Horizontal Double Screen Bitmap Scroller

Done with VSP

Download

```
//-------------------------------------------------------------------------
-----------------------------------------
// Horizontal Double Screen Bitmap Scroller, done with VSP
// For the C64pixels.com Double Screen Pixeling Compo
// Code: Cruzer/CML, April 2011
// Based on a routine from Codebase64 by Martin Piper
// Can be used/modified/distributed freely
// Asm: KickAss 3.15
// Tab: 16 chars
//-------------------------------------------------------------------------
-----------------------------------------
// Picture size: 80x25 chars, in either multicolor or hires bitmap.
// Gfx can be imported either as two c64 pics or a PNG file.
// C64 pics:
// - Set importMode = C64_MODE
// - Save pics as "pic0.prg" and "pic1.prg", or *.p00
// - For multicolor pics use Koala format
// - For hires pics use Topaz Hires Editor format (starts with color screen,
followed by bitmap at a $400 offset)
// - Set c64FileType to "prg" or "p00"
// PNG:
// - Set importMode = PNG_MODE
// - Save pic as "pic.png"
// - Size: 640x201 (only the lower 200 lines are used for gfx)
// - The upper line is used for defining the c64 palette, with hires or
multicolor sized pixels depending on the mode
// - The pixel to the right of the palette defines the background color
// For both formats:
// - Set multiColorMode = true/false
// - Set borderColor manually or leave it as -1 to make it the same as the
bg color
// - Adjust sineLength for slower/faster speed
//-------------------------------------------------------------------------
-----------------------------------------
// Code info:
// VSP can only push the screen 40 chars, so with extended borders the
visible pic could only be 78 chars wide.
// To get full double screen size I added an extra displaced bitmap screen
for the two last chars.
// When switching to/from this, the d800 colors need to be moved, since they
can't be bank switched.
// This would of course have been easier with linecrunch, but I wasn't in
the mood for finding out whether it was
```

```
// possible while keeping the gfx 25 chars high.
//-------------------------------------------------------------------------------
----------------------------------------
//params...
.enum {PNG_MODE, C64_MODE}
.const importMode = PNG_MODE
.const multiColorMode = true
.const c64FileType = "prg"
.const sineAmp = $150
.const sineLength = $cc
.var borderColor = -1
//-------------------------------------------------------------------------------
----------------------------------------
//zeropage...
.const pnt0 =       $02 //2
.const pnt1 =       $04 //2
.const xPos =       $06 //1
.const yPos =       $07 //1
.const xPosChr =    $08 //1
.const source =     $09 //2
.const target =     $0b //3
.const srcPos =     $0e //1
.const destPos =    $0f //1
.const destPosBitmap =  $10 //2
.const dir =        $12 //1
.const d16 =        $13 //1
.const dmaDelay =   $14 //1
//-------------------------------------------------------------------------------
----------------------------------------
//memory...
.const basic =      $0801 //080f
.const tune =       $1000 //1fff
.const main =       $2000 //3fff
.const sineLo =     $4400 //47ff
.const sineHi =     $4800 //4bff
.const bitmapLut =  $4c00 //8bff
.const screen =     $8c00 //8fff
.const screenLut =  $9000 //97ff
.const d800Lut =    $9800 //9fff
.const bitmap =     $a000 //bfff
.const bitmap2 =    $c000 //dfff
.const screen2 =    $e000 //e3ff
//-------------------------------------------------------------------------------
----------------------------------------
.const unused = $fff6
.const border = unused
.const irqLine = $2d
.const ProcessorPortDDRDefault = %00101111
.const ProcessorPortAllRAMWithIO = %00100101
.const CIA1InterruptControl =   $dc0d
```

```
.const CIA1TimerAControl =  $dc0e
.const CIA1TimerBControl =  $dc0f
.const CIA2InterruptControl =   $dd0d
.const CIA2TimerAControl =  $dd0e
.const CIA2TimerBControl =  $dd0f
.const VIC2InteruptControl =    $d01a
.const VIC2InteruptStatus = $d019
.var bgColor
//-------------------------------------------------------------------------
-----------------------------------------
//general scripts...

