# mpcre2

Generated by Doxygen 1.8.13

# Contents

# Chapter 1

# MPCRE2: An M Interface to Libprcre2

This package provides an M plugin interface to the libpcre2 regular expression C library. That library provides Perl Compatible Regular Expressions and is hosted at `https://www.pcre.org/`. The library is currently on major version 2, and that is the version MPCRE2 supports.

MPCRE2 has been developed on Ubuntu 18.04.4 LTS x86_64 Linux with the GT.M version supplied by the Ubuntu package manager, currently 6.3-003a-2. It should work on any recent versions of GT.M and Linux though that has not been tested. I see no reason why it should not work on 32 bit Linux, or even AIX, but have made no attempt to verify that.

MPCRE2 is Open Source released under a BSD license. Feel free to do with it as you will.

All of the source code for MPCRE2 is contained in the file **mpcre2.c** (p. 9). While this makes for a lengthy file, each function is generally quite short, and doing it this way, with the function prototypes in the same file, allows us to avoid requiring a .h file for the package.

I believe I have provided an M wrapper for every function described in the PCRE2 documentation at `https↩ ://www.pcre.org/current/doc/html/pcre2api.html` with the exception of those functions marked as "obsolete". Some functions are very C specific and would be difficult to use from M without some additional support. I have nonetheless provided M wrappers for these functions, but have not in general provided the extra helper functions that would make them useful. This work is left as an exercise for the reader.

The PCRE2 library makes extensive use of C pointers. The strategy MPCRE2 uses here is to pass pointers in and out of C as strings formatted as decimal integers equivalent to the pointer values. This strategy means these values are visible in M and that an M program could alter these values. Don't do that. With great power comes great responsibility, and it would certainly crash the M runtime.

The mapping of libpcre2 function names to M is straight-forward but ugly. Since M identifiers cannot use underscores and since the libpcre2 function names make extensive use of underscores, an M identifier is constructed by simply removing all of the underscores. Thus, for instance, the M identifer for invoking "pcre2_match_data_free()" is "pcre2matchdatafree".

Each wrapped PCRE2 function is documented in the Doxygen reference manual for MPCRE2. With one exception, the C parameters are mapped one-to-one to M parameters, so if you are familiar with libpcre2, the M mapping should be unsurprisng. The exception is for C parameters which specify the string length of another parameter. Since M strings are length self-identifying, these length parameters have been dropped in the mapping.

# Chapter 2

# Class Index

## 2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

# Chapter 3

# File Index

## 3.1 File List

Here is a list of all documented files with brief descriptions:

# Chapter 4

# Class Documentation

## 4.1 opt_tab Struct Reference

#### Public Attributes

- const char ∗ **name**

  *M string.*
- uint32_t **val**

  *C value mapping.*

### 4.1.1 Detailed Description

This type is used in tables which translate M strings to C macro values

The documentation for this struct was generated from the following file:

- **mpcre2.c**

# Chapter 5

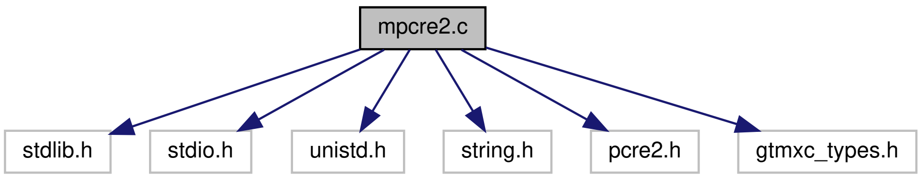
# File Documentation

## 5.1 mpcre2.c File Reference

The mpcre2 plugin provides an M interfact to the C pcre2 regular expression library.

