Libidn2 Reference Manual

COLLABORATORS			
	TITLE : Libidn2 Reference Manual		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		May 18, 2018	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

1 Libidn2 Overview 1

1.1 idn2 1

2 Index 22

Chapter 1

Libidn2 Overview

Libidn2 is a free software implementation of IDNA2008 and TR46.

1.1 idn2

idn2 —

Functions

int	idn2_lookup_u8 ()
int	idn2_register_u8 ()
int	idn2_lookup_ul ()
int	idn2_register_ul ()
int	idn2_to_ascii_4i ()
int	idn2_to_ascii_4z ()
int	idn2_to_ascii_8z ()
int	idn2_to_ascii_lz ()
int	idn2_to_unicode_8z4z ()
int	idn2_to_unicode_4z4z ()
int	idn2_to_unicode_44i ()
int	idn2_to_unicode_8z8z ()
int	idn2_to_unicode_8zlz ()
int	idn2_to_unicode_lzlz ()
const char *	idn2_strerror ()
const char *	idn2_strerror_name ()
const char *	idn2_check_version ()
void	idn2_free ()
#define	idna_to_ascii_4i()
#define	idna_to_ascii_4z()
#define	idna_to_ascii_8z()
#define	idna_to_ascii_lz()

Types and Values

#define	G_GNUC_IDN2_ATTRIBUTE_PURE
#define	G_GNUC_IDN2_ATTRIBUTE_CONST
#define	G_GNUC_UNUSED

#define	IDN2_VERSION
#define	IDN2_VERSION_NUMBER
#define	IDN2_VERSION_MAJOR
#define	IDN2_VERSION_MINOR
#define	IDN2_VERSION_PATCH
#define	IDN2_LABEL_MAX_LENGTH
#define	IDN2_DOMAIN_MAX_LENGTH
enum	idn2_flags
enum	idn2_rc
enum	Idna_rc
enum	Idna_flags
#define	idna_to_unicode_8z4z
#define	idna_to_unicode_4z4z
#define	idna_to_unicode_44i
#define	idna_to_unicode_8z8z
#define	idna_to_unicode_8zlz
#define	idna_to_unicode_lzlz
#define	idna_strerror
#define	idn_free

Description

Functions

idn2_lookup_u8 ()

```
int
idn2_lookup_u8 (const uint8_t *src,
                uint8_t **lookupname,
                int flags);
```

Perform IDNA2008 lookup string conversion on domain name `src` , as described in section 5 of RFC 5891. Note that the input string must be encoded in UTF-8 and be in Unicode NFC form.

Pass IDN2_NFC_INPUT in `flags` to convert input to NFC form before further processing. IDN2_TRANSITIONAL and IDN2_NONTRANSITIONAL do already imply IDN2_NFC_INPUT. Pass IDN2_ALABEL_ROUNDTRIP in `flags` to convert any input A-labels to U-labels and perform additional testing (not implemented yet). Pass IDN2_TRANSITIONAL to enable Unicode TR46 transitional processing, and IDN2_NONTRANSITIONAL to enable Unicode TR46 non-transitional processing. Multiple flags may be specified by binary or:ing them together.

After version 2.0.3: IDN2_USE_STD3_ASCII_RULES disabled by default. Previously we were eliminating non-STD3 characters from domain strings such as _443._tcp.example.com, or IPs 1.2.3.4/24 provided to libidn2 functions. That was an unexpected regression for applications switching from libidn and thus it is no longer applied by default. Use IDN2_USE_STD3_ASCII_RULES to enable that behavior again.

After version 0.11: `lookupname` may be NULL to test lookup of `src` without allocating memory.

Parameters

src	input zero-terminated UTF-8 string in Unicode NFC normalized form.	
lookupname	newly allocated output variable with name to lookup in DNS.	
flags	optional idn2_flags to modify behaviour.	

Returns

On successful conversion IDN2_OK is returned, if the output domain or any label would have been too long IDN2_TOO_BIG_DOMAIN or IDN2_TOO_BIG_LABEL is returned, or another error code is returned.

