# ztd.cuneicode

***Release 0.0.0***

**ThePhD & Shepherd's Oasis, LLC**

**Jul 19, 2023**

# CONTENTS:

1 Who Is This Library For? 3

2 Indices & Search 161

Index 163

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 Quick 'n' Dirty Tutorial

### 1.1.1 Setup

Use of this library is officially supported through the use of CMake. Getting an updated CMake is difficult on non-Windows machines, especially if they come from your system's package manager distribution which tends to be several (dozen?) minor revisions out of date, or an entire major revision behind on CMake. To get a very close to up-to-date CMake, Python maintains an version that works across all systems. You can get it (and the ninja build system) by using the following command in your favorite command line application (assuming Python is already installed):

```
python -m pip install --user --update cmake ninja
```

If you depend on calling these executables using shorthand and not their full path, make sure that the Python "downloaded binaries" folder is contained with the `PATH` environment variable. Usually this is already done, but if you have trouble invoking `cmake --version` on your typical command line, please see the Python pip install documentation for more details for more information, in particular about the `--user` option.

If you do not have Python or CMake/ninja, you must get a recent enough version directly from CMake and build/install it and have a suitable build system around for CMake to pick up on (MSBuild from installing Visual Studio, make in most GNU distributions / MinGW on Windows on your PATH environment variable, and/or a personal installation of ninja).

### 1.1.2 Using CMake

Here's a sample of the `CMakeLists.txt` to create a new project and pull in ztd.text in the simplest possible way:

```
project(my_app
        VERSION 1.0.0
        DESCRIPTION "My application."
        HOMEPAGE_URL "https://ztdcuneicode.readthedocs.io/en/latest/quick.html"
        LANGUAGES C
)

include(FetchContent)

FetchContent_Declare(ztd.cuneicode
        GIT_REPOSITORY https://github.com/soasis/cuneicode.git
        GIT_SHALLOW    ON
        GIT_TAG        main)
FetchContent_MakeAvailable(ztd.cuneicode)
```

This will automatically download and set up all the dependencies ztd.cuneicode needs (in this case, simply ztd.cmake, ztd.platform, and ztd.idk ). You can override how ztd.cuneicode gets these dependencies using the standard FetchContent described in the CMake FetchContent Documentation. When done configuring, simply use CMake's `target_link_libraries(...)` to add it to the code:

```
# ...

file(GLOB_RECURSE my_app_sources
        LIST_DIRECTORIES OFF
        CONFIGURE_DEPENDS
        source/*.c
)

add_executable(my_app ${my_app_sources})

target_link_libraries(my_app PRIVATE ztd::cuneicode)
```

Once you have everything configured and set up the way you like, you can then use ztd.cuneicode in your code, as shown below:

```
#include <ztd/cuneicode.h>

#include <ztd/idk/size.h>

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

int main(int argc, char* argv[]) {
        if (argc < 2) {
                fprintf(stderr, "A name argument must be given to the program!");
                return 1;
        }

        const char* name      = argv[1];
```

(continues on next page)

(continued from previous page)

```c
        const size_t name_len = strlen(name);

        char utf8_name_buffer[4096]    = { 0 };
        const size_t utf8_name_max_len = ztdc_c_string_array_size(utf8_name_buffer);
        char* utf8_name                = utf8_name_buffer;
        const size_t name_len_limit    = (name_len * CNC_C8_MAX);

        if (name_len_limit >= utf8_name_max_len) {
                fprintf(stderr,
                     "The name provided tot hsi program was, unfortunately, too big!");
                return 2;
        }

        size_t utf8_name_len_after = utf8_name_max_len;
        size_t name_len_after      = name_len;
        cnc_mcerr err              = cnc_mcsntomcsn_exec_utf8(
          &utf8_name_len_after, &utf8_name, &name_len_after, &name);
        const size_t input_consumed = name_len - name_len_after;
        const size_t output_written = utf8_name_max_len - utf8_name_len_after;
        if (err != cnc_mcerr_ok) {
                const char* err_str = cnc_mcerr_to_str(err);
                fprintf(stderr,
                     "An error occurred when attempting to transcribe your name from "
                     "the execution encoding to UTF-8, stopping at input element %zu "
                     "and only writing %zu elements "
                     "(error name: %s).",
                     input_consumed, output_written, err_str);
                return 3;
        }

        printf("Hello there, ");
        fwrite(utf8_name_buffer, 1, output_written, stdout);
        printf("!\n");

        return 0;
}
```

Let's get started by digging into some examples!

---

**Note:** If you would like to see more examples and additional changes besides what is covered below, please do feel free to make requests for them here! This is not a very full-on tutorial and there is a lot of functionality that, still, needs explanation!

---

### 1.1.3 Simple Conversions

Simple conversions are provided for UTF-8, UTF-16, UTF-32, *execution encoding*, and *wide execution encoding*. They allow an end-user to use bit-based types.

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

```c

#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;
        size_t input_size           = ztdc_c_string_array_size(utf16_text);
        size_t output_size          = ztdc_c_array_size(utf8_text);
        cnc_mcstate_t state         = { 0 };
        // call the function with the right parameters!
        cnc_mcerr err               = cnc_c16snrtoc8sn( // formatting
          &output_size, &p_output,     // output first
          &input_size, &p_input,       // input second
          &state);                     // state parameter
        const size_t input_consumed = (ztdc_c_array_size(utf16_text) - input_size);
        const size_t output_written = (ztdc_c_array_size(utf8_text) - output_size);
        if (err != cnc_mcerr_ok) {
                const char* err_str = cnc_mcerr_to_str(err);
                fprintf(stderr,
                     "An (unexpected) error occurred and the conversion could not "
                     "happen! Error string: %s (code: '%d')\n",
                     err_str, (int)err);
                return 1;
        }

        printf(
             "Converted %zu UTF-16 code units to %zu UTF-8 code units, giving the "
             "text:",
             input_consumed, output_written);
        // requires a capable terminal / output, but will be
        // UTF-8 text!
        fwrite(utf8_text, sizeof(ztd_char8_t), output_written, stdout);
        printf("\n");

        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.

### Counting

More often than not, the exact amount of input and output might not be known before-hand. Therefore, it may be useful to count how many elements of output would be required before allocating exactly that much space to hold the result. In this case, simply passing `NULL` for the data output pointer instead of a real pointer, while providing a non-`NULL` pointer for the output size argument, will give an accurate reading for the amount of space that is necessary:

```
1
2  #include <ztd/cuneicode.h>
3
4  #include <ztd/idk/size.h>
5
6  #include <stdio.h>
7  #include <string.h>
8  #include <stdlib.h>
9
10 int main() {
11
12         const ztd_char16_t utf16_text[] = u"";
13
14         const ztd_char16_t* p_count_input = utf16_text;
15         // This size does NOT include the null terminating character.
16         size_t count_input_size   = ztdc_c_string_array_size(utf16_text);
17         cnc_mcstate_t count_state = { 0 };
18         size_t output_size_after  = SIZE_MAX;
19         // Use the function but with "nullptr" for the output pointer
20         cnc_mcerr count_err = cnc_c16snrtoc8sn(
21              // To get the proper size for this conversion, we use the same
22              // function but with "NULL" specificers:
23              &output_size_after, NULL,
24              // input second
25              &count_input_size, &p_count_input,
26              // state parameter
27              &count_state);
28         // Compute the needed space:
29         const size_t output_size_needed = SIZE_MAX - output_size_after;
30         if (count_err != cnc_mcerr_ok) {
31                 const char* err_str = cnc_mcerr_to_str(count_err);
32                 fprintf(stderr,
```

(continues on next page)

(continued from previous page)

```
                    "An (unexpected) error occurred and the counting could not "
                    "happen! Error string: %s (code: '%d')\n",
                    err_str, (int)count_err);
            return 1;
        }

        ztd_char8_t* utf8_text = malloc(output_size_needed * sizeof(ztd_char8_t));

        // prepare for potential error return and error handling
        int return_value = 0;

        if (utf8_text == NULL) {
                return_value = 2;
                goto early_exit;
        }
        ztd_char8_t* p_output = utf8_text;
        cnc_mcstate_t state   = { 0 };

        // Now, actually output it
        const ztd_char16_t* p_input = utf16_text;
        // ztdc_c_array_size INCLUDES the null terminator in the size!
        size_t input_size  = ztdc_c_string_array_size(utf16_text);
        size_t output_size = output_size_needed;
        cnc_mcerr err      = cnc_c16snrtoc8sn(
          // output first
          &output_size, &p_output,
          // input second
          &input_size, &p_input,
          // state parameter
          &state);
        const size_t input_consumed
             = ztdc_c_string_array_size(utf16_text) - input_size;
        const size_t output_written  = output_size_needed - output_size;
        const bool conversion_failed = err != cnc_mcerr_ok;
        if (conversion_failed) {
                // get error string to describe error code
                const char* err_str = cnc_mcerr_to_str(err);
                fprintf(stderr,
                    "An (unexpected) error occurred and the conversion could not "
                    "happen! The error occurred at UTF-16 input element #%zu, and only "
                    "managed to output %zu UTF-8 elements. Error string: %s (code: "
                    "'%d')\n",
                    input_consumed, output_written, err_str, (int)err);
                return_value = 3;
                goto early_exit;
        }
        // requires a capable terminal / output, but will be
        // UTF-8 text!
        printf("Converted UTF-8 text:\n");
        fwrite(utf8_text, sizeof(ztd_char8_t), output_written, stdout);
        printf("\n");

```

(continues on next page)

(continued from previous page)

```
early_exit:
        if (utf8_text != NULL)
                free(utf8_text);

        return return_value;
}
```

Here, we find out the `output_size_needed` by taking the size before the call, then subtracting it by the decremented size from after the function call. Then, after checking for errors, we do the actual conversion with a properly sized buffer that includes a null terminator so the conversion result is suitable for printing to a (UTF-8 capable) terminal. Finally, after completing our task, we free the memory and return a proper error code.

### Unbounded Output Writing

Sometimes, it is know ahead of time that there is enough space in a given buffer for a given conversion result because the inputs are not at all associated with user input or user-facing anything (e.g., static storage duration string literals with known sizes and elements). If that is the case, then a `NULL` value can be passed in for the output size argument, and the function will assume that there is enough space for writing:

```

#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"";

        const ztd_char16_t* p_count_input = utf16_text;
        // This size does NOT include the null terminating character.
        size_t count_input_size   = ztdc_c_string_array_size(utf16_text);
        cnc_mcstate_t count_state = { 0 };
        size_t output_size_after  = SIZE_MAX;
        // Use the function but with "nullptr" for the output pointer
        cnc_mcerr count_err = cnc_c16snrtoc8sn(
             // To get the proper size for this conversion, we use the same
             // function but with "NULL" specificers:
             &output_size_after, NULL,
             // input second
             &count_input_size, &p_count_input,
             // state parameter
             &count_state);
        // Compute the needed space:
        const size_t output_size_needed = SIZE_MAX - output_size_after;
        if (count_err != cnc_mcerr_ok) {
                const char* err_str = cnc_mcerr_to_str(count_err);
                fprintf(stderr,
```

(continues on next page)

(continued from previous page)

```
                    "An (unexpected) error occurred and the counting could not "
                    "happen! Error string: %s (code: '%d')\n",
                    err_str, (int)count_err);
            return 1;
        }

        ztd_char8_t* utf8_text = malloc(output_size_needed * sizeof(ztd_char8_t));

        // prepare for potential error return and error handling
        int return_value = 0;

        if (utf8_text == NULL) {
                return_value = 2;
                goto early_exit;
        }
        ztd_char8_t* p_output = utf8_text;
        cnc_mcstate_t state   = { 0 };

        // Now, actually output it
        const ztd_char16_t* p_input = utf16_text;
        // ztdc_c_array_size INCLUDES the null terminator in the size!
        size_t input_size  = ztdc_c_string_array_size(utf16_text);
        size_t output_size = output_size_needed;
        cnc_mcerr err      = cnc_c16snrtoc8sn(
          // output first
          &output_size, &p_output,
          // input second
          &input_size, &p_input,
          // state parameter
          &state);
        const size_t input_consumed
             = ztdc_c_string_array_size(utf16_text) - input_size;
        const size_t output_written  = output_size_needed - output_size;
        const bool conversion_failed = err != cnc_mcerr_ok;
        if (conversion_failed) {
                // get error string to describe error code
                const char* err_str = cnc_mcerr_to_str(err);
                fprintf(stderr,
                    "An (unexpected) error occurred and the conversion could not "
                    "happen! The error occurred at UTF-16 input element #%zu, and only "
                    "managed to output %zu UTF-8 elements. Error string: %s (code: "
                    "'%d')\n",
                    input_consumed, output_written, err_str, (int)err);
                return_value = 3;
                goto early_exit;
        }
        // requires a capable terminal / output, but will be
        // UTF-8 text!
        printf("Converted UTF-8 text:\n");
        fwrite(utf8_text, sizeof(ztd_char8_t), output_written, stdout);
        printf("\n");

```

(continues on next page)

(continued from previous page)

```
early_exit:
        if (utf8_text != NULL)
                free(utf8_text);

        return return_value;
}
```

This can be useful for performance-oriented scenarios, where writing without doing any bounds checking may result in deeply improved speed.

### Validating

Validation is similar to counting, except that the output size argument is `NULL`. This effectively allows someone to check if the input is not only valid for that encoding, but also if it can be transcoded to the output assuming enough size.

