libidn

1.42

Generated by Doxygen 1.9.1

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, iSCSI, 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:
https://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) 2002-2024 Simon Josefsson
 *
 * This file is part of GNU Libidn.
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program 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 General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <https://www.gnu.org/licenses/>.
 *
 */
#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': ª
 * Before locale2utf8 (length 2): aa 0a
 * Before stringprep (length 3): c2 aa 0a
 * After stringprep (length 2): 61 0a
 * $
 *
 */
int
main (void)
{
  char buf[BUFSIZ];
  char *p;
  int rc;
  size_t i;
  setlocale (LC_ALL, "");
  printf ("Input string encoded as '%s': ", stringprep_locale_charset ());
  fflush (stdout);
  if (!fgets (buf, BUFSIZ, stdin))
    perror ("fgets");
  buf[strlen (buf) - 1] = '\0';
  printf ("Before locale2utf8 (length %ld): ", (long int) strlen (buf));
  for (i = 0; i < strlen (buf); i++)
    printf ("%02x ", (unsigned) 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 %ld): ", (long int) strlen (buf));
  for (i = 0; i < strlen (buf); i++)
    printf ("%02x ", (unsigned) 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 %ld): ", (long int) strlen (buf));
      for (i = 0; i < strlen (buf); i++)
        printf ("%02x ", (unsigned) buf[i] & 0xFF);
      printf ("\n");
    }
  return 0;
}
/* example3.c --- Example ToASCII() code showing how to use Libidn.
 * Copyright (C) 2002-2024 Simon Josefsson
 *
 * This file is part of GNU Libidn.
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program 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 General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <https://www.gnu.org/licenses/>.
 *
 */
#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.räksmörgåsª.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 (void)
{
  char buf[BUFSIZ];
  char *p;
  int rc;
  size_t i;
  setlocale (LC_ALL, "");
  printf ("Input domain encoded as `%s': ", stringprep_locale_charset ());
  fflush (stdout);
  if (!fgets (buf, BUFSIZ, stdin))
    perror ("fgets");
  buf[strlen (buf) - 1] = '\0';
  printf ("Read string (length %ld): ", (long int) strlen (buf));
  for (i = 0; i < strlen (buf); i++)
    printf ("%02x ", (unsigned) 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));
      return EXIT_FAILURE;
    }
  printf ("ACE label (length %ld): '%s'\n", (long int) strlen (p), p);
  for (i = 0; i < strlen (p); i++)
    printf ("%02x ", (unsigned) p[i] & 0xFF);
  printf ("\n");
  free (p);
  return 0;
}
/* example4.c --- Example ToUnicode() code showing how to use Libidn.
 * Copyright (C) 2002-2024 Simon Josefsson
 *
 * This file is part of GNU Libidn.
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
```

```
 * This program 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 General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <https://www.gnu.org/licenses/>.
 *
 */
#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 61
      6d 70 6c 65
 * 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 (void)
{
  char buf[BUFSIZ];
  char *p;
  int rc;
  size_t i;
  setlocale (LC_ALL, "");
  printf ("Input domain encoded as `%s': ", stringprep_locale_charset ());
  fflush (stdout);
  if (!fgets (buf, BUFSIZ, stdin))
    perror ("fgets");
  buf[strlen (buf) - 1] = '\0';
  printf ("Read string (length %ld): ", (long int) strlen (buf));
  for (i = 0; i < strlen (buf); i++)
    printf ("%02x ", (unsigned) 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));
      return EXIT_FAILURE;
    }
  printf ("ACE label (length %ld): '%s'\n", (long int) strlen (p), p);
  for (i = 0; i < strlen (p); i++)
    printf ("%02x ", (unsigned) p[i] & 0xFF);
  printf ("\n");
  free (p);
  return 0;
}
/* example5.c --- Example TLD checking.
 * Copyright (C) 2004-2024 Simon Josefsson
 *
 * This file is part of GNU Libidn.
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program 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 General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <https://www.gnu.org/licenses/>.
 *
 */
#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': fooß.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': gr€€n.no
 * Read string (length 12): 67 72 e2 82 ac e2 82 ac 6e 2e 6e 6f
 * ToASCII string (length 16): xn--grn-150aa.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 (void)
{
  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);
  if (!fgets (buf, BUFSIZ, stdin))
    perror ("fgets");
  buf[strlen (buf) - 1] = '\0';
  printf ("Read string (length %ld): ", (long int) strlen (buf));
  for (i = 0; i < strlen (buf); i++)
    printf ("%02x ", (unsigned) 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 %ld): %s\n", (long int) 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 %ld\n",
              (long int) 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

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

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 1826 of file gunidecomp.h.

4.1.2 Field Documentation

4.1.2.1 canon_offset

```
guint16 decomposition::canon_offset
```

Definition at line 1829 of file gunidecomp.h.

4.1.2.2 ch

```
gunichar decomposition::ch
```

Definition at line 1828 of file gunidecomp.h.

4.1.2.3 compat_offset

```
guint16 decomposition::compat_offset
```

Definition at line 1830 of file gunidecomp.h.

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

- gunidecomp.h

4.2 Pr29 Struct Reference

Data Fields

- const uint32_t * first
- const uint32_t * last

4.2.1 Detailed Description

Definition at line 1172 of file pr29.c.

4.2.2 Field Documentation

4.2.2.1 first

```
const uint32_t* Pr29::first
```

Definition at line 1174 of file pr29.c.

4.2.2.2 last

```
const uint32_t* Pr29::last
```

Definition at line 1175 of file pr29.c.

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

- pr29.c

4.3 Stringprep_profiles Struct Reference

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

Data Fields

- const char * name
- const Stringprep_profile * tables

4.3.1 Detailed Description

Stringprep_profiles:

Parameters

name	name of stringprep profile.
tables	zero-terminated array of Stringprep_profile elements.

Element structure

Definition at line 169 of file stringprep.h.

4.3.2 Field Documentation

4.3.2.1 name

```
const char* Stringprep_profiles::name
```

Definition at line 171 of file stringprep.h.

4.3.2.2 tables

```
const Stringprep_profile* Stringprep_profiles::tables
```

Definition at line 172 of file stringprep.h.

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

- stringprep.h

4.4 Stringprep_table Struct Reference

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

Data Fields

- Stringprep_profile_steps operation
- Stringprep_profile_flags flags
- const Stringprep_table_element * table
- size_t table_size

4.4.1 Detailed Description

Stringprep_table:

Parameters

operation	a Stringprep_profile_steps value
flags	a Stringprep_profile_flags value
table	zero-terminated array of Stringprep_table_element elements.
table_size	size of @table, to speed up searching.

Stringprep profile table.

Definition at line 148 of file stringprep.h.

4.4.2 Field Documentation

4.4.2.1 flags

```
Stringprep_profile_flags Stringprep_table::flags
```

Definition at line 151 of file stringprep.h.

4.4.2.2 operation

```
Stringprep_profile_steps Stringprep_table::operation
```

Definition at line 150 of file stringprep.h.

4.4.2.3 table

```
const Stringprep_table_element* Stringprep_table::table
```

Definition at line 152 of file stringprep.h.

4.4.2.4 table_size

```
size_t Stringprep_table::table_size
```

Definition at line 153 of file stringprep.h.

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

- stringprep.h

4.5 Stringprep_table_element Struct Reference

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

Data Fields

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

4.5.1 Detailed Description

Stringprep_table_element:

Parameters

start	starting codepoint.
end	ending codepoint, 0 if only one character.
map	codepoints to map @start into, NULL if end is not 0.

Stringprep profile table element.

Definition at line 131 of file stringprep.h.

4.5.2 Field Documentation

4.5.2.1 end

```
uint32_t Stringprep_table_element::end
```

Definition at line 134 of file stringprep.h.

4.5.2.2 map

```
uint32_t Stringprep_table_element::map[STRINGPREP_MAX_MAP_CHARS]
```

Definition at line 135 of file stringprep.h.

4.5.2.3 start

```
uint32_t Stringprep_table_element::start
```

Definition at line 133 of file stringprep.h.

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

- stringprep.h

4.6 Tld_table Struct Reference

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

Data Fields

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

4.6.1 Detailed Description

Tld_table:

Parameters

name	TLD name, e.g., "no".
version	Version string from TLD file.
nvalid	Number of entries in data.
valid	Sorted array (of size @nvalid) of valid code points.

List valid code points in a TLD.

Definition at line 94 of file tld.h.

4.6.2 Field Documentation

4.6.2.1 name

```
const char* Tld_table::name
```

Definition at line 96 of file tld.h.

4.6.2.2 nvalid

```
size_t Tld_table::nvalid
```

Definition at line 98 of file tld.h.

4.6.2.3 valid

```
const Tld_table_element* Tld_table::valid
```

Definition at line 99 of file tld.h.

4.6.2.4 version

```
const char* Tld_table::version
```

Definition at line 97 of file tld.h.

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

- tld.h

4.7 Tld_table_element Struct Reference

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

Data Fields

- uint32_t start
- uint32_t end

4.7.1 Detailed Description

Tld_table_element:

Parameters

start	Start of range.
end	End of range, end == start if single.

Interval of valid code points in the TLD.

Definition at line 78 of file tld.h.

4.7.2 Field Documentation

4.7.2.1 end

```
uint32_t Tld_table_element::end
```

Definition at line 81 of file tld.h.

4.7.2.2 start

```
uint32_t Tld_table_element::start
```

Definition at line 80 of file tld.h.

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

- tld.h

Chapter 5

File Documentation

5.1 gunibreak.h File Reference

Macros

- #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 Macro Definition Documentation

5.1.1.1 G_UNICODE_DATA_VERSION

```
#define G_UNICODE_DATA_VERSION "3.2"
```

Definition at line 8 of file gunibreak.h.

5.1.1.2 G_UNICODE_LAST_CHAR

```
#define G_UNICODE_LAST_CHAR 0x10FFFF
```

Definition at line 10 of file gunibreak.h.

5.1.1.3 G_UNICODE_LAST_CHAR_PART1

```
#define G_UNICODE_LAST_CHAR_PART1 0x2FAFF
```

Definition at line 15 of file gunibreak.h.