Since: 0.1

idn2_register_u8 ()

```
int
idn2_register_u8 (const uint8_t *ulabel,
                  const uint8_t *alabel,
                  uint8_t **insertname,
                  int flags);
```

Perform IDNA2008 register string conversion on domain label `ulabel` and `alabel` , as described in section 4 of RFC 5891. Note that the input `ulabel` must be encoded in UTF-8 and be in Unicode NFC form.

Pass IDN2_NFC_INPUT in `flags` to convert input `ulabel` to NFC form before further processing.

It is recommended to supply both `ulabel` and `alabel` for better error checking, but supplying just one of them will work. Passing in only `alabel` is better than only `ulabel` . See RFC 5891 section 4 for more information.

After version 0.11: `insertname` may be NULL to test conversion of `src` without allocating memory.

Parameters

ulabel	input zero-terminated UTF-8 and Unicode NFC string, or NULL.	
alabel	input zero-terminated ACE encoded string (xn--), or NULL.	
insertname	newly allocated output variable with name to register in DNS.	
flags	optional idn2_flags to modify behaviour.	

Returns

On successful conversion IDN2_OK is returned, when the given `ulabel` and `alabel` does not match each other IDN2_UALABEL_MIS is returned, when either of the input labels are too long IDN2_TOO_BIG_LABEL is returned, when `alabel` does does not appear to be a proper A-label IDN2_INVALID_ALABEL is returned, or another error code is returned.

idn2_lookup_ul ()

```
int
idn2_lookup_ul (const char *src,
                char **lookupname,
                int flags);
```

Perform IDNA2008 lookup string conversion on domain name `src` , as described in section 5 of RFC 5891. Note that the input is assumed to be encoded in the locale's default coding system, and will be transcoded to UTF-8 and NFC normalized by this function.

Pass IDN2_ALABEL_ROUNDTRIP in `flags` to convert any input A-labels to U-labels and perform additional testing. Pass IDN2_TRANSITIONAL to enable Unicode TR46 transitional processing, and IDN2_NONTRANSITIONAL to enable Unicode

TR46 non-transitional processing. Multiple flags may be specified by binary or:ing them together, for example IDN2_ALABEL_ROUND | IDN2_NONTRANSITIONAL. The IDN2_NFC_INPUT in *flags* is always enabled in this function.

After version 0.11: *lookupname* may be NULL to test lookup of *src* without allocating memory.

Parameters

src	input zero-terminated locale encoded string.	
lookupname	newly allocated output variable with name to lookup in DNS.	
flags	optional idn2_flags to modify behaviour.	

Returns

On successful conversion IDN2_OK is returned, if conversion from locale to UTF-8 fails then IDN2_ICONV_FAIL is returned, if the output domain or any label would have been too long IDN2_TOO_BIG_DOMAIN or IDN2_TOO_BIG_LABEL is returned, or another error code is returned.

Since: 0.1

idn2_register_ul ()

```
int
idn2_register_ul (const char *ulabel,
                  const char *alabel,
                  char **insertname,
                  int flags);
```

Perform IDNA2008 register string conversion on domain label *ulabel* and *alabel* , as described in section 4 of RFC 5891. Note that the input *ulabel* is assumed to be encoded in the locale's default coding system, and will be transcoded to UTF-8 and NFC normalized by this function.

It is recommended to supply both *ulabel* and *alabel* for better error checking, but supplying just one of them will work. Passing in only *alabel* is better than only *ulabel* . See RFC 5891 section 4 for more information.

After version 0.11: *insertname* may be NULL to test conversion of *src* without allocating memory.

Parameters

ulabel	input zero-terminated locale encoded string, or NULL.	
alabel	input zero-terminated ACE encoded string (xn--), or NULL.	
insertname	newly allocated output variable with name to register in DNS.	
flags	optional idn2_flags to modify behaviour.	

