

C listing: EMMC.C

```

/*****
/*
/*-----
/* Description      : a collection of function for using EMS
/*                   memory (Expanded Memory).
/*-----
/* Author           : MICHAEL TISCHER
/* developed on      : 08/30/1988
/* last update       : 04/07/1995
/*-----
/* Memory model      : one with FAR pointer to the file, also
/*                   Compact, Large or Huge
/*-----
*****/

/*== Include files =====*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include <dos.h>

/*== Typedefs =====*/

typedef unsigned char BYTE;           /* build ourselves a byte */
typedef unsigned int WORD;
typedef BYTE BOOL;                    /*like BOOLEAN in Pascal */

/*== Macros =====*/

/*-- MK_FP creates a FAR pointer out of segment and offset addresses--*/
```

```

/*-- to on objetc                                     -----*/

#ifdef MK_FP                                           /* is MK_FP defined yet */
#undef MK_FP
#endif
#define MK_FP(seg, ofs) ((void far *) ((unsigned long) (seg)<<16|(ofs)))

/*-- PAGE_ADR returns a pointer to the physical page X within the ----*/
/*-- page frame of the EMS memory.                      ----*/

#define PAGE_ADR(x) ((void *) MK_FP(ems_frame_seg() + ((x) << 10), 0))

/*== Constants =====*/

#define TRUE 1                                         /* constants for working with BOOL */
#define FALSE 0

#define EMS_INT 0x67                                  /* Interrupt number for access to the EMM */
#define EMS_ERR -1                                    /* returned on error */

/*== global Variables =====*/

BYTE emm_ec;                                          /* the EMM error codes are placed here */

/*****
*   Function      : E M S _ I N S T
**-----**
*   Description   : Determines if EMS memory and the associated
*                   EMS driver (EMM) are installed.
*   Input Parameter : none
*   Return value   : TRUE, when EMS memory is installed, else
*                   FALSE.
*****/

```

*****/

BOOL ems_inst()

```
{
    static char emm_name[] = { 'E', 'M', 'M', 'X', 'X', 'X', 'X', '0' };
    union REGS regs;          /* Processor register for interrupt call */
    struct SREGS sregs;       /* Segment register for the interrupt call */

    /*-- construct pointer to name in the header of a switch driver----*/

    regs.x.ax = 0x3567;        /* ftn nr.: get interrupt vector 0x67 */
    intdosx(&regs, &regs, &sregs); /* call DOS-Interrupt 0x21 */
    return !memcmp( MK_FP(sregs.es, 10), emm_name, sizeof emm_name );
}
```

/*****

* Function : E M S _ N U M _ P A G E *

* Output : Determines the total number of EMS pages. *

* Input parameter : none *

* Return value : EMS_ERR on error, else the number of *

* EMS pages. *

*****/

int ems_num_page()

```
{
    union REGS regs;          /* Processor register for the interrump call */

    regs.h.ah = 0x42;         /* Fnt.nr.: get number of pages */
    int86(EMS_INT, &regs, &regs); /* Call EMM */
    if ((int) (emm_ec = regs.h.ah)) /* did an error occur? */
        return(EMS_ERR);        /* YES, display error */
}
```

```

else
    return( regs.x.dx );
}
/* no error */
/* return total number of pages */

```

```

/*****
* Function      : E M S _ F R E E _ P A G E
**-----**
* Description    : Returns the number of free EMS pages.
* Input parameter : none
* Return value   : EMS_ERR on error, else the number of free EMS
*                  pages.
*****/

```

```

int ems_free_page()
{
    union REGS regs;
    /* Processor register for the interrupt call */

    regs.h.ah = 0x42;
    int86(EMS_INT, &regs, &regs);
    if ((int) (emm_ec = regs.h.ah))
        return(EMS_ERR);
    else
        return( regs.x.bx );
}
/* Ftn.nr.: get number of pages */
/* Call EMM */
/* did an error occur? */
/* Yes, display error */
/* no error */
/* return number of free pages */

```

```

/*****
* Function      : E M S _ F R A M E _ S E G
**-----**
* Description    : Determine the Segment address of the EMS page
*                  frames.
* Input parameter : none
* Return value   : EMS_ERR on error, else the segment address of

```

```

*                               the Page frames.                               *
*****/

```

```

WORD ems_frame_seg()
{
    union REGS regs;          /* Processor register for the interrupt call */

    regs.h.ah = 0x41;          /* Fnt.nr.: get segment addr page frame */
    int86(EMS_INT, &regs, &regs);          /* Call EMM */
    if ((int) (emm_ec = regs.h.ah))          /*did an error occur? */
        return(EMS_ERR);          /* Yes, display error */
    else          /* no error */
        return( regs.x.bx );          /* return segment address */
}

```

```

/*****
*   Function           : E M S _ A L L O C                               *
**-----**
*   Description        : Allocates the specified number of pages and      *
*                       returns a handle for accessing these pages.        *
*   Input paramater    : PAGES : the number of pages to be allocated      *
*                       (each 16 Kbyte)                                     *
*   Return value       : EMS_ERR on error, else the EMS handle            *
*****/