5.1.1.4 G_UNICODE_MAX_TABLE_INDEX

```
#define G_UNICODE_MAX_TABLE_INDEX 10000
```

Definition at line 12 of file gunibreak.h.

5.2 gunicomp.h File Reference

Macros

- #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 Macro Definition Documentation

5.2.1.1 COMPOSE_FIRST_SINGLE_START

```
#define COMPOSE_FIRST_SINGLE_START 147
```

Definition at line 6 of file gunicomp.h.

5.2.1.2 COMPOSE_FIRST_START

```
#define COMPOSE_FIRST_START 1
```

Definition at line 5 of file gunicomp.h.

5.2.1.3 COMPOSE_SECOND_SINGLE_START

```
#define COMPOSE_SECOND_SINGLE_START 388
```

Definition at line 8 of file gunicomp.h.

5.2.1.4 COMPOSE_SECOND_START

```
#define COMPOSE_SECOND_START 357
```

Definition at line 7 of file gunicomp.h.

5.2.1.5 COMPOSE_TABLE_LAST

```
#define COMPOSE_TABLE_LAST 48
```

Definition at line 10 of file gunicomp.h.

5.3 gunidecomp.h File Reference

Data Structures

- struct decomposition

Macros

- #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 Macro Definition Documentation

5.3.1.1 G_UNICODE_LAST_CHAR

```
#define G_UNICODE_LAST_CHAR 0x10ffff
```

Definition at line 8 of file gunidecomp.h.

5.3.1.2 G_UNICODE_LAST_CHAR_PART1

```
#define G_UNICODE_LAST_CHAR_PART1 0x2FAFF
```

Definition at line 12 of file gunidecomp.h.

5.3.1.3 G_UNICODE_LAST_PAGE_PART1

```
#define G_UNICODE_LAST_PAGE_PART1 762
```

Definition at line 14 of file gunidecomp.h.

5.3.1.4 G_UNICODE_MAX_TABLE_INDEX

```
#define G_UNICODE_MAX_TABLE_INDEX (0x110000 / 256)
```

Definition at line 10 of file gunidecomp.h.

5.3.1.5 G_UNICODE_NOT_PRESENT_OFFSET

```
#define G_UNICODE_NOT_PRESENT_OFFSET 65535
```

Definition at line 16 of file gunidecomp.h.

5.4 idn-free.c File Reference

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

Functions

- void idn_free (void *ptr)

5.4.1 Function Documentation

5.4.1.1 idn_free()

```
void idn_free (
            void * ptr )
```

idn_free:

Parameters

ptr	memory region to deallocate, or NULL.

Deallocates memory region by calling free(). If @ptr is NULL no operation is performed.

Normally applications de-allocate strings allocated by libidn by calling free() directly. Under Windows, different parts of the same application may use different heap memory, and then it is important to deallocate memory allocated within the same module that allocated it. This function makes that possible.

Definition at line 52 of file idn-free.c.

5.5 idn-free.h File Reference

Macros

- #define IDNAPI

Functions

- void IDNAPI idn_free (void ∗ptr)

5.5.1 Macro Definition Documentation

5.5.1.1 IDNAPI

```
#define IDNAPI
```

SECTION:idn-free

Parameters

title	idn-free.h
short_description	Memory deallocation functions

Memory deallocation functions.

Definition at line 49 of file idn-free.h.

5.5.2 Function Documentation

5.5.2.1 idn_free()

```
void IDNAPI idn_free (
            void * ptr )
```

idn_free:

Parameters

ptr	memory region to deallocate, or NULL.

Deallocates memory region by calling free(). If @ptr is NULL no operation is performed.

Normally applications de-allocate strings allocated by libidn by calling free() directly. Under Windows, different parts of the same application may use different heap memory, and then it is important to deallocate memory allocated within the same module that allocated it. This function makes that possible.

Definition at line 52 of file idn-free.c.

5.6 idn-int.h File Reference

```
#include <stdint.h>
```

5.7 idna.c File Reference

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

Macros

- #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)

5.7.1 Macro Definition Documentation

5.7.1.1 DOTP

```
#define DOTP(
            c )
```

Value:

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

Definition at line 44 of file idna.c.

5.7.2 Function Documentation

5.7.2.1 idna_to_ascii_4i()

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

idna_to_ascii_4i:

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 domain 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 To↩ ASCII 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 81 of file idna.c.

5.7.2.2 idna_to_ascii_4z()

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

idna_to_ascii_4z:

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 477 of file idna.c.

5.7.2.3 idna_to_ascii_8z()

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

idna_to_ascii_8z:

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 576 of file idna.c.

5.7.2.4 idna_to_ascii_lz()

```
int idna_to_ascii_lz (
            const char * input,
```

```
                char ** output,
                int flags )
```

idna_to_ascii_lz:

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 609 of file idna.c.

5.7.2.5 idna_to_unicode_44i()

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

idna_to_unicode_44i:

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 domain 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 To↩ Unicode 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 437 of file idna.c.

5.7.2.6 idna_to_unicode_4z4z()

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

idna_to_unicode_4z4z:

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 640 of file idna.c.

5.7.2.7 idna_to_unicode_8z4z()

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

idna_to_unicode_8z4z:

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 719 of file idna.c.

5.7.2.8 idna_to_unicode_8z8z()

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

idna_to_unicode_8z8z:

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 750 of file idna.c.

5.7.2.9 idna_to_unicode_8zlz()

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

idna_to_unicode_8zlz:

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 784 of file idna.c.

5.7.2.10 idna_to_unicode_lzlz()

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

idna_to_unicode_lzlz:

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 819 of file idna.c.

5.8 idna.h File Reference

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

Macros

- #define IDNAPI
- #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

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

5.8.1 Macro Definition Documentation

5.8.1.1 IDNA_ACE_PREFIX

```
#define IDNA_ACE_PREFIX "xn--"
```

Definition at line 99 of file idna.h.

5.8.1.2 IDNAPI

```
#define IDNAPI
```

SECTION:idna

Parameters

title	idna.h
short_description	IDNA-related functions

IDNA-related functions. IDNAPI:

Symbol holding shared library API visibility decorator.

This is used internally by the library header file and should never be used or modified by the application.

https://www.gnu.org/software/gnulib/manual/html_node/Exported-Symbols-of-Shared-Libraries.html

Definition at line 59 of file idna.h.

5.8.2 Enumeration Type Documentation

5.8.2.1 Idna_flags

```
enum Idna_flags
```

Enumerator

IDNA_ALLOW_UNASSIGNED	
IDNA_USE_STD3_ASCII_RULES	

Definition at line 92 of file idna.h.

5.8.2.2 Idna_rc

```
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 72 of file idna.h.

5.8.3 Function Documentation

5.8.3.1 idna_strerror()

```
IDNAPI const char* idna_strerror (
            Idna_rc rc )
```

idna_strerror:

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: Character encoding conversion error. 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 @rc.

Definition at line 73 of file strerror-idna.c.

5.8.3.2 idna_to_ascii_4i()

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

idna_to_ascii_4i:

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 domain 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 81 of file idna.c.

5.8.3.3 idna_to_ascii_4z()

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

idna_to_ascii_4z:

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 477 of file idna.c.

5.8.3.4 idna_to_ascii_8z()

```
IDNAPI int idna_to_ascii_8z (
            const char * input,
            char ** output,
            int flags )
```

idna_to_ascii_8z:

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 576 of file idna.c.

5.8.3.5 idna_to_ascii_lz()

```
IDNAPI int idna_to_ascii_lz (
            const char * input,
            char ** output,
            int flags )
```

idna_to_ascii_lz:

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 609 of file idna.c.

5.8.3.6 idna_to_unicode_44i()

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

idna_to_unicode_44i:

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 domain 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 To↩ Unicode 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 437 of file idna.c.

5.8.3.7 idna_to_unicode_4z4z()

```
IDNAPI int idna_to_unicode_4z4z (
            const uint32_t * input,
            uint32_t ** output,
            int flags )
```

idna_to_unicode_4z4z:

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 640 of file idna.c.

5.8.3.8 idna_to_unicode_8z4z()

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

idna_to_unicode_8z4z:

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 719 of file idna.c.

5.8.3.9 idna_to_unicode_8z8z()

```
IDNAPI int idna_to_unicode_8z8z (
            const char * input,
            char ** output,
            int flags )
```

idna_to_unicode_8z8z:

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 750 of file idna.c.

5.8.3.10 idna_to_unicode_8zlz()

```
IDNAPI int idna_to_unicode_8zlz (
            const char * input,
            char ** output,
            int flags )
```

idna_to_unicode_8zlz:

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 784 of file idna.c.

5.8.3.11 idna_to_unicode_lzlz()

```
IDNAPI int idna_to_unicode_lzlz (
            const char * input,
            char ** output,
            int flags )
```

idna_to_unicode_lzlz:

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 819 of file idna.c.

5.9 nfkc.c File Reference

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

Macros

- #define gboolean int
- #define gchar char
- #define guchar unsigned char
- #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 g_return_val_if_fail(expr, val)
- #define FALSE (0)
- #define TRUE (!FALSE)
- #define G_N_ELEMENTS(arr) (sizeof (arr) / sizeof ((arr)[0]))
- #define G_UNLIKELY(expr) (expr)
- #define g_utf8_next_char(p) ((p) + g_utf8_skip[*(const guchar *)(p)])
- #define UTF8_COMPUTE(Char, Mask, Len)
- #define UTF8_LENGTH(Char)
- #define UTF8_GET(Result, Chars, Count, Mask, Len)
- #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 (const uint32_t *str, ssize_t len)

5.9.1 Macro Definition Documentation

5.9.1.1 CC_PART1

```
#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 530 of file nfkc.c.

5.9.1.2 CC_PART2

```
#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 535 of file nfkc.c.

5.9.1.3 CI

```
#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 703 of file nfkc.c.