.function _16bit_nextArgument(arg) {
    .if (arg.getType()==AT_IMMEDIATE) .return CmdArgument(arg.getType(),
>arg.getValue())
    .return CmdArgument(arg.getType(),arg.getValue()+1)
}

.pseudocommand mb arg1;arg2 { //move byte
    lda arg1
    sta arg2
}

.pseudocommand mw src;tar { //move word
    lda src
    sta tar

    .if (src.getType() == AT_IZEROPAGEY) {
        iny
        lda src
    } else {
        lda _16bit_nextArgument(src)
    }

    .if (tar.getType() == AT_IZEROPAGEY) {
        .if (src.getType() != AT_IZEROPAGEY) iny
        sta tar
    } else {
        sta _16bit_nextArgument(tar)
    }
}
//-------------------------------------------------------------------------
-----------------------------------------
//scripting for png import...

.function getC64Color(xPos, yPos, pic, rgb2c64) {
    .if (multiColorMode) .eval xPos = xPos*2
    .var rgb = pic.getPixel(xPos, yPos+1)
    .var c64Color = rgb2c64.get(rgb)
    .return c64Color
}
```

```
.var charWidth = 8
.var numBlockColors = 2
.if (multiColorMode) {
    .eval charWidth = 4
    .eval numBlockColors = 3
}
.function getBlockColors(chrX, chrY, pic, rgb2c64) {
    .var colorCounts = List()
    .for (var i=0; i<16; i++) .eval colorCounts.add(0)
    .for (var pixY=0; pixY<8; pixY++) {
    .for (var pixX=0; pixX<charWidth; pixX++) {
        .var c64Color = getC64Color(chrX*charWidth + pixX, chrY*8 + pixY,
pic, rgb2c64)
        .eval colorCounts.set(c64Color, colorCounts.get(c64Color) + 1)
    }}
    .if (multiColorMode) .eval colorCounts.set(bgColor,0)
    .for (var i=0; i<16; i++) .eval colorCounts.set(i, [colorCounts.get(i)
<< 4] | i)
    .eval colorCounts.sort()
    .eval colorCounts.reverse()
    .var blockColors = List()
    .for (var i=0; i<16; i++) .eval blockColors.add(0)
    .for (var i=0; i<numBlockColors; i++) {
        .var c64Color = colorCounts.get(i) & $0f
        .eval blockColors.set(c64Color, i+1)
    }
    .return blockColors
}

.function getBlock(chrX, chrY, pic, rgb2c64) {
    .var blockColors = getBlockColors(chrX, chrY, pic, rgb2c64)
    .var blockData = List()
    .for (var pixY=0; pixY<8; pixY++) {
        .var bitmapByte = 0
    .for (var pixX=0; pixX<charWidth; pixX++) {
        .var c64Color = getC64Color(chrX*charWidth + pixX, chrY*8 + pixY,
pic, rgb2c64)
        .var multiColor = blockColors.get(c64Color)
        .if (multiColorMode) .eval bitmapByte = bitmapByte | [multiColor <<
[6 - pixX*2]]
        .if (!multiColorMode) .eval bitmapByte = bitmapByte |
[[[multiColor-1]^1] << [7 - pixX]]
    }
        .eval blockData.add(bitmapByte)
    }
    .for (var multiColor=1; multiColor<4; multiColor++) {
        .var c64Color = 0
        .for (var i=1; i<16; i++) {
            .if (blockColors.get(i) == multiColor) .eval c64Color = i
        }
```

```
        .eval blockData.add(c64Color)
    }
    .return blockData
}

.var bitmapData = List()
.var screenData = List()
.var d800Data = List()

.function parseDoublePic(pic, rgb2c64) {
    .for (var picNo=0; picNo<2; picNo++) {
    .for (var chrY=0; chrY<25; chrY++) {
    .for (var chrX=0; chrX<40; chrX++) {
        .var block = getBlock(chrX+picNo*40, chrY, pic, rgb2c64)
        .for (var i=0; i<8; i++) .eval bitmapData.add(block.get(i))
        .var scrColor = [block.get(8) << 4] | [block.get(9) & $f]
        .eval screenData.add(scrColor)
        .if (multiColorMode) .eval d800Data.add(block.get(10))
    }}}
}

.function initPalette(pngPic) {
    .var rgb2c64 = Hashtable()
    .for (var i=0; i<16; i++) {
        .var xPos = i
        .if (multiColorMode) .eval xPos = i*2
        .var rgb = pngPic.getPixel(xPos,0)
        .eval rgb2c64.put(rgb, i)
    }
    .var bgRgb = pngPic.getPixel(32,0)
    .eval bgColor = rgb2c64.get(bgRgb)
    .if (borderColor == -1) .eval borderColor = bgColor
    .return rgb2c64
}

.if (importMode == PNG_MODE) {
    .var pngPic = LoadPicture("pic.png")
    .var rgb2c64 = initPalette(pngPic)
    .eval parseDoublePic(pngPic, rgb2c64)
}
//-------------------------------------------------------------------------
-----------------------------------------
//scripting for c64 file import...

