

Pascal listing: EMMP.PAS

```
{*****}
{*               E M M P               *}
{*-----*}
{*  Description   : Implement certain function to demonstrate *}
{*                access to EMS meory using EMMM.             *}
{*-----*}
{*  Author        : MICHAEL TISCHER                             *}
{*  Developed on   : 08/30/1988                                   *}
{*  Last update on : 04/07/1995                                   *}
{*-----*}
{*  Changes       : 02/19/92 : MK_FP replaced through PTR      *}
{*****}
```

program EMMP;

```
Uses Dos, CRT;                                { Add DOS and CRT units }

type  ByteBuf = array[0..1000] of byte;    { One memory range as bytes }
      CharBuf = array[0..1000] of char;    { one memory range as Char }
      BytePtr = ^ByteBuf;                  { Pointer to byte range }
      CharPtr = ^CharBuf;                  { Pointer to character range }

const EMS_INT   = $67;                      { Interrupt # for access to EMM }
      EMS_ERR   = -1;                      { Error if this occurs }
      W_EMS_ERR = $FFFF;                   { error code in WORD form }
      EmmName   : array[0..7] of char = 'EMMXXX0'; { Name of EMM }

var   EmmEC,                                { Allocation of EMM erro codes. }
      i      : byte;                        { loop counter }
      Handle, { handle for acces to EMS memory }
      EmmVer  : integer;                    { Version number of EMM }
```

```

    NumPage,                                { Number of the EMS pages }
    PageSeg  : word;                        { Segment address of page frame }
    KeyPress   : char;

{*****}
{ * EmsInst : Determine the existence of EMS and correspondign EMM * }
{ * Input   : none * }
{ * Output  : TRUE,when EMS memory is available, else FALSE * }
{*****}

function EmsInst : boolean;

type EmmName = array [1..8] of char; { Name the EMM in driver header }
    EmmNaPtr = ^EmmName;             { Pointer to name in driver header }

const Name : EmmName = 'EMMXXX0';    { Name of EMS driver }

var Regs : Registers;                { Processor register for interrupt call }

begin
    Regs.ax := $35 shl 8 + EMS_INT;   { over interrupt vector $67 }
    msdos( Regs );                     { Get DOS-Function 35h }

    EmsInst := (EmmNaPtr(Ptr(Regs.ES,10))^ = Name); { Driver name compare }
end;

{*****}
{ * EmsNumPage: Determine the total of EMS pages. * }
{ * Input     : none * }
{ * Output    : EMS_ERR if error occurs, otherwise number of EMS pages. * }
{*****}

```

```

function EmsNumPage : integer;

var Regs : Registers;      { Processor register for the interrupt call }

begin
  Regs.ah := $42;           { Fnt.nr.: Determine number of pages }
  Intr(EMS_INT, Regs);      { call EMM }
  if (Regs.ah <> 0 ) then    { did an error occur? }
    begin                  { Yes }
      EmmEC := Regs.ah;     { get error code }
      EmsNumPage := EMS_ERR; { display error }
    end
  else                      { no error }
    EmsNumPage := Regs.dx;   { Return total number of pages }
end;

{ ***** }
{ * EmsFreePage: Determines the number of free EMS pages. * }
{ * Input      : none * }
{ * Output     : EMS_ERR if error occurs, otherwise the number of un- * }
{ *             used EMS pages. * }
{ ***** }

function EmsFreePage : integer;

var Regs : Registers;      { Processor register for the interrupt call }

begin
  Regs.ah := $42;           { Fnt.nr.: Determine no. of pages }
  Intr(EMS_INT, Regs);      { call EMM }
  if (Regs.ah <> 0 ) then    { did an error occur? }
    begin                  { Yes }

```

```

        EmmEC := Regs.ah;                                { get error code }
        EmsFreePage := EMS_ERR;                          { display error }
    end
    else                                                  { no error }
        EmsFreePage := Regs.bx;                          { Return number of free pages }
end;