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <pcre2.h>
#include "gtmxc_types.h"
```

Include dependency graph for mpcre2.c:



### Classes

- struct **opt_tab**

### Macros

- #define **PCRE2_CODE_UNIT_WIDTH** 8

  *This macro must be defined before including pcre2.h. It sets our default code unit size to 8 (byte)*

### Typedefs

- typedef struct **opt_tab** **opt_tab_t**

## Functions

- static void ∗ **m_pcre2_malloc** (PCRE2_SIZE size, void ∗memory_data)

    *Pcre2 compatibile wrapper for M malloc()*
- static void **m_pcre2_free** (void ∗ptr, void ∗memory_data)

    *Pcre2 compatible wrapper for M free()*
- static void ∗ **pointer_decode** (char ∗pstr)

    *Decode a string handle into a pointer.*
- static void **pointer_encode** (void ∗ptr, char ∗buf, int size)

    *Encode a pointer as a string handle.*
- static pcre2_general_context ∗ **get_general_context** (char ∗general_context_str)

    *Get or create a general context.*
- static pcre2_compile_context ∗ **get_compile_context** (char ∗context_str, int ∗must_free)

    *Get or create a compile context.*
- static pcre2_match_context ∗ **get_match_context** (char ∗context_str, int ∗must_free)

    *Get or create a match context.*
- static int **parse_pcre2_options** (struct **opt_tab** ∗option_table, int option_count, char ∗opt_tag, char ∗options, uint32_t ∗result)

    *Turn various pcre2 options presented as C strings into integers.*
- gtm_char_t ∗ **mpcre2_get_general_context** (int count)

    *Get the default MPCRE2 general context.*
- void **mpcre2_get_ov_pair** (int count, gtm_char_t ∗ovector_str, gtm_long_t index, gtm_long_t ∗p0, gtm_←
long_t ∗p1)

    *Return an output vector pair.*
- gtm_string_t ∗ **mpcre2_get_mstring_from_buf** (int count, gtm_char_t ∗buffer_ptr_str, gtm_long_t len)

    *Copy a buffer allocated in C to an M string.*
- gtm_long_t **mpcre2_get_substring_list_count** (int count, gtm_char_t ∗listptr_str)

    *Return the number of captured substrings from a substring list.*
- gtm_string_t ∗ **mpcre2_get_mstring_from_substring_list** (int count, gtm_char_t ∗listptr_str, gtm_char_t ∗lenptr_str, gtm_long_t index)

    *Return an M string from a pcre2 substring list.*
- gtm_char_t ∗ **mpcre2_compile** (int count, gtm_string_t ∗pattern, gtm_char_t ∗options, gtm_long_←
t ∗errorcode, gtm_ulong_t ∗erroroffset, gtm_char_t ∗ccontext_str)

    *Wrapper for M calls to pcre2_compile.*
- void **mpcre2_code_free** (int count, gtm_char_t ∗code)

    *Wrap pcre2_code_free()*
- gtm_char_t ∗ **mpcre2_match_data_create** (int count, gtm_long_t ovecsize, gtm_char_t ∗gcontext_str)

    *Wrap the pcre2_match_data_create() function.*
- gtm_char_t ∗ **mpcre2_match_data_create_from_pattern** (int count, gtm_char_t ∗code_str, gtm_char_←
t ∗gcontext_str)

    *Wrap pcre2_match_data_create_from_pattern()*
- gtm_long_t **mpcre2_match** (int count, gtm_char_t ∗code_str, gtm_string_t ∗subject, gtm_long_t startoffset, gtm_char_t ∗options_str, gtm_char_t ∗match_data_str, gtm_char_t ∗mcontext_str)

    *wrap pcre2_match() for M*
- gtm_long_t **mpcre2_dfa_match** (int count, gtm_char_t ∗code_str, gtm_string_t ∗subject, gtm_long_←
t startoffset, gtm_char_t ∗options_str, gtm_char_t ∗match_data_str, gtm_char_t ∗mcontext_str, gtm_long_t wscount)

    *wrap pcre2_dfa_match() for M*
- void **mpcre2_match_data_free** (int count, gtm_char_t ∗match_data_str)

    *Wrap the pcre2_match_data_free() function.*
- gtm_string_t ∗ **mpcre2_get_mark** (int count, gtm_char_t ∗match_data_str)

    *wrap pcre2_get_mark()*

- gtm_long_t **mpcre2_get_ovector_count** (int count, gtm_char_t ∗match_data_str)

  *wrap the pcre2_get_ovector_count() function*
- gtm_char_t ∗ **mpcre2_get_ovector_pointer** (int count, gtm_char_t ∗match_data_str)

  *wrap the pcre2_get_ovector_pointer() function*
- gtm_long_t **mpcre2_get_startchar** (int count, char ∗match_data_str)

  *wrap pcre2_get_startchar()*
- gtm_char_t ∗ **mpcre2_general_context_create** (int count, gtm_char_t ∗malloc_ptr_str, gtm_char_t ∗free←↩ _ptr_str, gtm_char_t ∗data_ptr_str)

  *Wrap the pcre2_general_context_create() function.*
- gtm_char_t ∗ **mpcre2_general_context_copy** (int count, gtm_char_t ∗general_context_str)

  *Wrap the pcre2_general_context_copy() function.*
- void **mpcre2_general_context_free** (int count, gtm_char_t ∗gc_str)

  *Wrap pcre2_general_context_free()*
- gtm_char_t ∗ **mpcre2_compile_context_create** (int count, gtm_char_t ∗gc_str)

  *Wrap the pcre2_compile_context_create() function.*
- gtm_char_t ∗ **mpcre2_compile_context_copy** (int count, gtm_char_t ∗ccontext_str)

  *Wrap the pcre2_compile_context_copy() function.*
- void **mpcre2_compile_context_free** (int count, gtm_char_t ∗ccontext_str)

  *Wrap the pcre2_compile_context_free() function.*
- gtm_long_t **mpcre2_set_bsr** (int count, gtm_char_t ∗ccontext_str, gtm_char_t ∗value_str)

  *Wrap the pcre2_set_bsr() function.*
- gtm_long_t **mpcre2_set_character_tables** (int count, gtm_char_t ∗ccontext_str, gtm_char_t ∗tables_str)

  *Wrap the pcre2_set_character_tables() function.*
- gtm_long_t **mpcre2_set_compile_extra_options** (int count, gtm_char_t ∗ccontext_str, gtm_char_t ∗extra←↩ _options_str)

  *Wrap the pcre2_set_compile_extra_options() function.*
- gtm_long_t **mpcre2_set_max_pattern_length** (int count, gtm_char_t ∗ccontext_str, gtm_long_t value)

  *Wrap the pcre2_set_max_pattern_length() function.*
- gtm_long_t **mpcre2_set_newline** (int count, gtm_char_t ∗ccontext_str, gtm_char_t ∗value_str)

  *Wrap pcre2_set_newline() function.*
- gtm_long_t **mpcre2_set_parens_nest_limit** (int count, gtm_char_t ∗ccontext_str, gtm_long_t value)

  *Wrap the pcre2_set_parens_nest_limit() function.*
- gtm_long_t **mpcre2_set_compile_recursion_guard** (int count, gtm_char_t ∗ccontext_str, gtm_char_←↩ t ∗guard_function_str, gtm_char_t ∗user_data_str)

  *Wrap the pcre2_set_compile_recursion_guard() function.*
- gtm_char_t ∗ **mpcre2_match_context_create** (int count, gtm_char_t ∗gcontext_str)

  *Wrap the pcre2_match_context_create() function.*
- gtm_char_t ∗ **mpcre2_match_context_copy** (int count, gtm_char_t ∗mcontext_str)

  *Wrap the pcre2_match_context_copy() function.*
- void **mpcre2_match_context_free** (int count, gtm_char_t ∗mcontext_str)

  *Wrap the pcre2_match_context_free() function.*
- gtm_long_t **mpcre2_set_callout** (int count, gtm_char_t ∗mcontext_str, gtm_char_t ∗callout_function_str, gtm_char_t ∗callout_data_str)

  *Wrap the pcre2_set_callout() function.*
- gtm_long_t **mpcre2_set_offset_limit** (int count, gtm_char_t ∗mcontext_str, gtm_long_t value)

  *Wrap the pcre2_set_offset_limit() function.*
- gtm_long_t **mpcre2_set_heap_limit** (int count, gtm_char_t ∗mcontext_str, gtm_long_t value)

  *Wrap the pcre2_set_heap_limit() function.*
- gtm_long_t **mpcre2_set_match_limit** (int count, gtm_char_t ∗mcontext_str, gtm_long_t value)

  *Wrap the pcre2_set_match_limit() function.*
- gtm_long_t **mpcre2_set_depth_limit** (int count, gtm_char_t ∗mcontext_str, gtm_long_t value)

*Wrap the pcre2_set_depth_limit() function.*

- gtm_long_t **mpcre2_substring_copy_byname** (int count, gtm_char_t ∗match_data_str, gtm_char_t ∗name, gtm_string_t ∗buffer)

  *Wrap the pcre2_substring_copy_byname() function.*
- gtm_long_t **mpcre2_substring_copy_bynumber** (int count, gtm_char_t ∗match_data_str, gtm_long_t number, gtm_string_t ∗buffer)

  *Wrap the pcre2_substring_copy_bynumber() function.*
- void **mpcre2_substring_free** (int count, gtm_char_t ∗buffer_str)

  *Wrap the void pcre2_substring_free() function.*
- int **mpcre2_substring_get_byname** (int count, gtm_char_t ∗match_data_str, gtm_char_t ∗name, gtm_↩ string_t ∗bufferptr_str, gtm_long_t ∗bufflen)

  *Wrap the pcre2_substring_get_byname() function.*
- int **mpcre2_substring_get_bynumber** (int count, gtm_char_t ∗match_data_str, gtm_long_t number, gtm↩ _string_t ∗bufferptr_str, gtm_long_t ∗bufflen)

  *Wrap the pcre2_substring_get_bynumber() function.*
- gtm_long_t **mpcre2_substring_length_byname** (int count, gtm_char_t ∗match_data_str, gtm_char_↩ t ∗name, gtm_long_t ∗length)

  *Wrap the pcre2_substring_length_byname() function.*
- gtm_long_t **mpcre2_substring_length_bynumber** (int count, gtm_char_t ∗match_data_str, gtm_long_↩ t number, gtm_long_t ∗length)

  *Wrap the pcre2_substring_length_bynumber() function.*
- gtm_long_t **mpcre2_substring_number_from_name** (int count, gtm_char_t ∗code_str, gtm_char_t ∗name)

  *Wrap the pcre2_substring_number_from_name() function.*
- void **mpcre2_substring_list_free** (int count, gtm_char_t ∗list_str)

  *Wrap the pcre2_substring_list_free() function.*
- gtm_long_t **mpcre2_substring_list_get** (int count, gtm_char_t ∗match_data_str, gtm_string_t ∗listptr_str, gtm_string_t ∗lengthsptr_str)

  *Wrap the pcre2_substring_list_get() function.*
- gtm_long_t **mpcre2_substitute** (int count, gtm_char_t ∗code_str, gtm_string_t ∗subject, gtm_long_↩ t startoffset, gtm_char_t ∗options_str, gtm_char_t ∗match_data_str, gtm_char_t ∗mcontext_str, gtm_string↩ _t ∗replacement, gtm_string_t ∗outputbuffer, gtm_long_t ∗outputlengthptr)

  *Wrap the pcre2_substitute() function.*
- gtm_long_t **mpcre2_jit_compile** (int count, gtm_char_t ∗code_str, gtm_char_t ∗options_str)

  *Wrap pcre2_jit_compile()*
- gtm_long_t **mpcre2_jit_match** (int count, gtm_char_t ∗code_str, gtm_string_t ∗subject, gtm_long_t startoffset, gtm_char_t ∗options_str, gtm_char_t ∗match_data_str, gtm_char_t ∗mcontext_str)

  *Wrap the pcre2_jit_match() function.*
- void **mpcre2_jit_free_unused_memory** (int count, gtm_char_t ∗gcontext_str)

  *Wrap the pcre2_jit_free_unused_memory() function.*
- gtm_char_t ∗ **mpcre2_jit_stack_create** (int count, gtm_long_t startsize, gtm_long_t maxsize, gtm_char_t ∗gcontext_str)

  *Wrap the pcre2_jit_stack_create() function.*
- void **mpcre2_jit_stack_assign** (int count, gtm_char_t ∗mcontext_str, gtm_char_t ∗callback_function_str, gtm_char_t ∗callback_data_str)

  *Wrap the pcre2_jit_stack_assign() function.*
- void **mpcre2_jit_stack_free** (int count, gtm_char_t ∗jit_stack_str)

  *Wrap the pcre2_jit_stack_free() function.*
- gtm_long_t **mpcre2_serialize_decode** (int count, gtm_string_t ∗codes_str, gtm_long_t number_of_codes, gtm_char_t ∗bytes_str, gtm_char_t ∗gcontext_str)

  *Wrap the pcre2_serialize_decode() function.*
- gtm_long_t **mpcre2_serialize_encode** (int count, gtm_char_t ∗codes_str, gtm_long_t number_of_codes, gtm_string_t ∗serialized_bytes_str, gtm_long_t ∗serialized_size, gtm_char_t ∗gcontext_str)

*Wrap the pcre2_serialize_encode() function.*
- void **mpcre2_serialize_free** (int count, gtm_char_t ∗bytes_str)

  *Wrap the void pcre2_serialize_free() function.*
- gtm_long_t **mpcre2_serialize_get_number_of_codes** (int count, gtm_char_t ∗bytes_str)

  *Wrap the pcre2_serialize_get_number_of_codes() function.*
- gtm_char_t ∗ **mpcre2_code_copy** (int count, gtm_char_t ∗code_str)

  *Wrap the pcre2_code_copy() function.*
- gtm_char_t ∗ **mpcre2_code_copy_with_tables** (int count, gtm_char_t ∗code_str)

  *Wrap the pcre2_code_copy_with_tables() function.*
- gtm_long_t **mpcre2_get_error_message** (int count, gtm_long_t errorcode, gtm_string_t ∗buffer)

  *Translate a pcre2_compile() error code into a text message.*
- gtm_char_t ∗ **mpcre2_maketables** (int count, gtm_char_t ∗gcontext_str)

  *Wrap the pcre2_maketables() function.*
- gtm_long_t **mpcre2_pattern_info** (int count, gtm_char_t ∗code_str, gtm_char_t ∗what_str, gtm_string_↩ t ∗where)

  *Wrap the pcre2_pattern_info() function.*
- gtm_long_t **mpcre2_callout_enumerate** (int count, gtm_char_t ∗code_str, gtm_char_t ∗callback_str, gtm↩ _char_t ∗user_data_str)

  *Wrap the pcre2_callout_enumerate() function.*

## Variables

- static struct **opt_tab** **compile_opts** [ ]
- static int **n_compile_opts** = sizeof( **compile_opts**) / sizeof(struct **opt_tab**)

  *The number of compile options implemented.*
- static struct **opt_tab** **extra_compile_opts** [ ]
- static int **n_extra_compile_opts** = sizeof( **extra_compile_opts**) / sizeof(struct **opt_tab**)

  *The number of extra compile options implemented.*
- static struct **opt_tab** **match_opts** [ ]
- static int **n_match_opts** = sizeof( **match_opts**) / sizeof(struct **opt_tab**)

  *The number of match and substitute options supported.*
- static struct **opt_tab** **jit_opts** [ ]
- static int **n_jit_opts** = sizeof( **jit_opts**) / sizeof(struct **opt_tab**)

  *The number of JIT options supported.*
- static struct **opt_tab** **bsr_opts** [ ]
- static int **n_bsr_opts** = sizeof( **bsr_opts**) / sizeof(struct **opt_tab**)

  *The number of BSR options supported.*
- static struct **opt_tab** **newline_opts** [ ]
- static int **n_newline_opts** = sizeof( **newline_opts**) / sizeof(struct **opt_tab**)

  *The number of newline options supported.*
- static struct **opt_tab** **info_opts** [ ]
- static int **n_info_opts** = sizeof( **info_opts**) / sizeof(struct **opt_tab**)

  *The number of info options supported.*

### 5.1.1 Detailed Description

The mpcre2 plugin provides an M interfact to the C pcre2 regular expression library.

### 5.1.2 Typedef Documentation

#### 5.1.2.1 opt_tab_t

