



embedded-os.de
a little world of RTOS and data-communication protocols

pC/MEM Reference

V1.30a

Haftungsausschluß

Der Autor übernimmt keinerlei Haftung für durch diesen Code entstandene oder entstehende Schäden an Hard- und Software. Er versichert lediglich, daß er den Code vielfältigen Test´s auf unterschiedlicher Hardware unterzogen hat, um seinerseits keine Fehler bestehen zu wissen. Sollten dennoch Fehler auftauchen oder Vorschläge zur Verbesserung des Codes an den Autor weitergegeben werden, so ist dieser bestrebt, Fehler schnellstmöglich auszumerzen oder Vorschläge einzuarbeiten.

liability exclusion:

The author takes over no liability for through this code originated or emerging damages to hardware and software. He assures merely that he subjected the code of diverse tests on different hardware, about for his part no mistakes to know exists. Mistakes nevertheless should appear or suggestions are passed on at the author to the improvement of the code, so this is striving, mistakes fastest to wipe out or to incorporate suggestions.

The introduced Heap-manager requires on administration of $\leq 64\text{kB}$ 2(3)bytes and up to 4GB 4(5)bytes overhead per entry (depends on linked with `MEM_CleanUp`). Additionally he can administer RAM and EE-PROM parallel and gives HW-independend tools for write accesses to the EE-PROM memory over the port.

User-Functions:

Memory-Manager:	
<code>MEM_Init</code>	Initialization of the Heap-Manager
<code>MEM_Alloc</code>	Allocation of storage
<code>MEM_Free</code>	release allocated storage
<code>MEM_Resize</code>	Size of allocated storage alters
<code>Heap_Write_EE</code>	on allocated EEPROM-heap write
<code>Heap_Fill_EE</code>	allocated EEPROM-heap fill
optional	
<code>MEM_CleanUp</code>	release all allocated storage-elements of a task

Error-Codes:

Name	Decimal_Value	Description
<code>MEM_NO_ERR</code>	0	no error
<code>MEM_WR_PTR</code>	120	Pointer is not in the storage area or not allocated or no valid entry into this address
<code>MEM_OVF</code>	121	memory pool fully
<code>MEM_ERR</code>	122	error in the memory management

general

MEM_Init

U08 MEM_Init(void)

If initializes the heap-manager. If the EE-heap should be recognized as unformatted, so this is executed.

This function must be called before all other heap services at the system initialization once.

With utilization of the Linux-HOSTs, an existing heap-IMAGE is tried before the EE-formattest to load for the EE-simulation from a Linux-file.

Parameters

none

Return Value

MEM_NO_ERR	initializes successfully
MEM_ERR	Mistakes in the memory management
MEM_INVALID	EE-IMAGE invalid (only with Linux-HOST)

Example

```
void main(void)
{
    U08 returnOk;

    .
    returnOk=MEM_Init();
    .
    .
}
```

MEM_Flush

U08 MEM_Flush(void)

*Only with utilization of the Linux_HOST.
It saves the EE-heap as image into a Linux-file.*

Parameters

none

Return Value

MEM_NO_ERR	secured successfully
...	...

Example

```
void main(void)
{
    U08 returnOk;

    .
    returnOk=MEM_Init();
    .
    .
    returnOk=MEM_Flush();
}
```

MEM_Alloc

U08 OS_HUGE *MEM_Alloc(MEM_LONG size, U08 type)

Allocate the stated storage in the demanded memory type (RAM/EE) and returns the start address.

Parameters

size	size of array in bytes
type	type of Memory (0=RAM / 1=EE)

Return Value

If the returned address equally ZERO, so you get the following error-codes from MEM_GetErrNo()

MEM_OVF	Store fully
MEM_ERR	Mistakes in the memory management

Example

```
void main(void)
{
    U08  returnOk;
    U08  OS_HUGE *ptr;
    .
    returnOk=MEM_Init();
    .
    ptr=MEM_Alloc(100, MEM_RAM);
    if(ptr==NULL) {
        returnOk=MEM_GetErrno();
    }
    .
    .
}
```

MEM_Free

U08 MEM_Free(U08 OS_HUGE *ptr)

If releases the allocated storage again. The type of memory (RAM/EE) is determined on that occasion itself.