```

```

{*****}
{ * EmsFrameSeg: Determines the segment address of the page frame. * }
{ * Input   : none                                           4 * }
{ * Output  : EMS_ERR if error occurs, else the segment address. * }
{*****}

```

```
function EmsFrameSeg : word;
```

```
var Regs : Registers;      { Processor register for the interrupt call }
```

```
begin
```

```

    Regs.ah := $41;                                { Fnt.nr.: get Segment adr. page frame }
    Intr(EMS_INT, Regs);                            { call EMM }
    if (Regs.ah <> 0 ) then                          { did an error occur? }
    begin                                           { Yes }
        EmmEC := Regs.ah;                          { get error code }
        EmsFrameSeg := W_EMS_ERR;                  { display error }
    end
    else                                           { no error }
        EmsFrameSeg := Regs.bx;      { return segment address of page frame }
end;

```

```

{*****}
{ * EmsAlloc: Allocate the specified number of pages and returns a * }
{ *           handle for access to these pages.                   * }

```

```

{ * Input   : PAGES: the number of allocated pages.          * }
{ * Output  : EMS_ERR returns the error, else the handle.     * }
{ ***** }

```

```
function EmsAlloc( Pages : integer ) : integer;
```

```
var Regs : Registers;          { Processor register for the interrupt call }
```

```
begin
```

```

    Regs.ah := $43;                { Fnt.nr.: Pages allocated }
    Regs.bx := Pages;              { set number of allocate pages }
    Intr(EMS_INT, Regs);           { call EMM }
    if (Regs.ah <> 0) then          { did an error occur? }
    begin                          { Yes }
        EmmeC := Regs.ah;          { get error code }
        EmsAlloc := EMS_ERR;      { display error }
    end

```

```

    else                          { no error }
        EmsAlloc := Regs.dx;      { Handle returned }
end;
```

```

{ ***** }
{ * EmsMap   : Creates an allocated logical page from a physical page * }
{ *           in the page frame.                                     * }
{ * Input    : HANDLE: Handle recieved from EmsAlloc.              * }
{ *           LOGP  : Logical page about to be created              * }
{ *           PHYSP : The physical page in the page frame.          * }
{ * Output   : FALSE if error ,else TRUE.                           * }
{ ***** }

```

```
function EmsMap(Handle, LogP : integer; PhysP : byte) : boolean;
```

```

var Regs : Registers;          { Processor register for the interrupt call }

begin
  Regs.ah := $44;               { Fnt.nr.: set mapping }
  Regs.al := PhysP;             { set physical page }
  Regs.bx := LogP;              { set logical page }
  Regs.dx := Handle;            { set EMS handle }
  Intr(EMS_INT, Regs);          { call EMM }
  EmmEC := Regs.ah;             { get error code }
  EmsMap := (Regs.ah = 0)       { TRUE returned, when no error }
end;

{*****}
{ * EmsFree : Frees memory when given with an allocated handle. * }
{ * Input   : HANDLE: Handle received by EmsAlloc. * }
{ * Output  : FALSE on error, else TRUE. * }
{*****}

function EmsFree(Handle : integer) : boolean;

var Regs : Registers;          { Processor register for the interrupt call }

begin
  Regs.ah := $45;               { Fnt.nr.: Release pages }
  Regs.dx := handle;            { set EMS handle }
  Intr(EMS_INT, Regs);          { call EMM }
  EmmEC := Regs.ah;             { get error code }
  EmsFree := (Regs.ah = 0)      { TRUE returned, when no error }
end;

{*****}
{ * EmsVersion: Determines the version number of EMM. * }

```

```

{ * Input      : none                                     * }
{ * Output     : EMS_ERR on error, otherwise the version number. 6 * }
{ *           ( 11 for 1.1, 40 for 4.0 etc.)                * }
{ ***** }