```

#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"";

        const ztd_char16_t* count_input_ptr = utf16_text;
        // ztdc_c_array_size INCLUDES the null terminator in the size!
        const size_t initial_count_input_size = ztdc_c_array_size(utf16_text);
        size_t count_input_size               = initial_count_input_size;
        cnc_mcstate_t count_state             = { 0 };
        // Use the function but with "nullptr" for the output pointer
        cnc_mcerr err = cnc_c16snrtoc8sn(
             // To get the proper size for this conversion, we use the same
             // function but with "NULL" specificers:
             NULL, NULL,
             // input second
             &count_input_size, &count_input_ptr,
             // state parameter
             &count_state);
        size_t input_read = (size_t)(initial_count_input_size - count_input_size);
        if (err != cnc_mcerr_ok) {
                const char* err_str = cnc_mcerr_to_str(err);
                fprintf(stderr,
                     "An (unexpected) error occurred and the counting/validating could "
                     "not happen!\nThe error happened at code unit %zu in the UTF-16 "
                     "input.\nError string: %s (code: '%d')\n",
                     input_read, err_str, (int)err);
        }
```

(continues on next page)

(continued from previous page)

```

        printf(
             "The input UTF-16 is valid and consumed all %zu code units (elements) "
             "of input.\n",
             input_read);

        return 0;
}
```

In many instances, simply validating that text can be converted rather than attempting the conversion can provide *a far greater degree of speed using specialized algorithms and instruction sets*.

### 1.1.4 Registry-Based Conversions

Conversion registries in cuneicode provide a way to obtain potentially runtime-defined encodings. It can be added to and removed from by a user, and all access to data (save for those which are defined to access global state such as the `char/execution` and `wchar_t/wide execution` encodings) is referenced straight from the objects created and involved and should involve no global, mutable state. This should enable users to create, use, and pass around registry objects freely without the burden of pre-allocated or statically-shared state, resulting in programs that are easier to reason about. Here is an example of converting between the Windows-1251 encoding and the UTF-8 encoding by passing both names to `` `cnc_conv_new(...)` ``:

```

#include <ztd/cuneicode.h>

#include <ztd/idk/size.h>

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

int main() {

        cnc_conversion_registry* registry = NULL;
        {
                cnc_open_err err
                     = cnc_registry_new(&registry, cnc_registry_options_default);
                if (err != cnc_open_err_ok) {
                        const char* err_str = cnc_open_err_to_str(err);
                        fprintf(stderr,
                             "An unexpected error has occurred: '%s' (code: '%d')", err_
↪str,
                             (int)err);
                        return 1;
                }
        }

        // Now that we've allocated, have a return value
        // just in case
        int return_value     = 0;
        cnc_conversion* conv = NULL;
```

(continues on next page)

(continued from previous page)

```
        {
                cnc_conversion_info conv_info = { 0 };
                cnc_open_err err              = cnc_conv_new(
               registry, "windows-1251", "utf-8", &conv, &conv_info);
                if (err != cnc_open_err_ok) {
                        const char* err_str = cnc_open_err_to_str(err);
                        fprintf(stderr, "An unexpected error has occurred: %s (code: '%d
↪')",
                                err_str, (int)err);
                        return_value = 2;
                        goto early_exit0;
                }
                // the conversion info structure can tell us about things
                printf(
                     "Successfully opened a registry conversion between %.*s and "
                     "%.*s!\n",
                     (int)conv_info.from_code_size,
                     (const char*)conv_info.from_code_data, (int)conv_info.to_code_size,
                     (const char*)conv_info.to_code_data);
                if (conv_info.is_indirect) {
                        // the strings used for this printf are UTF-8 encoded, but we
                        // know the names are ASCII-compatible, charset-invariant strings
                        // thanks to the request above, so we don't do the special
                        // printing method.
                        printf(
                             "(It is an indirect conversion, going from %.*s to %.*s, "
                             "then %.*s to %.*s for the conversion.)\n",
                             (int)conv_info.from_code_size,
                             (const char*)conv_info.from_code_data,
                             (int)conv_info.indirect_code_size,
                             (const char*)conv_info.indirect_code_data,
                             (int)conv_info.indirect_code_size,
                             (const char*)conv_info.indirect_code_data,
                             (int)conv_info.to_code_size,
                             (const char*)conv_info.to_code_data);
                }
                else {
                        printf(
                             "(The conversion is a direct conversion and deos not take␣
↪an "
                             "intermediate conversion path.)\n");
                }
        }

        const char input[]
             = "\xd1\xeb\xe0\xe2\xe0\x20\xd3\xea\xf0\xe0\xbf\xed\xb3\x21\x0a";

        const unsigned char* count_input_last = (const unsigned char*)input;
        size_t count_input_byte_size_leftover = ztdc_c_array_size(input);
        size_t count_output_byte_size         = SIZE_MAX;

        const cnc_mcerr count_err = cnc_conv(conv, &count_output_byte_size, NULL,
```

(continues on next page)

(continued from previous page)

```
        &count_input_byte_size_leftover, &count_input_last);
    const size_t output_byte_size_needed = SIZE_MAX - count_output_byte_size;
    const size_t count_input_byte_size_consumed
        = ztdc_c_array_size(input) - count_input_byte_size_leftover;
    if (count_err != cnc_mcerr_ok) {
        const char* err_str = cnc_mcerr_to_str(count_err);
        fprintf(stderr,
            "The counting step failed with the error %s (code: "
            "'%d') at byte #%zu in the input (which presently needs %zu bytes "
            "of output space)",
            err_str, (int)count_err, count_input_byte_size_consumed,
            output_byte_size_needed);
        return_value = 3;
        goto early_exit1;
    }

    const unsigned char* input_last = (const unsigned char*)input;
    size_t input_size               = ztdc_c_array_size(input);
    // not strictly necessary to multiply by unsigned char since it's
    // defined to be 1, but it's consistent with other places where
    // malloc gets used...
    unsigned char* output
        = malloc(output_byte_size_needed * sizeof(unsigned char));
    if (output == NULL) {
        return_value = 4;
        goto early_exit2;
    }
    unsigned char* output_last       = output;
    size_t output_byte_size_leftover = output_byte_size_needed;
    cnc_mcerr err = cnc_conv(conv, &output_byte_size_leftover, &output_last,
        &input_size, &input_last);
    const size_t output_byte_size_written
        = output_byte_size_needed - output_byte_size_leftover;
    const size_t input_byte_size_consumed
        = ztdc_c_array_size(input) - count_input_byte_size_leftover;
    if (err != cnc_mcerr_ok) {
        const char* err_str = cnc_mcerr_to_str(err);
        fprintf(stderr,
            "The conversion step failed to convert with the error %s "
            "(code: '%d') at byte #%zu in the input after writing as far as "
            "byte #%zu in the output",
            err_str, (int)count_err, input_byte_size_consumed,
            output_byte_size_written);
        return_value = 5;
        goto early_exit2;
    }

    printf(
        "The registry conversion was successful, writing %zu output bytes after "
        "reading %zu input bytes.\nThe output is:\n",
        output_byte_size_written, input_byte_size_consumed);
    // It's UTF-8: this should print correctly on a UTF-8 capable terminal.
```

(continues on next page)

(continued from previous page)

```
        // We do not mill through `printf` because it can do a (potentially lossy)
        // conversion.
        fwrite((const char*)output, sizeof(char), output_byte_size_written, stdout);
        printf("\n");

early_exit2:
        if (output != NULL) {
                free(output);
        }
early_exit1:
        if (conv != NULL) {
                cnc_conv_delete(conv);
        }
early_exit0:
        if (registry != NULL) {
                cnc_registry_delete(registry);
        }
        return return_value;
}
```

Care must be taken that, upon allocating one of these types, it is deallocated with care. A large number of additional registry functionality is described in the *registry design documentation* and the *registry API documentation*, including an example of registering an encoding which contains its own state and must be stored in a `cnc_conv*` handle.

Most importantly in this short example is that there is no direct conversion between Windows-1251 and UTF-8 in the default offerings of cuneicode. Instead, the registry knows how to negotiate a pathway between the registered Windows-1251 encoding (which goes from itself to UTF-32 and back) to UTF-8 (which goes from itself to UTF-32 and back). This automatic handling of indirection is provided by-default and is described in the *registry design for indirections*.

## 1.2 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.

### 1.2.1 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 truths must hold if an indivisible unit of work is attempted and completed:

- enough input is consumed to perform an output or change the internal state;
- output is written from a (potentially accumulated) internal state;
- 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`, `cnc_mcstate_t`, or similar `state`-type data pointer - may or may not change during any of these operations, and may be left in an indeterminate state after an error occurs.

Using this concept, we can have multi/"bulk" conversion be defined as the use of multiple successfully completed indivisible units of work. This provides us with a solid base from which to work from as we start working with various different encodings and their constraints.

For the purposes of cuneicode, it primarily deals in taking a pointer (or a pointer-to-pointer) to data and sizes, updating those if and only if an indivisible unit of work is successfully completed. For bulk conversions, it stops at the last successfully completed unit of work.

### 1.2.2 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, the libiconv **interface** itself serves as a very good blueprint. Some minor modifications and we end up with a general-form API that tends to work for just about every kind of conversion we would potentially come across in the usual C environment:

```
error_code conversion_name(
        size_t* ptr_input_size, // (0)
        input_char_type** ptr_to_input, // (1)
        size_t* ptr_output_size, // (2)
        output_char_type** ptr_to_input, // (3)
        state_type* state  // (4)
)
```

The types used here are as follows:

0. A pointer to the size of the input.

1. A pointer to a buffer of type *input_char_type*. The type can be anything that suits the input data, such as e.g. `char32_t`.
2. A pointer to the size of the output, **if needed**. Not providing specifically this allows for assuming the output buffer has enough space to handle the input.
3. A pointer to a buffer of type *output_char_type*. The type can be anything that suits the output data, such as e.g. `char32_t` for transcoding to UTF-32.

There are a lot of alternative designs for this kind of functionality, but this form was ultimately chosen for two specific reasons:

- it's ability to be optimized by a cognizant implementation that can take advantage of null pointers for the *ptr_to_output* and *ptr_to_output_size* parameters; and,
- it's ability to provide unmitigated access to **both** input and output size and progress simultaneously without needing to rely on implementation-defined/undefined behaviors (pointer subtraction that may fall outside the range of `ptrdiff_t`, ).
-
-
- **no** replacement characters are inserted by the conversion routines; it is up to the user to insert conversions within the output space or similar where they deem it necessary.

### 1.2.3 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.

### 1.2.4 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), per doing *one indivisible unit of work*.

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 doing an output operation will never be more than a statically-known, maximally-sized buffer (and thus avoid ever having to handle the `cnc_mcerr_insufficient_output` return code). This is necessary for low-memory environments which may need to process input with the smallest possible memory guarantee available. As an example, given the general form of a function (whose parameters and style is discussed in the *functions section*), one can put together a concrete general-purpose

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 indivisible units of input at a time through much more efficient means.

The benefit of a Single Conversion API that only performs one indivisible unit of work is the ability to build every single other piece of the API through this very functionality.

conversion algorithms to become single conversion algorithms, though the performance of such an algorithm is noticeably worse.

### 1.2.5 Multiple (Bulk) Conversion

Bulk conversions are a way to convert multiple complete *indivisible units of work* 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 either (a) input exhaustion AND state completion, or (b) 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 and whether (valid) input is given to the transcoding operation given, 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_mcerr_insufficient_output`, which must be handled accordingly in order to fully process the given input.

#### Simulating Single Conversions with Bulk Conversions

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_mcerr_ok*) or failure. If a known maximum upper limit is known for a given encoding, one can limit the incremental loop that increases the size of the input to that maximum limit.

Note that the above process can be incredibly slow since it requires walking over the same input over and over again until a non-`cnc_mcerr_incomplete_sequence` error code occurs. All of the named encoding functions that are not specialized for speed are instead implemented using the technique described in *the indivisible unit of work* documentation, and this technique is only employed for runtime-added, registry-based encodings that only provide bulk conversion through e.g. *cnc_registry_add_multi()*.

### 1.2.6 State

Occasionally, state objects are passed to the routines that perform conversions. These can do things such as hold onto shift state, have special flags for special processing (e.g., for *assuming input for a given function is valid*). In general, the state parameter comes last and is a pointer to a state the user creates (or, if using the *typed conversion functions*, may be created for you and used.) The function therefore usually ends up of the form:

```
#include <uchar.h>

typedef struct my_state {
        bool assume_valid_input;
        char accumulation[2];
} my_state;

cnc_mcerr my_single_conversion_to_utf32(size_t* p_output_size,
        const char32_t** p_output,
        size_t* p_input_size,
        const char** p_input,
        my_state* p_state);
```

As described by the *indivisible unit of work*, reading input can change the state internally, even accumulate data in the state while outputting 0 characters. Because state may influence whether or not a conversion is complete (e.g., it has accumulated data inside of it and must empty it out), complex state objects should add a visible inspection function to check if the state is "complete" (has nothing left to output **and** is awaiting no further characters):

```
#include <uchar.h>

typedef struct my_conversion_state {
        char accumulation[2];
        unsigned char accumulation_count;
        bool assume_valid_input;
} my_conversion_state;

bool my_state_is_complete(const my_conversion_state* p_state);

cnc_mcerr my_single_conversion_to_utf32(size_t* p_output_size,
        const char32_t** p_output,
        size_t* p_input_size,
        const char** p_input,
        my_conversion_state* p_state);
```

The *cnc_mcstate_t* type has a similar function named *cnc_mcstate_is_complete()*, used for this purpose. As a completely made up example for a complete made up encoding,

```c
#include <ztd/cuneicode.h>

#include <ztd/idk/size.h>
#include <ztd/idk/restrict.h>

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

cnc_mcerr my_bulk_mcsnrtoc8sn(size_t* output_size, unsigned char** output,
     size_t* input_size, const char** restrict input, cnc_mcstate_t* state) {
        cnc_mcstate_t invocation_unique_internal_state;
        if (state == NULL) {
                invocation_unique_internal_state = (cnc_mcstate_t) { 0 };
                state                            = &invocation_unique_internal_state;
        }
        if (input == NULL || *input == NULL) {
                return cnc_mcnrtoc8n(output_size, output, input_size, input, state);
        }
        for (;;) {
                cnc_mcerr err
                     = cnc_mcnrtoc8n(output_size, output, input_size, input, state);
                if (err != cnc_mcerr_ok) {
                        return err;
                }
                if (*input_size > 0) {
                        continue;
                }
                bool state_finished = cnc_mcstate_is_complete(state);
                if (!state_finished) {
                        continue;
                }
                return err;
        }
}

int main() {
        const char input[]              = "abc";
        const char* input_ptr           = input;
        const size_t initial_input_size = ztdc_c_array_size(input);
        size_t input_size               = initial_input_size;

        unsigned char output[CNC_C8_MAX * ztdc_c_array_size(input)] = { 0 };
        unsigned char* output_ptr                                   = output;
        const size_t initial_output_size = ztdc_c_array_size(output);
        size_t output_size               = initial_output_size;

        cnc_mcerr err = my_bulk_mcsnrtoc8sn(
             &output_size, &output_ptr, &input_size, &input_ptr, NULL);
        const size_t output_written = initial_output_size - output_size;
```

(continues on next page)

(continued from previous page)

```
        const size_t input_read     = initial_input_size - input_size;
        if (err != cnc_mcerr_ok) {
                fprintf(stderr, "The conversion failed with an unexpected error of %s.",
                     cnc_mcerr_to_str(err));
                return 1;
        }

        const unsigned char expected_output[] = "abc";

        if (memcmp(output, expected_output, output_written) != 0) {
                fprintf(stderr,
                     "The expected intput was not the same as the expected output.");
                return 2;
        }

        fprintf(stdout, "Read: %zu units\nWrote: %zu units\nWritten value: %s\n",
             input_read, output_written, (const char*)output);

