Libidn2 Reference Manual

COLLABORATORS			
	TITLE : Libidn2 Reference Manual		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		September 28, 2011	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Libidn2 Overview

Libidn2 is a free software implementation of IDNA2008.

1.1 idn2

idn2 —

Synopsis

```
#define             IDN2_VERSION
#define             IDN2_VERSION_NUMBER
#define             IDN2_LABEL_MAX_LENGTH
#define             IDN2_DOMAIN_MAX_LENGTH
enum                idn2_flags;
int                 idn2_lookup_u8                  (const uint8_t *src,
                                                     uint8_t **lookupname,
                                                     int flags);
int                 idn2_register_u8                (const uint8_t *ulabel,
                                                     const uint8_t *alabel,
                                                     uint8_t **insertname,
                                                     int flags);
int                 idn2_lookup_ul                  (const char *src,
                                                     char **lookupname,
                                                     int flags);
int                 idn2_register_ul                (const char *ulabel,
                                                     const char *alabel,
                                                     char **insertname,
                                                     int flags);
enum                idn2_rc;
const char *        idn2_strerror                   (int rc);
const char *        idn2_strerror_name              (int rc);
const char *        idn2_check_version              (const char *req_version);
void                idn2_free                       (void *ptr);
```

Description

Details

IDN2_VERSION

```
#define IDN2_VERSION "0.8"
```

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 0x00080000
```

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_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

```
typedef enum
{
  IDN2_NFC_INPUT = 1,
  IDN2_ALABEL_ROUNDTRIP = 2,
} idn2_flags;
```

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

IDN2_NFC_INPUT Normalize input string using normalization form C.

IDN2_ALABEL_ROUNDTRIP Perform optional IDNA2008 lookup roundtrip check.

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. Pass IDN2_ALABEL_ROUNDTRIP in *flags* to convert any input A-labels to U-labels and perform additional testing. Multiple flags may be specified by binary or:ing them together, for example IDN2_NFC_INPUT | IDN2_ALABEL_ROUNDTRIP.

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_ or IDN2_TOO_BIG_LABEL is returned, or another error code is returned.

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.

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_MISMATCH 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.

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.

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.

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_MISMATCH 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.

enum idn2_rc

```
typedef enum
{
  IDN2_OK = 0,
  IDN2_MALLOC = -100,
  IDN2_NO_CODESET = -101,
  IDN2_ICONV_FAIL = -102,
  IDN2_ENCODING_ERROR = -200,
  IDN2_NFC = -201,
  IDN2_PUNYCODE_BAD_INPUT = -202,
  IDN2_PUNYCODE_BIG_OUTPUT = -203,
  IDN2_PUNYCODE_OVERFLOW = -204,
  IDN2_TOO_BIG_DOMAIN = -205,
  IDN2_TOO_BIG_LABEL = -206,
  IDN2_INVALID_ALABEL = -207,
  IDN2_UALABEL_MISMATCH = -208,
  IDN2_NOT_NFC = -300,
  IDN2_2HYPHEN = -301,
  IDN2_HYPHEN_STARTEND = -302,
  IDN2_LEADING_COMBINING = -303,
  IDN2_DISALLOWED = -304,
  IDN2_CONTEXTJ = -305,
```

```
    IDN2_CONTEXTJ_NO_RULE = -306,
    IDN2_CONTEXTO = -307,
    IDN2_CONTEXTO_NO_RULE = -308,
    IDN2_UNASSIGNED = -309,
    IDN2_BIDI = -310
  } idn2_rc;
```

Return codes for IDN2 functions. All return codes are negative except for the successful code IDN2_OK which are guaranteed to be 0. 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.

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_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_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.

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.

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.

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.

ptr **:** pointer to deallocate

Chapter 2

Index