

Micrium

Empowering Embedded Systems

μC/LIB

V1.19

User's Manual

www.Micrium.com

Disclaimer

Specifications written in this manual are believed to be accurate, but are not guaranteed to be entirely free of error. Specifications in this manual may be changed for functional or performance improvements without notice. Please make sure your manual is the latest edition. While the information herein is assumed to be accurate, Micrium assumes no responsibility for any errors or omissions and makes no warranties. Micrium specifically disclaims any implied warranty of fitness for a particular purpose.

Copyright notice

You may not extract portions of this manual or modify the PDF file in any way without the prior written permission of Micrium. The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such a license.

© 2004-2006 Micrium, Weston, Florida 33327-1848, U.S.A.

Trademarks

Names mentioned in this manual may be trademarks of their respective companies. Brand and product names are trademarks or registered trademarks of their respective holders.

Registration

Please register the software via email. This way we can make sure you will receive updates or notifications of updates as soon as they become available. For registration please provide the following information:

- Your full name and the name of your supervisor
- Your company name
- Your job title
- Your email address and telephone number
- Company name and address
- Your company's main phone number
- Your company's web site address
- Name and version of the product

Please send this information to: licensing@micrium.com

Contact address

Micrium

949 Crestview Circle
Weston, FL 33327-1848
U.S.A.
Phone : +1 954 217 2036
FAX : +1 954 217 2037
WEB : www.micrium.com
Email : support@micrium.com

Manual versions

If you find any errors in this document, please inform us and we will make the appropriate corrections for future releases.

Manual Version	Date	By	Description
V1.18	2005/08/30	ITJ	Manual Created
V1.19	2006/04/25	JJL	Updated

Table Of Contents

	Introduction.....	6
I.1	Portable	6
I.2	Configurable	6
I.3	Coding Standards	6
I.4	MISRA C	6
I.5	Safety Critical Certification	7
I.6	µC/LIB Limitations	7
	Getting Started with µC/LIB.....	8
1.00	Installing µC/LIB	8
	µC/LIB Constant and Macro Library.....	11
2.00.01	Boolean Constants.....	11
2.00.02	Bit Constants	11
2.00.03	Octet Constants	11
2.00.04	Integer Constants	11
2.00.05	Time Constants	12
2.10	Macros	13
2.10.01.01	DEF_BIT().....	13
2.10.01.02	DEF_BIT_MASK()	14
2.10.01.03	DEF_BIT_FIELD().....	15
2.10.01.04	DEF_BIT_SET().....	16
2.10.01.05	DEF_BIT_CLR()	17
2.10.01.06	DEF_BIT_IS_SET()	18
2.10.01.07	DEF_BIT_IS_CLR().....	19
2.10.01.08	DEF_BIT_IS_SET_ANY().....	20
2.10.01.09	DEF_BIT_IS_CLR_ANY()	21
2.10.02.01	DEF_MIN().....	22
2.10.02.02	DEF_MAX()	23
2.10.02.03	DEF_ABS().....	24

	μC/LIB Memory Library	25
3.00	μC/LIB Memory Library Configuration	25
3.10.01	Mem_Clr()	26
3.10.02	Mem_Set()	27
3.10.03	Mem_Copy()	28
3.10.04	Mem_Cmp()	29
3.20	μC/LIB Memory Library Optimization	30
3.20.01	Mem_Copy() Optimization	30
	 μC/LIB String Library	 31
4.00	μC/LIB String Library Configuration	31
4.10.01	Str_Len()	32
4.10.02	Str_Copy()	33
4.10.03	Str_Cat()	34
4.10.04	Str_Cmp()	35
4.10.05	Str_Cmp_N()	36
4.10.06	Str_Char()	37
4.10.07	Str_Char_Last()	38
4.10.08	Str_Str()	39
4.10.09	Str_FmtNbr_32()	40
	 μC/LIB Licensing Policy	 42

Introduction

Designed with Micrium's renowned quality, scalability and reliability, the purpose of μ C/LIB is to provide a clean, organized ANSI C implementation of the most common standard library functions, macros, and constants.

I.1 Portable

μ C/LIB was designed for the vast variety of embedded applications. The source code for μ C/LIB is designed to be independent of and used with any processor (CPU) and compiler.

I.2 Configurable

The memory footprint of μ C/LIB can be adjusted at compile time based on the features you need and the desired level of run-time performance.

