

SBUTIL.PAS

```
{*****}
{               S B U T I L . P A S               }
{*-----*}
Task           : Provides functions for initializing the SB
                base structure.
{*-----*}
Author          : Michael Tischer / Bruno Jennrich
Developed on   : 03/20/1994
Last update    : 04/6/1995
{*****}
```

Unit SBUTIL;

Interface

Type

```
SBBASE = Record { describes installed sound card }
  iDspPort,           { Base port of the DS processor }
  iMixPort,           { Base port of the mixer }
  iMpuPort,           { Base port of the MP unit }
  iDspDmaB,           { 8 bit DMA channel being used }
  iDspDmaW,           { 16 bit DMA channel being used }
  iDspIrq             : Integer; { Interrupt line being used }
  uDspVersion         : Word;    { DSP version number }
  pDspName            : String;  { Name of Sound Blaster card }
End;
```

{-- Prototypes -----}

```
Function sb_GetEnviron( var sbbase : SBBASE; Env : String ) : Integer;
Procedure sb_Print( var sbbase : SBBASE );
Function sb_LoadDriver( Name : String ) : Longint;
```

```
Procedure sb_UnloadDriver( lpEntry : Longint );
```

```
Implementation
```

```
Uses DOS,ARGS;
```

```
Const
```

```
    ERROR      = -1;
```

```
    NO_ERROR   = 0;
```

```
{*****}
{ Hex : converts a hex number into a hex string }
{ *-----* }
{ Input : hz - hex value to be converted }
{ Output : converted hex string }
{*****}
```

```
Function hex( hz : Word ) : String;
```

```
var i : Integer;
```

```
    hs : String;
```

```
Begin
```

```
    hs := '';
```

```
    Repeat
```

```
        i := hz mod 16;
```

```
        hz := hz div 16;
```

```
        case i of
```

```
            0..9 : hs := chr( i + ord( '0' ) ) + hs;
```

```
            10..15: hs := chr( i - 10 + ord( 'A' ) ) + hs;
```

```
        End;
```

```
    Until hz = 0;
```

```
    if Length( hs ) mod 2 <> 0 then hs := '0' + hs;
```

```
    hex := hs;
```

End;

```
{*****}
{  sb_GetEnviron : Initialize SB base structure based on passed
                    environment string (BLASTER environment variable)
{*****}
{*-----*}
{  Input : SBBASE - structure to be initialized
           Env   - BLASTER environment variable
{  Output : = 0 : Initialization free of error
           = -1 : Error during initialization
{  Info : - This structure must be passed to the individual modules
           which then carry out addition initializations of the
           structure (e.g.; DSP version number).
{*****}
Function sb_GetEnviron( var sbbase : SBBASE; Env : String ) : Integer;
```

Begin

if Env <> '' then

Begin

sbbase.iDspPort := htoi(strchr(Env, 'A'), -1);

sbbase.iMixPort := htoi(strchr(Env, 'M'), -1);

if sbbase.iMixPort = -1 then

sbbase.iMixPort := sbbase.iDspPort;

sbbase.iMpuPort := htoi(strchr(Env, 'P'), -1);

sbbase.uDspVersion := \$FFFF; { will be initialized later }

sbbase.iDspIrq := htoi(strchr(Env, 'I'), -1);

sbbase.iDspDmaB := htoi(strchr(Env, 'D'), -1);

sbbase.iDspDmaW := htoi(strchr(Env, 'H'), -1);

sb_GetEnviron := NO_ERROR;

End

else

sb_GetEnviron := ERROR;

End;

```
{ ***** }
{ sb_Print : Display SBBASE structure }
{ *-----* }
{ Input : SBBASE - structure to be displayed }
{ ***** }
Procedure sb_Print( var sbbase : SBBASE );
```