```
typedef struct opt_tab opt_tab_t
```

This type is used in tables which translate M strings to C macro values

### 5.1.3 Function Documentation

#### 5.1.3.1 get_compile_context()

```
static pcre2_compile_context* get_compile_context (
             char * context_str,
             int * must_free )  [static]
```

Get or create a compile context.

If we are supplied "0" or "NULL", we create a generic compile context. Otherwise we decode the given context.

If we created the context, we set the free flag, otherwise not

**Parameters**

| | |
|---|---|
| *context_str* | The incoming compile context handle |
| *must_free* | Pointer to our free flag |

**Returns**

a pcre2_compile_context pointer or NULL

References get_general_context(), and pointer_decode().

Referenced by mpcre2_compile().

#### 5.1.3.2 get_general_context()

```
static pcre2_general_context* get_general_context (
             char * general_context_str )  [static]
```

Get or create a general context.

PCRE2 uses the pcre2_general_context type as the hook for replacing the system malloc()/free() functions with custom versions. Since we want to play nice with M, we use this hook to install the M supplied versions. The practical effect of this is that the first time a pcre2 function that needs to allocate memory is called, we set up a static pcre2_general context with the custom memory functions.

Future requests for a general context return this one. The exception is if we are passed an encoded handle from M, in which case we decode it as a pointer. We don't actually expect this to happen as we don't provide a mechanism for creating such. Currently the general context created here is not freed, so that's a few bytes the M process will never get back from PCRE2.

This function is based on an example in the GT.M Programmers Guide, and sets us up to use the M malloc() & free() functions, which is recommended for C code called from M. We could use names if we linked to the callin shared library, but that would be an additional dependency.

**Parameters**

| | |
|---|---|
| *general_context_str* | For "0" or "NULL" we create & return (or just return) the static GC. Otherwise decode as a pointer |

**Returns**

A pcre2_general_context pointer, or NULL on failure

References m_pcre2_free(), m_pcre2_malloc(), and pointer_decode().

Referenced by get_compile_context(), get_match_context(), mpcre2_get_general_context(), mpcre2_jit_free_↩ unused_memory(), mpcre2_jit_stack_create(), mpcre2_maketables(), mpcre2_match_context_create(), mpcre2↩ _match_data_create(), mpcre2_match_data_create_from_pattern(), mpcre2_serialize_decode(), and mpcre2_↩ serialize_encode().

#### 5.1.3.3 get_match_context()

```
static pcre2_match_context* get_match_context (
            char * context_str,
            int * must_free )  [static]
```

Get or create a match context.

If we are supplied "0" or "NULL", we create a generic match context. Otherwise we decode the given context.

If we created the context, we set the free flag, otherwise not

**Parameters**

| | |
|---|---|
| *context_str* | The incoming match context handle |
| *must_free* | Pointer to our free flag |

**Returns**

a pcre2_match_context pointer or NULL

References get_general_context(), and pointer_decode().

Referenced by mpcre2_dfa_match(), mpcre2_jit_match(), mpcre2_match(), and mpcre2_substitute().

#### 5.1.3.4 m_pcre2_free()

```
static void m_pcre2_free (
            void * ptr,
            void * memory_data )  [static]
```

Pcre2 compatible wrapper for M free()

This function wraps the GT.M memory deallocator in a PCRE2 signature.

PCRE2 has a scheme for passing an arbitrary data value to its allocator and deallocator functions. This function does not use that data.

The first time we are called, we initialize the free function from the M runtime.

**Parameters**

| | |
|---|---|
| *ptr* | Pointer to memory to free |
| *memory_data* | Unused |

**Returns**

None

Referenced by get_general_context(), and parse_pcre2_options().

#### 5.1.3.5 m_pcre2_malloc()

```
static void* m_pcre2_malloc (
            PCRE2_SIZE size,
            void * memory_data )  [static]
```

Pcre2 compatibile wrapper for M malloc()

GT.M would prefer that any plugin C code use the GT.M memory allocator rather than the system malloc() & free() functions. Mpcre2 tries to do this, though it complicates the code some. This function creates a PCRE2 signature allocator which wraps the GT.M malloc.

PCRE2 has a scheme for passing an arbitrary data value to its allocator and deallocator functions. This function does not store or use that data.

The first time this function is called, it queries the M runtime for the M malloc function.

**Parameters**

| | |
|---|---|
| *size* | Number of bytes to allocate |
| *memory_data* | Unused |

**Returns**

Pointer to allocated memory or NULL on error

Referenced by get_general_context(), and parse_pcre2_options().

#### 5.1.3.6 mpcre2_callout_enumerate()

```
gtm_long_t mpcre2_callout_enumerate (
              int count,
              gtm_char_t * code_str,
              gtm_char_t * callback_str,
              gtm_char_t * user_data_str )
```

Wrap the pcre2_callout_enumerate() function.

This function is of very limited utilty in an M environment as it provides a facility to register callback functions for PCRE2 pattern callouts, but does not provide a facility for creating such callback functions. This could be done with another plugin.

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *code_str* | String handle for a compiled PCRE2 regular expression |
| *callback_str* | String handle for a callback function (which was not created by Mpcre2) |
| *user_data_str* | Handle for arbitrary user data passed to callbacks. |

**Returns**

A bit unclear, but 0 on success and non-zero on error

References pointer_decode().

#### 5.1.3.7 mpcre2_code_copy()

```
gtm_char_t* mpcre2_code_copy (
              int count,
              gtm_char_t * code_str )
```

Wrap the pcre2_code_copy() function.

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *code_str* | A string handle to a pcre2 compiled expression |

**Returns**

A string handle for a copy of the input compiled expression

References pointer_decode(), and pointer_encode().

#### 5.1.3.8 mpcre2_code_copy_with_tables()

```
gtm_char_t* mpcre2_code_copy_with_tables (
             int count,
             gtm_char_t * code_str )
```

Wrap the pcre2_code_copy_with_tables() function.

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *code_str* | A string handel for a pcre2 compiled expression |

**Returns**

A string handle for a copy of the input compiled expression

References pointer_decode(), and pointer_encode().

#### 5.1.3.9 mpcre2_code_free()

```
void mpcre2_code_free (
             int count,
             gtm_char_t * code )
```

Wrap pcre2_code_free()

We pass pcre2 code pointers back and forth to M as text. We need to re-create the code pointer and free it.

**Parameters**

| | |
|---|---|
| *count* | Count of parameters from M API |
| *code* | String value of a code pointer from M |

**Returns**

None

References pointer_decode().

#### 5.1.3.10 mpcre2_compile()

```
gtm_char_t* mpcre2_compile (
             int count,
             gtm_string_t * pattern,
             gtm_char_t * options,
             gtm_long_t * errorcode,
             gtm_ulong_t * erroroffset,
             gtm_char_t * ccontext_str )
```

Wrapper for M calls to pcre2_compile.

This function serves as a wrapper so M can call pcre2_compile(). The parameters are similar to the C call, but since we use an M string, for the pattern, we know the length, and don't have to pass that as a separate parameter.

In general, refer to the PCRE2 documentation for a fuller description of these parameters. The handle returned on success will be a decimal encoded pointer.

**Parameters**

| | |
|---|---|
| *count* | M API supplied count of arguments to this function |
| *pattern* | The regular expression we are compiling |
| *options* | Compile options |
| *errorcode* | Output parameter to return an error if compile fails |
| *erroroffset* | Output parameter indicating the byte offset of a compile failure |
| *ccontext_str* | Compile context for compile, "0" for defaults |

**Returns**

A handle for this compiled pattern or a "0" string on failure

References get_compile_context(), n_compile_opts, parse_pcre2_options(), and pointer_encode().

#### 5.1.3.11 mpcre2_compile_context_copy()

```
gtm_char_t* mpcre2_compile_context_copy (
             int count,
             gtm_char_t * ccontext_str )
```

Wrap the pcre2_compile_context_copy() function.

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *ccontext_str* | String handle for a pcre2 compile context |

**Returns**

String handle for a copy of the given context

References pointer_decode(), and pointer_encode().

#### 5.1.3.12 mpcre2_compile_context_create()

```
gtm_char_t* mpcre2_compile_context_create (
             int count,
             gtm_char_t * gc_str )
```

Wrap the pcre2_compile_context_create() function.

**Parameters**

| | |
|---|---|
| *count* | Parameter count provided by the M API |
| *gc_str* | String handle to a pcre2 general context |

**Returns**

String handle to a pcre2 compile context

References pointer_decode(), and pointer_encode().

#### 5.1.3.13 mpcre2_compile_context_free()

```
void mpcre2_compile_context_free (
             int count,
             gtm_char_t * ccontext_str )
```

Wrap the pcre2_compile_context_free() function.

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *ccontext_str* | String handle for a pcre2 compile context |

**Returns**

None

References pointer_decode().

#### 5.1.3.14 mpcre2_dfa_match()

```
gtm_long_t mpcre2_dfa_match (
             int count,
             gtm_char_t * code_str,
             gtm_string_t * subject,
             gtm_long_t startoffset,
             gtm_char_t * options_str,
             gtm_char_t * match_data_str,
             gtm_char_t * mcontext_str,
             gtm_long_t wscount )
```

wrap pcre2_dfa_match() for M

We bring in "subject" as an M string, so we can have embedded zero bytes. This gives us a length, so we don't have a separate parameter for that. Additionally, there is no way, or reason to pass in a vector of ints for "workspace from M, so we simply accept a count, and allocate that many on the stack.

**Parameters**

| | |
|---|---|
| *count* | Count of parameters from the M API |
| *code_str* | A pcre2 compiled regular expression pointer in string format |
| *subject* | The string to search for matches |
| *startoffset* | The byte offset at which to start the match search |
| *options_str* | Pcre2 match options in string form |
| *match_data_str* | A Pcre2 match data pointer in string format |
| *mcontext_str* | A Pcre2 match context pointer in string format, or "0" |
| *wscount* | Number of entries to create in the workspace vector |

**Returns**

$<$ 0 on error or no match, 0 vector offests too small, else one more than the highest numbered capturing pair that has been set

References get_match_context(), parse_pcre2_options(), and pointer_decode().

#### 5.1.3.15 mpcre2_general_context_copy()