I.3 Coding Standards

Coding standards have been established early in the design of μ C/LIB and include the following:

- C coding style
- Naming convention for #define constants, macros, variables and functions
- Commenting
- Directory structure

I.4 MISRA C

The source code for μ C/LIB follows the Motor Industry Software Reliability Association (MISRA) C Coding Standards. These standards were created by MISRA to improve the reliability and predictability of C programs in critical automotive systems. Members of the MISRA consortium include Delco Electronics, Ford Motor Company, Jaguar Cars Ltd., Lotus Engineering, Lucas Electronics, Rolls-Royce, Rover Group Ltd., and other firms and universities dedicated to improving safety and reliability in automotive electronics. Full details of this standard can be obtained directly from the MISRA web site, <http://www.misra.org.uk>.

I.5 Safety Critical Certification

μ C/LIB has been designed and implemented with safety critical certification in mind. μ C/LIB is intended for use in any high-reliability, safety-critical systems including avionics RTCA DO-178B and EUROCAE ED-12B, medical FDA 510(k), and IEC 61058 transportation and nuclear systems.

For example, the FAA (Federal Aviation Administration) requires that ALL the source code for an application be available in source form and conforming to specific software standards in order to be certified for avionics systems. Since most standard library functions are provided by compiler vendors in uncertifiable binary format, μ C/LIB provides its library functions in certifiable source-code format.

If your product is NOT safety critical, you should view the software and safety-critical standards as proof that μ C/LIB is a very robust and highly-reliable software module.

I.6 μ C/LIB Limitations

By design, we have limited some of the feature of μ C/LIB. Table I-1 describes those limitations.

Does not support variable argument library functions

Table I-1, μ C/LIB limitations for current software version

Getting Started with μ C/LIB

This chapter provides information on the distribution and installation of μ C/LIB.

1.00 Installing μ C/LIB

The distribution of μ C/LIB is typically included in a ZIP file called: `uC-LIB-Vxyy.zip`. μ C/LIB could also have been included in the distribution of another Micrium ZIP file (μ C/OS-II, μ C/TCP-IP, μ C/FS). The ZIP file contains all the source code and documentation for μ C/LIB as well as all other required software modules. All modules are placed in their respective directories as shown in Figure 1-1.

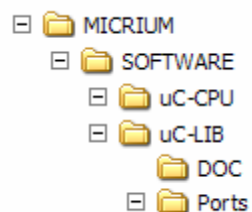


Figure 1-1, μ C/LIB Module Directories

\uC-CPU This directory contains CPU-specific code which depends on the processor and compiler used. The directory contains additional sub-directories specific for each processor/compiler combination organized as follows :

```
\MICRIUM\SOFTWARE\uC-CPU<CPU Type>\<Compiler>
```

The μ C/CPU directory contains one master CPU file :

```
\MICRIUM\SOFTWARE\uC-CPU\cpu_def.h
```

cpu_def.h

This file declares `#define` constants for CPU word sizes, endianness, critical section methods, and other processor configuration.

Each sub-directory contains source files specific for each processor/compiler combination :

```
\MICRIUM\SOFTWARE\uC-CPU\<CPU Type>\<Compiler>\cpu.h  
\MICRIUM\SOFTWARE\uC-CPU\<CPU Type>\<Compiler>\cpu_a.asm
```

cpu.h

This file contains configuration specific to the processor, such as data type definitions, processor address and data word sizes, endianness, and critical section implementation. The data type definitions are declared so as to be independent of processor and compiler word sizes.

cpu_a.asm

This file contains assembly code to enable/disable interrupts, implement critical section methods, and any other code specific to the processor.

\uC-LIB

This directory contains the μ C/LIB library source files common to many Micrium products and is shown in Figure 1-2.

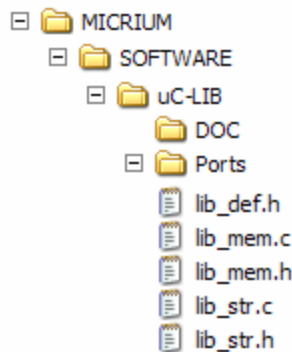


Figure 1-2, μ C/LIB Library Files

lib_def.h

This file declares #define constants for many common values such as TRUE/FALSE, YES/NO, ENABLED/DISABLED, as well as for integer, octet, and bit values. This file also contains macros for common mathematical operations like min()/max(), abs(), bit_set()/bit_clr(). See Chapter 2 for more details.

lib_mem.c and lib_mem.h

These files contain source code to replace standard library functions memclr(), memset(), memcpy(), memcmp(), etc. These functions are replaced with Mem_Clr(), Mem_Set(), Mem_Copy(), and Mem_Cmp(), respectively. See Chapter 3 for more details.

lib_str.c and lib_str.h

These files contain source code to replace standard library functions strlen(), strcpy(), strcmp(), etc. These functions are replaced with Str_Len(), Str_Copy(), and Str_Cmp(), respectively. See Chapter 4 for more details.