        return 0;
}
```

### 1.2.7 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.

There is a priority ordering to which encodings are chosen as indirect liaisons or indirect substrates to help encode from one unit of text to the other, and they are as follows:

1. UTF-32
2. UTF-32 Unchecked
3. UTF-8
4. UTF-8 Unchecked
5. UTF-16
6. UTF-16 Unchecked
7. Everything Else.

Indirect encoding conversions use the *cnc_pivot_info* type that denotes a buffer of space to use as scratch space if any algorithm cannot perform a direct conversion. Pivots can help avoid any implementation-defined, stack-allocated buffer size that might be too large for the inputs used (and thus overflow the stack) or too small for the inputs used (and thus require multiple calls to the conversion algorithm).

#### 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.

## 1.3 API Reference

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

### 1.3.1 General Structures, Enumerations, and Constants

#### cnc_mcerr

const char ***cnc_mcerr_to_str**(cnc_mcerr __err)
Returns a string representing the error code's name.

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

enumerator **cnc_mcerr_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_mcerr_invalid_sequence* should be used instead.

---

enumerator **cnc_mcerr_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_mcerr_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_mcerr_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_err

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

*Values:*

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

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

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

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

enumerator **cnc_open_err_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_mcerr_insufficient_output` must be returned. Similarly, if there is insufficient data and the data is at the very end, then `cnc_mcerr_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 *cnc_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 *cnc_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.

### State Functions

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_is_assuming_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 return 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.

### 1.3.2 Typed Conversions

#### Encodings

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

#### Available Encodings

All of the encodings available here are identical to the ones available in ztd.text, and the documentation there gives a long list. The only thing that is different is that this uses function calls and, perhaps, state objects to achieve a similar effect. The ones with interesting consequences for the API shape are linked below; the rest follow a more or less identical pattern to preexisting functionality.

#### Special, Stateful APIs

#### 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 being derived from non-ASCII characters.

#### Transcoding Functions

cnc_mcerr **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_mcerr **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_mcerr **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_mcerr **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*

### State Type

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.

### State Functions

void **cnc_pny_decode_state_set_input_incomplete**(*cnc_pny_decode_state_t* *__state)
Tells the state that input should still be expected.

**Parameters** **__state** – **[inout]** The state to remove the expectation that input is complete from.

void **cnc_pny_encode_state_set_input_incomplete**(*cnc_pny_encode_state_t* *__state)
Tells the state that input should still be expected.

**Parameters** **__state** – **[inout]** The state to turn off its current completion state.

void **cnc_pny_decode_state_set_input_complete**(*cnc_pny_decode_state_t* *__state)
Returns whether or not the given *cnc_pny_decode_state_t* is expecting anymore input.

**Parameters** **__state** – **[inout]** The state to trigger the completion on.

void **cnc_pny_encode_state_set_input_complete**(*cnc_pny_encode_state_t* *__state)
Tells the state that input should still be expected.

**Parameters** **__state** – **[inout]** The state to remove the expectation that input is complete from.

bool **cnc_pny_decode_state_is_input_complete**(const *cnc_pny_decode_state_t* *__state)
Returns whether or not the given *cnc_pny_decode_state_t* is expecting anymore input.

**Parameters** **__state** – **[inout]** The state to to inspect.

bool **cnc_pny_encode_state_is_input_complete**(const *cnc_pny_encode_state_t* *__state)
Returns whether or not the given *cnc_pny_encode_state_t* is expecting anymore input.

**Parameters** **__state** – **[inout]** The state to inspect.

void **cnc_pny_decode_state_set_assume_valid**(*cnc_pny_decode_state_t* *__state, bool __value)
Returns whether or not the given *cnc_pny_decode_state_t* has no more data that needs to be output.

**Parameters** **__state** – **[inout]** The state to make operations assume the input is valid.

void **cnc_pny_encode_state_set_assume_valid**(*cnc_pny_encode_state_t* *__state, bool __value)
Returns whether or not the given *cnc_pny_encode_state_t* has no more data that needs to be output.

**Parameters** **__state** – **[inout]** The state to make operations assume the input is valid.

bool **cnc_pny_decode_state_is_assuming_valid**(const *cnc_pny_decode_state_t* *__state)
Returns whether or not the given *cnc_pny_decode_state_t* is assuming input data is valid.

**Parameters** **__state** – **[inout]** The state to inspect.

bool **cnc_pny_encode_state_is_assuming_valid**(const *cnc_pny_encode_state_t* *__state)
Returns whether or not the given *cnc_pny_encode_state_t* is assuming input data is valid.

**Parameters** **__state** – **[inout]** The state to inspect.

### Single and Multibyte Encoding APIs

These APIs use *the typical cnc_mcstate_t* state (and may not do anything to it internally in the functions, if the API is stateless). They do not have any higher-level or special functionality associated with them.

### Single Byte Encodings

Here's a list of named function functions that are multi-byte encodings that are driven by more complex logic and algorithms which are not so simple. The descriptions for these encodings with matching names can be found at the ztd.text encodings documentation.

### Known Named Encoding Functions

cnc_mcerr **cnc_mcnrtoc32n_big5_hkscs**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_big5_hkscs**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_big5_hkscs**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_big5_hkscs**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_gb18030**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_gb18030**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_gb18030**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_gb18030**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_gbk**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_gbk**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_gbk**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_gbk**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_shift_jis_x0208**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_shift_jis_x0208**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_shift_jis_x0208**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_shift_jis_x0208**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32snrtomcsn*

### Single Byte Encodings

Here's a list of named function functions that are effectively single-byte encodings, driven by simple algorithms or table-based lookup. The descriptions for these encodings with matching names can be found at the ztd.text single byte encoding documentation.

### Known Named Encoding Functions

cnc_mcerr **cnc_mcnrtoc32n_ascii**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_ascii**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_ascii**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_ascii**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_atari_st**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_atari_st**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_atari_st**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_atari_st**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_atascii**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_atascii**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_atascii**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_atascii**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_ibm_424_hebrew_bulletin**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_ibm_424_hebrew_bulletin**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_ibm_424_hebrew_bulletin**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_ibm_424_hebrew_bulletin**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_ibm_856_hebrew**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_ibm_856_hebrew**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_ibm_856_hebrew**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_ibm_856_hebrew**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_ibm_866_cyrillic**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_ibm_866_cyrillic**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_ibm_866_cyrillic**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_ibm_866_cyrillic**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_ibm_1006_urdu**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_ibm_1006_urdu**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_ibm_1006_urdu**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_ibm_1006_urdu**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_iso_8859_1_1985**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_iso_8859_1_1985**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_iso_8859_1_1985**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_iso_8859_1_1985**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_iso_8859_1_1998**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_iso_8859_1_1998**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_iso_8859_1_1998**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_iso_8859_1_1998**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_iso_8859_1**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_iso_8859_1**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_iso_8859_1**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_iso_8859_1**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_iso_8859_2**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_iso_8859_2**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_iso_8859_2**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_iso_8859_2**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_iso_8859_3**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_iso_8859_3**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_iso_8859_3**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_iso_8859_3**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_iso_8859_4**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_iso_8859_4**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_iso_8859_4**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_iso_8859_4**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_iso_8859_5**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_iso_8859_5**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_iso_8859_5**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_iso_8859_5**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_iso_8859_6**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_iso_8859_6**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_iso_8859_6**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_iso_8859_6**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_iso_8859_7**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_iso_8859_7**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_iso_8859_7**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_iso_8859_7**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_iso_8859_8**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_iso_8859_8**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_iso_8859_8**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_iso_8859_8**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_iso_8859_10**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_iso_8859_10**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_iso_8859_10**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_iso_8859_10**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_iso_8859_13**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_iso_8859_13**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_iso_8859_13**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_iso_8859_13**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_iso_8859_14**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_iso_8859_14**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_iso_8859_14**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_iso_8859_14**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_iso_8859_15**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_iso_8859_15**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_iso_8859_15**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_iso_8859_15**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_iso_8859_16**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_iso_8859_16**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_iso_8859_16**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_iso_8859_16**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_kamenicky**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_kamenicky**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_kamenicky**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_kamenicky**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_kazakh_strk1048**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_kazakh_strk1048**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_kazakh_strk1048**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_kazakh_strk1048**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_koi8_r**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_koi8_r**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_koi8_r**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_koi8_r**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_koi8_u**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_koi8_u**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_koi8_u**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_koi8_u**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_petscii_unshifted**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, cnc_petscii_state_t *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_petscii_unshifted**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, cnc_petscii_state_t *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_petscii_unshifted**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, cnc_petscii_state_t *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_petscii_unshifted**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, cnc_petscii_state_t *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_petscii_shifted**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, cnc_petscii_state_t *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_petscii_shifted**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, cnc_petscii_state_t *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_petscii_shifted**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, cnc_petscii_state_t *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_petscii_shifted**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, cnc_petscii_state_t *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_tatar_ansi**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_tatar_ansi**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_tatar_ansi**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_tatar_ansi**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_tatar_ascii**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_tatar_ascii**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_tatar_ascii**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_tatar_ascii**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_windows_437_dos_latin_us**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_windows_437_dos_latin_us**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_windows_437_dos_latin_us**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_windows_437_dos_latin_us**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_windows_865_dos_nordic**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_windows_865_dos_nordic**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_windows_865_dos_nordic**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_windows_865_dos_nordic**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_windows_874**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_windows_874**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_windows_874**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_windows_874**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_windows_1251**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_windows_1251**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_windows_1251**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_windows_1251**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_windows_1252**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_windows_1252**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_windows_1252**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_windows_1252**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_windows_1253**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_windows_1253**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_windows_1253**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_windows_1253**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_windows_1254**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_windows_1254**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_windows_1254**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_windows_1254**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_windows_1255**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_windows_1255**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_windows_1255**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_windows_1255**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_windows_1256**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_windows_1256**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_windows_1256**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_windows_1256**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_windows_1257**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_windows_1257**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_windows_1257**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_windows_1257**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32snrtomcsn*

