimiter, initial_bias, initial_n, punycode_big_output, punycode_overflow, punycode_success, tmax, and tmin.

Referenced by idna_to_ascii_4i().

5.16.3.3 const char∗ punycode_strerror (Punycode_status *rc*)

punycode_strerror - return string describing punycode error code

Parameters:

rc an Punycode_status return code.

Convert a return code integer to a text string. This string can be used to output a diagnostic message to the user.

PUNYCODE_SUCCESS: Successful operation. This value is guaranteed to always be zero, the remaining ones are only guaranteed to hold non-zero values, for logical comparison purposes. PUNYCODE_BAD_INPUT: Input is invalid. PUNYCODE_BIG_OUTPUT: Output would exceed the space provided. PUNYCODE_OVERFLOW: Input needs wider integers to process.

Return value: Returns a pointer to a statically allocated string containing a description of the error with the return code .

Definition at line 49 of file strerror-punycode.c.

References _, PUNYCODE_BAD_INPUT, PUNYCODE_BIG_OUTPUT, PUNYCODE_OVERFLOW, and PUNYCODE_SUCCESS.

5.17 rfc3454.c File Reference

```
#include "stringprep.h"
```

Variables

- const Stringprep_table_element stringprep_rfc3454_A_1 [ ]
- const Stringprep_table_element stringprep_rfc3454_B_1 [ ]
- const Stringprep_table_element stringprep_rfc3454_B_2 [ ]
- const Stringprep_table_element stringprep_rfc3454_B_3 [ ]
- const Stringprep_table_element stringprep_rfc3454_C_1_1 [ ]
- const Stringprep_table_element stringprep_rfc3454_C_1_2 [ ]
- const Stringprep_table_element stringprep_rfc3454_C_2_1 [ ]
- const Stringprep_table_element stringprep_rfc3454_C_2_2 [ ]
- const Stringprep_table_element stringprep_rfc3454_C_3 [ ]
- const Stringprep_table_element stringprep_rfc3454_C_4 [ ]
- const Stringprep_table_element stringprep_rfc3454_C_5 [ ]
- const Stringprep_table_element stringprep_rfc3454_C_6 [ ]
- const Stringprep_table_element stringprep_rfc3454_C_7 [ ]
- const Stringprep_table_element stringprep_rfc3454_C_8 [ ]
- const Stringprep_table_element stringprep_rfc3454_C_9 [ ]
- const Stringprep_table_element stringprep_rfc3454_D_1 [ ]
- const Stringprep_table_element stringprep_rfc3454_D_2 [ ]

5.17.1 Variable Documentation

5.17.1.1 const Stringprep_table_element stringprep_rfc3454_A_1[ ]

Definition at line 11 of file rfc3454.c.

5.17.1.2 const Stringprep_table_element stringprep_rfc3454_B_1[ ]

Initial value:

```
{
  { 0x0000AD           },
  { 0x00034F           },
  { 0x001806           },
  { 0x00180B           },
  { 0x00180C           },
  { 0x00180D           },
  { 0x00200B           },
  { 0x00200C           },
  { 0x00200D           },
  { 0x002060           },
  { 0x00FE00           },
  { 0x00FE01           },
  { 0x00FE02           },
  { 0x00FE03           },
  { 0x00FE04           },
  { 0x00FE05           },
  { 0x00FE06           },
  { 0x00FE07           },
  { 0x00FE08           },
```

```
  { 0x00FE09            },
  { 0x00FE0A            },
  { 0x00FE0B            },
  { 0x00FE0C            },
  { 0x00FE0D            },
  { 0x00FE0E            },
  { 0x00FE0F            },
  { 0x00FEFF            },
  { 0 },
}
```

Definition at line 417 of file rfc3454.c.

5.17.1.3 const Stringprep_table_element stringprep_rfc3454_B_2[ ]

Definition at line 454 of file rfc3454.c.

5.17.1.4 const Stringprep_table_element stringprep_rfc3454_B_3[ ]

Definition at line 1996 of file rfc3454.c.

5.17.1.5 const Stringprep_table_element stringprep_rfc3454_C_1_1[ ]

Initial value:

```
 {
  { 0x000020            },
  { 0 },
}
```

Definition at line 2947 of file rfc3454.c.

5.17.1.6 const Stringprep_table_element stringprep_rfc3454_C_1_2[ ]

Initial value:

```
 {
  { 0x0000A0            },
  { 0x001680            },
  { 0x002000            },
  { 0x002001            },
  { 0x002002            },
  { 0x002003            },
  { 0x002004            },
  { 0x002005            },
  { 0x002006            },
  { 0x002007            },
  { 0x002008            },
  { 0x002009            },
  { 0x00200A            },
  { 0x00200B            },
  { 0x00202F            },
  { 0x00205F            },
  { 0x003000            },
  { 0 },
}
```

Definition at line 2957 of file rfc3454.c.

5.17.1.7 const Stringprep_table_element stringprep_rfc3454_C_2_1[ ]

Initial value:

```
 {
  { 0x000000, 0x00001F },
  { 0x00007F           },
  { 0 },
}
```

Definition at line 2984 of file rfc3454.c.

5.17.1.8 const Stringprep_table_element stringprep_rfc3454_C_2_2[ ]

Initial value:

```
 {
  { 0x000080, 0x00009F },
  { 0x0006DD           },
  { 0x00070F           },
  { 0x00180E           },
  { 0x00200C           },
  { 0x00200D           },
  { 0x002028           },
  { 0x002029           },
  { 0x002060           },
  { 0x002061           },
  { 0x002062           },
  { 0x002063           },
  { 0x00206A, 0x00206F },
  { 0x00FEFF           },
  { 0x00FFF9, 0x00FFFC },
  { 0x01D173, 0x01D17A },
  { 0 },
}
```

Definition at line 2996 of file rfc3454.c.

5.17.1.9 const Stringprep_table_element stringprep_rfc3454_C_3[ ]

Initial value:

```
 {
  { 0x00E000, 0x00F8FF },
  { 0x0F0000, 0x0FFFFD },
  { 0x100000, 0x10FFFD },
  { 0 },
}
```

Definition at line 3022 of file rfc3454.c.

5.17.1.10 const Stringprep_table_element stringprep_rfc3454_C_4[ ]

Initial value:

```
 {
  { 0x00FDD0, 0x00FDEF },
  { 0x00FFFE, 0x00FFFF },
  { 0x01FFFE, 0x01FFFF },
  { 0x02FFFE, 0x02FFFF },
  { 0x03FFFE, 0x03FFFF },
  { 0x04FFFE, 0x04FFFF },
  { 0x05FFFE, 0x05FFFF },
  { 0x06FFFE, 0x06FFFF },
  { 0x07FFFE, 0x07FFFF },
  { 0x08FFFE, 0x08FFFF },
  { 0x09FFFE, 0x09FFFF },
  { 0x0AFFFE, 0x0AFFFF },
  { 0x0BFFFE, 0x0BFFFF },
  { 0x0CFFFE, 0x0CFFFF },
  { 0x0DFFFE, 0x0DFFFF },
  { 0x0EFFFE, 0x0EFFFF },
  { 0x0FFFFE, 0x0FFFFF },
  { 0x10FFFE, 0x10FFFF },
  { 0 },
}
```

Definition at line 3035 of file rfc3454.c.

5.17.1.11 const Stringprep_table_element stringprep_rfc3454_C_5[ ]

Initial value:

```
 {
  { 0x00D800, 0x00DFFF },
  { 0 },
}
```

Definition at line 3063 of file rfc3454.c.

5.17.1.12 const Stringprep_table_element stringprep_rfc3454_C_6[ ]

Initial value:

```
 {
  { 0x00FFF9          },
  { 0x00FFFA          },
  { 0x00FFFB          },
  { 0x00FFFC          },
  { 0x00FFFD          },
  { 0 },
}
```

Definition at line 3074 of file rfc3454.c.

5.17.1.13 const Stringprep_table_element stringprep_rfc3454_C_7[ ]

Initial value:

```
 {
  { 0x002FF0, 0x002FFB },
  { 0 },
}
```

Definition at line 3089 of file rfc3454.c.

5.17.1.14 const Stringprep_table_element stringprep_rfc3454_C_8[ ]

Initial value:

```
 {
  { 0x000340           },
  { 0x000341           },
  { 0x00200E           },
  { 0x00200F           },
  { 0x00202A           },
  { 0x00202B           },
  { 0x00202C           },
  { 0x00202D           },
  { 0x00202E           },
  { 0x00206A           },
  { 0x00206B           },
  { 0x00206C           },
  { 0x00206D           },
  { 0x00206E           },
  { 0x00206F           },
  { 0 },
}
```

Definition at line 3100 of file rfc3454.c.

5.17.1.15 const Stringprep_table_element stringprep_rfc3454_C_9[ ]

Initial value:

```
 {
  { 0x0E0001           },
  { 0x0E0020, 0x0E007F },
  { 0 },
}
```

Definition at line 3125 of file rfc3454.c.

5.17.1.16 const Stringprep_table_element stringprep_rfc3454_D_1[ ]

Definition at line 3137 of file rfc3454.c.

5.17.1.17 const Stringprep_table_element stringprep_rfc3454_D_2[ ]

Definition at line 3181 of file rfc3454.c.

5.18 strerror-idna.c File Reference

```
#include "idna.h"
#include "gettext.h"
```

Defines

- #define _(String) dgettext (PACKAGE, String)

Functions

- const char * idna_strerror (Idna_rc rc)

5.18.1 Define Documentation

5.18.1.1 #define _(String) dgettext (PACKAGE, String)

Definition at line 29 of file strerror-idna.c.

Referenced by idna_strerror(), pr29_strerror(), punycode_strerror(), stringprep_strerror(), and tld_strerror().

5.18.2 Function Documentation

5.18.2.1 const char* idna_strerror (Idna_rc *rc*)

idna_strerror - return string describing idna error code

Parameters:

rc an Idna_rc return code.

Convert a return code integer to a text string. This string can be used to output a diagnostic message to the user.

