

Statics Strings

STM32F1XX

Generated by Doxygen 1.8.18

1 Static Strings	1
1.1 Features:	1
1.2 Getting Started	1
1.2.1 Suggested names	1
1.2.2 First of all initialize the library	2
1.2.3 Creating a string	2
1.2.4 Also a string can created this way	2
1.2.5 Split a local scope string	2
1.2.6 Getting a substring	2
1.2.7 Concatenate two strings and search for a substring and a character in the result	2
1.2.8 Compare two equals and non equals strings	3
1.2.9 Transform a integer and a float to a string	3
1.2.10 Copy, move and clone a string	3
1.2.11 Concatenate and clean two strings	4
1.2.12 Concatenate multiple strings	4
1.2.13 Configure quantity and size of the memory arrays	4
2 Module Index	5
2.1 Modules	5
3 Data Structure Index	7
3.1 Data Structures	7
4 File Index	9
4.1 File List	9
5 Module Documentation	11
5.1 String types size and quantity	11
5.1.1 Detailed Description	11
5.2 String types	12
5.2.1 Detailed Description	12
5.3 String status	13
5.3.1 Detailed Description	13
5.4 Error handling	14
5.4.1 Detailed Description	14
5.4.2 Variable Documentation	14
5.4.2.1 static_strings_error_code	14
5.5 Static memory arrays	15
5.5.1 Detailed Description	15
5.6 String descriptors	16
5.6.1 Detailed Description	16
6 Data Structure Documentation	17
6.1 static_strings_string_descriptor Struct Reference	17

6.1.1 Detailed Description	17
6.2 static_strings_string_splitter_parameters Struct Reference	17
6.2.1 Detailed Description	17
7 File Documentation	19
7.1 int_types.h File Reference	19
7.1.1 Detailed Description	19
7.2 static_strings.c File Reference	19
7.2.1 Detailed Description	21
7.2.2 Function Documentation	21
7.2.2.1 static_strings_allocate()	21
7.2.2.2 static_strings_clone()	22
7.2.2.3 static_strings_compare()	22
7.2.2.4 static_strings_concatenate()	23
7.2.2.5 static_strings_concatenate_all()	23
7.2.2.6 static_strings_concatenate_and_clean()	24
7.2.2.7 static_strings_contains_char()	24
7.2.2.8 static_strings_contains_string()	24
7.2.2.9 static_strings_copy()	26
7.2.2.10 static_strings_create_custom_string()	26
7.2.2.11 static_strings_deallocate()	27
7.2.2.12 static_strings_double_to_string()	27
7.2.2.13 static_strings_float_to_string()	27
7.2.2.14 static_strings_get_string_max_length()	28
7.2.2.15 static_strings_init()	28
7.2.2.16 static_strings_int16_to_string()	28
7.2.2.17 static_strings_int32_to_string()	29
7.2.2.18 static_strings_int8_to_string()	29
7.2.2.19 static_strings_is_line()	30
7.2.2.20 static_strings_move()	30
7.2.2.21 static_strings_save()	30
7.2.2.22 static_strings_string_splitter_get_next_token()	31
7.2.2.23 static_strings_string_splitter_set_parameters()	31
7.2.2.24 static_strings_strlen()	32
7.2.2.25 static_strings_substring()	32
7.2.2.26 static_strings_uint16_to_string()	33
7.2.2.27 static_strings_uint32_to_string()	33
7.2.2.28 static_strings_uint8_to_string()	33
7.2.3 Variable Documentation	34
7.2.3.1 static_strings_string_splitter	34
7.3 static_strings.h File Reference	34
7.3.1 Detailed Description	37

7.3.2 Function Documentation	37
7.3.2.1 static_strings_allocate()	37
7.3.2.2 static_strings_clone()	38
7.3.2.3 static_strings_compare()	38
7.3.2.4 static_strings_concatenate()	39
7.3.2.5 static_strings_concatenate_all()	39
7.3.2.6 static_strings_concatenate_and_clean()	40
7.3.2.7 static_strings_contains_char()	40
7.3.2.8 static_strings_contains_string()	40
7.3.2.9 static_strings_copy()	42
7.3.2.10 static_strings_create_custom_string()	42
7.3.2.11 static_strings_deallocate()	43
7.3.2.12 static_strings_double_to_string()	43
7.3.2.13 static_strings_float_to_string()	43
7.3.2.14 static_strings_get_string_max_length()	44
7.3.2.15 static_strings_init()	44
7.3.2.16 static_strings_int16_to_string()	44
7.3.2.17 static_strings_int32_to_string()	45
7.3.2.18 static_strings_int8_to_string()	45
7.3.2.19 static_strings_is_line()	46
7.3.2.20 static_strings_move()	46
7.3.2.21 static_strings_save()	46
7.3.2.22 static_strings_string_splitter_get_next_token()	47
7.3.2.23 static_strings_string_splitter_set_parameters()	47
7.3.2.24 static_strings_strlen()	48
7.3.2.25 static_strings_substring()	48
7.3.2.26 static_strings_uint16_to_string()	49
7.3.2.27 static_strings_uint32_to_string()	49
7.3.2.28 static_strings_uint8_to_string()	49
7.3.3 Variable Documentation	50
7.3.3.1 static_strings_string_splitter	50

Chapter 1

Static Strings

1.1 Features:

- Developed for the STM32F103.
- Global scope strings.
- No dynamic memory allocation.
- Customizable quantity and size of string types.
- Create custom string function to create local scope strings.
- String length function.
- String can be \0 terminated and \r\n terminated.
- String split function.
- Fast string creation with save.
- Low level string creation with allocate.
- Reusable memory with deallocate.
- is_line function.
- Substring, concatenate, concatenate and clean, concatenate all.
- Contains string, contains char and compare function.
- Transforms integers and floats to strings
- Get string maximum length.

1.2 Getting Started

1.2.1 Suggested names

```
static_strings_string_descriptor string_name;  
uint8_t string_name_memory[];
```

1.2.2 First of all initialize the library

```
static_strings_init();
```

1.2.3 Creating a string

```
uint8_t test_memory[] = "Hello Word\r\n";
static_strings_string_descriptor *test = static_strings_save(test_memory);
if (test == NULL) {
    Error Handling.
}
else {
    Some work.
    static_strings_deallocate(test);
}
```

DON'T FORGET TO DEALLOCATE AFTER USING.

1.2.4 Also a string can created this way

```
#include "string.h"
uint8_t test_memory[] = "Hello Word\r\n";
uint16_t test_length = static_strings_strlen(test_memory);
static_strings_string_descriptor *test = static_strings_allocate(test_length);
if (test == NULL) {
    Error Handling.
}
else {
    memcpy(test->string, test_memory, test_length);
    test->length = test_length;
    Some work.
    static_strings_deallocate(test);
}
```

DON'T FORGET TO DEALLOCATE AFTER USING.

1.2.5 Split a local scope string

```
uint8_t split_memory[10] = "123,56,8\r\n";
static_strings_string_descriptor split;
static_strings_create_custom_string(&split, split_memory);
static_strings_string_descriptor *token;
static_strings_string_splitter_set_parameters(split, ',', ' ');
while (static_strings_string_splitter_get_next_token(&token)) {
    HAL_UART_Transmit(&huart1, token->string, token->length, HAL_MAX_DELAY);
}
```

1.2.6 Getting a substring

```
uint8_t custom[10] = "123,56,89\0";
static_strings_create_custom_string(string_descriptor, custom);
static_strings_string_descriptor *substring = static_strings_substring(string_descriptor, 2, 8);
if (substring != NULL) {
    HAL_UART_Transmit(&huart1, substring->string, substring->length, HAL_MAX_DELAY);
    static_strings_deallocate(substring);
}
```

1.2.7 Concatenate two strings and search for a substring and a character in the result

```
uint8_t concatenate_at_memory[] = "Hello \0";
static_strings_string_descriptor concatenate_at;
static_strings_create_custom_string(&concatenate_at, concatenate_at_memory);
uint8_t concatenate_memory[] = "World\r\n";
static_strings_string_descriptor concatenate;
static_strings_create_custom_string(&concatenate, concatenate_memory);
static_strings_string_descriptor *concatenated;
concatenated = static_strings_concatenate(&concatenate_at, &concatenate);
if (concatenated != NULL) {
    HAL_UART_Transmit(&huart1, concatenated->string, concatenated->length, HAL_MAX_DELAY);
    if (static_strings_contains_string(concatenated, &concatenate_at)) {
        HAL_UART_Transmit(&huart1, (uint8_t *) "1\r\n", 3, HAL_MAX_DELAY);
    }
}
```



```

    }
    else{
        HAL_UART_Transmit(&huart1, (uint8_t *) "0\r\n", 3, HAL_MAX_DELAY);
    }
    if(static_strings_contains_string(concatenated, 'W')){
        HAL_UART_Transmit(&huart1, (uint8_t *) "1\r\n", 3, HAL_MAX_DELAY);
    }
    else{
        HAL_UART_Transmit(&huart1, (uint8_t *) "0\r\n", 3, HAL_MAX_DELAY);
    }
    static_strings_deallocate(concatenated);
}

