

## Listing 1: MemOvrly.Inc

```
(*****  
*                                     *  
*   Turbo Pascal Memory Overlay Routines   *  
*                                     *  
*   Copyright (C) 1986 by Steve McMahon   *  
*                                     *  
*   All Rights Reserved.                   *  
*                                     *  
*****)
```

(\*

### Limitations:

These routines have been tested only for Turbo 3.01A (both PC-DOS and generic MS-DOS). They may not work under 3.0 (the celebrated FileSize bug may cause trouble) and will certainly not work under 2.0XX.

Memory overlay files must be < 64k in size!

NORMAL overlays nested inside memory overlays should work, but trying to nest memory overlays inside memory overlays would be disastrous!

OvrPath will not work in conjunction with memory overlays! (Writing a replacement routine would be simple if the code below makes sense to you.)

I/O testing in InitOverlay is just Turbo's Native. Anyone really needing memory overlays will probably wish to install their own I/O error checking.

\*)

### CONST

```
RequiredHeap = $1000;    {Paragraphs of Heap Required by Program  
                        for other purposes than memory overlays.  
                        Change this to suit your needs for dynamic  
                        storage.}
```

### TYPE

```
{Type used in both InitOverlay and DisposeOverlayStorage}  
OverlayProcedure = RECORD  
    CASE Boolean OF  
        True :  
            ( OldCall   : ARRAY[1..3] OF Byte;  
              OldOffset : Integer;  
              FileName  : ARRAY[1..13] OF Char;  
              );  
        False :  
            ( NewCallInstruction : ARRAY[1..3] OF Byte;  
              NewCallAddress    : Integer;
```

```

CurrentOffset : Integer;
OverlayCodeLoc : ^Byte;
NewRoutineLoc : Integer;
OverlaySize : Integer;
)
END;

```

```
PROCEDURE NewOverlayHandler;
```

```
BEGIN
```

```
  INLINE(
```

```
    {When this routine receives control, AX contains the
    number of bytes in the desired overlay & BX contains the
    offset (in pages) of the desired overlay within the
    overlay file (now on the heap).}
```

```
    {First, check to see if the desired overlay is already in
    place by comparing DX with the offset recorded in memory
    immediately after the call instruction. If they match,
    no load is necessary}
```

```

$5E/          {POP   SI      }
$2E/$3B/$14/  {CMP   DX,CS:[SI]}
$74/$1B/      {JZ    RUN_OVERLAY}

```

```
{Save vital registers}
```

```

$56/          {PUSH  SI      }
$1E/          {PUSH  DS      }

```

```
{Load ES:DI with destination address (the point the
code will run at). Displace to account for header.}
```

```

$0E/          {PUSH  CS      }
$07/          {POP   ES      }
$8B/$FE/      {MOV   DI,SI   }
$83/$C7/$0D/  {ADD   DI,0DH   }

```

```
{Fetch heap address of source overlay code from memory
position two bytes after first byte after call to this
routine. Store it in DS:SI}
```

```

$46/          {INC   SI      }
$46/          {INC   SI      }
$2E/$C5/$34/  {LDS   SI,CS:[SI]}

```

```
{Multiply overlay page by 100H to get number of bytes code
is displaced from start of overlay code area (on heap).
```

```
Add to source offset in SI.}
```

```

$8A/$F2/      {MOV   DH,DL   }
$32/$D2/      {XOR   DL,DL   }
$03/$F2/      {ADD   SI,DX   }

```

```
{Put number of bytes to move in CX}
```

```
$8B/$C8/      {MOV   CX,AX   }
```

```
{Copy CX bytes from DS:SI to ES:DI}
```

```

$FC/          {CLD           }
$F3/$A4/      {REPZ  MOVSB   }

```

```

{Recover mauled registers}
$1F/      {POP  DS      }
$5E/      {POP  SI      }

{RUN_OVERLAY:}
$83/$C6/$0D/  {ADD  SI,0DH  }
$FF/$E6      {JMP  SI      }
);
END;

```

```

PROCEDURE InitOverlay(OverlayCallOffset : Integer);
VAR
  OverlayCallPtr : ^OverlayProcedure;
  TestSize, i    : Integer;
  s              : STRING[13];
  f              : FILE;
BEGIN
  OverlayCallPtr := Ptr(CSeg, OverlayCallOffset);
  WITH OverlayCallPtr^ DO
  BEGIN
    {Obtain overlay file name}
    i := 1;
    s := "";
    WHILE FileName[i] <> #0 DO
      BEGIN
        s := s + FileName[i];
        i := i + 1;
      END;
    {Open overlay file as untyped file}
    Assign(f, s);
    Reset(f);
    {determine file size in $80-byte sectors}
    TestSize := FileSize(f);
    {Check to see if there's enough space on the heap.}
    {If there isn't, leave the overlay on disk}
    IF (MemAvail > (RequiredHeap + TestSize * 8)) AND
       (MaxAvail >= TestSize * 8) THEN {there's enough space}
      BEGIN      {install overlay}
        OverlaySize := TestSize;
        GetMem(OverlayCodeLoc, OverlaySize * $80);
        BlockRead(f, OverlayCodeLoc^, OverlaySize, i);
        NewCallInstruction[1] := $2E; {CS:}
        NewCallInstruction[2] := $FF;
        NewCallInstruction[3] := $16; {indirect near call}
        NewCallAddress := Ofs(NewRoutineLoc);
        NewRoutineLoc := Ofs(NewOverlayHandler) + 7;
        {extra 7 bytes skips turbo's procedure overhead}
        CurrentOffset := $FFFF; {force load on first call}
      END;
    Close(f);
  END;
END;
END;

```

```

PROCEDURE DisposeOverlayStorage(OverlayCallOffset : Integer);
VAR
  OverlayCallPtr : ^OverlayProcedure;
BEGIN
  OverlayCallPtr := Ptr(CSeg, OverlayCallOffset);
  WITH OverlayCallPtr^ DO
    IF NewCallInstruction[3] = $16 THEN {Overlay is in memory}
      FreeMem(OverlayCodeLoc, OverlaySize * $80);
END;

```

```

PROGRAM OverlayTest;

```

```

(* Memory Overlay Demonstration Program. *)

```

```

{$I MEMOVRLY.INC}

```

```

VAR
  c : Char;

```

```

OVERLAY PROCEDURE One;
BEGIN
  WriteLn('This is Overlay Procedure One. ');
END;

```

```

OVERLAY PROCEDURE Two;
BEGIN
  WriteLn('This is Overlay Procedure Two. ');
END;

```

```

BEGIN

```

```

  {Install the new overlay handler by passing it the address
  offset of ONE procedure or function from the overlay group.
  Multiple invocations for multiple overlay groups should be
  no problem.}

```

```

  InitOverlay(Ofs(One));

```

```

REPEAT
  Write('Hit any key to run the overlays (^Z to stop): ');
  Read(Kbd, c);
  WriteLn;
  IF c <> ^Z THEN
    BEGIN
      One;
      Two;
    END;
  WriteLn;

```

UNTIL c = ^Z;

{Free up the heap space used by the replacement overlay handler by passing the same offset as above to the DisposeOverlayStora