```
// "Build Engine & Tools" Copyright (c) 1993-1997 Ken Silverman
// Ken Silverman's official web site: "http://www.advsys.net/ken"
// See the included license file "BUILDLIC.TXT" for license info.

Build information to get you started:

The first half of this file explains the .ART, .MAP, and PALETTE.DAT formats.
The second half has documentation about every BUILD engine function, what it
   does, and what the parameters are.

-Ken S.

------------------------------------------------------------------------------
Documentation on Ken's .ART file format                    by Ken Silverman

   I am documenting my ART format to allow you to program your own custom
art utilites if you so desire.  I am still planning on writing the script
system.

   All art files must have xxxxx###.ART.  When loading an art file you
should keep trying to open new xxxxx###'s, incrementing the number, until
an art file is not found.


1. long artversion;

      The first 4 bytes in the art format are the version number.  The current
   current art version is now 1.  If artversion is not 1 then either it's the
   wrong art version or something is wrong.

2. long numtiles;

      Numtiles is not really used anymore.  I wouldn't trust it.  Actually
   when I originally planning art version 1 many months ago, I thought I
   would need this variable, but it turned it is was unnecessary.  To get
   the number of tiles, you should search all art files, and check the
   localtilestart and localtileend values for each file.

3. long localtilestart;

      Localtilestart is the tile number of the first tile in this art file.

4. long localtileend;

      Localtileend is the tile number of the last tile in this art file.
      Note:  Localtileend CAN be higher than the last used slot in an art
      file.

         Example:  If you chose 256 tiles per art file:
      TILES000.ART -> localtilestart = 0,   localtileend = 255
      TILES001.ART -> localtilestart = 256, localtileend = 511
      TILES002.ART -> localtilestart = 512, localtileend = 767
      TILES003.ART -> localtilestart = 768, localtileend = 1023

5. short tilesizx[localtileend-localtilestart+1];
```

```
      This is an array of shorts of all the x dimensions of the tiles
   in this art file.  If you chose 256 tiles per art file then
   [localtileend-localtilestart+1] should equal 256.

6. short tilesizy[localtileend-localtilestart+1];

      This is an array of shorts of all the y dimensions.

7. long picanm[localtileend-localtilestart+1];

      This array of longs stores a few attributes for each tile that you
   can set inside EDITART.  You probably won't be touching this array, but
   I'll document it anyway.

   Bit:  |31           24|23           16|15            8|7             0|
         ---------------------------------------------------------------
         | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
         ---------------------------------------------------------------
                 | Anim. |  Signed char  |  Signed char  |   |  Animate  |
                 | Speed |   Y-center    |   X-center    |   |   number  |
                 --------|    offset     |    offset     |   |------------
                         ---------------------------------|   ------------
                                                          | Animate type:|
                                                          | 00 - NoAnm   |
                                                          | 01 - Oscil   |
                                                          | 10 - AnmFd   |
                                                          | 11 - AnmBk   |
                                                          ----------------
          You probably recognize these:
       Animate speed -            EDITART key: 'A', + and - to adjust
       Signed char x&y offset -   EDITART key: '``', Arrows to adjust
       Animate number&type -      EDITART key: +/- on keypad

8. After the picanm's, the rest of the file is straight-forward rectangular
      art data.  You must go through the tilesizx and tilesizy arrays to find
      where the artwork is actually stored in this file.