```

1.2.8 Compare two equals and non equals strings

```

uint8_t equal_a_memory[] = "Hall\0";
static_strings_string_descriptor equal_a;
uint8_t equal_b_memory[] = "Hall\0";
static_strings_string_descriptor equal_b;
uint8_t non_equal_memory[] = "oil\0";
static_strings_string_descriptor non_equal;
static_strings_create_custom_string(&equal_a, equal_a_memory);
static_strings_create_custom_string(&equal_b, equal_b_memory);
static_strings_create_custom_string(&non_equal, non_equal_memory);
if(static_strings_compare(&equal_a, &equal_b)) {
    HAL_UART_Transmit(&huart1, (uint8_t *) "1\r\n", 3, HAL_MAX_DELAY);
}
else{
    HAL_UART_Transmit(&huart1, (uint8_t *) "0\r\n", 3, HAL_MAX_DELAY);
}
if(static_strings_compare(&equal_a, &non_equal)) {
    HAL_UART_Transmit(&huart1, (uint8_t *) "1\r\n", 3, HAL_MAX_DELAY);
}
else{
    HAL_UART_Transmit(&huart1, (uint8_t *) "0\r\n", 3, HAL_MAX_DELAY);
}

```

1.2.9 Transform a integer and a float to a string

```

static_strings_string_descriptor *var_string;
uint8_t uint8 = 200;
var_string = static_strings_uint8_to_string(uint8);
if(var_string != NULL){
    HAL_UART_Transmit(&huart1, var_string->string, var_string->length, HAL_MAX_DELAY);
    static_strings_deallocate(var_string);
}
float float_number = 19.60232;
var_string = static_strings_float_to_string(float_number);
if(var_string != NULL){
    HAL_UART_Transmit(&huart1, var_string->string, var_string->length, HAL_MAX_DELAY);
    static_strings_deallocate(var_string);
}

```

1.2.10 Copy, move and clone a string

```

static_strings_string_descriptor *copy_test_source_string = static_strings_save((uint8_t *) "I am a copy
test\r\n");
if(copy_test_source_string != NULL){
    static_strings_string_descriptor *copy_test_target_string = static_strings_allocate(100);
    if(static_strings_copy(copy_test_target_string, copy_test_source_string, 0) != NULL){

        HAL_UART_Transmit(&huart1, copy_test_target_string->string, copy_test_target_string->length, HAL_MAX_DELAY);
        static_strings_deallocate(copy_test_source_string);
        static_strings_deallocate(copy_test_target_string);
    }
}
static_strings_string_descriptor *move_test_source_string = static_strings_save((uint8_t *) "I am a move
test\r\n");
if(copy_test_source_string != NULL){
    static_strings_string_descriptor *move_test_target_string = static_strings_allocate(100);
    *move_test_target_string->string = '.';
    if(static_strings_move(move_test_target_string, move_test_source_string, 1) != NULL){

        HAL_UART_Transmit(&huart1, move_test_target_string->string, move_test_target_string->length, HAL_MAX_DELAY);
        static_strings_deallocate(move_test_source_string);
    }
}

```

```
static_strings_string_descriptor *clone_test_source_string = static_strings_save((uint8_t *) "I am a clone
test\r\n");
if(copy_test_source_string != NULL){
    static_strings_string_descriptor *clone_test_target_string =
        static_strings_clone(clone_test_source_string);
    if(clone_test_target_string != NULL){
        HAL_UART_Transmit(&huart1, clone_test_target_string->string, clone_test_target_string->length, HAL_MAX_DELAY);
        static_strings_deallocate(clone_test_source_string);
        static_strings_deallocate(clone_test_target_string);
    }
}
```

1.2.11 Concatenate and clean two strings

```
static_strings_string_descriptor *concatenate_at = static_strings_save((uint8_t *) "I am a ");
static_strings_string_descriptor *concatenate = static_strings_save((uint8_t *) "concatenate test\r\n");
if(concatenate_at != NULL && concatenate != NULL){
    static_strings_string_descriptor *concatenated_string =
        static_strings_concatenate_and_clean(concatenate_at, concatenate);
    if(concatenated_string != NULL){
        HAL_UART_Transmit(&huart1, concatenated_string->string, concatenated_string->length, HAL_MAX_DELAY);
        static_strings_deallocate(concatenate);
        static_strings_deallocate(concatenated_string);
    }
}
```

1.2.12 Concatenate multiple strings

```
static_strings_string_descriptor *one = static_strings_save((uint8_t *) "I am a ");
static_strings_string_descriptor *two = static_strings_save((uint8_t *) "concatenate all ");
static_strings_string_descriptor *three = static_strings_save((uint8_t *) "test\r\n");
if(one != NULL && two != NULL && three != NULL){
    static_strings_string_descriptor *concatenated_string = static_strings_concatenate_all(3, one, two, three);
    if(concatenated_string != NULL){
        HAL_UART_Transmit(&huart1, concatenated_string->string, concatenated_string->length, HAL_MAX_DELAY);
        static_strings_deallocate(one);
        static_strings_deallocate(two);
        static_strings_deallocate(three);
        static_strings_deallocate(concatenated_string);
    }
}
```

1.2.13 Configure quantity and size of the memory arrays

Just edit these constants in [static_strings.h](#)

```
#define STATIC_STRINGS_VERY_SHORT_STRING_SIZE 50
#define STATIC_STRINGS_VERY_SHORT_STRING_QUANTITY 10
#define STATIC_STRINGS_SHORT_STRING_SIZE 100
#define STATIC_STRINGS_SHORT_STRING_QUANTITY 6
#define STATIC_STRINGS_MEDIUM_STRING_SIZE 200
#define STATIC_STRINGS_MEDIUM_STRING_QUANTITY 2
#define STATIC_STRINGS_LONG_STRING_SIZE 500
#define STATIC_STRINGS_LONG_STRING_QUANTITY 1
#define STATIC_STRINGS_VERY_LONG_STRING_SIZE 1000
#define STATIC_STRINGS_VERY_LONG_STRING_QUANTITY 1
```

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

String types size and quantity	11
String types	12
String status	13
Error handling	14
Static memory arrays	15
String descriptors	16

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

static_strings_string_descriptor	
Meta data of a string	17
static_strings_string_splitter_parameters	
Definition of the structure to hold the parameters to static_strings_string_splitter_get_next_token function	17

Chapter 4

File Index

4.1 File List

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

int_types.h	The fprintf() PRI[d,u,x,o,i,X][8,16,32] macros for 32 bits signed and unsigned integers	19
static_strings.c	Strings allocation with static memory	19
static_strings.h	Strings allocation with static memory	34

Chapter 5

Module Documentation

5.1 String types size and quantity

Constants to reserve a memory for the different types of strings according to their length.

Macros

- `#define STATIC_STRINGS_VERY_SHORT_STRING_SIZE 50`
- `#define STATIC_STRINGS_VERY_SHORT_STRING_QUANTITY 10`
- `#define STATIC_STRINGS_SHORT_STRING_SIZE 100`
- `#define STATIC_STRINGS_SHORT_STRING_QUANTITY 6`
- `#define STATIC_STRINGS_MEDIUM_STRING_SIZE 200`
- `#define STATIC_STRINGS_MEDIUM_STRING_QUANTITY 2`
- `#define STATIC_STRINGS_LONG_STRING_SIZE 500`
- `#define STATIC_STRINGS_LONG_STRING_QUANTITY 1`
- `#define STATIC_STRINGS_VERY_LONG_STRING_SIZE 1000`
- `#define STATIC_STRINGS_VERY_LONG_STRING_QUANTITY 1`

5.1.1 Detailed Description

Constants to reserve a memory for the different types of strings according to their length.

5.2 String types

Constants to identify the different types of strings according to their length.

Macros

- `#define STATIC_STRINGS_STRING_TYPE_VERY_SHORT 0`
- `#define STATIC_STRINGS_STRING_TYPE_SHORT 1`
- `#define STATIC_STRINGS_STRING_TYPE_MEDIUM 2`
- `#define STATIC_STRINGS_STRING_TYPE_LONG 3`
- `#define STATIC_STRINGS_STRING_TYPE_VERY_LONG 4`
- `#define STATIC_STRINGS_STRING_TYPE_CUSTOM 5`

5.2.1 Detailed Description

Constants to identify the different types of strings according to their length.

5.3 String status

Constants to define the status of a string.

Macros

- `#define STATIC_STRINGS_STRING_STATUS_DEALLOCATED 0`
- `#define STATIC_STRINGS_STRING_STATUS_ALLOCATED 1`
- `#define STATIC_STRINGS_STRING_STATUS_CONSTANT 2`

5.3.1 Detailed Description

Constants to define the status of a string.

5.4 Error handling

Error codes.

Macros

- `#define STATIC_STRINGS_ERROR_CODE_NO_MEMORY_AVAILABLE 0`
- `#define STATIC_STRINGS_ERROR_CODE_INVALID_STRING 1`
- `#define STATIC_STRINGS_ERROR_CODE_STRING_TOO_LONG 2`
- `#define STATIC_STRINGS_ERROR_CODE_SUBSTRING_START_INDEX_OUT_OF_RANGE 3`
- `#define STATIC_STRINGS_ERROR_CODE_SUBSTRING_FINISH_INDEX_OUT_OF_RANGE 4`
- `#define STATIC_STRINGS_ERROR_CODE_STRING_OVERFLOW 5`

Variables

- `uint8_t static_strings_error_code`
Global variable to store error code.

5.4.1 Detailed Description

Error codes.

5.4.2 Variable Documentation

5.4.2.1 static_strings_error_code

`uint8_t static_strings_error_code`

Global variable to store error code.