Returns

On successful conversion IDN2_OK is returned, when the given `ulabel` and `alabel` does not match each other IDN2_UALABEL_MIS is returned, when either of the input labels are too long IDN2_TOO_BIG_LABEL is returned, when `alabel` does does not appear to be a proper A-label IDN2_INVALID_ALABEL is returned, when `ulabel` locale to UTF-8 conversion failed IDN2_ICONV_FAIL is returned, or another error code is returned.

idn2_to_ascii_4i ()

```
int
idn2_to_ascii_4i (const uint32_t *input,
                  size_t inlen,
                  char *output,
                  int flags);
```

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 dealing with this failure is application-specific.

The inputs to ToASCII are a sequence of code points.

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 effect as applying it just once.

The default behavior of this function (when flags are zero) is to apply the IDNA2008 rules without the TR46 amendments. As the TR46 non-transitional processing is nowdays ubiquitous, when unsure, it is recommended to call this function with the IDN2_NONTRANSITIONAL and the IDN2_NFC_INPUT flags for compatibility with other software.

Parameters

input	zero terminated input Unicode (UCS-4) string.	
inlen	number of elements in `input` .	
output	pointer to newly allocated zero-terminated output string.	
flags	optional idn2_flags to modify behaviour.	

Returns

Returns IDN2_OK on success, or error code.

Since: 2.0.0

idn2_to_ascii_4z ()

```
int
idn2_to_ascii_4z (const uint32_t *input,
                  char **output,
                  int flags);
```

Convert UCS-4 domain name to ASCII string using the IDNA2008 rules. The domain name may contain several labels, separated by dots. The output buffer must be deallocated by the caller.

The default behavior of this function (when flags are zero) is to apply the IDNA2008 rules without the TR46 amendments. As the TR46 non-transitional processing is nowdays ubiquitous, when unsure, it is recommended to call this function with the IDN2_NONTRANSITIONAL and the IDN2_NFC_INPUT flags for compatibility with other software.

Parameters

input	zero terminated input Unicode (UCS-4) string.	
output	pointer to newly allocated zero-terminated output string.	
flags	optional idn2_flags to modify behaviour.	

Returns

Returns IDN2_OK on success, or error code.

Since: 2.0.0

idn2_to_ascii_8z ()

```
int
idn2_to_ascii_8z (const char *input,
                  char **output,
                  int flags);
```

Convert UTF-8 domain name to ASCII string using the IDNA2008 rules. The domain name may contain several labels, separated by dots. The output buffer must be deallocated by the caller.

The default behavior of this function (when flags are zero) is to apply the IDNA2008 rules without the TR46 amendments. As the TR46 non-transitional processing is nowdays ubiquitous, when unsure, it is recommended to call this function with the IDN2_NONTRANSITIONAL and the IDN2_NFC_INPUT flags for compatibility with other software.

Parameters

input	zero terminated input UTF-8 string.	
output	pointer to newly allocated output string.	
flags	optional idn2_flags to modify behaviour.	

Returns

Returns IDN2_OK on success, or error code.

Since: 2.0.0

idn2_to_ascii_lz ()

```
int
idn2_to_ascii_lz (const char *input,
                  char **output,
                  int flags);
```

Convert a domain name in locale's encoding to ASCII string using the IDNA2008 rules. The domain name may contain several labels, separated by dots. The output buffer must be deallocated by the caller.

The default behavior of this function (when flags are zero) is to apply the IDNA2008 rules without the TR46 amendments. As the TR46 non-transitional processing is nowdays ubiquitous, when unsure, it is recommended to call this function with the IDN2_NONTRANSITIONAL and the IDN2_NFC_INPUT flags for compatibility with other software.

Parameters

input	zero terminated input UTF-8 string.	
output	pointer to newly allocated output string.	
flags	optional idn2_flags to modify behaviour.	

Returns

IDN2_OK on success, or error code. Same as described in idn2_lookup_ul() documentation.

Since: 2.0.0

idn2_to_unicode_8z4z ()

```
int
idn2_to_unicode_8z4z (const char *input,
                      uint32_t **output,
                      int flags);
```

Converts a possibly ACE encoded domain name in UTF-8 format into a UTF-32 string (punycode decoding). The output buffer will be zero-terminated and must be deallocated by the caller.

