ztd.cuneicode

Release 0.0.0

ThePhD & Shepherd's Oasis, LLC

Nov 07, 2022

CONTENTS:

The premiere library for handling text in different encoding forms for C software.

CHAPTER

ONE

WHO IS THIS LIBRARY FOR?

If:

- you want to convert from one Unicode encoding to another Unicode encoding;
- you want to prevent data in the wrong encoding from infiltrating your application and causing Mojibake;
- you want to have a flexible, growable registry of encodings that does not require compile-time modification of source code to expand;
- you want a design that is better than the C Standard Library's and actually handles your encoding;
- you want safe defaults for working with text;

then `ztd.cuneicode` is for you!

1.1 Getting Started (In Progress)

> **Warning:** This isn't finished yet! Come check back by the next major or minor version update.

1.2 Quick 'n' Dirty Tutorial (In Progress)

> **Warning:** This isn't finished yet! Come check back by the next major or minor version update.

cuneicode is a C library whose headers work in both C and C++. Its implementation is currently done in C++. To use it, use:

- one of the many CMake methods (*add_subdirectory*, *FetchContent*, or similar)
- directly add and build all the sources to your project

> **Warning:** Adding sources directly to your project is not guaranteed to work in future major revisions, as certain build steps might generate code in the future.

Once the library is appropriately included, you can start using cuneicode.

1.2.1 Simple Conversions

To convert from UTF-16 to UTF-8, use the appropriately *c8* and *c16*-marked free functions in the library:

```

#include <ztd/cuneicode.h>

#include <ztd/idk/size.h>

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

int main() {

        const ztd_char16_t utf16_text[] = u"";
        ztd_char8_t utf8_text[9]        = { 0 };

        // Now, actually output it
        const ztd_char16_t* p_input = utf16_text;
        ztd_char8_t* p_output       = utf8_text;
        // ztd_c_array_size INCLUDES the null terminator in the size!
        size_t input_size   = ztd_c_array_size(utf16_text);
        size_t output_size  = ztd_c_array_size(utf8_text);
        cnc_mcstate_t state = { 0 };
        // call the function with the right parameters!
        cnc_mcerror err = cnc_c16snrtoc8sn( // formatting
             &output_size, &p_output,       // output first
             &input_size, &p_input,         // input second
             &state);                       // state parameter
        if (err != CNC_MCERROR_OK) {
                const char* err_str = cnc_mcerror_to_str(err);
                printf(
                     "An (unexpected) error occurred and the conversion could not "
                     "happen! Error string: %s\n",
                     err_str);
                return 1;
        }

        // requires a capable terminal / output, but will be
        // UTF-8 text!
        printf("Converted UTF-8 text: %s\n", (const char*)utf8_text);

        return 0;
}
```

We use raw `printf` to print the UTF-8 text. It may not appear correctly on a terminal whose encoding which is not UTF-8, which may be the case for older Microsoft terminals, some Linux kernel configurations, and deliberately misconfigured Mac OSX terminals. There are also some other properties that can be gained from the use of the function:

- the amount of data read (using `initial_input_size - input_size`);
- the amount of data written out (using `initial_output_size - output_size`);
- a pointer to any extra input after the operation (`p_input`);

- and, a pointer to any extra output that was not written to after the operation (`p_output`).

One can convert from other forms of UTF-8/16/32 encodings, and from the *wide execution encodings*/*execution encoding* (encodings used by default for `const char[]` and `const wchar_t[]` strings) using the various different *prefixed-based* functions.

1.3 Design Goals and Philosophy

The goal of this library are to:

- enable people to write new code that can properly handle encoded information, specifically text;
- give them effective means to convert that information in various ways for their own encodings;
- do so without using hidden allocations or other conversions that are not controllable;
- and, allow them to not need to provide pairwise conversions for every encoding pair they care about.

To this end, there are 2 sets of functionality: typed and static conversions which utilize proper input and output buffer types, and untyped conversions which go through a level of indirection and explicitly work on byte-based (`unsigned char`) buffers. Each of the 2 sets of functionality are further subdivided into 2 use cases that users are about:

- Singular Conversions: where a user wants to encode, decode, or transcode one complete unit of information at a time and receive an error when that conversion fails.
- Multiple (bulk) Conversions: where a user wants to encode, decode, or transcode the maximum amount of information possible in a single given call, for speed or throughput reasons.

Finally, to make sure this library is capable of scaling and does not have users hobbling together pairwise function calls for each encoding pair they care about, we provide **automatic** translation through an indirect layer that leverages one of the popular omni-encodings (e.g. Unicode) so that users do not have to provide conversion routines to every possible other encoding: just the ones they care about.

1.3.1 Indivisible Unit of Work

When doing transcoding, in order to properly develop an algorithm that scales across all encodings and similar, one needs to be able to define certain things about the input, how it is consumed, how any related state is managed, and what outputs - if any - are created. The central way to describe this is with the concept of an *indivisible Unit of work*.

An *indivisible unit* is the smallest possible input, as defined by the input encoding, that:

- can produce one or more outputs;
- and/or, perform a transformation of any internal state.

The conversion of these indivisible units is called an *indivisible unit of work*, and they are used to complete all encoding operations. One or more of the following options must hold if an indivisible unit of work is attempted:

- enough input is consumed to perform an output or change the internal state;
- enough input is consumed to output is written from consuming input, or from the internal state which causes the internal state to change;
- or, an error occurs and both the input and output do not change relative to the last completed indivisible unit of work, if any.

If the third condition happens, then neither the first or the second condition may happen. The state - managed through the mbstate_t pointer - may or may not change during any of these operations, and may be left in an indeterminate state after an error occurs. For the multi unit functions, the process acts as if it completes one indivisible unit of work

repeatedly. When an error occurs, only the input successfully consumed and the output successfully written according to the last indivisible unit of work are reflected in the output values: no other values are written.

1.3.2 Function Design and Shape

It is surprisingly hard to provide the right kind of function to C and C++ that covers all use cases for text. The kinds of things people do with text are varied, but the core of the actions that conversion should cover are:

- **Convert as much as as possible, or simply fail (where a conversion either happens or it does not, and if it does not it fails).**
  - Most text converion APIs behave in this way, especially when they are responsible for allocating the memory.
  - Converting text "in bulk" is a heavily researched area and text validation "in under 1 instruction per byte" is a thing that has had time heavily invested into it. There are large archives of government data and similar not stored in Unicode, for one reason or another: converting potentially terabytes or petabytes of data may be required.
- **Convert, skip over bad input, then resume converting, alternating between skipping bad text and converting text until the**
  - Input sanitizers for non-critical input text may behave this way, although it is noted that this loses information (e.g., some kind of understanding that bad text happened by using explicit .)
- **Convert, substitute bad input in the output, then resume converting, alternating between subsitutions and conversions unt**
  - This is a variant of the above style of usage and is used in most places, especially user-facing places, that attempt to handle text failures including Web Browsers such as Apple Safari, Google Chrome, Microsoft Edge, Mozilla Firefox, and more.
- **Convert some indivisible unit of text, one at a time, to a known text encoding and feed it to a specific rendering engine.**
  - Useful for the guts of a rendering engine that converts either a single code point or a small piece of text into a small buffer and uses it to layout their text.
  - Useful to delineate each code point boundary without having to convert the full text in-memory.
  - Useful for a well-specified form of round-tripping.
- **Convert between text encodings that may or may not have a direct code path between them.**
  - For example, it is not feasible for someone to need to write a conversion directly from EUC-KR to UTF-8, or TASCII to UTF-16. It should be easy to go from TASCII or EUC-JP or Latin-1 to any other encoding, where reasonable, without needing to write a new encoding path. There are well over 100 encodings in the world: writing by-hand, pairwise conversions between all of them is an infeasible task.

The API must have a general shape and usage to it that enables all of these use cases without causing deep undue burden to software engineers utilizing the library. It also needs to combat some serious issues with preexisting C APIs of the day.

- The C Standard Library only takes one *code unit* at a time. This means they deeply restrict themselves to only a very limited set of encodings.
- libiconv, as defined by POSIX, allows too wide a variety of implementation techniques. Insertion of Byte Order Marks, even when not asked for, is common for certain Unicode types and there exists no agreement between implementations whether to treat data as Big Endian or Little Endian. Furthermore, insertion of invalid characters

without the request or approval of the software engineer (? on some GNU derivatives, * on musl libc) is horrible for cross-platform expansion.

- Platform encoding functions often do not provide adequate error information (Microsoft Windows's `MultibyteToWideChar` function does not report how many characters it successfully converted when it returns with an error, leaving the end-user to discard the all output and input if they do not have intimate knowledge of the input data already).

It is a lot of issues we have to fix in one API. We need APIs that:

- allows for custom error handling (to cover the replacement use case and the "skip over bad input" use case);
- allows for fast, bulk conversion (to cover the "convert terabytes of text as fast as possible" archival use case);
- and, allows for a way to convert disparate encodings that may not have a hand-crafted encoding path (to prevent needing to write encodings for *100^2* one-way encoding functions).

The good news is that, while libiconv's specification under POSIX is terrible and too permissive of a wide variety of implementation strategies and failures

1.3.3 Naming Design and Mapping

The names of the functions are written in a compressed C-style, which makes them not the most friendly in terms of readability. But, thankfully, all the function names for the strongly-typed single and bulk conversion functions follow the same convention, composed of a number of parts.

The mapping of the prefix/suffix and the name is listed below:

Table 1: Relationship Table For Prefix/Suffix

PreSuffix Name	Type	Encoding Used	Max Output
`mc`	`char`	*execution encoding*	`CNC_MC_MAX`
`mwc`	`wchar_t`	*wide execution encoding*	`CNC_MWC_MAX`
`c8`	`ztd_char8_t`	*UTF-8*	`CNC_C8_MAX`
`c16`	`ztd_char16_t`	*UTF-16*	`CNC_C16_MAX`
`c32`	`ztd_char32_t`	*UTF-32*	`CNC_C32_MAX`
`cx` or `xy`	`input/output-deduced`	`input/output-deduced`	`output-dependent`

Those parts are as follows:

- `{prefix}` - the prefix identifying what character set the function will be converting from.
- `s` - if present, this denotes that this is a *bulk conversion*; otherwise, it is a *single conversion*.
- `n` - if present, this is a function which will optionally take a count value to denote how much space, in number of **elements** (not bytes), is present in the source (input) data.
- `r` - if present, this denotes that this is a "restartable" function; i.e., that this function takes a state parameter and operates on no invisible state; otherwise, it is "non-restartable" and creates an automatic storage duration state object internally that will be discarded after the function completes.
- `to` - present in all names, simply signifies the start of the next portion of the function name and is help as the English "to", as in "A to B" or "Beef to Buns".
- `{suffix}` - the suffix identifying what character set the function will be converting to.
- `s` - similar to above, if present, this denotes that the name is a *bulk conversion*; otherwise, it is a *single conversion*.
- `n` - if present, this is a function which will optionally take a count value to denote how much space, in number of **elements** (not bytes), is present in the destination (output) data.

For example, a function which is named `c8ntomcn` effectively reads "UTF-8, with count, converted to Locale-Based Execution Encoding, with count, non-restartable".

The *maximum output macros* are part of a series of macros used with a function with the appropriately associated suffix / output type. These allow a caller to always have a suitably-sized output buffer for a single complete output operation.

1.3.4 Singular Conversion

Singular conversions are foundational because they are guaranteed to only encode a single complete unit of information and output a single complete unit of information (or change the operating state in some manner). They guarantee forward progress in that every singular conversion must consume some portion of the input, even if it generates no actual output data and simply mutates the state of an in-progress encoding.

Singular conversions also have a maximally-bounded input size that is guaranteed to either change the state or compute one complete unit of output (which may result in multiple bytes/code units/code points being written out). This means that one can, for each individual operation, provide enough space to guarantee that writing out a single value works just fine. This is necessary for low-memory environments which may need to process input in chunks or with the smallest possible memory guarantee available.

Looping over a set of input using a singular conversion is a valid way to transform a singular conversion into a bulk conversion. This technique is handy for guaranteeing correctness from composing a bulk operation from a correctly-implemented single conversion, but hinders throughput and speed due to needing to check for safety and constantly recalculate and update sizes. This does not necessarily mean that singular conversion must always be done: there are techniques that can convert multiple (bulk) conversion algorithms to become single conversion algorithms, though the performance of such an algorithm is noticeably worse.

1.3.5 Multiple (Bulk) Conversion

Bulk conversions are a way to convert multiple complete units of input into multiple complete units of output (bytes/code units/code points written and/or state changes). Bulk conversions greedily consume input and only stop on exhaustion or error. Unlike *single conversions*, there is no theoretical or logical upper bound for the output buffer that can be given for an encoding transformation without knowing intimate details about the encoding, especially if the input or output encoding is a variable-width (consumes or outputs 1 or more code units depending on what kind of input goes in). Therefore, it is much more probable and likely that an error will return `CNC_MCERROR_INSUFFICIENT_OUTPUT`, which must be handled accordingly in order to fully process the given input.

A bulk conversion can be used to simulate a single conversion by arbitrarily limiting the size of an incoming input to 1 code unit, and seeing if it successfully transcodes. If it reports that the input is incomplete, keep incrementing the size of the incoming input 1 more code unit, until it either eventually reports success (`CNC_MCERROR_OK`) or failure

1.3.6 Conversion Registries

One of the many problems with ICU, libiconv, and so many other libraries is their deep inflexibility. In particular, for *iconv*, support for encodings must be enabled and then built, which means that if an Operating System's default dstribution does not provide for the conversion, it does not exist for that particular end-user. It also makes it frustrating when other applications base their changes off of whatever conversions the platform-blessed C Standard Library give, or what the platform-blessed package respositories hand out by default. Code that works on OpenBSD may fail spectacularly on Ubuntu, IBM platforms may have many many exotic encodings that do not exist on your Apple, and so on and so forth. This becomes problematic for developers who want to provide a more standard experience for their encodings, often leading them to either ship their own libiconv or engage in the platform encoding free-for-all.

This is where cuneicode's `cnc_conversion_registry` comes in. The conversion registry is a structure that stores all of the information necessary to provide conversions from one encoding to another encoding in a type-agonstic way. Furthermore, it also provides a way to add conversions both (*both* single and bulk) to a registry. This means that a

single registry can be infinitely extensible at runtime rather than requiring recompilation. Developers can provide their own encodings by programimng it in, or adding conversion routines based on things that hook into the registry, and any other techniques that a software engineer can come up with.

Thusly, it becomes far easier and far more portable to guarantee a set of encodings is always available to the your end-users, rather than gambling on whatever platform support or whatever offering your current POSIX or C Standard Library has at the moment. Below, you can select a more specific topic for how this works, and how the registry enables users to have the same kind of powerful conversion routines and extensibility between two options.

Registry Conversions

Registry conversions are a form of converting that are type-erased and directed through an object of type *cnc_conversion_registry*. The function used for this is the omni-function, *cnc_conv()*.

Indirect Conversions

When a registry *conversion rountine* provides an encoding path to a common encoding, but not to each other, it can be difficult to get data in one shape to another. For example, if you only register a conversion routine from SHIFT-JIS to UTF-8, and then from UTF-8 to UTF-32, you have provided no direct path between SHIFT-JIS and UTF-32. But, what if it was possible for the the library to realize that UTF-8 is a possible substract between the two libraries? What if it could automatically detect certain "Universal Encodings", like the Unicode Encodings, and use those as a bridge between 2 disparate encodings?

This is a technique that has been in use for glibc, musl libc, ICU, libiconv, and many encoders for over a decade now. They provide a conversion to a common substrate — generally, Unicode in the form of UTF-8, UTF-16, or UTF-32 (mostly this last one) — and then use it to convert for well over a decade now.

how the *conversion registry* will bridge the two encodings together without a developer needing to specifically write the encodings through an encoding pair. This is done by utilizing UTF-32 as a go-between for the two functions. This is a technique that is common among text transcoding engines, albeit the process of doing so for other libraries is generally explicit, involved, and sometimes painful.

When *opening a new conversion routine*, use the `is_indirect` member and related information on the *cnc_conversion_info* structure to find out if the conversion has been opened through an intermediate. Note that this is the only time the routines will tell you this: this information may not be accessible later.

Indirect Liaisons

Indirect encoding paths will not link together arbitrarily long encoding conversion steps to get from one encoding to another: it does not attempt to create a connectivity graph between all encodings (though, wouldn't that be a fun project?). Remember that each intermediate encoding that the data must travel through imposes overhead! So, only one encoding is allowed to be the go-between for encodings.

Unfortunately, not all encodings are recognized as liaison encodings. For example, writing an encoding conversion from `"UTF-8"` to `SHIFT-JIS` and then from `SHIFT-JIS` to `EUC-JP` is not an indirect path the library will string together. Right now, is simply will check if you encoding to a Unicode Encoding like `UTF-32`, and then see if there is a conversion from that Unicode Encoding to your desired destination. The full list of Indirect Liaison encodings (in order of preference) is:

1. UTF-32
2. UTF-32 Unchecked
3. UTF-16
4. UTF-16 Unchecked

5. UTF-8
6. UTF-8 Unchecked

Registry Allocation

At many times, the registry may need to access additional information to, for example, store extra information. This could be done with `malloc` and handled on cleanup with `free`, but that can be prohibitive to C implementations which may not want to draw their memory from either of these global pools. Therefore, in order to allow a user to customize the way allocation works, there are 2 ways to customize how memory allocation is done with the library.

The first, high-level way to control all allocation is to use the *cnc_conversion_heap*. That structure provides 5 functions which will control all non-automatic storage duration space created by the registry.