.const KOALA_PRG = "Bitmap=$0002, Screen=$1f42, D800=$232a,
BackgroundColor=$2712"
.const KOALA_P00 = "Bitmap=$001c, Screen=$1f5c, D800=$2344,
BackgroundColor=$272c"
.const HIRES_PRG = "Screen=$0002, Bitmap=$0402"
.const HIRES_P00 = "Screen=$001c, Bitmap=$041c"
.var c64Pics = List()
```

```
.if (importMode == C64_MODE) {
    .if (multiColorMode && c64FileType == "prg") {
        .eval c64Pics.add(LoadBinary("pic0.prg", KOALA_PRG))
        .eval c64Pics.add(LoadBinary("pic1.prg", KOALA_PRG))
    }
    .if (multiColorMode && c64FileType == "p00") {
        .eval c64Pics.add(LoadBinary("pic0.p00", KOALA_P00))
        .eval c64Pics.add(LoadBinary("pic1.p00", KOALA_P00))
    }
    .if (!multiColorMode && c64FileType == "prg") {
        .eval c64Pics.add(LoadBinary("pic0.prg", HIRES_PRG))
        .eval c64Pics.add(LoadBinary("pic1.prg", HIRES_PRG))
    }
    .if (!multiColorMode && c64FileType == "p00") {
        .eval c64Pics.add(LoadBinary("pic0.p00", HIRES_P00))
        .eval c64Pics.add(LoadBinary("pic1.p00", HIRES_P00))
    }
    .if (multiColorMode) .eval bgColor = c64Pics.get(0).getBackgroundColor()
    .if (!multiColorMode) .eval bgColor = 0
    .if (borderColor == -1) .eval borderColor = bgColor
}
//-------------------------------------------------------------------------
----------------------------------------
//generate look-up tables with bitmap data...

.pc = bitmapLut "bitmapLut"
.for (var yPos=0; yPos<200; yPos++) {
    .for (var pic=0; pic<2; pic++) {
        .for (var xPos=0; xPos<40; xPos++) {
            .if (importMode == C64_MODE) {
                .by c64Pics.get(pic).getBitmap([yPos>>3]*$140 + [yPos&7] +
xPos*8)
            }
            .if (importMode == PNG_MODE) {
                .by bitmapData.get([yPos>>3]*$140 + [yPos&7] + xPos*8 +
pic*8000)
            }
        }
    }
}

.pc = screenLut "screenLut"
.for (var yPos=0; yPos<25; yPos++) {
    .for (var pic=0; pic<2; pic++) {
        .for (var xPos=0; xPos<40; xPos++) {
            .if (importMode == C64_MODE) {
                .by c64Pics.get(pic).getScreen(yPos*40 + xPos)
            }
            .if (importMode == PNG_MODE) {
                .by screenData.get(yPos*40 + xPos + pic*1000)
```

```
            }
        }
    }
}

.if (multiColorMode) {
.pc = d800Lut "d800Lut"
.for (var yPos=0; yPos<25; yPos++) {
    .for (var pic=0; pic<2; pic++) {
        .for (var xPos=0; xPos<40; xPos++) {
            .if (importMode == C64_MODE) {
                .by c64Pics.get(pic).getD800(yPos*40 + xPos)
            }
            .if (importMode == PNG_MODE) {
                .by d800Data.get(yPos*40 + xPos + pic*1000)
            }
        }
    }
}
}
//--------------------------------------------------------------------------
-----------------------------------------
.pc = basic "basic"
:BasicUpstart(main)

.var sineValues = List()
.for (var i=0; i<sineLength; i++) {
    .var val = sineAmp/2 * sin(i/[sineLength/2/PI])
    .eval val = sineAmp/2 + val * 0.999999999
    .eval sineValues.add(floor(val))
}

.pc = sineLo "sineLo"
    .var d016mc = $00
    .if (multiColorMode) .eval d016mc = $10
    .fill sineLength, [sineValues.get(i) & 7] | d016mc

.pc = sineHi "sineHi"
    .fill sineLength, sineValues.get(i) >> 3
//--------------------------------------------------------------------------
-----------------------------------------
.pc = main "main"
    sei
    cld
    ldx #$ff
    txs
    jsr init
    cli
//--------------------------------------------------------------------------