`output` may be NULL to test lookup of `input` without allocating memory.

Parameters

input	Input zero-terminated UTF-8 string.	
output	Newly allocated UTF-32/UCS-4 output string.	
flags	optional idn2_flags to modify behaviour.	

Returns

IDN2_OK: The conversion was successful. IDN2_TOO_BIG_DOMAIN: The domain is too long. IDN2_TOO_BIG_LABEL: A label is would have been too long. IDN2_ENCODING_ERROR: Character conversion failed. IDN2_MALLOC: Memory allocation failed.

Since: 2.0.0

idn2_to_unicode_4z4z ()

```
int
idn2_to_unicode_4z4z (const uint32_t *input,
                      uint32_t **output,
                      int flags);
```

Converts a possibly ACE encoded domain name in UTF-32 format into a UTF-32 string (punycode decoding). The output buffer will be zero-terminated and must be deallocated by the caller.

`output` may be NULL to test lookup of `input` without allocating memory.

Parameters

input	Input zero-terminated UTF-32 string.	
output	Newly allocated UTF-32 output string.	
flags	optional idn2_flags to modify behaviour.	

Returns

IDN2_OK: The conversion was successful. IDN2_TOO_BIG_DOMAIN: The domain is too long. IDN2_TOO_BIG_LABEL: A label is would have been too long. IDN2_ENCODING_ERROR: Character conversion failed. IDN2_MALLOC: Memory allocation failed.

Since: 2.0.0

idn2_to_unicode_44i ()

```
int
idn2_to_unicode_44i (const uint32_t *in,
                     size_t inlen,
                     uint32_t *out,
                     size_t *outlen,
                     int flags);
```

The ToUnicode operation takes a sequence of UTF-32 code points that make up one domain label and returns a sequence of UTF-32 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.

`output` may be NULL to test lookup of `input` without allocating memory.

Parameters

in	Input array with UTF-32 code points.	
inlen	number of code points of input array	
out	output array with UTF-32 code points.	

outlen	on input, maximum size of output array with UTF-32 code points, on exit, actual size of output array with UTF-32 code points.	
flags	optional idn2_flags to modify behaviour.	

Returns

IDN2_OK: The conversion was successful. IDN2_TOO_BIG_DOMAIN: The domain is too long. IDN2_TOO_BIG_LABEL: A label is would have been too long. IDN2_ENCODING_ERROR: Character conversion failed. IDN2_MALLOC: Memory allocation failed.

Since: 2.0.0

idn2_to_unicode_8z8z ()

```
int
idn2_to_unicode_8z8z (const char *input,
                      char **output,
                      int flags);
```

Converts a possibly ACE encoded domain name in UTF-8 format into a UTF-8 string (punycode decoding). The output buffer will be zero-terminated and must be deallocated by the caller.

`output` may be NULL to test lookup of *`input`* without allocating memory.

Parameters

input	Input zero-terminated UTF-8 string.	
output	Newly allocated UTF-8 output string.	
flags	optional idn2_flags to modify behaviour.	

Returns

IDN2_OK: The conversion was successful. IDN2_TOO_BIG_DOMAIN: The domain is too long. IDN2_TOO_BIG_LABEL: A label is would have been too long. IDN2_ENCODING_ERROR: Character conversion failed. IDN2_MALLOC: Memory allocation failed.

Since: 2.0.0

idn2_to_unicode_8zlz ()

```
int
idn2_to_unicode_8zlz (const char *input,
                      char **output,
                      int flags);
```

Converts a possibly ACE encoded domain name in UTF-8 format into a string encoded in the current locale's character set (punycode decoding). The output buffer will be zero-terminated and must be deallocated by the caller.

`output` may be NULL to test lookup of *`input`* without allocating memory.

Parameters

input	Input zero-terminated UTF-8 string.	
output	Newly allocated output string in current locale's character set.	
flags	optional idn2_flags to modify behaviour.	