```
gtm_char_t* mpcre2_general_context_copy (
             int count,
             gtm_char_t * general_context_str )
```

Wrap the pcre2_general_context_copy() function.

**Parameters**

| | |
|---|---|
| *count* | Parameter count provided by the M API |
| *general_context_str* | String handle to a pcre2 general context |

**Returns**

A string handle to a copy of the provided general context or "0"

References pointer_decode(), and pointer_encode().

#### 5.1.3.16 mpcre2_general_context_create()

```
gtm_char_t* mpcre2_general_context_create (
             int count,
             gtm_char_t * malloc_ptr_str,
             gtm_char_t * free_ptr_str,
             gtm_char_t * data_ptr_str )
```

Wrap the pcre2_general_context_create() function.

In general M code should never need to call this function, as m_pcre2 creates a hidden general context when needed. Also, there is no way from M to specify custom C level arguments (unless the M runtime has another plugin which returns appropriate handles).

However, if "0" is passed for all three parameters, NULL will be used, and a general context using the system (not M) memory functions will be returned.

**Parameters**

| | |
|---|---|
| *count* | Count of parameters provided by the M API |
| *malloc_ptr_str* | String handle for a pointer to a pcre2 malloc() compatible function |
| *free_ptr_str* | String handle for a pointer to a pcre2 free() compatible function |
| *data_ptr_str* | String handle for a pointer to a data item to be used for marking memory |

**Returns**

A string handle to the created pcre2_general_context (or "0" on failure)

References pointer_decode(), and pointer_encode().

#### 5.1.3.17 mpcre2_general_context_free()

```
void mpcre2_general_context_free (
             int count,
             gtm_char_t * gc_str )
```

Wrap pcre2_general_context_free()

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *gc_str* | String handle to a pcre2 general context |

**Returns**

None

References pointer_decode().

#### 5.1.3.18 mpcre2_get_error_message()

```
gtm_long_t mpcre2_get_error_message (
             int count,
             gtm_long_t errorcode,
             gtm_string_t * buffer )
```

Translate a pcre2_compile() error code into a text message.

This function provides an M interface to translate a pcre2_compile() error code to a string. The interface differs a bit from the C function wrapped as the M string supplies its own (max) length.

**Parameters**

| | |
|---|---|
| *count* | M API supplied argument count |
| *errorcode* | Code to be mapped to a string |
| *buffer* | M string to hold the output message |

#### 5.1.3.19 mpcre2_get_general_context()

```
gtm_char_t* mpcre2_get_general_context (
             int count )
```

Get the default MPCRE2 general context.

MPCRE2 creates a default general context which is used behind the scenes. This general context uses M memory allocation and dallocation. If this is needed on the M side we use this function to access it.

**Parameters**

| | |
|---|---|
| *count* | The parameter count from the M API |

**Returns**

A string handle for the GC

References get_general_context(), and pointer_encode().

#### 5.1.3.20 mpcre2_get_mark()

```
gtm_string_t* mpcre2_get_mark (
             int count,
             gtm_char_t * match_data_str )
```

wrap pcre2_get_mark()

Note this text from the PCRE2 documentation:

```
After a successful match, a partial match
(PCRE2_ERROR_PARTIAL), or a failure to match
(PCRE2_ERROR_NOMATCH), a mark name may be
available. The function pcre2_get_mark()
can be called to access this name, which
can be specified in the pattern by any of
the backtracking control verbs, not just
(*MARK). The same function applies to all
the verbs. It returns a pointer to the
zero-terminated name, which is within the
compiled pattern. If no name is available,
NULL is returned. The length of the name
(excluding the terminating zero) is stored
in the code unit that precedes the name. You
should use this length instead of relying
on the terminating zero if the name might
contain a binary zero.
```

So we are given liberty to index *BACKWARD* from the given string, as unusual as that might seem. In fact we do do this, so we can construct an M string which will still be OK if it contains zero bytes. Since we only handle 8 bit code units, the length of the mark name ipso facto must be <= 255.

**Parameters**

| | |
|---|---|
| *count* | Paramater count from the M API |
| *match_data_str* | A match data handle |

**Returns**

A mark name, or zero length string if none is available

References pointer_decode().

#### 5.1.3.21 mpcre2_get_mstring_from_buf()

```
gtm_string_t* mpcre2_get_mstring_from_buf (
            int count,
            gtm_char_t * buffer_ptr_str,
            gtm_long_t len )
```

Copy a buffer allocated in C to an M string.

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *buffer_ptr_str* | String handle for the buffer pointer |
| *len* | Length of the buffer |

**Returns**

a gtm_string_t with the buffer address and length

References pointer_decode().

#### 5.1.3.22 mpcre2_get_mstring_from_substring_list()

```
gtm_string_t* mpcre2_get_mstring_from_substring_list (
            int count,
            gtm_char_t * listptr_str,
            gtm_char_t * lenptr_str,
            gtm_long_t index )
```

Return an M string from a pcre2 substring list.

This is a utility function to support pcre2_substring_list_get()

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *listptr_str* | String handle for a pcre2 substring list |
| *lenptr_str* | String handle for a pcre2 length list |
| *index* | Index in the list of the string to return |

**Returns**

The given string value as an M string

References pointer_decode().

#### 5.1.3.23 mpcre2_get_ov_pair()

```
void mpcre2_get_ov_pair (
             int count,
             gtm_char_t * ovector_str,
             gtm_long_t index,
             gtm_long_t * p0,
             gtm_long_t * p1 )
```

Return an output vector pair.

Since M cannot directly manipulate output vector pointers, this access function is needed to take the handle and an index and return the queried output vector pair

**Parameters**

| | |
|---|---|
| *count* | Parameter count provided by the M API |
| *ovector_str* | A string handle for a pointer to the output vector pairs |
| *index* | Which pair to return |
| *p0* | Pointer to the first element of the pair to return |
| *p1* | Pointer to the second element of the pair to return |

**Returns**

None

References pointer_decode().

#### 5.1.3.24 mpcre2_get_ovector_count()

```
gtm_long_t mpcre2_get_ovector_count (
             int count,
             gtm_char_t * match_data_str )
```

wrap the pcre2_get_ovector_count() function

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *match_data_str* | String handle for a pcre2 *match_data pointer |

**Returns**

The number of pairs in the output vector for this match

References pointer_decode().

#### 5.1.3.25 mpcre2_get_ovector_pointer()

```
gtm_char_t* mpcre2_get_ovector_pointer (
            int count,
            gtm_char_t * match_data_str )
```

wrap the pcre2_get_ovector_pointer() function

This function is provided, but does not have much direct utility in M, as there is no native way to *use* the returned handle. Instead, the result must be used with the utility function **mpcre2_get_ov_pair()** (p. 25) (pcre2getovpair in M)

**Parameters**

| | |
|---|---|
| *count* | The parameter count from the M API |
| *match_data_str* | String handle for a match data pointer |

**Returns**

A string handle for the output vector pointer

References pointer_decode(), and pointer_encode().

#### 5.1.3.26 mpcre2_get_startchar()

```
gtm_long_t mpcre2_get_startchar (
            int count,
            char * match_data_str )
```

wrap pcre2_get_startchar()

**Parameters**

| | |
|---|---|
| *count* | Parameter count from M API |
| *match_data_str* | Match data handle |

**Returns**

Code unit offset of successful match

References pointer_decode().

#### 5.1.3.27 mpcre2_get_substring_list_count()

```
gtm_long_t mpcre2_get_substring_list_count (
            int count,
            gtm_char_t * listptr_str )
```

Return the number of captured substrings from a substring list.

This is a helper function for pcre2_substring_list_get() and requires that function to be called first to establish the two pointers used here.

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *listptr_str* | String handle for the substring list |

**Returns**

The number of substrings in the list

References pointer_decode().

#### 5.1.3.28 mpcre2_jit_compile()

```
gtm_long_t mpcre2_jit_compile (
            int count,
            gtm_char_t * code_str,
            gtm_char_t * options_str )
```

Wrap pcre2_jit_compile()

**Parameters**

| | |
|---|---|
| *count* | Parameter count from M API |
| *code_str* | Handle for a previously compiled PCRE2 regular expression |
| *options_str* | JIT options |

**Returns**

0 on success $<$ 0 on error

References parse_pcre2_options(), and pointer_decode().

#### 5.1.3.29 mpcre2_jit_free_unused_memory()

```
void mpcre2_jit_free_unused_memory (
            int count,
            gtm_char_t * gcontext_str )
```

Wrap the pcre2_jit_free_unused_memory() function.

If a string handle for a general context is passed, it will be used. If "0" is passed, the internal general context, which used M malloc & free functions will be used.

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *gcontext_str* | A string handle for a general context or "0" for the internal general context |

**Returns**

None

References get_general_context().

#### 5.1.3.30 mpcre2_jit_match()

```
gtm_long_t mpcre2_jit_match (
             int count,
             gtm_char_t * code_str,
             gtm_string_t * subject,
             gtm_long_t startoffset,
             gtm_char_t * options_str,
             gtm_char_t * match_data_str,
             gtm_char_t * mcontext_str )
```

Wrap the pcre2_jit_match() function.

We bring in "subject" as an M string, so we can have embedded zero bytes. This gives us a length, so we don't have a separate parameter for that.

**Parameters**

| | |
|---|---|
| *count* | Count of parameters from the M API |
| *code_str* | A pcre2 JIT compiled regular expression pointer in string format |
| *subject* | The string to search for matches |
| *startoffset* | The byte offset at which to start the match search |
| *options_str* | Pcre2 match options in string form |
| *match_data_str* | A Pcre2 match data pointer in string format |
| *mcontext_str* | A Pcre2 match context pointer in string format, or "0" |

**Returns**

$<$ 0 on error or no match, 0 vector offests too small, else one more than the highest numbered capturing pair that has been set

References get_match_context(), parse_pcre2_options(), and pointer_decode().

#### 5.1.3.31 mpcre2_jit_stack_assign()