-----------------------------------------
{
```

```
mainLoop:
!:  lda topIRQDone
    beq !-
    :mb #0; topIRQDone
    lda $d011
    bpl *-3
    inc border

    jsr setScrollParams
    .if (multiColorMode) jsr updateD800
    jsr updateBitmap
    jsr incCnt
    :mb #0; border
    jmp mainLoop
}
//---------------------------------------------------------------------------
-----------------------------------------
.align $100
topIrq:
    //stabilize raster with double irq
    //@ line 45
    pha
    txa
    pha
    tya
    pha

    inc topIRQDone

    :mw #irq2; $fffe
    inc $d012
    :mb #1; VIC2InteruptStatus    //ack raster interupt

    tsx
    cli
    //these nops never really finish due to the raster IRQ triggering again
    nop
    nop
    nop
    nop
    nop
    nop
    nop
    nop
    nop
    nop
    nop
    nop
    nop
    nop
    nop
```

```
//-------------------------------------------------------------------------
----------------------------------------
irq2:
    //@ line 46
    txs

    ldx #9
!:  dex
    bne !-

    //final cycle wobble check
    lda #irqLine+1
    cmp $d012
    beq *+2

    //the raster is now stable! \o/

    //@ line 47
    :mb #$31; $d011

    //waste more cycles
    nop
    nop
    nop
    nop
    nop
    nop
    nop
    nop
    nop
    nop
    nop
    nop
    nop
    nop

    //still @ line 47
    lda dmaDelay
    lsr     //divide by 2 to get the number of nops to skip
    sta branch+1
    clv     //force branch always

    //@ line 48

    bcc branch    //1 cycle extra delay depending on the least significant
bit of the x offset
branch: bvc *    //branch somewhere into the nops depending on the x offset
position
    nop
    nop
```

```
    nop
    nop
    nop
    nop
    nop
    nop
    nop
    nop
    nop
    nop
    nop
    nop
    nop
    nop
    nop
    nop
    nop

    //show raster position
    inc border
    dec border

    //do the VSP by tweaking d011 at the correct time
    lda #$3b
    dec $d011
    inc $d011
    sta $d011

    ldx extraScreenOn
    :mb d018s,x; $d018
    :mb dd00s,x; $dd00
    :mb d16; $d016
    //restart the IRQ chain
    :mw #topIrq; $fffe
    :mb #irqLine; $d012
    :mb #1; VIC2InteruptStatus    //ack raster interupt

    pla
    tay
    pla
    tax
    pla
interruptReturn:
    rti

d018s:  .by $38,$80
dd00s:  .by $95,$94
extraScreenOn:  .by 1,1
topIRQDone: .by 0
```

```
//-------------------------------------------------------------------------------
-----------------------------------------
incCnt:
    inc cnt+0
    bne !+
    inc cnt+1
!:  lda cnt+0
    cmp #<sineLength
    bne !+
    lda cnt+1
    cmp #>sineLength
    bne !+
    :mw #0; cnt
!:  rts

cnt:    .wo sineLength/4  //start to the left
//-------------------------------------------------------------------------------
-----------------------------------------
setScrollParams:
    :mb #0; pnt0+0
    lda cnt+1
    clc
    adc #>sineLo
    sta pnt0+1
    ldy cnt+0
    :mb (pnt0),y; d16
    lda cnt+1
    clc
    adc #>sineHi
    sta pnt0+1
    ldx #0
    lda (pnt0),y
    cmp xPosChr
    beq !+
    ldx #1
    cmp xPosChr
    bpl !+
    ldx #-1
!:  sta xPosChr
    stx dir

    ldy #0
    lda #39
    sec
    sbc xPosChr
    bcs !+
    adc #40
    iny
!:  sta dmaDelay
    sty extraScreenOn
    rts
```

```
//-------------------------------------------------------------------------------
-----------------------------------------
updateD800:
.if (multiColorMode) {
    ldy extraScreenOn
    cpy extraScreenOn+1
    beq !+
    cpy #1
    beq extra