cnc_mcerr **cnc_mcnrtoc32n_windows_1258**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcnrtoc32n*

cnc_mcerr **cnc_c32nrtomcn_windows_1258**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char32_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_c32nrtomcn*

cnc_mcerr **cnc_mcsnrtoc32sn_windows_1258**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_t **__p_src, *cnc_mcstate_t* *__p_state)

**See also:**

*cnc_mcsnrtoc32sn*

cnc_mcerr **cnc_c32snrtomcsn_windows_1258**(size_t *__p_maybe_dst_len, ztd_char_t **__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*. This page covers the canonical conversions, which are from UTF-8/16/32 and the execution/wide execution encodings, to each other.

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 or similar state 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).

**Important:** Non-restartable functions mean that a automatic storage duration object unique to the function invocation is created for you, before being used. If there is any necessary state left in the object before the function ends, it ends up being discarded and lost if the non-restartable functions are used.

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 function 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 and DANGEROUS.

### Bulk Conversion Functions

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

cnc_mcerr **cnc_mcsntomcsn**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **cnc_mcsnrtomcsn**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_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_mcerr_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_mcerr_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_mcerr **cnc_mcsntomwcsn**(size_t *__p_maybe_dst_len, ztd_wchar_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **cnc_mcsnrtomwcsn**(size_t *__p_maybe_dst_len, ztd_wchar_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_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_mcerr_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_mcerr_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_mcerr **cnc_mcsntoc8sn**(size_t *__p_maybe_dst_len, ztd_char8_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **cnc_mcsnrtoc8sn**(size_t *__p_maybe_dst_len, ztd_char8_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_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_mcerr_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_mcerr_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_mcerr **cnc_mcsntoc16sn**(size_t *__p_maybe_dst_len, ztd_char16_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **cnc_mcsnrtoc16sn**(size_t *__p_maybe_dst_len, ztd_char16_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_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_mcerr_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_mcerr_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_mcerr **cnc_mcsntoc32sn**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **cnc_mcsnrtoc32sn**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_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_mcerr_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_mcerr_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_mcerr **cnc_mwcsntomcsn**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **cnc_mwcsnrtomcsn**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_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_mcerr_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_mcerr_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_mcerr **cnc_mwcsntomwcsn**(size_t *__p_maybe_dst_len, ztd_wchar_t **__p_maybe_dst, size_t *__p_src_len, const ztd_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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **cnc_mwcsnrtomwcsn**(size_t *__p_maybe_dst_len, ztd_wchar_t **__p_maybe_dst, size_t *__p_src_len, const ztd_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_mcerr_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_mcerr_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_mcerr **cnc_mwcsntoc8sn**(size_t *__p_maybe_dst_len, ztd_char8_t **__p_maybe_dst, size_t *__p_src_len, const ztd_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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **cnc_mwcsnrtoc8sn**(size_t *__p_maybe_dst_len, ztd_char8_t **__p_maybe_dst, size_t *__p_src_len, const ztd_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_mcerr_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_mcerr_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_mcerr **cnc_mwcsntoc16sn**(size_t *__p_maybe_dst_len, ztd_char16_t **__p_maybe_dst, size_t *__p_src_len, const ztd_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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **cnc_mwcsnrtoc16sn**(size_t *__p_maybe_dst_len, ztd_char16_t **__p_maybe_dst, size_t *__p_src_len, const ztd_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_mcerr_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_mcerr_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_mcerr **cnc_mwcsntoc32sn**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **cnc_mwcsnrtoc32sn**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_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_mcerr_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_mcerr_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_mcerr **cnc_c8sntomcsn**(size_t *__p_maybe_dst_len, ztd_char_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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **cnc_c8snrtomcsn**(size_t *__p_maybe_dst_len, ztd_char_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_mcerr_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_mcerr_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_mcerr **cnc_c8sntomwcsn**(size_t *__p_maybe_dst_len, ztd_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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **cnc_c8snrtomwcsn**(size_t *__p_maybe_dst_len, ztd_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_mcerr_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_mcerr_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_mcerr **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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **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_mcerr_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_mcerr_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_mcerr **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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **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_mcerr_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_mcerr_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_mcerr **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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **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_mcerr_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_mcerr_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_mcerr **cnc_c16sntomcsn**(size_t *__p_maybe_dst_len, ztd_char_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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **cnc_c16snrtomcsn**(size_t *__p_maybe_dst_len, ztd_char_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_mcerr_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_mcerr_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_mcerr **cnc_c16sntomwcsn**(size_t *__p_maybe_dst_len, ztd_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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **cnc_c16snrtomwcsn**(size_t *__p_maybe_dst_len, ztd_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_mcerr_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_mcerr_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_mcerr **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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **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_mcerr_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_mcerr_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_mcerr **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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **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_mcerr_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_mcerr_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_mcerr **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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **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_mcerr_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_mcerr_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_mcerr **cnc_c32sntomcsn**(size_t *__p_maybe_dst_len, ztd_char_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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **cnc_c32snrtomcsn**(size_t *__p_maybe_dst_len, ztd_char_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_mcerr_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_mcerr_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_mcerr **cnc_c32sntomwcsn**(size_t *__p_maybe_dst_len, ztd_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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **cnc_c32snrtomwcsn**(size_t *__p_maybe_dst_len, ztd_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_mcerr_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_mcerr_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_mcerr **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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **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_mcerr_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_mcerr_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_mcerr **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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **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_mcerr_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_mcerr_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_mcerr **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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **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_mcerr_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_mcerr_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_mcerr **cnc_mcntomcn**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **cnc_mcnrtomcn**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_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_mcerr_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_mcerr_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_mcerr **cnc_mcntomwcn**(size_t *__p_maybe_dst_len, ztd_wchar_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **cnc_mcnrtomwcn**(size_t *__p_maybe_dst_len, ztd_wchar_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_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_mcerr_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_mcerr_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_mcerr **cnc_mcntoc8n**(size_t *__p_maybe_dst_len, ztd_char8_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **cnc_mcnrtoc8n**(size_t *__p_maybe_dst_len, ztd_char8_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_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_mcerr_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_mcerr_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_mcerr **cnc_mcntoc16n**(size_t *__p_maybe_dst_len, ztd_char16_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **cnc_mcnrtoc16n**(size_t *__p_maybe_dst_len, ztd_char16_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_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_mcerr_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_mcerr_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_mcerr **cnc_mcntoc32n**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **cnc_mcnrtoc32n**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_char_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_mcerr_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_mcerr_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_mcerr **cnc_mwcntomcn**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **cnc_mwcnrtomcn**(size_t *__p_maybe_dst_len, ztd_char_t **__p_maybe_dst, size_t *__p_src_len, const ztd_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_mcerr_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_mcerr_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_mcerr **cnc_mwcntomwcn**(size_t *__p_maybe_dst_len, ztd_wchar_t **__p_maybe_dst, size_t *__p_src_len, const ztd_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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **cnc_mwcnrtomwcn**(size_t *__p_maybe_dst_len, ztd_wchar_t **__p_maybe_dst, size_t *__p_src_len, const ztd_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_mcerr_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_mcerr_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_mcerr **cnc_mwcntoc8n**(size_t *__p_maybe_dst_len, ztd_char8_t **__p_maybe_dst, size_t *__p_src_len, const ztd_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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **cnc_mwcnrtoc8n**(size_t *__p_maybe_dst_len, ztd_char8_t **__p_maybe_dst, size_t *__p_src_len, const ztd_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_mcerr_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_mcerr_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_mcerr **cnc_mwcntoc16n**(size_t *__p_maybe_dst_len, ztd_char16_t **__p_maybe_dst, size_t *__p_src_len, const ztd_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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **cnc_mwcnrtoc16n**(size_t *__p_maybe_dst_len, ztd_char16_t **__p_maybe_dst, size_t *__p_src_len, const ztd_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_mcerr_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_mcerr_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_mcerr **cnc_mwcntoc32n**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **cnc_mwcnrtoc32n**(size_t *__p_maybe_dst_len, ztd_char32_t **__p_maybe_dst, size_t *__p_src_len, const ztd_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_mcerr_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_mcerr_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_mcerr **cnc_c8ntomcn**(size_t *__p_maybe_dst_len, ztd_char_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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **cnc_c8nrtomcn**(size_t *__p_maybe_dst_len, ztd_char_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_mcerr_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_mcerr_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_mcerr **cnc_c8ntomwcn**(size_t *__p_maybe_dst_len, ztd_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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **cnc_c8nrtomwcn**(size_t *__p_maybe_dst_len, ztd_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_mcerr_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_mcerr_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_mcerr **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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **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_mcerr_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_mcerr_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_mcerr **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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **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_mcerr_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_mcerr_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_mcerr **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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **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_mcerr_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_mcerr_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_mcerr **cnc_c16ntomcn**(size_t *__p_maybe_dst_len, ztd_char_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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **cnc_c16nrtomcn**(size_t *__p_maybe_dst_len, ztd_char_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_mcerr_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_mcerr_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_mcerr **cnc_c16ntomwcn**(size_t *__p_maybe_dst_len, ztd_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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **cnc_c16nrtomwcn**(size_t *__p_maybe_dst_len, ztd_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_mcerr_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_mcerr_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_mcerr **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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **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_mcerr_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_mcerr_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_mcerr **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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **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_mcerr_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_mcerr_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_mcerr **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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **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_mcerr_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_mcerr_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_mcerr **cnc_c32ntomcn**(size_t *__p_maybe_dst_len, ztd_char_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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **cnc_c32nrtomcn**(size_t *__p_maybe_dst_len, ztd_char_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_mcerr_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_mcerr_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_mcerr **cnc_c32ntomwcn**(size_t *__p_maybe_dst_len, ztd_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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **cnc_c32nrtomwcn**(size_t *__p_maybe_dst_len, ztd_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_mcerr_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_mcerr_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_mcerr **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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **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_mcerr_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_mcerr_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_mcerr **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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **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_mcerr_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_mcerr_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_mcerr **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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **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_mcerr_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_mcerr_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_mcerr err = cnc_cxsntocysn(&required_len, (ztd_char_t**)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_mcerr_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_mcerr_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_mcerr err = cnc_cxsntocysn(&required_len, (ztd_char_t**)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_mcerr_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_mcerr_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.

### 1.3.3 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_err* **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_err* **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_err* **cnc_registry_add**(*cnc_conversion_registry* *__registry, const ztd_char_t *__from, const ztd_char_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_err* **cnc_registry_add_n**(*cnc_conversion_registry* *__registry, size_t __from_size, const ztd_char_t __from[ZTD_PTR_EXTENT(*__from_size*)], size_t __to_size, const ztd_char_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_err* **cnc_registry_add_multi**(*cnc_conversion_registry* *__registry, const ztd_char_t *__from, const ztd_char_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_err* **cnc_registry_add_multi_n**(*cnc_conversion_registry* *__registry, size_t __from_size, const ztd_char_t __from[ZTD_PTR_EXTENT(*__from_size*)], size_t __to_size, const ztd_char_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_err* **cnc_registry_add_single**(*cnc_conversion_registry* *__registry, const ztd_char_t *__from, const ztd_char_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_err* **cnc_registry_add_single_n**(*cnc_conversion_registry* *__registry, size_t __from_size, const ztd_char_t __from[ZTD_PTR_EXTENT(*__from_size*)], size_t __to_size, const ztd_char_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`.

*cnc_open_err* **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_err* **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_err* **cnc_registry_add_multi_c8**(*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_err* **cnc_registry_add_multi_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_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_err* **cnc_registry_add_single_c8**(*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_err* **cnc_registry_add_single_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 *__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.

const *cnc_conversion_heap* ***cnc_registry_heap**(const *cnc_conversion_registry* *__registry)

Retrieves the heap associated with this registry that is used to perform any and all allocations.

**Parameters** **__registry** – **[in]** The conversion registry to retrieve the heap of.

void **cnc_registry_pairs_list_c8n**(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_registry_pairs_list_n**(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.

### cnc_pivot_info

This type's sole purpose is to provide an intermediate buffer for operations that may require it, rather than relying on any internal and implementation-defined buffering or technique to do a conversion. This can allow for optimal conversion rates, especially among bulk conversions performed by the *registry-based transcoding APIs*.

struct **cnc_pivot_info**
A structure containing information for a "pivot buffer".

**Remark**

When a failure happens due to an intermediate conversion failing, the `result` member of the *cnc_pivot_info* will be set to a non-cnc_mcerr_ok value reflecting the type of failure that happened within the intermediate conversion.

#### Public Members

size_t **bytes_size**
The number of bytes pointed to by bytes.

unsigned char ***bytes**
A pointer to a byte buffer to use for intermediate conversions.

**Remark**

If this is a null pointer, it signifies that an internal buffer created in some fashion by the implementation should be used instead to perform the conversion. Otherwise, even if the buffer is insufficiently small, it will use this buffer.

cnc_mcerr **error**
The error code representing any failed conversion specific to the intermediate/pivot step.

**Remark**

If a conversion involving the intermediate buffer / pivot buffer - even the implementation-defined one - fails for any reason, it will be reported in this variable.

### 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_err* **cnc_conv_open**(*cnc_conversion_registry* *__registry, const ztd_char_t *__from, const ztd_char_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_err* **cnc_conv_open_n**(*cnc_conversion_registry* *__registry, size_t __from_size, const ztd_char_t __from[ZTD_PTR_EXTENT(*__from_size*)], size_t __to_size, const ztd_char_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_err* **`cnc_conv_open_select`**(*cnc_conversion_registry* *__registry, const ztd_char_t *__from, const ztd_char_t *__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_err* **cnc_conv_open_select_n**(*cnc_conversion_registry* *__registry, size_t __from_size, const ztd_char_t __from[ZTD_PTR_EXTENT(*__from_size*)], size_t __to_size, const ztd_char_t __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_err* **cnc_conv_new**(*cnc_conversion_registry* *__registry, const ztd_char_t *__from, const ztd_char_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_err* **cnc_conv_new_n**(*cnc_conversion_registry* *__registry, size_t __from_size, const ztd_char_t __from[ZTD_PTR_EXTENT(*__from_size*)], size_t __to_size, const ztd_char_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_err* **cnc_conv_new_select**(*cnc_conversion_registry* *__registry, const ztd_char_t *__from, const ztd_char_t *__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`** – **[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 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_err* **cnc_conv_new_select_n**(*cnc_conversion_registry* *__registry, size_t __from_size, const ztd_char_t __from[ZTD_PTR_EXTENT(*__from_size*)], size_t __to_size, const ztd_char_t __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_err* **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_err* **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_err* **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_err* **cnc_conv_open_select_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_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_err* **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_err* **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_err* **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** – **[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 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_err* **cnc_conv_new_select_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_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

The conversion functions are split into single conversion (contains a `_one` in the name), which perform one unit of indivisible work, and those that are "multi" conversions (without the `_one` in the name), which do it in bulk. There is also some functions to check the status of the

bool **cnc_conv_state_is_complete**(const *cnc_conversion* *__conversion)

Checks if any associated stored state has no outstanding data left.

**Parameters** `__conversion` – **[in]** The cnc_conversion handle to check for state completion. Shall not be `nullptr`.

### Single Conversion Functions

cnc_mcerr **cnc_conv_one_pivot**(*cnc_conversion* *__conversion, size_t *__p_output_bytes_size, unsigned char **__p_output_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

__p_output_bytes serves as a way to validate the input for one completely unit of work. Providing only __p_output_bytes but not __p_output_bytes_size is a way to indicate that the output space is sufficiently large for the input space. Providing __p_output_bytes_size but not __p_output_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.
- **__p_output_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 __p_output_bytes is not nullptr).
- **__p_output_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 __p_output_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_mcerr_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_mcerr_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_mcerr_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_mcerr **cnc_conv_one**(*cnc_conversion* *__conversion, size_t *__p_output_bytes_size, unsigned char **__p_output_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 __p_output_bytes serves as a way to validate the input for one completely unit of work. Providing only __p_output_bytes but not __p_output_bytes_size is a way to indicate that the output space is sufficiently large for the input space. Providing __p_output_bytes_size but not __p_output_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.
- **__p_output_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 __p_output_bytes is not nullptr).

- **__p_output_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 `__p_output_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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **cnc_conv_one_count_pivot**(*cnc_conversion* *__conversion, size_t *__p_output_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 `__p_output_bytes` argument sent to `nullptr`.

**Parameters**

- **__conversion** – **[in]** The cnc_conversion handle indicating the format to be used. Shall not be `nullptr`.
- **__p_output_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_mcerr_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_mcerr_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_mcerr_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_mcerr **cnc_conv_one_count**(*cnc_conversion* *__conversion, size_t *__p_output_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 `__p_output_bytes` argument sent to `nullptr`.

**Parameters**

- **__conversion** – **[in]** The cnc_conversion handle indicating the format to be used. Shall not be `nullptr`.
- **__p_output_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_mcerr_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_mcerr_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 `__p_output_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_mcerr_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 `__p_output_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_mcerr **cnc_conv_one_unbounded_pivot**(*cnc_conversion* *__conversion, unsigned char **__p_output_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 `__p_output_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_output_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 `__p_output_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_mcerr_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_mcerr **cnc_conv_one_unbounded**(*cnc_conversion* *__conversion, unsigned char **__p_output_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 `__p_output_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_output_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 `__p_output_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.

### Multi ("Bulk") Conversion Functions

cnc_mcerr **cnc_conv_pivot**(*cnc_conversion* *__conversion, size_t *__p_output_bytes_size, unsigned char **__p_output_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 `__p_output_bytes` serves as a way to validate the input. Providing only `__p_output_bytes` but not `__p_output_bytes_size` is a way to indicate that the output space is sufficiently large for the input space. Providing `__p_output_bytes_size` but not `__p_output_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_mcerr_insufficient_output errors unrelated to the actual `__p_output_bytes` buffer passed into the function.

---

**Parameters**

- **__conversion** – **[in]** The cnc_conversion handle indicating the format to be used. Shall not be `nullptr`.
- **__p_output_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 `__p_output_bytes` is not `nullptr`).
- **__p_output_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 `__p_output_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_mcerr_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_mcerr_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_mcerr_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_mcerr **cnc_conv**(*cnc_conversion* *__conversion, size_t *__p_output_bytes_size, unsigned char **__p_output_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 `__p_output_bytes` serves as a way to validate the input. Providing only `__p_output_bytes` but not `__p_output_bytes_size` is a way to indicate that the output space is sufficiently large for the input space. Providing `__p_output_bytes_size` but not `__p_output_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`.
- **__p_output_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 `__p_output_bytes` is not `nullptr`).
- **__p_output_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 `__p_output_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_mcerr_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_mcerr_ok is returned.

cnc_mcerr **cnc_conv_count_pivot**(*cnc_conversion* *__conversion, size_t *__p_output_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 `__p_output_bytes` argument sent to `nullptr`.

---

**Parameters**

- **__conversion** – **[in]** The cnc_conversion handle indicating the format to be used. Shall not be nullptr.
- **__p_output_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_mcerr_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_mcerr_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_mcerr_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_mcerr **cnc_conv_count**(*cnc_conversion* *__conversion, size_t *__p_output_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 __p_output_bytes argument sent to nullptr.

**Parameters**

- **__conversion** – **[in]** The cnc_conversion handle indicating the format to be used. Shall not be nullptr.