```

```

int ems_alloc(int pages)
{
    union REGS regs;          /* Processor register for the interrupt call */

    regs.h.ah = 0x43;          /* Fnt.nr.: Pages allocated */
    regs.x.bx = pages;          /* set number of pages to be allocated */
    int86(EMS_INT, &regs, &regs);          /* Call EMM */
}

```

```

if ((int) (emm_ec = regs.h.ah))          /* did an error occur? */
    return(EMS_ERR);                     /* Yes, display error */
else                                     /* no error */
    return( regs.x.dx );                 /* return EMS handle */
}

```

```

/*****
* Function      : E M S _ M A P
**-----**
* Description   : Maps one of the allocated pages specified
*                by the given handle onto a physical page in the
*                page frame.
* Input parameter : HANDLE: the handle returned by EMS_ALLOC
*                LOGP  : the logical page (0 to n-1)
*                PHYSP : the physical page (0 to 3)
* Return value  : FALSE on error , else TRUE.
*****/

```

```

BOOL ems_map(int handle, int logp, BYTE physp)

```

```

{
    union REGS regs;          /* Processor register for the interrupt call */

    regs.h.ah = 0x44;         /* Fnt.nr.: set mapping */
    regs.h.al = physp;        /* set physical page */
    regs.x.bx = logp;         /* set logical page */
    regs.x.dx = handle;       /* set EMS handle */
    int86(EMS_INT, &regs, &regs); /* call EMM */
    return (!(emm_ec = regs.h.ah));
}

```

```

/*****
* Function      : E M S _ F R E E
*

```

```

**-----**
* Description      : Releases the memory specified by the handle      *
* Input parameter  : HANDLE: the handle returned by EMS_ALLOC.       *
* Return value     : FALSE on error, else TRUE.                       *
*****/

```

```

BOOL ems_free(int handle)

```

```

{
    union REGS regs;          /* Processor register for the interrupt call */

    regs.h.ah = 0x45;          /* Fnt.nr.: release pages */
    regs.x.dx = handle;        /* set EM handle */
    int86(EMS_INT, &regs, &regs); /* call EMM */
    return (!(emm_ec = regs.h.ah)); /* if AH contains 0, all is o.k. */
}

```

```

/*****

```

```

* Function          : E M S _ V E R S I O N                          *
**-----**

```

```

* Description       : Determines the EMM version number              *
* Input parameter   : none                                           *
* Return value      : EMS_ERR on error, else the EMM version number. *
* Info             : In th eversion number, 10 stands for 1.0, 11 for *
*                   1.1, 34 for 3.4 etc.                             *
*****/

```

```

BYTE ems_version()

```

```

{
    union REGS regs;          /* Processor register for the interrupt call */

    regs.h.ah = 0x46;          /* Fnt.nr.: get EMM version number*/
    int86(EMS_INT, &regs, &regs); /* call EMM */
}

```

```

if ((int) (emm_ec = regs.h.ah))          /* did an error occur? */
    return(EMS_ERR);                     /* Yes, display error */
else                                     /* no error, calculate version number from BCD number */
    return( (regs.h.al & 15) + (regs.h.al >> 4) * 10);
}

```

```

/*****
* Function      : E M S _ S A V E _ M A P
**-----**
* Description   : Saves the mapping between the logical and
*                physical pages.
* Input parameter : HANDLE: the handle returned by EMS_ALLOC.
* Return value  : FALSE on error, else TRUE.
*****/

```

```

BOOL ems_save_map(int handle)

```

```

{
    union REGS regs;          /* Processor register for the interrupt call */

    regs.h.ah = 0x47;         /* Fnt.nr.: save mapping */
    regs.x.dx = handle;       /* set EMS handle */
    int86(EMS_INT, &regs, &regs); /* call EMM */
    return (!(emm_ec = regs.h.ah)); /* if AH contains 0 everything is OK */
}

```

```

/*****
* Function      : E M S _ R E S T O R E _ M A P
**-----**
* Description   : Restores a mapping between logical and physical
*                pages saved with EMS_SAVE_MAP.
* Input parameters : HANDLE: the handle returned by EMS_ALLOC.
* Return value    : FALSE on error, else TRUE.
*****/