IDNA_SUCCESS: Successful operation. This value is guaranteed to always be zero, the remaining ones are only guaranteed to hold non-zero values, for logical comparison purposes. IDNA_STRINGPREP_ERROR: Error during string preparation. IDNA_PUNYCODE_ERROR: Error during punycode operation. IDNA_CONTAINS_NON_LDH: For IDNA_USE_STD3_ASCII_RULES, indicate that the string contains non-LDH ASCII characters. IDNA_CONTAINS_MINUS: For IDNA_USE_STD3_ASCII_RULES, indicate that the string contains a leading or trailing hyphen-minus (U+002D). IDNA_INVALID_LENGTH: The final output string is not within the (inclusive) range 1 to 63 characters. IDNA_NO_ACE_PREFIX: The string does not contain the ACE prefix (for ToUnicode). IDNA_ROUNDTRIP_VERIFY_ERROR: The ToASCII operation on output string does not equal the input. IDNA_CONTAINS_ACE_PREFIX: The input contains the ACE prefix (for ToASCII). IDNA_ICONV_ERROR: Could not convert string in locale encoding. IDNA_MALLOC_ERROR: Could not allocate buffer (this is typically a fatal error). IDNA_DLOPEN_ERROR: Could not dlopen the libcidn DSO (only used internally in libc).

Return value: Returns a pointer to a statically allocated string containing a description of the error with the return code .

Definition at line 65 of file strerror-idna.c.

References _, IDNA_CONTAINS_ACE_PREFIX, IDNA_CONTAINS_MINUS, IDNA_CONTAINS_-NON_LDH, IDNA_DLOPEN_ERROR, IDNA_ICONV_ERROR, IDNA_INVALID_LENGTH, IDNA_-MALLOC_ERROR, IDNA_NO_ACE_PREFIX, IDNA_PUNYCODE_ERROR, IDNA_ROUNDTRIP_-VERIFY_ERROR, IDNA_STRINGPREP_ERROR, and IDNA_SUCCESS.

5.19 strerror-pr29.c File Reference

```
#include "pr29.h"
#include "gettext.h"
```

Defines

- #define _(String) dgettext (PACKAGE, String)

Functions

- const char * pr29_strerror (Pr29_rc rc)

5.19.1 Define Documentation

5.19.1.1 #define _(String) dgettext (PACKAGE, String)

Definition at line 29 of file strerror-pr29.c.

5.19.2 Function Documentation

5.19.2.1 const char* pr29_strerror (Pr29_rc *rc*)

pr29_strerror - return string describing pr29 error code

Parameters:

rc an Pr29_rc return code.

Convert a return code integer to a text string. This string can be used to output a diagnostic message to the user.

PR29_SUCCESS: Successful operation. This value is guaranteed to always be zero, the remaining ones are only guaranteed to hold non-zero values, for logical comparison purposes. PR29_PROBLEM: A problem sequence was encountered. PR29_STRINGPREP_ERROR: The character set conversion failed (only for pr29_8() and pr29_8z()).

Return value: Returns a pointer to a statically allocated string containing a description of the error with the return code .

Definition at line 49 of file strerror-pr29.c.

References _, PR29_PROBLEM, PR29_STRINGPREP_ERROR, and PR29_SUCCESS.

5.20 strerror-punycode.c File Reference

```
#include "punycode.h"
#include "gettext.h"
```

Defines

- #define _(String) dgettext (PACKAGE, String)

Functions

- const char * punycode_strerror (Punycode_status rc)

5.20.1 Define Documentation

5.20.1.1 #define _(String) dgettext (PACKAGE, String)

Definition at line 29 of file strerror-punycode.c.

5.20.2 Function Documentation

5.20.2.1 const char* punycode_strerror (Punycode_status *rc*)

punycode_strerror - return string describing punycode error code

Parameters:

rc an Punycode_status return code.

Convert a return code integer to a text string. This string can be used to output a diagnostic message to the user.

PUNYCODE_SUCCESS: Successful operation. This value is guaranteed to always be zero, the remaining ones are only guaranteed to hold non-zero values, for logical comparison purposes. PUNYCODE_BAD_INPUT: Input is invalid. PUNYCODE_BIG_OUTPUT: Output would exceed the space provided. PUNYCODE_OVERFLOW: Input needs wider integers to process.

Return value: Returns a pointer to a statically allocated string containing a description of the error with the return code .

Definition at line 49 of file strerror-punycode.c.

References _, PUNYCODE_BAD_INPUT, PUNYCODE_BIG_OUTPUT, PUNYCODE_OVERFLOW, and PUNYCODE_SUCCESS.

5.21 strerror-stringprep.c File Reference

```
#include "stringprep.h"
#include "gettext.h"
```

Defines

- #define _(String) dgettext (PACKAGE, String)

Functions

- const char * stringprep_strerror (Stringprep_rc rc)

5.21.1 Define Documentation

5.21.1.1 #define _(String) dgettext (PACKAGE, String)

Definition at line 29 of file strerror-stringprep.c.

5.21.2 Function Documentation

5.21.2.1 const char* stringprep_strerror (Stringprep_rc *rc*)

stringprep_strerror - return string describing stringprep error code

Parameters:

rc a Stringprep_rc return code.

Convert a return code integer to a text string. This string can be used to output a diagnostic message to the user.

STRINGPREP_OK: Successful operation. This value is guaranteed to always be zero, the remaining ones are only guaranteed to hold non-zero values, for logical comparison purposes. STRINGPREP_CONTAINS_UNASSIGNED: String contain unassigned Unicode code points, which is forbidden by the profile. STRINGPREP_CONTAINS_PROHIBITED: String contain code points prohibited by the profile. STRINGPREP_BIDI_BOTH_L_AND_RAL: String contain code points with conflicting bidirection category. STRINGPREP_BIDI_LEADTRAIL_NOT_RAL: Leading and trailing character in string not of proper bidirectional category. STRINGPREP_BIDI_CONTAINS_PROHIBITED: Contains prohibited code points detected by bidirectional code. STRINGPREP_TOO_SMALL_BUFFER: Buffer handed to function was too small. This usually indicate a problem in the calling application. STRINGPREP_PROFILE_ERROR: The stringprep profile was inconsistent. This usually indicate an internal error in the library. STRINGPREP_FLAG_ERROR: The supplied flag conflicted with profile. This usually indicate a problem in the calling application. STRINGPREP_UNKNOWN_PROFILE: The supplied profile name was not known to the library. STRINGPREP_NFKC_FAILED: The Unicode NFKC operation failed. This usually indicate an internal error in the library. STRINGPREP_MALLOC_ERROR: The malloc() was out of memory. This is usually a fatal error.

Return value: Returns a pointer to a statically allocated string containing a description of the error with the return code .

Definition at line 69 of file strerror-stringprep.c.

References _, STRINGPREP_BIDI_BOTH_L_AND_RAL, STRINGPREP_BIDI_CONTAINS_-PROHIBITED, STRINGPREP_BIDI_LEADTRAIL_NOT_RAL, STRINGPREP_CONTAINS_-PROHIBITED, STRINGPREP_CONTAINS_UNASSIGNED, STRINGPREP_FLAG_ERROR, STRINGPREP_MALLOC_ERROR, STRINGPREP_NFKC_FAILED, STRINGPREP_OK, STRINGPREP_PROFILE_ERROR, STRINGPREP_TOO_SMALL_BUFFER, and STRINGPREP_-UNKNOWN_PROFILE.

5.22 strerror-tld.c File Reference

```
#include "tld.h"
#include "gettext.h"
```

Defines

- #define _(String) dgettext (PACKAGE, String)

Functions

- const char * tld_strerror (Tld_rc rc)

5.22.1 Define Documentation

5.22.1.1 #define _(String) dgettext (PACKAGE, String)

Definition at line 29 of file strerror-tld.c.

5.22.2 Function Documentation

5.22.2.1 const char* tld_strerror (Tld_rc *rc*)

tld_strerror - return string describing tld error code

Parameters:

rc tld return code

Convert a return code integer to a text string. This string can be used to output a diagnostic message to the user.

TLD_SUCCESS: Successful operation. This value is guaranteed to always be zero, the remaining ones are only guaranteed to hold non-zero values, for logical comparison purposes. TLD_INVALID: Invalid character found. TLD_NODATA: No input data was provided. TLD_MALLOC_ERROR: Error during memory allocation. TLD_ICONV_ERROR: Error during iconv string conversion. TLD_NO_TLD: No top-level domain found in domain string.

Return value: Returns a pointer to a statically allocated string containing a description of the error with the return code .

Definition at line 51 of file strerror-tld.c.

References _, TLD_ICONV_ERROR, TLD_INVALID, TLD_MALLOC_ERROR, TLD_NO_TLD, TLD_NODATA, and TLD_SUCCESS.

5.23 stringprep.c File Reference

```
#include <stdlib.h>
#include <string.h>
#include "stringprep.h"
```

Defines

- #define INVERTED(x) ((x) & ((~0UL) >> 1))
- #define UNAPPLICAPLEFLAGS(flags, profileflags)

Functions

- int stringprep_4i (uint32_t *ucs4, size_t *len, size_t maxucs4len, Stringprep_profile_flags flags, const Stringprep_profile *profile)
- int stringprep_4zi (uint32_t *ucs4, size_t maxucs4len, Stringprep_profile_flags flags, const Stringprep_profile *profile)
- int stringprep (char *in, size_t maxlen, Stringprep_profile_flags flags, const Stringprep_profile *profile)
- int stringprep_profile (const char *in, char **out, const char *profile, Stringprep_profile_flags flags)

5.23.1 Define Documentation

5.23.1.1 #define INVERTED(x) ((x) & ((~0UL) >> 1))

Definition at line 101 of file stringprep.c.

5.23.1.2 #define UNAPPLICAPLEFLAGS(flags, profileflags)

Value:

```
((!INVERTED(profileflags) && !(profileflags & flags) && profileflags) || \
   ( INVERTED(profileflags) && (profileflags & flags)))
```

Definition at line 102 of file stringprep.c.

Referenced by stringprep_4i().

5.23.2 Function Documentation

5.23.2.1 int stringprep (char * *in*, size_t *maxlen*, Stringprep_profile_flags *flags*, const Stringprep_profile * *profile*)

stringprep - prepare internationalized string

Parameters:

in input/ouput array with string to prepare.

maxlen maximum length of input/output array.

flags a Stringprep_profile_flags value, or 0.

profile pointer to Stringprep_profile to use.

