

## Statics Strings

STM32F1XX

Generated by Doxygen 1.8.18



<b>1 Static Strings</b>	<b>1</b>
<b>2 Module Index</b>	<b>5</b>
2.1 Modules	5
<b>3 Data Structure Index</b>	<b>7</b>
3.1 Data Structures	7
<b>4 File Index</b>	<b>9</b>
4.1 File List	9
<b>5 Module Documentation</b>	<b>11</b>
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
<b>6 Data Structure Documentation</b>	<b>17</b>
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
<b>7 File Documentation</b>	<b>19</b>
7.1 static_strings.c File Reference	19
7.1.1 Detailed Description	20
7.1.2 Function Documentation	20
7.1.2.1 static_strings_allocate()	20
7.1.2.2 static_strings_compare()	21
7.1.2.3 static_strings_concatenate()	21
7.1.2.4 static_strings_contains_char()	21
7.1.2.5 static_strings_contains_string()	23
7.1.2.6 static_strings_create_custom_string()	23
7.1.2.7 static_strings_deallocate()	24

7.1.2.8 static_strings_init()	24
7.1.2.9 static_strings_is_line()	24
7.1.2.10 static_strings_save()	25
7.1.2.11 static_strings_string_splitter_get_next_token()	25
7.1.2.12 static_strings_string_splitter_set_parameters()	25
7.1.2.13 static_strings_strlen()	26
7.1.2.14 static_strings_substring()	26
7.1.3 Variable Documentation	27
7.1.3.1 static_strings_string_splitter	27
7.2 static_strings.h File Reference	27
7.2.1 Detailed Description	29
7.2.2 Function Documentation	29
7.2.2.1 static_strings_allocate()	29
7.2.2.2 static_strings_compare()	30
7.2.2.3 static_strings_concatenate()	30
7.2.2.4 static_strings_contains_char()	31
7.2.2.5 static_strings_contains_string()	31
7.2.2.6 static_strings_create_custom_string()	31
7.2.2.7 static_strings_deallocate()	32
7.2.2.8 static_strings_init()	32
7.2.2.9 static_strings_is_line()	32
7.2.2.10 static_strings_save()	33
7.2.2.11 static_strings_string_splitter_get_next_token()	33
7.2.2.12 static_strings_string_splitter_set_parameters()	34
7.2.2.13 static_strings_strlen()	34
7.2.2.14 static_strings_substring()	34
7.2.3 Variable Documentation	35
7.2.3.1 static_strings_string_splitter	35
<b>Index</b>	<b>37</b>

# Chapter 1

## Static Strings

### Author

Ramsés F. Pérez

### Date

August 2020

### Version

1.0.1

### Features:

- Developed for the STM32F103.
- Global scope strings.
- Configurable quantity and size of the memory arrays.
- No dynamic memory allocation.
- Customizable quantity and length 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, contains string, contains char and compare function.

## GETTING STARTED

### Suggested names

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

### 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.

### 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.

### 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(&uart1, token->string, token->length, HAL_MAX_DELAY);
}
```

### 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(&uart1, substring->string, substring->length, HAL_MAX_DELAY);
    static_strings_deallocate(substring);
}
```

### 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);
}

```

## 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);
}

```

## 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:

<a href="#">static_strings_string_descriptor</a>	
Meta data of a string . . . . .	17
<a href="#">static_strings_string_splitter_parameters</a>	
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:

<a href="#">static_strings.c</a>	
Strings allocation with static memory . . . . .	19
<a href="#">static_strings.h</a>	
Strings allocation with static memory . . . . .	27



## 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`

### 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 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.*
- [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. To get all the string from a start index use the length in the `finish_index`.
- `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.

## Variables

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

### 7.1.1 Detailed Description

Strings allocation with static memory.

### 7.1.2 Function Documentation

#### 7.1.2.1 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

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

#### Returns

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



### 7.1.2.2 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.

### 7.1.2.3 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. 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)
```

#### 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.

### 7.1.2.4 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.

**7.1.2.5 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.

**7.1.2.6 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`.

#### 7.1.2.7 `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.
--------------------------	---

#### 7.1.2.8 `static_strings_init()`

```
void static_strings_init ( )
```

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

```
void static_strings_init()
```

#### 7.1.2.9 `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`.

**7.1.2.10 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`.

**7.1.2.11 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.

**7.1.2.12 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.

7.1.2.13 `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`.

7.1.2.14 `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.

## 7.1.3 Variable Documentation

### 7.1.3.1 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.

## 7.2 static\_strings.h File Reference

Strings allocation with static memory.