    jsr moveD800Normal
    jmp !+
extra:  jsr moveD800Extra
!:
    :mb extraScreenOn; extraScreenOn+1
    rts
}
//-------------------------------------------------------------------------------
-----------------------------------------
updateBitmap:
{
    lda dir
    bne !+
    //no char-pos change
    rts
!:
    lda extraScreenOn
    beq !+
    //no bitmap update needed for extra screen
    rts
!:
    lda xPosChr
    ldx dir
    bmi !+
    clc
    adc #39
!:  sta srcPos

    lda #81
    sec
    sbc srcPos
    sta srcPos
    tax
    lda #40
    ldy dir
    bpl !+
    lda #79
!:  sec
    sbc xPosChr
    sta destPos
    tay
```

```
.for (var i=0; i<24; i++) {
    lda screenLut-1 + i*80,x
    sta screen + i*40,y
    lda d800Lut-1 + i*80,x
    sta $d800 + i*40,y
}

    //special case for lowest line, which needs to wrap to top of screen
when it scrolls far enough
    cpy #64
    bpl wrap
normal:
    lda screenLut-1 + 24*80,x
    sta screen + 24*40,y
    lda d800Lut-1 + 24*80,x
    sta $d800 + 24*40,y
    jmp done
wrap:
    lda screenLut-1 + 24*80,x
    sta screen-$400 + 24*40,y
    lda d800Lut-1 + 24*80,x
    sta $d800-$400 + 24*40,y
done:
    :mb destPos; destPosBitmap+0
    lda #0
    asl destPosBitmap+0
    rol
    asl destPosBitmap+0
    rol
    asl destPosBitmap+0
    rol
    sta destPosBitmap+1
    ldy destPosBitmap+0
    lda destPosBitmap+1
    bne !+
    jmp storeBitmap0
!:  cmp #1
    bne !+
    jmp storeBitmap1
!:  jmp storeBitmap2
}
//-------------------------------------------------------------------------
-----------------------------------------
.macro storeBitmap(offset) {
    .for (var i=0; i<200; i++) {
        lda bitmapLut - 1 + i*80,x
        .var target = bitmap + offset + [i&7] + [[i>>3]*$140]
        .if (offset == $200 && i >= 192) .eval target = target - $2000
//wrapping
        sta target,y
    }
```

```
    rts
}
storeBitmap0:   :storeBitmap($000)
storeBitmap1:   :storeBitmap($100)
storeBitmap2:   :storeBitmap($200)
//-------------------------------------------------------------------------
-----------------------------------------
//move d800 data when switching between normal/special bitmap screen...
.macro moveD800(srcOffset, tarOffset) {
    .for (var yPos=0; yPos<25; yPos++) {
        .if (yPos < 24) {
                ldx #9
            !:
            .for (var xPos=0; xPos<40; xPos=xPos+10) {
                lda d800Lut + xPos + yPos*80 + srcOffset,x
                sta $d800 + [[xPos + yPos*40 + tarOffset] & $3ff],x
            }
                dex
                bpl !-
        } else {
            //avoid loops for lowest line to make it wrappable...
            .for (var xPos=0; xPos<40; xPos++) {
                lda d800Lut + xPos + yPos*80 + srcOffset
                sta $d800 + [[xPos + yPos*40 + tarOffset] & $3ff]
            }
        }
    }
    rts
}
moveD800Normal: .if (multiColorMode) :moveD800($02,$01)
moveD800Extra:  .if (multiColorMode) :moveD800($00,$27)
//-------------------------------------------------------------------------
-----------------------------------------
init:
    :mb #$0b; $d011
    :mb #$00; $d015
    :mb #borderColor; $d020
    :mb #bgColor; $d021
    jsr clearInterrupts
    jsr initGfx
    jsr setScrollParams
    jsr initInterrupts
    rts
//-------------------------------------------------------------------------
-----------------------------------------
clearInterrupts:
    :mb #ProcessorPortDDRDefault; $00
    :mb #ProcessorPortAllRAMWithIO; $01
    // Clear all CIA to known state, interrupts off.