Prepare the input zero terminated UTF-8 string according to the stringprep profile, and write back the result to the input string.

Note that you must convert strings entered in the systems locale into UTF-8 before using this function, see stringprep_locale_to_utf8().

Since the stringprep operation can expand the string, indicate how large the buffer holding the string is. This function will not read or write to characters outside that size.

The are one of Stringprep_profile_flags values, or 0.

The contain the Stringprep_profile instructions to perform. Your application can define new profiles, possibly re-using the generic stringprep tables that always will be part of the library, or use one of the currently supported profiles.

Return value: Returns STRINGPREP_OK iff successful, or an error code.

Definition at line 359 of file stringprep.c.

References stringprep_4i(), STRINGPREP_MALLOC_ERROR, STRINGPREP_OK, STRINGPREP_TOO_SMALL_BUFFER, stringprep_ucs4_to_utf8(), stringprep_utf8_to_ucs4(), and uint32_t.

Referenced by stringprep_profile().

5.23.2.2 int stringprep_4i (uint32_t * *ucs4*, size_t * *len*, size_t *maxucs4len*, Stringprep_profile_flags *flags*, const Stringprep_profile * *profile*)

stringprep_4i - prepare internationalized string

Parameters:

ucs4 input/output array with string to prepare.

len on input, length of input array with Unicode code points, on exit, length of output array with Unicode code points.

maxucs4len maximum length of input/output array.

flags a Stringprep_profile_flags value, or 0.

profile pointer to Stringprep_profile to use.

Prepare the input UCS-4 string according to the stringprep profile, and write back the result to the input string.

The input is not required to be zero terminated ([] = 0). The output will not be zero terminated unless [] = 0. Instead, see stringprep_4zi() if your input is zero terminated or if you want the output to be.

Since the stringprep operation can expand the string, indicate how large the buffer holding the string is. This function will not read or write to code points outside that size.

The are one of Stringprep_profile_flags values, or 0.

The contain the Stringprep_profile instructions to perform. Your application can define new profiles, possibly re-using the generic stringprep tables that always will be part of the library, or use one of the currently supported profiles.

Return value: Returns STRINGPREP_OK iff successful, or an Stringprep_rc error code.

Definition at line 138 of file stringprep.c.

References Stringprep_table::operation, STRINGPREP_BIDI, STRINGPREP_BIDI_BOTH_L_-AND_RAL, STRINGPREP_BIDI_CONTAINS_PROHIBITED, STRINGPREP_BIDI_L_TABLE, STRINGPREP_BIDI_LEADTRAIL_NOT_RAL, STRINGPREP_BIDI_PROHIBIT_TABLE, STRINGPREP_BIDI_RAL_TABLE, STRINGPREP_CONTAINS_PROHIBITED, STRINGPREP_-CONTAINS_UNASSIGNED, STRINGPREP_FLAG_ERROR, STRINGPREP_MAP_TABLE, STRINGPREP_NFKC, STRINGPREP_NFKC_FAILED, STRINGPREP_NO_NFKC, STRINGPREP_-NO_UNASSIGNED, STRINGPREP_OK, STRINGPREP_PROFILE_ERROR, STRINGPREP_-PROHIBIT_TABLE, STRINGPREP_TOO_SMALL_BUFFER, stringprep_ucs4_nfkc_normalize(), STRINGPREP_UNASSIGNED_TABLE, uint32_t, and UNAPPLICAPLEFLAGS.

Referenced by stringprep().

5.23.2.3 int stringprep_4zi (uint32_t * *ucs4*, size_t *maxucs4len*, Stringprep_profile_flags *flags*, const Stringprep_profile * *profile*)

stringprep_4zi - prepare internationalized string

Parameters:

ucs4 input/output array with zero terminated string to prepare.

maxucs4len maximum length of input/output array.

flags a Stringprep_profile_flags value, or 0.

profile pointer to Stringprep_profile to use.

Prepare the input zero terminated UCS-4 string according to the stringprep profile, and write back the result to the input string.

Since the stringprep operation can expand the string, indicate how large the buffer holding the string is. This function will not read or write to code points outside that size.

The are one of Stringprep_profile_flags values, or 0.

The contain the Stringprep_profile instructions to perform. Your application can define new profiles, possibly re-using the generic stringprep tables that always will be part of the library, or use one of the currently supported profiles.

Return value: Returns STRINGPREP_OK iff successful, or an Stringprep_rc error code.

Definition at line 319 of file stringprep.c.

5.23.2.4 int stringprep_profile (const char * *in*, char ** *out*, const char * *profile*, Stringprep_profile_flags *flags*)

stringprep_profile - prepare internationalized string

Parameters:

in input array with UTF-8 string to prepare.

out output variable with pointer to newly allocate string.

profile name of stringprep profile to use.

flags a Stringprep_profile_flags value, or 0.

Prepare the input zero terminated UTF-8 string according to the stringprep profile, and return the result in a newly allocated variable.

Note that you must convert strings entered in the systems locale into UTF-8 before using this function, see stringprep_locale_to_utf8().

The output variable must be deallocated by the caller.

The are one of Stringprep_profile_flags values, or 0.

The specifies the name of the stringprep profile to use. It must be one of the internally supported stringprep profiles.

Return value: Returns STRINGPREP_OK iff successful, or an error code.

Definition at line 438 of file stringprep.c.

References Stringprep_profiles::name, stringprep(), STRINGPREP_MALLOC_ERROR, stringprep_profiles, STRINGPREP_UNKNOWN_PROFILE, and Stringprep_profiles::tables.

5.24 stringprep.h File Reference

```
#include <stddef.h>
#include <unistd.h>
#include <idn-int.h>
```

Data Structures

- struct Stringprep_table_element
- struct Stringprep_table
- struct Stringprep_profiles

Defines

- #define STRINGPREP_VERSION "0.6.7"
- #define STRINGPREP_MAX_MAP_CHARS 4
- #define stringprep_nameprep(in, maxlen) stringprep(in, maxlen, 0, stringprep_nameprep)
- #define stringprep_nameprep_no_unassigned(in, maxlen) stringprep(in, maxlen, STRINGPREP_NO_UNASSIGNED, stringprep_nameprep)
- #define stringprep_plain(in, maxlen) stringprep(in, maxlen, 0, stringprep_plain)
- #define stringprep_kerberos5(in, maxlen) stringprep(in, maxlen, 0, stringprep_kerberos5)
- #define stringprep_xmpp_nodeprep(in, maxlen) stringprep(in, maxlen, 0, stringprep_xmpp_nodeprep)
- #define stringprep_xmpp_resourceprep(in, maxlen) stringprep(in, maxlen, 0, stringprep_xmpp_resourceprep)
- #define stringprep_iscsi(in, maxlen) stringprep(in, maxlen, 0, stringprep_iscsi)

Typedefs

- typedef Stringprep_table_element Stringprep_table_element
- typedef Stringprep_table Stringprep_profile
- typedef Stringprep_profiles Stringprep_profiles

Enumerations

- enum Stringprep_rc {

  STRINGPREP_OK = 0, STRINGPREP_CONTAINS_UNASSIGNED = 1, STRINGPREP_CONTAINS_PROHIBITED = 2, STRINGPREP_BIDI_BOTH_L_AND_RAL = 3,

  STRINGPREP_BIDI_LEADTRAIL_NOT_RAL = 4, STRINGPREP_BIDI_CONTAINS_PROHIBITED = 5, STRINGPREP_TOO_SMALL_BUFFER = 100, STRINGPREP_PROFILE_ERROR = 101,

  STRINGPREP_FLAG_ERROR = 102, STRINGPREP_UNKNOWN_PROFILE = 103, STRINGPREP_NFKC_FAILED = 200, STRINGPREP_MALLOC_ERROR = 201 }
- enum Stringprep_profile_flags { STRINGPREP_NO_NFKC = 1, STRINGPREP_NO_BIDI = 2, STRINGPREP_NO_UNASSIGNED = 4 }

- enum Stringprep_profile_steps {

  STRINGPREP_NFKC = 1, STRINGPREP_BIDI = 2, STRINGPREP_MAP_TABLE = 3, STRINGPREP_UNASSIGNED_TABLE = 4,

  STRINGPREP_PROHIBIT_TABLE = 5, STRINGPREP_BIDI_PROHIBIT_TABLE = 6, STRINGPREP_BIDI_RAL_TABLE = 7, STRINGPREP_BIDI_L_TABLE = 8 }

Functions

- int stringprep_4i (uint32_t *ucs4, size_t *len, size_t maxucs4len, Stringprep_profile_flags flags, const Stringprep_profile *profile)
- int stringprep_4zi (uint32_t *ucs4, size_t maxucs4len, Stringprep_profile_flags flags, const Stringprep_profile *profile)
- int stringprep (char *in, size_t maxlen, Stringprep_profile_flags flags, const Stringprep_profile *profile)
- int stringprep_profile (const char *in, char **out, const char *profile, Stringprep_profile_flags flags)
- const char * stringprep_strerror (Stringprep_rc rc)
- const char * stringprep_check_version (const char *req_version)
- int stringprep_unichar_to_utf8 (uint32_t c, char *outbuf)
- uint32_t stringprep_utf8_to_unichar (const char *p)
- uint32_t * stringprep_utf8_to_ucs4 (const char *str, ssize_t len, size_t *items_written)
- char * stringprep_ucs4_to_utf8 (const uint32_t *str, ssize_t len, size_t *items_read, size_t *items_written)
- char * stringprep_utf8_nfkc_normalize (const char *str, ssize_t len)
- uint32_t * stringprep_ucs4_nfkc_normalize (uint32_t *str, ssize_t len)
- const char * stringprep_locale_charset (void)
- char * stringprep_convert (const char *str, const char *to_codeset, const char *from_codeset)
- char * stringprep_locale_to_utf8 (const char *str)
- char * stringprep_utf8_to_locale (const char *str)