`static_strings_error_code`

5.5 Static memory arrays

Static memory arrays to allocate strings.

Variables

- `uint8_t static_strings_very_short_string_memory` [STATIC_STRINGS_VERY_SHORT_STRING_QUANTITY][STATIC_STRINGS_VERY_SHORT_STRING_SIZE]
- `uint8_t static_strings_short_string_memory` [STATIC_STRINGS_SHORT_STRING_QUANTITY][STATIC_STRINGS_SHORT_STRING_SIZE]
- `uint8_t static_strings_medium_string_memory` [STATIC_STRINGS_MEDIUM_STRING_QUANTITY][STATIC_STRINGS_MEDIUM_STRING_SIZE]
- `uint8_t static_strings_long_string_memory` [STATIC_STRINGS_LONG_STRING_QUANTITY][STATIC_STRINGS_LONG_STRING_SIZE]
- `uint8_t static_strings_very_long_string_memory` [STATIC_STRINGS_VERY_LONG_STRING_QUANTITY][STATIC_STRINGS_VERY_LONG_STRING_SIZE]

5.5.1 Detailed Description

Static memory arrays to allocate strings.

5.6 String descriptors

Descriptors for all the string types.

Variables

- [static_strings_string_descriptor](#) **static_strings_very_short_strings_descriptors** [STATIC_STRINGS_VERY_SHORT_STRING_QUANTITY]
- [static_strings_string_descriptor](#) **static_strings_short_strings_descriptors** [STATIC_STRINGS_SHORT_STRING_QUANTITY]
- [static_strings_string_descriptor](#) **static_strings_medium_strings_descriptors** [STATIC_STRINGS_MEDIUM_STRING_QUANTITY]
- [static_strings_string_descriptor](#) **static_strings_long_strings_descriptors** [STATIC_STRINGS_LONG_STRING_QUANTITY]
- [static_strings_string_descriptor](#) **static_strings_very_long_strings_descriptors** [STATIC_STRINGS_VERY_LONG_STRING_QUANTITY]

5.6.1 Detailed Description

Descriptors for all the string types.

Chapter 6

Data Structure Documentation

6.1 static_strings_string_descriptor Struct Reference

Meta data of a string.

```
#include <static_strings.h>
```

Data Fields

- `uint8_t * string`
- `uint16_t length`
- `uint8_t type`
- `uint8_t status`

6.1.1 Detailed Description

Meta data of a string.

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

- [static_strings.h](#)

6.2 static_strings_string_splitter_parameters Struct Reference

Definition of the structure to hold the parameters to static_strings_string_splitter_get_next_token function.

```
#include <static_strings.h>
```

Data Fields

- [static_strings_string_descriptor](#) * `string_descriptor`
- `uint8_t * next_token_start`
- `uint8_t delimiter`

6.2.1 Detailed Description

Definition of the structure to hold the parameters to static_strings_string_splitter_get_next_token function.

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

- [static_strings.h](#)

Chapter 7

File Documentation

7.1 `int_types.h` File Reference

The `fprintf()` `PR[d,u,x,o,i,X][8,16,32]` macros for 32 bits signed and unsigned integers.

Macros

- `#define PRId8 "hd"`
- `#define PRId16 "d"`
- `#define PRId32 "ld"`
- `#define PRIu8 "hu"`
- `#define PRIu16 "u"`
- `#define PRIu32 "lu"`
- `#define PRIx8 "hX"`
- `#define PRIx16 "X"`
- `#define PRIx32 "lX"`
- `#define PRlo8 "ho"`
- `#define PRlo16 "o"`
- `#define PRlo32 "lo"`
- `#define PRIi8 "hi"`
- `#define PRIi16 "i"`
- `#define PRIi32 "li"`
- `#define PRIx8 "hX"`
- `#define PRIx16 "X"`
- `#define PRIx32 "lX"`

7.1.1 Detailed Description

The `fprintf()` `PR[d,u,x,o,i,X][8,16,32]` macros for 32 bits signed and unsigned integers.

7.2 `static_strings.c` File Reference

Strings allocation with static memory.

```
#include "static_strings.h"
```

Functions

- void `static_strings_init ()`
Link the descriptors with the arrays and initialize the status as deallocated.
- int `static_strings_get_string_max_length (static_strings_string_descriptor *string)`
get the maximum length allowed by the type of the string.
- `static_strings_string_descriptor * static_strings_copy (static_strings_string_descriptor *copy_to, static_strings_string_descriptor *copy_from, uint16_t copy_to_offset)`
Copy a string into another at determinate offset. Leaves intact the string values before the offset. Can throw `STATIC_↵IC_STRINGS_ERROR_CODE_STRING_OVERFLOW`.
- `static_strings_string_descriptor * static_strings_move (static_strings_string_descriptor *move_to, static_strings_string_descriptor *move_from, uint16_t move_to_offset)`
Move a string into another at determinate offset, if success the move_to string is deallocated. Can throw `STATIC_↵STRINGS_ERROR_CODE_STRING_OVERFLOW`. Leaves intact the string values before the offset.
- `static_strings_string_descriptor * static_strings_clone (static_strings_string_descriptor *clone_from)`
Clone a string into a new one.
- `static_strings_string_descriptor * static_strings_allocate (uint16_t string_size)`
Request memory for a string with its size, the user must copy the string with the descriptor and specify the size. Also see `static_strings_save`.
- `static_strings_string_descriptor * static_strings_save (uint8_t *string)`
Calculate the string size, allocate memory, copy the string and set the size. String must end with `\r\n` or `\0`, if `\r` is found but `\n` is not found, it is added, size of string include line ending but not `\0`. Also see `static_strings_allocate`.
- int `static_strings_create_custom_string (static_strings_string_descriptor *string_descriptor, uint8_t *string)`
Bind the provided string descriptor with the data of a string. String must end with `\r\n` or `\0`.
- void `static_strings_deallocate (static_strings_string_descriptor *string_descriptor)`
Set the descriptor status as deallocated. Custom strings can't be deallocated.
- int `static_strings_is_line (static_strings_string_descriptor *string_descriptor)`
Look at the last two characters of a string to see if the string has a line ending `\r\n`.
- uint16_t `static_strings_strlen (uint8_t *string)`
Calculate the length of a string that ends with `\r\n` or `\0`, line ending is included in length. Maximum length is `STATIC_↵IC_STRINGS_VERY_LONG_STRING_SIZE`.
- void `static_strings_string_splitter_set_parameters (static_strings_string_descriptor *string_descriptor, uint8_t delimiter)`
Set the parameters to the `static_strings_string_splitter_get_next_token` function.
- int `static_strings_string_splitter_get_next_token (static_strings_string_descriptor **string_descriptor)`
Bind the provided string descriptor with the next token data. Can be placed in a while condition as it returns 1 if success or 0 if no token available and retrieves the token in the `string_descriptor` parameter. If no delimiter the whole string is taken as token. The token is placed in a new string.
- `static_strings_string_descriptor * static_strings_substring (static_strings_string_descriptor *string, uint16_t start_index, uint16_t finish_index)`
Return a new string with the characters between the `start_index` and the `finish_index`. Not including the character at `finish_index`. Returned string has to be deallocated. To get all the string from a start index use the length in the `finish_index`.
- `static_strings_string_descriptor * static_strings_concatenate (static_strings_string_descriptor *concatenate_↵_at, static_strings_string_descriptor *concatenate)`
Concatenate the second string at the end of the first in a new string.
- `static_strings_string_descriptor * static_strings_concatenate_and_clean (static_strings_string_descriptor *concatenate_at, static_strings_string_descriptor *concatenate)`
Concatenate the second string at the end of the first in a new string and deallocate the concatenate at parameter if success.
- `static_strings_string_descriptor * static_strings_concatenate_all (uint16_t arguments_quantity,...)`
Concatenates multiple strings in the order of the arguments, the number of arguments must be provided in the first parameter. This function must be used careful.

- `int static_strings_contains_string (static_strings_string_descriptor *search_in, static_strings_string_descriptor *search_for)`
Search a string in other string.
- `int static_strings_contains_char (static_strings_string_descriptor *search_in, uint8_t search_for)`
Search a character in a string.
- `int static_strings_compare (static_strings_string_descriptor *compare_string_one, static_strings_string_descriptor *compare_string_two)`
Compare two strings to see if they are equals.
- `static_strings_string_descriptor * static_strings_uint8_to_string (uint8_t uint8)`
Create a string with the value of the parameter.
- `static_strings_string_descriptor * static_strings_uint16_to_string (uint16_t uint16)`
Create a string with the value of the parameter.
- `static_strings_string_descriptor * static_strings_uint32_to_string (uint32_t uint32)`
Create a string with the value of the parameter.
- `static_strings_string_descriptor * static_strings_int8_to_string (int8_t int8)`
Create a string with the value of the parameter.
- `static_strings_string_descriptor * static_strings_int16_to_string (int16_t int16)`
Create a string with the value of the parameter.
- `static_strings_string_descriptor * static_strings_int32_to_string (int32_t int32)`
Create a string with the value of the parameter.
- `static_strings_string_descriptor * static_strings_float_to_string (float float_arg)`
Create a string with the value of the parameter.
- `static_strings_string_descriptor * static_strings_double_to_string (double double_arg)`
Create a string with the value of the parameter.