\Application

This directory represents the application's directory or directory tree. Application files which intend to make use of μ C/LIB constants, macros, or functions should `#include` the desired μ C/LIB header files.

app_cfg.h

This application-specific configuration file declares `#define` constants used to configure Micrium products and/or non-Micrium-related application files. This file is required by μ C/LIB to `#define` its configuration constants.

Chapter 2

µC/LIB Constant and Macro Library

µC/LIB contains many standard constants and macros. Common constants include Boolean, bit-mask, and integer values; common macros include minimum, maximum, and absolute value operations. All µC/LIB constants and macros are prefixed with `DEF_` to provide a consistent naming convention and to avoid namespace conflicts with other constants and macros in your application. These constants and macros are defined in `lib_def.h`.

2.00.01 Boolean Constants

µC/LIB contains many Boolean constants such as `DEF_TRUE/DEF_FALSE`, `DEF_YES/DEF_NO`, `DEF_ON/DEF_OFF`, and `DEF_ENABLED/DEF_DISABLED`. These constants should be used to configure, assign, and test Boolean values or variables.

2.00.02 Bit Constants

µC/LIB contains bit constants such as `DEF_BIT_00`, `DEF_BIT_07`, and `DEF_BIT_15`, which define values corresponding to specific bit positions. Currently, µC/LIB supports bit constants up to 32-bits (`DEF_BIT_31`). These constants should be used to configure, assign, and test appropriately-sized bit-field or integer values or variables.

2.00.03 Octet Constants

µC/LIB contains octet constants such as `DEF_OCTET_NBR_BITS` and `DEF_OCTET_MASK` which define octet or octet-related values. These constants should be used to configure, assign, and test appropriately-sized, octet-related integer values or variables.

2.00.04 Integer Constants

µC/LIB contains octet constants such as `DEF_INT_08_MASK`, `DEF_INT_16U_MAX_VAL`, and `DEF_INT_32S_MIN_VAL` which define integer-related values. These constants should be used to configure, assign, and test appropriately-sized, octet-related integer values or variables.

2.00.05**Time Constants**

`µC/LIB` contains time constants such as `DEF_TIME_NBR_HR_PER_DAY`, `DEF_TIME_NBR_SEC_PER_MIN`, and `DEF_TIME_NBR_mS_PER_SEC` which define time or time-related values. These constants should be used to configure, assign, and test time-related values or variables.

2.10 Macros

`μC/LIB` contains many common bit and arithmetic macros. Bit macros modify or test values based on bit masks. Arithmetic macros perform simple mathematical operations or tests.

2.10.01.01 DEF_BIT()

This macro is called to create a bit mask based on a single bit-number position.

Prototype

```
DEF_BIT(bit)
```

Arguments

`bit` This is the bit number of the bit mask to set.

Returned Value

Bit mask with the single `bit` number position set.

Notes / Warnings

None

Example

```
void AppFnct (void)
{
    CPU_INT16U mask;
    :
    :
    mask = DEF_BIT(12);
    :
    :
}
```

2.10.01.02 DEF_BIT_MASK()

This macro is called to shift a bit mask.

Prototype

```
DEF_BIT_MASK(bit_mask, bit_shift)
```

Arguments

`bit_mask` This is the bit mask to shift.

`bit_shift` This is the number of bit positions to left-shift the bit mask.

Returned Value

`bit_mask` left-shifted by `bit_shift` number of bits.

Notes / Warnings

None

Example

```
void AppFnct (void)
{
    CPU_INT16U  mask;
    CPU_INT16U  mask_hi;
    :
    :
    mask        = 0x0064;
    mask_hi     = DEF_BIT_MASK(mask, 8);
    :
    :
}
```

2.10.01.03 DEF_BIT_FIELD()

This macro is called to create a contiguous, multi-bit bit field.

Prototype

```
DEF_BIT_FIELD(bit_field, bit_shift)
```

Arguments

`bit_field` This is the number of contiguous bits to set in the bit field.

`bit_shift` This is the number of bit positions to left-shift the bit field.

Returned Value

Contiguous bit field of `bit_field` number of bits left-shifted by `bit_shift` number of bits.

Notes / Warnings

None

Example

```
void AppFnct (void)
{
    CPU_INT08U  upper_nibble;
    :
    :
    upper_nibble = DEF_BIT_FIELD(4, 4);
    :
    :
}
```

2.10.01.04 **DEF_BIT_SET()**

This macro is called to set the appropriate bits in a value according to a specified bit mask.

