# Libidn2 Reference Manual

| COLLABORATORS | | | |
|---|---|---|---|
| | *TITLE* :<br><br>Libidn2 Reference Manual | | |
| *ACTION* | *NAME* | *DATE* | *SIGNATURE* |
| WRITTEN BY | | November 14, 2019 | |

| REVISION HISTORY | | | |
|---|---|---|---|
| NUMBER | DATE | DESCRIPTION | NAME |
| | | | |

# Contents

# Chapter 1

# Libidn2 Overview

Libidn2 is a free software implementation of IDNA2008 and TR46.

## 1.1 idn2

idn2 —

### Functions

| | |
|---|---|
| #define | GCC_VERSION_AT_LEAST() |
| int | idn2_lookup_u8 () |
| int | idn2_register_u8 () |
| int | idn2_lookup_ul () |
| int | idn2_register_ul () |
| int | idn2_to_ascii_4i2 () |
| 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 () |
| 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_DEPRECATED |
| #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

### GCC_VERSION_AT_LEAST()

```
# define GCC_VERSION_AT_LEAST(major, minor) ((__GNUC__ > (major)) || (__GNUC__ == (major)  ↩
    && __GNUC_MINOR__ >= (minor)))
```

Pre-processor symbol to check the gcc version.

#### Parameters

| | | |
|---|---|---|
| major | gcc major version number to compare with | |
| minor | gcc minor version number to compare with | |

### 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. This is default since version 2.2. To switch this behavior off, pass IDN2_NO_ALABEL_ROUNDTRIP

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. This is default since version 2.2. To switch this behavior off, pass IDN2_NO_ALABEL_ROUNDTRIP

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_ROUNDTRIP | IDN2_NONTRANSITION

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_4i2 ()

```
int
idn2_to_ascii_4i2 (const uint32_t *input,
                   size_t inlen,
                   char **output,
                   int flags);
```

**Warning**

`idn2_to_ascii_4i2` is deprecated and should not be used in newly-written code.

### 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 nowadays 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 nowadays 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 nowadays 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 | Currently unused. | |

#### 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 | Currently unused. | |

#### 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 | Currently unused. | |

#### 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 | Currently unused. | |

#### 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 | Currently unused. | |

#### 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 | Currently unused. | |

#### 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_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_DEPRECATED

```
# define G_GNUC_DEPRECATED __attribute__((deprecated))
```

Function attribute: Function is deprecated.

#### G_GNUC_UNUSED

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

Parameter attribute: Parameter is not used.

#### IDN2_VERSION

```
#define IDN2_VERSION "2.3.0"
```

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 0x02030000
```

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 3
```

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

### IDN2_VERSION_PATCH

```
#define IDN2_VERSION_PATCH 0
```

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 (default). |
| IDN2_TRANSITIONAL | Perform Unicode TR46 transitional processing. |
| IDN2_NONTRANSITIONAL | Perform Unicode TR46 non-transitional processing (default). |
| 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. |
| IDN2_NO_ALABEL_ROUNDTRIP | Disable ALabel lookup roundtrip check. |

### 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 contextj 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 bidirectional 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). |
| IDN2_ALABEL_ROUNDTRIP_FAILED | ALabel -> Ulabel -> ALabel result differs from input. |

### enum Idna_rc

Return codes for transition to / compatibility with libidn2.

Please be aware that return codes from idna_ functions might be unexpected when linked / built with libidn2.

#### Members

| | |
|---|---|
| IDNA_SUCCESS | Same as IDN2_OK |

| | |
|---|---|
| IDNA_STRINGPREP_ERROR | Same as IDN2_ENCODING_ERROR |
| IDNA_PUNYCODE_ERROR | Same as IDN2_PUNYCODE_BAD_INPUT |
| IDNA_CONTAINS_NON_LDH | Same as IDN2_ENCODING_ERROR |
| IDNA_CONTAINS_LDH | Same as IDNA_CONTAINS_NON_LDH |
| IDNA_CONTAINS_MINUS | Same as IDN2_ENCODING_ERROR |
| IDNA_INVALID_LENGTH | Same as IDN2_DISALLOWED |
| IDNA_NO_ACE_PREFIX | Same as IDN2_ENCODING_ERROR |
| IDNA_ROUNDTRIP_VERIFY_ERROR | Same as IDN2_ENCODING_ERROR |
| IDNA_CONTAINS_ACE_PREFIX | Same as IDN2_ENCODING_ERROR |
| IDNA_ICONV_ERROR | Same as IDN2_ENCODING_ERROR |
| IDNA_MALLOC_ERROR | Same as IDN2_MALLOC |
| IDNA_DLOPEN_ERROR | Same as IDN2_MALLOC |

### enum Idna_flags

Flags for transition to / compatibility with libidn2.

#### Members

| | |
|---|---|
| IDNA_ALLOW_UNASSIGNED | Same as IDN2_ALLOW_UNASSIGNED |
| IDNA_USE_STD3_ASCII_RULES | Same as IDN2_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