```
#!/usr/local/bin/basic

'
' Version check
'
if not fre then
    print "cb-shpdraw.bas requires the GUI version of Chipmunk Basic"
    bye
endif

'
' Global configuration input
'
input "Enter window width: ";viewWidth
input "Enter window height: ";viewHeight
input "Enter window color (r,g,b percents): ";viewRed,viewGreen,viewBlue

'
' Graphics initialization
'
macfunction("wintitle","cb-shpdraw")
graphics window 50, 50, viewWidth, viewHeight
graphics color viewRed,viewGreen,viewBlue
graphics fillrect 0,0,viewWidth,viewHeight
let title$ = "Layers:"
let xmin = viewWidth * 0.05
let ymin = viewHeight * 0.05
let areaWidth = viewWidth * 0.9
let areaHeight = viewHeight * 0.9
let firstMinX = 0
let firstMinY = 0
let wRatio = 0
let hRatio = 0

while

    '
    ' Layer configuration
    '
    print
    'files
    input "Specify shapefile (none to stop): ";filename$
    if filename$ = "" then exit while
    input "Symbol color (r,g,b percents): ";symbolRed,symbolGreen,symbolBlue
    input "Symbol size (>=1): ";symbolSize

    '
    ' Determine filenames from input argument
    '
    if right$(filename$, 4) = ".shp" then
        let shpName$ = filename$
        let shxName$ = left$(shpName$,len(shpName$)-4) + ".shx"
```

```
else
    if right$(filename$, 4) = ".shx" then
        let shxName$ = filename$
        let shpName$ = left$(shxName$,len(shxName$)-4) + ".shp"
    else
        let shpName$ = filename$ + ".shp"
        let shxName$ = filename$ + ".shx"
    endif
endif

'
' Open files
'
on error goto checkopen
open shpName$ for data input as #1
open shxName$ for data input as #2
checkopen:
if eof(1) = -1 then
    print "Error: ";shpName$;" could not be opened."
    goto closefiles
endif
if eof(2) = -1 then
    print "Error: ";shxName$;" could not be opened."
    goto closefiles
endif

'
' Read SHP file header
'
let shpCode = ReadInteger(1,1)
              ReadInteger(1,1)
              ReadInteger(1,1)
              ReadInteger(1,1)
              ReadInteger(1,1)
              ReadInteger(1,1)
let shpSize = ReadInteger(1,1)
let shpVers = ReadInteger(1,0)
let shpType = ReadInteger(1,0)
let shpMinX = ReadDouble(1)
let shpMinY = ReadDouble(1)
let shpMaxX = ReadDouble(1)
let shpMaxY = ReadDouble(1)

'
' Read SHX file header
'
let shxCode = ReadInteger(2,1)
              ReadInteger(2,1)
              ReadInteger(2,1)
              ReadInteger(2,1)
              ReadInteger(2,1)
              ReadInteger(2,1)
```

```
let shxSize = ReadInteger(2,1)
let shxVers = ReadInteger(2,0)
let shxType = ReadInteger(2,0)
let shxMinX = ReadDouble(2)
let shxMinY = ReadDouble(2)
let shxMaxX = ReadDouble(2)
let shxMaxY = ReadDouble(2)

'
' Validate file headers
' Should also check matching extents and possibly sensible sizes
' Perhaps skip and warn if the extent does not lie within the
' extent established by the first layer at all.
'
if eof(1) or eof(2) then
    print "Error: End of file encountered while reading headers:"
    print "       ";shpName$;" or ";shxName$
    goto closefiles
endif
if (shpCode <> 9994) or (shxCode <> 9994) then
    print "Error: Invalid file codes (should be 9994):"
    print "       ";shpName$;": ";str$(shpCode);", ";shxName$;": ";str$(shxCode)
    goto closefiles
endif
if (shpVers <> 1000) or (shxVers <> 1000) then
    print "Warning: Unsupported version (expected 1000):"
    print "         ";shpName$;": ";str$(shpVers);", ";shxName$;": ";str$(shxVers)
    print "         Attempting to continue..."
endif
if shpType <> shxType then
    print "Error: Mismatched shape types:"
    print "       ";shpName$;": ";str$(shpType);", ";shxName$;": ";str$(shxType)
    goto closefiles
endif
if (shpType <> 1) and (shpType <> 3) and (shpType <> 5) and (shpType <> 8) then
    print "Error: Unsupported shape type (1, 3, 5, and 8 are valid):"
    print "       ";shpName$;": ";str$(shpType)
    goto closefiles
endif

'
' Interpret header info
' Since SHX index records are constant size (4 words or 8 bytes)
' the number of records (shapes) can be easily determined. The
' index file header is 100 bytes long (50 16-bit words).