Variables

- `static_strings_string_splitter_parameters static_strings_string_splitter = {NULL, '\0'}`

7.2.1 Detailed Description

Strings allocation with static memory.

7.2.2 Function Documentation

7.2.2.1 static_strings_allocate()

```
static_strings_string_descriptor* static_strings_allocate (
    uint16_t string_size )
```

Request memory for a string with its size, the user must copy the string with the descriptor and specify the size. Also see `static_strings_save`.

```
static_strings_string_descriptor *static_strings_allocate(uint16_t string_size)
```

Parameters

<i>string_size</i>	Size of the string in uint16_t.
--------------------	---------------------------------

Returns

A pointer to the string descriptor, if NULL check `static_strings_error_code`.

7.2.2.2 `static_strings_clone()`

```
static_strings_string_descriptor* static_strings_clone (
    static_strings_string_descriptor * clone_from )
```

Clone a string into a new one.

```
static_strings_string_descriptor *static_strings_clone(static_strings_string_descriptor *clone_from)
```

Parameters

<i>clone_from</i>	Pointer to the string to clone.
-------------------	---------------------------------

Returns

A pointer to the descriptor with the cloned string if success, if an error occur return NULL, check `static_strings_error_code` for further information.

7.2.2.3 `static_strings_compare()`

```
int static_strings_compare (
    static_strings_string_descriptor * compare_string_one,
    static_strings_string_descriptor * compare_string_two )
```

Compare two strings to see if they are equals.

```
int static_strings_compare(static_strings_string_descriptor* compare_string_one,static_strings_string_descriptor* compare_string_two)
```

Parameters

<i>compare_string_one</i>	A pointer to the first string to compare.
<i>compare_string_two</i>	A pointer to the second string to compare.

Returns

A pointer to the string descriptor with the concatenated string, if NULL check static_strings_error_code.

7.2.2.4 static_strings_concatenate()

```
static_strings_string_descriptor* static_strings_concatenate (
    static_strings_string_descriptor * concatenate_at,
    static_strings_string_descriptor * concatenate )
```

Concatenate the second string at the end of the first in a new string.

`static_strings_string_descriptor` *static_strings_concatenate*(`static_strings_string_descriptor` concatenate_↔
at,static_strings_string_descriptor* concatenate)

Parameters

<i>concatenate_↔ _at</i>	A pointer to the string to concatenate at.
<i>concatenate</i>	A pointer to the string to concatenate at the end of the concatenate_at string.

Returns

A pointer to the string descriptor with the concatenated string, if NULL check static_strings_error_code.

7.2.2.5 static_strings_concatenate_all()

```
static_strings_string_descriptor* static_strings_concatenate_all (
    uint16_t arguments_quantity,
    ... )
```

Concatenates multiple strings in the order of the arguments, the number of arguments must be provided in the first parameter. This function must be used careful.

`static_strings_string_descriptor` *static_strings_concatenate_all(uint16_t arguments_quantity,...)

Parameters

<i>arguments_quantity</i>	The number of strings to concatenate.
...	Multiple arguments of type static_strigs_string_descriptor pointer.

Returns

A pointer to the string descriptor with the concatenated string, if NULL check static_strings_error_code.

7.2.2.6 static_strings_concatenate_and_clean()

```
static_strings_string_descriptor* static_strings_concatenate_and_clean (
    static_strings_string_descriptor * concatenate_at,
    static_strings_string_descriptor * concatenate )
```

Concatenate the second string at the end of the first in a new string and deallocate the concatenate at parameter if success.

```
static_strings_string_descriptor static_strings_concatenate_and_clean(static_strings_string_descriptor concatenate_at,static_strings_string_descriptor* concatenate)
```

Parameters

<i>concatenate_at</i>	A pointer to the string to concatenate at, it is deallocated if success.
<i>concatenate</i>	A pointer to the string to concatenate at the end of the concatenate_at string.

Returns

A pointer to the string descriptor with the concatenated string, if NULL check static_strings_error_code.

7.2.2.7 static_strings_contains_char()

```
int static_strings_contains_char (
    static_strings_string_descriptor * search_in,
    uint8_t search_for )
```

Search a character in a string.

```
int static_strings_contains_char(static_strings_string_descriptor* search_in,uint8_t search_for)
```

Parameters

<i>search_in</i>	A pointer to the string in which the character will be search.
<i>search_for</i>	The searched character.

Returns

1 if the character is found, 0 if not.

7.2.2.8 static_strings_contains_string()

```
int static_strings_contains_string (
    static_strings_string_descriptor * search_in,
    static_strings_string_descriptor * search_for )
```

Search a string in other string.

```
int static_strings_contains_string(static_strings_string_descriptor* search_in,static_strings_string_descriptor* search_for)
```

Parameters

<i>search_in</i>	A pointer to the string in which the character will be search.
<i>search_for</i>	A pointer to the searched string.

Returns

1 if the string is found, 0 if not.

7.2.2.9 static_strings_copy()

```
static_strings_string_descriptor* static_strings_copy (
    static_strings_string_descriptor * copy_to,
    static_strings_string_descriptor * copy_from,
    uint16_t copy_to_offset )
```

Copy a string into another at determinate offset. Leaves intact the string values before the offset. Can throw `STATIC_STRINGS_ERROR_CODE_STRING_OVERFLOW`.

```
static_strings_string_descriptor *static_strings_copy(static_strings_string_descriptor *copy_to,static_strings_string_descriptor
*copy_from,uint16_t copy_to_offset)
```

Parameters

<i>copy_to</i>	Pointer to the string to copy in. String must have a defined type and length before use this function
<i>copy_from</i>	Pointer to the string to copy from.
<i>copy_to_offset</i>	Start copy index.

Returns

A pointer to the descriptor with the copied string if success, if an error occur return NULL, check `static_strings_error_code` for further information.

7.2.2.10 static_strings_create_custom_string()

```
int static_strings_create_custom_string (
    static_strings_string_descriptor * string_descriptor,
    uint8_t * string )
```

Bind the provided string descriptor with the data of a string. String must end with `\r\n` or `\0`.

```
void static_strings_create_custom_string(static_strings_string_descriptor *string_descriptor,uint8_t *string)
```


Parameters

<i>string_descriptor</i>	A pointer to a string descriptor.
<i>string</i>	A pointer to the string to bind the descriptor.

Returns

Return the length of the string, if 0 check static_strings_error_code.

7.2.2.11 static_strings_deallocate()

```
void static_strings_deallocate (
    static_strings_string_descriptor * string_descriptor )
```

Set the descriptor status as deallocated. Custom strings can't be deallocated.

```
void static_strings_deallocate(static_strings_string_descriptor *string_descriptor)
```

Parameters

<i>string_descriptor</i>	A pointer to the string descriptor to deallocate.
--------------------------	---

7.2.2.12 static_strings_double_to_string()

```
static_strings_string_descriptor* static_strings_double_to_string (
    double double_arg )
```

Create a string with the value of the parameter.

```
static_strings_string_descriptor *static_strings_double_to_string(double double_arg)
```

Parameters

<i>double_arg</i>	32 bits signed float (double).
-------------------	--------------------------------

Returns

A pointer to the string descriptor with the parameter as string.

7.2.2.13 static_strings_float_to_string()

```
static_strings_string_descriptor* static_strings_float_to_string (
    float float_arg )
```

Create a string with the value of the parameter.

`static_strings_string_descriptor *static_strings_float_to_string(float float_arg)`

Parameters

<i>float_arg</i>	16 bits signed float.
------------------	-----------------------

Returns

A pointer to the string descriptor with the parameter as string.

7.2.2.14 static_strings_get_string_max_length()

```
int static_strings_get_string_max_length (
    static_strings_string_descriptor * string )
```

get the maximum length allowed by the type of the string.

`int static_strings_get_string_max_length(static_strings_string_descriptor *string)`

Parameters

<i>string</i>	A pointer to a string descriptor.
---------------	-----------------------------------

Returns

The maximum allowed length of the string as an integer.

7.2.2.15 static_strings_init()

```
void static_strings_init ( )
```

Link the descriptors with the arrays and initialize the status as deallocated.

`void static_strings_init()`

7.2.2.16 static_strings_int16_to_string()

```
static_strings_string_descriptor* static_strings_int16_to_string (
    int16_t int16 )
```

Create a string with the value of the parameter.

`static_strings_string_descriptor *static_strings_int16_to_string(int16_t int16)`

Parameters

<i>int16</i>	16 bits signed integer.
--------------	-------------------------

Returns

A pointer to the string descriptor with the parameter as string.

7.2.2.17 static_strings_int32_to_string()

```
static_strings_string_descriptor* static_strings_int32_to_string (  
    int32_t int32 )
```

Create a string with the value of the parameter.

```
static_strings_string_descriptor *static_strings_int32_to_string(int32_t int32)
```

Parameters

<i>int32</i>	32 bits signed integer.
--------------	-------------------------

Returns

A pointer to the string descriptor with the parameter as string.

7.2.2.18 static_strings_int8_to_string()

```
static_strings_string_descriptor* static_strings_int8_to_string (  
    int8_t int8 )
```

Create a string with the value of the parameter.

```
static_strings_string_descriptor *static_strings_int8_to_string(int8_t int8)
```

Parameters

<i>int8</i>	8 bits signed integer.
-------------	------------------------

Returns

A pointer to the string descriptor with the parameter as string.