```
void mpcre2_jit_stack_assign (
            int count,
            gtm_char_t * mcontext_str,
            gtm_char_t * callback_function_str,
            gtm_char_t * callback_data_str )
```

Wrap the pcre2_jit_stack_assign() function.

The libm_pcre2 library currently provides no way to specify a callback function, but if another plugin creates an appropriate string handle, it may be used. Otherwise, specify "NULL" for the callback, and either a default stack will be used (if callback data is also "NULL") or a stack allocated by pcre2_jit_stack_create will be used (if callback data is not null).

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *mcontext_str* | String handle for a pcre2 match context |
| *callback_function_str* | String handle for a function to allocate a JIT stack, or "NULL". |
| *callback_data_str* | String handle for a pcre2 JIT stack, or "NULL" |

**Returns**

None

References pointer_decode().

#### 5.1.3.32 mpcre2_jit_stack_create()

```
gtm_char_t* mpcre2_jit_stack_create (
            int count,
            gtm_long_t startsize,
            gtm_long_t maxsize,
            gtm_char_t * gcontext_str )
```

Wrap the pcre2_jit_stack_create() function.

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *startsize* | Starting size for the stack |
| *maxsize* | Maximum size for the stack |
| *gcontext_str* | String handle for a pcre2 general context, "NULL" for the default. |

**Returns**

String handle for a pcre2_jit_stack

References get_general_context(), and pointer_encode().

#### 5.1.3.33 mpcre2_jit_stack_free()

```
void mpcre2_jit_stack_free (
             int count,
             gtm_char_t * jit_stack_str )
```

Wrap the pcre2_jit_stack_free() function.

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *jit_stack_str* | String handle for a pcre2 JIT stack |

**Returns**

None

References pointer_decode().

#### 5.1.3.34 mpcre2_maketables()

```
gtm_char_t* mpcre2_maketables (
             int count,
             gtm_char_t * gcontext_str )
```

Wrap the pcre2_maketables() function.

**Parameters**

| | |
|---|---|
| *count* | The parameter count from the M API |
| *gcontext_str* | A string handle for a PCRE2 general context |

References get_general_context(), and pointer_encode().

#### 5.1.3.35 mpcre2_match()

```
gtm_long_t mpcre2_match (
             int count,
             gtm_char_t * code_str,
             gtm_string_t * subject,
             gtm_long_t startoffset,
             gtm_char_t * options_str,
             gtm_char_t * match_data_str,
             gtm_char_t * mcontext_str )
```

wrap pcre2_match() for M

We bring in "subject" as an M string, so we can have embedded zero bytes. This gives us a length, so we don't have a separate parameter for that.

**Parameters**

| | |
|---|---|
| *count* | Count of parameters from the M API |
| *code_str* | A pcre2 compiled regular expression pointer in string format |
| *subject* | The string to search for matches |
| *startoffset* | The byte offset at which to start the match search |
| *options_str* | Pcre2 match options in string form |
| *match_data_str* | A Pcre2 match data pointer in string format |
| *mcontext_str* | A Pcre2 match context pointer in string format, or "0" |

**Returns**

$<$ 0 on error or no match, 0 vector offests too small, else one more than the highest numbered capturing pair that has been set

References get_match_context(), parse_pcre2_options(), and pointer_decode().

#### 5.1.3.36 mpcre2_match_context_copy()

```
gtm_char_t* mpcre2_match_context_copy (
              int count,
              gtm_char_t * mcontext_str )
```

Wrap the pcre2_match_context_copy() function.

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *mcontext_str* | String handle to a pcre2 match context |

**Returns**

String handle for a new pcre2 match context

References pointer_decode(), and pointer_encode().

#### 5.1.3.37 mpcre2_match_context_create()

```
gtm_char_t* mpcre2_match_context_create (
            int count,
            gtm_char_t * gcontext_str )
```

Wrap the pcre2_match_context_create() function.

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *gcontext_str* | String handle to a pcre2 general context |

**Returns**

String handle to a pcre2 match context

References get_general_context(), and pointer_encode().

#### 5.1.3.38 mpcre2_match_context_free()

```
void mpcre2_match_context_free (
            int count,
            gtm_char_t * mcontext_str )
```

Wrap the pcre2_match_context_free() function.

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *mcontext_str* | String handle for a pcre2 match context |

**Returns**

None

References pointer_decode().

#### 5.1.3.39 mpcre2_match_data_create()

```
gtm_char_t* mpcre2_match_data_create (
            int count,
            gtm_long_t ovecsize,
            gtm_char_t * gcontext_str )
```

Wrap the pcre2_match_data_create() function.

If we are called with a non "0" general context, use it. Otherwise we use our internal general context.

**Parameters**

| | |
|---|---|
| *count* | Count of parameters from the M API |
| *ovecsize* | The size of the output vector to create |
| *gcontext_str* | A custom general context or "0" in string format |

**Returns**

A stringified match data handle

References get_general_context(), and pointer_encode().

#### 5.1.3.40 mpcre2_match_data_create_from_pattern()

```
gtm_char_t* mpcre2_match_data_create_from_pattern (
             int count,
             gtm_char_t * code_str,
             gtm_char_t * gcontext_str )
```

Wrap pcre2_match_data_create_from_pattern()

If we are called with a non "0" general context, use it. Otherwise we use our internal general context.

**Parameters**

| | |
|---|---|
| *count* | Count of parameters from the M API |
| *code_str* | A compiled pcre2 regular expression pointer in string format |
| *gcontext_str* | A custom general context or "0" in string format |

**Returns**

A stringified match data pointer

References get_general_context(), pointer_decode(), and pointer_encode().

#### 5.1.3.41 mpcre2_match_data_free()

```
void mpcre2_match_data_free (
             int count,
             gtm_char_t * match_data_str )
```

Wrap the pcre2_match_data_free() function.

**Parameters**

| | |
|---|---|
| *count* | M API argument count |
| *match_data_str* | Stringified match data pointer |

**Returns**

None

References pointer_decode().

#### 5.1.3.42 mpcre2_pattern_info()

```
gtm_long_t mpcre2_pattern_info (
             int count,
             gtm_char_t * code_str,
             gtm_char_t * what_str,
             gtm_string_t * where )
```

Wrap the pcre2_pattern_info() function.

Note that currently no decoding is provided for the results of this function. The programmer must consult the PCRE2 documentation and the pcre2.h file to understand the return values.

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *code_str* | String handle for compiled PCRE2 expression |
| *what_str* | String representation of the PCRE2 INFO option being queried |
| *where* | M string into which the result is stored |

References parse_pcre2_options(), pointer_decode(), and pointer_encode().

#### 5.1.3.43 mpcre2_serialize_decode()

```
gtm_long_t mpcre2_serialize_decode (
             int count,
             gtm_string_t * codes_str,
             gtm_long_t number_of_codes,
             gtm_char_t * bytes_str,
             gtm_char_t * gcontext_str )
```

Wrap the pcre2_serialize_decode() function.

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *codes_str* | A string handle for an array of pcre2 compiled regular expression code slots |
| *number_of_codes* | The number of code slots in the codes array |
| *bytes_str* | A string handle for a block of serialized pcre2 compiled regexp codes |
| *gcontext_str* | String handle for a pcre2 general context or "NULL" |

**Returns**

A non-negative count on the number of codes loaded or a negative error code.

References get_general_context(), pointer_decode(), and pointer_encode().

#### 5.1.3.44 mpcre2_serialize_encode()

```
gtm_long_t mpcre2_serialize_encode (
             int count,
             gtm_char_t * codes_str,
             gtm_long_t number_of_codes,
             gtm_string_t * serialized_bytes_str,
             gtm_long_t * serialized_size,
             gtm_char_t * gcontext_str )
```

Wrap the pcre2_serialize_encode() function.

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *codes_str* | String handle for a pcre2 array of compiled regular expressions |
| *number_of_codes* | The number of codes in the array |
| *serialized_bytes_str* | M storage to hold the serialized expressions |
| *serialized_size* | Where to store the number of bytes in the serialized rendition |
| *gcontext_str* | String handle for a pcre2 general context |

**Returns**

A non-negative count of serialized patterns, or a negative error number

References get_general_context(), pointer_decode(), and pointer_encode().

#### 5.1.3.45 mpcre2_serialize_free()

```
void mpcre2_serialize_free (
             int count,
             gtm_char_t * bytes_str )
```

Wrap the void pcre2_serialize_free() function.

It is important to note that only storage allocated inside pcre2 should be freed here. If you, say, read the serialized data from disk, you should *not* free that here.

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *bytes_str* | String handle to a pcre2 allocated block of serialized regex codes |

**Returns**

None

References pointer_decode().

#### 5.1.3.46 mpcre2_serialize_get_number_of_codes()

```
gtm_long_t mpcre2_serialize_get_number_of_codes (
             int count,
             gtm_char_t * bytes_str )
```

Wrap the pcre2_serialize_get_number_of_codes() function.

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *bytes_str* | String handle to a block of serialized pcre2 regex codes |

**Returns**

A non-negative count of the number of codes in the block, or a negative error code

References pointer_decode().

#### 5.1.3.47 mpcre2_set_bsr()

```
gtm_long_t mpcre2_set_bsr (
             int count,
             gtm_char_t * ccontext_str,
             gtm_char_t * value_str )
```

Wrap the pcre2_set_bsr() function.

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *ccontext_str* | String handle for a pcre2 compile context |
| *value_str* | String form of pcre2 code for handling \R mapping |

**Returns**

0 on success, non zero on failure

References parse_pcre2_options(), and pointer_decode().

#### 5.1.3.48 mpcre2_set_callout()