      Note:  The tiles are stored in the opposite coordinate system than
         the screen memory is stored.  Example on a 4*4 file:

         Offsets:
         -----------------
         | 0 | 4 | 8 |12 |
         -----------------
         | 1 | 5 | 9 |13 |
         -----------------
         | 2 | 6 |10 |14 |
         -----------------
         | 3 | 7 |11 |15 |
         -----------------



------------------------------------------------------------------------------
   If you wish to display the artwork, you will also need to load your
palette.  To load the palette, simply read the first 768 bytes of your
palette.dat and write it directly to the video card - like this:

   Example:
      long i, fil;

      fil = open("palette.dat",O_BINARY|O_RDWR,S_IREAD);
      read(fil,&palette[0],768);
      close(fil);

      outp(0x3c8,0);
      for(i=0;i<768;i++)
         outp(0x3c9,palette[i]);
------------------------------------------------------------------------------

Packet format for DUKE3D (specifically for network mode 1, n(n-1) mode):

Example bunch of packets:
A  B  C   D  E  F     G      H      I   J  K  L  M   N...       O
--------------------------------------------------------------------
```

```
d9 00 d9  11 01 00  -  -   -  -   -  -   -  -  -  -  4f       16 31
da 00 da  11 01 00  -  -   -  -   -  -   -  -  -  -  b2       b7 9d
db 00 db  11 01 00  -  -   -  -   -  -   -  -  -  -  b1       24 62
dc 00 dc  11 01 00  -  -   -  -   -  -   -  -  -  -  ca       1d 58
dd 00 dd  11 01 00  -  -   -  -   -  -   -  -  -  -  a9       94 14
de 00 de  11 01 05  00 00  -  -   03 00  -  -  -  -  c5       50 b9
df 00 df  11 01 0f  a1 ff  fe 09  00 00  26 -  -  -  e2       88 6f
e0 00 e0  11 01 04  -  -   -  -   fd ff  -  -  -  -  77       51 d7
e1 00 e1  11 01 03  1f 00  ff 09  -  -   -  -  -  -  ac       14 b7
e2 00 e2  11 01 0b  9c 00  fb 09  -  -   24 -  -  -  f8       6c 22

GAME sends fields D-N
MMULTI adds fields A-C and O for error correction.

A: Packet count sending modulo 256
B: Error state.  Usually 0.  To request a resend, bit 0 is set.  In order
      to catch up on networks, sending many packets is bad, so 2 packets
      are sent in 1 IPX packet.  To send 2 packets in 1 packet, bit 1 is set.
      In special cases, this value may be different.
C: Packet count receiving modulo 256

D: Message header byte.  These are all the possible values currently.  You
      are probably only interested in case 17.  Note that fields E-N apply
      to case 17 only.
     0: send movement info from master to slave (network mode 0 only)
     1: send movement info from slave to master (network mode 0 only)
     4: user-typed messages
     5: Re-start level with given parameters
     6: Send player name
     7: Play Remote Ridicule sound
     8: Re-start level with given parameters for a user map
    17: send movement info to everybody else (network mode 1 only)
   250: Wait for Everybody (Don't start until everybody's done loading)
   255: Player quit to DOS

E: Timing byte used to calculate lag time.  This prevents the 2 computer's
      timers from drifting apart.

F: Bits field byte.  Fields G-M are sent only when certain bits
      in this byte are set.

G: X momentum update (2 bytes).  Sent only if ((F&1) != 0)

H: Y momentum update (2 bytes).  Sent only if ((F&2) != 0)

I: Angle momentum update (2 bytes).  Sent only if ((F&4) != 0)

J: The states of 8 different keys (1 byte).  Sent only if ((F&8) != 0)
K: The states of 8 different keys (1 byte).  Sent only if ((F&16) != 0)
L: The states of 8 different keys (1 byte).  Sent only if ((F&32) != 0)
M: The states of 8 different keys (1 byte).  Sent only if ((F&64) != 0)

N: Sync checking byte.  Useful for debugging programming errors.  Can be a
   variable number of bytes.  Actual number of sync checking bytes is
   calculated by length of the whole packet minus the rest of the bytes sent.