'
let shpCount  = (shxSize - 50) / 4
let shpWidth  = shpMaxX - shpMinX
let shpHeight = shpMaxY - shpMinY

'
' Configure layer graphics
```

```
' Scaling parameters tied to the first shapefile displayed
'
let title$ = title$ + " " + shpName$
macfunction( "wintitle", title$ )
graphics pensetup symbolSize, symbolSize
graphics color symbolRed,symbolGreen,symbolBlue
if not firstMinX then let firstMinX = shpMinX
if not firstMinY then let firstMinY = shpMinY
if not wRatio then let wRatio = areaWidth / shpWidth
if not hRatio then let hRatio = areaHeight / shpHeight
let symbolOffset = symbolSize / 2

' There is a good deal of code repetition in the
' following sections, but it is so that we are
' not hampered by pokey conditionals for every
' single line, shape, and point in the file.

'
' Handle polyline and polygon shapefiles
'
if (shpType = 3) or (shpType = 5) then

    print "Rendering polyline or polygon shapefile"

    for shpi = 0 to shpCount - 1

        '
        ' Seek to index record to locate shape record
        '
        fseek #2, ((8 * shpi) + 100)
        let shpOffset = ReadInteger(2,1)
        let shpLength = ReadInteger(2,1)

        '
        ' Seek and read generic shape record header
        '
        fseek #1, (2 * shpOffset)
        let recNumber = ReadInteger(1,1)
        let recLength = ReadInteger(1,1)

        '
        ' Display record if non-null
        '
        let recType = ReadInteger(1,0)
        if (recType = 3) or (recType = 5) then

            '
            ' Read remainder of shape record header
            '
            let recMinX = ReadDouble(1)
            let recMinY = ReadDouble(1)
            let recMaxX = ReadDouble(1)
```

```
            let recMaxY = ReadDouble(1)
            let recParts = ReadInteger(1,0)
            let recPoints = ReadInteger(1,0)

            '
            ' Read past part indices (use them, someday)
            '
            for parti = 1 to recParts
                ReadInteger(1,0)
            next parti

            '
            ' Read and display line segments
            '
            let viewX = ((ReadDouble(1) - firstMinX) * wRatio) + xmin
            let viewY = areaHeight - ((ReadDouble(1) - firstMinY) * hRatio) + ymin
            graphics moveto viewX-symbolOffset, viewY-symbolOffset
            for pointi = 2 to recPoints
                let viewX = ((ReadDouble(1) - firstMinX) * wRatio) + xmin
                let viewY = areaHeight - ((ReadDouble(1) - firstMinY) * hRatio) + ym
                graphics lineto viewX-symbolOffset, viewY-symbolOffset
            next pointi

        endif

    next shpi

endif

'
' Handle multipoint shapefiles
'
if (shpType = 8) then

    print "Rendering multipoint shapefile"

    for shpi = 0 to shpCount - 1

        '
        ' Seek to index record to locate shape record
        '
        fseek #2, ((8 * shpi) + 100)
        let shpOffset = ReadInteger(2,1)
        let shpLength = ReadInteger(2,1)

        '
        ' Seek and read generic shape record header
        '
        fseek #1, (2 * shpOffset)
        let recNumber = ReadInteger(1,1)
        let recLength = ReadInteger(1,1)
```

```
        '
        ' Display record if non-null
        '
        if ReadInteger(1,0) = 8 then

            '
            ' Read remainder of shape record header
            '
            let recMinX   = ReadDouble(1)
            let recMinY   = ReadDouble(1)
            let recMaxX   = ReadDouble(1)
            let recMaxY   = ReadDouble(1)
            let recPoints = ReadInteger(1,0)

            '
            ' Read and plot multipoints
            '
            for pointi = 1 to recPoints
                let viewX = ((ReadDouble(1) - firstMinX) * wRatio) + xmin
                let viewY = areaHeight - ((ReadDouble(1) - firstMinY) * hRatio) + ym
                'graphics pset viewX, viewY
                graphics moveto viewX, viewY
                graphics circle symbolSize
            next pointi

        endif

    next shpi

endif

'
' Handle single-point shapefiles
'
if (shpType = 1) then

    print "Rendering single-point shapefile"

    for shpi = 0 to shpCount - 1

        '
        ' Seek to index record to locate shape record
        '
        fseek #2, ((8 * shpi) + 100) ' if always in sequence, unnecessary
        let shpOffset = ReadInteger(2,1)
        let shpLength = ReadInteger(2,1)

        '
        ' Seek and read generic shape record header
        '
        fseek #1, (2 * shpOffset)
        let recNumber = ReadInteger(1,1)
```

```
            let recLength = ReadInteger(1,1)

            '
            ' Display record point if non-null
            '
            if ReadInteger(1,0) = 1 then
                let viewX = ((ReadDouble(1) - firstMinX) * wRatio) + xmin
                let viewY = areaHeight - ((ReadDouble(1) - firstMinY) * hRatio) + ymin
                'graphics pset viewX, viewY
                graphics moveto viewX, viewY
                graphics circle symbolSize
            endif

        next shpi

    endif

    '
    ' Close files (if open)
    '
    closefiles:
    close #1
    close #2

wend

'
' Quit
'
end



'
' ReadInteger
'
' Return the value of a four-byte integer read from file #filenum
'
' Parameters:
'  filenum, the input file channel number
'  big, boolean indicating whether to interpret as a "big-endian" value
'
sub ReadInteger (filenum, big)

    let byte1 = fgetbyte(filenum) ' big endian high / little endian low
    let byte2 = fgetbyte(filenum)
    let byte3 = fgetbyte(filenum)
    let byte4 = fgetbyte(filenum) ' big endian low / little endian high

    if big then
        ReadInteger = (16777216 * byte1) + (65536 * byte2) + (256 * byte3) + (byte4)
    else
```

```
        ReadInteger = (16777216 * byte4) + (65536 * byte3) + (256 * byte2) + (byte1)
    endif

end sub

'
' ReadDouble
'
' Return the value of an eight-byte floating point "double-precision" number
' read from file #filenum. All shapefile doubles appear to be little endian.