Variables

- const Stringprep_profiles stringprep_profiles [ ]
- const Stringprep_table_element stringprep_rfc3454_A_1 [ ]
- const Stringprep_table_element stringprep_rfc3454_B_1 [ ]
- const Stringprep_table_element stringprep_rfc3454_B_2 [ ]
- const Stringprep_table_element stringprep_rfc3454_B_3 [ ]
- const Stringprep_table_element stringprep_rfc3454_C_1_1 [ ]
- const Stringprep_table_element stringprep_rfc3454_C_1_2 [ ]
- const Stringprep_table_element stringprep_rfc3454_C_2_1 [ ]
- const Stringprep_table_element stringprep_rfc3454_C_2_2 [ ]
- const Stringprep_table_element stringprep_rfc3454_C_3 [ ]
- const Stringprep_table_element stringprep_rfc3454_C_4 [ ]
- const Stringprep_table_element stringprep_rfc3454_C_5 [ ]
- const Stringprep_table_element stringprep_rfc3454_C_6 [ ]
- const Stringprep_table_element stringprep_rfc3454_C_7 [ ]
- const Stringprep_table_element stringprep_rfc3454_C_8 [ ]
- const Stringprep_table_element stringprep_rfc3454_C_9 [ ]
- const Stringprep_table_element stringprep_rfc3454_D_1 [ ]
- const Stringprep_table_element stringprep_rfc3454_D_2 [ ]

- const Stringprep_profile stringprep_nameprep [ ]
- const Stringprep_profile stringprep_saslprep [ ]
- const Stringprep_profile stringprep_plain [ ]
- const Stringprep_profile stringprep_trace [ ]
- const Stringprep_profile stringprep_kerberos5 [ ]
- const Stringprep_profile stringprep_xmpp_nodeprep [ ]
- const Stringprep_profile stringprep_xmpp_resourceprep [ ]
- const Stringprep_table_element stringprep_xmpp_nodeprep_prohibit [ ]
- const Stringprep_profile stringprep_iscsi [ ]

5.24.1 Define Documentation

5.24.1.1 #define stringprep_iscsi(in, maxlen) stringprep(in, maxlen, 0, stringprep_iscsi)

Definition at line 164 of file stringprep.h.

5.24.1.2 #define stringprep_kerberos5(in, maxlen) stringprep(in, maxlen, 0, stringprep_kerberos5)

Definition at line 146 of file stringprep.h.

5.24.1.3 #define STRINGPREP_MAX_MAP_CHARS 4

Definition at line 77 of file stringprep.h.

5.24.1.4 #define stringprep_nameprep(in, maxlen) stringprep(in, maxlen, 0, stringprep_nameprep)

Definition at line 127 of file stringprep.h.

Referenced by idna_to_ascii_4i().

5.24.1.5 #define stringprep_nameprep_no_unassigned(in, maxlen) stringprep(in, maxlen, STRINGPREP_NO_UNASSIGNED, stringprep_nameprep)

Definition at line 130 of file stringprep.h.

Referenced by idna_to_ascii_4i().

5.24.1.6 #define stringprep_plain(in, maxlen) stringprep(in, maxlen, 0, stringprep_plain)

Definition at line 139 of file stringprep.h.

5.24.1.7 #define STRINGPREP_VERSION "0.6.7"

Definition at line 34 of file stringprep.h.

5.24.1.8 #define stringprep_xmpp_nodeprep(in, maxlen) stringprep(in, maxlen, 0, stringprep_xmpp_nodeprep)

Definition at line 155 of file stringprep.h.

5.24.1.9 #define stringprep_xmpp_resourceprep(in, maxlen) stringprep(in, maxlen, 0, stringprep_xmpp_resourceprep)

Definition at line 157 of file stringprep.h.

5.24.2 Typedef Documentation

5.24.2.1 typedef struct Stringprep_table Stringprep_profile

Definition at line 93 of file stringprep.h.

5.24.2.2 typedef struct Stringprep_profiles Stringprep_profiles

Definition at line 100 of file stringprep.h.

5.24.2.3 typedef struct Stringprep_table_element Stringprep_table_element

Definition at line 85 of file stringprep.h.

5.24.3 Enumeration Type Documentation

5.24.3.1 enum Stringprep_profile_flags

Enumerator:

STRINGPREP_NO_NFKC

STRINGPREP_NO_BIDI

STRINGPREP_NO_UNASSIGNED

Definition at line 57 of file stringprep.h.

5.24.3.2 enum Stringprep_profile_steps

Enumerator:

STRINGPREP_NFKC

STRINGPREP_BIDI

STRINGPREP_MAP_TABLE

STRINGPREP_UNASSIGNED_TABLE

STRINGPREP_PROHIBIT_TABLE

STRINGPREP_BIDI_PROHIBIT_TABLE

STRINGPREP_BIDI_RAL_TABLE

STRINGPREP_BIDI_L_TABLE

Definition at line 65 of file stringprep.h.

5.24.3.3 enum Stringprep_rc

Enumerator:

STRINGPREP_OK
STRINGPREP_CONTAINS_UNASSIGNED
STRINGPREP_CONTAINS_PROHIBITED
STRINGPREP_BIDI_BOTH_L_AND_RAL
STRINGPREP_BIDI_LEADTRAIL_NOT_RAL
STRINGPREP_BIDI_CONTAINS_PROHIBITED
STRINGPREP_TOO_SMALL_BUFFER
STRINGPREP_PROFILE_ERROR
STRINGPREP_FLAG_ERROR
STRINGPREP_UNKNOWN_PROFILE
STRINGPREP_NFKC_FAILED
STRINGPREP_MALLOC_ERROR

Definition at line 37 of file stringprep.h.

5.24.4 Function Documentation

5.24.4.1 int stringprep (char * *in*, size_t *maxlen*, Stringprep_profile_flags *flags*, const Stringprep_profile * *profile*)

stringprep - prepare internationalized string

Parameters:

in input/ouput array with string to prepare.
maxlen maximum length of input/output array.
flags a Stringprep_profile_flags value, or 0.
profile pointer to Stringprep_profile to use.

Prepare the input zero terminated UTF-8 string according to the stringprep profile, and write back the result to the input string.

Note that you must convert strings entered in the systems locale into UTF-8 before using this function, see stringprep_locale_to_utf8().

Since the stringprep operation can expand the string, indicate how large the buffer holding the string is. This function will not read or write to characters outside that size.

The are one of Stringprep_profile_flags values, or 0.

The contain the Stringprep_profile instructions to perform. Your application can define new profiles, possibly re-using the generic stringprep tables that always will be part of the library, or use one of the currently supported profiles.

Return value: Returns STRINGPREP_OK iff successful, or an error code.

Definition at line 359 of file stringprep.c.

References stringprep_4i(), STRINGPREP_MALLOC_ERROR, STRINGPREP_OK, STRINGPREP_TOO_SMALL_BUFFER, stringprep_ucs4_to_utf8(), stringprep_utf8_to_ucs4(), and uint32_t.

Referenced by stringprep_profile().

5.24.4.2 int stringprep_4i (uint32_t * *ucs4*, size_t * *len*, size_t *maxucs4len*, Stringprep_profile_flags *flags*, const Stringprep_profile * *profile*)

stringprep_4i - prepare internationalized string

Parameters:

ucs4 input/output array with string to prepare.

len on input, length of input array with Unicode code points, on exit, length of output array with Unicode code points.

maxucs4len maximum length of input/output array.

flags a Stringprep_profile_flags value, or 0.

profile pointer to Stringprep_profile to use.

Prepare the input UCS-4 string according to the stringprep profile, and write back the result to the input string.

The input is not required to be zero terminated ([] = 0). The output will not be zero terminated unless [] = 0. Instead, see stringprep_4zi() if your input is zero terminated or if you want the output to be.

Since the stringprep operation can expand the string, indicate how large the buffer holding the string is. This function will not read or write to code points outside that size.

The are one of Stringprep_profile_flags values, or 0.

The contain the Stringprep_profile instructions to perform. Your application can define new profiles, possibly re-using the generic stringprep tables that always will be part of the library, or use one of the currently supported profiles.

Return value: Returns STRINGPREP_OK iff successful, or an Stringprep_rc error code.

Definition at line 138 of file stringprep.c.

References Stringprep_table::operation, STRINGPREP_BIDI, STRINGPREP_BIDI_BOTH_L_-AND_RAL, STRINGPREP_BIDI_CONTAINS_PROHIBITED, STRINGPREP_BIDI_L_TABLE, STRINGPREP_BIDI_LEADTRAIL_NOT_RAL, STRINGPREP_BIDI_PROHIBIT_TABLE, STRINGPREP_BIDI_RAL_TABLE, STRINGPREP_CONTAINS_PROHIBITED, STRINGPREP_-CONTAINS_UNASSIGNED, STRINGPREP_FLAG_ERROR, STRINGPREP_MAP_TABLE, STRINGPREP_NFKC, STRINGPREP_NFKC_FAILED, STRINGPREP_NO_NFKC, STRINGPREP_-NO_UNASSIGNED, STRINGPREP_OK, STRINGPREP_PROFILE_ERROR, STRINGPREP_-PROHIBIT_TABLE, STRINGPREP_TOO_SMALL_BUFFER, stringprep_ucs4_nfkc_normalize(), STRINGPREP_UNASSIGNED_TABLE, uint32_t, and UNAPPLICAPLEFLAGS.

Referenced by stringprep().

5.24.4.3 int stringprep_4zi (uint32_t * *ucs4*, size_t *maxucs4len*, Stringprep_profile_flags *flags*, const Stringprep_profile * *profile*)

stringprep_4zi - prepare internationalized string

Parameters:

ucs4 input/output array with zero terminated string to prepare.

maxucs4len maximum length of input/output array.

flags a Stringprep_profile_flags value, or 0.

profile pointer to Stringprep_profile to use.

Prepare the input zero terminated UCS-4 string according to the stringprep profile, and write back the result to the input string.

Since the stringprep operation can expand the string, indicate how large the buffer holding the string is. This function will not read or write to code points outside that size.

The are one of Stringprep_profile_flags values, or 0.

The contain the Stringprep_profile instructions to perform. Your application can define new profiles, possibly re-using the generic stringprep tables that always will be part of the library, or use one of the currently supported profiles.