Prototype

```
DEF_BIT_SET(val, mask)
```

Arguments

val	This is the value to modify by setting the specified bits.
mask	This is the mask of bits to set in the value.

Returned Value

Modified value with specified bits set.

Notes / Warnings

None

Example

```
void AppFnct (void)
{
    CPU_INT16U  flags;
    CPU_INT16U  flags_alarm;
    :
    :
    flags       = 0x0000;
    flags_alarm = DEF_BIT_00 | DEF_BIT_03;
    DEF_BIT_SET(flags, flags_alarm);
    :
    :
}
```

2.10.01.05 DEF_BIT_CLR()

This macro is called to clear the appropriate bits in a value according to a specified bit mask.

Prototype

```
DEF_BIT_CLR(val, mask)
```

Arguments

val	This is the value to modify by clearing the specified bits.
mask	This is the mask of bits to clear in the value.

Returned Value

Modified value with specified bits clear.

Notes / Warnings

None

Example

```
void AppFunct (void)
{
    CPU_INT16U  flags;
    CPU_INT16U  flags_alarm;
    :
    :
    flags       = 0x0FFF;
    flags_alarm = DEF_BIT_00 | DEF_BIT_03;
    DEF_BIT_CLR(flags, flags_alarm);
    :
    :
}
```

2.10.01.06 DEF_BIT_IS_SET()

This macro is called to determine if all the specified bits in a value are set according to a specified bit mask.

Prototype

```
DEF_BIT_IS_SET(val, mask)
```

Arguments

val	This is the value to test if the specified bits are set.
mask	This is the mask of bits to check if set in the value.

Returned Value

DEF_YES, if ALL the bits in the bit mask are set in val.

DEF_NO, if ALL the bits in the bit mask are NOT set in val.

Notes / Warnings

None

Example

```
void AppFnct (void)
{
    CPU_INT16U    flags;
    CPU_INT16U    flags_mask;
    CPU_BOOLEAN   flags_set;
    :
    :
    flags         = 0x0369;
    flags_mask    = DEF_BIT_08 | DEF_BIT_09;
    flags_set     = DEF_BIT_IS_SET(flags, flag_mask);
    :
    :
}
```

2.10.01.07 DEF_BIT_IS_CLR()

This macro is called to determine if all the specified bits in a value are clear according to a specified bit mask.

Prototype

```
DEF_BIT_IS_CLR(val, mask)
```

Arguments

val This is the value to test if the specified bits are clear.

mask This is the mask of bits to check if clear in the value.

Returned Value

DEF_YES, if ALL the bits in the bit mask are clear in val.

DEF_NO, if ALL the bits in the bit mask are NOT clear in val.

Notes / Warnings

None

Example

```
void AppFnct (void)
{
    CPU_INT16U   alarms;
    CPU_INT16U   alarms_mask;
    CPU_BOOLEAN  alarms_clr;
    :
    :
    alarms       = 0x07F0;
    alarms_mask  = DEF_BIT_04 | DEF_BIT_03;
    alarms_clr   = DEF_BIT_IS_CLR(alarms, alarms_mask);
    :
    :
}
```

2.10.01.08 DEF_BIT_IS_SET_ANY()

This macro is called to determine if any of the specified bits in a value are set according to a specified bit mask.

Prototype

```
DEF_BIT_IS_SET_ANY(val, mask)
```

Arguments

val This is the value to test if any of the specified bits are set.

mask This is the mask of bits to check if set in the value.

Returned Value

DEF_YES, if ANY of the bits in the bit mask are set in val.

DEF_NO, if ALL the bits in the bit mask are NOT set in val.

Notes / Warnings

None

Example

```
void AppFnct (void)
{
    CPU_INT16U   flags;
    CPU_INT16U   flags_mask;
    CPU_BOOLEAN  flags_set;
    :
    :
    flags        = 0x0369;
    flags_mask   = DEF_BIT_08 | DEF_BIT_09;
    flags_set    = DEF_BIT_IS_SET_ANY(flags, flag_mask);
    :
    :
}
```

2.10.01.09 DEF_BIT_IS_CLR_ANY()

This macro is called to determine if any of the specified bits in a value are clear according to a specified bit mask.

Prototype

```
DEF_BIT_IS_CLR_ANY(val, mask)
```

Arguments

val This is the value to test if any of the specified bits are clear.

mask This is the mask of bits to check if clear in the value.

Returned Value

DEF_YES, if ANY of the bits in the bit mask are clear in val.

DEF_NO, if ALL the bits in the bit mask are NOT clear in val.