O: CRC-16

--------------------------------------------------------------------------------
|   @@@@@@@@@@     @@@       @@@  @@@@@@@@@@  @@@           @@@@@@@@@@       |
|   @@@@@@@@@@@@   @@@       @@@  @@@@@@@@@@  @@@           @@@@@@@@@@@@     |
|   @@@       @@@@  @@@       @@@     @@@     @@@           @@@    @@@@@     |
|   @@@         @@@  @@@       @@@     @@@     @@@           @@@      @@@@    |
|   @@@       @@@@  @@@       @@@     @@@     @@@           @@@       @@@    |
|   @@@@@@@@@@@@   @@@       @@@     @@@     @@@           @@@       @@@    |
|   @@@@@@@@@@@@   @@@       @@@     @@@     @@@           @@@       @@@    |
|   @@@       @@@@  @@@       @@@     @@@     @@@           @@@       @@@    |
|   @@@         @@@  @@@       @@@     @@@     @@@           @@@      @@@@    |
|   @@@       @@@@  @@@@     @@@@     @@@     @@@           @@@    @@@@@     |
|   @@@@@@@@@@@@    @@@@@@@@@@@   @@@@@@@@@@  @@@@@@@@@@@@  @@@@@@@@@@@      |
|   @@@@@@@@@@        @@@@@@@    @@@@@@@@@@  @@@@@@@@@@@@  @@@@@@@@@@       |
|                                                                              |
|                        M A P   F O R M A T !                                 |
--------------------------------------------------------------------------------
```

```
Here is Ken's documentation on the COMPLETE BUILD map format:
BUILD engine and editor programmed completely by Ken Silverman

Here's how you should read a BUILD map file:
{
   fil = open(???);

      //Load map version number (current version is 7L)
   read(fil,&mapversion,4);

      //Load starting position
   read(fil,posx,4);
   read(fil,posy,4);
   read(fil,posz,4);          //Note: Z coordinates are all shifted up 4
   read(fil,ang,2);           //All angles are from 0-2047, clockwise
   read(fil,cursectnum,2);    //Sector of starting point

      //Load all sectors (see sector structure described below)
   read(fil,&numsectors,2);
   read(fil,&sector[0],sizeof(sectortype)*numsectors);

      //Load all walls (see wall structure described below)
   read(fil,&numwalls,2);
   read(fil,&wall[0],sizeof(walltype)*numwalls);

      //Load all sprites (see sprite structure described below)
   read(fil,&numsprites,2);
   read(fil,&sprite[0],sizeof(spritetype)*numsprites);

   close(fil);
}

      ------------------------------------------------------------
      | @@@@@@@ @@@@@@@ @@@@@@@ @@@@@@@@ @@@@@@@ @@@@@@@  @@@@@@@ |
      | @@      @@      @@         @@    @@   @@ @@   @@@ @@      |
      | @@@@@@@ @@@@@   @@         @@    @@   @@ @@@@@@@  @@@@@@@ |
      |      @@ @@      @@         @@    @@   @@ @@   @@@      @@ |
      | @@@@@@@ @@@@@@@ @@@@@@@    @@    @@@@@@@ @@    @@ @@@@@@@ |
      ------------------------------------------------------------

   //sizeof(sectortype) = 40
typedef struct
{
   short wallptr, wallnum;
   long ceilingz, floorz;
   short ceilingstat, floorstat;
   short ceilingpicnum, ceilingheinum;
   signed char ceilingshade;
   char ceilingpal, ceilingxpanning, ceilingypanning;
   short floorpicnum, floorheinum;
   signed char floorshade;
   char floorpal, floorxpanning, floorypanning;
   char visibility, filler;
   short lotag, hitag, extra;
} sectortype;
sectortype sector[1024];

wallptr - index to first wall of sector
wallnum - number of walls in sector
z's - z coordinate (height) of ceiling / floor at first point of sector
stat's
   bit 0: 1 = parallaxing, 0 = not                              "P"
   bit 1: 1 = sloped, 0 = not
   bit 2: 1 = swap x&y, 0 = not                                 "F"
   bit 3: 1 = double smooshiness                                "E"
   bit 4: 1 = x-flip                                            "F"
   bit 5: 1 = y-flip                                            "F"
   bit 6: 1 = Align texture to first wall of sector             "R"
   bits 7-15: reserved
picnum's - texture index into art file
heinum's - slope value (rise/run) (0-parallel to floor, 4096-45 degrees)
shade's - shade offset of ceiling/floor
pal's - palette lookup table number (0 - use standard colors)
panning's - used to align textures or to do texture panning
visibility - determines how fast an area changes shade relative to distance
```

```
filler - useless byte to make structure aligned
lotag, hitag, extra - These variables used by the game programmer only