Return value: Returns STRINGPREP_OK iff successful, or an Stringprep_rc error code.

Definition at line 319 of file stringprep.c.

5.24.4.4 const char* stringprep_check_version (const char * *req_version*)

stringprep_check_version - check for library version

Parameters:

req_version Required version number, or NULL.

Check that the the version of the library is at minimum the requested one and return the version string; return NULL if the condition is not satisfied. If a NULL is passed to this function, no check is done, but the version string is simply returned.

See STRINGPREP_VERSION for a suitable string.

Return value: Version string of run-time library, or NULL if the run-time library does not meet the required version number.

Definition at line 81 of file version.c.

5.24.4.5 char* stringprep_convert (const char * *str*, const char * *to_codeset*, const char * *from_codeset*)

stringprep_convert - encode string using new character set

Parameters:

str input zero-terminated string.

to_codeset name of destination character set.

from_codeset name of origin character set, as used by .

Convert the string from one character set to another using the system's iconv() function.

Return value: Returns newly allocated zero-terminated string which is transcoded into to_codeset.

Definition at line 110 of file toutf8.c.

References iconv_string().

Referenced by stringprep_locale_to_utf8(), and stringprep_utf8_to_locale().

5.24.4.6 const char* stringprep_locale_charset (void)

stringprep_locale_charset - return charset used in current locale

Find out current locale charset. The function respect the CHARSET environment variable, but typically uses nl_langinfo(CODESET) when it is supported. It fall back on "ASCII" if CHARSET isn't set and nl_langinfo isn't supported or return anything.

Note that this function return the application's locale's preferred charset (or thread's locale's preffered charset, if your system support thread-specific locales). It does not return what the system may be using. Thus, if you receive data from external sources you cannot in general use this function to guess what charset it is encoded in. Use stringprep_convert from the external representation into the charset returned by this function, to have data in the locale encoding.

Return value: Return the character set used by the current locale. It will never return NULL, but use "ASCII" as a fallback.

Definition at line 79 of file toutf8.c.

Referenced by stringprep_locale_to_utf8(), and stringprep_utf8_to_locale().

5.24.4.7 char* stringprep_locale_to_utf8 (const char * *str*)

stringprep_locale_to_utf8 - convert locale encoded string to UTF-8

Parameters:

str input zero terminated string.

Convert string encoded in the locale's character set into UTF-8 by using stringprep_convert().

Return value: Returns newly allocated zero-terminated string which is transcoded into UTF-8.

Definition at line 137 of file toutf8.c.

References stringprep_convert(), and stringprep_locale_charset().

Referenced by idna_to_ascii_lz(), idna_to_unicode_lzlz(), and tld_check_lz().

5.24.4.8 int stringprep_profile (const char * *in*, char ** *out*, const char * *profile*, Stringprep_profile_flags *flags*)

stringprep_profile - prepare internationalized string

Parameters:

in input array with UTF-8 string to prepare.

out output variable with pointer to newly allocate string.

profile name of stringprep profile to use.

flags a Stringprep_profile_flags value, or 0.

Prepare the input zero terminated UTF-8 string according to the stringprep profile, and return the result in a newly allocated variable.

Note that you must convert strings entered in the systems locale into UTF-8 before using this function, see stringprep_locale_to_utf8().

The output variable must be deallocated by the caller.

The are one of Stringprep_profile_flags values, or 0.

The specifies the name of the stringprep profile to use. It must be one of the internally supported stringprep profiles.

Return value: Returns STRINGPREP_OK iff successful, or an error code.

Definition at line 438 of file stringprep.c.

References Stringprep_profiles::name, stringprep(), STRINGPREP_MALLOC_ERROR, stringprep_profiles, STRINGPREP_UNKNOWN_PROFILE, and Stringprep_profiles::tables.

5.24.4.9 const char* stringprep_strerror (Stringprep_rc *rc*)

stringprep_strerror - return string describing stringprep error code

Parameters:

rc a Stringprep_rc return code.

Convert a return code integer to a text string. This string can be used to output a diagnostic message to the user.

STRINGPREP_OK: Successful operation. This value is guaranteed to always be zero, the remaining ones are only guaranteed to hold non-zero values, for logical comparison purposes. STRINGPREP_CONTAINS_UNASSIGNED: String contain unassigned Unicode code points, which is forbidden by the profile. STRINGPREP_CONTAINS_PROHIBITED: String contain code points prohibited by the profile. STRINGPREP_BIDI_BOTH_L_AND_RAL: String contain code points with conflicting bidirection category. STRINGPREP_BIDI_LEADTRAIL_NOT_RAL: Leading and trailing character in string not of proper bidirectional category. STRINGPREP_BIDI_CONTAINS_PROHIBITED: Contains prohibited code points detected by bidirectional code. STRINGPREP_TOO_SMALL_BUFFER: Buffer handed to function was too small. This usually indicate a problem in the calling application. STRINGPREP_PROFILE_ERROR: The stringprep profile was inconsistent. This usually indicate an internal error in the library. STRINGPREP_FLAG_ERROR: The supplied flag conflicted with profile. This usually indicate a problem in the calling application. STRINGPREP_UNKNOWN_PROFILE: The supplied profile name was not known to the library. STRINGPREP_NFKC_FAILED: The Unicode NFKC operation failed. This usually indicate an internal error in the library. STRINGPREP_MALLOC_ERROR: The malloc() was out of memory. This is usually a fatal error.

Return value: Returns a pointer to a statically allocated string containing a description of the error with the return code .

Definition at line 69 of file strerror-stringprep.c.

References _, STRINGPREP_BIDI_BOTH_L_AND_RAL, STRINGPREP_BIDI_CONTAINS_PROHIBITED, STRINGPREP_BIDI_LEADTRAIL_NOT_RAL, STRINGPREP_CONTAINS_PROHIBITED, STRINGPREP_CONTAINS_UNASSIGNED, STRINGPREP_FLAG_ERROR, STRINGPREP_MALLOC_ERROR, STRINGPREP_NFKC_FAILED, STRINGPREP_OK, STRINGPREP_PROFILE_ERROR, STRINGPREP_TOO_SMALL_BUFFER, and STRINGPREP_UNKNOWN_PROFILE.

5.24.4.10 uint32_t* stringprep_ucs4_nfkc_normalize (uint32_t * *str*, ssize_t *len*)

stringprep_ucs4_nfkc_normalize - normalize Unicode string

Parameters:

str a Unicode string.

len length of array, or -1 if is nul-terminated.

Converts UCS4 string into UTF-8 and runs stringprep_utf8_nfkc_normalize().

Return value: a newly allocated Unicode string, that is the NFKC normalized form of .

Definition at line 1048 of file nfkc.c.

References G_NORMALIZE_NFKC, stringprep_ucs4_to_utf8(), and uint32_t.

Referenced by stringprep_4i().

5.24.4.11 char* stringprep_ucs4_to_utf8 (const uint32_t * *str*, ssize_t *len*, size_t * *items_read*, size_t * *items_written*)

stringprep_ucs4_to_utf8 - convert UCS-4 string to UTF-8

Parameters:

str a UCS-4 encoded string

len the maximum length of to use. If < 0, then the string is terminated with a 0 character.

items_read location to store number of characters read read, or NULL.

items_written location to store number of bytes written or NULL. The value here stored does not include the trailing 0 byte.

Convert a string from a 32-bit fixed width representation as UCS-4. to UTF-8. The result will be terminated with a 0 byte.

Return value: a pointer to a newly allocated UTF-8 string. This value must be freed with free(). If an error occurs, NULL will be returned and set.

Definition at line 1001 of file nfkc.c.

References glong.

Referenced by idna_to_ascii_4i(), idna_to_unicode_44i(), idna_to_unicode_8z8z(), stringprep(), and stringprep_ucs4_nfkc_normalize().

5.24.4.12 int stringprep_unichar_to_utf8 (uint32_t *c*, char * *outbuf*)

stringprep_unichar_to_utf8 - convert Unicode code point to UTF-8

Parameters:

c a ISO10646 character code

outbuf output buffer, must have at least 6 bytes of space. If NULL, the length will be computed and returned and nothing will be written to .

Converts a single character to UTF-8.

Return value: number of bytes written.

Definition at line 956 of file nfkc.c.

5.24.4.13 char* stringprep_utf8_nfkc_normalize (const char * *str*, ssize_t *len*)

stringprep_utf8_nfkc_normalize - normalize Unicode string

Parameters:

str a UTF-8 encoded string.

len length of , in bytes, or -1 if is nul-terminated.

Converts a string into canonical form, standardizing such issues as whether a character with an accent is represented as a base character and combining accent or as a single precomposed character.

The normalization mode is NFKC (ALL COMPOSE). It standardizes differences that do not affect the text content, such as the above-mentioned accent representation. It standardizes the "compatibility" characters in Unicode, such as SUPERSCRIPT THREE to the standard forms (in this case DIGIT THREE). Formatting information may be lost but for most text operations such characters should be considered the same. It returns a result with composed forms rather than a maximally decomposed form.

Return value: a newly allocated string, that is the NFKC normalized form of .

Definition at line 1031 of file nfkc.c.

References G_NORMALIZE_NFKC.

5.24.4.14 char* stringprep_utf8_to_locale (const char * *str*)

stringprep_utf8_to_locale - encode UTF-8 string to locale encoding

Parameters:

str input zero terminated string.

Convert string encoded in UTF-8 into the locale's character set by using stringprep_convert().

Return value: Returns newly allocated zero-terminated string which is transcoded into the locale's character set.

Definition at line 153 of file toutf8.c.

References stringprep_convert(), and stringprep_locale_charset().

Referenced by idna_to_unicode_8zlz().

5.24.4.15 uint32_t* stringprep_utf8_to_ucs4 (const char * *str*, ssize_t *len*, size_t * *items_written*)

stringprep_utf8_to_ucs4 - convert UTF-8 string to UCS-4

Parameters:

str a UTF-8 encoded string

len the maximum length of to use. If < 0, then the string is nul-terminated.

items_written location to store the number of characters in the result, or NULL.

Convert a string from UTF-8 to a 32-bit fixed width representation as UCS-4, assuming valid UTF-8 input. This function does no error checking on the input.