5.9.1.4 COMBINING_CLASS

```
#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 540 of file nfkc.c.

5.9.1.5 COMPOSE_INDEX

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

Definition at line 708 of file nfkc.c.

5.9.1.6 FALSE

```
#define FALSE (0)
```

Definition at line 80 of file nfkc.c.

5.9.1.7 g_free

```
#define g_free free
```

Definition at line 52 of file nfkc.c.

5.9.1.8 g_malloc

```
#define g_malloc malloc
```

Definition at line 51 of file nfkc.c.

5.9.1.9 G_N_ELEMENTS

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

Definition at line 87 of file nfkc.c.

5.9.1.10 g_return_val_if_fail

```
#define g_return_val_if_fail(
              expr,
              val )
```

Value:

```
    {           \
    if (!(expr))                                    \
      return (val);                                 \
  }
```

Definition at line 53 of file nfkc.c.

5.9.1.11 G_UNLIKELY

```
#define G_UNLIKELY(
              expr ) (expr)
```

Definition at line 89 of file nfkc.c.

5.9.1.12 g_utf8_next_char

```
#define g_utf8_next_char(
              p ) ((p) + g_utf8_skip[*(const guchar *)(p)])
```

Definition at line 127 of file nfkc.c.

5.9.1.13 gboolean

```
#define gboolean int
```

Definition at line 40 of file nfkc.c.

5.9.1.14 gchar

```
#define gchar char
```

Definition at line 41 of file nfkc.c.

5.9.1.15 gint

```
#define gint int
```

Definition at line 43 of file nfkc.c.

5.9.1.16 gint16

```
#define gint16 int16_t
```

Definition at line 46 of file nfkc.c.

5.9.1.17 gsize

```
#define gsize size_t
```

Definition at line 49 of file nfkc.c.

5.9.1.18 gssize

```
#define gssize ssize_t
```

Definition at line 50 of file nfkc.c.

5.9.1.19 guchar

```
#define guchar unsigned char
```

Definition at line 42 of file nfkc.c.

5.9.1.20 guint

```
#define guint unsigned int
```

Definition at line 44 of file nfkc.c.

5.9.1.21 guint16

```
#define guint16 uint16_t
```

Definition at line 47 of file nfkc.c.

5.9.1.22 gunichar

```
#define gunichar uint32_t
```

Definition at line 48 of file nfkc.c.

5.9.1.23 gushort

```
#define gushort unsigned short
```

Definition at line 45 of file nfkc.c.

5.9.1.24 LBase

```
#define LBase 0x1100
```

Definition at line 549 of file nfkc.c.

5.9.1.25 LCount

```
#define LCount 19
```

Definition at line 552 of file nfkc.c.

5.9.1.26 NCount

```
#define NCount (VCount * TCount)
```

Definition at line 555 of file nfkc.c.

5.9.1.27 SBase

```
#define SBase 0xAC00
```

Definition at line 548 of file nfkc.c.

5.9.1.28 SCount

```
#define SCount (LCount * NCount)
```

Definition at line 556 of file nfkc.c.

5.9.1.29 TBase

```
#define TBase 0x11A7
```

Definition at line 551 of file nfkc.c.

5.9.1.30 TCount

```
#define TCount 28
```

Definition at line 554 of file nfkc.c.

5.9.1.31 TRUE

```
#define TRUE (!FALSE)
```

Definition at line 84 of file nfkc.c.

5.9.1.32 UTF8_COMPUTE

```
#define UTF8_COMPUTE(
              Char,
              Mask,
              Len )
```

Definition at line 152 of file nfkc.c.

5.9.1.33 UTF8_GET

```
#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 193 of file nfkc.c.

5.9.1.34 UTF8_LENGTH

```
#define UTF8_LENGTH(
              Char )
```

Value:

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

Definition at line 186 of file nfkc.c.

5.9.1.35 VBase

```
#define VBase 0x1161
```

Definition at line 550 of file nfkc.c.

5.9.1.36 VCount

```
#define VCount 21
```

Definition at line 553 of file nfkc.c.

5.9.2 Enumeration Type Documentation

5.9.2.1 GNormalizeMode

```
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 114 of file nfkc.c.

5.9.3 Function Documentation

5.9.3.1 stringprep_ucs4_nfkc_normalize()

```
uint32_t* stringprep_ucs4_nfkc_normalize (
            const uint32_t * str,
            ssize_t len )
```

stringprep_ucs4_nfkc_normalize:

Parameters

str	a Unicode string.
len	length of @str array, or -1 if @str is nul-terminated.

Converts a UCS4 string into canonical form, see stringprep_utf8_nfkc_normalize() for more information.

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

Definition at line 1096 of file nfkc.c.

5.9.3.2 stringprep_ucs4_to_utf8()

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

stringprep_ucs4_to_utf8:

Parameters

str	a UCS-4 encoded string
len	the maximum length of @str to use. If @len $<$ 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 deallocated by the caller. If an error occurs, NULL will be returned.

Definition at line 1039 of file nfkc.c.

5.9.3.3 stringprep_unichar_to_utf8()

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

stringprep_unichar_to_utf8:

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 @outbuf.

Converts a single character to UTF-8.

Return value: number of bytes written.

Definition at line 982 of file nfkc.c.

5.9.3.4 stringprep_utf8_nfkc_normalize()

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

stringprep_utf8_nfkc_normalize:

Parameters

str	a UTF-8 encoded string.
len	length of @str, in bytes, or -1 if @str 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 @str.

Definition at line 1068 of file nfkc.c.

5.9.3.5 stringprep_utf8_to_ucs4()

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

stringprep_utf8_to_ucs4:

Parameters

str	a UTF-8 encoded string
len	the maximum length of @str to use. If @len $<$ 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. The function now performs error checking to verify that the input is valid UTF-8 (before it was documented to not do error checking).

Return value: a pointer to a newly allocated UCS-4 string. This value must be deallocated by the caller.

Definition at line 1006 of file nfkc.c.

5.9.3.6 stringprep_utf8_to_unichar()

```
uint32_t stringprep_utf8_to_unichar (
            const char * p )
```

stringprep_utf8_to_unichar:

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 965 of file nfkc.c.

5.10 pr29.c File Reference

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

Data Structures

- struct Pr29

Functions

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

5.10.1 Function Documentation

5.10.1.1 pr29_4()

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

pr29_4:

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 1247 of file pr29.c.

5.10.1.2 pr29_4z()

```
int pr29_4z (
            const uint32_t * in )
```

pr29_4z:

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 1288 of file pr29.c.

5.10.1.3 pr29_8z()

```
int pr29_8z (
            const char * in )
```

pr29_8z:

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 1313 of file pr29.c.

5.11 pr29.h File Reference

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

Macros

- #define IDNAPI

Enumerations

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

Functions

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

5.11.1 Macro Definition Documentation

5.11.1.1 IDNAPI

```
#define IDNAPI
```

SECTION:pr29

Parameters

title	pr29.h
short_description	PR29-related functions

PR29-related functions.

Definition at line 49 of file pr29.h.

5.11.2 Enumeration Type Documentation

5.11.2.1 Pr29_rc

```
enum Pr29_rc
```

Enumerator

PR29_SUCCESS	
PR29_PROBLEM	
PR29_STRINGPREP_ERROR	

Definition at line 65 of file pr29.h.

5.11.3 Function Documentation

5.11.3.1 pr29_4()

```
IDNAPI int pr29_4 (
            const uint32_t * in,
            size_t len )
```

pr29_4:

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 1247 of file pr29.c.

5.11.3.2 pr29_4z()

```
IDNAPI int pr29_4z (
            const uint32_t * in )
```

pr29_4z:

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 1288 of file pr29.c.

5.11.3.3 pr29_8z()

```
IDNAPI int pr29_8z (
            const char * in )
```

pr29_8z:

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 1313 of file pr29.c.

5.11.3.4 pr29_strerror()

```
IDNAPI const char* pr29_strerror (
            Pr29_rc rc )
```

pr29_strerror:

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_8z()).

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

Definition at line 57 of file strerror-pr29.c.

5.12 profiles.c File Reference

```
#include <config.h>
#include "stringprep.h"
#include "rfc3454.h"
```

Macros

- #define countof(a) (sizeof(a)/sizeof(∗(a)))
- #define TABLE(x) stringprep_rfc3454_##x, N_STRINGPREP_rfc3454_##x

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 [ ]

5.12.1 Macro Definition Documentation

5.12.1.1 countof

```
#define countof(
            a ) (sizeof(a)/sizeof(*(a)))
```

Definition at line 48 of file profiles.c.

5.12.1.2 TABLE

```
#define TABLE(
              x ) stringprep_rfc3454_##x, N_STRINGPREP_rfc3454_##x
```

Definition at line 51 of file profiles.c.

5.12.2 Variable Documentation

5.12.2.1 stringprep_iscsi

```
const Stringprep_profile stringprep_iscsi[]
```

Initial value:

```
= {
  {STRINGPREP_MAP_TABLE, 0, TABLE (B_1)},
  {STRINGPREP_MAP_TABLE, 0, TABLE (B_2)},
  {STRINGPREP_NFKC, 0, 0, 0},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_1_1)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_1_2)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_2_1)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_2_2)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_3)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_4)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_5)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_6)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_7)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_8)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_9)},
  {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_iscsi_prohibit,
   countof (stringprep_iscsi_prohibit) - 1},
  {STRINGPREP_BIDI, 0, 0, 0},
  {STRINGPREP_BIDI_PROHIBIT_TABLE, 0, TABLE (C_8)},
  {STRINGPREP_BIDI_RAL_TABLE, ~STRINGPREP_NO_BIDI, TABLE (D_1)},
  {STRINGPREP_BIDI_L_TABLE, ~STRINGPREP_NO_BIDI, TABLE (D_2)},
  {STRINGPREP_UNASSIGNED_TABLE, ~STRINGPREP_NO_UNASSIGNED, TABLE (A_1)},
  {0}
}
```

Definition at line 197 of file profiles.c.

5.12.2.2 stringprep_iscsi_prohibit

```
const Stringprep_table_element stringprep_iscsi_prohibit[]
```

Initial value:

```
= {
  {0x0000, 0x002C},
  {0x002F, 0x002F},
  {0x003B, 0x0040},
  {0x005B, 0x0060},
  {0x007B, 0x007F},
  {0x3002, 0x3002},
  {0}
}
```

Definition at line 187 of file profiles.c.

5.12.2.3 stringprep_kerberos5