```
gtm_long_t mpcre2_set_callout (
            int count,
            gtm_char_t * mcontext_str,
            gtm_char_t * callout_function_str,
            gtm_char_t * callout_data_str )
```

Wrap the pcre2_set_callout() function.

This function is unlikely to be of use in the libm_pcre2 context as it sets up a callout function in a pcre2 match context, and libm_pcre2 does not provide any way to create such callouts, or return handles for them. This could be done in another plugin and is left as an exercise for the reader.

**Parameters**

| | |
|---|---|
| *count* | Parameter count provided from the M API |
| *mcontext_str* | String handle for a pcre2 match context |
| *callout_function_str* | String handle for callout function |
| *callout_data_str* | String handle for callout data |

**Returns**

0 in all cases

References pointer_decode().

#### 5.1.3.49 mpcre2_set_character_tables()

```
gtm_long_t mpcre2_set_character_tables (
            int count,
            gtm_char_t * ccontext_str,
            gtm_char_t * tables_str )
```

Wrap the pcre2_set_character_tables() function.

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *ccontext_str* | String handle for a pcre2 compile context |
| *tables_str* | String handle for a result from pcre2_maketables, or "NULL" |

**Returns**

Always returns 0

References pointer_decode().

#### 5.1.3.50 mpcre2_set_compile_extra_options()

```
gtm_long_t mpcre2_set_compile_extra_options (
            int count,
            gtm_char_t * ccontext_str,
            gtm_char_t * extra_options_str )
```

Wrap the pcre2_set_compile_extra_options() function.

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *ccontext_str* | String handle for a pcre2 compile context |
| *extra_options_str* | String with pcre2 "extra" compile options |

**Returns**

0 in all cases

References parse_pcre2_options(), and pointer_decode().

#### 5.1.3.51 mpcre2_set_compile_recursion_guard()

```
gtm_long_t mpcre2_set_compile_recursion_guard (
            int count,
            gtm_char_t * ccontext_str,
            gtm_char_t * guard_function_str,
            gtm_char_t * user_data_str )
```

Wrap the pcre2_set_compile_recursion_guard() function.

This function is unlikely to be useful in the context of libm_pcre2, as it sets up a callback for use during expression compilation of parenthesized patterns, and libm_pcre2 does not provide any way to create such a function or return a handle to it. That functionality could be provided by another plugin, but is left as an exercise for the reader.

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *ccontext_str* | String handle for a pcre2 compile context |
| *guard_function_str* | String handle for a pointer to a guard function |
| *user_data_str* | String handle for a piece of user data |

**Returns**

0 in all cases

References pointer_decode().

#### 5.1.3.52 mpcre2_set_depth_limit()

```
gtm_long_t mpcre2_set_depth_limit (
            int count,
            gtm_char_t * mcontext_str,
            gtm_long_t value )
```

Wrap the pcre2_set_depth_limit() function.

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *mcontext_str* | String handle for a pcre2 depth context |
| *value* | Value for the backtracking depth limit field in the match context |

**Returns**

0 in all cases

References pointer_decode().

#### 5.1.3.53 mpcre2_set_heap_limit()

```
gtm_long_t mpcre2_set_heap_limit (
            int count,
            gtm_char_t * mcontext_str,
            gtm_long_t value )
```

Wrap the pcre2_set_heap_limit() function.

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *mcontext_str* | String handle for a pcre2 match context |
| *value* | Value for the backtracking heap limit field in the match context |

**Returns**

0 in all cases

References pointer_decode().

#### 5.1.3.54 mpcre2_set_match_limit()

```
gtm_long_t mpcre2_set_match_limit (
            int count,
            gtm_char_t * mcontext_str,
            gtm_long_t value )
```

Wrap the pcre2_set_match_limit() function.

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *mcontext_str* | String handle for a pcre2 match context |
| *value* | Value for the match limit field in the match context |

**Returns**

0 in all cases

References pointer_decode().

#### 5.1.3.55 mpcre2_set_max_pattern_length()

```
gtm_long_t mpcre2_set_max_pattern_length (
            int count,
            gtm_char_t * ccontext_str,
            gtm_long_t value )
```

Wrap the pcre2_set_max_pattern_length() function.

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *ccontext_str* | String handle for a pcre2 compile context |
| *value* | The maximum number of code units allowed in a pcre2 pattern |

**Returns**

0 in all cases

References pointer_decode().

#### 5.1.3.56 mpcre2_set_newline()

```
gtm_long_t mpcre2_set_newline (
            int count,
            gtm_char_t * ccontext_str,
            gtm_char_t * value_str )
```

Wrap pcre2_set_newline() function.

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *ccontext_str* | String handle for a pcre2 compile context |
| *value_str* | String representing the pcre2 newline convention to set |

**Returns**

0 on success, nonzero on error

References parse_pcre2_options(), and pointer_decode().

#### 5.1.3.57 mpcre2_set_offset_limit()

```
gtm_long_t mpcre2_set_offset_limit (
            int count,
            gtm_char_t * mcontext_str,
            gtm_long_t value )
```

Wrap the pcre2_set_offset_limit() function.

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *mcontext_str* | String handle for a pcre2 match context |
| *value* | Value for the offest limit field in the match context |

**Returns**

0 in all cases

References pointer_decode().

#### 5.1.3.58 mpcre2_set_parens_nest_limit()

```
gtm_long_t mpcre2_set_parens_nest_limit (
             int count,
             gtm_char_t * ccontext_str,
             gtm_long_t value )
```

Wrap the pcre2_set_parens_nest_limit() function.

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *ccontext_str* | String handle to a pcre2 compile context |
| *value* | The maximum allowed nesting depth for parens |

**Returns**

0 in all cases

References pointer_decode().

#### 5.1.3.59 mpcre2_substitute()

```
gtm_long_t mpcre2_substitute (
             int count,
             gtm_char_t * code_str,
             gtm_string_t * subject,
             gtm_long_t startoffset,
             gtm_char_t * options_str,
             gtm_char_t * match_data_str,
             gtm_char_t * mcontext_str,
             gtm_string_t * replacement,
             gtm_string_t * outputbuffer,
             gtm_long_t * outputlengthptr )
```

Wrap the pcre2_substitute() function.

**Parameters**

| | |
|---|---|
| *count* | M API argument count |
| *code_str* | String handle for a compiled PCRE2 regular expression |
| *subject* | The string in which to make the substitution |
| *startoffset* | The byte offset in the subject to start checking for substitutions |
| *options_str* | Match options as in pcre2_match() |
| *match_data_str* | String handle for match data as in pcre2_match() |
| *mcontext_str* | String handle for Pcre2 match context |
| *replacement* | The string to substitute for matched text |
| *outputbuffer* | Where to put the copy of the subject with the replacement(s) |
| *outputlengthptr* | Where to store the length of the copy of the subject with replacement(s) |

**Returns**

The number of substitutions or $<$ 0 on error

References get_match_context(), parse_pcre2_options(), and pointer_decode().

#### 5.1.3.60 mpcre2_substring_copy_byname()

```
gtm_long_t mpcre2_substring_copy_byname (
             int count,
             gtm_char_t * match_data_str,
             gtm_char_t * name,
             gtm_string_t * buffer )
```

Wrap the pcre2_substring_copy_byname() function.

This function will return one of several PCRE2 errors on failure: PCRE2_ERROR_NOSUBSTRING there are no groups of that name PCRE2_ERROR_UNAVAILABLE the ovector was too small for that group PCRE2_ERROR←_UNSET the group did not participate in the match PCRE2_ERROR_NOMEMORY the buffer is not big enough

In C, these symbolic error codes can be used in code. This is not the case in M, and libm_pcre2 does not currently provide a translation. The current numerical values for these are, respectively: -49, -54, -55 & -48

Note that this function does not need the "bufflen" parameter from the C version because the gtm_string_t type handles that.

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *match_data_str* | String handle for pcre2 match data |
| *name* | Name of the captured substring |
| *buffer* | to receive the string |

**Returns**

0 on success non-zero on error

References opt_tab::name, and pointer_decode().

#### 5.1.3.61 mpcre2_substring_copy_bynumber()

```
gtm_long_t mpcre2_substring_copy_bynumber (
             int count,
             gtm_char_t * match_data_str,
             gtm_long_t number,
             gtm_string_t * buffer )
```

Wrap the pcre2_substring_copy_bynumber() function.

This function will return one of several PCRE2 errors on failure: PCRE2_ERROR_NOSUBSTRING there are no groups of that number PCRE2_ERROR_UNAVAILABLE the ovector was too small for that group PCRE2_ERRO↩R_UNSET the group did not participate in the match PCRE2_ERROR_NOMEMORY the buffer is not big enough

In C, these symbolic error codes can be used in code. This is not the case in M, and libm_pcre2 does not currently provide a translation. The current numerical values for these are, respectively: -49, -54, -55 & -48

Note that this function does not need the "bufflen" parameter from the C version because the gtm_string_t type handles that.

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *match_data_str* | String handle for pcre2 match data |
| *number* | number of the captured substring |
| *buffer* | to receive the string |

**Returns**

0 on success non-zero on error

References pointer_decode().

#### 5.1.3.62 mpcre2_substring_free()

```
void mpcre2_substring_free (
             int count,
             gtm_char_t * buffer_str )
```

Wrap the void pcre2_substring_free() function.

Free memory allocated by pcre2_substring_get_byname() or pcre2_substring_get_bynumber().

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *buffer_str* | String handle for the buffer to free |

**Returns**

None

References pointer_decode().

#### 5.1.3.63 mpcre2_substring_get_byname()