Return value: a pointer to a newly allocated UCS-4 string. This value must be freed with free().

Definition at line 977 of file nfkc.c.

References glong.

Referenced by idna_to_ascii_4i(), idna_to_ascii_8z(), idna_to_unicode_8z4z(), pr29_8z(), stringprep(), and tld_check_8z().

5.24.4.16 uint32_t stringprep_utf8_to_unichar (const char * p)

stringprep_utf8_to_unichar - convert UTF-8 to Unicode code point

Parameters:

p a pointer to Unicode character encoded as UTF-8

Converts a sequence of bytes encoded as UTF-8 to a Unicode character. If `does` not point to a valid UTF-8 encoded character, results are undefined.

Return value: the resulting character.

Definition at line 939 of file nfkc.c.

5.24.5 Variable Documentation

5.24.5.1 const Stringprep_profile stringprep_iscsi[ ]

Definition at line 246 of file profiles.c.

5.24.5.2 const Stringprep_profile stringprep_kerberos5[ ]

Definition at line 60 of file profiles.c.

5.24.5.3 const Stringprep_profile stringprep_nameprep[ ]

Definition at line 37 of file profiles.c.

5.24.5.4 const Stringprep_profile stringprep_plain[ ]

Definition at line 144 of file profiles.c.

5.24.5.5 const Stringprep_profiles stringprep_profiles[ ]

Definition at line 24 of file profiles.c.

Referenced by stringprep_profile().

5.24.5.6 const Stringprep_table_element stringprep_rfc3454_A_1[ ]

Definition at line 11 of file rfc3454.c.

5.24.5.7 const Stringprep_table_element stringprep_rfc3454_B_1[ ]

Definition at line 417 of file rfc3454.c.

5.24.5.8 const Stringprep_table_element stringprep_rfc3454_B_2[ ]

Definition at line 454 of file rfc3454.c.

5.24.5.9 const Stringprep_table_element stringprep_rfc3454_B_3[ ]

Definition at line 1996 of file rfc3454.c.

5.24.5.10 const Stringprep_table_element stringprep_rfc3454_C_1_1[ ]

Definition at line 2947 of file rfc3454.c.

5.24.5.11 const Stringprep_table_element stringprep_rfc3454_C_1_2[ ]

Definition at line 2957 of file rfc3454.c.

5.24.5.12 const Stringprep_table_element stringprep_rfc3454_C_2_1[ ]

Definition at line 2984 of file rfc3454.c.

5.24.5.13 const Stringprep_table_element stringprep_rfc3454_C_2_2[ ]

Definition at line 2996 of file rfc3454.c.

5.24.5.14 const Stringprep_table_element stringprep_rfc3454_C_3[ ]

Definition at line 3022 of file rfc3454.c.

5.24.5.15 const Stringprep_table_element stringprep_rfc3454_C_4[ ]

Definition at line 3035 of file rfc3454.c.

5.24.5.16 const Stringprep_table_element stringprep_rfc3454_C_5[ ]

Definition at line 3063 of file rfc3454.c.

5.24.5.17 const Stringprep_table_element stringprep_rfc3454_C_6[ ]

Definition at line 3074 of file rfc3454.c.

5.24.5.18 const Stringprep_table_element stringprep_rfc3454_C_7[ ]

Definition at line 3089 of file rfc3454.c.

5.24.5.19 const Stringprep_table_element stringprep_rfc3454_C_8[ ]

Definition at line 3100 of file rfc3454.c.

5.24.5.20 const Stringprep_table_element stringprep_rfc3454_C_9[ ]

Definition at line 3125 of file rfc3454.c.

5.24.5.21 const Stringprep_table_element stringprep_rfc3454_D_1[ ]

Definition at line 3137 of file rfc3454.c.

5.24.5.22 const Stringprep_table_element stringprep_rfc3454_D_2[ ]

Definition at line 3181 of file rfc3454.c.

5.24.5.23 const Stringprep_profile stringprep_saslprep[ ]

Definition at line 292 of file profiles.c.

5.24.5.24 const Stringprep_profile stringprep_trace[ ]

Definition at line 160 of file profiles.c.

5.24.5.25 const Stringprep_profile stringprep_xmpp_nodeprep[ ]

Definition at line 97 of file profiles.c.

5.24.5.26 const Stringprep_table_element stringprep_xmpp_nodeprep_prohibit[ ]

Definition at line 85 of file profiles.c.

5.24.5.27 const Stringprep_profile stringprep_xmpp_resourceprep[ ]

Definition at line 122 of file profiles.c.

5.25 tld.c File Reference

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

Defines

- #define DOTP(c)

Functions

- const Tld_table * tld_get_table (const char *tld, const Tld_table **tables)
- const Tld_table * tld_default_table (const char *tld, const Tld_table **overrides)
- int tld_get_4 (const uint32_t *in, size_t inlen, char **out)
- int tld_get_4z (const uint32_t *in, char **out)
- int tld_get_z (const char *in, char **out)
- int tld_check_4t (const uint32_t *in, size_t inlen, size_t *errpos, const Tld_table *tld)
- int tld_check_4tz (const uint32_t *in, size_t *errpos, const Tld_table *tld)
- int tld_check_4 (const uint32_t *in, size_t inlen, size_t *errpos, const Tld_table **overrides)
- int tld_check_4z (const uint32_t *in, size_t *errpos, const Tld_table **overrides)
- int tld_check_8z (const char *in, size_t *errpos, const Tld_table **overrides)
- int tld_check_lz (const char *in, size_t *errpos, const Tld_table **overrides)

Variables

- const Tld_table * _tld_tables [ ]

5.25.1 Define Documentation

5.25.1.1 #define DOTP(c)

Value:

```
((c) == 0x002E || (c) == 0x3002 ||       \
                  (c) == 0xFF0E || (c) == 0xFF61)
```

Definition at line 95 of file tld.c.

5.25.2 Function Documentation

5.25.2.1 int tld_check_4 (const uint32_t * *in*, size_t *inlen*, size_t * *errpos*, const Tld_table ** *overrides*)

tld_check_4 - verify that characters are permitted

Parameters:

in Array of unicode code points to process. Does not need to be zero terminated.

inlen Number of unicode code points.

errpos Position of offending character is returned here.

overrides A Tld_table array of additional domain restriction structures that complement and supersede the built-in information.

Test each of the code points in for whether or not they are allowed by the information in or by the built-in TLD restriction data. When data for the same TLD is available both internally and in , the information in takes precedence. If several entries for a specific TLD are found, the first one is used. If is NULL, only the built-in information is used. The position of the first offending character is returned in .

Return value: Returns the Tld_rc value TLD_SUCCESS if all code points are valid or when is null, TLD_-INVALID if a character is not allowed, or additional error codes on general failure conditions.

Definition at line 348 of file tld.c.

References tld_check_4t(), tld_default_table(), tld_get_4(), TLD_NO_TLD, and TLD_SUCCESS.

Referenced by tld_check_4z(), and tld_check_8z().

5.25.2.2 int tld_check_4t (const uint32_t * *in*, size_t *inlen*, size_t * *errpos*, const Tld_table * *tld*)

tld_check_4t - verify that characters are permitted

Parameters:

in Array of unicode code points to process. Does not need to be zero terminated.

inlen Number of unicode code points.

errpos Position of offending character is returned here.

tld A Tld_table data structure representing the restrictions for which the input should be tested.

Test each of the code points in for whether or not they are allowed by the data structure in , return the position of the first character for which this is not the case in .

Return value: Returns the Tld_rc value TLD_SUCCESS if all code points are valid or when is null, TLD_-INVALID if a character is not allowed, or additional error codes on general failure conditions.

Definition at line 269 of file tld.c.

References TLD_SUCCESS, and uint32_t.

Referenced by tld_check_4(), and tld_check_4tz().

5.25.2.3 int tld_check_4tz (const uint32_t * *in*, size_t * *errpos*, const Tld_table * *tld*)

tld_check_4tz - verify that characters are permitted

Parameters:

in Zero terminated array of unicode code points to process.

errpos Position of offending character is returned here.

tld A Tld_table data structure representing the restrictions for which the input should be tested.

Test each of the code points in for whether or not they are allowed by the data structure in , return the position of the first character for which this is not the case in .

Return value: Returns the Tld_rc value TLD_SUCCESS if all code points are valid or when is null, TLD_-INVALID if a character is not allowed, or additional error codes on general failure conditions.

Definition at line 311 of file tld.c.

References tld_check_4t(), TLD_NODATA, and uint32_t.

5.25.2.4 int tld_check_4z (const uint32_t * *in*, size_t * *errpos*, const Tld_table ** *overrides*)

tld_check_4z - verify that characters are permitted

Parameters:

in Zero-terminated array of unicode code points to process.

errpos Position of offending character is returned here.

overrides A Tld_table array of additional domain restriction structures that complement and supersede the built-in information.

Test each of the code points in for whether or not they are allowed by the information in or by the built-in TLD restriction data. When data for the same TLD is available both internally and in , the information in takes precedence. If several entries for a specific TLD are found, the first one is used. If is NULL, only the built-in information is used. The position of the first offending character is returned in .

Return value: Returns the Tld_rc value TLD_SUCCESS if all code points are valid or when is null, TLD_-INVALID if a character is not allowed, or additional error codes on general failure conditions.

Definition at line 398 of file tld.c.

References tld_check_4(), TLD_NODATA, and uint32_t.

5.25.2.5 int tld_check_8z (const char * *in*, size_t * *errpos*, const Tld_table ** *overrides*)

tld_check_8z - verify that characters are permitted

Parameters:

in Zero-terminated UTF8 string to process.

errpos Position of offending character is returned here.

overrides A Tld_table array of additional domain restriction structures that complement and supersede the built-in information.

Test each of the characters in for whether or not they are allowed by the information in or by the built-in TLD restriction data. When data for the same TLD is available both internally and in , the information in takes precedence. If several entries for a specific TLD are found, the first one is used. If is NULL, only the built-in information is used. The position of the first offending character is returned in . Note that the error position refers to the decoded character offset rather than the byte position in the string.

Return value: Returns the Tld_rc value TLD_SUCCESS if all characters are valid or when is null, TLD_-INVALID if a character is not allowed, or additional error codes on general failure conditions.