          ------------------------------------------------
          | @@      @@ @@@@@@@@ @@      @@      @@@@@@@ |
          | @@      @@ @@    @@ @@      @@      @@      |
          | @@  @@  @@ @@@@@@@@ @@      @@      @@@@@@@ |
          | @@ @@@@ @@ @@    @@ @@      @@           @@ |
          |  @@@ @@@@  @@    @@ @@@@@@@ @@@@@@@ @@@@@@@ |
          ------------------------------------------------|

   //sizeof(walltype) = 32
typedef struct
{
   long x, y;
   short point2, nextwall, nextsector, cstat;
   short picnum, overpicnum;
   signed char shade;
   char pal, xrepeat, yrepeat, xpanning, ypanning;
   short lotag, hitag, extra;
} walltype;
walltype wall[8192];

x, y: Coordinate of left side of wall, get right side from next wall's left side
point2: Index to next wall on the right (always in the same sector)
nextwall: Index to wall on other side of wall (-1 if there is no sector)
nextsector: Index to sector on other side of wall (-1 if there is no sector)
cstat:
   bit 0: 1 = Blocking wall (use with clipmove, getzrange)        "B"
   bit 1: 1 = bottoms of invisible walls swapped, 0 = not         "2"
   bit 2: 1 = align picture on bottom (for doors), 0 = top        "O"
   bit 3: 1 = x-flipped, 0 = normal                               "F"
   bit 4: 1 = masking wall, 0 = not                               "M"
   bit 5: 1 = 1-way wall, 0 = not                                 "1"
   bit 6: 1 = Blocking wall (use with hitscan / cliptype 1)       "H"
   bit 7: 1 = Transluscence, 0 = not                              "T"
   bit 8: 1 = y-flipped, 0 = normal                               "F"
   bit 9: 1 = Transluscence reversing, 0 = normal                 "T"
   bits 10-15: reserved
picnum - texture index into art file
overpicnum - texture index into art file for masked walls / 1-way walls
shade - shade offset of wall
pal - palette lookup table number (0 - use standard colors)
repeat's - used to change the size of pixels (stretch textures)
pannings - used to align textures or to do texture panning
lotag, hitag, extra - These variables used by the game programmer only

      ------------------------------------------------------------
      | @@@@@@@ @@@@@@@ @@@@@@@   @@@@@@ @@@@@@@@ @@@@@@@ @@@@@@@ |
      | @@      @@   @@ @@   @@@    @@      @@    @@      @@      |
      | @@@@@@@ @@@@@@@ @@@@@@@    @@      @@    @@@@@   @@@@@@@ |
      |      @@ @@      @@    @@    @@      @@    @@           @@ |
      | @@@@@@@ @@      @@    @@  @@@@@@    @@    @@@@@@@ @@@@@@@ |
      ------------------------------------------------------------

   //sizeof(spritetype) = 44
typedef struct
{
   long x, y, z;
   short cstat, picnum;
   signed char shade;
   char pal, clipdist, filler;
   unsigned char xrepeat, yrepeat;
   signed char xoffset, yoffset;
   short sectnum, statnum;
   short ang, owner, xvel, yvel, zvel;
   short lotag, hitag, extra;
} spritetype;
spritetype sprite[4096];
x, y, z - position of sprite - can be defined at center bottom or center
cstat:
   bit 0: 1 = Blocking sprite (use with clipmove, getzrange)      "B"
   bit 1: 1 = transluscence, 0 = normal                           "T"
   bit 2: 1 = x-flipped, 0 = normal                               "F"
```

```
   bit 3: 1 = y-flipped, 0 = normal                                "F"
   bits 5-4: 00 = FACE sprite (default)                            "R"
             01 = WALL sprite (like masked walls)
             10 = FLOOR sprite (parallel to ceilings&floors)
   bit 6: 1 = 1-sided sprite, 0 = normal                           "1"
   bit 7: 1 = Real centered centering, 0 = foot center             "C"
   bit 8: 1 = Blocking sprite (use with hitscan / cliptype 1)      "H"
   bit 9: 1 = Transluscence reversing, 0 = normal                  "T"
   bits 10-14: reserved
   bit 15: 1 = Invisible sprite, 0 = not invisible
picnum - texture index into art file
shade - shade offset of sprite
pal - palette lookup table number (0 - use standard colors)
clipdist - the size of the movement clipping square (face sprites only)
filler - useless byte to make structure aligned
repeat's - used to change the size of pixels (stretch textures)
offset's - used to center the animation of sprites
sectnum - current sector of sprite
statnum - current status of sprite (inactive/monster/bullet, etc.)