1.4 API Reference

This is simply a listing of all the available pages containing various APIs, or links to pages that link to API documentation.

1.4.1 General Structures, Enumerations, and Constants

cnc_mcerror

const char ***cnc_mcerror_to_str**(cnc_mcerror err)
Returns a string representing the error code's name.

enumerator **CNC_MCERROR_OK**
Returned when processing of the input and writing of the output is successful and nothing has gone wrong.

enumerator **CNC_MCERROR_INCOMPLETE_INPUT**
Returned when there is not yet an error in processing the input, but the input is exhausted before a proper single unit of work is complete.

Remark

It is fundamentally important that this is returned from conversion routines when the input is fully exhausted AND the input sequence of bytes/code units/code points is not erroneous. If the input values are erroneous, *CNC_MCERROR_INVALID_SEQUENCE* should be used instead.

---

enumerator **CNC_MCERROR_INVALID_SEQUENCE**
Returned when there is an error processing any input. No output is written.

Remark

This can only be applied when processing the input finds an invalid sequence or an improper input value. It shall not be used for exhausted input or empty input. Critically, no output from **that specific unit of work** is written (but any previously written successful output remains with e.g. any bulk conversion function).

enumerator **CNC_MCERROR_INSUFFICIENT_OUTPUT**
Returned when the size of the output is not sufficiently large to handle the value of the error.

Remark

For single functions (functions that complete only a single unit of work), *CNC_MCERROR_INSUFFICIENT_OUTPUT* can be entirely avoided by providing a large enough statically sized buffer. This is typically done by using the maximum-output macros such as *CNC_C32_MAX* - see the documentation for more details.

cnc_open_error

enum **cnc_open_error**
The error that occurred when trying to open or create a conversion resource.

Values:

enumerator **CNC_OPEN_ERROR_OK**
Returned when everything was okay.

enumerator **CNC_OPEN_ERROR_NO_CONVERSION_PATH**
Returned when there is no conversion path between the specified from and to encodings.

enumerator **CNC_OPEN_ERROR_INSUFFICIENT_OUTPUT**
Returned when there iss not enough output space to write into for creating the resource.

enumerator **CNC_OPEN_ERROR_INVALID_PARAMETER**
Returned when there is an invalid parameter passed in for creating the resource.

enumerator **CNC_OPEN_ERROR_ALLOCATION_FAILURE**
Returned when a heap-related or allocation-related failure occurred.

cnc_mcstate_t

The state object is used during conversions to provide a place for the function to write any temporary data into. This is useful for encodings such as IBM or Microsoft's rendition of SHIFT-JIS, where specific shift sequences are used to provide additional sequences or information for a given input or output string.

Note: For the `c8`, `c16`, and `c32` prefixed/suffixed functions, it may **not** use the state objects to store "partial writes" or "partial reads" of the data. Any encoding defined as UTF-8, UTF-16, and UTF-32 used through the `mc` (*execution encoding*-related) or `mwc` (*wide execution encoding*-related) shall also not be used to store partial pieces of the input or partial pieces of the output in order to accumulate information before reading in more data or writing out. If there is insufficient space to do a write to the output, *CNC_MCERROR_INSUFFICIENT_OUTPUT* must be returned. Similarly, if there is insufficient data and the data is at the very end, then *CNC_MCERROR_INCOMPLETE_INPUT* must be returned.

An implementation may define encodings which are not UTF-8, UTF-16, or UTF-32 that **does** perform partial writes, such as a `"UTF-8-partial"` or `"UTF-32-partial"`. But it shall not have the same `LC_TYPE` identifier as the UTF-8, UTF-16, or UTF-32 encodings.

union **cnc_mcstate_t**
#include <mcstate.h> The state for the typed conversion functions.

Remark

This is a complete object, but none of its members should be accessed or relied upon in any way, shape or form. If you do so, it is Undefined Behavior.

Public Members

struct *cnc_mcstate_t*::*cnc_header_t* **header**

struct *cnc_mcstate_t*::*__raw_t* **raw**

struct **__raw_t**
#include <mcstate.h> The raw type for user use.

Public Members

cnc_mcstate_indicator **indicator**
The indactor. Must be set by any custom encoding routine using mcstate_t and desiring custom completion behavior to CNC_MCSTATE_INDICATOR_RAW.

unsigned int **assume_valid**
Universal “assume valid input” flag for use with “-unchecked”-suffixed encodings.

unsigned int **__paddding**
Padding to keep consistent sizing. Not meant to be part of any location. Do not access.

state_is_complete_function ***completion_function**
The completion function. If behavior beyond a check for the provided fixed-size data is zero is desired, then this must be set to a valid function pointer. Otherwise, it must be a null pointer.

unsigned char **raw_data**[(sizeof(void*) * 3)]
Leftover data blob for use by the user. The user is responsible for its management within the conversion functions.

struct **cnc_header_t**
#include <mcstate.h> Shared data as part of every structure within a *cnc_mcstate_t*.

Public Members

cnc_mcstate_indicator **indicator**
: The indactor. Must be set by any custom encoding routine using mcstate_t and desiring custom completion behavior to CNC_MCSTATE_INDICATOR_RAW.

unsigned int **__assume_valid**
: Universal "assume valid input" flag for use with "-unchecked"-suffixed encodings. Do not access.

unsigned int **__paddding**
: Padding to keep consistent sizing. Not meant to be part of any location. Do not access.

bool **cnc_mcstate_is_complete**(const *cnc_mcstate_t* *__state)
: Returns whether or not the given *cnc_mcstate_t* has no more data that needs to be output.

  Parameters **__state** – **[in]** The state object to check is complete/finished and has no pending writes to do or data to gather.

void **cnc_mcstate_set_assume_valid**(*cnc_mcstate_t* *__state, bool __check_validity)
: Sets internal state for the *cnc_mcstate_t* object which will set its assume valid bits, if applicable.

  Parameters

  - **__state** – **[inout]** The state to turn validity on for.
  - **__check_validity** – **[inout]** Whether or not to check for validity.

bool **cnc_mcstate_get_assume_valid**(const *cnc_mcstate_t* *__state)
: Gets the internal state for the *cnc_mcstate_t* object representing its current "assume valid" state.

  Parameters **__state** – **[inout]** The state to turn validity on for.

Constants

This is a list of constants that can be used from the headers to gain meaningful information for specific use cases.

Max Constants

The constants present here represent the maximum output that the **non-bulk**, **single conversion** functions can output in **number of elements** (**NOT** the number of bytes!).

CNC_MC_MAX
: The maximum size that can be output by a single `cnc_cxnrtomcn` function call.

CNC_MWC_MAX
: The maximum size that can be output by a single `cnc_cxnrtomwcn` function call.

CNC_C8_MAX
: The maximum size that can be output by a single `cnc_cxnrtoc8n` function call.

CNC_C16_MAX
: The maximum size that can be output by a single `cnc_cxnrtoc16n` function call.

CNC_C32_MAX
The maximum size that can be output by a single `cnc_cxnrtoc32n` function call.

1.4.2 Typed Conversion

Encodings

This is the list of encodings which have named functions available in cuneicode.

ASCII

The 7-bit American Standard Code for Information Interchange (ASCII) encoding.

cnc_mcerror **cnc_mcnrtoc32n_ascii**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const char **__p_src, *cnc_mcstate_t* *__p_state)

See also:

cnc_mcnrtoc32n

cnc_mcerror **cnc_c32nrtomcn_ascii**(size_t *__p_maybe_dst_len, char **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

See also:

cnc_c32nrtomcn

cnc_mcerror **cnc_mcsnrtoc32sn_ascii**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const char **__p_src, *cnc_mcstate_t* *__p_state)

See also:

cnc_mcsnrtoc32sn

cnc_mcerror **cnc_c32snrtomcsn_ascii**(size_t *__p_maybe_dst_len, char **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

See also:

cnc_c32snrtomcsn

Big5 (HKSCS)

The Big5 encoding using the Hong Kong Supplementary Character Set (HKSCS). Matches the encoding from the WHATWG Encoding standard.

cnc_mcerror **cnc_mcnrtoc32n_big5_hkscs**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const char **__p_src, *cnc_mcstate_t* *__p_state)

See also:

cnc_mcnrtoc32n

cnc_mcerror **cnc_c32nrtomcn_big5_hkscs**(size_t *__p_maybe_dst_len, char **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

See also:

cnc_c32nrtomcn

cnc_mcerror **cnc_mcsnrtoc32sn_big5_hkscs**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const char **__p_src, *cnc_mcstate_t* *__p_state)

See also:

cnc_mcsnrtoc32sn

cnc_mcerror **cnc_c32snrtomcsn_big5_hkscs**(size_t *__p_maybe_dst_len, char **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

See also:

cnc_c32snrtomcsn

GB18030

The GB18030 Unicode encoding. Matches the encoding from the WHATWG Encoding standard.

cnc_mcerror **cnc_mcnrtoc32n_gb18030**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const char **__p_src, *cnc_mcstate_t* *__p_state)

See also:

cnc_mcnrtoc32n

cnc_mcerror **cnc_c32nrtomcn_gb18030**(size_t *__p_maybe_dst_len, char **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

See also:

cnc_c32nrtomcn

cnc_mcerror **cnc_mcsnrtoc32sn_gb18030**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const char **__p_src, *cnc_mcstate_t* *__p_state)

See also:

cnc_mcsnrtoc32sn

cnc_mcerror **cnc_c32snrtomcsn_gb18030**(size_t *__p_maybe_dst_len, char **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

See also:

cnc_c32snrtomcsn

GBK

The legacy GBK encoding. Matches the encoding from the WHATWG Encoding standard.

cnc_mcerror **cnc_mcnrtoc32n_gbk**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const char **__p_src, *cnc_mcstate_t* *__p_state)

See also:

cnc_mcnrtoc32n

cnc_mcerror **cnc_c32nrtomcn_gbk**(size_t *__p_maybe_dst_len, char **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

See also:

cnc_c32nrtomcn

cnc_mcerror **cnc_mcsnrtoc32sn_gbk**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const char **__p_src, *cnc_mcstate_t* *__p_state)

See also:

cnc_mcsnrtoc32sn

cnc_mcerror **cnc_c32snrtomcsn_gbk**(size_t *__p_maybe_dst_len, char **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

See also:

cnc_c32snrtomcsn

Punycode/Punycode (IDNA)

Punycode is a Bootstring Encoding, using configuration and parameters for the Bootstring Algorithm described in RFC3492. Furthermore, there is an IDNA variant that prepends *"xn–"* to Unicode strings during encoding, and removes it during decoding (and otherwise does nothing). It uses custom states to manage the encodings.

Famously, Punycode is used for both Rust ABI identifier name mangling and in DNS for making Unicode names ASCII-only and clearly-marked as Unicode.

struct **cnc_pny_decode_state_t**
A structure containing all of the necessary information for a general-purpose from-punycode transformation to UTF-32.

struct **cnc_pny_encode_state_t**
A structure containing all of the necessary information for a general-purpose to-punycode transformation from UTF-32.

cnc_mcerror **cnc_mcnrtoc32n_punycode**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_pny_decode_state_t* *__p_state)

See also:

cnc_mcnrtoc32n

cnc_mcerror **cnc_c32nrtomcn_punycode**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_pny_encode_state_t* *__p_state)

See also:

cnc_c32nrtomcn

cnc_mcerror **cnc_mcsnrtoc32sn_punycode**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_pny_decode_state_t* *__p_state)

See also:

cnc_mcsnrtoc32sn

cnc_mcerror **cnc_c32snrtomcsn_punycode**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_pny_encode_state_t* *__p_state)

See also:

cnc_c32snrtomcsn

Shift-JIS

The legacy Shift-JIS encoding. Matches the encoding from the WHATWG Encoding standard.

cnc_mcerror **cnc_mcnrtoc32n_shift_jis**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const char **__p_src, *cnc_mcstate_t* *__p_state)

See also:

cnc_mcnrtoc32n

cnc_mcerror **cnc_c32nrtomcn_shift_jis**(size_t *__p_maybe_dst_len, char **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

See also:

cnc_c32nrtomcn

cnc_mcerror **cnc_mcsnrtoc32sn_shift_jis**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const char **__p_src, *cnc_mcstate_t* *__p_state)

See also:

cnc_mcsnrtoc32sn

cnc_mcerror **cnc_c32snrtomcsn_shift_jis**(size_t *__p_maybe_dst_len, char **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

See also:

cnc_c32snrtomcsn

Typed Conversions

Typed conversions are of the form `{prefix}(s?)n(r?)to{suffix}(s?)n`. The prefix/suffix mapping can be found in the *design documentation for conversions*.

The "prefix" represents the source data. The "suffix" represents the destination data. The *s* stands for "string", which means a bulk conversion. The *r* in the name stands for "restartable", which means the function takes an *cnc_mcstate_t* pointer. If there *s* is not present in the name, it is a single conversion function. If the *r* is not present in the name, it is the "non-restartable" version (the version that does not take the state).

Additional encodings not meant to be in the "core set" supported by a typical C or C++ implementation, and that have definitive names other than the unicode encodings, can be found in the *encodings documentation*.

Important: Any function which does not convert to the *execution encoding* or *wide execution encoding* are guaranteed not to touch the locale (as defined by `LC_CTYPE`).

Warning: If an encoding conversion goes to or from either the execution encoding or the wide execution encoding, it may touch the locale which may perform a lock or other operations. If multiple funtion calls are used and `LC_CTYPE` is changed between any of those function calls without properly clearing the *cnc_mcstate_t* object to the initial shift sequence, the behavior of the functions become unspecified.

Bulk Conversion Functions

Note: The description for most of these functions is identical. Any relevant information is contained above.

cnc_mcerror **cnc_mcsntomcsn**(size_t *__p_maybe_dst_len, char **__p_maybe_dst, size_t *__p_src_len, const char **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

This function will create an automatic storage duration `cnc_mcstate_t` object, initialize it to the initial shift sequence, and then pass it to the restartable version of this function. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended

to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_mcsnrtomcsn**(size_t *__p_maybe_dst_len, char **__p_maybe_dst, size_t *__p_src_len, const char **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

The documentation for the type encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (`= {0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_mcsntomwcsn**(size_t *__p_maybe_dst_len, wchar_t **__p_maybe_dst, size_t *__p_src_len, const char **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

This function will create an automatic storage duration *cnc_mcstate_t* object, initialize it to the initial shift sequence, and then pass it to the restartable version of this function. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_mcsnrtomwcsn**(size_t *__p_maybe_dst_len, wchar_t **__p_maybe_dst, size_t *__p_src_len, const char **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

The documentation for the type encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

- **__p_state** – **[inout]** A pointer to the conversion state. If this is nullptr, a value-initialized (= {0} or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_mcsntoc8sn**(size_t *__p_maybe_dst_len, ztd_char8_t **__p_maybe_dst, size_t *__p_src_len, const char **__p_src)

Converts from the encoding given by __p_src's character type to __p_maybe_dst's character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

This function will create an automatic storage duration *cnc_mcstate_t* object, initialize it to the initial shift sequence, and then pass it to the restartable version of this function. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is nullptr, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if __p_maybe_dst is not nullptr).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are nullptr, than this function will not write output data (it may still decrement the value pointed to by __p_maybe_dst_len).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is nullptr or points to a value equivalent to 0, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are nullptr, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_mcsnrtoc8sn**(size_t *__p_maybe_dst_len, ztd_char8_t **__p_maybe_dst, size_t *__p_src_len, const char **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by __p_src's character type to __p_maybe_dst's character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

The documentation for the type encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is nullptr, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if __p_maybe_dst is not nullptr).

- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (`= {0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_mcsntoc16sn**(size_t *__p_maybe_dst_len, ztd_char16_t **__p_maybe_dst, size_t *__p_src_len, const char **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

This function will create an automatic storage duration *cnc_mcstate_t* object, initialize it to the initial shift sequence, and then pass it to the restartable version of this function. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_mcsnrtoc16sn**(size_t *__p_maybe_dst_len, ztd_char16_t **__p_maybe_dst, size_t *__p_src_len, const char **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

The documentation for the type encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (= `{0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_mcsntoc32sn**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const char **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

This function will create an automatic storage duration *cnc_mcstate_t* object, initialize it to the initial shift sequence, and then pass it to the restartable version of this function. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_mcsnrtoc32sn**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const char **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

The documentation for the type encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (= `{0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_mwcsntomcsn**(size_t *__p_maybe_dst_len, char **__p_maybe_dst, size_t *__p_src_len, const wchar_t **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

This function will create an automatic storage duration *cnc_mcstate_t* object, initialize it to the initial shift sequence, and then pass it to the restartable version of this function. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).

- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_mwcsnrtomcsn**(size_t *__p_maybe_dst_len, char **__p_maybe_dst, size_t *__p_src_len, const wchar_t **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

The documentation for the type encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (= `{0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_mwcsntomwcsn**(size_t *__p_maybe_dst_len, wchar_t **__p_maybe_dst, size_t *__p_src_len, const wchar_t **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

This function will create an automatic storage duration `cnc_mcstate_t` object, initialize it to the initial shift sequence, and then pass it to the restartable version of this function. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended

to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_mwcsnrtomwcsn**(size_t *__p_maybe_dst_len, wchar_t **__p_maybe_dst, size_t *__p_src_len, const wchar_t **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

The documentation for the type encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (`= {0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_mwcsntoc8sn**(size_t *__p_maybe_dst_len, ztd_char8_t **__p_maybe_dst, size_t *__p_src_len, const wchar_t **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

This function will create an automatic storage duration *cnc_mcstate_t* object, initialize it to the initial shift sequence, and then pass it to the restartable version of this function. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_mwcsnrtoc8sn**(size_t *__p_maybe_dst_len, ztd_char8_t **__p_maybe_dst, size_t *__p_src_len, const wchar_t **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

The documentation for the type encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (= `{0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_mwcsntoc16sn**(size_t *__p_maybe_dst_len, ztd_char16_t **__p_maybe_dst, size_t *__p_src_len, const wchar_t **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

This function will create an automatic storage duration *cnc_mcstate_t* object, initialize it to the initial shift sequence, and then pass it to the restartable version of this function. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_mwcsnrtoc16sn**(size_t *__p_maybe_dst_len, ztd_char16_t **__p_maybe_dst, size_t *__p_src_len, const wchar_t **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

The documentation for the type encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).

- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (`= {0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_mwcsntoc32sn**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const wchar_t **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

This function will create an automatic storage duration *cnc_mcstate_t* object, initialize it to the initial shift sequence, and then pass it to the restartable version of this function. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_mwcsnrtoc32sn**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const wchar_t **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

The documentation for the type encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (= `{0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_c8sntomcsn**(size_t *__p_maybe_dst_len, char **__p_maybe_dst, size_t *__p_src_len, const ztd_char8_t **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

This function will create an automatic storage duration *cnc_mcstate_t* object, initialize it to the initial shift sequence, and then pass it to the restartable version of this function. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_c8snrtomcsn**(size_t *__p_maybe_dst_len, char **__p_maybe_dst, size_t *__p_src_len, const ztd_char8_t **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

The documentation for the type encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (= `{0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_c8sntomwcsn**(size_t *__p_maybe_dst_len, wchar_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char8_t **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

This function will create an automatic storage duration *cnc_mcstate_t* object, initialize it to the initial shift sequence, and then pass it to the restartable version of this function. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).

- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_c8snrtomwcsn**(size_t *__p_maybe_dst_len, wchar_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char8_t **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

The documentation for the type encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (= `{0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_c8sntoc8sn**(size_t *__p_maybe_dst_len, ztd_char8_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char8_t **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

This function will create an automatic storage duration `cnc_mcstate_t` object, initialize it to the initial shift sequence, and then pass it to the restartable version of this function. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended

to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_c8snrtoc8sn**(size_t *__p_maybe_dst_len, ztd_char8_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char8_t **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

The documentation for the type encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (`= {0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_c8sntoc16sn**(size_t *__p_maybe_dst_len, ztd_char16_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char8_t **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

This function will create an automatic storage duration *cnc_mcstate_t* object, initialize it to the initial shift sequence, and then pass it to the restartable version of this function. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_c8snrtoc16sn**(size_t *__p_maybe_dst_len, ztd_char16_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char8_t **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

The documentation for the type encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (= `{0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_c8sntoc32sn**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char8_t **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

This function will create an automatic storage duration *cnc_mcstate_t* object, initialize it to the initial shift sequence, and then pass it to the restartable version of this function. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_c8snrtoc32sn**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char8_t **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

The documentation for the type encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).

- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are nullptr, than this function will not write output data (it may still decrement the value pointed to by __p_maybe_dst_len).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is nullptr or points to a value equivalent to 0, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are nullptr, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is nullptr, a value-initialized (= {0} or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_c16sntomcsn**(size_t *__p_maybe_dst_len, char **__p_maybe_dst, size_t *__p_src_len, const ztd_char16_t **__p_src)

Converts from the encoding given by __p_src's character type to __p_maybe_dst's character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

This function will create an automatic storage duration *cnc_mcstate_t* object, initialize it to the initial shift sequence, and then pass it to the restartable version of this function. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is nullptr, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if __p_maybe_dst is not nullptr).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are nullptr, than this function will not write output data (it may still decrement the value pointed to by __p_maybe_dst_len).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is nullptr or points to a value equivalent to 0, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are nullptr, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_c16snrtomcsn**(size_t *__p_maybe_dst_len, char **__p_maybe_dst, size_t *__p_src_len, const ztd_char16_t **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by __p_src's character type to __p_maybe_dst's character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

The documentation for the type encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (= `{0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_c16sntomwcsn**(size_t *__p_maybe_dst_len, wchar_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char16_t **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

This function will create an automatic storage duration *cnc_mcstate_t* object, initialize it to the initial shift sequence, and then pass it to the restartable version of this function. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_c16snrtomwcsn**(size_t *__p_maybe_dst_len, wchar_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char16_t **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

The documentation for the type encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (= `{0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_c16sntoc8sn**(size_t *__p_maybe_dst_len, ztd_char8_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char16_t **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

This function will create an automatic storage duration *cnc_mcstate_t* object, initialize it to the initial shift sequence, and then pass it to the restartable version of this function. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).

- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_c16snrtoc8sn**(size_t *__p_maybe_dst_len, ztd_char8_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char16_t **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

The documentation for the type encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (= `{0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_c16sntoc16sn**(size_t *__p_maybe_dst_len, ztd_char16_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char16_t **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

This function will create an automatic storage duration `cnc_mcstate_t` object, initialize it to the initial shift sequence, and then pass it to the restartable version of this function. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended

to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_c16snrtoc16sn**(size_t *__p_maybe_dst_len, ztd_char16_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char16_t **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

The documentation for the type encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (= `{0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_c16sntoc32sn**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char16_t **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

This function will create an automatic storage duration *cnc_mcstate_t* object, initialize it to the initial shift sequence, and then pass it to the restartable version of this function. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_c16snrtoc32sn**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char16_t **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

The documentation for the type encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (= `{0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_c32sntomcsn**(size_t *__p_maybe_dst_len, char **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

This function will create an automatic storage duration *cnc_mcstate_t* object, initialize it to the initial shift sequence, and then pass it to the restartable version of this function. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_c32snrtomcsn**(size_t *__p_maybe_dst_len, char **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

The documentation for the type encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).

- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (`= {0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_c32sntomwcsn**(size_t *__p_maybe_dst_len, wchar_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

This function will create an automatic storage duration *cnc_mcstate_t* object, initialize it to the initial shift sequence, and then pass it to the restartable version of this function. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_c32snrtomwcsn**(size_t *__p_maybe_dst_len, wchar_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

The documentation for the type encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (`= {0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_c32sntoc8sn**(size_t *__p_maybe_dst_len, ztd_char8_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

This function will create an automatic storage duration *cnc_mcstate_t* object, initialize it to the initial shift sequence, and then pass it to the restartable version of this function. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_c32snrtoc8sn**(size_t *__p_maybe_dst_len, ztd_char8_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

The documentation for the type encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (= `{0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_c32sntoc16sn**(size_t *__p_maybe_dst_len, ztd_char16_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

This function will create an automatic storage duration *cnc_mcstate_t* object, initialize it to the initial shift sequence, and then pass it to the restartable version of this function. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).

- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_c32snrtoc16sn**(size_t *__p_maybe_dst_len, ztd_char16_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

The documentation for the type encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (= `{0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_c32sntoc32sn**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

This function will create an automatic storage duration `cnc_mcstate_t` object, initialize it to the initial shift sequence, and then pass it to the restartable version of this function. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended

to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_c32snrtoc32sn**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Performs one unit of indivisible work repeatedly, or stops with an error. Will always stop just before the last complete successful translation of input to output, or will read all input and point to the end if there were no errors.

Remark

The documentation for the type encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (= `{0}` or similar) *cnc_mcstate_t* is used.

Single Conversion Functions

Note: The description for most of these functions is identical. Any relevant information is contained above.

cnc_mcerror **cnc_mcntomcn**(size_t *__p_maybe_dst_len, char **__p_maybe_dst, size_t *__p_src_len, const char **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

The documentation for the `type` to encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **`__p_maybe_dst_len`** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **`__p_maybe_dst`** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **`__p_src_len`** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **`__p_src`** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_mcnrtomcn**(size_t *__p_maybe_dst_len, char **__p_maybe_dst, size_t *__p_src_len, const char **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

This function will create an automatic storage duration *cnc_mcstate_t* object, initialize it to the initial shift sequence, and then use it with this function if `__p_state` is `nullptr`. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **`__p_maybe_dst_len`** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).

- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (`= {0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_mcntomwcn**(size_t *__p_maybe_dst_len, wchar_t **__p_maybe_dst, size_t *__p_src_len, const char **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

The documentation for the `type` to encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_mcnrtomwcn**(size_t *__p_maybe_dst_len, wchar_t **__p_maybe_dst, size_t *__p_src_len, const char **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

This function will create an automatic storage duration `cnc_mcstate_t` object, initialize it to the initial shift sequence, and then use it with this function if `__p_state` is `nullptr`. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended

to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (`= {0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_mcntoc8n**(size_t *__p_maybe_dst_len, ztd_char8_t **__p_maybe_dst, size_t *__p_src_len, const char **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

The documentation for the `type` to encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_mcnrtoc8n**(size_t *__p_maybe_dst_len, ztd_char8_t **__p_maybe_dst, size_t *__p_src_len, const char **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

This function will create an automatic storage duration *cnc_mcstate_t* object, initialize it to the initial shift sequence, and then use it with this function if `__p_state` is `nullptr`. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (= `{0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_mcntoc16n**(size_t *__p_maybe_dst_len, ztd_char16_t **__p_maybe_dst, size_t *__p_src_len, const char **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

The documentation for the `type` to encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_mcnrtoc16n**(size_t *__p_maybe_dst_len, ztd_char16_t **__p_maybe_dst, size_t *__p_src_len, const char **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

This function will create an automatic storage duration *cnc_mcstate_t* object, initialize it to the initial shift sequence, and then use it with this function if `__p_state` is `nullptr`. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (`= {0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_mcntoc32n**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const char **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

The documentation for the `type` to encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).

- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_mcnrtoc32n**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const char **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

This function will create an automatic storage duration *cnc_mcstate_t* object, initialize it to the initial shift sequence, and then use it with this function if `__p_state` is `nullptr`. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (= `{0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_mwcntomcn**(size_t *__p_maybe_dst_len, char **__p_maybe_dst, size_t *__p_src_len, const wchar_t **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

The documentation for the `type` to encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_mwcnrtomcn**(size_t *__p_maybe_dst_len, char **__p_maybe_dst, size_t *__p_src_len, const wchar_t **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

This function will create an automatic storage duration *cnc_mcstate_t* object, initialize it to the initial shift sequence, and then use it with this function if `__p_state` is `nullptr`. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (= `{0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_mwcntomwcn**(size_t *__p_maybe_dst_len, wchar_t **__p_maybe_dst, size_t *__p_src_len, const wchar_t **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

The documentation for the `type` to encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_mwcnrtomwcn**(size_t *__p_maybe_dst_len, wchar_t **__p_maybe_dst, size_t *__p_src_len, const wchar_t **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

This function will create an automatic storage duration *cnc_mcstate_t* object, initialize it to the initial shift sequence, and then use it with this function if `__p_state` is `nullptr`. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (= `{0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_mwcntoc8n**(size_t *__p_maybe_dst_len, ztd_char8_t **__p_maybe_dst, size_t *__p_src_len, const wchar_t **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

The documentation for the `type` to encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_mwcnrtoc8n**(size_t *__p_maybe_dst_len, ztd_char8_t **__p_maybe_dst, size_t *__p_src_len, const wchar_t **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

This function will create an automatic storage duration `cnc_mcstate_t` object, initialize it to the initial shift sequence, and then use it with this function if `__p_state` is `nullptr`. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the `cnc_mcstate_t` object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).

- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (`= {0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_mwcntoc16n**(size_t *__p_maybe_dst_len, ztd_char16_t **__p_maybe_dst, size_t *__p_src_len, const wchar_t **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

The documentation for the `type` to encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_mwcnrtoc16n**(size_t *__p_maybe_dst_len, ztd_char16_t **__p_maybe_dst, size_t *__p_src_len, const wchar_t **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

This function will create an automatic storage duration `cnc_mcstate_t` object, initialize it to the initial shift sequence, and then use it with this function if `__p_state` is `nullptr`. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the `cnc_mcstate_t` object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (`= {0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_mwcntoc32n**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const wchar_t **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

The documentation for the `type` to encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_mwcnrtoc32n**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const wchar_t **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

This function will create an automatic storage duration *cnc_mcstate_t* object, initialize it to the initial shift sequence, and then use it with this function if `__p_state` is `nullptr`. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they

are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (= `{0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_c8ntomcn**(size_t *__p_maybe_dst_len, char **__p_maybe_dst, size_t *__p_src_len, const ztd_char8_t **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

The documentation for the `type` to encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_c8nrtomcn**(size_t *__p_maybe_dst_len, char **__p_maybe_dst, size_t *__p_src_len, const ztd_char8_t **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

This function will create an automatic storage duration *cnc_mcstate_t* object, initialize it to the initial shift sequence, and then use it with this function if `__p_state` is `nullptr`. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (= `{0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_c8ntomwcn**(size_t *__p_maybe_dst_len, wchar_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char8_t **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

The documentation for the `type` to encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.

- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_c8nrtomwcn**(size_t *__p_maybe_dst_len, wchar_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char8_t **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

This function will create an automatic storage duration *cnc_mcstate_t* object, initialize it to the initial shift sequence, and then use it with this function if `__p_state` is `nullptr`. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (= `{0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_c8ntoc8n**(size_t *__p_maybe_dst_len, ztd_char8_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char8_t **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

The documentation for the `type` to encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).

- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_c8nrtoc8n**(size_t *__p_maybe_dst_len, ztd_char8_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char8_t **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

This function will create an automatic storage duration *cnc_mcstate_t* object, initialize it to the initial shift sequence, and then use it with this function if `__p_state` is `nullptr`. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (= `{0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_c8ntoc16n**(size_t *__p_maybe_dst_len, ztd_char16_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char8_t **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

The documentation for the `type` to encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_c8nrtoc16n**(size_t *__p_maybe_dst_len, ztd_char16_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char8_t **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

This function will create an automatic storage duration *cnc_mcstate_t* object, initialize it to the initial shift sequence, and then use it with this function if `__p_state` is `nullptr`. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (= `{0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_c8ntoc32n**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char8_t **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

The documentation for the `type` to encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_c8nrtoc32n**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char8_t **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

This function will create an automatic storage duration `cnc_mcstate_t` object, initialize it to the initial shift sequence, and then use it with this function if `__p_state` is `nullptr`. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the `cnc_mcstate_t` object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).

- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (`= {0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_c16ntomcn**(size_t *__p_maybe_dst_len, char **__p_maybe_dst, size_t *__p_src_len, const ztd_char16_t **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

The documentation for the `type` to encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_c16nrtomcn**(size_t *__p_maybe_dst_len, char **__p_maybe_dst, size_t *__p_src_len, const ztd_char16_t **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

This function will create an automatic storage duration *cnc_mcstate_t* object, initialize it to the initial shift sequence, and then use it with this function if `__p_state` is `nullptr`. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (`= {0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_c16ntomwcn**(size_t *__p_maybe_dst_len, wchar_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char16_t **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

The documentation for the `type` to encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_c16nrtomwcn**(size_t *__p_maybe_dst_len, wchar_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char16_t **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

This function will create an automatic storage duration *cnc_mcstate_t* object, initialize it to the initial shift sequence, and then use it with this function if `__p_state` is `nullptr`. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they

are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (= `{0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_c16ntoc8n**(size_t *__p_maybe_dst_len, ztd_char8_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char16_t **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

The documentation for the `type` to encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_c16nrtoc8n**(size_t *__p_maybe_dst_len, ztd_char8_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char16_t **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

This function will create an automatic storage duration *cnc_mcstate_t* object, initialize it to the initial shift sequence, and then use it with this function if `__p_state` is `nullptr`. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (= `{0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_c16ntoc16n**(size_t *__p_maybe_dst_len, ztd_char16_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char16_t **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

The documentation for the `type` to encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.

- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_c16nrtoc16n**(size_t *__p_maybe_dst_len, ztd_char16_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char16_t **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

This function will create an automatic storage duration *cnc_mcstate_t* object, initialize it to the initial shift sequence, and then use it with this function if `__p_state` is `nullptr`. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (= `{0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_c16ntoc32n**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char16_t **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

The documentation for the `type` to encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).

- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_c16nrtoc32n**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char16_t **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

This function will create an automatic storage duration *cnc_mcstate_t* object, initialize it to the initial shift sequence, and then use it with this function if `__p_state` is `nullptr`. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (= `{0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_c32ntomcn**(size_t *__p_maybe_dst_len, char **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

The documentation for the `type` to encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_c32nrtomcn**(size_t *__p_maybe_dst_len, char **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

This function will create an automatic storage duration *cnc_mcstate_t* object, initialize it to the initial shift sequence, and then use it with this function if `__p_state` is `nullptr`. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (`= {0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_c32ntomwcn**(size_t *__p_maybe_dst_len, wchar_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

The documentation for the `type` to encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_c32nrtomwcn**(size_t *__p_maybe_dst_len, wchar_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

This function will create an automatic storage duration `cnc_mcstate_t` object, initialize it to the initial shift sequence, and then use it with this function if `__p_state` is `nullptr`. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the `cnc_mcstate_t` object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).

- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (`= {0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_c32ntoc8n**(size_t *__p_maybe_dst_len, ztd_char8_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

The documentation for the `type` to encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_c32nrtoc8n**(size_t *__p_maybe_dst_len, ztd_char8_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

This function will create an automatic storage duration *cnc_mcstate_t* object, initialize it to the initial shift sequence, and then use it with this function if `__p_state` is `nullptr`. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (`= {0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_c32ntoc16n**(size_t *__p_maybe_dst_len, ztd_char16_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

The documentation for the `type` to encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_c32nrtoc16n**(size_t *__p_maybe_dst_len, ztd_char16_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

This function will create an automatic storage duration *cnc_mcstate_t* object, initialize it to the initial shift sequence, and then use it with this function if `__p_state` is `nullptr`. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they

are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (= `{0}` or similar) *cnc_mcstate_t* is used.

cnc_mcerror **cnc_c32ntoc32n**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

The documentation for the `type` to encoding mapping can be found in the *Relationship Table For Prefix/Suffix* .

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_c32nrtoc32n**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type. Only performs one unit of indivisible work, or returns an error.

Remark

This function will create an automatic storage duration *cnc_mcstate_t* object, initialize it to the initial shift sequence, and then use it with this function if `__p_state` is `nullptr`. This object is not recoverable in any way and is not shared, and therefore should only be used if the end-user is sure there is no state to the encoding they are working with (e.g., conversions between Unicode Transformation Formats (UTFs). It is **NOT** recommended to use this with the execution encoding and wide execution encodings, which may have shift state and could lead to invalid reads of later data without that shift state information from the *cnc_mcstate_t* object.)

Parameters

- **`__p_maybe_dst_len`** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **`__p_maybe_dst`** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **`__p_src_len`** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **`__p_src`** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **`__p_state`** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (`= {0}` or similar) *cnc_mcstate_t* is used.

Generic Typed Conversions

Generic typed conversions rely on the types being put in to determine what encodings and conversions should be done. They are more flexibile when the input is generic, or the user has well-defined source and destination pointers to use with the API. The type-to-prefix/encoding mapping for this function is described in the *naming documentation*. When using these functions, using `nullptr` is ambiguous because the macro/template cannot understand what the to / from pointers should be. In those cases, cast the `nullptr` value with `(CharTypeHere**)nullptr`.

Bulk Conversion Functions

`cnc_cxsntocysn`(__p_maybe_dst_len, __p_maybe_dst, __p_src_len, __p_src)
Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type in **bulk**. Identical in behavior to *cnc_cxsnrtocysn*, with the `__p_state` argument passed in from a properly initialized *cnc_mcstate_t* object of automatic storage duration (it is a "stack"-based variable that does not live beyond the call of this function).

`cnc_cxsnrtocysn`(__p_maybe_dst_len, __p_maybe_dst, __p_src_len, __p_src, __p_state)
Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type **one at a time**.

Remark

Note that it is impossible for this type to handle `nullptr` properly for its destination and source arguments because of its templated nature, since that is how it derives the type. Therefore, the `nullptr` passed in must first be coerced to a type with a cast, for example:

```
cnc_mcerror err = cnc_cxsntocysn(&required_len, (char**)nullptr, &src_len, &src_ptr,
↪ &state);
```

Parameters

- **__p_maybe_dst_len** – **[inout]** A poinvter to the size of the output buffer. If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).
- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (`= {0}` or similar) *cnc_mcstate_t* is used.

Single Conversion Functions

cnc_cxntocyn(__p_maybe_dst_len, __p_maybe_dst, __p_src_len, __p_src)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type, one at a time. Identical in behavior to *cnc_cxnrtocyn*, with the `__p_state` argument passed in from a properly initialized *cnc_mcstate_t* object of automatic storage duration (it is a "stack"-based variable that does not live beyond the call of this function).

cnc_cxnrtocyn(__p_maybe_dst_len, __p_maybe_dst, __p_src_len, __p_src, __p_state)

Converts from the encoding given by `__p_src`'s character type to `__p_maybe_dst`'s character type.

Remark

Note that it is impossible for this type to handle `nullptr` properly for its destination and source arguments because of its templated nature, since that is how it derives the type. Therefore, the `nullptr` passed in must first be coerced to a type with a cast, for example:

```
cnc_mcerror err = cnc_cxsntocysn(&required_len, (char**)nullptr, &src_len, &src_ptr,
↪ &state);
```

Parameters

- **__p_maybe_dst_len** – **[inout]** A pointer to the size of the output buffer (in number of **elements**). If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__p_maybe_dst` is not `nullptr`).

- **__p_maybe_dst** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__p_maybe_dst_len`).
- **__p_src_len** – **[inout]** A pointer to the size of the input buffer (in number of **elements**). If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_src** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_state** – **[inout]** A pointer to the conversion state. If this is `nullptr`, a value-initialized (`= {0}` or similar) *cnc_mcstate_t* is used.

1.4.3 Registry (Untyped) Conversions

Conversion Registry

typedef struct *cnc_conversion_registry* **cnc_conversion_registry**

A `typedef` to allow for plain and normal usage of the type name `cnc_conversion_registry`.

cnc_open_error **cnc_registry_new**(*cnc_conversion_registry* **__out_p_registry, *cnc_registry_options* __registry_options)

Creates a new registry.

Remark

This will default to using a normal *cnc_conversion_heap* which uses the globally-available allocator (malloc, free, realloc, etc.).

Parameters

- **__out_p_registry** – **[inout]** The output pointer to the handle of the cnc_conversion_registry that will be created.
- **__registry_options** – **[in]** The options that affect how this registry is created.

cnc_open_error **cnc_registry_open**(*cnc_conversion_registry* **__out_p_registry, *cnc_conversion_heap* *__p_heap, *cnc_registry_options* __registry_options)

Creates a new registry.

Remark

All allocations shall be done through the passed-in heap if needed. It is unspecified if the implementation can or will use the heap at all (e.g., there is a small buffer optimization applied).

Parameters

- **__out_p_registry** – **[inout]** The output pointer to the handle of the cnc_conversion_registry that will be created.

- **__p_heap** – **[in]** A pointer to the heap to use. The heap this points to will be copied into the registry upon successful creation.
- **__registry_options** – **[in]** The options that affect how this registry is created.

cnc_open_error **cnc_registry_add**(*cnc_conversion_registry* *__registry, const char *__from, const char *__to, cnc_conversion_function *__multi_conversion_function, cnc_conversion_function *__single_conversion_function, cnc_state_is_complete_function *__state_is_complete_function, cnc_open_function *__open_function, cnc_close_function *__close_function)

Adds a new conversion from the specified `__from` and `__to` names to the specified registry.

Remark

This function has identical behavior to cnc_registry_add_n, where the `__from_size` and `__to_size` arguments are calculated by calling the equivalent of `strlen` on `__from` and `__to`, respectively. If `__from` or `__to` are `nullptr`, then the function will assume they are the empty string (and use the default name in that case).

Parameters

- **__registry** – **[in]** The registry to create the new conversion pair in.
- **__from** – **[in]** A pointer to a null-terminated c string encoded in UTF-8 representing the name of the encoding to convert from. Can be `nullptr`.
- **__to** – **[in]** A pointer to a null-terminated c string encoded in UTF-8 representing the name of the encoding to convert to. Can be `nullptr`.
- **__multi_conversion_function** – **[in]** The conversion cnc_conversion_function which will perform a bulk conversion (consumes as much input as is available until exhausted or an error occurs). Can be `nullptr`, but only if the `__single_conversion_function` is not `nullptr` as well.
- **__single_conversion_function** – **[in]** The conversion cnc_conversion_function which will perform a singular conversion (consumes only one completely unit of input and produces on complete unit of output). Can be `nullptr`, but only if the `__multi_conversion_function` is not `nullptr` as well.
- **__state_is_complete_function** – **[in]** A function to use to check if, when the input is empty, if there is still leftover data to be output from the state.
- **__open_function** – **[in]** The cnc_open_function to be used for allocating additional space during function calls which open new cnc_conversion handles. Can be `nullptr`.
- **__close_function** – **[in]** The cnc_close_function to be used for allocating additional space during function calls which open new cnc_conversion handles. Can be `nullptr`.

cnc_open_error **cnc_registry_add_n**(*cnc_conversion_registry* *__registry, size_t __from_size, const char __from[ZTD_PTR_EXTENT(*__from_size*)], size_t __to_size, const char __to[ZTD_PTR_EXTENT(*__to_size*)], cnc_conversion_function *__multi_conversion_function, cnc_conversion_function *__single_conversion_function, cnc_state_is_complete_function *__state_is_complete_function, cnc_open_function *__open_function, cnc_close_function *__close_function)

Adds a new conversion from the specified `__from` and `__to` names to the specified registry.

Remark

This function has identical behavior to cnc_registry_add_n, where the `__from_size` and `__to_size` arguments are calculated by calling the equivalent of `strlen` on `__from` and `__to`, respectively. If `__from` or `__to` are `nullptr`, then the function will assume they are the empty string (and use the default name in that case).

Parameters

- **`__registry`** – **[in]** The registry to create the new conversion pair in.
- **`__from_size`** – **[in]** The number of code units in the `__from` parameter.
- **`__from`** – **[in]** A pointer to a string encoded in UTF-8 representing the name of the encoding to convert from. The string need not be null-terminated. Can be `nullptr`.
- **`__to_size`** – **[in]** The number of code units in the `__to` parameter.
- **`__to`** – **[in]** A pointer to a string encoded in UTF-8 representing the name of the encoding to convert to. The string need not be null-terminated. Can be `nullptr`.
- **`__multi_conversion_function`** – **[in]** The conversion cnc_conversion_function which will perform a bulk conversion (consumes as much input as is available until exhausted or an error occurs). Can be `nullptr`, but only if the `__single_conversion_function` is not `nullptr` as well.
- **`__single_conversion_function`** – **[in]** The conversion cnc_conversion_function which will perform a singular conversion (consumes only one completely unit of input and produces on complete unit of output). Can be `nullptr`, but only if the `__multi_conversion_function` is not `nullptr` as well.
- **`__state_is_complete_function`** – **[in]** A function to use to check if, when the input is empty, if there is still leftover data to be output from the state.
- **`__open_function`** – **[in]** The cnc_open_function to be used for allocating additional space during function calls which open new cnc_conversion handles. Can be `nullptr`.
- **`__close_function`** – **[in]** The cnc_close_function to be used for allocating additional space during function calls which open new cnc_conversion handles. Can be `nullptr`.

cnc_open_error **`cnc_registry_add_multi`**(*cnc_conversion_registry* *__registry, const char *__from, const char *__to, cnc_conversion_function *__multi_conversion_function, cnc_state_is_complete_function *__state_is_complete_function, cnc_open_function *__open_function, cnc_close_function *__close_function)

Adds a new conversion from the specified `__from` and `__to` names to the specified registry.

Remark

Identical to calling cnc_registry_add_n, with the `__multi_conversion_function` parameter set to `nullptr`. The `__from_size` and `__to_size` arguments are calculated by calling the equivalent of `strlen` on `__from` and `__to`, respectively. If `__from` or `__to` are `nullptr`, then the function will assume they are the empty string (and use the default name in that case).

Parameters

- **__registry** – **[in]** The registry to create the new conversion pair in.
- **__from** – **[in]** A pointer to a null-terminated c string encoded in UTF-8 representing the encoding to convert from. Can be `nullptr`.
- **__to** – **[in]** A pointer to a null-terminated c string encoded in UTF-8 representing the encoding to convert to. Can be `nullptr`.
- **__multi_conversion_function** – **[in]** The conversion cnc_conversion_function which will perform a bulk conversion (consumes as much input as is available until exhausted or an error occurs). Can be `nullptr`, but only if the `__single_conversion_function` is not `nullptr` as well.
- **__state_is_complete_function** – **[in]** A function to use to check if, when the input is empty, if there is still leftover data to be output from the state.
- **__open_function** – **[in]** The cnc_open_function to be used for allocating additional space during function calls which open new cnc_conversion handles. Can be `nullptr`.
- **__close_function** – **[in]** The cnc_close_function to be used for allocating additional space during function calls which open new cnc_conversion handles. Can be `nullptr`.

cnc_open_error **cnc_registry_add_n_multi**(*cnc_conversion_registry* *__registry, size_t __from_size, const char __from[ZTD_PTR_EXTENT(*__from_size*)], size_t __to_size, const char __to[ZTD_PTR_EXTENT(*__to_size*)], cnc_conversion_function *__multi_conversion_function, cnc_state_is_complete_function *__state_is_complete_function, cnc_open_function *__open_function, cnc_close_function *__close_function)

Adds a new conversion from the specified `__from` and `__to` names to the specified registry.

Remark

Identical to calling cnc_registry_add_n, with the `__single_conversion_function` parameter set to `nullptr`.

Parameters

- **__registry** – **[in]** The registry to create the new conversion pair in.
- **__from_size** – **[in]** The number of code units in the `__from` parameter.
- **__from** – **[in]** A pointer to a string encoded in UTF-8 representing the encoding to convert from. The string need not be null-terminated. Can be `nullptr`.
- **__to_size** – **[in]** The number of code units in the `__to` parameter.
- **__to** – **[in]** A pointer to a string encoded in UTF-8 representing the encoding to convert to. The string need not be null-terminated. Can be `nullptr`.
- **__multi_conversion_function** – **[in]** The conversion cnc_conversion_function which will perform a bulk conversion (consumes as much input as is available until exhausted or an error occurs). Can be `nullptr`, but only if the `__single_conversion_function` is not `nullptr` as well.
- **__state_is_complete_function** – **[in]** A function to use to check if, when the input is empty, if there is still leftover data to be output from the state.

- **__open_function** – **[in]** The cnc_open_function to be used for allocating additional space during function calls which open new cnc_conversion handles. Can be `nullptr`.
- **__close_function** – **[in]** The cnc_close_function to be used for allocating additional space during function calls which open new cnc_conversion handles. Can be `nullptr`.

cnc_open_error **cnc_registry_add_single**(*cnc_conversion_registry* *__registry, const char *__from, const char *__to, cnc_conversion_function *__single_conversion_function, cnc_state_is_complete_function *__state_is_complete_function, cnc_open_function *__open_function, cnc_close_function *__close_function)

Adds a new conversion from the specified `__from` and `__to` names to the specified registry.

Remark

Identical to calling cnc_registry_add_n, with the `__multi_conversion_function` parameter set to `nullptr`. The `__from_size` and `__to_size` arguments are calculated by calling the equivalent of `strlen` on `__from` and `__to`, respectively. If `__from` or `__to` are `nullptr`, then the function will assume they are the empty string (and use the default name in that case).

Parameters

- **__registry** – **[in]** The registry to create the new conversion pair in.
- **__from** – **[in]** A pointer to a null-terminated c string encoded in UTF-8 representing the encoding to convert from. Can be `nullptr`.
- **__to** – **[in]** A pointer to a null-terminated c string encoded in UTF-8 representing the encoding to convert to. Can be `nullptr`.
- **__single_conversion_function** – **[in]** The conversion cnc_conversion_function which will perform a singular conversion (consumes only one completely unit of input and produces on complete unit of output). Shall not be `nullptr`.
- **__state_is_complete_function** – **[in]** A function to use to check if, when the input is empty, if there is still leftover data to be output from the state.
- **__open_function** – **[in]** The cnc_open_function to be used for allocating additional space during function calls which open new cnc_conversion handles. Can be `nullptr`.
- **__close_function** – **[in]** The cnc_close_function to be used for allocating additional space during function calls which open new cnc_conversion handles. Can be `nullptr`.

cnc_open_error **cnc_registry_add_n_single**(*cnc_conversion_registry* *__registry, size_t __from_size, const char __from[ZTD_PTR_EXTENT(*__from_size*)], size_t __to_size, const char __to[ZTD_PTR_EXTENT(*__to_size*)], cnc_conversion_function *__single_conversion_function, cnc_state_is_complete_function *__state_is_complete_function, cnc_open_function *__open_function, cnc_close_function *__close_function)

Adds a new conversion from the specified `__from` and `__to` names to the specified registry.

Remark

Identical to calling cnc_registry_add_n, with the `__multi_conversion_function` parameter set to `nullptr`.

Parameters

- **__registry** – **[in]** The registry to create the new conversion pair in.
- **__from_size** – **[in]** The number of code units in the `__from` parameter.
- **__from** – **[in]** A pointer to a string encoded in UTF-8 representing the encoding to convert from. The string need not be null-terminated. Can be `nullptr`.
- **__to_size** – **[in]** The number of code units in the `__to` parameter.
- **__to** – **[in]** A pointer to a string encoded in UTF-8 representing the encoding to convert to. The string need not be null-terminated. Can be `nullptr`.
- **__single_conversion_function** – **[in]** The conversion cnc_conversion_function which will perform a singular conversion (consumes only one completely unit of input and produces on complete unit of output). Shall not be `nullptr`.
- **__state_is_complete_function** – **[in]** A function to use to check if, when the input is empty, if there is still leftover data to be output from the state.
- **__open_function** – **[in]** The cnc_open_function to be used for allocating additional space during function calls which open new cnc_conversion handles. Can be `nullptr`.
- **__close_function** – **[in]** The cnc_close_function to be used for allocating additional space during function calls which open new cnc_conversion handles. Can be `nullptr`.

cnc_open_error **cnc_registry_add_c8**(*cnc_conversion_registry* *__registry, const ztd_char8_t *__from, const ztd_char8_t *__to, cnc_conversion_function *__multi_conversion_function, cnc_conversion_function *__single_conversion_function, cnc_state_is_complete_function *__state_is_complete_function, cnc_open_function *__open_function, cnc_close_function *__close_function)

Adds a new conversion from the specified `__from` and `__to` names to the specified registry.