7.2.2.19 static_strings_is_line()

```
int static_strings_is_line (
    static_strings_string_descriptor * string_descriptor )
```

Look at the last two characters of a string to see if the string has a line ending `\r\n`.

```
int static_strings_is_line(static_strings_string_descriptor *string_descriptor)
```

Parameters

<i>string</i>	A pointer to the string descriptor.
---------------	-------------------------------------

Returns

Return 0 if the string doesn't have a line ending `\r\n` and 1 if the string has a line ending `\r\n`.

7.2.2.20 static_strings_move()

```
static_strings_string_descriptor* static_strings_move (
    static_strings_string_descriptor * move_to,
    static_strings_string_descriptor * move_from,
    uint16_t move_to_offset )
```

Move a string into another at determinate offset, if success the move_to string is deallocated. Can throw `STATIC_STRINGS_ERROR_CODE_STRING_OVERFLOW`. Leaves intact the string values before the offset.

```
static_strings_string_descriptor *static_strings_move(static_strings_string_descriptor *move_to,static_strings_string_descriptor
*move_from,uint16_t move_to_offset)
```

Parameters

<i>move_to</i>	Pointer to the string to move in. String must have a defined type and length before use this function
<i>move_from</i>	Pointer to the string to move from.
<i>move_to_offset</i>	Start move index.

Returns

A pointer to the descriptor with the moved string if success, if an error occur return NULL, check `static_strings_error_code` for further information.

7.2.2.21 static_strings_save()

```
static_strings_string_descriptor* static_strings_save (
    uint8_t * string )
```

Calculate the string size, allocate memory, copy the string and set the size. String must end with `\r\n` or `\0`, if `\r` is found but `\n` is not found, it is added, size of string include line ending but not `\0`. Also see `static_strings_allocate`.

`static_strings_string_descriptor *static_strings_save(uint8_t *string)`

Parameters

<i>string</i>	A pointer to the string start.
---------------	--------------------------------

Returns

A pointer to the string descriptor, if NULL check `static_strings_error_code`.

7.2.2.22 static_strings_string_splitter_get_next_token()

```
int static_strings_string_splitter_get_next_token (
    static_strings_string_descriptor ** string_descriptor )
```

Bind the provided string descriptor with the next token data. Can be placed in a while condition as it returns 1 if success or 0 if no token available and retrieves the token in the `string_descriptor` parameter. If no delimiter the whole string is taken as token. The token is placed in a new string.

`int static_strings_string_splitter_get_next_token(static_strings_string_descriptor **string_descriptor)`

Parameters

<i>string_descriptor</i>	A pointer to a pointer to a string descriptor that will contain the token.
--------------------------	--

Returns

1 if success or 0 if no token is available.

7.2.2.23 static_strings_string_splitter_set_parameters()

```
void static_strings_string_splitter_set_parameters (
    static_strings_string_descriptor * string_descriptor,
    uint8_t delimiter )
```

Set the parameters to the `static_strings_string_splitter_get_next_token` function.

`void static_strings_string_splitter_set_parameters(static_strings_string_descriptor *string_descriptor,uint8_t delimiter)`

Parameters

<i>string_descriptor</i>	A pointer to the string descriptor of the string to split.
<i>delimiter</i>	The delimiter for the tokens.

7.2.2.24 static_strings_strlen()

```
uint16_t static_strings_strlen (
    uint8_t * string )
```

Calculate the length of a string that ends with `\r\n` or `\0`, line ending is included in length. Maximum length is `STATIC_STRINGS_VERY_LONG_STRING_SIZE`.

```
uint16_t static_strings_strlen(uint8_t *string)
```

Parameters

<i>string</i>	A pointer to the string.
---------------	--------------------------

Returns

Length of the string in `uint16_t`. If 0 check `static_strings_error_code`.

7.2.2.25 static_strings_substring()

```
static_strings_string_descriptor* static_strings_substring (
    static_strings_string_descriptor * string,
    uint16_t start_index,
    uint16_t finish_index )
```

Return a new string with the characters between the `start_index` and the `finish_index`. Not including the character at `finish_index`. Returned string has to be deallocated. To get all the string from a start index use the length in the `finish_index`.

```
static_strings_string_descriptor static_strings_substring(static_strings_string_descriptor string_descriptor,uint16_t start_index,uint16_t finish_index)
```

Parameters

<i>string_descriptor</i>	A pointer to the string which contains the substring.
<i>start_index</i>	The index of the first character.
<i>finish_index</i>	The index of the last character, not included.

Returns

A pointer to the string descriptor of the substring, if NULL check `static_strings_error_code`.

7.2.2.26 static_strings_uint16_to_string()

```
static_strings_string_descriptor* static_strings_uint16_to_string (
    uint16_t uint16 )
```

Create a string with the value of the parameter.

```
static_strings_string_descriptor *static_strings_uint16_to_string(uint16_t uint16)
```

Parameters

<i>uint16</i>	16 bits unsigned integer.
---------------	---------------------------

Returns

A pointer to the string descriptor with the parameter as string.

7.2.2.27 static_strings_uint32_to_string()

```
static_strings_string_descriptor* static_strings_uint32_to_string (
    uint32_t uint32 )
```

Create a string with the value of the parameter.

```
static_strings_string_descriptor *static_strings_uint32_to_string(uint32_t uint32)
```

Parameters

<i>uint32</i>	32 bits unsigned integer.
---------------	---------------------------

Returns

A pointer to the string descriptor with the parameter as string.

7.2.2.28 static_strings_uint8_to_string()

```
static_strings_string_descriptor* static_strings_uint8_to_string (
    uint8_t uint8 )
```

Create a string with the value of the parameter.

```
static_strings_string_descriptor *static_strings_uint8_to_string(uint8_t uint8)
```

Parameters

<code>uint8</code>	8 bits unsigned integer.
--------------------	--------------------------

Returns

A pointer to the string descriptor with the parameter as string.

7.2.3 Variable Documentation

7.2.3.1 `static_strings_string_splitter`

```
static_strings_string_splitter_parameters static_strings_string_splitter = {NULL, '\0'}
```

Parameters to `static_strings_string_splitter_get_next_token` function. Initialized in null and \0.

7.3 `static_strings.h` File Reference

Strings allocation with static memory.

```
#include "stm32f1xx_hal.h"
#include "string.h"
#include "int_types.h"
#include "stdarg.h"
#include "stdio.h"
```

Data Structures

- struct `static_strings_string_descriptor`

Meta data of a string.

- struct `static_strings_string_splitter_parameters`

Definition of the structure to hold the parameters to `static_strings_string_splitter_get_next_token` function.

Macros

- #define `STATIC_STRINGS_VERY_SHORT_STRING_SIZE` 50
- #define `STATIC_STRINGS_VERY_SHORT_STRING_QUANTITY` 10
- #define `STATIC_STRINGS_SHORT_STRING_SIZE` 100
- #define `STATIC_STRINGS_SHORT_STRING_QUANTITY` 6
- #define `STATIC_STRINGS_MEDIUM_STRING_SIZE` 200
- #define `STATIC_STRINGS_MEDIUM_STRING_QUANTITY` 2
- #define `STATIC_STRINGS_LONG_STRING_SIZE` 500
- #define `STATIC_STRINGS_LONG_STRING_QUANTITY` 1
- #define `STATIC_STRINGS_VERY_LONG_STRING_SIZE` 1000
- #define `STATIC_STRINGS_VERY_LONG_STRING_QUANTITY` 1
- #define `STATIC_STRINGS_STRING_TYPE_VERY_SHORT` 0
- #define `STATIC_STRINGS_STRING_TYPE_SHORT` 1
- #define `STATIC_STRINGS_STRING_TYPE_MEDIUM` 2
- #define `STATIC_STRINGS_STRING_TYPE_LONG` 3
- #define `STATIC_STRINGS_STRING_TYPE_VERY_LONG` 4
- #define `STATIC_STRINGS_STRING_TYPE_CUSTOM` 5
- #define `STATIC_STRINGS_STRING_STATUS_DEALLOCATED` 0
- #define `STATIC_STRINGS_STRING_STATUS_ALLOCATED` 1
- #define `STATIC_STRINGS_STRING_STATUS_CONSTANT` 2
- #define `STATIC_STRINGS_ERROR_CODE_NO_MEMORY_AVAILABLE` 0
- #define `STATIC_STRINGS_ERROR_CODE_INVALID_STRING` 1
- #define `STATIC_STRINGS_ERROR_CODE_STRING_TOO_LONG` 2
- #define `STATIC_STRINGS_ERROR_CODE_SUBSTRING_START_INDEX_OUT_OF_RANGE` 3
- #define `STATIC_STRINGS_ERROR_CODE_SUBSTRING_FINISH_INDEX_OUT_OF_RANGE` 4
- #define `STATIC_STRINGS_ERROR_CODE_STRING_OVERFLOW` 5

Typedefs

- typedef struct `static_strings_string_descriptor` `static_strings_string_descriptor`
- typedef struct `static_strings_string_splitter_parameters` `static_strings_string_splitter_parameters`

Functions

- void `static_strings_init` ()
Link the descriptors with the arrays and initialize the status as deallocated.
- int `static_strings_get_string_max_length` (`static_strings_string_descriptor` *string)
get the maximum length allowed by the type of the string.
- `static_strings_string_descriptor` * `static_strings_copy` (`static_strings_string_descriptor` *copy_to, `static_strings_string_descriptor` *copy_from, uint16_t copy_to_offset)
Copy a string into another at determinate offset. Leaves intact the string values before the offset. Can throw `STATIC_STRINGS_ERROR_CODE_STRING_OVERFLOW`.
- `static_strings_string_descriptor` * `static_strings_move` (`static_strings_string_descriptor` *move_to, `static_strings_string_descriptor` *move_from, uint16_t move_to_offset)
Move a string into another at determinate offset, if success the move_to string is deallocated. Can throw `STATIC_STRINGS_ERROR_CODE_STRING_OVERFLOW`. Leaves intact the string values before the offset.
- `static_strings_string_descriptor` * `static_strings_clone` (`static_strings_string_descriptor` *clone_from)
Clone a string into a new one.
- `static_strings_string_descriptor` * `static_strings_allocate` (uint16_t string_size)
Request memory for a string with its size, the user must copy the string with the descriptor and specify the size. Also see `static_strings_save`.