```

```
function EmsVersion : integer;
```

```
var Regs : Registers;      { Processor register for the interrupt call }
```

```
begin
```

```

  Regs.ah := $46;                { Fnt.nr.: Determine EMM version }
  Intr(EMS_INT, Regs);           { call EMM }
  if (Regs.ah <> 0) then          { did an error occur? }
    begin                        { Yes }
      EmmEC := Regs.ah;          { get error code }
      EmsVersion := EMS_ERR;     { display error }
    end
  else                           { no error, computer version number from BCD number }
    EmsVersion := (Regs.al and 15) + (Regs.al shr 4) * 10;
  end;

```

```
end;
```

```

{ ***** }
{ * EmsSaveMap: Saves display between logical and physical pages of the* }
{ *           given handle.                                           * }
{ * Input      : HANDLE: Handle assigned by EmsAlloc.                * }
{ * Output     : FALSE on error, else TRUE.                          * }
{ ***** }

```

```
function EmsSaveMap( Handle : integer ) : boolean;
```

```
var Regs : Registers;      { Processor register for the interrupt call }
```

```

begin
  Regs.ah := $47;                                { Fnt.nr.: Save mapping }
  Regs.dx := handle;                              { set EMS handle }
  Intr(EMS_INT, Regs);                             { call EMM }
  EmmEC := Regs.ah;                               { get erro code }
  EmsSaveMap := (Regs.ah = 0)                      { TRUE returned, when no error }
end;

```

```

{*****}
{ * EmsRestoreMap: Returns display between logical and physical pages, * }
{ * from the page saved by EmsSaveMap * }
{ * Input : HANDLE: Handle assigned by EmsAlloc * }
{ * Output : FALSE if an error occurs, otherwise TRUE * }
{*****}

```

```

function EmsRestoreMap( Handle : integer ) : boolean;

```

```

var Regs : Registers;      { Processor register for the interrupt call }

```

```

begin
  Regs.ah := $48;                                { Fnt.nr.: Restore mapping }
  Regs.dx := handle;                              { set EMS handle }
  Intr(EMS_INT, Regs);                             { call EMM }
  EmmEC := Regs.ah;                               { mark erro code }
  EmsRestoreMap := (Regs.ah = 0)                  { TRUE returned when no error }
end;

```

```

{*****}
{ * PrintErr: Displays an error message and ends the program * }
{ * Input : none * }
{ * Output : none * }
{ * Info : This function is called only if an error occurs during a * }

```



```
{*                function call within this module                *}
{*****}
```

```
procedure PrintErr;
```

```
begin
```

```
  writeln('ATTENTION! Error during EMS memory access');
```

```
  write('      ... ');
```

```
  if ((EmmEC<$80) or (EmmEC>$8E) or (EmmEC=$82)) then
```

```
    writeln('Unidentifiable error')
```

```
  else
```

```
    case EmmEC of
```

```
      $80 : writeln('EMS driver error (EMM trouble)');
```

```
      $81 : writeln('EMS hardware error');
```

```
      $83 : writeln('Illegal EMM handle');
```

```
      $84 : writeln('Called EMS function does not exist');
```

```
      $85 : writeln('No more free EMS handles available');
```

```
      $86 : writeln('Error while saving or restoring mapping ');
```

```
      $87 : writeln('More pages requested than are actually ',
                    'available');
```

```
      $88 : writeln('More pages requested than are free');
```

```
      $89 : writeln('No pages requested');
```

```
      $8A : writeln('Logical page does not belong to handle');
```

```
      $8B : writeln('Illegal physical page number');
```

```
      $8C : writeln('Mapping memory range is full');
```

```
      $8D : writeln('Map save has already been done');
```

```
      $8E : writeln('Mapping must be saved before it can',
                    'be restored');
```

```
    end;
```

```
  Halt;
```

```
  { Program end }
```

```
end;
```

```

{*****}
{* VrAdr: Returns a pointer to video RAM                                     *}
{* Input   : none                                                         *}
{* Output  : Pointer to video RAM                                         *}
{*****}

```

```
function VrAdr : BytePtr;
```

```
var Regs : Registers;      { Processor register for the interrupt call }
```

```
begin
```

```

    Regs.ah := $0f;                { Fnt.nr.: Determine video mode }
    Intr($10, Regs);               { call BIOS-Video-Interrupt }
    if (Regs.al = 7) then           { monochrome video card? }
        VrAdr := ptr($B000, 0)     { Yes, Video-RAM at B000:0000 }
    else                            { Color-, EGA- or VGA-card }
        VrAdr := ptr($B800, 0);    { Video-RAM at B800:0000 }