```
const Stringprep_profile stringprep_kerberos5[]
```

Initial value:

```
= {

  {STRINGPREP_MAP_TABLE, 0, TABLE (B_1)},
  {STRINGPREP_MAP_TABLE, 0, TABLE (B_3)},
  {STRINGPREP_NFKC, 0, 0, 0},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_1_2)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_2_2)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_3)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_4)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_5)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_6)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_7)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_8)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_9)},
  {STRINGPREP_BIDI, 0, 0, 0},
  {STRINGPREP_BIDI_PROHIBIT_TABLE, ~STRINGPREP_NO_BIDI, TABLE (C_8)},
  {STRINGPREP_BIDI_RAL_TABLE, 0, TABLE (D_1)},
  {STRINGPREP_BIDI_L_TABLE, 0, TABLE (D_2)},
  {STRINGPREP_UNASSIGNED_TABLE, ~STRINGPREP_NO_UNASSIGNED, TABLE (A_1)},
  {0}
}
```

Definition at line 74 of file profiles.c.

5.12.2.4 stringprep_nameprep

```
const Stringprep_profile stringprep_nameprep[]
```

Initial value:

```
= {
  {STRINGPREP_MAP_TABLE, 0, TABLE (B_1)},
  {STRINGPREP_MAP_TABLE, 0, TABLE (B_2)},
  {STRINGPREP_NFKC, 0, 0, 0},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_1_2)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_2_2)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_3)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_4)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_5)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_6)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_7)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_8)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_9)},
  {STRINGPREP_BIDI, 0, 0, 0},
  {STRINGPREP_BIDI_PROHIBIT_TABLE, ~STRINGPREP_NO_BIDI, TABLE (C_8)},
  {STRINGPREP_BIDI_RAL_TABLE, 0, TABLE (D_1)},
  {STRINGPREP_BIDI_L_TABLE, 0, TABLE (D_2)},
  {STRINGPREP_UNASSIGNED_TABLE, ~STRINGPREP_NO_UNASSIGNED, TABLE (A_1)},
  {0}
}
```

Definition at line 53 of file profiles.c.

5.12.2.5 stringprep_plain

```
const Stringprep_profile stringprep_plain[]
```

Initial value:

```
= {
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_2_1)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_2_2)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_3)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_4)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_5)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_6)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_8)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_9)},
  {STRINGPREP_BIDI, 0, 0, 0},
  {STRINGPREP_BIDI_PROHIBIT_TABLE, 0, TABLE (C_8)},
  {STRINGPREP_BIDI_RAL_TABLE, ~STRINGPREP_NO_BIDI, TABLE (D_1)},
  {STRINGPREP_BIDI_L_TABLE, ~STRINGPREP_NO_BIDI, TABLE (D_2)},
  {0}
}
```

Definition at line 155 of file profiles.c.

5.12.2.6 stringprep_profiles

```
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 34 of file profiles.c.

5.12.2.7 stringprep_saslprep

```
const Stringprep_profile stringprep_saslprep[]
```

Initial value:

```
= {
  {STRINGPREP_MAP_TABLE, 0, stringprep_saslprep_space_map,
   countof (stringprep_saslprep_space_map) - 1},
  {STRINGPREP_MAP_TABLE, 0, TABLE (B_1)},
  {STRINGPREP_NFKC, 0, 0, 0},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_1_2)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_2_1)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_2_2)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_3)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_4)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_5)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_6)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_7)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_8)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_9)},
  {STRINGPREP_BIDI, 0, 0, 0},
  {STRINGPREP_BIDI_PROHIBIT_TABLE, 0, TABLE (C_8)},
  {STRINGPREP_BIDI_RAL_TABLE, ~STRINGPREP_NO_BIDI, TABLE (D_1)},
  {STRINGPREP_BIDI_L_TABLE, ~STRINGPREP_NO_BIDI, TABLE (D_2)},
  {STRINGPREP_UNASSIGNED_TABLE, ~STRINGPREP_NO_UNASSIGNED, TABLE (A_1)},
  {0}
}
```

Definition at line 243 of file profiles.c.

5.12.2.8 stringprep_saslprep_space_map

```
const Stringprep_table_element stringprep_saslprep_space_map[]
```

Initial value:

```
= {
  {0x00A0, 0x00A0, {0x0020}},
  {0x1680, 0x1680, {0x0020}},
  {0x2000, 0x200B, {0x0020}},
  {0x202F, 0x202F, {0x0020}},
  {0x205F, 0x205F, {0x0020}},
  {0x3000, 0x3000, {0x0020}},
  {0}
}
```

Definition at line 222 of file profiles.c.

5.12.2.9 stringprep_trace

```
const Stringprep_profile stringprep_trace[]
```

Initial value:

```
= {
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_2_1)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_2_2)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_3)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_4)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_5)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_6)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_8)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_9)},
  {STRINGPREP_BIDI, 0, 0, 0},
  {STRINGPREP_BIDI_PROHIBIT_TABLE, 0, TABLE (C_8)},
  {STRINGPREP_BIDI_RAL_TABLE, ~STRINGPREP_NO_BIDI, TABLE (D_1)},
  {STRINGPREP_BIDI_L_TABLE, ~STRINGPREP_NO_BIDI, TABLE (D_2)},
  {0}
}
```

Definition at line 171 of file profiles.c.

5.12.2.10 stringprep_xmpp_nodeprep

```
const Stringprep_profile stringprep_xmpp_nodeprep[]
```

Initial value:

```
= {
  {STRINGPREP_MAP_TABLE, 0, TABLE (B_1)},
  {STRINGPREP_MAP_TABLE, 0, TABLE (B_2)},
  {STRINGPREP_NFKC, 0, 0, 0},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_1_1)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_1_2)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_2_1)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_2_2)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_3)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_4)},
```

```
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_5)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_6)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_7)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_8)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_9)},
  {STRINGPREP_PROHIBIT_TABLE, 0, stringprep_xmpp_nodeprep_prohibit,
   countof (stringprep_xmpp_nodeprep_prohibit) - 1},
  {STRINGPREP_BIDI, 0, 0, 0},
  {STRINGPREP_BIDI_PROHIBIT_TABLE, 0, TABLE (C_8)},
  {STRINGPREP_BIDI_RAL_TABLE, 0, TABLE (D_1)},
  {STRINGPREP_BIDI_L_TABLE, 0, TABLE (D_2)},
  {STRINGPREP_UNASSIGNED_TABLE, ~STRINGPREP_NO_UNASSIGNED, TABLE (A_1)},
  {0}
}
```

Definition at line 109 of file profiles.c.

5.12.2.11 stringprep_xmpp_nodeprep_prohibit

```
const Stringprep_table_element stringprep_xmpp_nodeprep_prohibit[]
```

Initial value:

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

Definition at line 97 of file profiles.c.

5.12.2.12 stringprep_xmpp_resourceprep

```
const Stringprep_profile stringprep_xmpp_resourceprep[]
```

Initial value:

```
= {
  {STRINGPREP_MAP_TABLE, 0, TABLE (B_1)},
  {STRINGPREP_NFKC, 0, 0, 0},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_1_2)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_2_1)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_2_2)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_3)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_4)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_5)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_6)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_7)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_8)},
  {STRINGPREP_PROHIBIT_TABLE, 0, TABLE (C_9)},
  {STRINGPREP_BIDI, 0, 0, 0},
  {STRINGPREP_BIDI_PROHIBIT_TABLE, 0, TABLE (C_8)},
  {STRINGPREP_BIDI_RAL_TABLE, ~STRINGPREP_NO_BIDI, TABLE (D_1)},
  {STRINGPREP_BIDI_L_TABLE, ~STRINGPREP_NO_BIDI, TABLE (D_2)},
  {STRINGPREP_UNASSIGNED_TABLE, ~STRINGPREP_NO_UNASSIGNED, TABLE (A_1)},
  {0}
}
```

Definition at line 134 of file profiles.c.

5.13 punycode.c File Reference

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

Macros

- #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[ ])

5.13.1 Macro Definition Documentation

5.13.1.1 basic

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

Definition at line 82 of file punycode.c.

5.13.1.2 delim

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

Definition at line 85 of file punycode.c.

5.13.1.3 flagged

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

Definition at line 116 of file punycode.c.

5.13.2 Enumeration Type Documentation

5.13.2.1 anonymous enum

```
anonymous enum
```

Enumerator

base	
tmin	
tmax	
skew	
damp	
initial_bias	
initial_n	
delimiter	

Definition at line 76 of file punycode.c.

5.13.3 Function Documentation

5.13.3.1 punycode_decode()

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

punycode_decode:

Parameters

input_length	The number of ASCII code points in the @input array.
input	An array of ASCII code points (0..7F).

Parameters

output_length	The caller passes in the maximum number of code points that it can receive into the @output array (which is also the maximum number of flags that it can receive into the @case_flags array, if @case_flags 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 @output_length 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 @output 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 @output_length, @output, and @case_flags might contain garbage.

Definition at line 348 of file punycode.c.

5.13.3.2 punycode_encode()

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

punycode_encode:

Parameters

input_length	The number of code points in the @input array and the number of flags in the @case_flags 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 @input 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 @case_flags 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 @input 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 @output_size and @output might contain garbage.

Definition at line 196 of file punycode.c.

5.14 punycode.h File Reference

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

Macros

- #define IDNAPI

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

- IDNAPI const char ∗ punycode_strerror (Punycode_status rc)
- IDNAPI int punycode_encode (size_t input_length, const punycode_uint input[ ], const unsigned char case↩ _flags[ ], size_t ∗output_length, char output[ ])
- IDNAPI int punycode_decode (size_t input_length, const char input[ ], size_t ∗output_length, punycode_uint output[ ], unsigned char case_flags[ ])

5.14.1 Macro Definition Documentation

5.14.1.1 IDNAPI

```
#define IDNAPI
```

SECTION:punycode

Parameters

title	punycode.h
short_description	Punycode-related functions

Punycode-related functions.

Definition at line 85 of file punycode.h.

5.14.2 Typedef Documentation

5.14.2.1 punycode_uint

```
typedef uint32_t punycode_uint
```

Definition at line 123 of file punycode.h.