- `static_strings_string_descriptor * static_strings_save (uint8_t *string)`
Calculate the string size, allocate memory, copy the string and set the size. String must end with `\r\n` or `\0`, if `\r` is found but `\n` is not found, it is added, size of string include line ending but not `\0`. Also see `static_strings_allocate`.
- `int static_strings_create_custom_string (static_strings_string_descriptor *string_descriptor, uint8_t *string)`
Bind the provided string descriptor with the data of a string. String must end with `\r\n` or `\0`.
- `void static_strings_deallocate (static_strings_string_descriptor *string_descriptor)`
Set the descriptor status as deallocated. Custom strings can't be deallocated.
- `int static_strings_is_line (static_strings_string_descriptor *string_descriptor)`
Look at the last two characters of a string to see if the string has a line ending `\r\n`.
- `uint16_t static_strings_strlen (uint8_t *string)`
Calculate the length of a string that ends with `\r\n` or `\0`, line ending is included in length. Maximum length is `STATIC_STRINGS_VERY_LONG_STRING_SIZE`.
- `void static_strings_string_splitter_set_parameters (static_strings_string_descriptor *string_descriptor, uint8_t delimiter)`
Set the parameters to the `static_strings_string_splitter_get_next_token` function.
- `int static_strings_string_splitter_get_next_token (static_strings_string_descriptor **string_descriptor)`
Bind the provided string descriptor with the next token data. Can be placed in a while condition as it returns 1 if success or 0 if no token available and retrieves the token in the `string_descriptor` parameter. If no delimiter the whole string is taken as token. The token is placed in a new string.
- `static_strings_string_descriptor * static_strings_substring (static_strings_string_descriptor *string, uint16_t start_index, uint16_t finish_index)`
Return a new string with the characters between the `start_index` and the `finish_index`. Not including the character at `finish_index`. Returned string has to be deallocated. To get all the string from a start index use the length in the `finish_index`.
- `static_strings_string_descriptor * static_strings_concatenate (static_strings_string_descriptor *concatenate_at, static_strings_string_descriptor *concatenate)`
Concatenate the second string at the end of the first in a new string.
- `static_strings_string_descriptor * static_strings_concatenate_and_clean (static_strings_string_descriptor *concatenate_at, static_strings_string_descriptor *concatenate)`
Concatenate the second string at the end of the first in a new string and deallocate the concatenate at parameter if success.
- `static_strings_string_descriptor * static_strings_concatenate_all (uint16_t arguments_quantity,...)`
Concatenates multiple strings in the order of the arguments, the number of arguments must be provided in the first parameter. This function must be used careful.
- `int static_strings_contains_string (static_strings_string_descriptor *search_in, static_strings_string_descriptor *search_for)`
Search a string in other string.
- `int static_strings_contains_char (static_strings_string_descriptor *search_in, uint8_t search_for)`
Search a character in a string.
- `int static_strings_compare (static_strings_string_descriptor *compare_string_one, static_strings_string_descriptor *compare_string_two)`
Compare two strings to see if they are equals.
- `static_strings_string_descriptor * static_strings_uint8_to_string (uint8_t uint8)`
Create a string with the value of the parameter.
- `static_strings_string_descriptor * static_strings_uint16_to_string (uint16_t uint16)`
Create a string with the value of the parameter.
- `static_strings_string_descriptor * static_strings_uint32_to_string (uint32_t uint32)`
Create a string with the value of the parameter.
- `static_strings_string_descriptor * static_strings_int8_to_string (int8_t int8)`
Create a string with the value of the parameter.
- `static_strings_string_descriptor * static_strings_int16_to_string (int16_t int16)`
Create a string with the value of the parameter.
- `static_strings_string_descriptor * static_strings_int32_to_string (int32_t int32)`

Create a string with the value of the parameter.

- [static_strings_string_descriptor](#) * [static_strings_float_to_string](#) (float float_arg)

Create a string with the value of the parameter.

- [static_strings_string_descriptor](#) * [static_strings_double_to_string](#) (double double_arg)

Create a string with the value of the parameter.

Variables

- `uint8_t` [static_strings_error_code](#)

Global variable to store error code.

- [static_strings_string_splitter_parameters](#) [static_strings_string_splitter](#)
- `uint8_t` [static_strings_very_short_string_memory](#) [STATIC_STRINGS_VERY_SHORT_STRING_QUANTITY][STATIC_STRINGS_VERY_SHORT_STRING_SIZE]
- `uint8_t` [static_strings_short_string_memory](#) [STATIC_STRINGS_SHORT_STRING_QUANTITY][STATIC_STRINGS_SHORT_STRING_SIZE]
- `uint8_t` [static_strings_medium_string_memory](#) [STATIC_STRINGS_MEDIUM_STRING_QUANTITY][STATIC_STRINGS_MEDIUM_STRING_SIZE]
- `uint8_t` [static_strings_long_string_memory](#) [STATIC_STRINGS_LONG_STRING_QUANTITY][STATIC_STRINGS_LONG_STRING_SIZE]
- `uint8_t` [static_strings_very_long_string_memory](#) [STATIC_STRINGS_VERY_LONG_STRING_QUANTITY][STATIC_STRINGS_VERY_LONG_STRING_SIZE]
- [static_strings_string_descriptor](#) [static_strings_very_short_strings_descriptors](#) [STATIC_STRINGS_VERY_SHORT_STRING_QUANTITY]
- [static_strings_string_descriptor](#) [static_strings_short_strings_descriptors](#) [STATIC_STRINGS_SHORT_STRING_QUANTITY]
- [static_strings_string_descriptor](#) [static_strings_medium_strings_descriptors](#) [STATIC_STRINGS_MEDIUM_STRING_QUANTITY]
- [static_strings_string_descriptor](#) [static_strings_long_strings_descriptors](#) [STATIC_STRINGS_LONG_STRING_QUANTITY]
- [static_strings_string_descriptor](#) [static_strings_very_long_strings_descriptors](#) [STATIC_STRINGS_VERY_LONG_STRING_QUANTITY]

7.3.1 Detailed Description

Strings allocation with static memory.

7.3.2 Function Documentation

7.3.2.1 static_strings_allocate()

```
static_strings_string_descriptor* static_strings_allocate (
    uint16_t string_size )
```

Request memory for a string with its size, the user must copy the string with the descriptor and specify the size. Also see [static_strings_save](#).

```
static_strings_string_descriptor *static_strings_allocate(uint16_t string_size)
```

Parameters

<i>string_size</i>	Size of the string in uint16_t.
--------------------	---------------------------------

Returns

A pointer to the string descriptor, if NULL check `static_strings_error_code`.

7.3.2.2 static_strings_clone()

```
static_strings_string_descriptor* static_strings_clone (
    static_strings_string_descriptor * clone_from )
```

Clone a string into a new one.

```
static_strings_string_descriptor *static_strings_clone(static_strings_string_descriptor *clone_from)
```

Parameters

<i>clone_from</i>	Pointer to the string to clone.
-------------------	---------------------------------

Returns

A pointer to the descriptor with the cloned string if success, if an error occur return NULL, check `static_strings_error_code` for further information.

7.3.2.3 static_strings_compare()

```
int static_strings_compare (
    static_strings_string_descriptor * compare_string_one,
    static_strings_string_descriptor * compare_string_two )
```

Compare two strings to see if they are equals.

```
int static_strings_compare(static_strings_string_descriptor* compare_string_one,static_strings_string_descriptor* compare_string_two)
```

Parameters

<i>compare_string_one</i>	A pointer to the first string to compare.
<i>compare_string_two</i>	A pointer to the second string to compare.

Returns

A pointer to the string descriptor with the concatenated string, if NULL check static_strings_error_code.

7.3.2.4 static_strings_concatenate()

```
static_strings_string_descriptor* static_strings_concatenate (
    static_strings_string_descriptor * concatenate_at,
    static_strings_string_descriptor * concatenate )
```

Concatenate the second string at the end of the first in a new string.

`static_strings_string_descriptor` `static_strings_concatenate(static_strings_string_descriptor concatenate_↔
at,static_strings_string_descriptor* concatenate)`

Parameters

<code>concatenate_↔ _at</code>	A pointer to the string to concatenate at.
<code>concatenate</code>	A pointer to the string to concatenate at the end of the concatenate_at string.

Returns

A pointer to the string descriptor with the concatenated string, if NULL check static_strings_error_code.