Definition at line 436 of file tld.c.

References stringprep_utf8_to_ucs4(), tld_check_4(), TLD_MALLOC_ERROR, TLD_NODATA, and uint32_t.

Referenced by tld_check_lz().

5.25.2.6 int tld_check_lz (const char * *in*, size_t * *errpos*, const Tld_table ** *overrides*)

tld_check_lz - verify that characters are permitted

Parameters:

in Zero-terminated string in the current locales encoding to process.

errpos Position of offending character is returned here.

overrides A Tld_table array of additional domain restriction structures that complement and supersede the built-in information.

Test each of the characters in for whether or not they are allowed by the information in or by the built-in TLD restriction data. When data for the same TLD is available both internally and in , the information in takes precedence. If several entries for a specific TLD are found, the first one is used. If is NULL, only the built-in information is used. The position of the first offending character is returned in . Note that the error position refers to the decoded character offset rather than the byte position in the string.

Return value: Returns the Tld_rc value TLD_SUCCESS if all characters are valid or when is null, TLD_-INVALID if a character is not allowed, or additional error codes on general failure conditions.

Definition at line 481 of file tld.c.

References stringprep_locale_to_utf8(), tld_check_8z(), TLD_ICONV_ERROR, and TLD_NODATA.

5.25.2.7 const Tld_table* tld_default_table (const char * *tld*, const Tld_table ** *overrides*)

tld_default_table - get table for a TLD name

Parameters:

tld TLD name (e.g. "com") as zero terminated ASCII byte string.

overrides Additional zero terminated array of Tld_table info-structures for TLDs, or NULL to only use library deault tables.

Get the TLD table for a named TLD, using the internal defaults, possibly overrided by the (optional) supplied tables.

Return value: Return structure corresponding to TLD , first looking through then thru built-in list, or NULL if no such structure found.

Definition at line 79 of file tld.c.

References _tld_tables, and tld_get_table().

Referenced by tld_check_4().

5.25.2.8 int tld_get_4 (const uint32_t * *in*, size_t *inlen*, char ** *out*)

tld_get_4 - extract top level domain part in input Unicode string

Parameters:

in Array of unicode code points to process. Does not need to be zero terminated.

inlen Number of unicode code points.

out Zero terminated ascii result string pointer.

Isolate the top-level domain of and return it as an ASCII string in .

Return value: Return TLD_SUCCESS on success, or the corresponding Tld_rc error code otherwise.

Definition at line 112 of file tld.c.

References DOTP, TLD_MALLOC_ERROR, TLD_NO_TLD, TLD_NODATA, TLD_SUCCESS, and uint32_t.

Referenced by tld_check_4(), tld_get_4z(), and tld_get_z().

5.25.2.9 int tld_get_4z (const uint32_t * *in*, char ** *out*)

tld_get_4z - extract top level domain part in input Unicode string

Parameters:

in Zero terminated array of unicode code points to process.

out Zero terminated ascii result string pointer.

Isolate the top-level domain of and return it as an ASCII string in .

Return value: Return TLD_SUCCESS on success, or the corresponding Tld_rc error code otherwise.

Definition at line 160 of file tld.c.

References tld_get_4(), TLD_NODATA, and uint32_t.

5.25.2.10 const Tld_table* tld_get_table (const char * *tld*, const Tld_table ** *tables*)

tld_get_table - get table for a TLD name in table

Parameters:

tld TLD name (e.g. "com") as zero terminated ASCII byte string.

tables Zero terminated array of Tld_table info-structures for TLDs.

Get the TLD table for a named TLD by searching through the given TLD table array.

Return value: Return structure corresponding to TLD by going thru , or return NULL if no such structure is found.

Definition at line 50 of file tld.c.

References Stringprep_profiles::tables.

Referenced by tld_default_table().

5.25.2.11 int tld_get_z (const char * *in*, char ** *out*)

tld_get_z - extract top level domain part in input string

Parameters:

in Zero terminated character array to process.

out Zero terminated ascii result string pointer.

Isolate the top-level domain of and return it as an ASCII string in . The input string may be UTF-8, ISO-8859-1 or any ASCII compatible character encoding.

Return value: Return TLD_SUCCESS on success, or the corresponding Tld_rc error code otherwise.

Definition at line 186 of file tld.c.

References tld_get_4(), TLD_MALLOC_ERROR, and uint32_t.

5.25.3 Variable Documentation

5.25.3.1 const Tld_table* _tld_tables[ ]

Definition at line 58 of file tlds.c.

Referenced by tld_default_table().

5.26 tld.h File Reference

```
#include <stdlib.h>
#include <idn-int.h>
```

Data Structures

- struct Tld_table_element
- struct Tld_table

Typedefs

- typedef Tld_table_element Tld_table_element
- typedef Tld_table Tld_table

Enumerations

- enum Tld_rc {
  TLD_SUCCESS = 0, TLD_INVALID = 1, TLD_NODATA = 2, TLD_MALLOC_ERROR = 3,
  TLD_ICONV_ERROR = 4, TLD_NO_TLD = 5, TLD_NOTLD = TLD_NO_TLD }

Functions

- const char * tld_strerror (Tld_rc rc)
- int tld_get_4 (const uint32_t *in, size_t inlen, char **out)
- int tld_get_4z (const uint32_t *in, char **out)
- int tld_get_z (const char *in, char **out)
- const Tld_table * tld_get_table (const char *tld, const Tld_table **tables)
- const Tld_table * tld_default_table (const char *tld, const Tld_table **overrides)
- int tld_check_4t (const uint32_t *in, size_t inlen, size_t *errpos, const Tld_table *tld)
- int tld_check_4tz (const uint32_t *in, size_t *errpos, const Tld_table *tld)
- int tld_check_4 (const uint32_t *in, size_t inlen, size_t *errpos, const Tld_table **overrides)
- int tld_check_4z (const uint32_t *in, size_t *errpos, const Tld_table **overrides)
- int tld_check_8z (const char *in, size_t *errpos, const Tld_table **overrides)
- int tld_check_lz (const char *in, size_t *errpos, const Tld_table **overrides)

5.26.1 Typedef Documentation

5.26.1.1 typedef struct Tld_table Tld_table

Definition at line 55 of file tld.h.

5.26.1.2 typedef struct Tld_table_element Tld_table_element

Definition at line 45 of file tld.h.

5.26.2 Enumeration Type Documentation

5.26.2.1 enum Tld_rc

Enumerator:

TLD_SUCCESS

TLD_INVALID

TLD_NODATA

TLD_MALLOC_ERROR

TLD_ICONV_ERROR

TLD_NO_TLD

TLD_NOTLD

Definition at line 58 of file tld.h.

5.26.3 Function Documentation

5.26.3.1 int tld_check_4 (const uint32_t * *in*, size_t *inlen*, size_t * *errpos*, const Tld_table ** *overrides*)

tld_check_4 - verify that characters are permitted

Parameters:

in Array of unicode code points to process. Does not need to be zero terminated.

inlen Number of unicode code points.

errpos Position of offending character is returned here.

overrides A Tld_table array of additional domain restriction structures that complement and supersede the built-in information.

Test each of the code points in for whether or not they are allowed by the information in or by the built-in TLD restriction data. When data for the same TLD is available both internally and in , the information in takes precedence. If several entries for a specific TLD are found, the first one is used. If is NULL, only the built-in information is used. The position of the first offending character is returned in .

Return value: Returns the Tld_rc value TLD_SUCCESS if all code points are valid or when is null, TLD_-INVALID if a character is not allowed, or additional error codes on general failure conditions.

Definition at line 348 of file tld.c.

References tld_check_4t(), tld_default_table(), tld_get_4(), TLD_NO_TLD, and TLD_SUCCESS.

Referenced by tld_check_4z(), and tld_check_8z().

5.26.3.2 int tld_check_4t (const uint32_t * *in*, size_t *inlen*, size_t * *errpos*, const Tld_table * *tld*)

tld_check_4t - verify that characters are permitted

Parameters:

in Array of unicode code points to process. Does not need to be zero terminated.

inlen Number of unicode code points.

errpos Position of offending character is returned here.

tld A Tld_table data structure representing the restrictions for which the input should be tested.

Test each of the code points in for whether or not they are allowed by the data structure in , return the position of the first character for which this is not the case in .

Return value: Returns the Tld_rc value TLD_SUCCESS if all code points are valid or when is null, TLD_-INVALID if a character is not allowed, or additional error codes on general failure conditions.

Definition at line 269 of file tld.c.

References TLD_SUCCESS, and uint32_t.

Referenced by tld_check_4(), and tld_check_4tz().

5.26.3.3 int tld_check_4tz (const uint32_t * *in*, size_t * *errpos*, const Tld_table * *tld*)

tld_check_4tz - verify that characters are permitted

Parameters:

in Zero terminated array of unicode code points to process.

errpos Position of offending character is returned here.

tld A Tld_table data structure representing the restrictions for which the input should be tested.

Test each of the code points in for whether or not they are allowed by the data structure in , return the position of the first character for which this is not the case in .

Return value: Returns the Tld_rc value TLD_SUCCESS if all code points are valid or when is null, TLD_-INVALID if a character is not allowed, or additional error codes on general failure conditions.

Definition at line 311 of file tld.c.

References tld_check_4t(), TLD_NODATA, and uint32_t.

5.26.3.4 int tld_check_4z (const uint32_t * *in*, size_t * *errpos*, const Tld_table ** *overrides*)

tld_check_4z - verify that characters are permitted

Parameters:

in Zero-terminated array of unicode code points to process.

errpos Position of offending character is returned here.

overrides A Tld_table array of additional domain restriction structures that complement and supersede the built-in information.

Test each of the code points in for whether or not they are allowed by the information in or by the built-in TLD restriction data. When data for the same TLD is available both internally and in , the information in takes precedence. If several entries for a specific TLD are found, the first one is used. If is NULL, only the built-in information is used. The position of the first offending character is returned in .

Return value: Returns the Tld_rc value TLD_SUCCESS if all code points are valid or when is null, TLD_-INVALID if a character is not allowed, or additional error codes on general failure conditions.