ang - angle the sprite is facing
owner, xvel, yvel, zvel, lotag, hitag, extra - These variables used by the
                                               game programmer only
--------------------------------------------------------------------------------




--------------------------------------------------------------------------------
|                        IMPORTANT ENGINE FUNCTIONS:                           |
--------------------------------------------------------------------------------

initengine()
   Initializes many variables for the BUILD engine.  You should call this
   once before any other functions of the BUILD engine are used.

uninitengine();
   Frees buffers.  You should call this once at the end of the program
   before quitting to dos.

loadboard(char *filename, long *posx, long *posy, long *posz, short *ang, short *cursectnum)
saveboard(char *filename, long *posx, long *posy, long *posz, short *ang, short *cursectnum)
   Loads/saves the given board file from memory.  Returns -1 if file not
   found.  If no extension is given, .MAP will be appended to the filename.

loadpics(char *filename);
   Loads the given artwork file into memory for the BUILD engine.
   Returns -1 if file not found.  If no extension is given, .ART will
   be appended to the filename.

loadtile(short tilenum)
   Loads a given tile number from disk into memory if it is not already in
   memory.  This function calls allocache internally.  A tile is not in the
   cache if (waloff[tilenum] == 0)

--------------------------------------------------------------------------------
|                           SCREEN STATUS FUNCTIONS:                           |
--------------------------------------------------------------------------------

setgamemode(char vidoption, long xdim, long ydim);
   This function sets the video mode to 320*200*256color graphics.
   Since BUILD supports several different modes including mode x,
   mode 13h, and other special modes, I don't expect you to write
   any graphics output functions.  (Soon I have all the necessary
   functions)  If for some reason, you use your own graphics mode,
   you must call this function again before using the BUILD drawing
   functions.

   vidoption can be anywhere from 0-6
   xdim,ydim can be any vesa resolution if vidoption = 1
   xdim,ydim must be 320*200 for any other mode.
      (see graphics mode selection in my setup program)

setview(long x1, long y1, long x2, long y2)
   Sets the viewing window to a given rectangle of the screen.
```

```
   Example: For full screen 320*200, call like this: setview(0L,0L,319L,199L);

nextpage();
   This function flips to the next video page.  After a screen is prepared,
   use this function to view the screen.

------------------------------------------------------------------------------
|                            DRAWING FUNCTIONS:                              |
------------------------------------------------------------------------------

drawrooms(long posx, long posy, long posz, short ang, long horiz, short cursectnum)
   This function draws the 3D screen to the current drawing page,
   which is not yet shown.  This way, you can overwrite some things
   over the 3D screen such as a gun.  Be sure to call the drawmasks()
   function soon after you call the drawrooms() function.  To view
   the screen, use the nextpage() function.  The nextpage() function
   should always be called sometime after each draw3dscreen()
   function.

drawmasks();
   This function draws all the sprites and masked walls to the current
   drawing page which is not yet shown.  The reason I have the drawing
   split up into these 2 routines is so you can animate just the
   sprites that are about to be drawn instead of having to animate
   all the sprites on the whole board.  Drawrooms() prepares these
   variables:  spritex[], spritey[], spritepicnum[], thesprite[],
   and spritesortcnt.  Spritesortcnt is the number of sprites about
   to be drawn to the page.  To change the sprite's picnum, simply
   modify the spritepicnum array  If you want to change other parts
   of the sprite structure, then you can use the thesprite array to
   get an index to the actual sprite number.

clearview(long col)
   Clears the current video page to the given color

clearallviews(long col)
   Clears all video pages to the given color

drawmapview (long x, long y, long zoom, short ang)
   Draws the 2-D texturized map at the given position into the viewing window.