```

```
end;
```

```

{*****}
{* PageAdr : Returns address of a physical page in page frame           *}
{* Input   : PAGE: Physical page number (0-3)                           *}
{* Output  : Pointer to the physical page                                *}
{*****}

```

```
function PageAdr( Page : integer ) : BytePtr;
```

```
begin
```

```
    PageAdr := Ptr( EmsFrameSeg + (Page shl 10), 0 );
```

```
end;
```

```
{*****}
```

```

{**                                     MAIN PRORAM                                     **}
{*****}

```

```

begin
  ClrScr;                                     { clear screen }
  writeln('EMMP - (c) 1988, 92 by MICHAEL TISCHER', #13#10);
  if EmsInst then                             { is EMS memory installed? }
  begin                                       { Yes }
    {*--Display informationen about the EMS memory -----*}

    EmmVer := EmsVersion;                    { Determine EMM version number }
    if EmmVer = EMS_ERR then                 { did an error occur? }
    PrintErr;                               { Yes, display error message and end program }
    writeln('EMM-Version number              : ', EmmVer div 10, '.',
      EmmVer mod 10);

    NumPage := EmsNumPage;                   { Determine total no. pages }
    if NumPage = EMS_ERR then               { did an error occur? }
    PrintErr;                               { Yes, display erro rmessage and end program }
    writeln('Number of EMS pages             : ', NumPage, ' (',
      NumPage shl 4, ' KByte)');

    NumPage := EmsFreePage;                 { Determine number of free pages }
    if NumPage = EMS_ERR then               { did an error occur? }
    PrintErr;                               { Yes, display erro message and end program }
    writeln('... free                       : ', NumPage, ' (',
      NumPage shl 4, ' KByte)');

    PageSeg := EmsFrameSeg;                 { Segment addresse of page frame }
    if PageSeg = W_EMS_ERR then            { did an error occur? }
    PrintErr;                               { Yes, displat erro rmessage and end program }
    writeln('Segment address of page frame: ', PageSeg);

```

```

writeln;
writeln('Now a page from EMS memory can be allocated, and the');
writeln('screen contents can be copied from video RAM into this');
writeln('page.');
```

writeln('	... Please press a key');
KeyPress := ReadKey;	{ Wait for a keypress }

```

{ *-- Page is allocated, and the data is passed to the first-----* }
{ *-- logical page in the page frame                                -----* }
```

Handle := EmsAlloc(1);	{ Allocate one page }
if Handle = EMS_ERR then	{ did an error occur? }
PrintErr;	{ Yes, display error message and end program }
if not(EmsMap(Handle, 0, 0)) then	{ Set mapping }
PrintErr;	{ Error: display error and end program }

```

{ *-- Copy 4000 Bytes from Video-RAM to EMS memory----- * }
```

```

Move(VrAdr^, PageAdr(0)^, 4000);
```

ClrScr;	{ Clear screen }
while KeyPressed do	{ Read keyboard buffer }
KeyPress := ReadKey;	
writeln('Old screen contents are cleared. However, the data ');	
writeln('from the screen is in EMS, and can be re-copied onto ');	
writeln('the screen.');	
writeln('	... Please press a key');
KeyPress := ReadKey;	{ Wait for a keypress }

```

{ *-- Copy contents of video RAM from EMS memory and release  --* }
{ *-- the allocated EMS memory                                --* }
```

```

    Move(PageAdr(0)^, VrAdr^, 4000);           { Copy over Video RAM }
    if not(EmsFree(Handle)) then               { Release memory }
        PrintErr;      { Error: display message and end program }
        GotoXY(1, 15);
        writeln('END')
    end
else                                           { the EMD driver not available }
    writeln('No EMS memory installed.');
```

end. .i).Expanded memory;