5.14.3 Enumeration Type Documentation

5.14.3.1 punycode_status

```
enum punycode_status
```

Enumerator

punycode_success	
punycode_bad_input	
punycode_big_output	
punycode_overflow	

Definition at line 100 of file punycode.h.

5.14.3.2 Punycode_status

```
enum Punycode_status
```

Enumerator

PUNYCODE_SUCCESS	

Enumerator

PUNYCODE_BAD_INPUT	
PUNYCODE_BIG_OUTPUT	
PUNYCODE_OVERFLOW	

Definition at line 108 of file punycode.h.

5.14.4 Function Documentation

5.14.4.1 punycode_decode()

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

punycode_decode:

Parameters

input_length	The number of ASCII code points in the @input 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 @output array (which is also the maximum number of flags that it can receive into the @case_flags array, if @case_flags 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 @output_length 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 @output 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 @output_length, @output, and @case_flags might contain garbage.

Definition at line 348 of file punycode.c.

5.14.4.2 punycode_encode()

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

punycode_encode:

Parameters

input_length	The number of code points in the @input array and the number of flags in the @case_flags 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 @input 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 @case_flags 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 @input 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 @output_size and @output might contain garbage.

Definition at line 196 of file punycode.c.

5.14.4.3 punycode_strerror()

```
IDNAPI const char* punycode_strerror (
            Punycode_status rc )
```

punycode_strerror:

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 @rc.

Definition at line 57 of file strerror-punycode.c.

5.15 rfc3454.c File Reference

```
#include <config.h>
#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 [ ]

5.15.1 Variable Documentation

5.15.1.1 stringprep_rfc3454_A_1

```
const Stringprep_table_element stringprep_rfc3454_A_1[]
```

Definition at line 13 of file rfc3454.c.

5.15.1.2 stringprep_rfc3454_B_1

```
const Stringprep_table_element stringprep_rfc3454_B_1[]
```

Initial value:

```
= {
  { 0x0000AD, 0x0000AD },
  { 0x00034F, 0x00034F },
  { 0x001806, 0x001806 },
  { 0x00180B, 0x00180B },
  { 0x00180C, 0x00180C },
  { 0x00180D, 0x00180D },
  { 0x00200B, 0x00200B },
  { 0x00200C, 0x00200C },
  { 0x00200D, 0x00200D },
  { 0x002060, 0x002060 },
  { 0x00FE00, 0x00FE00 },
  { 0x00FE01, 0x00FE01 },
  { 0x00FE02, 0x00FE02 },
  { 0x00FE03, 0x00FE03 },
  { 0x00FE04, 0x00FE04 },
  { 0x00FE05, 0x00FE05 },
  { 0x00FE06, 0x00FE06 },
  { 0x00FE07, 0x00FE07 },
  { 0x00FE08, 0x00FE08 },
  { 0x00FE09, 0x00FE09 },
  { 0x00FE0A, 0x00FE0A },
  { 0x00FE0B, 0x00FE0B },
  { 0x00FE0C, 0x00FE0C },
  { 0x00FE0D, 0x00FE0D },
  { 0x00FE0E, 0x00FE0E },
  { 0x00FE0F, 0x00FE0F },
  { 0x00FEFF, 0x00FEFF },
  { 0 },
}
```

Definition at line 419 of file rfc3454.c.

5.15.1.3 stringprep_rfc3454_B_2

```
const Stringprep_table_element stringprep_rfc3454_B_2[]
```

Definition at line 456 of file rfc3454.c.

5.15.1.4 stringprep_rfc3454_B_3

```
const Stringprep_table_element stringprep_rfc3454_B_3[]
```

Definition at line 2484 of file rfc3454.c.

5.15.1.5 stringprep_rfc3454_C_1_1

```
const Stringprep_table_element stringprep_rfc3454_C_1_1[]
```

Initial value:

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

Definition at line 3473 of file rfc3454.c.

5.15.1.6 stringprep_rfc3454_C_1_2

```
const Stringprep_table_element stringprep_rfc3454_C_1_2[]
```

Initial value:

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

Definition at line 3524 of file rfc3454.c.

5.15.1.7 stringprep_rfc3454_C_2_1

```
const Stringprep_table_element stringprep_rfc3454_C_2_1[]
```

Initial value:

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

Definition at line 3609 of file rfc3454.c.

5.15.1.8 stringprep_rfc3454_C_2_2

```
const Stringprep_table_element stringprep_rfc3454_C_2_2[]
```

Initial value:

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

Definition at line 3682 of file rfc3454.c.

5.15.1.9 stringprep_rfc3454_C_3

```
const Stringprep_table_element stringprep_rfc3454_C_3[]
```

Initial value:

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

Definition at line 3709 of file rfc3454.c.

5.15.1.10 stringprep_rfc3454_C_4

```
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 3723 of file rfc3454.c.

5.15.1.11 stringprep_rfc3454_C_5

```
const Stringprep_table_element stringprep_rfc3454_C_5[]
```

Initial value:

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

Definition at line 3752 of file rfc3454.c.

5.15.1.12 stringprep_rfc3454_C_6

```
const Stringprep_table_element stringprep_rfc3454_C_6[]
```

Initial value:

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

Definition at line 3763 of file rfc3454.c.

5.15.1.13 stringprep_rfc3454_C_7

```
const Stringprep_table_element stringprep_rfc3454_C_7[]
```

Initial value:

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

Definition at line 3778 of file rfc3454.c.

5.15.1.14 stringprep_rfc3454_C_8

```
const Stringprep_table_element stringprep_rfc3454_C_8[]
```

Initial value:

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

Definition at line 3791 of file rfc3454.c.

5.15.1.15 stringprep_rfc3454_C_9

```
const Stringprep_table_element stringprep_rfc3454_C_9[]
```

Initial value:

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

Definition at line 3834 of file rfc3454.c.

5.15.1.16 stringprep_rfc3454_D_1

```
const Stringprep_table_element stringprep_rfc3454_D_1[]
```

Definition at line 3846 of file rfc3454.c.

5.15.1.17 stringprep_rfc3454_D_2

```
const Stringprep_table_element stringprep_rfc3454_D_2[]
```

Definition at line 3890 of file rfc3454.c.

5.16 rfc3454.h File Reference

Macros

- #define N_STRINGPREP_rfc3454_A_1 396
- #define N_STRINGPREP_rfc3454_B_1 27
- #define N_STRINGPREP_rfc3454_B_2 1371
- #define N_STRINGPREP_rfc3454_B_3 838
- #define N_STRINGPREP_rfc3454_C_1_1 1
- #define N_STRINGPREP_rfc3454_C_1_2 17
- #define N_STRINGPREP_rfc3454_C_2_1 2
- #define N_STRINGPREP_rfc3454_C_2_2 16
- #define N_STRINGPREP_rfc3454_C_3 3
- #define N_STRINGPREP_rfc3454_C_4 18
- #define N_STRINGPREP_rfc3454_C_5 1
- #define N_STRINGPREP_rfc3454_C_6 5
- #define N_STRINGPREP_rfc3454_C_7 1
- #define N_STRINGPREP_rfc3454_C_8 15
- #define N_STRINGPREP_rfc3454_C_9 2
- #define N_STRINGPREP_rfc3454_D_1 34
- #define N_STRINGPREP_rfc3454_D_2 360

5.16.1 Macro Definition Documentation

5.16.1.1 N_STRINGPREP_rfc3454_A_1

```
#define N_STRINGPREP_rfc3454_A_1 396
```

Definition at line 4 of file rfc3454.h.

5.16.1.2 N_STRINGPREP_rfc3454_B_1

```
#define N_STRINGPREP_rfc3454_B_1 27
```

Definition at line 5 of file rfc3454.h.

5.16.1.3 N_STRINGPREP_rfc3454_B_2

```
#define N_STRINGPREP_rfc3454_B_2 1371
```

Definition at line 6 of file rfc3454.h.

5.16.1.4 N_STRINGPREP_rfc3454_B_3

```
#define N_STRINGPREP_rfc3454_B_3 838
```

Definition at line 7 of file rfc3454.h.

5.16.1.5 N_STRINGPREP_rfc3454_C_1_1

```
#define N_STRINGPREP_rfc3454_C_1_1 1
```

Definition at line 8 of file rfc3454.h.

5.16.1.6 N_STRINGPREP_rfc3454_C_1_2

```
#define N_STRINGPREP_rfc3454_C_1_2 17
```

Definition at line 9 of file rfc3454.h.

5.16.1.7 N_STRINGPREP_rfc3454_C_2_1

```
#define N_STRINGPREP_rfc3454_C_2_1 2
```

Definition at line 10 of file rfc3454.h.

5.16.1.8 N_STRINGPREP_rfc3454_C_2_2

```
#define N_STRINGPREP_rfc3454_C_2_2 16
```

Definition at line 11 of file rfc3454.h.

5.16.1.9 N_STRINGPREP_rfc3454_C_3

```
#define N_STRINGPREP_rfc3454_C_3 3
```

Definition at line 12 of file rfc3454.h.

5.16.1.10 N_STRINGPREP_rfc3454_C_4

```
#define N_STRINGPREP_rfc3454_C_4 18
```

Definition at line 13 of file rfc3454.h.

5.16.1.11 N_STRINGPREP_rfc3454_C_5

```
#define N_STRINGPREP_rfc3454_C_5 1
```

Definition at line 14 of file rfc3454.h.

5.16.1.12 N_STRINGPREP_rfc3454_C_6

```
#define N_STRINGPREP_rfc3454_C_6 5
```

Definition at line 15 of file rfc3454.h.

5.16.1.13 N_STRINGPREP_rfc3454_C_7

```
#define N_STRINGPREP_rfc3454_C_7 1
```

Definition at line 16 of file rfc3454.h.

5.16.1.14 N_STRINGPREP_rfc3454_C_8

```
#define N_STRINGPREP_rfc3454_C_8 15
```

Definition at line 17 of file rfc3454.h.

5.16.1.15 N_STRINGPREP_rfc3454_C_9

```
#define N_STRINGPREP_rfc3454_C_9 2
```

Definition at line 18 of file rfc3454.h.