Begin

```
Writeln( 'DSP port: ', hex( sbbase.iDspPort ) );
Writeln( 'MIX port: ', hex( sbbase.iMixPort ) );
Writeln( 'MPU port: ', hex( sbbase.iMpuPort ) );
if sbbase.uDspVersion <> $FFFF then
  Begin
    Writeln( 'DSP version: ', HI( sbbase.uDspVersion ), '.',
              LO( sbbase.uDspVersion ) );
    Writeln( 'card : ', sbbase.pDspName );
  End
else
  Writeln( 'DSP not initialized!' );

Writeln( 'DSP-IRQ: ', sbbase.iDspIrq );
Writeln( 'DSP- 8 Bit DMA: ', sbbase.iDspDmaB );
Writeln( 'DSP-16 Bit DMA: ', sbbase.iDspDmaW );
End;
```

```
{ ***** }
{ AllocMem : Allocate memory through DOS }
{ *-----* }
{ Input : lNumPara - Size of memory area to be allocated }
{           in paragraphs }
{ }
```

```

{          uSegment - Word-Variable, in which the segment address
{          of the allocated memory area is
{          returned
{*****}
Function AllocMem( lNumPara : Word; var uSegment : Word ) : Integer;

```

```

var regs : Registers;

```

```

Begin

```

```

    regs.ah := $48;
    regs.bx := lNumPara;
    msdos( regs );
    if ( regs.flags and 1 ) = 1 then
        allocmem := 0 { no memory could be allocated }
    else
        Begin
            allocmem := -1; { everything is OK }
            uSegment := regs.ax; { return segment address }
        End;

```

```

End;

```

```

{*****}
{ ReleaseMem : Release memory allocated by
{       AllocMem
{*****}
{-----*
{ Input : usegment - Segment returned by a call for
{       AllocMem
{*****}

```

```

Procedure ReleaseMem( uSegment : Word );

```

```

var regs : Registers;

```

Begin

```

regs.ah := $49;
regs.es := uSegment;
msdos( regs );
End;

{*****}
{ sb_LoadDriver : Load Sound Blaster driver }
{ *-----* }
{ Input : Name - Name of driver to be loaded (complete file name) }
{ Output : Address of driver entry point or NULL, if the }
{          driver could not be loaded }
{*****}
Function sb_LoadDriver( Name:String ) : Longint;

var   iInc, uDummy,
      uSegment      : Word;
      lFileSize,
      lNumSeg,lpMem : Longint;
      iHandle       : File;

Begin
  uSegment := 0;
  {$i-}
  Assign( iHandle, Name );
  Reset( iHandle, 1 );
  {$i+}

  if ioresult = 0 then
    Begin { Able to open file }
      lFileSize := FileSize( iHandle ); { Get size of driver }
      lNumSeg := (lFileSize+15) div 16; { Size in paragraphs }
      if lNumSeg < $FFFF then

```

```

if AllocMem( Word ( lNumSeg ), uSegment ) <> 0 then
  Begin { able to allocate enough memory }
    lpMem := Longint( uSegment ) shl 16;
    while lFileSize > 0 do
      Begin
        if lFileSize > 65535 then iInc := 65535
          else iInc := lFileSize;
        BlockRead( iHandle, Pointer( lpMem )^, iInc, uDummy);
        if iInc = uDummy then
          Begin
            lpMem := lpmem + iInc;
            lFileSize := lFileSize - iInc;
          End
        else
          { Release after read error }
          Begin
            ReleaseMem( uSegment );
            uSegment := 0;
            lFileSize := 0;
          End;
        End;
      End
    else
      uSegment := 0;
    close( iHandle );
  End;
  sb_Loaddriver := Longint( uSegment ) shl 16 ;
End;

{ ***** }
{ sb_UnloadDriver : remove Sound Blaster driver previously loaded by }
{ sb_LoadDriver from memory }
{ *-----* }

```



```

{ Input : lpEntry : Address of memory block which the driver      }
{                               is allocating.                     }
{ ***** }
Procedure sb_UnloadDriver( lpEntry : Longint );

Begin
    if lpEntry <> 0 then ReleaseMem( lpEntry shr 16 );
End;

End.

```