```
int mpcre2_substring_get_byname (
            int count,
            gtm_char_t * match_data_str,
            gtm_char_t * name,
            gtm_string_t * bufferptr_str,
            gtm_long_t * bufflen )
```

Wrap the pcre2_substring_get_byname() function.

It is not clear that this function has any utility in an M environment. It is only useful with the helper function **mpcre2↩ _get_mstring_from_buf()** (p. 24).

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *match_data_str* | String handle for pcre2 match data |
| *name* | Name of the capture group |
| *bufferptr_str* | Where to store the pointer to the allocated buffer |
| *bufflen* | Where to store the length of the allocated buffer |

**Returns**

0 on success non-zero on error

References pointer_decode(), and pointer_encode().

#### 5.1.3.64 mpcre2_substring_get_bynumber()

```
int mpcre2_substring_get_bynumber (
            int count,
            gtm_char_t * match_data_str,
            gtm_long_t number,
            gtm_string_t * bufferptr_str,
            gtm_long_t * bufflen )
```

Wrap the pcre2_substring_get_bynumber() function.

It is not clear that this function has any utility in an M environment. It is only useful with the helper function **mpcre2↩ _get_mstring_from_buf()** (p. 24).

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *match_data_str* | String handle for pcre2 match data |
| *number* | Number of the capture group |
| *bufferptr_str* | Where to store the pointer to the allocated buffer |
| *bufflen* | Where to store the length of the allocated buffer |

**Returns**

0 on success non-zero on error

References pointer_decode(), and pointer_encode().

#### 5.1.3.65 mpcre2_substring_length_byname()

```
gtm_long_t mpcre2_substring_length_byname (
             int count,
             gtm_char_t * match_data_str,
             gtm_char_t * name,
             gtm_long_t * length )
```

Wrap the pcre2_substring_length_byname() function.

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *match_data_str* | String handle for pcre2 match data |
| *name* | Name of the capture group |
| *length* | Where to store the length of the capture |

**Returns**

0 on success non-zero otherwise

References pointer_decode().

#### 5.1.3.66 mpcre2_substring_length_bynumber()

```
gtm_long_t mpcre2_substring_length_bynumber (
             int count,
             gtm_char_t * match_data_str,
             gtm_long_t number,
             gtm_long_t * length )
```

Wrap the pcre2_substring_length_bynumber() function.

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *match_data_str* | String handle for pcre2 match data |
| *number* | Number of the capture group |
| *length* | Where to store the length of the capture |

**Returns**

0 on success non-zero otherwise

References pointer_decode().

#### 5.1.3.67 mpcre2_substring_list_free()

```
void mpcre2_substring_list_free (
             int count,
             gtm_char_t * list_str )
```

Wrap the pcre2_substring_list_free() function.

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *list_str* | String handle for a pcre2 substring list |

**Returns**

None

References pointer_decode().

#### 5.1.3.68 mpcre2_substring_list_get()

```
gtm_long_t mpcre2_substring_list_get (
             int count,
             gtm_char_t * match_data_str,
             gtm_string_t * listptr_str,
             gtm_string_t * lengthsptr_str )
```

Wrap the pcre2_substring_list_get() function.

To be useful from M, this function must be used in conjunction with the utilitiy functions pcre2_get_substring_list↩
_count() & pcre2_get_mstring_from_substring_list().

On the C side, we can pass NULL for lengthsptr. We don't accept that here.

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *match_data_str* | String handle for pcre2 match data |
| *listptr_str* | Variable to hold the handle for the substring list |
| *lengthsptr_str* | Variable to hold the handle for the lengths list |

**Returns**

0 on success, non-zero on error

References pointer_decode(), and pointer_encode().

#### 5.1.3.69 mpcre2_substring_number_from_name()

```
gtm_long_t mpcre2_substring_number_from_name (
             int count,
             gtm_char_t * code_str,
             gtm_char_t * name )
```

Wrap the pcre2_substring_number_from_name() function.

**Parameters**

| | |
|---|---|
| *count* | Parameter count from the M API |
| *code_str* | String handle for a pcre2 compiled regular expression |
| *name* | name of the capture group to map |

**Returns**

Either the number of the parentheses for the name or a negative error number (see pcre2.h)

References pointer_decode().

#### 5.1.3.70 parse_pcre2_options()

```
static int parse_pcre2_options (
             struct opt_tab * option_table,
             int option_count,
             char * opt_tag,
             char * options,
             uint32_t * result ) [static]
```

Turn various pcre2 options presented as C strings into integers.

Normally, when using options in certain pcre2 functions, the options are presented as a bitmap constructed by logically ORing macros. When the options come in from M, they are a string that looks the same as the C interface, but which must be parsed out since this is not easily done on the M side.

This means we are called with strings like:

```
"PCRE2_ANCHORED|PCRE2_ALLOW_EMPTY_CLASS|PCRE2_ALT_BSUX"
```

If an error is encountered, -1 will be returned, otherwise the indicated logical OR will be.

This parse function is used with several different option tables.

**Parameters**

| | |
|---|---|
| *option_table* | table mapping strings to bitmasks |
| *option_count* | number of entries in the option table |
| *opt_tag* | String to use as a label for this set of options in error messages |
| *options* | A null terminated C string indicating flags. |
| *result* | Pointer to the storage for the ORed flags result |

**Returns**

0 on success or -1 on error

References m_pcre2_free(), m_pcre2_malloc(), opt_tab::name, and opt_tab::val.

Referenced by mpcre2_compile(), mpcre2_dfa_match(), mpcre2_jit_compile(), mpcre2_jit_match(), mpcre2_↩ match(), mpcre2_pattern_info(), mpcre2_set_bsr(), mpcre2_set_compile_extra_options(), mpcre2_set_newline(), and mpcre2_substitute().

#### 5.1.3.71 pointer_decode()

```
static void* pointer_decode (
             char * pstr )  [static]
```

Decode a string handle into a pointer.

We pass pointers out to M as strings, and deode strings from M into pointers. This function does the decode. There is some special casing which translates the strings "0", or "NULL" into NULL pointers. Otherwise the string is assumed to contain a base-10 number which is cast into a void pointer.

**Parameters**

| | |
|---|---|
| *pstr* | A pointer value encoded as a string handle |

**Returns**

A void pointer which must be cast before use

Referenced by get_compile_context(), get_general_context(), get_match_context(), mpcre2_callout_enumerate(), mpcre2_code_copy(), mpcre2_code_copy_with_tables(), mpcre2_code_free(), mpcre2_compile_context_↩ copy(), mpcre2_compile_context_create(), mpcre2_compile_context_free(), mpcre2_dfa_match(), mpcre2_↩ general_context_copy(), mpcre2_general_context_create(), mpcre2_general_context_free(), mpcre2_get_mark(), mpcre2_get_mstring_from_buf(), mpcre2_get_mstring_from_substring_list(), mpcre2_get_ov_pair(), mpcre2_↩ get_ovector_count(), mpcre2_get_ovector_pointer(), mpcre2_get_startchar(), mpcre2_get_substring_list_count(), mpcre2_jit_compile(), mpcre2_jit_match(), mpcre2_jit_stack_assign(), mpcre2_jit_stack_free(), mpcre2_match(), mpcre2_match_context_copy(), mpcre2_match_context_free(), mpcre2_match_data_create_from_pattern(), mpcre2_match_data_free(), mpcre2_pattern_info(), mpcre2_serialize_decode(), mpcre2_serialize_encode(), mpcre2_serialize_free(), mpcre2_serialize_get_number_of_codes(), mpcre2_set_bsr(), mpcre2_set_callout(), mpcre2_set_character_tables(), mpcre2_set_compile_extra_options(), mpcre2_set_compile_recursion_guard(), mpcre2_set_depth_limit(), mpcre2_set_heap_limit(), mpcre2_set_match_limit(), mpcre2_set_max_pattern_↩ length(), mpcre2_set_newline(), mpcre2_set_offset_limit(), mpcre2_set_parens_nest_limit(), mpcre2_substitute(), mpcre2_substring_copy_byname(), mpcre2_substring_copy_bynumber(), mpcre2_substring_free(), mpcre2_↩ substring_get_byname(), mpcre2_substring_get_bynumber(), mpcre2_substring_length_byname(), mpcre2_↩ substring_length_bynumber(), mpcre2_substring_list_free(), mpcre2_substring_list_get(), and mpcre2_substring↩ _number_from_name().

#### 5.1.3.72 pointer_encode()

```
static void pointer_encode (
            void * ptr,
            char * buf,
            int size ) [static]
```

Encode a pointer as a string handle.

A pointer is converted to a string representing a decimal number, ie a pointer holding the value, say, 0x10abc becomes the string "68284". This string is written into the caller supplied buffer.

**Parameters**

| | |
|---|---|
| *ptr* | The pointer value to encode |
| *buf* | String storage |
| *size* | Size of storage |

**Returns**

None

Referenced by mpcre2_code_copy(), mpcre2_code_copy_with_tables(), mpcre2_compile(), mpcre2_compile_↩ context_copy(), mpcre2_compile_context_create(), mpcre2_general_context_copy(), mpcre2_general_context↩ _create(), mpcre2_get_general_context(), mpcre2_get_ovector_pointer(), mpcre2_jit_stack_create(), mpcre2↩ _maketables(), mpcre2_match_context_copy(), mpcre2_match_context_create(), mpcre2_match_data_create(), mpcre2_match_data_create_from_pattern(), mpcre2_pattern_info(), mpcre2_serialize_decode(), mpcre2↩ _serialize_encode(), mpcre2_substring_get_byname(), mpcre2_substring_get_bynumber(), and mpcre2_↩ substring_list_get().

### 5.1.4 Variable Documentation

#### 5.1.4.1 bsr_opts

```
struct opt_tab bsr_opts[] [static]
```

**Initial value:**

```
= {
        { "PCRE2_BSR_ANYCRLF", PCRE2_BSR_ANYCRLF },
        { "PCRE2_BSR_UNICODE", PCRE2_BSR_UNICODE },
}
```

This table maps PCRE2 BSR options from M strings to C macro values

#### 5.1.4.2 compile_opts