```



```
*****/
```

```
BOOL ems_restore_map(int handle)
```

```
{
    union REGS regs;          /* Processor register for the interrupt call */

    regs.h.ah = 0x48;          /* Fnt.nr.: restore mapping */
    regs.x.dx = handle;        /* set EMS handle */
    int86(EMS_INT, &regs, &regs); /* call EMM */
    return (!(emm_ec = regs.h.ah)); /* if AH contains 0, all is OK. */
}
```

```
/*****
```

```
* Function : P R I N T _ E R R *
```

```
**-----**
```

```
* Description : Prints and EMS error message on the screen and *
* ends the program. *
* Input parameter : none *
```

```
* Return value : none *
```

```
* Info : This function may only be called if an error *
```

```
* occurred on a prior call to the EMM. *
```

```
*****/
```

```
void print_err()
```

```
{
    static char nid[] = "unidentifiable";
    static char *err_vec[] =
    { "Error in the EMS driver (EMM destroyed)", /* 0x80 */
      "Error in the EMS hardware", /* 0x81 */
      nid, /* 0x82 */
      "Illegal EMM handle", /* 0x83 */
      "EMS function called does not exist", /* 0x84 */
    }
```

```

        "No more EMS handle available",                /* 0x85 */
        "Error while saving or restoring the mapping",  /* 0x86 */
        "More pages requested than physically present", /* 0x87 */
        "More pages requested than are still free",     /* 0x88 */
        "Zero page requested",                          /* 0x89 */
        "Logicalpage does not belong to handle",        /* 0x8A */
        "Illegal physical page number",                 /* 0x8B */
        "Mapping storag eis full",                      /* 0x8C */
        "The mapping has already been saved",           /* 0x8D */
        "Restored mapping without saving first"
    };

    printf("\nERROR! Error in access to EMS memory\n");
    printf("    ... %s\n", (emm_ec<0x80 || emm_ec>0x8E) ?
                                nid : err_vec[emm_ec-0x80]);
    exit( 1 ); /* End program with error code */
}

/*****
*   Function      : V R _ A D R
**-----**
*   Description   : Returns a pointer to video RAM.
*   Input parameter : none
*   Return value  : VOID pointer to the video RAM.
*****/

void *vr_adr()
{
    union REGS regs; /* Processor register for the interrupt call */

    regs.h.ah = 0x0f; /* Fnt.nr.: get video mode */
    int86(0x10, &regs, &regs); /* call BIOS Video Interrupt */
}

```

```

    return ( MK_FP((regs.h.al==7) ? 0xb000 : 0xb800, 0) );
}

/*****
**                                MAIN PROGRAM                                **
*****/

void main()
{
    int  numpage,                /* number of EMS pages */
        handle,                /* handle to access the EMS memory */
        i;                     /* loop counter */
    WORD pageseg ;              /* segment address of the page frame */
    BYTE emmver;                /* EMM version number */

    printf("EMMC - (c) 1988, 92 by MICHAEL TISCHER\n\n");
    if ( ems_inst() )            /* is EMS memory installed? */
    {                             /* Yes */
        /*-- output information about the EMS memory -----*/

        if ( (int) (emmver = ems_version()) == EMS_ERR) /* get version num */
            print_err();        /* error: output error message and end program */
        else                    /* no error */
            printf("EMM-Version number          : %d.%d\n",
                   emmver/10, emmver%10);

        if ( (numpage = ems_num_page()) == EMS_ERR) /* get number pages */
            print_err();        /* Error: output error message and end program */
        printf("Number of EMS pages           : %d (%d KByte)\n",
               numpage, numpage << 4);

        if ( (numpage = ems_free_page()) == EMS_ERR)

```

```

    print_err();          /* Error: output error message and end program */
    printf("...         free          : %d (%d KByte)\n",
           numpage, numpage << 4);

if ( (int) (pageseg = ems_frame_seg()) == EMS_ERR)
    print_err();          /* Error: output error message and end program */
printf("Segment address of the page frame: %X\n", pageseg);

printf("\nNow a page will be allocated from the EMS memory and\n");
printf("the screen contents will be copied from video RAM\n");
printf("to this page.\n");
printf("                ... press any key\n");
getch();                  /* wait for a key */

/*-- allocate a page and map it to the first logical page in      ---*/
/*-- page frame.                                                  ---*/

if ( (handle = ems_alloc(1)) == EMS_ERR)
    print_err();          /* Error: output error message and end program */
if ( !ems_map(handle, 0, 0) )                                     /* set mapping */
    print_err();          /* Error: output error message and end program */

/*-- copy 4000 bytes from the video RAM to the EMS memory -----*/

memcpy(PAGE_ADR(0), vr_adr(), 4000);

for (i=0; i<24; ++i)                                           /* clear the screen */
    printf("\n");

printf("The old screen contents will now be cleared and will\n");
printf("be lost. But since it was stored in the EMS memory, they\n");
printf("can be copied from there back into the video RAM\n");

```

```

printf("                                ...press any key\n");
getch();                                /* wait for a key */

/*-- copy contents of the video RAM from the EMS memory -----*/
/*-- and release the allocated EMS memory again.                ----*/

memcpy(vr_adr(), PAGE_ADR(0), 4000);    /* copy V-RAM back */
if ( !ems_free(handle) )                /* free memory */
    print_err();                        /* Error: output error message and end program */
printf("END");
}
else                                    /* the EMS driver was not detected */
    printf("No EMS memory installed\n");
}

```