7.3.2.5 static_strings_concatenate_all()

```
static_strings_string_descriptor* static_strings_concatenate_all (
    uint16_t arguments_quantity,
    ... )
```

Concatenates multiple strings in the order of the arguments, the number of arguments must be provided in the first parameter. This function must be used careful.

`static_strings_string_descriptor` *static_strings_concatenate_all(uint16_t arguments_quantity,...)

Parameters

<code>arguments_quantity</code>	The number of strings to concatenate.
<code>...</code>	Multiple arguments of type static_strigs_string_descriptor pointer.

Returns

A pointer to the string descriptor with the concatenated string, if NULL check static_strings_error_code.

7.3.2.6 static_strings_concatenate_and_clean()

```
static_strings_string_descriptor* static_strings_concatenate_and_clean (
    static_strings_string_descriptor * concatenate_at,
    static_strings_string_descriptor * concatenate )
```

Concatenate the second string at the end of the first in a new string and deallocate the concatenate at parameter if success.

```
static_strings_string_descriptor static_strings_concatenate_and_clean(static_strings_string_descriptor concatenate_at,static_strings_string_descriptor* concatenate)
```

Parameters

<i>concatenate_at</i>	A pointer to the string to concatenate at, it is deallocated if success.
<i>concatenate</i>	A pointer to the string to concatenate at the end of the concatenate_at string.

Returns

A pointer to the string descriptor with the concatenated string, if NULL check static_strings_error_code.

7.3.2.7 static_strings_contains_char()

```
int static_strings_contains_char (
    static_strings_string_descriptor * search_in,
    uint8_t search_for )
```

Search a character in a string.

```
int static_strings_contains_char(static_strings_string_descriptor* search_in,uint8_t search_for)
```

Parameters

<i>search_in</i>	A pointer to the string in which the character will be search.
<i>search_for</i>	The searched character.

Returns

1 if the character is found, 0 if not.

7.3.2.8 static_strings_contains_string()

```
int static_strings_contains_string (
    static_strings_string_descriptor * search_in,
    static_strings_string_descriptor * search_for )
```

Search a string in other string.

```
int static_strings_contains_string(static_strings_string_descriptor* search_in,static_strings_string_descriptor* search_for)
```

Parameters

<i>search_in</i>	A pointer to the string in which the character will be search.
<i>search_for</i>	A pointer to the searched string.

Returns

1 if the string is found, 0 if not.

7.3.2.9 static_strings_copy()

```
static_strings_string_descriptor* static_strings_copy (
    static_strings_string_descriptor * copy_to,
    static_strings_string_descriptor * copy_from,
    uint16_t copy_to_offset )
```

Copy a string into another at determinate offset. Leaves intact the string values before the offset. Can throw `STATIC_STRINGS_ERROR_CODE_STRING_OVERFLOW`.

```
static_strings_string_descriptor *static_strings_copy(static_strings_string_descriptor *copy_to,static_strings_string_descriptor
*copy_from,uint16_t copy_to_offset)
```

Parameters

<i>copy_to</i>	Pointer to the string to copy in. String must have a defined type and length before use this function
<i>copy_from</i>	Pointer to the string to copy from.
<i>copy_to_offset</i>	Start copy index.

Returns

A pointer to the descriptor with the copied string if success, if an error occur return NULL, check `static_strings_error_code` for further information.

7.3.2.10 static_strings_create_custom_string()

```
int static_strings_create_custom_string (
    static_strings_string_descriptor * string_descriptor,
    uint8_t * string )
```

Bind the provided string descriptor with the data of a string. String must end with `\r\n` or `\0`.

```
void static_strings_create_custom_string(static_strings_string_descriptor *string_descriptor,uint8_t *string)
```


Parameters

<i>string_descriptor</i>	A pointer to a string descriptor.
<i>string</i>	A pointer to the string to bind the descriptor.

Returns

Return the length of the string, if 0 check static_strings_error_code.

7.3.2.11 static_strings_deallocate()

```
void static_strings_deallocate (
    static_strings_string_descriptor * string_descriptor )
```

Set the descriptor status as deallocated. Custom strings can't be deallocated.

```
void static_strings_deallocate(static_strings_string_descriptor *string_descriptor)
```

Parameters

<i>string_descriptor</i>	A pointer to the string descriptor to deallocate.
--------------------------	---

7.3.2.12 static_strings_double_to_string()

```
static_strings_string_descriptor* static_strings_double_to_string (
    double double_arg )
```

Create a string with the value of the parameter.

```
static_strings_string_descriptor *static_strings_double_to_string(double double_arg)
```

Parameters

<i>double_arg</i>	32 bits signed float (double).
-------------------	--------------------------------

Returns

A pointer to the string descriptor with the parameter as string.

7.3.2.13 static_strings_float_to_string()

```
static_strings_string_descriptor* static_strings_float_to_string (
    float float_arg )
```

Create a string with the value of the parameter.

`static_strings_string_descriptor *static_strings_float_to_string(float float_arg)`

Parameters

<i>float_arg</i>	16 bits signed float.
------------------	-----------------------

Returns

A pointer to the string descriptor with the parameter as string.

7.3.2.14 static_strings_get_string_max_length()

```
int static_strings_get_string_max_length (
    static_strings_string_descriptor * string )
```

get the maximum length allowed by the type of the string.

`int static_strings_get_string_max_length(static_strings_string_descriptor *string)`

Parameters

<i>string</i>	A pointer to a string descriptor.
---------------	-----------------------------------

Returns

The maximum allowed length of the string as an integer.

7.3.2.15 static_strings_init()

```
void static_strings_init ( )
```

Link the descriptors with the arrays and initialize the status as deallocated.

`void static_strings_init()`

7.3.2.16 static_strings_int16_to_string()

```
static_strings_string_descriptor* static_strings_int16_to_string (
    int16_t int16 )
```

Create a string with the value of the parameter.

`static_strings_string_descriptor *static_strings_int16_to_string(int16_t int16)`

Parameters

<i>int16</i>	16 bits signed integer.
--------------	-------------------------

Returns

A pointer to the string descriptor with the parameter as string.

7.3.2.17 static_strings_int32_to_string()

```
static_strings_string_descriptor* static_strings_int32_to_string (  
    int32_t int32 )
```

Create a string with the value of the parameter.

```
static_strings_string_descriptor *static_strings_int32_to_string(int32_t int32)
```

Parameters

<i>int32</i>	32 bits signed integer.
--------------	-------------------------

Returns

A pointer to the string descriptor with the parameter as string.

7.3.2.18 static_strings_int8_to_string()

```
static_strings_string_descriptor* static_strings_int8_to_string (  
    int8_t int8 )
```

Create a string with the value of the parameter.

```
static_strings_string_descriptor *static_strings_int8_to_string(int8_t int8)
```

Parameters

<i>int8</i>	8 bits signed integer.
-------------	------------------------

Returns

A pointer to the string descriptor with the parameter as string.

7.3.2.19 static_strings_is_line()

```
int static_strings_is_line (
    static_strings_string_descriptor * string_descriptor )
```

Look at the last two characters of a string to see if the string has a line ending `\r\n`.

```
int static_strings_is_line(static_strings_string_descriptor *string_descriptor)
```

Parameters

<i>string</i>	A pointer to the string descriptor.
---------------	-------------------------------------

Returns

Return 0 if the string doesn't have a line ending `\r\n` and 1 if the string has a line ending `\r\n`.

7.3.2.20 static_strings_move()

```
static_strings_string_descriptor* static_strings_move (
    static_strings_string_descriptor * move_to,
    static_strings_string_descriptor * move_from,
    uint16_t move_to_offset )
```

Move a string into another at determinate offset, if success the move_to string is deallocated. Can throw `STATIC_STRINGS_ERROR_CODE_STRING_OVERFLOW`. Leaves intact the string values before the offset.

```
static_strings_string_descriptor *static_strings_move(static_strings_string_descriptor *move_to,static_strings_string_descriptor
*move_from,uint16_t move_to_offset)
```

Parameters

<i>move_to</i>	Pointer to the string to move in. String must have a defined type and length before use this function
<i>move_from</i>	Pointer to the string to move from.
<i>move_to_offset</i>	Start move index.

Returns

A pointer to the descriptor with the moved string if success, if an error occur return NULL, check `static_strings_error_code` for further information.

7.3.2.21 static_strings_save()

```
static_strings_string_descriptor* static_strings_save (
    uint8_t * string )
```

Calculate the string size, allocate memory, copy the string and set the size. String must end with `\r\n` or `\0`, if `\r` is found but `\n` is not found, it is added, size of string include line ending but not `\0`. Also see `static_strings_allocate`.

`static_strings_string_descriptor *static_strings_save(uint8_t *string)`

Parameters

<i>string</i>	A pointer to the string start.
---------------	--------------------------------

Returns

A pointer to the string descriptor, if NULL check `static_strings_error_code`.

7.3.2.22 static_strings_string_splitter_get_next_token()

```
int static_strings_string_splitter_get_next_token (
    static_strings_string_descriptor ** string_descriptor )
```

Bind the provided string descriptor with the next token data. Can be placed in a while condition as it returns 1 if success or 0 if no token available and retrieves the token in the `string_descriptor` parameter. If no delimiter the whole string is taken as token. The token is placed in a new string.