5.16.1.16 N_STRINGPREP_rfc3454_D_1

```
#define N_STRINGPREP_rfc3454_D_1 34
```

Definition at line 19 of file rfc3454.h.

5.16.1.17 N_STRINGPREP_rfc3454_D_2

```
#define N_STRINGPREP_rfc3454_D_2 360
```

Definition at line 20 of file rfc3454.h.

5.17 strerror-idna.c File Reference

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

Macros

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

Functions

- const char * idna_strerror (Idna_rc rc)

5.17.1 Macro Definition Documentation

5.17.1.1 _

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

Definition at line 37 of file strerror-idna.c.

5.17.2 Function Documentation

5.17.2.1 idna_strerror()

```
const char* idna_strerror (
            Idna_rc rc )
```

idna_strerror:

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: Character encoding conversion error. 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 @rc.

Definition at line 73 of file strerror-idna.c.

5.18 strerror-pr29.c File Reference

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

Macros

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

Functions

- const char ∗ pr29_strerror (Pr29_rc rc)

5.18.1 Macro Definition Documentation

5.18.1.1 _

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

Definition at line 37 of file strerror-pr29.c.

5.18.2 Function Documentation

5.18.2.1 pr29_strerror()

```
const char* pr29_strerror (
            Pr29_rc rc )
```

pr29_strerror:

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_8z()).

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

Definition at line 57 of file strerror-pr29.c.

5.19 strerror-punycode.c File Reference

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

Macros

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

Functions

- const char * punycode_strerror (Punycode_status rc)

5.19.1 Macro Definition Documentation

5.19.1.1 _

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

Definition at line 37 of file strerror-punycode.c.

5.19.2 Function Documentation

5.19.2.1 punycode_strerror()

```
const char* punycode_strerror (
            Punycode_status rc )
```

punycode_strerror:

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 @rc.

Definition at line 57 of file strerror-punycode.c.

5.20 strerror-stringprep.c File Reference

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

Macros

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

Functions

- const char ∗ stringprep_strerror (Stringprep_rc rc)

5.20.1 Macro Definition Documentation

5.20.1.1 _

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

Definition at line 37 of file strerror-stringprep.c.

5.20.2 Function Documentation

5.20.2.1 stringprep_strerror()

```
const char* stringprep_strerror (
            Stringprep_rc rc )
```

stringprep_strerror:

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_ICONV_ERROR: Character encoding conversion error. 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 @rc.

Definition at line 78 of file strerror-stringprep.c.

5.21 strerror-tld.c File Reference

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

Macros

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

Functions

- const char ∗ tld_strerror (Tld_rc rc)

5.21.1 Macro Definition Documentation

5.21.1.1 _

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

Definition at line 37 of file strerror-tld.c.

5.21.2 Function Documentation

5.21.2.1 tld_strerror()

```
const char* tld_strerror (
            Tld_rc rc )
```

tld_strerror:

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: Character encoding conversion error. 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 @rc.

Definition at line 59 of file strerror-tld.c.

5.22 stringprep.c File Reference

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

Macros

- #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)

5.22.1 Macro Definition Documentation

5.22.1.1 INVERTED

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

Definition at line 144 of file stringprep.c.

5.22.1.2 UNAPPLICAPLEFLAGS

```
#define UNAPPLICAPLEFLAGS(
            flags,
            profileflags )
```

Value:

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

Definition at line 145 of file stringprep.c.

5.22.2 Function Documentation

5.22.2.1 stringprep()

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

stringprep:

Parameters

in	input/output 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, @maxlen indicate how large the buffer holding the string is. This function will not read or write to characters outside that size.

The @flags are one of Stringprep_profile_flags values, or 0.

The @profile 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 414 of file stringprep.c.

5.22.2.2 stringprep_4i()

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

stringprep_4i:

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 (@ucs4[@len] = 0). The output will not be zero terminated unless @ucs4[@len] = 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, @maxucs4len indicate how large the buffer holding the string is. This function will not read or write to code points outside that size.

The @flags are one of Stringprep_profile_flags values, or 0.

The @profile 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 181 of file stringprep.c.

5.22.2.3 stringprep_4zi()

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

stringprep_4zi:

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, @maxucs4len indicate how large the buffer holding the string is. This function will not read or write to code points outside that size.

The @flags are one of Stringprep_profile_flags values, or 0.

The @profile 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 374 of file stringprep.c.

5.22.2.4 stringprep_profile()

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

stringprep_profile:

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 @out variable must be deallocated by the caller.

The @flags are one of Stringprep_profile_flags values, or 0.

The @profile 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 493 of file stringprep.c.

5.23 stringprep.h File Reference

```
#include <stddef.h>
#include <sys/types.h>
#include <idn-int.h>
```

Data Structures

- struct Stringprep_table_element
- struct Stringprep_table
- struct Stringprep_profiles

Macros

- #define IDNAPI
- #define STRINGPREP_VERSION "1.42"
- #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 struct Stringprep_table_element Stringprep_table_element
- typedef struct Stringprep_table Stringprep_profile
- typedef struct 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_ICONV_ERROR = 104 , 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

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

Variables

- IDNAPI const Stringprep_profiles stringprep_profiles [ ]
- IDNAPI const Stringprep_table_element stringprep_rfc3454_A_1 [ ]
- IDNAPI const Stringprep_table_element stringprep_rfc3454_B_1 [ ]
- IDNAPI const Stringprep_table_element stringprep_rfc3454_B_2 [ ]
- IDNAPI const Stringprep_table_element stringprep_rfc3454_B_3 [ ]
- IDNAPI const Stringprep_table_element stringprep_rfc3454_C_1_1 [ ]
- IDNAPI const Stringprep_table_element stringprep_rfc3454_C_1_2 [ ]
- IDNAPI const Stringprep_table_element stringprep_rfc3454_C_2_1 [ ]
- IDNAPI const Stringprep_table_element stringprep_rfc3454_C_2_2 [ ]
- IDNAPI const Stringprep_table_element stringprep_rfc3454_C_3 [ ]
- IDNAPI const Stringprep_table_element stringprep_rfc3454_C_4 [ ]

- IDNAPI const Stringprep_table_element stringprep_rfc3454_C_5 [ ]
- IDNAPI const Stringprep_table_element stringprep_rfc3454_C_6 [ ]
- IDNAPI const Stringprep_table_element stringprep_rfc3454_C_7 [ ]
- IDNAPI const Stringprep_table_element stringprep_rfc3454_C_8 [ ]
- IDNAPI const Stringprep_table_element stringprep_rfc3454_C_9 [ ]
- IDNAPI const Stringprep_table_element stringprep_rfc3454_D_1 [ ]
- IDNAPI const Stringprep_table_element stringprep_rfc3454_D_2 [ ]
- IDNAPI const Stringprep_profile stringprep_nameprep [ ]
- IDNAPI const Stringprep_profile stringprep_saslprep [ ]
- IDNAPI const Stringprep_table_element stringprep_saslprep_space_map [ ]
- IDNAPI const Stringprep_profile stringprep_plain [ ]
- IDNAPI const Stringprep_profile stringprep_trace [ ]
- IDNAPI const Stringprep_profile stringprep_kerberos5 [ ]
- IDNAPI const Stringprep_profile stringprep_xmpp_nodeprep [ ]
- IDNAPI const Stringprep_profile stringprep_xmpp_resourceprep [ ]
- IDNAPI const Stringprep_table_element stringprep_xmpp_nodeprep_prohibit [ ]
- IDNAPI const Stringprep_profile stringprep_iscsi [ ]
- IDNAPI const Stringprep_table_element stringprep_iscsi_prohibit [ ]

5.23.1 Macro Definition Documentation

5.23.1.1 IDNAPI

```
#define IDNAPI
```

SECTION:stringprep

Parameters

title	stringprep.h
short_description	Stringprep-related functions

Stringprep-related functions.

Definition at line 49 of file stringprep.h.

5.23.1.2 stringprep_iscsi

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

Definition at line 243 of file stringprep.h.

5.23.1.3 stringprep_kerberos5

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

Definition at line 223 of file stringprep.h.

5.23.1.4 STRINGPREP_MAX_MAP_CHARS

```
#define STRINGPREP_MAX_MAP_CHARS 4
```

Definition at line 106 of file stringprep.h.

5.23.1.5 stringprep_nameprep

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

Definition at line 202 of file stringprep.h.

5.23.1.6 stringprep_nameprep_no_unassigned

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

Definition at line 205 of file stringprep.h.

5.23.1.7 stringprep_plain

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

Definition at line 216 of file stringprep.h.

5.23.1.8 STRINGPREP_VERSION

```
#define STRINGPREP_VERSION "1.42"
```

Definition at line 62 of file stringprep.h.

5.23.1.9 stringprep_xmpp_nodeprep

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

Definition at line 233 of file stringprep.h.

5.23.1.10 stringprep_xmpp_resourceprep

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

Definition at line 235 of file stringprep.h.

5.23.2 Typedef Documentation

5.23.2.1 Stringprep_profile

```
typedef struct Stringprep_table Stringprep_profile
```

Stringprep_profile:

Stringprep profile table.

Definition at line 1 of file stringprep.h.

5.23.2.2 Stringprep_profiles

```
typedef struct Stringprep_profiles Stringprep_profiles
```

Definition at line 1 of file stringprep.h.

5.23.2.3 Stringprep_table_element

```
typedef struct Stringprep_table_element Stringprep_table_element
```

Definition at line 1 of file stringprep.h.

5.23.3 Enumeration Type Documentation

5.23.3.1 Stringprep_profile_flags

```
enum Stringprep_profile_flags
```

Enumerator

STRINGPREP_NO_NFKC	
STRINGPREP_NO_BIDI	
STRINGPREP_NO_UNASSIGNED	

Definition at line 86 of file stringprep.h.

5.23.3.2 Stringprep_profile_steps

```
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 94 of file stringprep.h.

5.23.3.3 Stringprep_rc

```
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_ICONV_ERROR	
STRINGPREP_NFKC_FAILED	
STRINGPREP_MALLOC_ERROR	

Definition at line 65 of file stringprep.h.