Remark

This function has identical behavior to cnc_registry_add_n, where the `__from_size` and `__to_size` arguments are calculated by calling the equivalent of `strlen` on `__from` and `__to`, respectively. If `__from` or `__to` are `nullptr`, then the function will assume they are the empty string (and use the default name in that case).

Parameters

- **__registry** – **[in]** The registry to create the new conversion pair in.
- **__from** – **[in]** A pointer to a null-terminated c string encoded in UTF-8 representing the name of the encoding to convert from. Can be `nullptr`.
- **__to** – **[in]** A pointer to a null-terminated c string encoded in UTF-8 representing the name of the encoding to convert to. Can be `nullptr`.
- **__multi_conversion_function** – **[in]** The conversion cnc_conversion_function which will perform a bulk conversion (consumes as much input as is available until exhausted or an error occurs). Can be `nullptr`, but only if the `__single_conversion_function` is not `nullptr` as well.

- **__single_conversion_function** – **[in]** The conversion cnc_conversion_function which will perform a singular conversion (consumes only one completely unit of input and produces on complete unit of output). Can be `nullptr`, but only if the `__multi_conversion_function` is not `nullptr` as well.
- **__state_is_complete_function** – **[in]** A function to use to check if, when the input is empty, if there is still leftover data to be output from the state.
- **__open_function** – **[in]** The cnc_open_function to be used for allocating additional space during function calls which open new cnc_conversion handles. Can be `nullptr`.
- **__close_function** – **[in]** The cnc_close_function to be used for allocating additional space during function calls which open new cnc_conversion handles. Can be `nullptr`.

cnc_open_error **cnc_registry_add_c8n**(*cnc_conversion_registry* *__registry, size_t __from_size, const ztd_char8_t __from[ZTD_PTR_EXTENT(*__from_size*)], size_t __to_size, const ztd_char8_t __to[ZTD_PTR_EXTENT(*__to_size*)], cnc_conversion_function *__multi_conversion_function, cnc_conversion_function *__single_conversion_function, cnc_state_is_complete_function *__state_is_complete_function, cnc_open_function *__open_function, cnc_close_function *__close_function)

Adds a new conversion from the specified `__from` and `__to` names to the specified registry.

Remark

This function has identical behavior to cnc_registry_add_n, where the `__from_size` and `__to_size` arguments are calculated by calling the equivalent of `strlen` on `__from` and `__to`, respectively. If `__from` or `__to` are `nullptr`, then the function will assume they are the empty string (and use the default name in that case).

Parameters

- **__registry** – **[in]** The registry to create the new conversion pair in.
- **__from_size** – **[in]** The number of code units in the `__from` parameter.
- **__from** – **[in]** A pointer to a string encoded in UTF-8 representing the name of the encoding to convert from. The string need not be null-terminated. Can be `nullptr`.
- **__to_size** – **[in]** The number of code units in the `__to` parameter.
- **__to** – **[in]** A pointer to a string encoded in UTF-8 representing the name of the encoding to convert to. The string need not be null-terminated. Can be `nullptr`.
- **__multi_conversion_function** – **[in]** The conversion cnc_conversion_function which will perform a bulk conversion (consumes as much input as is available until exhausted or an error occurs). Can be `nullptr`, but only if the `__single_conversion_function` is not `nullptr` as well.
- **__single_conversion_function** – **[in]** The conversion cnc_conversion_function which will perform a singular conversion (consumes only one completely unit of input and produces on complete unit of output). Can be `nullptr`, but only if the `__multi_conversion_function` is not `nullptr` as well.
- **__state_is_complete_function** – **[in]** A function to use to check if, when the input is empty, if there is still leftover data to be output from the state.

- **__open_function** – **[in]** The cnc_open_function to be used for allocating additional space during function calls which open new cnc_conversion handles. Can be `nullptr`.
- **__close_function** – **[in]** The cnc_close_function to be used for allocating additional space during function calls which open new cnc_conversion handles. Can be `nullptr`.

cnc_open_error **cnc_registry_add_c8_multi**(*cnc_conversion_registry* *__registry, const ztd_char8_t *__from, const ztd_char8_t *__to, cnc_conversion_function *__multi_conversion_function, cnc_state_is_complete_function *__state_is_complete_function, cnc_open_function *__open_function, cnc_close_function *__close_function)

Adds a new conversion from the specified `__from` and `__to` names to the specified registry.

Remark

Identical to calling cnc_registry_add_n, with the `__multi_conversion_function` parameter set to `nullptr`. The `__from_size` and `__to_size` arguments are calculated by calling the equivalent of `strlen` on `__from` and `__to`, respectively. If `__from` or `__to` are `nullptr`, then the function will assume they are the empty string (and use the default name in that case).

Parameters

- **__registry** – **[in]** The registry to create the new conversion pair in.
- **__from** – **[in]** A pointer to a null-terminated c string encoded in UTF-8 representing the encoding to convert from. Can be `nullptr`.
- **__to** – **[in]** A pointer to a null-terminated c string encoded in UTF-8 representing the encoding to convert to. Can be `nullptr`.
- **__multi_conversion_function** – **[in]** The conversion cnc_conversion_function which will perform a bulk conversion (consumes as much input as is available until exhausted or an error occurs). Can be `nullptr`, but only if the `__single_conversion_function` is not `nullptr` as well.
- **__state_is_complete_function** – **[in]** A function to use to check if, when the input is empty, if there is still leftover data to be output from the state.
- **__open_function** – **[in]** The cnc_open_function to be used for allocating additional space during function calls which open new cnc_conversion handles. Can be `nullptr`.
- **__close_function** – **[in]** The cnc_close_function to be used for allocating additional space during function calls which open new cnc_conversion handles. Can be `nullptr`.

cnc_open_error **cnc_registry_add_c8n_multi**(*cnc_conversion_registry* *__registry, size_t __from_size, const ztd_char8_t __from[ZTD_PTR_EXTENT(*__from_size*)], size_t __to_size, const ztd_char8_t __to[ZTD_PTR_EXTENT(*__to_size*)], cnc_conversion_function *__multi_conversion_function, cnc_state_is_complete_function *__state_is_complete_function, cnc_open_function *__open_function, cnc_close_function *__close_function)

Adds a new conversion from the specified `__from` and `__to` names to the specified registry.

Remark

Identical to calling cnc_registry_add_n, with the `__single_conversion_function` parameter set to `nullptr`.

Parameters

- **__registry** – **[in]** The registry to create the new conversion pair in.
- **__from_size** – **[in]** The number of code units in the `__from` parameter.
- **__from** – **[in]** A pointer to a string encoded in UTF-8 representing the encoding to convert from. The string need not be null-terminated. Can be `nullptr`.
- **__to_size** – **[in]** The number of code units in the `__to` parameter.
- **__to** – **[in]** A pointer to a string encoded in UTF-8 representing the encoding to convert to. The string need not be null-terminated. Can be `nullptr`.
- **__multi_conversion_function** – **[in]** The conversion cnc_conversion_function which will perform a bulk conversion (consumes as much input as is available until exhausted or an error occurs). Can be `nullptr`, but only if the `__single_conversion_function` is not `nullptr` as well.
- **__state_is_complete_function** – **[in]** A function to use to check if, when the input is empty, if there is still leftover data to be output from the state.
- **__open_function** – **[in]** The cnc_open_function to be used for allocating additional space during function calls which open new cnc_conversion handles. Can be `nullptr`.
- **__close_function** – **[in]** The cnc_close_function to be used for allocating additional space during function calls which open new cnc_conversion handles. Can be `nullptr`.

cnc_open_error **cnc_registry_add_c8_single**(*cnc_conversion_registry* *__registry, const ztd_char8_t *__from, const ztd_char8_t *__to, cnc_conversion_function *__single_conversion_function, cnc_state_is_complete_function *__state_is_complete_function, cnc_open_function *__open_function, cnc_close_function *__close_function)

Adds a new conversion from the specified `__from` and `__to` names to the specified registry.

Remark

Identical to calling cnc_registry_add_n, with the `__multi_conversion_function` parameter set to `nullptr`. The `__from_size` and `__to_size` arguments are calculated by calling the equivalent of `strlen` on `__from` and `__to`, respectively. If `__from` or `__to` are `nullptr`, then the function will assume they are the empty string (and use the default name in that case).

Parameters

- **__registry** – **[in]** The registry to create the new conversion pair in.
- **__from** – **[in]** A pointer to a null-terminated c string encoded in UTF-8 representing the encoding to convert from. Can be `nullptr`.
- **__to** – **[in]** A pointer to a null-terminated c string encoded in UTF-8 representing the encoding to convert to. Can be `nullptr`.

- **__single_conversion_function** – **[in]** The conversion cnc_conversion_function which will perform a singular conversion (consumes only one completely unit of input and produces on complete unit of output). Shall not be `nullptr`.
- **__state_is_complete_function** – **[in]** A function to use to check if, when the input is empty, if there is still leftover data to be output from the state.
- **__open_function** – **[in]** The cnc_open_function to be used for allocating additional space during function calls which open new cnc_conversion handles. Can be `nullptr`.
- **__close_function** – **[in]** The cnc_close_function to be used for allocating additional space during function calls which open new cnc_conversion handles. Can be `nullptr`.

cnc_open_error **cnc_registry_add_c8n_single**(*cnc_conversion_registry* *__registry, size_t __from_size, const ztd_char8_t __from[ZTD_PTR_EXTENT(*__from_size*)], size_t __to_size, const ztd_char8_t __to[ZTD_PTR_EXTENT(*__to_size*)], cnc_conversion_function *__single_conversion_function, cnc_state_is_complete_function *__state_is_complete_function, cnc_open_function *__open_function, cnc_close_function *__close_function)

Adds a new conversion from the specified `__from` and `__to` names to the specified registry.

Remark

Identical to calling cnc_registry_add_n, with the `__multi_conversion_function` parameter set to `nullptr`.

Parameters

- **__registry** – **[in]** The registry to create the new conversion pair in.
- **__from_size** – **[in]** The number of code units in the `__from` parameter.
- **__from** – **[in]** A pointer to a string encoded in UTF-8 representing the encoding to convert from. The string need not be null-terminated. Can be `nullptr`.
- **__to_size** – **[in]** The number of code units in the `__to` parameter.
- **__to** – **[in]** A pointer to a string encoded in UTF-8 representing the encoding to convert to. The string need not be null-terminated. Can be `nullptr`.
- **__single_conversion_function** – **[in]** The conversion cnc_conversion_function which will perform a singular conversion (consumes only one completely unit of input and produces on complete unit of output). Shall not be `nullptr`.
- **__state_is_complete_function** – **[in]** A function to use to check if, when the input is empty, if there is still leftover data to be output from the state.
- **__open_function** – **[in]** The cnc_open_function to be used for allocating additional space during function calls which open new cnc_conversion handles. Can be `nullptr`.
- **__close_function** – **[in]** The cnc_close_function to be used for allocating additional space during function calls which open new cnc_conversion handles. Can be `nullptr`.

void **cnc_registry_close**(*cnc_conversion_registry* *__registry)

Closes an open registry.

Remark

This function MUST be paired with *cnc_registry_open*. It cannot be paired with any other creation function. This will close the registry before it's memory is deleted/freed: see cnc_registry_close for more information. If `registry` is `nullptr`, this function will do nothing.

Parameters **__registry** – **[in]** The registry to delete.

void **cnc_registry_delete**(*cnc_conversion_registry* *__registry)

Deletes a registry.

Remark

This function MUST be paired with *cnc_registry_new*. It cannot be paired with any other creation function. This will close the registry before it's memory is deleted/freed: see cnc_registry_close for more information. If `registry` is `nullptr`, this function will do nothing.

Parameters **__registry** – **[in]** The registry to delete.

void **cnc_pairs_c8_list**(const *cnc_conversion_registry* *__registry, cnc_conversion_registry_pair_c8_function *__callback_function, void *__user_data)

Provides the list of encoding conversions currently registered to the provided `__registry`.

Remark

This functions does not modify the contents of the registry and therefore can be called from any number of threads simultaneously. The callback function is invoked once more each pair. Note that each conversion pair is distinct from the others: a conversion pair of ("UTF-8", "SHIFT-JIS") is distinct from ("SHIFT-JIS", "UTF-8"). If `registry` or `__callback_function` is `nullptr`, this function will do nothing.

Parameters

- **__registry** – **[in]** The conversion registry whose conversion pairs should be iterated through and passed to the function.
- **__callback_function** – **[in]** The function that should be called with each conversion pair. See cnc_conversion_registry_pair_function for more details about the expectation of each given parameter.
- **__user_data** – **[in]** A pointer to data that should be given to the `__callback_function`.

void **cnc_pairs_list**(const *cnc_conversion_registry* *__registry, cnc_conversion_registry_pair_function *__callback_function, void *__user_data)

Provides the list of encoding conversions currently registered to the provided `__registry`.

Remark

This functions does not modify the contents of the registry and therefore can be called from any number of threads simultaneously. The callback function is invoked once more each pair. Note that each conversion pair is distinct from the others: a conversion pair of ("UTF-8", "SHIFT-JIS") is distinct from ("SHIFT-JIS", "UTF-8"). If `registry` or `__callback_function` is `nullptr`, this function will do nothing.

Parameters

- **__registry** – **[in]** The conversion registry whose conversion pairs should be iterated through and passed to the function.
- **__callback_function** – **[in]** The function that should be called with each conversion pair. See cnc_conversion_registry_pair_function for more details about the expectation of each given parameter.
- **__user_data** – **[in]** A pointer to data that should be given to the `__callback_function`.

cnc_registry_options

enum **cnc_registry_options**
The options which change how a registry is initialized and adjusted upon creation.

Values:

enumerator **CNC_REGISTRY_OPTIONS_NONE**
No options.

enumerator **CNC_REGISTRY_OPTIONS_EMPTY**
Start with an empty registry that contains none of the platorm's default conversion entries.

enumerator **CNC_REGISTRY_OPTIONS_DEFAULT**
Use the default options recommended when starting a registry.

Registry Function Types

There are many function types used to perform work related to the registry, or hook in user behavior. They are detailed below and are used in the various *registry functions*.

Note: Breathe and Doxygen is broken by the function typedefs, so they are not shown here at the moment.

---

cnc_conversion_info

struct **cnc_conversion_info**
A structure which tracks information about the final opened cnc_conversion handle.

Remark

This structure is the only time the collection of cnc_conversion creating and opening functions will return information about whether or not it uses an indirect conversion and that conversion's properties.

Public Members

const ztd_char8_t ***from_code_data**
A pointer to the the `from_code` data.

Remark

This will always be a null-terminated pointer, but does not guarantee there may not be embedded nulls.

size_t **from_code_size**
The size of the `from_code` data.

const ztd_char8_t ***to_code_data**
A pointer to the the `to_code` data.

Remark

This will always be a null-terminated pointer, but does not guarantee there may not be embedded nulls.

size_t **to_code_size**
The size of the `to_code` data.

bool **is_indirect**
Whether or not this conversion uses an indirect conversion.

Remark

An indirect conversion goes through an intermediate encoding to reach it's final destination. This is typical for most conversions which encoding to and from some form of Unicode, but do not translate to each other.

const ztd_char8_t ***indirect_code_data**
The name of the indirect encoding.

Remark

This is `nullptr` if is_indirect is false. If it is not `nullptr`, this will be a null-terminated pointer. But, it does not guarantee there may not be embedded nulls.

size_t **indirect_code_size**
The size of the indirect_code data.

Remark

This is 0 if is_indirect is false.

Registry-Based Conversions: Resource Handles

cnc_conversion is first and foremost a handle to a resource. It must be opened/created like one, and destroyed like one. The *_new flavor of functions perform the allocation automatically using the *associated registry's heap*. The *_open flavor fo functions let the user pass in an area of memory (or probe to find out the exact area of memory) to open the handle into. The *_open flavor of functions is for particularly advanced users who want maximum control over where the type is placed and allocated and is much more complicated to use: users are recommended to use *cnc_conv_new()* and *cnc_conv_new_n()* where appropriate to save on precision handling and hassle.

cnc_conversion Creation

typedef struct *cnc_conversion* **cnc_conversion**
A typedef to allow for plain and normal usage of the type name cnc_conversion.

cnc_open_error **cnc_conv_open**(*cnc_conversion_registry* *__registry, const char *__from, const char *__to, *cnc_conversion* **__out_p_conversion, size_t *__p_available_space, unsigned char *__space, *cnc_conversion_info* *__p_info)
Opens a new encoding in the provided space.

Remark

This call defers to calling cnc_conv_open_n after computing the length of the __from and __to parameters, if they are not nullptr. If either is nullptr, their size is assumed to be 0.

Parameters

- **__registry** – **[in]** The registry to use for opening the cnc_conversion handle.
- **__from** – **[in]** A pointer to data encoded as UTF-8 representing the encoding to transcode from.
- **__to** – **[in]** A pointer to data encoded as UTF-8 representing the encoding to transcode to.
- **__out_p_conversion** – **[inout]** A pointer to the cnc_conversion handle to open.
- **__p_available_space** – **[inout]** The amount of space available in the __space parameter, in the number of bytes, that can be used to allocate any necessary data for the cnc_conversion handle.
- **__space** – **[inout]** A pointer to space which can be used for opening up the conversion handle, which has the size indicated by __p_available_space.

- **__p_info** – **[inout]** A pointer to an already-created *cnc_conversion_info* that will be filled out with information regarding how the transcoding operation was opened, if it was successful.

cnc_open_error **cnc_conv_open_n**(*cnc_conversion_registry* *__registry, size_t __from_size, const char __from[ZTD_PTR_EXTENT(*__from_size*)], size_t __to_size, const char __to[ZTD_PTR_EXTENT(*__to_size*)], *cnc_conversion* **__out_p_conversion, size_t *__p_available_space, void *__space, *cnc_conversion_info* *__p_info)

Opens a new encoding in the provided space an returns it through `__out_p_conversion`.