- **__p_output_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_mcerr_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_mcerr_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 __p_output_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_mcerr_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 `__p_output_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_mcerr **cnc_conv_unbounded_pivot**(*cnc_conversion* *__conversion, unsigned char **__p_output_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 `__p_output_bytes` serves as a way to validate the input. Providing only `__p_output_bytes` but not `__p_output_bytes_size` is a way to indicate that the output space is sufficiently large for the input space. Providing `__p_output_bytes_size` but not `__p_output_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`.
- **__p_output_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 `__p_output_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_mcerr_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_mcerr_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_mcerr_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_mcerr **cnc_conv_unbounded**(*cnc_conversion* *__conversion, unsigned char **__p_output_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 `__p_output_bytes` serves as a way to validate the input. Providing only `__p_output_bytes` but not `__p_output_bytes_size` is a way to indicate that the output space is sufficiently large for the input space. Providing `__p_output_bytes_size` but not `__p_output_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`.
- **__p_output_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 `__p_output_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_mcerr_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_mcerr_ok is returned.

### Conversion Heap and Functions

For the type-erased, registry-based conversions, a heap is used to perform all allocations to ensure the end-user is in control of how much memory is ultimately used, even if they do not have full control over the type-erased data structures. To this end, the *cnc_conversion_heap* is used, and it is described below in the convenience functions which will pass through the designated `user_data` and similar to the specified heap's given functionality.

The default heap uses `malloc`, `free`, and `realloc` for its implementation, ignoring related alignment and user data parameters. It also performs no attempted shrinking or expanding of any given allocation, single ignoring all parameters and following the specification for performing no action.

### Function Calls

void ***cnc_heap_allocate**(*cnc_conversion_heap* *__heap, size_t __requested_size, size_t __alignment, size_t *__p_actual_size)

Attempts to allocate memory from the given `__heap`. Calls the `__heap`'s `allocate` function, using the `user_data` member from the given `__heap` for the last `void*` parameter.

If `0` is used for the `__requested_size` and `nullptr` is returned, then the implementation does not support zero-sized objects and the user of this functinon shall assume that this is an error case and respond identically as if a non-zero `__requested_size` is used and `nullptr` was returned.

---

**Remark**

`0` is a valid value for the `__requested_size`. If `__requested_size` is zero, then a so-called "zero sized object" may be allocated that may also be "freed" (in whatever fashion, including a potential no-op) using a matching call to `cnc_heap_deallocate`. Every allocation must have a separate and distinct address, except for zero-sized objects and zero-sized allocations. They may not have distinct addresses, even within the same address space (e.g., using a so-called sentinel object that marks zero-sized allocations).

---

**Parameters**

- **`__heap`** – **[inout]** The heap to allocate from.
- **`__requested_size`** – **[in]** The newly requested size for the given pointer.
- **`__alignment`** – **[in]** The alignment of the given pointer.
- **`__p_actual_size`** – **[inout]** An output parameter for the size of the allocated pointer. If this function cannot allocate, this will be set to 0. Otherwise, it will be the size of the new memory region in bytes.

**Returns** A pointer to the newly created space of at least `__requested_size` bytes. It may be larger than request, but it must not be smaller than requested. The real size is stored in `*__p_actual_size`. If the allocation failed, or the allocator does not support 0-sized objects (in the case of a `__requested_size` of 0), then the returned pointer is shall be `nullptr`.

void **cnc_heap_deallocate**(*cnc_conversion_heap* *__heap, void *__ptr, size_t __ptr_size, size_t __alignment)

Deallocates the given memory using the provided heap. Calls the `__heap`'s `deallocate` function, using the `user_data` member from the given `__heap` for the last `void*` parameter.

---

**Remark**

The allocation must have come from this heap. If it does, then it is an error and the behavior after such an error is committed is undefined. The memory region of the bytes denoted by [__ptr, __ptr + __ptr_size) can no longer be accessed after this function returns; doing so is an error and the behavior after such an error is committed is undefined.

---

**Parameters**

- **__heap** – **[inout]** The heap from which __ptr was allocated from and manipulated with.
- **__ptr** – **[in]** The original pointer to deallocate.
- **__ptr_size** – **[in]** The size of the allocation for the original pointer.
- **__alignment** – **[in]** The alignment of the given pointer. Not necessarily required in all cases for all allocators, but must be provided by the user.

void *cnc_heap_reallocate_allocation(*cnc_conversion_heap* *__heap, void *__original, size_t __original_size, size_t __requested_size, size_t __alignment, size_t *__p_actual_size)

Attempts to reallocate an allocation from a given __heap, replacing it. Calls the __heap's reallocate function, using the user_data member from the given __heap for the last void* parameter.

---

**Remark**

If the return value is nullptr, that means that reallocation failed and the original pointer was not deallocated.

---

**Parameters**

- **__heap** – **[inout]** The heap from which __original was allocated from and manipulated with.
- **__original** – **[in]** The original pointer to deallocate.
- **__original_size** – **[in]** The size of the allocation for the original pointer.
- **__requested_size** – **[in]** The newly requested size for the given pointer.
- **__alignment** – **[in]** The alignment of the given pointer. cases.
- **__p_actual_size** – **[inout]** An output parameter for the size of the reallocated pointer. If this function cannot reallocate, this will be set to 0. Otherwise, it will be the size of the new memory region in bytes.

**Returns** A new pointer whose value may be moved and may not be identical to __original. If this function returns nullptr, then the block pointer to be __original may still be used to perform work and may be dereferenced, accessed, compared, and similar within the valid region of [__original, __original + __original_size). Otherwise, the value may not be used in any way, shape, or form and all work must be done with the returned pointer with its byte size indicated by the value of *__p_actual_size.

void *cnc_heap_expand_allocation(*cnc_conversion_heap* *__heap, void *__original, size_t __original_size, size_t __alignment, size_t __expand_left, size_t __expand_right, size_t *__p_actual_size)

Attempts to expand the given allocation from the provided __heap. Calls the __heap's expand function, using the user_data member from the given __heap for the last void* parameter.

Commonly, `__expand_left` is set to zero while `__expand_right` contains the amount to expand the allocation by. If `__expand_left` is not set to zero, then this function may move memory from its original starting location to the new starting location VIA a careful overlapping copy. Expansion is a request that allows an implementation to:

---

**Remark**

If the return is `nullptr`, then the memory region of the bytes spanning [`__original`, `__original + __original_size`) may be accessed. If the return is not `nullptr`, then the memory region of the bytes spanning [`__original`, `__original + __original_size`) may not be accessed in any way, shape, or form, and its new size is stored in `*__p_actual_size` at an address specified by the return value.

---

- expand to the exact specified region;
- or, expand to a larger region but that is at least as large as what is requested.

An implementation may expand any further than what was requested.

If `__expand_left` is zero and `__expand_right` is zero, then the function sets `*__p_actual_size` to 0, returns `nullptr`, and takes no other heap-related action.

**Parameters**

- **`__heap`** – **[inout]** The heap from which `__original` was allocated from and manipulated with.
- **`__original`** – **[in]** The original pointer to expand.
- **`__original_size`** – **[in]** The size of the allocation for the original pointer, in bytes.
- **`__alignment`** – **[in]** The alignment for `__original`.
- **`__expand_left`** – **[in]** The amount to grow the "left" (start base `__original` value, extending outwards by further decrementing the `__original` start pointer value) by, in bytes.
- **`__expand_right`** – **[in]** The amount to grow the "right" (end base `__original + __original_size` value, extending outwards by further incrementing `__original + __original_size` end pointer value) by, in bytes.
- **`__p_actual_size`** – **[inout]** An output parameter for the size of the expanded pointer. If this function cannot expand, this will be set to 0. Otherwise, it will be the size of the new memory region in bytes.

**Returns** A new pointer of the shrank memory region. If this is `nullptr`, then no expansion has occurred.

void *`cnc_heap_shrink_allocation`(*cnc_conversion_heap* *__heap, void *__original, size_t __original_size, size_t __alignment, size_t __reduce_left, size_t __reduce_right, size_t *__p_actual_size)

Attempts to shrink the given allocation from the provided `__heap`. Calls the `__heap`'s `shink` function, using the `user_data` member from the given `__heap` for the last `void*` parameter.

Commonly, `__reduce_left` is set to zero while `__reduce_right` contains the amount to shrink the allocation by. If `__reduce_left` is not set to zero, then this function may move memory from its original starting location to the new starting location VIA a careful overlapping copy. Shrinking is a request that allows an implementation to:

**Remark**

If the return is `nullptr`, then the memory region of the bytes spanning [`__original`, `__original + __original_size`) may be accessed. If the return is not `nullptr`, then the memory region of the bytes spanning [`__original`, `__original + __original_size`) may not be accessed in any way, shape, or form, and its new size is stored in `*__p_actual_size` at an address specified by the return value.

- shrink to the exact specified region;
- or, shrink to a smaller region that at least as large as what is requested.

An implementation must not shrink any further than what was requested and it is a violation of this API's conditions if it does.

If `__reduce_left` is zero and `__reduce_right` is non-zero, is it not allowed for an implementation to do nothing but then tell the implementation that the new size is smaller than the original without affecting the allocation in some fashion. An implementation that does nothing must return `nullptr` and set `*__p_actual_size` to `0`.

If `__reduce_left` is zero and `__reduce_right` is zero, then the function sets `*__p_actual_size` to 0, returns `nullptr`, and takes no other heap-related action.

**Parameters**

- **`__heap`** – **[inout]** The heap from which `__original` was allocated from and manipulated with.
- **`__original`** – **[in]** The original pointer to shrink.
- **`__original_size`** – **[in]** The size of the allocation for the original pointer, in bytes.
- **`__alignment`** – **[in]** The alignment for `__original`.
- **`__reduce_left`** – **[in]** The amount to reduce the "left" (start base `__original` value, pulling inwards by further incrementing the `__original` start pointer value) by, in bytes.
- **`__reduce_right`** – **[in]** The amount to reduce the "right" (end base `__original + __original_size` value, pulling inwards by further decrementing the `__original + __original_size` end pointer value) by, in bytes.
- **`__p_actual_size`** – **[inout]** An output parameter for the size of the shrunk pointer. If this function cannot shrink, this will be set to 0. Otherwise, it will be the size of the new memory region in bytes.

**Returns** A new pointer of the shrank memory region. If this is `nullptr`, then no shrinking has occurred and `*__p_actual_size` is set to `0`.

### Heap Object

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_heap_allocate_function_ptr* __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. See *cnc_heap_allocate* for more information.

*cnc_heap_deallocate_function_ptr* __deallocate__
The allocation deallocate function. It takes the original pointer, its size, its alignment, and a userdata passed in during heap creation. See *cnc_heap_deallocate* for more information. See cnc_heap_deallocate for more information.

*cnc_heap_reallocate_allocation_function_ptr* __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, if successful. See *cnc_heap_reallocate_allocation* for more information.

*cnc_heap_expand_allocation_function_ptr* __expand__
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 pointer. It returns the new pointer and writes out the new size, if successful. See *cnc_heap_expand_allocation* for more information.

*cnc_heap_shrink_allocation_function_ptr* __shrink__
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 pointer. It returns the new pointer and writes out the new size, if successful. See *cnc_heap_shrink_allocation* for more information.

### Types

typedef void *(*__cnc_heap_allocate_function_ptr__)(size_t __requested_size, size_t __alignment, size_t *__p_actual_size, void *__user_data)
A heap allocate function type. For a description of the parameters, see *cnc_heap_allocate*.

typedef void (*__cnc_heap_deallocate_function_ptr__)(void *__ptr, size_t __ptr_size, size_t __alignment, void *__user_data)
A heap deallocate function type. For a description of the parameters, see *cnc_heap_deallocate*.

typedef void *(*__cnc_heap_reallocate_allocation_function_ptr__)(void *__original, size_t __original_size, size_t __requested_size, size_t __alignment, size_t *__p_actual_size, void *__user_data)
A heap reallocate function type. For a description of the parameters, see *cnc_heap_allocate*.

typedef void *(*__cnc_heap_expand_allocation_function_ptr__)(void *__original, size_t __original_size, size_t __alignment, size_t __expand_left, size_t __expand_right, size_t *__p_actual_size, void *__user_data)
A heap allocation expand function type. For a description of the parameters, see *cnc_heap_expand_allocation*.

typedef void *(*cnc_heap_shrink_allocation_function_ptr)(void *__original, size_t __original_size, size_t __alignment, size_t __reduce_left, size_t __reduce_right, size_t *__p_actual_size, void *__user_data)

A heap allocation shrink function type. For a description of the parameters, see *cnc_heap_shrink_allocation*.

## 1.4 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.

## 1.5 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!

### 1.5.1 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.

### 1.5.2 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).

## 1.6 Benchmarks

All benchmarks are done in conjunction with the ztd.text library, and can be found there.

## 1.7 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!

### 1.7.1 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.

### 1.7.2 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!)

## 1.8 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.

**simdutf** Daniel Lemire et. al. "simdutf: Unicode at Gigabytes per Second." January 28th, 2022. URL: https://github.com/simdutf/simdutf.

CHAPTER

TWO

# INDICES & SEARCH

## 2.1 Index

# INDEX

## C


character, **155**
CNC_C16_MAX (*C macro*), 27
cnc_c16nrtoc16n (*C++ function*), 106
cnc_c16nrtoc32n (*C++ function*), 107
cnc_c16nrtoc8n (*C++ function*), 105
cnc_c16nrtomcn (*C++ function*), 103
cnc_c16nrtomwcn (*C++ function*), 104
cnc_c16ntoc16n (*C++ function*), 106
cnc_c16ntoc32n (*C++ function*), 107
cnc_c16ntoc8n (*C++ function*), 105
cnc_c16ntomcn (*C++ function*), 102
cnc_c16ntomwcn (*C++ function*), 103
cnc_c16snrtoc16sn (*C++ function*), 77
cnc_c16snrtoc32sn (*C++ function*), 79
cnc_c16snrtoc8sn (*C++ function*), 76
cnc_c16snrtomcsn (*C++ function*), 74
cnc_c16snrtomwcsn (*C++ function*), 75
cnc_c16sntoc16sn (*C++ function*), 77
cnc_c16sntoc32sn (*C++ function*), 78
cnc_c16sntoc8sn (*C++ function*), 76
cnc_c16sntomcsn (*C++ function*), 73
cnc_c16sntomwcsn (*C++ function*), 75
CNC_C32_MAX (*C macro*), 27
cnc_c32nrtoc16n (*C++ function*), 112
cnc_c32nrtoc32n (*C++ function*), 113
cnc_c32nrtoc8n (*C++ function*), 111
cnc_c32nrtomcn (*C++ function*), 109
cnc_c32nrtomcn_ascii (*C++ function*), 32
cnc_c32nrtomcn_atari_st (*C++ function*), 33
cnc_c32nrtomcn_atascii (*C++ function*), 33
cnc_c32nrtomcn_big5_hkscs (*C++ function*), 30
cnc_c32nrtomcn_gb18030 (*C++ function*), 31
cnc_c32nrtomcn_gbk (*C++ function*), 31
cnc_c32nrtomcn_ibm_1006_urdu (*C++ function*), 36
cnc_c32nrtomcn_ibm_424_hebrew_bulletin (*C++ function*), 34
cnc_c32nrtomcn_ibm_856_hebrew (*C++ function*), 34
cnc_c32nrtomcn_ibm_866_cyrillic (*C++ function*), 35
cnc_c32nrtomcn_iso_8859_1 (*C++ function*), 37
cnc_c32nrtomcn_iso_8859_10 (*C++ function*), 42
cnc_c32nrtomcn_iso_8859_13 (*C++ function*), 42
cnc_c32nrtomcn_iso_8859_14 (*C++ function*), 43
cnc_c32nrtomcn_iso_8859_15 (*C++ function*), 44
cnc_c32nrtomcn_iso_8859_16 (*C++ function*), 44
cnc_c32nrtomcn_iso_8859_1_1985 (*C++ function*), 36
cnc_c32nrtomcn_iso_8859_1_1998 (*C++ function*), 37