Returns

IDN2_OK: The conversion was successful. IDN2_TOO_BIG_DOMAIN: The domain is too long. IDN2_TOO_BIG_LABEL: A label is would have been too long. IDN2_ENCODING_ERROR: Character conversion failed. IDN2_MALLOC: Memory allocation failed.

Since: 2.0.0

idn2_to_unicode_lzlz ()

```
int
idn2_to_unicode_lzlz (const char *input,
                      char **output,
                      int flags);
```

Converts a possibly ACE encoded domain name in the locale's character set into a string encoded in the current locale's character set (punycode decoding). The output buffer will be zero-terminated and must be deallocated by the caller.

`output` may be NULL to test lookup of `input` without allocating memory.

Parameters

input	Input zero-terminated string encoded in the current locale's character set.	
output	Newly allocated output string in current locale's character set.	
flags	optional idn2_flags to modify behaviour.	

Returns

IDN2_OK: The conversion was successful. IDN2_TOO_BIG_DOMAIN: The domain is too long. IDN2_TOO_BIG_LABEL: A label is would have been too long. IDN2_ENCODING_ERROR: Output character conversion failed. IDN2_ICONV_FAIL: Input character conversion failed. IDN2_MALLOC: Memory allocation failed.

Since: 2.0.0

idn2_strerror ()

```
const char~*
idn2_strerror (int rc);
```

Convert internal libidn2 error code to a humanly readable string. The returned pointer must not be de-allocated by the caller.

Parameters

rc	return code from another libidn2 function.

Returns

A humanly readable string describing error.

idn2_strerror_name ()

```
const char~*
idn2_strerror_name (int rc);
```

Convert internal libidn2 error code to a string corresponding to internal header file symbols. For example, idn2_strerror_name(IDN2_MA will return the string "IDN2_MALLOC".

The caller must not attempt to de-allocate the returned string.

Parameters

rc	return code from another libidn2 function.

Returns

A string corresponding to error code symbol.

idn2_check_version ()

```
const char~*
idn2_check_version (const char *req_version);
```

Check IDN2 library version. This function can also be used to read out the version of the library code used. See IDN2_VERSION for a suitable `req_version` string, it corresponds to the idn2.h header file version. Normally these two version numbers match, but if you are using an application built against an older libidn2 with a newer libidn2 shared library they will be different.

Parameters

req_version	version string to compare with, or NULL.

Returns

Check that the version of the library is at minimum the one given as a string in `req_version` and return the actual version string of the library; return NULL if the condition is not met. If NULL is passed to this function no check is done and only the version string is returned.

idn2_free ()

```
void
idn2_free (void *ptr);
```

Call free(3) on the given pointer.

This function is typically only useful on systems where the library malloc heap is different from the library caller malloc heap, which happens on Windows when the library is a separate DLL.

Parameters

ptr	pointer to deallocate

idna_to_ascii_4i()

```
#define idna_to_ascii_4i(i,l,o,f)  idn2_to_ascii_4i(i,l,o,f|IDN2_NFC_INPUT| ↩
    IDN2_NONTRANSITIONAL)
```

idna_to_ascii_4z()

```
#define idna_to_ascii_4z(i,o,f)  idn2_to_ascii_4z(i,o,f|IDN2_NFC_INPUT| ↩
    IDN2_NONTRANSITIONAL)
```

idna_to_ascii_8z()

```
#define idna_to_ascii_8z(i,o,f)  idn2_to_ascii_8z(i,o,f|IDN2_NFC_INPUT| ↩
    IDN2_NONTRANSITIONAL)
```

idna_to_ascii_lz()

```
#define idna_to_ascii_lz(i,o,f)  idn2_to_ascii_lz(i,o,f|IDN2_NFC_INPUT| ↩
    IDN2_NONTRANSITIONAL)
```

Types and Values

G_GNUC_IDN2_ATTRIBUTE_PURE

```
# define G_GNUC_IDN2_ATTRIBUTE_PURE __attribute__ ((__pure__))
```

Function attribute: Function is a pure function.