' The tricky thing is that cbas uses 8-byte doubles (or more?) for everything.
'
' Notes:
'  This routine does not check for or recognize "infinity" or "NaN" status
'
' Sources:
'  http://www.cs.princeton.edu/introcs/91float/
'  http://babbage.cs.qc.edu/courses/cs341/IEEE-754.html
'  http://en.wikipedia.org/wiki/Binary_numeral_system
'  http://en.wikipedia.org/wiki/IEEE_floating-point_standard
'
' Parameters:
'  filenum, the input file channel number
'
sub ReadDouble (filenum)

    '
    ' Useful values
    '
    let highBit      =  128 ' 128 = b10000000
    let nonHighBits  =  127 ' 127 = b01111111
    let highNibble   =  240 ' 240 = b11110000
    let lowNibble    =   15 '  15 = b00001111
    let exponentBias = 1023 ' subtract from stored exponent to determine actual exponent
    let twoToThe52   = 4503599627370496

    '
    ' Read bytes from file
    '
    let byte1 = fgetbyte(filenum) ' little endian low
    let byte2 = fgetbyte(filenum)
    let byte3 = fgetbyte(filenum)
    let byte4 = fgetbyte(filenum)
    let byte5 = fgetbyte(filenum)
    let byte6 = fgetbyte(filenum)
    let byte7 = fgetbyte(filenum)
    let byte8 = fgetbyte(filenum) ' little endian high

    '
    ' Determine sign
    '  Encoded in the highest bit of the highest byte
    '
```

```
    if (byte8 and highBit) = highBit then
        let sign = -1
    else
        let sign = 1
    endif

    '
    ' Determine exponent
    '  Stored from the second highest bit of the highest byte
    '  to the fourth bit of the second highest byte (11 bits)
    '
    ' ignore the high bit used for the sign
    let exponent = (byte8 and nonHighBits)
    ' shift those 7 bits left four places to make room for the
    ' bits stored in the upper half of byte 7 (times 2 to shift
    ' 1 place; times 4 to shift 2 places, times 8 to shift 3, etc.)
    let exponent = exponent * 16
    ' add the value of the upper half of byte 7 to the exponent
    ' divide by 16 to shift it right to occupy the places freed above
    let exponent = exponent + ((byte7 and highNibble) / 16)
    ' subtract the exponent bias from the calculated value
    ' this allows a value from negative exponentBias to positive
    ' exponentBias to be stored in an unsigned fashion from 0
    ' to twice exponentBias.
    let exponent = exponent - exponentBias

    '
    ' Determine mantissa (fractional component of scientific notation expression)
    ' These terms could easily be put in one line, but I like the aesthetics here
    '
    let mantissa = 0
    let mantissa = mantissa + (281474976710656 * (byte7 and lowNibble))
    let mantissa = mantissa + (1099511627776 * byte6)
    let mantissa = mantissa + (4294967296 * byte5)
    let mantissa = mantissa + (16777216 * byte4)
    let mantissa = mantissa + (65536 * byte3)
    let mantissa = mantissa + (256 * byte2)
    let mantissa = mantissa + (byte1)
    ' if the value at this point is zero,
    ' leave it as-is so the result will be zero too
    if mantissa <> 0 then
        ' bring the integer representation into a 1.fraction form
        let mantissa = mantissa / twoToThe52
        let mantissa = mantissa + 1
    endif

    '
    ' Return value
    '
    ReadDouble = sign * mantissa * (2 ^ exponent)

end sub
```