`int static_strings_string_splitter_get_next_token(static_strings_string_descriptor **string_descriptor)`

Parameters

<i>string_descriptor</i>	A pointer to a pointer to a string descriptor that will contain the token.
--------------------------	--

Returns

1 if success or 0 if no token is available.

7.3.2.23 static_strings_string_splitter_set_parameters()

```
void static_strings_string_splitter_set_parameters (
    static_strings_string_descriptor * string_descriptor,
    uint8_t delimiter )
```

Set the parameters to the `static_strings_string_splitter_get_next_token` function.

`void static_strings_string_splitter_set_parameters(static_strings_string_descriptor *string_descriptor,uint8_t delimiter)`

Parameters

<i>string_descriptor</i>	A pointer to the string descriptor of the string to split.
<i>delimiter</i>	The delimiter for the tokens.

7.3.2.24 static_strings_strlen()

```
uint16_t static_strings_strlen (
    uint8_t * string )
```

Calculate the length of a string that ends with `\r\n` or `\0`, line ending is included in length. Maximum length is `STATIC_STRINGS_VERY_LONG_STRING_SIZE`.

```
uint16_t static_strings_strlen(uint8_t *string)
```

Parameters

<i>string</i>	A pointer to the string.
---------------	--------------------------

Returns

Length of the string in `uint16_t`. If 0 check `static_strings_error_code`.

7.3.2.25 static_strings_substring()

```
static_strings_string_descriptor* static_strings_substring (
    static_strings_string_descriptor * string,
    uint16_t start_index,
    uint16_t finish_index )
```

Return a new string with the characters between the `start_index` and the `finish_index`. Not including the character at `finish_index`. Returned string has to be deallocated. To get all the string from a start index use the length in the `finish_index`.

```
static_strings_string_descriptor static_strings_substring(static_strings_string_descriptor string_descriptor,uint16_t start_index,uint16_t finish_index)
```

Parameters

<i>string_descriptor</i>	A pointer to the string which contains the substring.
<i>start_index</i>	The index of the first character.
<i>finish_index</i>	The index of the last character, not included.

Returns

A pointer to the string descriptor of the substring, if NULL check `static_strings_error_code`.

7.3.2.26 static_strings_uint16_to_string()

```
static_strings_string_descriptor* static_strings_uint16_to_string (
    uint16_t uint16 )
```

Create a string with the value of the parameter.

```
static_strings_string_descriptor *static_strings_uint16_to_string(uint16_t uint16)
```

Parameters

<i>uint16</i>	16 bits unsigned integer.
---------------	---------------------------

Returns

A pointer to the string descriptor with the parameter as string.

7.3.2.27 static_strings_uint32_to_string()

```
static_strings_string_descriptor* static_strings_uint32_to_string (
    uint32_t uint32 )
```

Create a string with the value of the parameter.

```
static_strings_string_descriptor *static_strings_uint32_to_string(uint32_t uint32)
```

Parameters

<i>uint32</i>	32 bits unsigned integer.
---------------	---------------------------

Returns

A pointer to the string descriptor with the parameter as string.

7.3.2.28 static_strings_uint8_to_string()

```
static_strings_string_descriptor* static_strings_uint8_to_string (
    uint8_t uint8 )
```

Create a string with the value of the parameter.

```
static_strings_string_descriptor *static_strings_uint8_to_string(uint8_t uint8)
```

Parameters

<i>uint8</i>	8 bits unsigned integer.
--------------	--------------------------

Returns

A pointer to the string descriptor with the parameter as string.

7.3.3 Variable Documentation

7.3.3.1 static_strings_string_splitter

`static_strings_string_splitter_parameters` `static_strings_string_splitter`

Parameters to `static_strings_string_splitter_get_next_token` function. Initialized in null and \0.

Index

Error handling, [14](#)

static_strings_error_code, [14](#)

int_types.h, [19](#)

Static memory arrays, [15](#)

static_strings.c, [19](#)

static_strings_allocate, [21](#)

static_strings_clone, [22](#)

static_strings_compare, [22](#)

static_strings_concatenate, [23](#)

static_strings_concatenate_all, [23](#)

static_strings_concatenate_and_clean, [23](#)

static_strings_contains_char, [24](#)

static_strings_contains_string, [24](#)

static_strings_copy, [26](#)

static_strings_create_custom_string, [26](#)

static_strings_deallocate, [27](#)

static_strings_double_to_string, [27](#)

static_strings_float_to_string, [27](#)

static_strings_get_string_max_length, [28](#)

static_strings_init, [28](#)

static_strings_int16_to_string, [28](#)

static_strings_int32_to_string, [29](#)

static_strings_int8_to_string, [29](#)

static_strings_is_line, [29](#)

static_strings_move, [30](#)

static_strings_save, [30](#)

static_strings_string_splitter, [34](#)

static_strings_string_splitter_get_next_token, [31](#)

static_strings_string_splitter_set_parameters, [31](#)

static_strings_strlen, [32](#)

static_strings_substring, [32](#)

static_strings_uint16_to_string, [32](#)

static_strings_uint32_to_string, [33](#)

static_strings_uint8_to_string, [33](#)

static_strings.h, [34](#)

static_strings_allocate, [37](#)

static_strings_clone, [38](#)

static_strings_compare, [38](#)

static_strings_concatenate, [39](#)

static_strings_concatenate_all, [39](#)

static_strings_concatenate_and_clean, [39](#)

static_strings_contains_char, [40](#)

static_strings_contains_string, [40](#)

static_strings_copy, [42](#)

static_strings_create_custom_string, [42](#)

static_strings_deallocate, [43](#)

static_strings_double_to_string, [43](#)

static_strings_float_to_string, [43](#)

static_strings_get_string_max_length, [44](#)

static_strings_init, [44](#)

static_strings_int16_to_string, [44](#)

static_strings_int32_to_string, [45](#)

static_strings_int8_to_string, [45](#)

static_strings_is_line, [45](#)

static_strings_move, [46](#)

static_strings_save, [46](#)

static_strings_string_splitter, [50](#)

static_strings_string_splitter_get_next_token, [47](#)

static_strings_string_splitter_set_parameters, [47](#)

static_strings_strlen, [48](#)

static_strings_substring, [48](#)

static_strings_uint16_to_string, [48](#)

static_strings_uint32_to_string, [49](#)

static_strings_uint8_to_string, [49](#)

static_strings_allocate

static_strings.c, [21](#)

static_strings.h, [37](#)

static_strings_clone

static_strings.c, [22](#)

static_strings.h, [38](#)

static_strings_compare

static_strings.c, [22](#)

static_strings.h, [38](#)

static_strings_concatenate

static_strings.c, [23](#)

static_strings.h, [39](#)

static_strings_concatenate_all

static_strings.c, [23](#)

static_strings.h, [39](#)

static_strings_concatenate_and_clean

static_strings.c, [23](#)

static_strings.h, [39](#)

static_strings_contains_char

static_strings.c, [24](#)

static_strings.h, [40](#)

static_strings_contains_string

static_strings.c, [24](#)

static_strings.h, [40](#)

static_strings_copy

static_strings.c, [26](#)

static_strings.h, [42](#)

static_strings_create_custom_string

static_strings.c, [26](#)

static_strings.h, [42](#)

static_strings_deallocate

static_strings.c, [27](#)

static_strings.h, [43](#)

- static_strings_double_to_string
 - static_strings.c, [27](#)
 - static_strings.h, [43](#)
- static_strings_error_code
 - Error handling, [14](#)
- static_strings_float_to_string
 - static_strings.c, [27](#)
 - static_strings.h, [43](#)
- static_strings_get_string_max_length
 - static_strings.c, [28](#)
 - static_strings.h, [44](#)
- static_strings_init
 - static_strings.c, [28](#)
 - static_strings.h, [44](#)
- static_strings_int16_to_string
 - static_strings.c, [28](#)
 - static_strings.h, [44](#)
- static_strings_int32_to_string
 - static_strings.c, [29](#)
 - static_strings.h, [45](#)
- static_strings_int8_to_string
 - static_strings.c, [29](#)
 - static_strings.h, [45](#)
- static_strings_is_line
 - static_strings.c, [29](#)
 - static_strings.h, [45](#)
- static_strings_move
 - static_strings.c, [30](#)
 - static_strings.h, [46](#)
- static_strings_save
 - static_strings.c, [30](#)
 - static_strings.h, [46](#)
- static_strings_string_descriptor, [17](#)
- static_strings_string_splitter
 - static_strings.c, [34](#)
 - static_strings.h, [50](#)
- static_strings_string_splitter_get_next_token
 - static_strings.c, [31](#)
 - static_strings.h, [47](#)
- static_strings_string_splitter_parameters, [17](#)
- static_strings_string_splitter_set_parameters
 - static_strings.c, [31](#)
 - static_strings.h, [47](#)
- static_strings_strlen
 - static_strings.c, [32](#)
 - static_strings.h, [48](#)
- static_strings_substring
 - static_strings.c, [32](#)
 - static_strings.h, [48](#)
- static_strings_uint16_to_string
 - static_strings.c, [32](#)
 - static_strings.h, [48](#)
- static_strings_uint32_to_string
 - static_strings.c, [33](#)
 - static_strings.h, [49](#)
- static_strings_uint8_to_string
 - static_strings.c, [33](#)
 - static_strings.h, [49](#)
- String descriptors, [16](#)
- String status, [13](#)
- String types, [12](#)
- String types size and quantity, [11](#)