G_GNUC_IDN2_ATTRIBUTE_CONST

```
# define G_GNUC_IDN2_ATTRIBUTE_CONST __attribute__ ((__const__))
```

Function attribute: Function is a const function.

G_GNUC_UNUSED

```
# define G_GNUC_UNUSED __attribute__ ((__unused__))
```

Parameter attribute: Parameter is not used.

IDN2_VERSION

```
#define IDN2_VERSION "2.0.5"
```

Pre-processor symbol with a string that describe the header file version number. Used together with idn2_check_version() to verify header file and run-time library consistency.

IDN2_VERSION_NUMBER

```
#define IDN2_VERSION_NUMBER 0x02000005
```

Pre-processor symbol with a hexadecimal value describing the header file version number. For example, when the header version is 1.2.4711 this symbol will have the value 0x01021267. The last four digits are used to enumerate development snapshots, but for all public releases they will be 0000.

IDN2_VERSION_MAJOR

```
#define IDN2_VERSION_MAJOR 2
```

Pre-processor symbol for the major version number (decimal). The version scheme is major.minor.patchlevel.

IDN2_VERSION_MINOR

```
#define IDN2_VERSION_MINOR 0
```

Pre-processor symbol for the minor version number (decimal). The version scheme is major.minor.patchlevel.

IDN2_VERSION_PATCH

```
#define IDN2_VERSION_PATCH 5
```

Pre-processor symbol for the patch level number (decimal). The version scheme is major.minor.patchlevel.

IDN2_LABEL_MAX_LENGTH

```
#define IDN2_LABEL_MAX_LENGTH 63
```

Constant specifying the maximum length of a DNS label to 63 characters, as specified in RFC 1034.

IDN2_DOMAIN_MAX_LENGTH

```
#define IDN2_DOMAIN_MAX_LENGTH 255
```

Constant specifying the maximum size of the wire encoding of a DNS domain to 255 characters, as specified in RFC 1034. Note that the usual printed representation of a domain name is limited to 253 characters if it does not end with a period, or 254 characters if it ends with a period.

enum idn2_flags

Flags to IDNA2008 functions, to be binary or:ed together. Specify only 0 if you want the default behaviour.

Members

IDN2_NFC_INPUT	Normalize input string using normalization form C.
IDN2_ALABEL_ROUNDTRIP	Perform optional IDNA2008 lookup roundtrip check (not implemented yet).
IDN2_TRANSITIONAL	Perform Unicode TR46 transitional processing.
IDN2_NONTRANSITIONAL	Perform Unicode TR46 non-transitional processing.
IDN2_ALLOW_UNASSIGNED	Libidn compatibility flag, unused.

IDN2_USE_STD3_ASCII_RULES	Use STD3 ASCII rules. This is a TR46 only flag, and will be ignored when set without either *IDN2_TRANSITIONAL* or *IDN2_NONTRANSITIONAL* .
IDN2_NO_TR46	Disable Unicode TR46 processing (default).

enum idn2_rc

Return codes for IDN2 functions. All return codes are negative except for the successful code IDN2_OK which are guaranteed to be

1. Positive values are reserved for non-error return codes.

Note that the idn2_rc enumeration may be extended at a later date to include new return codes.

Members

IDN2_OK	Successful return.
IDN2_MALLOC	Memory allocation error.

IDN2_NO_CODESET	Could not determine locale string encoding format.
IDN2_ICONV_FAIL	Could not transcode locale string to UTF-8.
IDN2_ENCODING_ERROR	Unicode data encoding error.
IDN2_NFC	Error normalizing string.
IDN2_PUNYCODE_BAD_INPUT	Punycode invalid input.
IDN2_PUNYCODE_BIG_OUTPUT	Punycode output buffer too small.
IDN2_PUNYCODE_OVERFLOW	Punycode conversion would overflow.