Notes / Warnings

None

Example

```
void AppFnct (void)
{
    CPU_INT16U   alarms;
    CPU_INT16U   alarms_mask;
    CPU_BOOLEAN  alarms_clr;
    :
    :
    alarms       = 0x07F0;
    alarms_mask  = DEF_BIT_04 | DEF_BIT_03;
    alarms_clr   = DEF_BIT_IS_CLR_ANY(alarms, alarms_mask);
    :
    :
}
```

2.10.02.01 DEF_MIN()

This macro is called to determine the minimum of two values.

Prototype

DEF_MIN(a, b)

Arguments

a	First value in minimum comparison.
b	Second value in minimum comparison.

Returned Value

The lesser of the two values, a or b.

Notes / Warnings

None

Example

```
void AppFnct (void)
{
    CPU_INT16S  x;
    CPU_INT16S  y;
    CPU_INT16S  z;
    :
    :
    x =  100;
    y = -101;
    z = DEF_MIN(x, y);
    :
    :
}
```

2.10.02.02 DEF_MAX()

This macro is called to determine the maximum of two values.

Prototype

DEF_MAX(a, b)

Arguments

a	First value in maximum comparison.
b	Second value in maximum comparison.

Returned Value

The greater of the two values, a or b.

Notes / Warnings

None

Example

```
void AppFnct (void)
{
    CPU_INT16S  x;
    CPU_INT16S  y;
    CPU_INT16S  z;
    :
    :
    x =  100;
    y = -101;
    z = DEF_MAX(x, y);
    :
    :
}
```

2.10.02.03 **DEF_ABS()**

This macro is called to determine the absolute value of a value.

Prototype

DEF_ABS (a)

Arguments

a Value to calculate absolute value.

Returned Value

The absolute value of a.

Notes / Warnings

None

Example

```
void AppFnct (void)
{
    CPU_INT16S  y;
    CPU_INT16S  z;
    :
    :
    y = -101;
    z = DEF_ABS(y);
    :
    :
}
```

μC/LIB Memory Library

μC/LIB contains library functions that replace standard library memory functions such as `memset()`, `memcpy()`, `memcmp()`, etc. These functions are defined in `lib_mem.c`.

3.00 μC/LIB Memory Library Configuration

The following μC/LIB memory library configuration may be optionally configured in `app_cfg.h`:

`uC_CFG_OPTIMIZE_ASM_EN`

Implement certain functionality in assembly-optimized files (see Section 3.20).

3.10.01 Mem_Clr()

This function is called to clear a memory buffer. In other words, set all octets in the memory buffer to a value of '0'.

Prototype

```
void  Mem_Clr (void      *pmem,  
              CPU_SIZE_T  size);
```

Arguments

pmem	This is the pointer to the memory buffer to be clear.
size	This is the number of memory buffer octets to clear.

Returned Value

None

Notes / Warnings

- 1) Zero-sized clears allowed.

Example

```
void AppFnct (void)  
{  
    CPU_CHAR  AppBuf[10];  
    :  
    :  
    Mem_Clr((void      *)&AppBuf[0],  
            (CPU_SIZE_T) sizeof(AppBuf));  
    :  
    :  
}
```

3.10.02 Mem_Set()

This function is called to fill a memory buffer with a specific value. In other words, set all octets in the memory buffer to the specific value.

Prototype

```
void Mem_Set (void      *pmem,  
              CPU_INT08U data_val,  
              CPU_SIZE_T size);
```

Arguments

pmem	This is the pointer to the memory buffer to be set with a specific value.
data_val	This is the value to set.
size	This is the number of memory buffer octets to set.

Returned Value

None

Notes / Warnings

- 1) Zero-sized sets allowed.

Example

```
void AppFnct (void)  
{  
    CPU_CHAR AppBuf[10];  
    :  
    :  
    Mem_Set((void *) &AppBuf[0],  
            (CPU_INT08U) 0x64,  
            (CPU_SIZE_T) sizeof(AppBuf));  
    :  
    :  
}
```

3.10.03 Mem_Copy()

This function is called to copy values from one memory buffer to another memory buffer.

Prototype

```
void Mem_Copy (void      *pdest,  
               void      *psrc,  
               CPU_SIZE_T size);
```

Arguments

pdest	This is the pointer to the memory buffer to copy octets into.
psrc	This is the pointer to the memory buffer to copy octets from.
size	This is the number of memory buffer octets to copy.

Returned Value

None

Notes / Warnings

- 1) Zero-sized copies allowed.
- 2) Memory buffers NOT checked for overlapping.
- 3) This function can be configured to build an assembly-optimized version (see Sections 3.00 and 3.20.01).