```
#include "stm32f1xx_hal.h"
#include "string.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

## 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.*
- [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. To get all the string from a start index use the length in the `finish_index`.*
- 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.*



## 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]

### 7.2.1 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 <code>uint16_t</code> .
--------------------	---

**Returns**

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

**7.2.2.2 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**

<code>compare_string_one</code>	A pointer to the first string to compare.
<code>compare_string_two</code>	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`.

**7.2.2.3 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. 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`)

**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 <code>concatenate_at</code> string.

**Returns**

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

#### 7.2.2.4 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.

#### 7.2.2.5 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.

#### 7.2.2.6 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`.

**7.2.2.7 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.
--------------------------	---

**7.2.2.8 static\_strings\_init()**

```
void static_strings_init ( )
```

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

```
void static_strings_init()
```

**7.2.2.9 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`.

#### 7.2.2.10 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`.

#### 7.2.2.11 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.

### 7.2.2.12 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.

### 7.2.2.13 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`.

### 7.2.2.14 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.

## 7.2.3 Variable Documentation

### 7.2.3.1 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](#)

Static memory arrays, [15](#)

static\_strings.c, [19](#)

static\_strings\_allocate, [20](#)

static\_strings\_compare, [20](#)

static\_strings\_concatenate, [21](#)

static\_strings\_contains\_char, [21](#)

static\_strings\_contains\_string, [23](#)

static\_strings\_create\_custom\_string, [23](#)

static\_strings\_deallocate, [24](#)

static\_strings\_init, [24](#)

static\_strings\_is\_line, [24](#)

static\_strings\_save, [24](#)

static\_strings\_string\_splitter, [27](#)

static\_strings\_string\_splitter\_get\_next\_token, [25](#)

static\_strings\_string\_splitter\_set\_parameters, [25](#)

static\_strings\_strlen, [26](#)

static\_strings\_substring, [26](#)

static\_strings.h, [27](#)

static\_strings\_allocate, [29](#)

static\_strings\_compare, [30](#)

static\_strings\_concatenate, [30](#)

static\_strings\_contains\_char, [30](#)

static\_strings\_contains\_string, [31](#)

static\_strings\_create\_custom\_string, [31](#)

static\_strings\_deallocate, [32](#)

static\_strings\_init, [32](#)

static\_strings\_is\_line, [32](#)

static\_strings\_save, [33](#)

static\_strings\_string\_splitter, [35](#)

static\_strings\_string\_splitter\_get\_next\_token, [33](#)

static\_strings\_string\_splitter\_set\_parameters, [33](#)

static\_strings\_strlen, [34](#)

static\_strings\_substring, [34](#)

static\_strings\_allocate

static\_strings.c, [20](#)

static\_strings.h, [29](#)

static\_strings\_compare

static\_strings.c, [20](#)

static\_strings.h, [30](#)

static\_strings\_concatenate

static\_strings.c, [21](#)

static\_strings.h, [30](#)

static\_strings\_contains\_char

static\_strings.c, [21](#)

static\_strings.h, [30](#)

static\_strings\_contains\_string

static\_strings.c, [23](#)

static\_strings.h, [31](#)

static\_strings\_create\_custom\_string

static\_strings.c, [23](#)

static\_strings.h, [31](#)

static\_strings\_deallocate

static\_strings.c, [24](#)

static\_strings.h, [32](#)

static\_strings\_error\_code

Error handling, [14](#)

static\_strings\_init

static\_strings.c, [24](#)

static\_strings.h, [32](#)

static\_strings\_is\_line

static\_strings.c, [24](#)

static\_strings.h, [32](#)

static\_strings\_save

static\_strings.c, [24](#)

static\_strings.h, [33](#)

static\_strings\_string\_descriptor, [17](#)

static\_strings\_string\_splitter

static\_strings.c, [27](#)

static\_strings.h, [35](#)

static\_strings\_string\_splitter\_get\_next\_token

static\_strings.c, [25](#)

static\_strings.h, [33](#)

static\_strings\_string\_splitter\_parameters, [17](#)

static\_strings\_string\_splitter\_set\_parameters

static\_strings.c, [25](#)

static\_strings.h, [33](#)

static\_strings\_strlen

static\_strings.c, [26](#)

static\_strings.h, [34](#)

static\_strings\_substring

static\_strings.c, [26](#)

static\_strings.h, [34](#)

String descriptors, [16](#)

String status, [13](#)

String types, [12](#)

String types size and quantity, [11](#)