    lda #$7f
```

```
    sta CIA1InterruptControl
    sta CIA2InterruptControl
    lda #0
    sta VIC2InteruptControl
    sta CIA1TimerAControl
    sta CIA1TimerBControl
    sta CIA2TimerAControl
    sta CIA2TimerBControl
    // Ack any interrupts that might have happened from the CIAs
    lda CIA1InterruptControl
    lda CIA2InterruptControl
    // Ack any interrupts that have happened from the VIC2
    :mb #$ff; VIC2InteruptStatus
    // Setup kernal and user mode IRQ vectors to point to a blank routine
    :mw #interruptReturn; $fffe
    :mw #interruptReturn; $fffa
    rts
//-------------------------------------------------------------------------
-----------------------------------------
initInterrupts:
    // Setup raster IRQ
    :mw #topIrq; $fffe
    :mb #1; VIC2InteruptControl
    :mb #irqLine; $d012
    :mb #$3b; $d011

    // Ack any interrupts that might have happened from the CIAs
    lda CIA1InterruptControl
    lda CIA2InterruptControl
    // Ack any interrupts that have happened from the VIC2
    :mb #$ff; VIC2InteruptStatus
    rts
//-------------------------------------------------------------------------
-----------------------------------------
initGfx:
    //init bitmap screens, color screens and d800
    :mw #bitmapLut+2; source
    :mw #$08; target
    :mb #>bitmap; target+2
    jsr initBitmap
    :mw #screenLut+2; source
    :mw #$01; target
    :mb #>screen; target+2
    jsr initScreen
    .if (multiColorMode) jsr moveD800Extra
    :mb #$34; $01
    :mw #bitmapLut; source
    :mw #$138; target
    :mb #>bitmap2; target+2
    jsr initBitmap
    :mw #screenLut; source
```

```
    :mw #$27; target
    :mb #>screen2; target+2
    jsr initScreen
    :mb #$35; $01
    rts
//---------------------------------------------------------------------------
-----------------------------------------
initBitmap:
{
    :mb #0; yPos
yLoop:
    :mw source; pnt0
    :mb target+0; pnt1+0
    lda target+1
    ora target+2
    sta pnt1+1
    :mb #0; xPos
xLoop:
    ldy #0
    :mb (pnt0),y; (pnt1),y
    inc pnt0+0
    bne !+
    inc pnt0+1
!:
    lda #8
    clc
    adc pnt1+0
    sta pnt1+0
    lda pnt1+1
    adc #$00
    and #$1f
    ora target+2
    sta pnt1+1

    inc xPos
    lda xPos
    cmp #40
    bne xLoop

    lda #80
    clc
    adc source+0
    sta source+0
    bcc !+
    inc source+1
!:
    lda #1
    clc
    adc target+0
    sta target+0
```

```
    bcc !+
    inc target+1
!:
    lda yPos
    and #$07
    cmp #$07
    bne !+
    lda #<$138
    clc
    adc target+0
    sta target+0
    lda #>$138
    adc target+1
    and #$1f
    sta target+1
!:
    inc yPos
    lda yPos
    cmp #200
    beq !+
    jmp yLoop
!:
    rts
}
//-------------------------------------------------------------------------------
-----------------------------------------
initScreen:
{
    :mb #0; yPos
yLoop:
    :mw source; pnt0
    :mb target+0; pnt1+0
    lda target+1
    ora target+2
    sta pnt1+1
    :mb #0; xPos
xLoop:
    ldy #0
    :mb (pnt0),y; (pnt1),y
    inc pnt0+0
    bne !+
    inc pnt0+1
!:
    lda #1
    clc
    adc pnt1+0
    sta pnt1+0
    lda pnt1+1
    adc #$00
    and #$03
    ora target+2
```

```
    sta pnt1+1

    inc xPos
    lda xPos
    cmp #40
    bne xLoop

    lda #80
    clc
    adc source+0
    sta source+0
    bcc !+
    inc source+1
!:
    lda #40
    clc
    adc target+0
    sta target+0
    bcc !+
    inc target+1
!:
    inc yPos
    lda yPos
    cmp #25
    beq !+
    jmp yLoop
!:
    rts
}
//-------------------------------------------------------------------------------
-----------------------------------------
```