5.23.4 Function Documentation

5.23.4.1 stringprep()

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

stringprep:

Parameters

in	input/output 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, @maxlen indicate how large the buffer holding the string is. This function will not read or write to characters outside that size.

The @flags are one of Stringprep_profile_flags values, or 0.

The @profile 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 414 of file stringprep.c.

5.23.4.2 stringprep_4i()

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

stringprep_4i:

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 (@ucs4[@len] = 0). The output will not be zero terminated unless @ucs4[@len] = 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, @maxucs4len indicate how large the buffer holding the string is. This function will not read or write to code points outside that size.

The @flags are one of Stringprep_profile_flags values, or 0.

The @profile 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 181 of file stringprep.c.

5.23.4.3 stringprep_4zi()

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

stringprep_4zi:

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, @maxucs4len indicate how large the buffer holding the string is. This function will not read or write to code points outside that size.

The @flags are one of Stringprep_profile_flags values, or 0.

The @profile 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 374 of file stringprep.c.

5.23.4.4 stringprep_check_version()

```
IDNAPI const char* stringprep_check_version (
            const char * req_version )
```

stringprep_check_version:

Parameters

req_version	Required version number, or NULL.

Check that 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 @req_version 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 53 of file version.c.

5.23.4.5 stringprep_convert()

```
IDNAPI char* stringprep_convert (
            const char * str,
            const char * to_codeset,
            const char * from_codeset )
```

stringprep_convert:

Parameters

str	input zero-terminated string.
to_codeset	name of destination character set.
from_codeset	name of origin character set, as used by @str.

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 @str transcoded into to_codeset.

Definition at line 116 of file toutf8.c.

5.23.4.6 stringprep_locale_charset()

```
IDNAPI const char* stringprep_locale_charset (
            void )
```

stringprep_locale_charset:

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 preferred 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 85 of file toutf8.c.

5.23.4.7 stringprep_locale_to_utf8()

```
IDNAPI char* stringprep_locale_to_utf8 (
            const char * str )
```

stringprep_locale_to_utf8:

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 @str transcoded into UTF-8.

Definition at line 145 of file toutf8.c.

5.23.4.8 stringprep_profile()

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

stringprep_profile:

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 @out variable must be deallocated by the caller.

The @flags are one of Stringprep_profile_flags values, or 0.

The @profile 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 493 of file stringprep.c.

5.23.4.9 stringprep_strerror()

```
IDNAPI const char* stringprep_strerror (
            Stringprep_rc rc )
```

stringprep_strerror:

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_ICONV_ERROR: Character encoding conversion error. 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 @rc.

Definition at line 78 of file strerror-stringprep.c.

5.23.4.10 stringprep_ucs4_nfkc_normalize()

```
IDNAPI uint32_t* stringprep_ucs4_nfkc_normalize (
            const uint32_t * str,
            ssize_t len )
```

stringprep_ucs4_nfkc_normalize:

Parameters

str	a Unicode string.
len	length of @str array, or -1 if @str is nul-terminated.

Converts a UCS4 string into canonical form, see stringprep_utf8_nfkc_normalize() for more information.

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

Definition at line 1096 of file nfkc.c.

5.23.4.11 stringprep_ucs4_to_utf8()

```
IDNAPI char* stringprep_ucs4_to_utf8 (
            const uint32_t * str,
```

```
            ssize_t len,
            size_t * items_read,
            size_t * items_written )
```

stringprep_ucs4_to_utf8:

Parameters

str	a UCS-4 encoded string
len	the maximum length of @str to use. If @len $<$ 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 deallocated by the caller. If an error occurs, NULL will be returned.

Definition at line 1039 of file nfkc.c.

5.23.4.12 stringprep_unichar_to_utf8()

```
IDNAPI int stringprep_unichar_to_utf8 (
            uint32_t c,
            char * outbuf )
```

stringprep_unichar_to_utf8:

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 @outbuf.

Converts a single character to UTF-8.

Return value: number of bytes written.

Definition at line 982 of file nfkc.c.

5.23.4.13 stringprep_utf8_nfkc_normalize()

```
IDNAPI char* stringprep_utf8_nfkc_normalize (
            const char * str,
            ssize_t len )
```

stringprep_utf8_nfkc_normalize:

Parameters

str	a UTF-8 encoded string.
len	length of @str, in bytes, or -1 if @str 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 @str.

Definition at line 1068 of file nfkc.c.

5.23.4.14 stringprep_utf8_to_locale()

```
IDNAPI char* stringprep_utf8_to_locale (
            const char * str )
```

stringprep_utf8_to_locale:

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 @str transcoded into the locale's character set.

Definition at line 161 of file toutf8.c.

5.23.4.15 stringprep_utf8_to_ucs4()

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

stringprep_utf8_to_ucs4:

Parameters

str	a UTF-8 encoded string
len	the maximum length of @str to use. If @len $<$ 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. The function now performs error checking to verify that the input is valid UTF-8 (before it was documented to not do error checking).

Return value: a pointer to a newly allocated UCS-4 string. This value must be deallocated by the caller.

Definition at line 1006 of file nfkc.c.

5.23.4.16 stringprep_utf8_to_unichar()

```
IDNAPI uint32_t stringprep_utf8_to_unichar (
            const char * p )
```

stringprep_utf8_to_unichar:

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 965 of file nfkc.c.

5.23.5 Variable Documentation

5.23.5.1 stringprep_iscsi

```
IDNAPI const Stringprep_profile stringprep_iscsi[] [extern]
```

Definition at line 197 of file profiles.c.

5.23.5.2 stringprep_iscsi_prohibit

```
IDNAPI const Stringprep_table_element stringprep_iscsi_prohibit[] [extern]
```

Definition at line 187 of file profiles.c.

5.23.5.3 stringprep_kerberos5

```
IDNAPI const Stringprep_profile stringprep_kerberos5[] [extern]
```

Definition at line 74 of file profiles.c.

5.23.5.4 stringprep_nameprep

```
IDNAPI const Stringprep_profile stringprep_nameprep[] [extern]
```

Definition at line 53 of file profiles.c.

5.23.5.5 stringprep_plain

```
IDNAPI const Stringprep_profile stringprep_plain[] [extern]
```

Definition at line 155 of file profiles.c.

5.23.5.6 stringprep_profiles

```
IDNAPI const Stringprep_profiles stringprep_profiles[] [extern]
```

Definition at line 34 of file profiles.c.

5.23.5.7 stringprep_rfc3454_A_1

```
IDNAPI const Stringprep_table_element stringprep_rfc3454_A_1[] [extern]
```

Definition at line 13 of file rfc3454.c.

5.23.5.8 stringprep_rfc3454_B_1

```
IDNAPI const Stringprep_table_element stringprep_rfc3454_B_1[] [extern]
```

Definition at line 419 of file rfc3454.c.

5.23.5.9 stringprep_rfc3454_B_2

```
IDNAPI const Stringprep_table_element stringprep_rfc3454_B_2[] [extern]
```

Definition at line 456 of file rfc3454.c.

5.23.5.10 stringprep_rfc3454_B_3

```
IDNAPI const Stringprep_table_element stringprep_rfc3454_B_3[] [extern]
```

Definition at line 2484 of file rfc3454.c.

5.23.5.11 stringprep_rfc3454_C_1_1

```
IDNAPI const Stringprep_table_element stringprep_rfc3454_C_1_1[] [extern]
```

Definition at line 3473 of file rfc3454.c.

5.23.5.12 stringprep_rfc3454_C_1_2

```
IDNAPI const Stringprep_table_element stringprep_rfc3454_C_1_2[] [extern]
```

Definition at line 3524 of file rfc3454.c.

5.23.5.13 stringprep_rfc3454_C_2_1

```
IDNAPI const Stringprep_table_element stringprep_rfc3454_C_2_1[] [extern]
```

Definition at line 3609 of file rfc3454.c.

5.23.5.14 stringprep_rfc3454_C_2_2

```
IDNAPI const Stringprep_table_element stringprep_rfc3454_C_2_2[] [extern]
```

Definition at line 3682 of file rfc3454.c.

5.23.5.15 stringprep_rfc3454_C_3

```
IDNAPI const Stringprep_table_element stringprep_rfc3454_C_3[] [extern]
```

Definition at line 3709 of file rfc3454.c.

5.23.5.16 stringprep_rfc3454_C_4

```
IDNAPI const Stringprep_table_element stringprep_rfc3454_C_4[] [extern]
```

Definition at line 3723 of file rfc3454.c.

5.23.5.17 stringprep_rfc3454_C_5

```
IDNAPI const Stringprep_table_element stringprep_rfc3454_C_5[] [extern]
```

Definition at line 3752 of file rfc3454.c.

5.23.5.18 stringprep_rfc3454_C_6

```
IDNAPI const Stringprep_table_element stringprep_rfc3454_C_6[] [extern]
```

Definition at line 3763 of file rfc3454.c.

5.23.5.19 stringprep_rfc3454_C_7

```
IDNAPI const Stringprep_table_element stringprep_rfc3454_C_7[] [extern]
```

Definition at line 3778 of file rfc3454.c.

5.23.5.20 stringprep_rfc3454_C_8

```
IDNAPI const Stringprep_table_element stringprep_rfc3454_C_8[] [extern]
```

Definition at line 3791 of file rfc3454.c.

5.23.5.21 stringprep_rfc3454_C_9

```
IDNAPI const Stringprep_table_element stringprep_rfc3454_C_9[] [extern]
```

Definition at line 3834 of file rfc3454.c.

5.23.5.22 stringprep_rfc3454_D_1

```
IDNAPI const Stringprep_table_element stringprep_rfc3454_D_1[] [extern]
```

Definition at line 3846 of file rfc3454.c.

5.23.5.23 stringprep_rfc3454_D_2

```
IDNAPI const Stringprep_table_element stringprep_rfc3454_D_2[] [extern]
```

Definition at line 3890 of file rfc3454.c.

5.23.5.24 stringprep_saslprep

```
IDNAPI const Stringprep_profile stringprep_saslprep[] [extern]
```

Definition at line 243 of file profiles.c.

5.23.5.25 stringprep_saslprep_space_map

```
IDNAPI const Stringprep_table_element stringprep_saslprep_space_map[] [extern]
```

Definition at line 222 of file profiles.c.