Example

```
void AppFnct (void)  
{  
    CPU_CHAR AppBuf[10];  
    :  
    :  
    Mem_Copy((void *) &AppBuf[0],  
             (void *) "ABCD",  
             (CPU_SIZE_T) sizeof("ABCD"));  
    :  
    :  
}
```

3.10.04 Mem_Cmp()

This function is called to compare values from two memory buffers.

Prototype

```
CPU_BOOLEAN  Mem_Copy (void          *p1_mem,
                        void          *p2_mem,
                        CPU_SIZE_T    size);
```

Arguments

p1_mem	This is the pointer to the first memory buffer to compare.
p2_mem	This is the pointer to the second memory buffer to compare.
size	This is the number of memory buffer octets to compare.

Returned Value

DEF_YES, if size number of octets are identical in both memory buffers.

DEF_NO, otherwise.

Notes / Warnings

- 1) Zero-sized compares allowed; DEF_YES returned for identical NULL compare.

Example

```
void AppFnct (void)
{
    CPU_CHAR    AppBuf[10];
    CPU_BOOLEAN  cmp;
    :
    :
    Mem_Copy((void      *)&AppBuf[0],
             (void      *) "ABCD",
             (CPU_SIZE_T) sizeof("ABCD"));
    cmp = Mem_Cmp((void      *)&AppBuf[2],
                 (void      *) "CD",
                 (CPU_SIZE_T) sizeof("CD"));
    :
    :
}
```

3.20 μ C/LIB Memory Library Optimization

All μ C/LIB memory functions have been C-optimized for improved run-time performance, independent of processor or compiler optimizations. This is accomplished by performing memory operations on CPU-aligned word boundaries whenever possible.

In addition, some μ C/LIB memory functions have been assembly-optimized for certain processors/compiler. If These optimizations are defined in assembly files found in appropriate port directories for each specific processor/compiler combination. See Figure 3-1 for an example port directory :

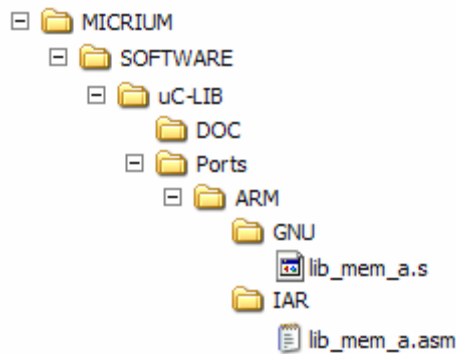


Figure 3-1, μ C/LIB Example Port Directory

3.20.01 Mem_Copy() Optimization

Future Release

µC/LIB String Library

µC/LIB contains library functions that replace standard library string functions such as `strlen()`, `strcpy()`, `strcmp()`, etc. These functions are defined in `lib_str.c`.

4.00 µC/LIB String Library Configuration

The following µC/LIB string library configuration may be optionally configured in `app_cfg.h`:

`LIB_STR_CFG_FP_EN`

Enable floating-point string conversion functions (see Section 4.10.09).

4.10.01 **Str_Len()**

This function is called to determine the length of a string.

Prototype

```
CPU_SIZE_T   Str_Len (CPU_CHAR   *pstr);
```

Arguments

pstr This is the pointer to the string.

Returned Value

Length of string in number of characters in string before but NOT including the terminating NULL character.

Notes / Warnings

- 1) String buffer NOT modified.
- 2) String length calculation terminates if string pointer points to or overlaps the NULL address.

Example

```
void AppFnct (void)
{
    CPU_INT16U   len;
    :
    :
    len = (CPU_INT16U)Str_Len("Hello World.");
    :
    :
}
```

4.10.02 Str_Copy()

This function is called to copy string character values from one string memory buffer to another memory buffer.

Prototype

```
CPU_CHAR  *Str_Copy (CPU_CHAR  *pdest,  
                    CPU_CHAR  *psrc);
```

Arguments

pdest	This is the pointer to the string memory buffer to copy string characters into.
psrc	This is the pointer to the string memory buffer to copy string characters from.

Returned Value

Pointer to copied destination string,	if NO errors.
Pointer to NULL,	otherwise.

Notes / Warnings

- 1) Destination buffer size NOT validated; buffer overruns MUST be prevented by caller.
- 2) String copy terminates if either string pointer points to or overlaps the NULL address.

Example

```
void AppFnct (void)  
{  
    CPU_CHAR  AppBuf[20];  
    CPU_CHAR  *pstr;  
    :  
    :  
    pstr = Str_Copy(&AppBuf[0], "Hello World!");  
    :  
    :  
}
```

4.10.03 **Str_Cat()**

This function is called to concatenate a string to the end of another string.