Definition at line 398 of file tld.c.

References tld_check_4(), TLD_NODATA, and uint32_t.

5.26.3.5 int tld_check_8z (const char * *in*, size_t * *errpos*, const Tld_table ** *overrides*)

tld_check_8z - verify that characters are permitted

Parameters:

in Zero-terminated UTF8 string to process.

errpos Position of offending character is returned here.

overrides A Tld_table array of additional domain restriction structures that complement and supersede the built-in information.

Test each of the characters in for whether or not they are allowed by the information in or by the built-in TLD restriction data. When data for the same TLD is available both internally and in , the information in takes precedence. If several entries for a specific TLD are found, the first one is used. If is NULL, only the built-in information is used. The position of the first offending character is returned in . Note that the error position refers to the decoded character offset rather than the byte position in the string.

Return value: Returns the Tld_rc value TLD_SUCCESS if all characters are valid or when is null, TLD_-INVALID if a character is not allowed, or additional error codes on general failure conditions.

Definition at line 436 of file tld.c.

References stringprep_utf8_to_ucs4(), tld_check_4(), TLD_MALLOC_ERROR, TLD_NODATA, and uint32_t.

Referenced by tld_check_lz().

5.26.3.6 int tld_check_lz (const char * *in*, size_t * *errpos*, const Tld_table ** *overrides*)

tld_check_lz - verify that characters are permitted

Parameters:

in Zero-terminated string in the current locales encoding to process.

errpos Position of offending character is returned here.

overrides A Tld_table array of additional domain restriction structures that complement and supersede the built-in information.

Test each of the characters in for whether or not they are allowed by the information in or by the built-in TLD restriction data. When data for the same TLD is available both internally and in , the information in takes precedence. If several entries for a specific TLD are found, the first one is used. If is NULL, only the built-in information is used. The position of the first offending character is returned in . Note that the error position refers to the decoded character offset rather than the byte position in the string.

Return value: Returns the Tld_rc value TLD_SUCCESS if all characters are valid or when is null, TLD_-INVALID if a character is not allowed, or additional error codes on general failure conditions.

Definition at line 481 of file tld.c.

References stringprep_locale_to_utf8(), tld_check_8z(), TLD_ICONV_ERROR, and TLD_NODATA.

5.26.3.7 const Tld_table* tld_default_table (const char * *tld*, const Tld_table ** *overrides*)

tld_default_table - get table for a TLD name

Parameters:

tld TLD name (e.g. "com") as zero terminated ASCII byte string.

overrides Additional zero terminated array of Tld_table info-structures for TLDs, or NULL to only use library deault tables.

Get the TLD table for a named TLD, using the internal defaults, possibly overrided by the (optional) supplied tables.

Return value: Return structure corresponding to TLD , first looking through then thru built-in list, or NULL if no such structure found.

Definition at line 79 of file tld.c.

References _tld_tables, and tld_get_table().

Referenced by tld_check_4().

5.26.3.8 int tld_get_4 (const uint32_t * *in*, size_t *inlen*, char ** *out*)

tld_get_4 - extract top level domain part in input Unicode string

Parameters:

in Array of unicode code points to process. Does not need to be zero terminated.

inlen Number of unicode code points.

out Zero terminated ascii result string pointer.

Isolate the top-level domain of and return it as an ASCII string in .

Return value: Return TLD_SUCCESS on success, or the corresponding Tld_rc error code otherwise.

Definition at line 112 of file tld.c.

References DOTP, TLD_MALLOC_ERROR, TLD_NO_TLD, TLD_NODATA, TLD_SUCCESS, and uint32_t.

Referenced by tld_check_4(), tld_get_4z(), and tld_get_z().

5.26.3.9 int tld_get_4z (const uint32_t * *in*, char ** *out*)

tld_get_4z - extract top level domain part in input Unicode string

Parameters:

in Zero terminated array of unicode code points to process.

out Zero terminated ascii result string pointer.

Isolate the top-level domain of and return it as an ASCII string in .

Return value: Return TLD_SUCCESS on success, or the corresponding Tld_rc error code otherwise.

Definition at line 160 of file tld.c.

References tld_get_4(), TLD_NODATA, and uint32_t.

5.26.3.10 const Tld_table* tld_get_table (const char * *tld*, const Tld_table ** *tables*)

tld_get_table - get table for a TLD name in table

Parameters:

tld TLD name (e.g. "com") as zero terminated ASCII byte string.

tables Zero terminated array of Tld_table info-structures for TLDs.

Get the TLD table for a named TLD by searching through the given TLD table array.

Return value: Return structure corresponding to TLD by going thru , or return NULL if no such structure is found.

Definition at line 50 of file tld.c.

References Stringprep_profiles::tables.

Referenced by tld_default_table().

5.26.3.11 int tld_get_z (const char * *in*, char ** *out*)

tld_get_z - extract top level domain part in input string

Parameters:

in Zero terminated character array to process.

out Zero terminated ascii result string pointer.

Isolate the top-level domain of and return it as an ASCII string in . The input string may be UTF-8, ISO-8859-1 or any ASCII compatible character encoding.

Return value: Return TLD_SUCCESS on success, or the corresponding Tld_rc error code otherwise.

Definition at line 186 of file tld.c.

References tld_get_4(), TLD_MALLOC_ERROR, and uint32_t.

5.26.3.12 const char* tld_strerror (Tld_rc *rc*)

tld_strerror - return string describing tld error code

Parameters:

rc tld return code

Convert a return code integer to a text string. This string can be used to output a diagnostic message to the user.

TLD_SUCCESS: Successful operation. This value is guaranteed to always be zero, the remaining ones are only guaranteed to hold non-zero values, for logical comparison purposes. TLD_INVALID: Invalid character found. TLD_NODATA: No input data was provided. TLD_MALLOC_ERROR: Error during memory allocation. TLD_ICONV_ERROR: Error during iconv string conversion. TLD_NO_TLD: No top-level domain found in domain string.

Return value: Returns a pointer to a statically allocated string containing a description of the error with the return code .

Definition at line 51 of file strerror-tld.c.

References _, TLD_ICONV_ERROR, TLD_INVALID, TLD_MALLOC_ERROR, TLD_NO_TLD, TLD_NODATA, and TLD_SUCCESS.

5.27 tlds.c File Reference

```
#include "tld.h"
```

Variables

- const Tld_table * _tld_tables [ ]

5.27.1 Variable Documentation

5.27.1.1 const Tld_table* _tld_tables[ ]

Initial value:

```
{
  &_tld_fr,
  &_tld_no,
  NULL
}
```

Definition at line 58 of file tlds.c.

Referenced by tld_default_table().

5.28 toutf8.c File Reference

```
#include "stringprep.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "iconvme.h"
```

Functions

- const char * stringprep_locale_charset (void)
- char * stringprep_convert (const char *str, const char *to_codeset, const char *from_codeset)
- char * stringprep_locale_to_utf8 (const char *str)
- char * stringprep_utf8_to_locale (const char *str)

5.28.1 Function Documentation

5.28.1.1 char* stringprep_convert (const char * *str*, const char * *to_codeset*, const char * *from_codeset*)

stringprep_convert - encode string using new character set

Parameters:

str input zero-terminated string.

to_codeset name of destination character set.

from_codeset name of origin character set, as used by .

Convert the string from one character set to another using the system's iconv() function.

Return value: Returns newly allocated zero-terminated string which is transcoded into to_codeset.

Definition at line 110 of file toutf8.c.

References iconv_string().

Referenced by stringprep_locale_to_utf8(), and stringprep_utf8_to_locale().

5.28.1.2 const char* stringprep_locale_charset (void)

stringprep_locale_charset - return charset used in current locale

Find out current locale charset. The function respect the CHARSET environment variable, but typically uses nl_langinfo(CODESET) when it is supported. It fall back on "ASCII" if CHARSET isn't set and nl_langinfo isn't supported or return anything.

Note that this function return the application's locale's preferred charset (or thread's locale's preffered charset, if your system support thread-specific locales). It does not return what the system may be using. Thus, if you receive data from external sources you cannot in general use this function to guess what charset it is encoded in. Use stringprep_convert from the external representation into the charset returned by this function, to have data in the locale encoding.

Return value: Return the character set used by the current locale. It will never return NULL, but use "ASCII" as a fallback.

Definition at line 79 of file toutf8.c.

Referenced by stringprep_locale_to_utf8(), and stringprep_utf8_to_locale().

5.28.1.3 char* stringprep_locale_to_utf8 (const char * *str*)

stringprep_locale_to_utf8 - convert locale encoded string to UTF-8

Parameters:

str input zero terminated string.

Convert string encoded in the locale's character set into UTF-8 by using stringprep_convert().

Return value: Returns newly allocated zero-terminated string which is transcoded into UTF-8.

Definition at line 137 of file toutf8.c.

References stringprep_convert(), and stringprep_locale_charset().

Referenced by idna_to_ascii_lz(), idna_to_unicode_lzlz(), and tld_check_lz().

5.28.1.4 char* stringprep_utf8_to_locale (const char * *str*)

stringprep_utf8_to_locale - encode UTF-8 string to locale encoding

Parameters:

str input zero terminated string.

Convert string encoded in UTF-8 into the locale's character set by using stringprep_convert().

Return value: Returns newly allocated zero-terminated string which is transcoded into the locale's character set.

Definition at line 153 of file toutf8.c.

References stringprep_convert(), and stringprep_locale_charset().

Referenced by idna_to_unicode_8zlz().

5.29 version.c File Reference

```
#include <ctype.h>
#include <string.h>
#include "stringprep.h"
```

Functions

- const char * stringprep_check_version (const char *req_version)

5.29.1 Function Documentation

5.29.1.1 const char* stringprep_check_version (const char * *req_version*)

stringprep_check_version - check for library version

Parameters:

req_version Required version number, or NULL.

Check that the the version of the library is at minimum the requested one and return the version string; return NULL if the condition is not satisfied. If a NULL is passed to this function, no check is done, but the version string is simply returned.

See STRINGPREP_VERSION for a suitable string.

Return value: Version string of run-time library, or NULL if the run-time library does not meet the required version number.

Definition at line 81 of file version.c.

Index