Remark

If there is a conversion that can be achieved by using an intermediate encoding (e.g., converting from the desired `__from` encoding to UTF-32, then from UTF-32 to the desired `__to` encoding), then an indirect transcode will be opened if a direct encoding cannot be opened. If there is not enough space to write out, then this function will provide the amount needed in `__p_available_space` directly. Otherwise, it will decrement the value pointed to be `__p_available_space` by the amount of space used.

Parameters

- **__registry** – **[in]** The registry to use for opening the cnc_conversion handle.
- **__from_size** – **[in]** The size of the `__from` string.
- **__from** – **[in]** A pointer to data encoded as UTF-8 representing the encoding to transcode from.
- **__to_size** – **[in]** The size of the `__to` string.
- **__to** – **[in]** A pointer to data encoded as UTF-8 representing the encoding to transcode to.
- **__out_p_conversion** – **[inout]** A pointer to the cnc_conversion handle to open.
- **__p_available_space** – **[inout]** The amount of space available in the `__space` parameter, in the number of bytes, that can be used to allocate any necessary data for the cnc_conversion handle.
- **__space** – **[inout]** A pointer to space which can be used for opening up the conversion handle, which has the size indicated by `__p_available_space`.
- **__p_info** – **[inout]** A pointer to an already-created *cnc_conversion_info* that will be filled out with information regarding how the transcoding operation was opened, if it was successful.

cnc_open_error **cnc_conv_open_select**(*cnc_conversion_registry* *__registry, const char *__from, const char *__to, cnc_indirect_selection_function *__selection, *cnc_conversion* **__out_p_conversion, size_t *__p_available_space, void *__space, *cnc_conversion_info* *__p_info)

Opens a new encoding in the provided space an returns it through `__out_p_conversion`.

Remark

If there is a conversion that can be achieved by using an intermediate encoding (e.g., converting from the desired `__from` encoding to UTF-32, then from UTF-32 to the desired `__to` encoding), then an indirect transcode will

be opened if a direct encoding cannot be opened. If there is not enough space to write out, then this function will provide the amount needed in `__p_available_space` directly. Otherwise, it will decrement the value pointed to be `__p_available_space` by the amount of space used.

Parameters

- **`__registry`** – **[in]** The registry to use for opening the cnc_conversion handle.
- **`__from`** – **[in]** A pointer to data encoded as UTF-8 representing the encoding to transcode from.
- **`__to`** – **[in]** A pointer to data encoded as UTF-8 representing the encoding to transcode to.
- **`__selection`** – **[in]** A function pointer to an indirect selection function.
- **`__out_p_conversion`** – **[inout]** A pointer to the cnc_conversion handle to open.
- **`__p_available_space`** – **[inout]** The amount of space available in the `__space` parameter, in the number of bytes, that can be used to allocate any necessary data for the cnc_conversion handle.
- **`__space`** – **[inout]** A pointer to space which can be used for opening up the conversion handle, which has the size indicated by `__p_available_space`.
- **`__p_info`** – **[inout]** A pointer to an already-created *cnc_conversion_info* that will be filled out with information regarding how the transcoding operation was opened, if it was successful.

cnc_open_error **cnc_conv_open_n_select**(*cnc_conversion_registry* *__registry, size_t __from_size, const char __from[ZTD_PTR_EXTENT(*__from_size*)], size_t __to_size, const char __to[ZTD_PTR_EXTENT(*__to_size*)], cnc_indirect_selection_function *__selection, *cnc_conversion* **__out_p_conversion, size_t *__p_available_space, void *__space, *cnc_conversion_info* *__p_info)

Opens a new encoding in the provided space an returns it through `__out_p_conversion`.

Remark

If there is a conversion that can be achieved by using an intermediate encoding (e.g., converting from the desired `__from` encoding to UTF-32, then from UTF-32 to the desired `__to` encoding), then an indirect transcode will be opened if a direct encoding cannot be opened. If there is not enough space to write out, then this function will provide the amount needed in `__p_available_space` directly. Otherwise, it will decrement the value pointed to be `__p_available_space` by the amount of space used.

Parameters

- **`__registry`** – **[in]** The registry to use for opening the cnc_conversion handle.
- **`__from_size`** – **[in]** The size of the `__from` string.
- **`__from`** – **[in]** A pointer to data encoded as UTF-8 representing the encoding to transcode from.
- **`__to_size`** – **[in]** The size of the `__to` string.
- **`__to`** – **[in]** A pointer to data encoded as UTF-8 representing the encoding to transcode to.

- **__selection** – **[in]** A function pointer to an indirect selection function.
- **__out_p_conversion** – **[inout]** A pointer to the cnc_conversion handle to open.
- **__p_available_space** – **[inout]** The amount of space available in the `__space` parameter, in the number of bytes, that can be used to allocate any necessary data for the cnc_conversion handle.
- **__space** – **[inout]** A pointer to space which can be used for opening up the conversion handle, which has the size indicated by `__p_available_space`.
- **__p_info** – **[inout]** A pointer to an already-created *cnc_conversion_info* that will be filled out with information regarding how the transcoding operation was opened, if it was successful.

cnc_open_error **cnc_conv_new**(*cnc_conversion_registry* *__registry, const char *__from, const char *__to, *cnc_conversion* **__out_p_conversion, *cnc_conversion_info* *__p_info)

Creates a new encoding using the heap provided to the `__registry`.

Remark

This call defers to calling cnc_conv_new_n after computing the length of the `__from` and `__to` parameters, if they are not `nullptr`. If either is `nullptr`, their size is assumed to be 0.

Parameters

- **__registry** – **[in]** The registry to use for opening the cnc_conversion handle.
- **__from** – **[in]** A pointer to data encoded as UTF-8 representing the encoding to transcode from.
- **__to** – **[in]** A pointer to data encoded as UTF-8 representing the encoding to transcode to.
- **__out_p_conversion** – **[inout]** A pointer to the cnc_conversion handle to open.
- **__p_info** – **[inout]** A pointer to an already-created *cnc_conversion_info* that will be filled out with information regarding how the transcoding operation was opened, if it was successful.

cnc_open_error **cnc_conv_new_n**(*cnc_conversion_registry* *__registry, size_t __from_size, const char __from[ZTD_PTR_EXTENT(*__from_size*)], size_t __to_size, const char __to[ZTD_PTR_EXTENT(*__to_size*)], *cnc_conversion* **__out_p_conversion, *cnc_conversion_info* *__p_info)

Creates a new encoding using the heap provided to the `__registry`.

Remark

This call defers to calling cnc_conv_open_n after computing the necessary size from cnc_conv_open_n.

Parameters

- **__registry** – **[in]** The registry to use for opening the cnc_conversion handle.
- **__from_size** – **[in]** The size of the `__from` string.

- **__from** – **[in]** A pointer to data encoded as UTF-8 representing the encoding to transcode from.
- **__to_size** – **[in]** The size of the __to string.
- **__to** – **[in]** A pointer to data encoded as UTF-8 representing the encoding to transcode to.
- **__out_p_conversion** – **[inout]** A pointer to the cnc_conversion handle to open.
- **__p_info** – **[inout]** A pointer to an already-created *cnc_conversion_info* that will be filled out with information regarding how the transcoding operation was opened, if it was successful.

cnc_open_error **cnc_conv_new_select**(*cnc_conversion_registry* *__registry, const char *__from, const char *__to, cnc_indirect_selection_function *__selection, *cnc_conversion* **__out_p_conversion, *cnc_conversion_info* *__p_info)

Creates a new encoding using the heap provided to the __registry.

Remark

This call defers to calling cnc_conv_open_n after computing the necessary size from cnc_conv_open_n.

Parameters

- **__registry** – **[in]** The registry to use for opening the cnc_conversion handle.
- **__from_size** – **[in]** The size of the __from string.
- **__from** – **[in]** A pointer to data encoded as UTF-8 representing the encoding to transcode from.
- **__to_size** – **[in]** The size of the __to string.
- **__to** – **[in]** A pointer to data encoded as UTF-8 representing the encoding to transcode to.
- **__selection** – **[in]** A function pointer to a indirect selection function.
- **__out_p_conversion** – **[inout]** A pointer to the cnc_conversion handle to open.
- **__p_info** – **[inout]** A pointer to an already-created *cnc_conversion_info* that will be filled out with information regarding how the transcoding operation was opened, if it was successful.

cnc_open_error **cnc_conv_new_n_select**(*cnc_conversion_registry* *__registry, size_t __from_size, const char __from[ZTD_PTR_EXTENT(*__from_size*)], size_t __to_size, const char __to[ZTD_PTR_EXTENT(*__to_size*)], cnc_indirect_selection_function *__selection, *cnc_conversion* **__out_p_conversion, *cnc_conversion_info* *__p_info)

Creates a new encoding using the heap provided to the __registry.

Remark

This call defers to calling cnc_conv_open_n after computing the necessary size from cnc_conv_open_n.

Parameters

- **__registry** – **[in]** The registry to use for opening the cnc_conversion handle.
- **__from_size** – **[in]** The size of the `__from` string.
- **__from** – **[in]** A pointer to data encoded as UTF-8 representing the encoding to transcode from.
- **__to_size** – **[in]** The size of the `__to` string.
- **__to** – **[in]** A pointer to data encoded as UTF-8 representing the encoding to transcode to.
- **__selection** – **[in]** A function pointer to a indirect selection function.
- **__out_p_conversion** – **[inout]** A pointer to the cnc_conversion handle to open.
- **__p_info** – **[inout]** A pointer to an already-created *cnc_conversion_info* that will be filled out with information regarding how the transcoding operation was opened, if it was successful.

cnc_open_error **cnc_conv_open_c8**(*cnc_conversion_registry* *__registry, const ztd_char8_t *__from, const ztd_char8_t *__to, *cnc_conversion* **__out_p_conversion, size_t *__p_available_space, unsigned char *__space, *cnc_conversion_info* *__p_info)

Opens a new encoding in the provided space.

Remark

This call defers to calling cnc_conv_open_n after computing the length of the `__from` and `__to` parameters, if they are not `nullptr`. If either is `nullptr`, their size is assumed to be 0.

Parameters

- **__registry** – **[in]** The registry to use for opening the cnc_conversion handle.
- **__from** – **[in]** A pointer to data encoded as UTF-8 representing the encoding to transcode from.
- **__to** – **[in]** A pointer to data encoded as UTF-8 representing the encoding to transcode to.
- **__out_p_conversion** – **[inout]** A pointer to the cnc_conversion handle to open.
- **__p_available_space** – **[inout]** The amount of space available in the `__space` parameter, in the number of bytes, that can be used to allocate any necessary data for the cnc_conversion handle.
- **__space** – **[inout]** A pointer to space which can be used for opening up the conversion handle, which has the size indicated by `__p_available_space`.
- **__p_info** – **[inout]** A pointer to an already-created *cnc_conversion_info* that will be filled out with information regarding how the transcoding operation was opened, if it was successful.

cnc_open_error **cnc_conv_open_c8n**(*cnc_conversion_registry* *__registry, size_t __from_size, const ztd_char8_t __from[ZTD_PTR_EXTENT(*__from_size*)], size_t __to_size, const ztd_char8_t __to[ZTD_PTR_EXTENT(*__to_size*)], *cnc_conversion* **__out_p_conversion, size_t *__p_available_space, void *__space, *cnc_conversion_info* *__p_info)

Opens a new encoding in the provided space an returns it through `__out_p_conversion`.

Remark

If there is a conversion that can be achieved by using an intermediate encoding (e.g., converting from the desired `__from` encoding to UTF-32, then from UTF-32 to the desired `__to` encoding), then an indirect transcode will be opened if a direct encoding cannot be opened. If there is not enough space to write out, then this function will provide the amount needed in `__p_available_space` directly. Otherwise, it will decrement the value pointed to be `__p_available_space` by the amount of space used.

Parameters

- **`__registry`** – **[in]** The registry to use for opening the cnc_conversion handle.
- **`__from_size`** – **[in]** The size of the `__from` string.
- **`__from`** – **[in]** A pointer to data encoded as UTF-8 representing the encoding to transcode from.
- **`__to_size`** – **[in]** The size of the `__to` string.
- **`__to`** – **[in]** A pointer to data encoded as UTF-8 representing the encoding to transcode to.
- **`__out_p_conversion`** – **[inout]** A pointer to the cnc_conversion handle to open.
- **`__p_available_space`** – **[inout]** The amount of space available in the `__space` parameter, in the number of bytes, that can be used to allocate any necessary data for the cnc_conversion handle.
- **`__space`** – **[inout]** A pointer to space which can be used for opening up the conversion handle, which has the size indicated by `__p_available_space`.
- **`__p_info`** – **[inout]** A pointer to an already-created *cnc_conversion_info* that will be filled out with information regarding how the transcoding operation was opened, if it was successful.

cnc_open_error **cnc_conv_open_c8_select**(*cnc_conversion_registry* *__registry, const ztd_char8_t *__from, const ztd_char8_t *__to, cnc_indirect_selection_c8_function *__selection, *cnc_conversion* **__out_p_conversion, size_t *__p_available_space, void *__space, *cnc_conversion_info* *__p_info)

Opens a new encoding in the provided space an returns it through `__out_p_conversion`.

Remark

If there is a conversion that can be achieved by using an intermediate encoding (e.g., converting from the desired `__from` encoding to UTF-32, then from UTF-32 to the desired `__to` encoding), then an indirect transcode will be opened if a direct encoding cannot be opened. If there is not enough space to write out, then this function will provide the amount needed in `__p_available_space` directly. Otherwise, it will decrement the value pointed to be `__p_available_space` by the amount of space used.

Parameters

- **`__registry`** – **[in]** The registry to use for opening the cnc_conversion handle.

- **__from** – **[in]** A pointer to data encoded as UTF-8 representing the encoding to transcode from.
- **__to** – **[in]** A pointer to data encoded as UTF-8 representing the encoding to transcode to.
- **__selection** – **[in]** A function pointer to an indirect selection function.
- **__out_p_conversion** – **[inout]** A pointer to the cnc_conversion handle to open.
- **__p_available_space** – **[inout]** The amount of space available in the `__space` parameter, in the number of bytes, that can be used to allocate any necessary data for the cnc_conversion handle.
- **__space** – **[inout]** A pointer to space which can be used for opening up the conversion handle, which has the size indicated by `__p_available_space`.
- **__p_info** – **[inout]** A pointer to an already-created *cnc_conversion_info* that will be filled out with information regarding how the transcoding operation was opened, if it was successful.

cnc_open_error **cnc_conv_open_c8n_select**(*cnc_conversion_registry* *__registry, size_t __from_size, const ztd_char8_t __from[ZTD_PTR_EXTENT(*__from_size*)], size_t __to_size, const ztd_char8_t __to[ZTD_PTR_EXTENT(*__to_size*)], cnc_indirect_selection_c8_function *__selection, *cnc_conversion* **__out_p_conversion, size_t *__p_available_space, void *__space, *cnc_conversion_info* *__p_info)

Opens a new encoding in the provided space an returns it through `__out_p_conversion`.

Remark

If there is a conversion that can be achieved by using an intermediate encoding (e.g., converting from the desired `__from` encoding to UTF-32, then from UTF-32 to the desired `__to` encoding), then an indirect transcode will be opened if a direct encoding cannot be opened. If there is not enough space to write out, then this function will provide the amount needed in `__p_available_space` directly. Otherwise, it will decrement the value pointed to be `__p_available_space` by the amount of space used.

Parameters

- **__registry** – **[in]** The registry to use for opening the cnc_conversion handle.
- **__from_size** – **[in]** The size of the `__from` string.
- **__from** – **[in]** A pointer to data encoded as UTF-8 representing the encoding to transcode from.
- **__to_size** – **[in]** The size of the `__to` string.
- **__to** – **[in]** A pointer to data encoded as UTF-8 representing the encoding to transcode to.
- **__selection** – **[in]** A function pointer to an indirect selection function.
- **__out_p_conversion** – **[inout]** A pointer to the cnc_conversion handle to open.
- **__p_available_space** – **[inout]** The amount of space available in the `__space` parameter, in the number of bytes, that can be used to allocate any necessary data for the cnc_conversion handle.

- **__space** – **[inout]** A pointer to space which can be used for opening up the conversion handle, which has the size indicated by __p_available_space.
- **__p_info** – **[inout]** A pointer to an already-created *cnc_conversion_info* that will be filled out with information regarding how the transcoding operation was opened, if it was successful.

cnc_open_error **cnc_conv_new_c8**(*cnc_conversion_registry* *__registry, const ztd_char8_t *__from, const ztd_char8_t *__to, *cnc_conversion* **__out_p_conversion, *cnc_conversion_info* *__p_info)

Creates a new encoding using the heap provided to the __registry.

Remark

This call defers to calling cnc_conv_new_n after computing the length of the __from and __to parameters, if they are not nullptr. If either is nullptr, their size is assumed to be 0.