rotatesprite (long sx, long sy, long z, short a, short picnum,
              signed char dashade, char dapalnum, char dastat,
              long cx1, long cy1, long cx2, long cy2)
   (sx, sy) is the center of the sprite to draw defined as
       screen coordinates shifted up by 16.
   (z) is the zoom.  Normal zoom is 65536.
       Ex: 131072 is zoomed in 2X and 32768 is zoomed out 2X.
   (a) is the angle (0 is straight up)
   (picnum) is the tile number
   (dashade) is 0 normally but can be any standard shade up to 31 or 63.
   (dapalnum) can be from 0-255.
   if ((dastat&1) == 0) - no transluscence
   if ((dastat&1) != 0) - transluscence
   if ((dastat&2) == 0) - don't scale to setview's viewing window
   if ((dastat&2) != 0) - scale to setview's viewing window (windowx1,etc.)
   if ((dastat&4) == 0) - nuttin' special
   if ((dastat&4) != 0) - y-flip image
   if ((dastat&8) == 0) - clip to startumost/startdmost
   if ((dastat&8) != 0) - don't clip to startumost/startdmost
   if ((dastat&16) == 0) - use Editart center as point passed
   if ((dastat&16) != 0) - force point passed to be top-left corner
   if ((dastat&32) == 0) - nuttin' special
   if ((dastat&32) != 0) - use reverse transluscence
   if ((dastat&64) == 0) - masked drawing (check 255's) (slower)
   if ((dastat&64) != 0) - draw everything (don't check 255's) (faster)
   if ((dastat&128) == 0) - nuttin' special
   if ((dastat&128) != 0) - automatically draw to all video pages

   Note:  As a special case, if both ((dastat&2) != 0) and ((dastat&8) != 0)
      then rotatesprite will scale to the full screen (0,0,xdim-1,ydim-1)
      rather than setview's viewing window. (windowx1,windowy1,etc.)  This
      case is useful for status bars, etc.

      Ex: rotatesprite(160L<<16,100L<<16,65536,totalclock<<4,
```

```
                      DEMOSIGN,2,50L,50L,270L,150L);
          This example will draw the DEMOSIGN tile in the center of the
          screen and rotate about once per second.  The sprite will only
          get drawn inside the rectangle from (50,50) to (270,150)

drawline256(long x1, long y1, long x2, long y2, char col)
   Draws a solid line from (x1,y1) to (x2,y2) with color (col)
   For this function, screen coordinates are all shifted up 16 for precision.

printext256(long xpos, long ypos, short col, short backcol,
            char *message, char fontsize)
   Draws a text message to the screen.
   (xpos,ypos) - position of top left corner
   col - color of text
   backcol - background color, if -1, then background is transparent
   message - text message
   fontsize - 0 - 8*8 font
              1 - 4*6 font

---------------------------------------------------------------------------
|                        MOVEMENT COLLISION FUNCTIONS:                    |
---------------------------------------------------------------------------

clipmove(long *x, long *y, long *z, short *sectnum, long xvect, long yvect,
         long walldist, long ceildist, long flordist, unsigned long cliptype)
      Moves any object (x, y, z) in any direction at any velocity and will
   make sure the object will stay a certain distance from walls (walldist)
       Pass the pointers of the starting position (x, y, z).  Then
   pass the starting position's sector number as a pointer also.
   Also these values will be modified accordingly.  Pass the
   direction and velocity by using a vector (xvect, yvect).
   If you don't fully understand these equations, please call me.
         xvect = velocity * cos(angle)
         yvect = velocity * sin(angle)
      Walldist tells how close the object can get to a wall.  I use
    128L as my default.  If you increase walldist all of a sudden
    for a certain object, the object might leak through a wall, so
    don't do that!
       Cliptype is a mask that tells whether the object should be clipped
    to or not.  The lower 16 bits are anded with wall[].cstat and the higher
    16 bits are anded with sprite[].cstat.

     Clipmove can either return 0 (touched nothing)
                                32768+wallnum (wall first touched)
                                49152+spritenum (sprite first touched)

pushmove (long *x, long *y, long *z, short *sectnum,
          long walldist, long ceildist, long flordist, unsigned long cliptype)
      This function makes sure a player or monster (defined by x, y, z, sectnum)
   is not too close to a wall.  If it is, then it attempts to push it away.