It becomes tried this freely storage area directly at a free storage behind it to the decontrol and if directly existing, at a free storage before it, to hang. (defragmentation)

Parameters

*ptr	pointer of array (from MEM_Alloc)
------	-----------------------------------

Return Value

MEM_NO_ERR	Store released
MEM_WR_PTR	Pointer is not in the storage area or not allocated or no valid entry into this address
MEM_ERR	Mistakes in the memory management

Example

```
void main(void)
{
    U08  returnOk;
    U08  OS_HUGE *ptr;
    .
    returnOk=MEM_Init();
    .
    ptr=MEM_Alloc(100, 0);
    if(ptr==NULL) {
        returnOk=MEM_GetErrno();
    } else {
        .
        .
        returnOk=MEM_Free(ptr);
    }
    .
    .
}
```

MEM_CleanUp

U08 MEM_CleanUp(U08 prio)

If releases all allocated storage from the task with the priority `prio` again. The types of memory (RAM/EE) is determined on that occasion itself.

Parameters

prio	priority of task to free
------	--------------------------

Return Value

MEM_NO_ERR	Stores released
MEM_ERR	Mistakes in the memory management
OS_PRIO_INVALID	the priority is bigger OS_MAX_TASK
OS_TASK_SUSP_PRIO	under this priority, no Task is registered

Example

```
void OS_TaskDelete(void)
{
    U08 returnOk, prio;

    OS_Lock();
    prio = OSTCBCur->OSTCBPrio;
    returnOk=MEM_CleanUp(prio);
    OS_ENTER_CRITICAL();
    OS_Unlock();
    .
    .
}
```

MEM_Resize

```
U08 OS_HUGE *MEM_Resize(U08 OS_HUGE *ptr, MEM_LONG newsize)
```

Resize the allocated storage and returns the new start address.

It will try on that occasion when increasing, a possible free storage directly behind this entry, and if this is not yet enough to use a possible free storage exactly before this entry. (defragmentation)

Only if this is not enough, a new area becomes allocated by means of MEM_Alloc and the old one released after taken place copy of data by means of MEM_Free.

Parameters

*ptr	pointer of array (from MEM_Alloc)
newsize	new size of array in bytes

Return Value

If the returned address equally ZERO, so you get the following error-codes from MEM_GetErrNo()

MEM_OVF	Store fully
MEM_WR_PTR	Pointer is not in the storage area or not allocated or no valid entry into this address
MEM_ERR	Mistakes in the memory management

Example

```
void main(void)
{
    U08 returnOk;
    U08 OS_HUGE *ptr;
    U08 OS_HUGE *ptr_new;
    .
    returnOk=MEM_Init();
    .
    ptr=MEM_Alloc(100, 0);
    if(ptr==NULL) {
        returnOk=MEM_GetErrno();
    } else {
        ptr_new=MEM_Resize(ptr, 150);
        if(ptr_new==NULL) {
            returnOk=MEM_GetErrno();
            .
            returnOk=MEM_Free(ptr);
        } else {
            .
            returnOk=MEM_Free(ptr_new);
        }
    }
    .
}
```

EE-Heap Access

Heap_Write_EE

```
U08 Heap_Write_EE(U08 OS_HUGE *source, U08 OS_HUGE *dest, MEM_LONG length)
```

Write length bytes from *source to *dest into the EE-PROM under observation of the HW-spezifika.

Parameters

*source	source pointer (not in EE-PROM)
*dest	destination pointer into EE-PROM
length	bytes to write

Return Value

MEM_NO_ERR	Store written
MEM_WR_PTR	source-pointer shows into the EE-PROM
MEM_ERR	Mistakes in the memory management

Example

```
void main(void)
{
    U08  returnOk;
    U16  var1;
    U08  OS_HUGE *ptr;
    .
    returnOk=MEM_Init();
    .
    ptr=MEM_Alloc(sizeof(U16), MEM_EE);           // alloc EE-PROM
    if(ptr==NULL) {
        returnOk=MEM_GetErrno();
    } else {
        var1 = 4125;
        returnOk=Heap_Write_EE(&var1, ptr, sizeof(U16));
    }
    .
    .
}
```

Heap_Fill_EE

```
U08 Heap_Fill_EE(U08 OS_HUGE *dest, MEM_LONG length, U08 value)
```

Fills the area into the EE-PROM with value under observation of the HW-spezifika.

Parameters

*dest	destination pointer into EE-PROM
length	bytes to fill
value	byte to fill with it

Return Value

MEM_NO_ERR	Store written
MEM_WR_PTR	dest-pointer doesn't show into the EE-PROM
MEM_ERR	Mistakes in the memory management

Example

```
void main(void)
{
    U08 returnOk;
    U08 OS_HUGE *ptr;
    .
    returnOk=MEM_Init();
    .
    ptr=MEM_Alloc(20, MEM_EE);           // alloc EE-PROM
    if(ptr==NULL) {
        returnOk=MEM_GetErrno();
    } else {
        returnOk=Heap_Fill_EE(ptr, 20, 0x55);
        .
    }
    .
}
```

Comments

Comments