Parameters

- **__registry** – **[in]** The registry to use for opening the cnc_conversion handle.
- **__from** – **[in]** A pointer to data encoded as UTF-8 representing the encoding to transcode from.
- **__to** – **[in]** A pointer to data encoded as UTF-8 representing the encoding to transcode to.
- **__out_p_conversion** – **[inout]** A pointer to the cnc_conversion handle to open.
- **__p_info** – **[inout]** A pointer to an already-created *cnc_conversion_info* that will be filled out with information regarding how the transcoding operation was opened, if it was successful.

cnc_open_error **cnc_conv_new_c8n**(*cnc_conversion_registry* *__registry, size_t __from_size, const ztd_char8_t __from[ZTD_PTR_EXTENT(*__from_size*)], size_t __to_size, const ztd_char8_t __to[ZTD_PTR_EXTENT(*__to_size*)], *cnc_conversion* **__out_p_conversion, *cnc_conversion_info* *__p_info)

Creates a new encoding using the heap provided to the __registry.

Remark

This call defers to calling cnc_conv_open_n after computing the necessary size from cnc_conv_open_n.

Parameters

- **__registry** – **[in]** The registry to use for opening the cnc_conversion handle.
- **__from_size** – **[in]** The size of the __from string.
- **__from** – **[in]** A pointer to data encoded as UTF-8 representing the encoding to transcode from.
- **__to_size** – **[in]** The size of the __to string.
- **__to** – **[in]** A pointer to data encoded as UTF-8 representing the encoding to transcode to.

- **__out_p_conversion** – **[inout]** A pointer to the cnc_conversion handle to open.
- **__p_info** – **[inout]** A pointer to an already-created *cnc_conversion_info* that will be filled out with information regarding how the transcoding operation was opened, if it was successful.

cnc_open_error **cnc_conv_new_c8_select**(*cnc_conversion_registry* *__registry, const ztd_char8_t *__from, const ztd_char8_t *__to, cnc_indirect_selection_c8_function *__selection, *cnc_conversion* **__out_p_conversion, *cnc_conversion_info* *__p_info)

Creates a new encoding using the heap provided to the `__registry`.

Remark

This call defers to calling cnc_conv_open_n after computing the necessary size from cnc_conv_open_n.

Parameters

- **__registry** – **[in]** The registry to use for opening the cnc_conversion handle.
- **__from_size** – **[in]** The size of the `__from` string.
- **__from** – **[in]** A pointer to data encoded as UTF-8 representing the encoding to transcode from.
- **__to_size** – **[in]** The size of the `__to` string.
- **__to** – **[in]** A pointer to data encoded as UTF-8 representing the encoding to transcode to.
- **__selection** – **[in]** A function pointer to a indirect selection function.
- **__out_p_conversion** – **[inout]** A pointer to the cnc_conversion handle to open.
- **__p_info** – **[inout]** A pointer to an already-created *cnc_conversion_info* that will be filled out with information regarding how the transcoding operation was opened, if it was successful.

cnc_open_error **cnc_conv_new_c8n_select**(*cnc_conversion_registry* *__registry, size_t __from_size, const ztd_char8_t __from[ZTD_PTR_EXTENT(*__from_size*)], size_t __to_size, const ztd_char8_t __to[ZTD_PTR_EXTENT(*__to_size*)], cnc_indirect_selection_c8_function *__selection, *cnc_conversion* **__out_p_conversion, *cnc_conversion_info* *__p_info)

Creates a new encoding using the heap provided to the `__registry`.

Remark

This call defers to calling cnc_conv_open_n after computing the necessary size from cnc_conv_open_n.

Parameters

- **__registry** – **[in]** The registry to use for opening the cnc_conversion handle.
- **__from_size** – **[in]** The size of the `__from` string.

- **__from** – **[in]** A pointer to data encoded as UTF-8 representing the encoding to transcode from.
- **__to_size** – **[in]** The size of the `__to` string.
- **__to** – **[in]** A pointer to data encoded as UTF-8 representing the encoding to transcode to.
- **__selection** – **[in]** A function pointer to a indirect selection function.
- **__out_p_conversion** – **[inout]** A pointer to the cnc_conversion handle to open.
- **__p_info** – **[inout]** A pointer to an already-created *cnc_conversion_info* that will be filled out with information regarding how the transcoding operation was opened, if it was successful.

void **cnc_conv_close**(*cnc_conversion* *__conversion)

Closes (destroys) the data used by the cnc_conversion handle pointed to by `__conversion`.

Remark

This function, to use a C++ analogy, behaves much like a destructor. It does not free any memory: it simply destroys anything created or used by the cnc_open_function supplied when registrying the "from" and "to" conversion pair. If `__conversion` is `nullptr`, this function does nothing.

Parameters **__conversion** – **[in]** The cnc_conversion handle to destroy. Can be `nullptr`.

void **cnc_conv_delete**(*cnc_conversion* *__conversion)

Deletes and data used by the cnc_conversion handle pointed to by `__conversion`.

Remark

This function will call `cnc_conv_close` on the `__conversion` function, and then delete the memory. It must not be used if cnc_conv_new or cnc_conv_new_n was not used. If `__conversion` is `nullptr`, this function does nothing.

Parameters **__conversion** – **[in]** The cnc_conversion handle to destroy. Can be `nullptr`.

Registry-Based Conversions

There are two categories of conversion, as discussed in the *design*. Single conversion, and bulk conversion. The single conversions only do one unit of *indivisible work*. The bulk functions repeatedly perform *one unit of indivisible work*.

cnc_conversion Conversion Functions

cnc_mcerror **cnc_conv_pivot**(*cnc_conversion* *__conversion, size_t *__out_pput_bytes_size, unsigned char **__out_pput_bytes, size_t *__p_input_bytes_size, const unsigned char **__p_input_bytes, cnc_pivot_info *__p_pivot_info)

Converts a series of bytes in one encoding scheme to the other encoding scheme using the specified `__conversion` format.

Remark

The conversion functions take parameters as output parameters (pointers) so that they can provide information about how much of the input and output is used. Providing a `nullptr` for both `__p_ouput_bytes_size` and `__out_pput_bytes` serves as a way to validate the input. Providing only `__out_pput_bytes` but not `__out_pput_bytes_size` is a way to indicate that the output space is sufficiently large for the input space. Providing `__out_pput_bytes_size` but not `__out_pput_bytes` is a way to determine how much data will be written out for a given input without actually performing such a write. The pivot buffer is used when the conversion cannot be done directly (which is specified through the *cnc_conversion_info* structure returned from opening a conversion routine). If the pivot buffer does not point to a null / empty buffer, and it ends up being too small for the given conversion, it may produce spurious CNC_MCERROR_INSUFFICIENT_OUTPUT errors unrelated to the actual `__out_pput_bytes` buffer passed into the function.

Parameters

- **__conversion** – **[in]** The cnc_conversion handle indicating the format to be used. Shall not be `nullptr`.
- **__out_pput_bytes_size** – **[inout]** A pointer to the size of the output buffer. If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__out_pput_bytes` is not `nullptr`).
- **__out_pput_bytes** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__out_pput_bytes_size`).
- **__p_input_bytes_size** – **[inout]** A pointer to the size of the input buffer. If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_input_bytes** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_pivot_info** – **[inout]** A pointer to a pivot buffer and return error code. If the return value of this function is not CNC_MCERROR_OK, the pivot information is not NULL, and the error was caused by the intermediate conversion step failing, then the `error` member of cnc_pivot_info will be set to the error value that took place.

cnc_mcerror **cnc_conv**(*cnc_conversion* *__conversion, size_t *__out_pput_bytes_size, unsigned char **__out_pput_bytes, size_t *__p_input_bytes_size, const unsigned char **__p_input_bytes)

Converts a series of bytes in one encoding scheme to the other encoding scheme using the specified `__conversion` format.

Remark

The conversion functions take parameters as output parameters (pointers) so that they can provide information about how much of the input and output is used. Providing a `nullptr` for both `__p_ouput_bytes_size` and `__out_pput_bytes` serves as a way to validate the input. Providing only `__out_pput_bytes` but not `__out_pput_bytes_size` is a way to indicate that the output space is sufficiently large for the input space. Providing `__out_pput_bytes_size` but not `__out_pput_bytes` is a way to determine how much data will be written out for a given input without actually performing such a write.

Parameters

- **__conversion** – **[in]** The cnc_conversion handle indicating the format to be used. Shall not be `nullptr`.
- **__out_pput_bytes_size** – **[inout]** A pointer to the size of the output buffer. If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__out_pput_bytes` is not `nullptr`).
- **__out_pput_bytes** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__out_pput_bytes_size`).
- **__p_input_bytes_size** – **[inout]** A pointer to the size of the input buffer. If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_input_bytes** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_conv_count_pivot**(*cnc_conversion* *__conversion, size_t *__out_pput_bytes_size, size_t *__p_input_bytes_size, const unsigned char **__p_input_bytes, cnc_pivot_info *__p_pivot_info)

Counts the total number of bytes that can be successfully converted until an error occurs for the specified `__conversion` format.

Remark

This function is an ease-of-use shortcut for calling cnc_conv with the `__out_pput_bytes` argument sent to `nullptr`.

Parameters

- **__conversion** – **[in]** The cnc_conversion handle indicating the format to be used. Shall not be `nullptr`.
- **__out_pput_bytes_size** – **[inout]** A pointer to the size of the output buffer. If this is `nullptr`, then it will not update the count.
- **__p_input_bytes_size** – **[inout]** A pointer to the size of the input buffer. If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.

- **__p_input_bytes** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_pivot_info** – **[inout]** A pointer to a pivot buffer and return error code. If the return value of this function is not CNC_MCERROR_OK, the pivot information is not NULL, and the error was caused by the intermediate conversion step failing, then the `error` member of cnc_pivot_info will be set to the error value that took place.

cnc_mcerror **cnc_conv_count**(*cnc_conversion* *__conversion, size_t *__out_pput_bytes_size, size_t *__p_input_bytes_size, const unsigned char **__p_input_bytes)

Counts the total number of bytes that can be successfully converted until an error occurs for the specified `__conversion` format.

Remark

This function is an ease-of-use shortcut for calling cnc_conv with the `__out_pput_bytes` argument sent to `nullptr`.

Parameters

- **__conversion** – **[in]** The cnc_conversion handle indicating the format to be used. Shall not be `nullptr`.
- **__out_pput_bytes_size** – **[inout]** A pointer to the size of the output buffer. If this is `nullptr`, then it will not update the count.
- **__p_input_bytes_size** – **[inout]** A pointer to the size of the input buffer. If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_input_bytes** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

bool **cnc_conv_is_valid_pivot**(*cnc_conversion* *__conversion, size_t *__p_input_bytes_size, const unsigned char **__p_input_bytes, cnc_pivot_info *__p_pivot_info)

Checks whether or not the input can be successfully converted according to the format of `__conversion`.

Remark

This function is an ease-of-use shortcut for calling cnc_conv with the `__out_pput_bytes` argument sent to `nullptr`.

Parameters

- **__conversion** – **[in]** The cnc_conversion handle indicating the format to be used. Shall not be `nullptr`.
- **__p_input_bytes_size** – **[inout]** A pointer to the size of the input buffer. If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and `true` is returned.

- **__p_input_bytes** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and `true` is returned.
- **__p_pivot_info** – **[inout]** A pointer to a pivot buffer and return error code. If the return value of this function is not CNC_MCERROR_OK, the pivot information is not NULL, and the error was caused by the intermediate conversion step failing, then the `error` member of cnc_pivot_info will be set to the error value that took place.

bool **cnc_conv_is_valid**(*cnc_conversion* *__conversion, size_t *__p_input_bytes_size, const unsigned char **__p_input_bytes)

Checks whether or not the input can be successfully converted according to the format of `__conversion`.

Remark

This function is an ease-of-use shortcut for calling cnc_conv with the `__out_pput_bytes` argument sent to `nullptr`.

Parameters

- **__conversion** – **[in]** The cnc_conversion handle indicating the format to be used. Shall not be `nullptr`.
- **__p_input_bytes_size** – **[inout]** A pointer to the size of the input buffer. If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and `true` is returned.
- **__p_input_bytes** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and `true` is returned.

cnc_mcerror **cnc_conv_unbounded_pivot**(*cnc_conversion* *__conversion, unsigned char **__out_pput_bytes, size_t *__p_input_bytes_size, const unsigned char **__p_input_bytes, cnc_pivot_info *__p_pivot_info)

Converts a series of bytes in one encoding scheme to the other encoding scheme using the specified `__conversion` format.

Remark

The conversion functions take parameters as output parameters (pointers) so that they can provide information about how much of the input and output is used. Providing a `nullptr` for both `__p_ouput_bytes_size` and `__out_pput_bytes` serves as a way to validate the input. Providing only `__out_pput_bytes` but not `__out_pput_bytes_size` is a way to indicate that the output space is sufficiently large for the input space. Providing `__out_pput_bytes_size` but not `__out_pput_bytes` is a way to determine how much data will be written out for a given input without actually performing such a write.

Parameters

- **__conversion** – **[in]** The cnc_conversion handle indicating the format to be used. Shall not be `nullptr`.
- **__out_pput_bytes** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__out_pput_bytes_size`).

- **__p_input_bytes_size** – **[inout]** A pointer to the size of the input buffer. If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_input_bytes** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_pivot_info** – **[inout]** A pointer to a pivot buffer and return error code. If the return value of this function is not CNC_MCERROR_OK, the pivot information is not NULL, and the error was caused by the intermediate conversion step failing, then the `error` member of cnc_pivot_info will be set to the error value that took place.

cnc_mcerror **cnc_conv_unbounded**(*cnc_conversion* *__conversion, unsigned char **__out_pput_bytes, size_t *__p_input_bytes_size, const unsigned char **__p_input_bytes)

Converts a series of bytes in one encoding scheme to the other encoding scheme using the specified `__conversion` format.

Remark

The conversion functions take parameters as output parameters (pointers) so that they can provide information about how much of the input and output is used. Providing a `nullptr` for both `__p_ouput_bytes_size` and `__out_pput_bytes` serves as a way to validate the input. Providing only `__out_pput_bytes` but not `__out_pput_bytes_size` is a way to indicate that the output space is sufficiently large for the input space. Providing `__out_pput_bytes_size` but not `__out_pput_bytes` is a way to determine how much data will be written out for a given input without actually performing such a write.

Parameters

- **__conversion** – **[in]** The cnc_conversion handle indicating the format to be used. Shall not be `nullptr`.
- **__out_pput_bytes** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__out_pput_bytes_size`).
- **__p_input_bytes_size** – **[inout]** A pointer to the size of the input buffer. If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_input_bytes** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_conv_one_pivot**(*cnc_conversion* *__conversion, size_t *__out_pput_bytes_size, unsigned char **__out_pput_bytes, size_t *__p_input_bytes_size, const unsigned char **__p_input_bytes, cnc_pivot_info *__p_pivot_info)

Peforms at least one complete unit of work on the input and produces one complete unit of work into the output according to the format of `__conversion`.

Remark

This function only performs exactly one complete unit of work for the input and the output: nothing more. The conversion functions take parameters as output parameters (pointers) so that they can provide information about how much of the input and output is used. Providing a `nullptr` for both `__p_ouput_bytes_size` and `__out_pput_bytes` serves as a way to validate the input for one completely unit of work. Providing only `__out_pput_bytes` but not `__out_pput_bytes_size` is a way to indicate that the output space is sufficiently large for the input space. Providing `__out_pput_bytes_size` but not `__out_pput_bytes` is a way to determine how much data will be written out for a given input without actually performing such a write.

Parameters

- **`__conversion`** – **[in]** The cnc_conversion handle indicating the format to be used. Shall not be `nullptr`.
- **`__out_pput_bytes_size`** – **[inout]** A pointer to the size of the output buffer. If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__out_pput_bytes` is not `nullptr`).
- **`__out_pput_bytes`** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__out_pput_bytes_size`).
- **`__p_input_bytes_size`** – **[inout]** A pointer to the size of the input buffer. If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **`__p_input_bytes`** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **`__p_pivot_info`** – **[inout]** A pointer to a pivot buffer and return error code. If the return value of this function is not CNC_MCERROR_OK, the pivot information is not NULL, and the error was caused by the intermediate conversion step failing, then the `error` member of cnc_pivot_info will be set to the error value that took place.

cnc_mcerror **`cnc_conv_one`**(*cnc_conversion* *__conversion, size_t *__out_pput_bytes_size, unsigned char **__out_pput_bytes, size_t *__p_input_bytes_size, const unsigned char **__p_input_bytes)

Peforms at least one complete unit of work on the input and produces one complete unit of work into the output according to the format of `__conversion`.

Remark

This function only performs exactly one complete unit of work for the input and the output: nothing more. The conversion functions take parameters as output parameters (pointers) so that they can provide information about how much of the input and output is used. Providing a `nullptr` for both `__p_ouput_bytes_size` and `__out_pput_bytes` serves as a way to validate the input for one completely unit of work. Providing only `__out_pput_bytes` but not `__out_pput_bytes_size` is a way to indicate that the output space is sufficiently large for the input space. Providing `__out_pput_bytes_size` but not `__out_pput_bytes` is a way to determine how much data will be written out for a given input without actually performing such a write.

Parameters

- **`__conversion`** – **[in]** The cnc_conversion handle indicating the format to be used. Shall not be `nullptr`.

- **__out_pput_bytes_size** – **[inout]** A pointer to the size of the output buffer. If this is `nullptr`, then it will not update the count (and the output stream will automatically be considered large enough to handle all data, if `__out_pput_bytes` is not `nullptr`).
- **__out_pput_bytes** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__out_pput_bytes_size`).
- **__p_input_bytes_size** – **[inout]** A pointer to the size of the input buffer. If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_input_bytes** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

cnc_mcerror **cnc_conv_one_count_pivot**(*cnc_conversion* *__conversion, size_t *__out_pput_bytes_size, size_t *__p_input_bytes_size, const unsigned char **__p_input_bytes, cnc_pivot_info *__p_pivot_info)

Counts the total number of bytes that can be successfully converted for one complete unit of input, or if an error occurs, for the specified `__conversion` format.