cnc_c32nrtomcn_iso_8859_2 (*C++ function*), 38
cnc_c32nrtomcn_iso_8859_3 (*C++ function*), 38
cnc_c32nrtomcn_iso_8859_4 (*C++ function*), 39
cnc_c32nrtomcn_iso_8859_5 (*C++ function*), 40
cnc_c32nrtomcn_iso_8859_6 (*C++ function*), 40
cnc_c32nrtomcn_iso_8859_7 (*C++ function*), 41
cnc_c32nrtomcn_iso_8859_8 (*C++ function*), 41
cnc_c32nrtomcn_kamenicky (*C++ function*), 45
cnc_c32nrtomcn_kazakh_strk1048 (*C++ function*), 45
cnc_c32nrtomcn_koi8_r (*C++ function*), 46
cnc_c32nrtomcn_koi8_u (*C++ function*), 46
cnc_c32nrtomcn_petscii_shifted (*C++ function*), 47
cnc_c32nrtomcn_petscii_unshifted (*C++ function*), 47
cnc_c32nrtomcn_punycode (*C++ function*), 28
cnc_c32nrtomcn_shift_jis_x0208 (*C++ function*), 32
cnc_c32nrtomcn_tatar_ansi (*C++ function*), 48
cnc_c32nrtomcn_tatar_ascii (*C++ function*), 48
cnc_c32nrtomcn_windows_1251 (*C++ function*), 51
cnc_c32nrtomcn_windows_1252 (*C++ function*), 51
cnc_c32nrtomcn_windows_1253 (*C++ function*), 52
cnc_c32nrtomcn_windows_1254 (*C++ function*), 52
cnc_c32nrtomcn_windows_1255 (*C++ function*), 53
cnc_c32nrtomcn_windows_1256 (*C++ function*), 54
cnc_c32nrtomcn_windows_1257 (*C++ function*), 54
cnc_c32nrtomcn_windows_1258 (*C++ function*), 55
cnc_c32nrtomcn_windows_437_dos_latin_us (*C++ function*), 49
cnc_c32nrtomcn_windows_865_dos_nordic (*C++ function*), 50
cnc_c32nrtomcn_windows_874 (*C++ function*), 50

cnc_c32nrtomwcn (*C++ function*), 110
cnc_c32ntoc16n (*C++ function*), 111
cnc_c32ntoc32n (*C++ function*), 113
cnc_c32ntoc8n (*C++ function*), 110
cnc_c32ntomcn (*C++ function*), 108
cnc_c32ntomwcn (*C++ function*), 109
cnc_c32snrtoc16sn (*C++ function*), 83
cnc_c32snrtoc32sn (*C++ function*), 84
cnc_c32snrtoc8sn (*C++ function*), 82
cnc_c32snrtomcsn (*C++ function*), 80
cnc_c32snrtomcsn_ascii (*C++ function*), 33
cnc_c32snrtomcsn_atari_st (*C++ function*), 33
cnc_c32snrtomcsn_atascii (*C++ function*), 34
cnc_c32snrtomcsn_big5_hkscs (*C++ function*), 30
cnc_c32snrtomcsn_gb18030 (*C++ function*), 31
cnc_c32snrtomcsn_gbk (*C++ function*), 31
cnc_c32snrtomcsn_ibm_1006_urdu (*C++ function*), 36
cnc_c32snrtomcsn_ibm_424_hebrew_bulletin (*C++ function*), 34
cnc_c32snrtomcsn_ibm_856_hebrew (*C++ function*), 35
cnc_c32snrtomcsn_ibm_866_cyrillic (*C++ function*), 35
cnc_c32snrtomcsn_iso_8859_1 (*C++ function*), 38
cnc_c32snrtomcsn_iso_8859_10 (*C++ function*), 42
cnc_c32snrtomcsn_iso_8859_13 (*C++ function*), 43
cnc_c32snrtomcsn_iso_8859_14 (*C++ function*), 43
cnc_c32snrtomcsn_iso_8859_15 (*C++ function*), 44
cnc_c32snrtomcsn_iso_8859_16 (*C++ function*), 44
cnc_c32snrtomcsn_iso_8859_1_1985 (*C++ function*), 36
cnc_c32snrtomcsn_iso_8859_1_1998 (*C++ function*), 37
cnc_c32snrtomcsn_iso_8859_2 (*C++ function*), 38
cnc_c32snrtomcsn_iso_8859_3 (*C++ function*), 39
cnc_c32snrtomcsn_iso_8859_4 (*C++ function*), 39
cnc_c32snrtomcsn_iso_8859_5 (*C++ function*), 40
cnc_c32snrtomcsn_iso_8859_6 (*C++ function*), 40
cnc_c32snrtomcsn_iso_8859_7 (*C++ function*), 41
cnc_c32snrtomcsn_iso_8859_8 (*C++ function*), 42
cnc_c32snrtomcsn_kamenicky (*C++ function*), 45
cnc_c32snrtomcsn_kazakh_strk1048 (*C++ function*), 46
cnc_c32snrtomcsn_koi8_r (*C++ function*), 46
cnc_c32snrtomcsn_koi8_u (*C++ function*), 47
cnc_c32snrtomcsn_petscii_shifted (*C++ function*), 48
cnc_c32snrtomcsn_petscii_unshifted (*C++ function*), 47
cnc_c32snrtomcsn_punycode (*C++ function*), 28
cnc_c32snrtomcsn_shift_jis_x0208 (*C++ function*), 32
cnc_c32snrtomcsn_tatar_ansi (*C++ function*), 48
cnc_c32snrtomcsn_tatar_ascii (*C++ function*), 49
cnc_c32snrtomcsn_windows_1251 (*C++ function*), 51
cnc_c32snrtomcsn_windows_1252 (*C++ function*), 52
cnc_c32snrtomcsn_windows_1253 (*C++ function*), 52
cnc_c32snrtomcsn_windows_1254 (*C++ function*), 53
cnc_c32snrtomcsn_windows_1255 (*C++ function*), 53
cnc_c32snrtomcsn_windows_1256 (*C++ function*), 54
cnc_c32snrtomcsn_windows_1257 (*C++ function*), 54
cnc_c32snrtomcsn_windows_1258 (*C++ function*), 55
cnc_c32snrtomcsn_windows_437_dos_latin_us (*C++ function*), 49
cnc_c32snrtomcsn_windows_865_dos_nordic (*C++ function*), 50
cnc_c32snrtomcsn_windows_874 (*C++ function*), 50
cnc_c32snrtomwcsn (*C++ function*), 81
cnc_c32sntoc16sn (*C++ function*), 83
cnc_c32sntoc32sn (*C++ function*), 84
cnc_c32sntoc8sn (*C++ function*), 82
cnc_c32sntomcsn (*C++ function*), 79
cnc_c32sntomwcsn (*C++ function*), 80
CNC_C8_MAX (*C macro*), 27
cnc_c8nrtoc16n (*C++ function*), 101
cnc_c8nrtoc32n (*C++ function*), 102
cnc_c8nrtoc8n (*C++ function*), 99
cnc_c8nrtomcn (*C++ function*), 97
cnc_c8nrtomwcn (*C++ function*), 98
cnc_c8ntoc16n (*C++ function*), 100
cnc_c8ntoc32n (*C++ function*), 101
cnc_c8ntoc8n (*C++ function*), 99
cnc_c8ntomcn (*C++ function*), 97
cnc_c8ntomwcn (*C++ function*), 98
cnc_c8snrtoc16sn (*C++ function*), 72
cnc_c8snrtoc32sn (*C++ function*), 73
cnc_c8snrtoc8sn (*C++ function*), 70
cnc_c8snrtomcsn (*C++ function*), 68
cnc_c8snrtomwcsn (*C++ function*), 69
cnc_c8sntoc16sn (*C++ function*), 71
cnc_c8sntoc32sn (*C++ function*), 72
cnc_c8sntoc8sn (*C++ function*), 70
cnc_c8sntomcsn (*C++ function*), 68
cnc_c8sntomwcsn (*C++ function*), 69
cnc_conv (*C++ function*), 146
cnc_conv_close (*C++ function*), 139
cnc_conv_count (*C++ function*), 147
cnc_conv_count_pivot (*C++ function*), 146
cnc_conv_delete (*C++ function*), 140
cnc_conv_is_valid (*C++ function*), 148
cnc_conv_is_valid_pivot (*C++ function*), 147
cnc_conv_new (*C++ function*), 132
cnc_conv_new_c8 (*C++ function*), 137
cnc_conv_new_c8_select (*C++ function*), 138
cnc_conv_new_c8n (*C++ function*), 138
cnc_conv_new_n (*C++ function*), 133
cnc_conv_new_select (*C++ function*), 133

cnc_conv_new_select_c8n (*C++ function*), 139
cnc_conv_new_select_n (*C++ function*), 134
cnc_conv_one (*C++ function*), 141
cnc_conv_one_count (*C++ function*), 142
cnc_conv_one_count_pivot (*C++ function*), 142
cnc_conv_one_is_valid (*C++ function*), 143
cnc_conv_one_is_valid_pivot (*C++ function*), 143
cnc_conv_one_pivot (*C++ function*), 140
cnc_conv_one_unbounded (*C++ function*), 144
cnc_conv_one_unbounded_pivot (*C++ function*), 144
cnc_conv_open (*C++ function*), 130
cnc_conv_open_c8 (*C++ function*), 135
cnc_conv_open_c8_select (*C++ function*), 136
cnc_conv_open_c8n (*C++ function*), 135
cnc_conv_open_n (*C++ function*), 130
cnc_conv_open_select (*C++ function*), 131
cnc_conv_open_select_c8n (*C++ function*), 137
cnc_conv_open_select_n (*C++ function*), 132
cnc_conv_pivot (*C++ function*), 145
cnc_conv_state_is_complete (*C++ function*), 140
cnc_conv_unbounded (*C++ function*), 149
cnc_conv_unbounded_pivot (*C++ function*), 148
cnc_conversion (*C++ type*), 130
cnc_conversion_heap (*C++ struct*), 153
cnc_conversion_heap::allocate (*C++ member*), 154
cnc_conversion_heap::deallocate (*C++ member*), 154
cnc_conversion_heap::expand (*C++ member*), 154
cnc_conversion_heap::reallocate (*C++ member*), 154
cnc_conversion_heap::shrink (*C++ member*), 154
cnc_conversion_heap::user_data (*C++ member*), 154
cnc_conversion_info (*C++ struct*), 127
cnc_conversion_info::from_code_data (*C++ member*), 128
cnc_conversion_info::from_code_size (*C++ member*), 128
cnc_conversion_info::indirect_code_data (*C++ member*), 128
cnc_conversion_info::indirect_code_size (*C++ member*), 128
cnc_conversion_info::is_indirect (*C++ member*), 128
cnc_conversion_info::to_code_data (*C++ member*), 128
cnc_conversion_info::to_code_size (*C++ member*), 128
cnc_conversion_registry (*C++ type*), 116
cnc_cxnrtocyn (*C macro*), 115
cnc_cxntocyn (*C macro*), 115
cnc_cxsnrtocysn (*C macro*), 114
cnc_cxsntocysn (*C macro*), 114
cnc_heap_allocate (*C++ function*), 150
cnc_heap_allocate_function_ptr (*C++ type*), 154
cnc_heap_deallocate (*C++ function*), 150
cnc_heap_deallocate_function_ptr (*C++ type*), 154
cnc_heap_expand_allocation (*C++ function*), 151
cnc_heap_expand_allocation_function_ptr (*C++ type*), 154
cnc_heap_reallocate_allocation (*C++ function*), 151
cnc_heap_reallocate_allocation_function_ptr (*C++ type*), 154
cnc_heap_shrink_allocation (*C++ function*), 152
cnc_heap_shrink_allocation_function_ptr (*C++ type*), 154
CNC_MC_MAX (*C macro*), 27
cnc_mcerr_incomplete_input (*C++ enumerator*), 24
cnc_mcerr_insufficient_output (*C++ enumerator*), 24
cnc_mcerr_invalid_sequence (*C++ enumerator*), 24
cnc_mcerr_ok (*C++ enumerator*), 24
cnc_mcerr_to_str (*C++ function*), 24
cnc_mcnrtoc16n (*C++ function*), 89
cnc_mcnrtoc32n (*C++ function*), 90
cnc_mcnrtoc32n_ascii (*C++ function*), 32
cnc_mcnrtoc32n_atari_st (*C++ function*), 33
cnc_mcnrtoc32n_atascii (*C++ function*), 33
cnc_mcnrtoc32n_big5_hkscs (*C++ function*), 30
cnc_mcnrtoc32n_gb18030 (*C++ function*), 30
cnc_mcnrtoc32n_gbk (*C++ function*), 31
cnc_mcnrtoc32n_ibm_1006_urdu (*C++ function*), 35
cnc_mcnrtoc32n_ibm_424_hebrew_bulletin (*C++ function*), 34
cnc_mcnrtoc32n_ibm_856_hebrew (*C++ function*), 34
cnc_mcnrtoc32n_ibm_866_cyrillic (*C++ function*), 35
cnc_mcnrtoc32n_iso_8859_1 (*C++ function*), 37
cnc_mcnrtoc32n_iso_8859_10 (*C++ function*), 42
cnc_mcnrtoc32n_iso_8859_13 (*C++ function*), 42
cnc_mcnrtoc32n_iso_8859_14 (*C++ function*), 43
cnc_mcnrtoc32n_iso_8859_15 (*C++ function*), 43
cnc_mcnrtoc32n_iso_8859_16 (*C++ function*), 44
cnc_mcnrtoc32n_iso_8859_1_1985 (*C++ function*), 36
cnc_mcnrtoc32n_iso_8859_1_1998 (*C++ function*), 37
cnc_mcnrtoc32n_iso_8859_2 (*C++ function*), 38
cnc_mcnrtoc32n_iso_8859_3 (*C++ function*), 38
cnc_mcnrtoc32n_iso_8859_4 (*C++ function*), 39
cnc_mcnrtoc32n_iso_8859_5 (*C++ function*), 39
cnc_mcnrtoc32n_iso_8859_6 (*C++ function*), 40
cnc_mcnrtoc32n_iso_8859_7 (*C++ function*), 41
cnc_mcnrtoc32n_iso_8859_8 (*C++ function*), 41
cnc_mcnrtoc32n_kamenicky (*C++ function*), 45

cnc_mcnrtoc32n_kazakh_strk1048 (C++ function), 45
cnc_mcnrtoc32n_koi8_r (C++ function), 46
cnc_mcnrtoc32n_koi8_u (C++ function), 46
cnc_mcnrtoc32n_petscii_shifted (C++ function), 47
cnc_mcnrtoc32n_petscii_unshifted (C++ function), 47
cnc_mcnrtoc32n_punycode (C++ function), 28
cnc_mcnrtoc32n_shift_jis_x0208 (C++ function), 31
cnc_mcnrtoc32n_tatar_ansi (C++ function), 48
cnc_mcnrtoc32n_tatar_ascii (C++ function), 48
cnc_mcnrtoc32n_windows_1251 (C++ function), 51
cnc_mcnrtoc32n_windows_1252 (C++ function), 51
cnc_mcnrtoc32n_windows_1253 (C++ function), 52
cnc_mcnrtoc32n_windows_1254 (C++ function), 52
cnc_mcnrtoc32n_windows_1255 (C++ function), 53
cnc_mcnrtoc32n_windows_1256 (C++ function), 53
cnc_mcnrtoc32n_windows_1257 (C++ function), 54
cnc_mcnrtoc32n_windows_1258 (C++ function), 55
cnc_mcnrtoc32n_windows_437_dos_latin_us (C++ function), 49
cnc_mcnrtoc32n_windows_865_dos_nordic (C++ function), 49
cnc_mcnrtoc32n_windows_874 (C++ function), 50
cnc_mcnrtoc8n (C++ function), 88
cnc_mcnrtomcn (C++ function), 86
cnc_mcnrtomwcn (C++ function), 87
cnc_mcntoc16n (C++ function), 89
cnc_mcntoc32n (C++ function), 90
cnc_mcntoc8n (C++ function), 87
cnc_mcntomcn (C++ function), 85
cnc_mcntomwcn (C++ function), 86
cnc_mcsnrtoc16sn (C++ function), 60
cnc_mcsnrtoc32sn (C++ function), 61
cnc_mcsnrtoc32sn_ascii (C++ function), 32
cnc_mcsnrtoc32sn_atari_st (C++ function), 33
cnc_mcsnrtoc32sn_atascii (C++ function), 33
cnc_mcsnrtoc32sn_big5_hkscs (C++ function), 30
cnc_mcsnrtoc32sn_gb18030 (C++ function), 31
cnc_mcsnrtoc32sn_gbk (C++ function), 31
cnc_mcsnrtoc32sn_ibm_1006_urdu (C++ function), 36
cnc_mcsnrtoc32sn_ibm_424_hebrew_bulletin (C++ function), 34
cnc_mcsnrtoc32sn_ibm_856_hebrew (C++ function), 35
cnc_mcsnrtoc32sn_ibm_866_cyrillic (C++ function), 35
cnc_mcsnrtoc32sn_iso_8859_1 (C++ function), 37
cnc_mcsnrtoc32sn_iso_8859_10 (C++ function), 42
cnc_mcsnrtoc32sn_iso_8859_13 (C++ function), 43
cnc_mcsnrtoc32sn_iso_8859_14 (C++ function), 43
cnc_mcsnrtoc32sn_iso_8859_15 (C++ function), 44
cnc_mcsnrtoc32sn_iso_8859_16 (C++ function), 44
cnc_mcsnrtoc32sn_iso_8859_1_1985 (C++ function), 36
cnc_mcsnrtoc32sn_iso_8859_1_1998 (C++ function), 37
cnc_mcsnrtoc32sn_iso_8859_2 (C++ function), 38
cnc_mcsnrtoc32sn_iso_8859_3 (C++ function), 39
cnc_mcsnrtoc32sn_iso_8859_4 (C++ function), 39
cnc_mcsnrtoc32sn_iso_8859_5 (C++ function), 40
cnc_mcsnrtoc32sn_iso_8859_6 (C++ function), 40
cnc_mcsnrtoc32sn_iso_8859_7 (C++ function), 41
cnc_mcsnrtoc32sn_iso_8859_8 (C++ function), 41
cnc_mcsnrtoc32sn_kamenicky (C++ function), 45
cnc_mcsnrtoc32sn_kazakh_strk1048 (C++ function), 45
cnc_mcsnrtoc32sn_koi8_r (C++ function), 46
cnc_mcsnrtoc32sn_koi8_u (C++ function), 46
cnc_mcsnrtoc32sn_petscii_shifted (C++ function), 47
cnc_mcsnrtoc32sn_petscii_unshifted (C++ function), 47
cnc_mcsnrtoc32sn_punycode (C++ function), 28
cnc_mcsnrtoc32sn_shift_jis_x0208 (C++ function), 32
cnc_mcsnrtoc32sn_tatar_ansi (C++ function), 48
cnc_mcsnrtoc32sn_tatar_ascii (C++ function), 49
cnc_mcsnrtoc32sn_windows_1251 (C++ function), 51
cnc_mcsnrtoc32sn_windows_1252 (C++ function), 51
cnc_mcsnrtoc32sn_windows_1253 (C++ function), 52
cnc_mcsnrtoc32sn_windows_1254 (C++ function), 53
cnc_mcsnrtoc32sn_windows_1255 (C++ function), 53
cnc_mcsnrtoc32sn_windows_1256 (C++ function), 54
cnc_mcsnrtoc32sn_windows_1257 (C++ function), 54
cnc_mcsnrtoc32sn_windows_1258 (C++ function), 55
cnc_mcsnrtoc32sn_windows_437_dos_latin_us (C++ function), 49
cnc_mcsnrtoc32sn_windows_865_dos_nordic (C++ function), 50
cnc_mcsnrtoc32sn_windows_874 (C++ function), 50
cnc_mcsnrtoc8sn (C++ function), 59
cnc_mcsnrtomcsn (C++ function), 56
cnc_mcsnrtomwcsn (C++ function), 58
cnc_mcsntoc16sn (C++ function), 59
cnc_mcsntoc32sn (C++ function), 61
cnc_mcsntoc8sn (C++ function), 58
cnc_mcsntomcsn (C++ function), 56
cnc_mcsntomwcsn (C++ function), 57
cnc_mcstate_is_assuming_valid (C++ function), 27
cnc_mcstate_is_complete (C++ function), 27
cnc_mcstate_set_assume_valid (C++ function), 27
cnc_mcstate_t (C++ union), 25
cnc_mcstate_t::__raw_t (C++ struct), 26

cnc_mcstate_t::__raw_t::__paddding (*C++ member*), 26
cnc_mcstate_t::__raw_t::assume_valid (*C++ member*), 26
cnc_mcstate_t::__raw_t::completion_function (*C++ member*), 26
cnc_mcstate_t::__raw_t::indicator (*C++ member*), 26
cnc_mcstate_t::__raw_t::raw_data (*C++ member*), 26
cnc_mcstate_t::cnc_header_t (*C++ struct*), 26
cnc_mcstate_t::cnc_header_t::__assume_valid (*C++ member*), 26
cnc_mcstate_t::cnc_header_t::__paddding (*C++ member*), 26
cnc_mcstate_t::cnc_header_t::indicator (*C++ member*), 26
cnc_mcstate_t::header (*C++ member*), 26
cnc_mcstate_t::raw (*C++ member*), 26
CNC_MWC_MAX (*C macro*), 27
cnc_mwcnrtoc16n (*C++ function*), 95
cnc_mwcnrtoc32n (*C++ function*), 96
cnc_mwcnrtoc8n (*C++ function*), 94
cnc_mwcnrtomcn (*C++ function*), 91
cnc_mwcnrtomwcn (*C++ function*), 93
cnc_mwcntoc16n (*C++ function*), 94
cnc_mwcntoc32n (*C++ function*), 95
cnc_mwcntoc8n (*C++ function*), 93
cnc_mwcntomcn (*C++ function*), 91
cnc_mwcntomwcn (*C++ function*), 92
cnc_mwcsnrtoc16sn (*C++ function*), 66
cnc_mwcsnrtoc32sn (*C++ function*), 67
cnc_mwcsnrtoc8sn (*C++ function*), 65
cnc_mwcsnrtomcsn (*C++ function*), 62
cnc_mwcsnrtomwcsn (*C++ function*), 63
cnc_mwcsntoc16sn (*C++ function*), 65
cnc_mwcsntoc32sn (*C++ function*), 66
cnc_mwcsntoc8sn (*C++ function*), 64
cnc_mwcsntomcsn (*C++ function*), 62
cnc_mwcsntomwcsn (*C++ function*), 63
cnc_open_err (*C++ enum*), 25
cnc_open_err::cnc_open_err_allocation_failure (*C++ enumerator*), 25
cnc_open_err::cnc_open_err_insufficient_output (*C++ enumerator*), 25
cnc_open_err::cnc_open_err_invalid_parameter (*C++ enumerator*), 25
cnc_open_err::cnc_open_err_no_conversion_path (*C++ enumerator*), 25
cnc_open_err::cnc_open_err_ok (*C++ enumerator*), 25
cnc_pivot_info (*C++ struct*), 129
cnc_pivot_info::bytes (*C++ member*), 129
cnc_pivot_info::bytes_size (*C++ member*), 129
cnc_pivot_info::error (*C++ member*), 129
cnc_pny_decode_state_is_assuming_valid (*C++ function*), 30
cnc_pny_decode_state_is_input_complete (*C++ function*), 29
cnc_pny_decode_state_set_assume_valid (*C++ function*), 29
cnc_pny_decode_state_set_input_complete (*C++ function*), 29
cnc_pny_decode_state_set_input_incomplete (*C++ function*), 29
cnc_pny_decode_state_t (*C++ struct*), 29
cnc_pny_encode_state_is_assuming_valid (*C++ function*), 30
cnc_pny_encode_state_is_input_complete (*C++ function*), 29
cnc_pny_encode_state_set_assume_valid (*C++ function*), 29
cnc_pny_encode_state_set_input_complete (*C++ function*), 29
cnc_pny_encode_state_set_input_incomplete (*C++ function*), 29
cnc_pny_encode_state_t (*C++ struct*), 29
cnc_registry_add (*C++ function*), 116
cnc_registry_add_c8 (*C++ function*), 121
cnc_registry_add_c8n (*C++ function*), 122
cnc_registry_add_multi (*C++ function*), 118
cnc_registry_add_multi_c8 (*C++ function*), 122
cnc_registry_add_multi_c8n (*C++ function*), 123
cnc_registry_add_multi_n (*C++ function*), 119
cnc_registry_add_n (*C++ function*), 117
cnc_registry_add_single (*C++ function*), 119
cnc_registry_add_single_c8 (*C++ function*), 124
cnc_registry_add_single_c8n (*C++ function*), 125
cnc_registry_add_single_n (*C++ function*), 120
cnc_registry_close (*C++ function*), 125
cnc_registry_delete (*C++ function*), 126
cnc_registry_heap (*C++ function*), 126
cnc_registry_new (*C++ function*), 116
cnc_registry_open (*C++ function*), 116
cnc_registry_options (*C++ enum*), 127
cnc_registry_options::cnc_registry_options_default (*C++ enumerator*), 127
cnc_registry_options::cnc_registry_options_empty (*C++ enumerator*), 127
cnc_registry_options::cnc_registry_options_none (*C++ enumerator*), 127
cnc_registry_pairs_list_c8n (*C++ function*), 126
cnc_registry_pairs_list_n (*C++ function*), 126
code point, **155**
code unit, **155**
complete unit of work, **155**

## D

decode, 155

## E

encode, 155
encoding, 155
execution encoding, 155

## G

grapheme cluster, 155

## I

iconv, 159
ICU, 159
indivisible unit of work, 156

## L

literal encoding, 156

## M

mojibake, 156

## S

simdutf, 159

## T

transcode, 156

## U

unicode code point, 156
unicode scalar value, 156
UTF-16, 156
UTF-32, 156
UTF-8, 156

## W

wide execution encoding, 156
wide literal encoding, 156