Prototype

```
CPU_CHAR    *Str_Cat (CPU_CHAR    *pdest,  
                     CPU_CHAR    *pstr_cat);
```

Arguments

pdest	This is the pointer to the string memory buffer to append string characters into.
pstr_cat	This is the pointer to the string to concatenate onto the destination string.

Returned Value

Pointer to concatenated destination string, if NO errors.

Pointer to NULL, otherwise.

Notes / Warnings

- 1) Destination buffer size NOT validated; buffer overruns MUST be prevented by caller.
- 2) String concatenation terminates if either string pointer points to or overlaps the NULL address.

Example

```
void AppFnct (void)  
{  
    CPU_CHAR    AppBuf[30];  
    CPU_CHAR    *pstr;  
    :  
    :  
    pstr = Str_Copy(&AppBuf[0], "Hello    World!");  
    pstr = Str_Cat (&AppBuf[0], "Goodbye World!");  
    :  
    :  
}
```

4.10.04 **Str_Cmp()**

This function is called to determine if two strings are identical.

Prototype

```
CPU_INT16S  Str_Cmp (CPU_CHAR  *p1_str,  
                    CPU_CHAR  *p2_str);
```

Arguments

p1_str	This is the pointer to the first string.
p2_str	This is the pointer to the second string.

Returned Value

Zero value,	if strings are identical; i.e. both strings are identical in length and ALL characters.
Positive value,	if p1_str is greater than p2_str; i.e. p1_str points to a character of higher value than p2_str for the first non-matching character found.
Negative value,	if p1_str is less than p2_str; i.e. p1_str points to a character of lesser value than p2_str for the first non-matching character found.

Notes / Warnings

- 1) String buffers NOT modified.
- 2) String comparison terminates if either string pointer points to or overlaps the NULL address.
- 3) Since 16-bit signed arithmetic is performed to calculate a non-identical comparison return value, CPU_CHAR native data type size MUST be 8-bit.

Example

```
void AppFnct (void)  
{  
    CPU_INT16S  cmp;  
    :  
    :  
    cmp = Str_Cmp("Hello World!", "Hello World.");  
    :  
    :  
}
```

4.10.05 Str_Cmp_N()

This function is called to determine if two strings are identical for a specified length of characters.

Prototype

```
CPU_INT16S  Str_Cmp_N (CPU_CHAR    *p1_str,
                      CPU_CHAR    *p2_str,
                      CPU_SIZE_T   len);
```

Arguments

p1_str	This is the pointer to the first string.
p2_str	This is the pointer to the second string.
len	This is the maximum number of string characters to compare.

Returned Value

Zero value,	if strings are identical; i.e. both strings are identical for the specified length of characters.
Positive value,	if p1_str is greater than p2_str; i.e. p1_str points to a character of higher value than p2_str for the first non-matching character found.
Negative value,	if p1_str is less than p2_str; i.e. p1_str points to a character of lesser value than p2_str for the first non-matching character found.

Notes / Warnings

- 1) String buffers NOT modified.
- 2) String comparison terminates if either string pointer points to or overlaps the NULL address.
- 3) Since 16-bit signed arithmetic is performed to calculate a non-identical comparison return value, CPU_CHAR native data type size MUST be 8-bit.

Example

```
void AppFnct (void)
{
    CPU_INT16S  cmp;
    :
    :
    cmp = Str_Cmp_N("Hello World!", "Hello World.", 10);
    :
    :
}
```

4.10.06 **Str_Char()**

This function is called to find the first occurrence of a specific character in a string.

Prototype

```
CPU_CHAR    *Str_Char (CPU_CHAR    *pstr,  
                     CPU_CHAR    srch_char);
```

Arguments

pstr	This is the pointer to the string to search for the specified character.
srch_char	This is the character to search for in the string.

Returned Value

Pointer to first occurrence of character in string,	if NO errors.
Pointer to NULL,	otherwise.

Notes / Warnings

- 1) String buffer NOT modified.
- 2) String search terminates if string pointer points to or overlaps the NULL address.

Example

```
void AppFnct (void)  
{  
    CPU_CHAR    *pstr;  
    :  
    :  
    pstr = Str_Char("Hello World!", 'l');  
    :  
    :  
}
```

4.10.07 **Str_Char_Last()**

This function is called to find the last occurrence of a specific character in a string.

Prototype

```
CPU_CHAR    *Str_Char_Last (CPU_CHAR    *pstr,  
                         CPU_CHAR    srch_char);
```

Arguments

pstr	This is the pointer to the string to search for the specified character.
srch_char	This is the character to search for in the string.

Returned Value

Pointer to first occurrence of character in string,	if NO errors.
Pointer to NULL,	otherwise.