Remark

This function is an ease-of-use shortcut for calling cnc_conv_one with the `__out_pput_bytes` argument sent to `nullptr`.

Parameters

- **__conversion** – **[in]** The cnc_conversion handle indicating the format to be used. Shall not be `nullptr`.
- **__out_pput_bytes_size** – **[inout]** A pointer to the size of the output buffer. If this is `nullptr`, then it will not update the count.
- **__p_input_bytes_size** – **[inout]** A pointer to the size of the input buffer. If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_input_bytes** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_pivot_info** – **[inout]** A pointer to a pivot buffer and return error code. If the return value of this function is not CNC_MCERROR_OK, the pivot information is not NULL, and the error was caused by the intermediate conversion step failing, then the `error` member of cnc_pivot_info will be set to the error value that took place.

cnc_mcerror **cnc_conv_one_count**(*cnc_conversion* *__conversion, size_t *__out_pput_bytes_size, size_t *__p_input_bytes_size, const unsigned char **__p_input_bytes)

Counts the total number of bytes that can be successfully converted for one complete unit of input, or if an error occurs, for the specified `__conversion` format.

Remark

This function is an ease-of-use shortcut for calling cnc_conv_one with the `__out_pput_bytes` argument sent to `nullptr`.

Parameters

- **__conversion** – **[in]** The cnc_conversion handle indicating the format to be used. Shall not be `nullptr`.
- **__out_pput_bytes_size** – **[inout]** A pointer to the size of the output buffer. If this is `nullptr`, then it will not update the count.
- **__p_input_bytes_size** – **[inout]** A pointer to the size of the input buffer. If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and CNC_MCERROR_OK is returned.
- **__p_input_bytes** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and CNC_MCERROR_OK is returned.

bool **cnc_conv_one_is_valid_pivot**(*cnc_conversion* *__conversion, size_t *__p_input_bytes_size, const unsigned char **__p_input_bytes, cnc_pivot_info *__p_pivot_info)

Checks whether or not one complete unit of input can be successfully converted according to the format of `__conversion` to one complete unit of output.

Remark

This function is an ease-of-use shortcut for calling cnc_conv_one with the `__out_pput_bytes` argument and the `__out_pput_byte_size` argument set to `nullptr` sent to `nullptr`.

Parameters

- **__conversion** – **[in]** The cnc_conversion handle indicating the format to be used. Shall not be `nullptr`.
- **__p_input_bytes_size** – **[inout]** A pointer to the size of the input buffer. If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and `true` is returned.
- **__p_input_bytes** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and `true` is returned.
- **__p_pivot_info** – **[inout]** A pointer to a pivot buffer and return error code. If the return value of this function is not CNC_MCERROR_OK, the pivot information is not NULL, and the error was caused by the intermediate conversion step failing, then the `error` member of cnc_pivot_info will be set to the error value that took place.

bool **cnc_conv_one_is_valid**(*cnc_conversion* *__conversion, size_t *__p_input_bytes_size, const unsigned char **__p_input_bytes)

Checks whether or not one complete unit of input can be successfully converted according to the format of `__conversion` to one complete unit of output.

Remark

This function is an ease-of-use shortcut for calling cnc_conv_one with the `__out_pput_bytes` argument and the `__out_pput_byte_size` argument set to `nullptr` sent to `nullptr`.

Parameters

- **__conversion** – **[in]** The cnc_conversion handle indicating the format to be used. Shall not be `nullptr`.
- **__p_input_bytes_size** – **[inout]** A pointer to the size of the input buffer. If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and `true` is returned.
- **__p_input_bytes** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and `true` is returned.

cnc_mcerror **cnc_conv_one_unbounded_pivot**(*cnc_conversion* *__conversion, unsigned char **__out_pput_bytes, size_t *__p_input_bytes_size, const unsigned char **__p_input_bytes, cnc_pivot_info *__p_pivot_info)

Performs a conversion to the specified sequences, assuming that there is an appropriately sized buffer that will fit all of the output, no matter what.

Remark

This function is an ease-of-use shortcut for calling cnc_conv_one with the `__out_pput_bytes` argument and the `__out_pput_byte_size` argument set to `nullptr` sent to `nullptr`.

Parameters

- **__conversion** – **[in]** The cnc_conversion handle indicating the format to be used. Shall not be `nullptr`.
- **__out_pput_bytes** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__out_pput_bytes_size`).
- **__p_input_bytes_size** – **[inout]** A pointer to the size of the input buffer. If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and `true` is returned.
- **__p_input_bytes** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and `true` is returned.
- **__p_pivot_info** – **[inout]** A pointer to a pivot buffer and return error code. If the return value of this function is not CNC_MCERROR_OK, the pivot information is not NULL, and the error was caused by the intermediate conversion step failing, then the `error` member of cnc_pivot_info will be set to the error value that took place.

cnc_mcerror **cnc_conv_one_unbounded**(*cnc_conversion* *__conversion, unsigned char **__out_pput_bytes, size_t *__p_input_bytes_size, const unsigned char **__p_input_bytes)

Performs a conversion to the specified sequences, assuming that there is an appropriately sized buffer that will fit all of the output, no matter what.

Remark

This function is an ease-of-use shortcut for calling cnc_conv_one with the `__out_pput_bytes` argument and the `__out_pput_byte_size` argument set to `nullptr` sent to `nullptr`.

Parameters

- **__conversion** – **[in]** The cnc_conversion handle indicating the format to be used. Shall not be `nullptr`.
- **__out_pput_bytes** – **[inout]** A pointer to the pointer of the output buffer. If this or the pointer within are `nullptr`, than this function will not write output data (it may still decrement the value pointed to by `__out_pput_bytes_size`).
- **__p_input_bytes_size** – **[inout]** A pointer to the size of the input buffer. If this is `nullptr` or points to a value equivalent to `0`, then the input is considered empty and `true` is returned.
- **__p_input_bytes** – **[inout]** A pointer to the pointer of the input buffer. If this or the pointer within are `nullptr`, than the input is considered empty and `true` is returned.

cnc_conversion_heap

struct **cnc_conversion_heap**

The conversion heap through which all allocating and deallocating happens, as well as any related actions that require dynamic allocation.

Public Members

void ***user_data**

The userdata to be passed alng to the heap functions.

cnc_allocate_function ***allocate**

The allocation function. It takes a userdata passed in when creating the heap, and writes out the actual size of the returned allocated pointer.

cnc_reallocate_function ***reallocate**

The reallocation function. It takes the original pointer and the requested size alongside a userdata passed in when creating the heap, and writes out the actual size of the returned allocated pointer. The original pointer cannot be used.

cnc_allocation_expand_function ***shrink**

The allocation expanding function. It takes the original allocation, the amount to expand the allocation by to its left (descending, lowest memory order) and right (ascending, highest memory order), the original pointer, and the original size alongside a userdata point. It returns the new pointer and writes out the new size.

cnc_allocation_shrink_function ***expand**

The allocation shrink function. It takes the original allocation, the amount to shrink the allocation by to its

left (descending, lowest memory order) and right (ascending, highest memory order), the original pointer, and the original size alongside a userdata point. It returns the new pointer and writes out the new size.

cnc_deallocate_function ***deallocate**
The allocation deallocate function. It takes the original pointer, its size, its alignment, and a userdata passed in during heap creation.

1.5 Glossary of Terms & Definitions

Occasionally, we may need to use precise language to describe what we want. This contains a list of definitions that can be linked to from the documentation to help describe key concepts that are useful for the explication of the concepts and ideas found in this documentation.

character This word carries with it 2 meanings, thanks to C-style languages and their predecessors. Sometimes, `char`s, `wchar_t`s, `char8_t`s, and similar are called "narrow character"s, "wide character"s, "UTF-8 characters" and similar. This is the result of a poor legacy in software and hardware nomenclature. These are not character types, but rather types that _may_ represent the abstract notion of a character but frequently, and often, do not. After all, you wouldn't be here reading this if it did and non-English wasn't busted in your application, now would you?

The other definition is just an abstract unit of information in human languages and writing. The closest approximation that Unicode has for the human language/writing character is a *Grapheme Cluster*.

code point A single unit of decoded information. Most typically associated with *unicode code points*, but they can be other things such as *unicode scalar values* or even a 13-bit value.

Note that a single code point does not imply a *"character"*, as that is a complex entity in human language and writing that cannot be mapped easily to a single unit of decoded information.

code unit A single unit of encoded information. This is typically, 8-, 16-, or 32-bit entites arranged in some sequential fashion that, when read or treated in a certain manner, end up composing higher-level units which make up readable text. Much of the world's most useful encodings that encode text use multiple code units in sequence to give a specific meaning to something, which makes most encodings variable length encodings.

complete unit of work A complete unit of work is when as little as is possible to form a complete set of output operations is consumed. This can result in 1 or more output *code units* or *code points*, or a transition in shift state (which consumes some of the inpuut but may output nothing). It guarantees forward progress in some fashion through either an output or a state change.

decode Converting from a stream of input, typically code units, to a stream of output, typically code points. The output is generally in a form that is more widely consummable or easier to process than when it started. Frequently, this library expects and works with the goal that any decoding process is producing *unicode code points* or *unicode scalar values* from some set of *code units*.

encode Converting from a stream of input, typically code points, to a stream of output, typically code units. The output may be less suitable for general interchange or consumption, or is in a specific interchange format for the interoperation. Frequently, this library expects and works with the goal that any decoding process is producing *unicode code points* or *unicode scalar values* from some set of *code units*.

encoding A set of functionality that includes an encode process or a decode process (or both). The encode process takes in a stream of code points and puts out a stream of code units. The decode process takes in a stream of code units and puts out a stream of code points.

execution encoding The locale-based encoding related to "multibyte characters" (C and C++ magic words) processed during program evaluation/execution. It is directly related to the `std::set_locale(LC_CTYPE, ...)` calls. Note that this is different from *literal encoding*, which is the encoding of string literals. The two may not be (and many times, are not) the same.

grapheme cluster The closest the Unicode Standard gets to recognizing a *human-readable and writable character*, grapheme cluster's are arbitrarily sized bundles of *unicode code points* that compose of a single concept that might match what a *"character"* is in any given human language.

indivisible unit of work A single unit of transcoding effort when going from one encoding to another that consumes the smallest possible input to produce an output, to change the state, to both produce an output and change the state, or to produce an error. Unlike *unicode code points* or *unicode scalar values*, indivisible units of work do not have a fixed width or fixed definition and are dependent on the two encodings involved in the transcoding operation being performed.

literal encoding The encoding of string literals ("") during constant evaluation. This is usually controlled by command line arguments (MSVC and GCC) or fixed during compilation (Clang as UTF-8, though that may change). Typically defaults to the system's "locale" setting.

mojibake (Japanese: Pronunciation: [modibake] "unintelligible sequence of characters".) From Japanese (moji), meaning "character" and (bake), meaning change, is an occurence of incorrect unreadable characters displayed when computer software fails to render text correctly to its associated character encoding.

transcode Converting from one form of encoded information to another form of encoded information. In the context of this library, it means going from an input in one *encoding*'s code units to an output of another encoding's code units. Typically, this is done by invoking the *decode* of the original encoding to reach a common interchange format (such as *unicode code points*) before taking that intermediate output and piping it through the *encode* step of the other encoding. Different transcode operations may not need to go through a common interchange, and may transcode "directly", as a way to improve space utilization, time spent, or both.

unicode code point A single unit of decoded information for Unicode. It represents the smallest, non-encoded, and indivisible piece of information that can be used to talk about higher level algorithms, properties, and more from the Unicode Standard.

A unicode code point has been reserved to take at most 21 bits of space to identify itself.

A single unicode code point is NOT equivalent to a *character*, and multiple of them can be put together or taken apart and still have their sequence form a *"character"*. For a more holistic, human-like interpretation of code points or other data, see *grapheme clusters*.

unicode scalar value A single unit of decoded information for Unicode. It's definition is identical to that of *unicode code points*, with the additional constraint that every unicode scalar value may not be a "Surrogate Value". Surrogate values are non-characters used exclusively for the purpose of encoding and decoding specific sequences of code units, and therefore carry no useful meaning in general interchange. They may appear in text streams in certain encodings.

UTF-16 The Unicode Transformation Format-16 encoding. It is the encoding of `u` (Lowecase Latin-`u`) string literals (`u""`).

UTF-32 The Unicode Transformation Format-32 encoding. It is the encoding of `U` (Upperase Latin-`U`) string literals (`U""`).

UTF-8 The Unicode Transformation Format-8 encoding. It is the encoding of *u8* (Lowercase Latin-`u` and `8`) string literals (`u8""`).

wide execution encoding The locale-based encoding related to "wide characters" (C and C++ magic words) processing during program evaluation/execution. It is directly related to the `std::set_locale(LC_CTYPE, ...)` calls. Note that this is different from the *wide literal encoding*, which is the encoding of wide string literals. The two may not be (and many times, are not) the same. Nominally, wide string literals are usually not like this, but there are a handful of compilers were they use neither UTF-16 or UTF-32 as the wide execution encoding, and instead use, for example, EUC-TW.

wide literal encoding The encoding of wide string literals (`L""`) during constant evaluation. This is usually controlled by command line arguments (GCC) or fixed during compilation (Clang as UTF-32, though that may change). Typically defaults to the system's "locale" setting.

1.6 Known Unicode Encodings

The list of known Unicode encodings is identical to the one found in a consuming project for C++ called ztd.text. That list can be found here.

The known Unicode encodings are important because they are evaluated before all other candidates as an intermediate in the service of performing an indirect encoding conversion; not every Unicode encoding is given such an elevated status, though. Only the encodings listed on the *indirect conversion desing documentation* are given the elevated encoding status and checked before all else: otherwise, the order of finding and the priority of picking such an indirect conversion is unspecified. A stronger guarantee can be made by using the select-based functions when opening a *cuneicode conversion routine in the registry*.

1.7 Progress & Future Work

This is where the status and progress of the library will be kept up to date. You can also check the Issue Tracker for specific issues and things being worked on!

1.7.1 More Encodings

More encodings should be supported by this library, to make development for others easier and easier. A good start with be targeting all of *iconv*'s encodings first and foremost. Then, integrating new encodings as individuals voice their need for them.

1.7.2 Arbitrary Indirections

Right now, encodings only traffic through the 3 well-known Unicode Encoding forms (UTF-8, UTF-16, UTF-32). It would be far more beneficial to allow connectivity through **any** encoding pair (but with preference shown to any Unicode encoding before taking the first go-between encoding that matches).

1.8 Benchmarks (In Progress)

> **Warning:** This isn't finished yet! Come check back by the next major or minor version update.

It's probably fine for now.

Probably!

1.9 Licenses, Thanks and Attribution

`ztd.cuneicode` is dual-licensed under either the Apache 2 License, or a corporate license if you bought it with special support. See the LICENSE file or your copy of the corporate license agreement for more details!

1.9.1 Previous and Related Works

We thank two people in particular:

- Bruno Haible and Daiki Ueno; iconv.

Their work was groundbreaking when it first came about and employed similar concepts found in this library. We thank them for their efforts in moving Text Encoding, Unicode, and Systems Programming forward. We also appreciate the work of:

- Unicode Consortium; the Unicode Standard;
- Unicode Consortium; the ICU - International Components for Unicode Library; and,
- Henri Sivonen; the encoding.rs library.

1.9.2 Charitable Contributions

`ztd.cuneicode` has been made possible by charitable contributions from patrons and sponsors around the world:

- Shepherd's Oasis, LLC (https://soasis.org)
- Jane Lusby
- Orfeas Zafeiris
- Tom Honermann
- Lily Foster
- Camilla Löwy
- Leonardo Lima
- Piotr Piatkowski
- Cynthia Coan
- Johan Andersson
- Erekose Craft
- Christopher Crouzet
- Michael Schellenberger Costa
- Turig Eret
- Brent Beer
- Matt Godbolt
- Erica Brescia
- Carol Chen
- Jeremy Jung
- Max Stoiber

- Evan Lock
- Anil Kumar
- Vincent Weevers
- Ólafur Waage
- Jeff Trull
- Davide Faconti
- Anthony Nandaa
- Christ Drozdowski
- Douglas Creager
- superfunc
- Michael Caisse
- Joshua Fisher
- Billy O'Neal
- Sy Brand
- Eric Tremblay
- Michał Dominiak
- Zach Toogood
- beluga
- Alex Gilding
- Kirk Shoop
- Alex Hadd
- Jimmy "junoravin"
- Joel Falcou
- Pascal Menuet
- Elias Daler
- Randomnetcat
- Robert Maynard
- Martin Hořeňovský
- Hana Dusíková
- 7 more private sponsors
- And many, many more!

(If you are new to being a patron, sponsor, or donator and you don't see your name here, I may have bungled the export list, so please e-mail opensource@soasis.org!)

1.10 Bibliography

These are all the resources that this documentation links to, in alphabetical order.

iconv Bruno Haible and Daiki Ueno. libiconv. August 2020. URL: https://savannah.gnu.org/projects/libiconv/. A software library for working with and converting text. Typically ships on most, if not all, POSIX and Linux systems.

ICU Unicode Consortium. "International Components for Unicode". April 17th, 2019. URL: https://github.com/hsivonen/encoding_rs The premiere library for not only performing encoding conversions, but performing other Unicode-related algorithms on sequences of text.

CHAPTER

TWO

INDICES & SEARCH

2.1 Index

INDEX

C

D

E

G

I

L

M

T

U

W