5.23.5.26 stringprep_trace

```
IDNAPI const Stringprep_profile stringprep_trace[] [extern]
```

Definition at line 171 of file profiles.c.

5.23.5.27 stringprep_xmpp_nodeprep

```
IDNAPI const Stringprep_profile stringprep_xmpp_nodeprep[] [extern]
```

Definition at line 109 of file profiles.c.

5.23.5.28 stringprep_xmpp_nodeprep_prohibit

```
IDNAPI const Stringprep_table_element stringprep_xmpp_nodeprep_prohibit[] [extern]
```

Definition at line 97 of file profiles.c.

5.23.5.29 stringprep_xmpp_resourceprep

```
IDNAPI const Stringprep_profile stringprep_xmpp_resourceprep[] [extern]
```

Definition at line 134 of file profiles.c.

5.24 tld.c File Reference

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

Macros

- #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 [ ]

5.24.1 Macro Definition Documentation

5.24.1.1 DOTP

```
#define DOTP(
            c )
```

Value:

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

Definition at line 105 of file tld.c.

5.24.2 Function Documentation

5.24.2.1 tld_check_4()

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

tld_check_4:

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

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

Definition at line 359 of file tld.c.

5.24.2.2 tld_check_4t()

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

tld_check_4t:

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 @in for whether or not they are allowed by the data structure in @tld, return the position of the first character for which this is not the case in @errpos.

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

Definition at line 280 of file tld.c.

5.24.2.3 tld_check_4tz()

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

tld_check_4tz:

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 @in for whether or not they are allowed by the data structure in @tld, return the position of the first character for which this is not the case in @errpos.

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

Definition at line 322 of file tld.c.

5.24.2.4 tld_check_4z()

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

tld_check_4z:

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

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

Definition at line 409 of file tld.c.

5.24.2.5 tld_check_8z()

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

tld_check_8z:

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 @in for whether or not they are allowed by the information in @overrides or by the built-in TLD restriction data. When data for the same TLD is available both internally and in @overrides, the

information in @overrides takes precedence. If several entries for a specific TLD are found, the first one is used. If @overrides is NULL, only the built-in information is used. The position of the first offending character is returned in @errpos. 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 @tld is null, TLD_INVALID if a character is not allowed, or additional error codes on general failure conditions.

Definition at line 446 of file tld.c.

5.24.2.6 tld_check_lz()

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

tld_check_lz:

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 @in for whether or not they are allowed by the information in @overrides or by the built-in TLD restriction data. When data for the same TLD is available both internally and in @overrides, the information in @overrides takes precedence. If several entries for a specific TLD are found, the first one is used. If @overrides is NULL, only the built-in information is used. The position of the first offending character is returned in @errpos. 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 @tld is null, TLD_INVALID if a character is not allowed, or additional error codes on general failure conditions.

Definition at line 491 of file tld.c.

5.24.2.7 tld_default_table()

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

tld_default_table:

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
	default tables.

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

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

Definition at line 89 of file tld.c.

5.24.2.8 tld_get_4()

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

tld_get_4:

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 @in and return it as an ASCII string in @out.

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

Definition at line 122 of file tld.c.

5.24.2.9 tld_get_4z()

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

tld_get_4z:

Parameters

in	Zero terminated array of unicode code points to process.
out	Zero terminated ascii result string pointer.

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

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

Definition at line 171 of file tld.c.

5.24.2.10 tld_get_table()

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

tld_get_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 @tld by going thru @tables, or return NULL if no such structure is found.

Definition at line 60 of file tld.c.

5.24.2.11 tld_get_z()

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

tld_get_z:

Parameters

in	Zero terminated character array to process.
out	Zero terminated ascii result string pointer.

Isolate the top-level domain of @in and return it as an ASCII string in @out. The input string @in 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 197 of file tld.c.

5.24.3 Variable Documentation

5.24.3.1 _tld_tables

```
const Tld_table* _tld_tables[]  [extern]
```

Definition at line 60 of file tlds.c.

5.25 tld.h File Reference

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

Data Structures

- struct Tld_table_element
- struct Tld_table

Macros

- #define IDNAPI

Typedefs

- typedef struct Tld_table_element Tld_table_element
- typedef struct 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

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

5.25.1 Macro Definition Documentation

5.25.1.1 IDNAPI

```
#define IDNAPI
```

SECTION:tld

Parameters

title	tld.h
short_description	TLD-related functions

TLD-related functions.

Definition at line 52 of file tld.h.

5.25.2 Typedef Documentation

5.25.2.1 Tld_table

```
typedef struct Tld_table Tld_table
```

Definition at line 1 of file tld.h.

5.25.2.2 Tld_table_element

```
typedef struct Tld_table_element Tld_table_element
```

Definition at line 1 of file tld.h.

5.25.3 Enumeration Type Documentation

5.25.3.1 Tld_rc

```
enum Tld_rc
```

Enumerator

TLD_SUCCESS	
TLD_INVALID	
TLD_NODATA	
TLD_MALLOC_ERROR	
TLD_ICONV_ERROR	
TLD_NO_TLD	
TLD_NOTLD	

Definition at line 105 of file tld.h.

5.25.4 Function Documentation

5.25.4.1 tld_check_4()

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

tld_check_4:

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

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

Definition at line 359 of file tld.c.

5.25.4.2 tld_check_4t()

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

tld_check_4t:

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 @in for whether or not they are allowed by the data structure in @tld, return the position of the first character for which this is not the case in @errpos.

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

Definition at line 280 of file tld.c.

5.25.4.3 tld_check_4tz()

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

tld_check_4tz:

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 @in for whether or not they are allowed by the data structure in @tld, return the position of the first character for which this is not the case in @errpos.

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

Definition at line 322 of file tld.c.

5.25.4.4 tld_check_4z()

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

tld_check_4z:

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

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

Definition at line 409 of file tld.c.

5.25.4.5 tld_check_8z()

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

tld_check_8z:

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 @in for whether or not they are allowed by the information in @overrides or by the built-in TLD restriction data. When data for the same TLD is available both internally and in @overrides, the information in @overrides takes precedence. If several entries for a specific TLD are found, the first one is used. If @overrides is NULL, only the built-in information is used. The position of the first offending character is returned in @errpos. 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 @tld is null, TLD_INVALID if a character is not allowed, or additional error codes on general failure conditions.

Definition at line 446 of file tld.c.

5.25.4.6 tld_check_lz()

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

tld_check_lz:

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 @in for whether or not they are allowed by the information in @overrides or by the built-in TLD restriction data. When data for the same TLD is available both internally and in @overrides, the information in @overrides takes precedence. If several entries for a specific TLD are found, the first one is used. If @overrides is NULL, only the built-in information is used. The position of the first offending character is returned in @errpos. 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 @tld is null, TLD_INVALID if a character is not allowed, or additional error codes on general failure conditions.

Definition at line 491 of file tld.c.

5.25.4.7 tld_default_table()

```
IDNAPI const Tld_table* tld_default_table (
            const char * tld,
            const Tld_table ** overrides )
```

tld_default_table:

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 default tables.

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

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

Definition at line 89 of file tld.c.

5.25.4.8 tld_get_4()

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

tld_get_4:

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 @in and return it as an ASCII string in @out.

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

Definition at line 122 of file tld.c.

5.25.4.9 tld_get_4z()

```
IDNAPI int tld_get_4z (
            const uint32_t * in,
            char ** out )
```

tld_get_4z:

Parameters

in	Zero terminated array of unicode code points to process.
out	Zero terminated ascii result string pointer.

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

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

Definition at line 171 of file tld.c.

5.25.4.10 tld_get_table()

```
IDNAPI const Tld_table* tld_get_table (
            const char * tld,
            const Tld_table ** tables )
```

tld_get_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 @tld by going thru @tables, or return NULL if no such structure is found.

Definition at line 60 of file tld.c.

5.25.4.11 tld_get_z()

```
IDNAPI int tld_get_z (
            const char * in,
            char ** out )
```

tld_get_z:

Parameters

in	Zero terminated character array to process.
out	Zero terminated ascii result string pointer.

Isolate the top-level domain of @in and return it as an ASCII string in @out. The input string @in 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 197 of file tld.c.

5.25.4.12 tld_strerror()

```
IDNAPI const char* tld_strerror (
            Tld_rc rc )
```

tld_strerror:

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: Character encoding conversion error. 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 @rc.

Definition at line 59 of file strerror-tld.c.

5.26 tlds.c File Reference

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

Variables

- const Tld_table * _tld_tables [ ]

5.26.1 Variable Documentation

5.26.1.1 _tld_tables

```
const Tld_table* _tld_tables[]
```

Initial value:

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

Definition at line 60 of file tlds.c.

5.27 toutf8.c File Reference

```
#include "stringprep.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "striconv.h"
#include <locale.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)

5.27.1 Function Documentation

5.27.1.1 stringprep_convert()

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

stringprep_convert:

Parameters

str	input zero-terminated string.
to_codeset	name of destination character set.
from_codeset	name of origin character set, as used by @str.

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 @str transcoded into to_codeset.

Definition at line 116 of file toutf8.c.

5.27.1.2 stringprep_locale_charset()

```
const char* stringprep_locale_charset (
            void  )
```

stringprep_locale_charset:

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 preferred 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 85 of file toutf8.c.

5.27.1.3 stringprep_locale_to_utf8()

```
char* stringprep_locale_to_utf8 (
            const char * str )
```

stringprep_locale_to_utf8:

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 @str transcoded into UTF-8.

Definition at line 145 of file toutf8.c.

5.27.1.4 stringprep_utf8_to_locale()

```
char* stringprep_utf8_to_locale (
            const char * str )
```

stringprep_utf8_to_locale:

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 @str transcoded into the locale's character set.

Definition at line 161 of file toutf8.c.

5.28 version.c File Reference

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

Functions

- const char * stringprep_check_version (const char *req_version)

5.28.1 Function Documentation

5.28.1.1 stringprep_check_version()

```
const char* stringprep_check_version (
            const char * req_version )
```

stringprep_check_version:

Parameters

req_version	Required version number, or NULL.

Check that 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 @req_version 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 53 of file version.c.

Index