```
struct opt_tab compile_opts[] [static]
```

**Initial value:**

```
= {
        { "PCRE2_ANCHORED", PCRE2_ANCHORED },
        { "PCRE2_ALLOW_EMPTY_CLASS", PCRE2_ALLOW_EMPTY_CLASS },
        { "PCRE2_ALT_BSUX", PCRE2_ALT_BSUX },
        { "PCRE2_ALT_CIRCUMFLEX", PCRE2_ALT_CIRCUMFLEX },
        { "PCRE2_ALT_VERBNAMES", PCRE2_ALT_VERBNAMES },
        { "PCRE2_AUTO_CALLOUT", PCRE2_AUTO_CALLOUT },
        { "PCRE2_CASELESS", PCRE2_CASELESS },
        { "PCRE2_DOLLAR_ENDONLY", PCRE2_DOLLAR_ENDONLY },
        { "PCRE2_DOTALL", PCRE2_DOTALL },
        { "PCRE2_DUPNAMES", PCRE2_DUPNAMES },
        { "PCRE2_ENDANCHORED", PCRE2_ENDANCHORED },
        { "PCRE2_EXTENDED", PCRE2_EXTENDED },
        { "PCRE2_FIRSTLINE", PCRE2_FIRSTLINE },
        { "PCRE2_LITERAL", PCRE2_LITERAL },
        { "PCRE2_MATCH_UNSET_BACKREF", PCRE2_MATCH_UNSET_BACKREF },
        { "PCRE2_MULTILINE", PCRE2_MULTILINE },
        { "PCRE2_NEVER_BACKSLASH_C", PCRE2_NEVER_BACKSLASH_C },
        { "PCRE2_NEVER_UCP", PCRE2_NEVER_UCP },
        { "PCRE2_NEVER_UTF", PCRE2_NEVER_UTF },
        { "PCRE2_NO_AUTO_CAPTURE", PCRE2_NO_AUTO_CAPTURE },
        { "PCRE2_NO_AUTO_POSSESS", PCRE2_NO_AUTO_POSSESS },
        { "PCRE2_NO_DOTSTAR_ANCHOR", PCRE2_NO_DOTSTAR_ANCHOR },
        { "PCRE2_NO_START_OPTIMIZE", PCRE2_NO_START_OPTIMIZE },
        { "PCRE2_NO_UTF_CHECK", PCRE2_NO_UTF_CHECK },
        { "PCRE2_UCP", PCRE2_UCP },
        { "PCRE2_UNGREEDY", PCRE2_UNGREEDY },
        { "PCRE2_USE_OFFSET_LIMIT", PCRE2_USE_OFFSET_LIMIT },
        { "PCRE2_UTF", PCRE2_UTF },
}
```

This table maps PCRE2 regular expression compile options from M strings to C macro values

#### 5.1.4.3 extra_compile_opts

```
struct opt_tab extra_compile_opts[] [static]
```

**Initial value:**

```
= {
        { "PCRE2_EXTRA_ALLOW_SURROGATE_ESCAPES", PCRE2_EXTRA_ALLOW_SURROGATE_ESCAPES },
        { "PCRE2_EXTRA_BAD_ESCAPE_IS_LITERAL", PCRE2_EXTRA_BAD_ESCAPE_IS_LITERAL },
        { "PCRE2_EXTRA_MATCH_LINE", PCRE2_EXTRA_MATCH_LINE },
        { "PCRE2_EXTRA_MATCH_WORD", PCRE2_EXTRA_MATCH_WORD },
}
```

This table maps PCRE2 "extra" regular expression compile options from M strings to C macro values

#### 5.1.4.4 info_opts

```
struct opt_tab info_opts[] [static]
```

**Initial value:**

```
= {
        { "PCRE2_INFO_ALLOPTIONS", PCRE2_INFO_ALLOPTIONS },
        { "PCRE2_INFO_ARGOPTIONS", PCRE2_INFO_ARGOPTIONS },
        { "PCRE2_INFO_BACKREFMAX", PCRE2_INFO_BACKREFMAX },
        { "PCRE2_INFO_BSR", PCRE2_INFO_BSR },
        { "PCRE2_INFO_CAPTURECOUNT", PCRE2_INFO_CAPTURECOUNT },
        { "PCRE2_INFO_FIRSTCODEUNIT", PCRE2_INFO_FIRSTCODEUNIT },
        { "PCRE2_INFO_FIRSTCODETYPE", PCRE2_INFO_FIRSTCODETYPE },
        { "PCRE2_INFO_FIRSTBITMAP", PCRE2_INFO_FIRSTBITMAP },
        { "PCRE2_INFO_HASCRORLF", PCRE2_INFO_HASCRORLF },
        { "PCRE2_INFO_JCHANGED", PCRE2_INFO_JCHANGED },
        { "PCRE2_INFO_JITSIZE", PCRE2_INFO_JITSIZE },
        { "PCRE2_INFO_LASTCODEUNIT", PCRE2_INFO_LASTCODEUNIT },
        { "PCRE2_INFO_LASTCODETYPE", PCRE2_INFO_LASTCODETYPE },
        { "PCRE2_INFO_MATCHEMPTY", PCRE2_INFO_MATCHEMPTY },
        { "PCRE2_INFO_MATCHLIMIT", PCRE2_INFO_MATCHLIMIT },
        { "PCRE2_INFO_MAXLOOKBEHIND", PCRE2_INFO_MAXLOOKBEHIND },
        { "PCRE2_INFO_MINLENGTH", PCRE2_INFO_MINLENGTH },
        { "PCRE2_INFO_NAMECOUNT", PCRE2_INFO_NAMECOUNT },
        { "PCRE2_INFO_NAMEENTRYSIZE", PCRE2_INFO_NAMEENTRYSIZE },
        { "PCRE2_INFO_NAMETABLE", PCRE2_INFO_NAMETABLE },
        { "PCRE2_INFO_NEWLINE", PCRE2_INFO_NEWLINE },
        { "PCRE2_INFO_DEPTHLIMIT", PCRE2_INFO_DEPTHLIMIT },
        { "PCRE2_INFO_RECURSIONLIMIT", PCRE2_INFO_RECURSIONLIMIT },
        { "PCRE2_INFO_SIZE", PCRE2_INFO_SIZE },
        { "PCRE2_INFO_HASBACKSLASHC", PCRE2_INFO_HASBACKSLASHC },
        { "PCRE2_INFO_FRAMESIZE", PCRE2_INFO_FRAMESIZE },
        { "PCRE2_INFO_HEAPLIMIT", PCRE2_INFO_HEAPLIMIT },
        { "PCRE2_INFO_EXTRAOPTIONS", PCRE2_INFO_EXTRAOPTIONS },
}
```

This table maps PCRE2 info options for pcre2_pattern_info() from M strings to C macro values

The parser will allow ORing these with '|', but unlike other mapping tables, the C API doesn't support that, so don't do it.

#### 5.1.4.5 jit_opts

```
struct  opt_tab jit_opts[]  [static]
```

**Initial value:**

```
= {
        { "PCRE2_JIT_COMPLETE", PCRE2_JIT_COMPLETE },
        { "PCRE2_JIT_PARTIAL_SOFT", PCRE2_JIT_PARTIAL_SOFT },
        { "PCRE2_JIT_PARTIAL_HARD", PCRE2_JIT_PARTIAL_HARD },
}
```

This table maps PCRE2 Just In Time (JIT) options to from M strings to C macro values

#### 5.1.4.6 match_opts

```
struct  opt_tab match_opts[]  [static]
```

**Initial value:**

```
= {
        { "PCRE2_ANCHORED", PCRE2_ANCHORED },
        { "PCRE2_ENDANCHORED", PCRE2_ENDANCHORED },
        { "PCRE2_NOTBOL", PCRE2_NOTBOL },
        { "PCRE2_NOTEOL", PCRE2_NOTEOL },
        { "PCRE2_NOTEMPTY", PCRE2_NOTEMPTY },
        { "PCRE2_NOTEMPTY_ATSTART", PCRE2_NOTEMPTY_ATSTART },
        { "PCRE2_NO_JIT", PCRE2_NO_JIT },
        { "PCRE2_NO_UTF_CHECK", PCRE2_NO_UTF_CHECK },
        { "PCRE2_PARTIAL_HARD", PCRE2_PARTIAL_HARD },
        { "PCRE2_PARTIAL_SOFT", PCRE2_PARTIAL_SOFT },
        { "PCRE2_SUBSTITUTE_EXTENDED", PCRE2_SUBSTITUTE_EXTENDED },
        { "PCRE2_SUBSTITUTE_GLOBAL", PCRE2_SUBSTITUTE_GLOBAL },
        { "PCRE2_SUBSTITUTE_OVERFLOW_LENGTH", PCRE2_SUBSTITUTE_OVERFLOW_LENGTH },
        { "PCRE2_SUBSTITUTE_UNKNOWN_UNSET", PCRE2_SUBSTITUTE_UNKNOWN_UNSET },
        { "PCRE2_SUBSTITUTE_UNSET_EMPTY", PCRE2_SUBSTITUTE_UNSET_EMPTY },
}
```

This table maps PCRE2 match and substitute options from M strings to C macro values

It has both match and substitute options, because pcre2_substitute() takes many of the match options

#### 5.1.4.7 newline_opts

```
struct opt_tab newline_opts[] [static]
```

**Initial value:**

```
= {
        { "PCRE2_NEWLINE_CR", PCRE2_NEWLINE_CR },
        { "PCRE2_NEWLINE_LF", PCRE2_NEWLINE_LF },
        { "PCRE2_NEWLINE_CRLF", PCRE2_NEWLINE_CRLF },
        { "PCRE2_NEWLINE_ANYCRLF", PCRE2_NEWLINE_ANYCRLF },
        { "PCRE2_NEWLINE_ANY", PCRE2_NEWLINE_ANY },
        { "PCRE2_NEWLINE_NUL", PCRE2_NEWLINE_NUL },
}
```

This table maps PCRE2 newline options from M strings to C macro values

# Index