IDN2_TOO_BIG_DOMAIN	Domain name longer than 255 characters.
IDN2_TOO_BIG_LABEL	Domain label longer than 63 characters.
IDN2_INVALID_ALABEL	Input A-label is not valid.
IDN2_UALABEL_MISMATCH	Input A-label and U-label does not match.
IDN2_INVALID_FLAGS	Invalid combination of flags.
IDN2_NOT_NFC	String is not NFC.
IDN2_2HYPHEN	String has forbidden two hyphens.

IDN2_HYPHEN_STARTEND	String has forbidden starting/ending hyphen.
IDN2_LEADING_COMBINING	String has forbidden leading combining character.
IDN2_DISALLOWED	String has disallowed character.
IDN2_CONTEXTJ	String has forbidden context-j character.
IDN2_CONTEXTJ_NO_RULE	String has context-j character with no rull.

IDN2_CONTEXTO	String has forbidden context-o character.
IDN2_CONTEXTO_NO_RULE	String has context-o character with no rull.
IDN2_UNASSIGNED	String has forbidden unassigned character.
IDN2_BIDI	String has forbidden bi-directional properties.
IDN2_DOT_IN_LABEL	Label has forbidden dot (TR46).

IDN2_INVALID_TRANSITIONAL	Label has character forbidden in transitional mode (TR46).	
IDN2_INVALID_NONTRANSITIONAL	Label has character forbidden in non-transitional mode (TR46).	

enum Idna_rc

Members

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		

enum Idna_flags

Members

IDNA_ALLOW_UNASSIGNED		
IDNA_USE_STD3_ASCII_RULES		

idna_to_unicode_8z4z

```
#define idna_to_unicode_8z4z  idn2_to_unicode_8z4z
```

idna_to_unicode_4z4z

```
#define idna_to_unicode_4z4z  idn2_to_unicode_4z4z
```

idna_to_unicode_44i

```
#define idna_to_unicode_44i   idn2_to_unicode_44i
```

idna_to_unicode_8z8z

```
#define idna_to_unicode_8z8z  idn2_to_unicode_8z8z
```

idna_to_unicode_8zlz

```
#define idna_to_unicode_8zlz  idn2_to_unicode_8zlz
```

idna_to_unicode_lzlz

```
#define idna_to_unicode_lzlz  idn2_to_unicode_lzlz
```

idna_strerror

```
#define idna_strerror         idn2_strerror
```

idn_free

```
#define idn_free              idn2_free
```

Chapter 2

Index

G
G_GNUC_IDN2_ATTRIBUTE_CONST, 12
G_GNUC_IDN2_ATTRIBUTE_PURE, 12
G_GNUC_UNUSED, 12

I
idn2_check_version, 11
IDN2_DOMAIN_MAX_LENGTH, 13
idn2_flags, 13
idn2_free, 11
IDN2_LABEL_MAX_LENGTH, 13
idn2_lookup_u8, 2
idn2_lookup_ul, 3
idn2_rc, 15
idn2_register_u8, 3
idn2_register_ul, 4
idn2_strerror, 10
idn2_strerror_name, 11
idn2_to_ascii_4i, 5
idn2_to_ascii_4z, 5
idn2_to_ascii_8z, 6
idn2_to_ascii_lz, 6
idn2_to_unicode_44i, 8
idn2_to_unicode_4z4z, 8
idn2_to_unicode_8z4z, 7
idn2_to_unicode_8z8z, 9
idn2_to_unicode_8zlz, 9
idn2_to_unicode_lzlz, 10
IDN2_VERSION, 13
IDN2_VERSION_MAJOR, 13
IDN2_VERSION_MINOR, 13
IDN2_VERSION_NUMBER, 13
IDN2_VERSION_PATCH, 13
idn_free, 21
Idna_flags, 20
Idna_rc, 20
idna_strerror, 21
idna_to_ascii_4i, 12
idna_to_ascii_4z, 12
idna_to_ascii_8z, 12
idna_to_ascii_lz, 12
idna_to_unicode_44i, 21
idna_to_unicode_4z4z, 21
idna_to_unicode_8z4z, 20
idna_to_unicode_8z8z, 21
idna_to_unicode_8zlz, 21
idna_to_unicode_lzlz, 21