Notes / Warnings

- 1) String buffer NOT modified.
- 2) String search terminates if string pointer points to or overlaps the NULL address.

Example

```
void AppFnct (void)  
{  
    CPU_CHAR    *pstr;  
    :  
    :  
    pstr = Str_Char_Last("Hello World!", 'l');  
    :  
    :  
}
```

4.10.08 Str_Str()

This function is called to find the first occurrence of a specific string within another string.

Prototype

```
CPU_CHAR  *Str_Str (CPU_CHAR  *pstr,
                    CPU_CHAR  *psrch_str);
```

Arguments

pstr This is the pointer to the string to search for the specified string.

psrch_str	This is the pointer to the string to search for in the string.
-----------	--

Returned Value

Pointer to first occurrence of search string in string, if NO errors.

Pointer to NULL, otherwise.

Notes / Warnings

- 1) String buffers NOT modified.
- 2) String search terminates if string pointer points to or overlaps the NULL address.

Example

```
void AppFnct (void)
{
    CPU_CHAR  *pstr;
    :
    :
    pstr = Str_Str("Hello World!", "lo");
    :
    :
}
```

4.10.09 **Str_FmtNbr_32()**

This function is called to convert & format a 32-bit number into a string.

Prototype

```
CPU_CHAR    *Str_FmtNbr_32 ( CPU_FP32        nbr ,  
                             CPU_INT08U      nbr_dig ,  
                             CPU_INT08U      nbr_dp ,  
                             CPU_BOOLEAN     lead_zeros ,  
                             CPU_BOOLEAN     nul ,  
                             CPU_CHAR        *pstr_fmt ) ;
```

Arguments

<code>nbr</code>	This is the number to format into a string.
<code>nbr_dig</code>	This is the number of integer digits to format into the number string.
<code>nbr_dp</code>	This is the number of decimal digits to format into the number string.
<code>lead_zeros</code>	Option to prepend leading zeros into the formatted number string (see Note #2).
<code>nul</code>	Option to NULL-terminate the formatted number string (see Note #3).
<code>pstr_fmt</code>	This is the pointer to the string memory buffer to return the formatted number string (see Note #4).

Returned Value

Pointer to formatted number string,	if NO errors.
Pointer to NULL,	otherwise.

Notes / Warnings

- 1) This function enabled ONLY if `LIB_STR_CFG_FP_EN` enabled in `app_cfg.h` (see Section 4.00).
- 2)
 - a) Leading zeros option prepends leading '0's prior to the first non-zero digit. The number of leading zeros is such that the total number integer digits is equal to the requested number of integer digits to format (`nbr_dig`).
 - b)
 - 1) If leading zeros option DISABLED,
 - 2) ... number of digits to format is non-zero,
 - 3) ... & the integer value of the number to format is zero;
 - 4) ... then one digit of '0' value is formatted.

This is NOT a leading zero; but a single integer digit of '0' value.

- 3)
 - a) NULL-character terminate option DISABLED prevents overwriting previous character array formatting.
 - b) **WARNING:** Unless `pstr_fmt` character array is pre-/post-terminated, NULL-character terminate option DISABLED will cause character string run-on.
- 4)
 - a) Format buffer size NOT validated; buffer overruns MUST be prevented by caller.
 - b) To prevent character buffer overrun :

Character array size MUST be \geq (nbr_dig +
 nbr_dp +
 1 negative sign +
 1 decimal point +
 1 NUL terminator) characters

Example

```
void AppFnct (void)
{
    CPU_CHAR    AppBuf[20];
    CPU_CHAR    *pstr;
    :
    :
    pstr = Str_FmtNbr_32((CPU_FP32    )-1234.5678,
                        (CPU_INT08U ) 5,
                        (CPU_INT08U ) 2,
                        (CPU_BOOLEAN) DEF_YES,
                        (CPU_BOOLEAN) DEF_YES,
                        (CPU_CHAR    *)&AppBuf[0]);
    :
    :
}
```

Appendix A

μC/LIB Licensing Policy

You need to obtain an 'Object Code Distribution License' to embed μC/LIB in a product that is sold with the intent to make a profit. Each 'different' product (i.e. your product) requires its own license but, the license allows you to distribute an unlimited number of units for the life of your product. Please indicate the processor type(s) (i.e. ARM7, ARM9, MCF5272, MicroBlaze, Nios II, PPC, etc.) that you intend to use.

For licensing details, contact us at:

Micrium

949 Crestview Circle
Weston, FL 33327-1848
U.S.A.

Phone : +1 954 217 2036
FAX : +1 954 217 2037

WEB : www.micrium.com
Email : licensing@micrium.com