   If after 256 tries, it is unable to push it away, it returns -1, in which
   case the thing should gib.

getzrange(long x, long y, long z, short sectnum,
                  long *ceilz, long *ceilhit,
                  long *florz, long *florhit,
                  long walldist, unsigned long cliptype)

      Use this in conjunction with clipmove.  This function will keep the
   player from falling off cliffs when you're too close to the edge.  This
   function finds the highest and lowest z coordinates that your clipping
   BOX can get to.  It must search for all sectors (and sprites) that go
   into your clipping box.  This method is better than using
   sector[cursectnum].ceilingz and sector[cursectnum].floorz because this
   searches the whole clipping box for objects, not just 1 point.
      Pass x, y, z, sector normally.  Walldist can be 128.  Cliptype is
   defined the same way as it is for clipmove.  This function returns the
   z extents in ceilz and florz. It will return the object hit in ceilhit
   and florhit.  Ceilhit and florhit will also be either:
                                16384+sector (sector first touched) or
                                49152+spritenum (sprite first touched)

hitscan(long xstart, long ystart, long zstart, short startsectnum,
        long vectorx, long vectory, long vectorz,
        short *hitsect, short *hitwall, short *hitsprite,
```

```
582          long *hitx, long *hity, long *hitz);
583
584     Pass the starting 3D position:
585          (xstart, ystart, zstart, startsectnum)
586     Then pass the 3D angle to shoot (defined as a 3D vector):
587          (vectorx, vectory, vectorz)
588     Then set up the return values for the object hit:
589          (hitsect, hitwall, hitsprite)
590     and the exact 3D point where the ray hits:
591          (hitx, hity, hitz)
592
593     How to determine what was hit:
594        * Hitsect is always equal to the sector that was hit (always >= 0).
595
596        * If the ray hits a sprite then:
597             hitsect = thesectornumber
598             hitsprite = thespritenumber
599             hitwall = -1
600
601         * If the ray hits a wall then:
602             hitsect = thesectornumber
603             hitsprite = -1
604             hitwall = thewallnumber
605
606         * If the ray hits the ceiling of a sector then:
607             hitsect = thesectornumber
608             hitsprite = -1
609             hitwall = -1
610             vectorz < 0
611             (If vectorz < 0 then you're shooting upward which means
612                that you couldn't have hit a floor)
613
614         * If the ray hits the floor of a sector then:
615             hitsect = thesectornumber
616             hitsprite = -1
617             hitwall = -1
618             vectorz > 0
619             (If vectorz > 0 then you're shooting downard which means
620                that you couldn't have hit a ceiling)
621
622  neartag(long x, long y, long z, short sectnum, short ang,  //Starting position & angle
623          short *neartagsector,   //Returns near sector if sector[].tag != 0
624          short *neartagwall,     //Returns near wall if wall[].tag != 0
625          short *neartagsprite,   //Returns near sprite if sprite[].tag != 0
626          long *neartaghitdist,   //Returns actual distance to object (scale: 1024=largest grid size)
627          long neartagrange,      //Choose maximum distance to scan (scale: 1024=largest grid size)
628          char tagsearch)         //1-lotag only, 2-hitag only, 3-lotag&hitag
629        Neartag works sort of like hitscan, but is optimized to
630     scan only close objects and scan only objects with
631     tags != 0.  Neartag is perfect for the first line of your space bar code.
632     It will tell you what door you want to open or what switch you want to
633     flip.
634
635  cansee(long x1, long y1, long z1, short sectnum1,
636         long x2, long y2, long z2, short sectnum2);  returns 0 or 1
637     This function determines whether or not two 3D points can "see" each
638     other or not.  All you do is pass it the coordinates of a 3D line defined
639     by two 3D points (with their respective sectors)  The function will return
640     a 1 if the points can see each other or a 0 if there is something blocking
641     the two points from seeing each other.  This is how I determine whether a
642     monster can see you or not. Try playing DOOM1.DAT to fully enjoy this
643     great function!
644
645  updatesector(long x, long y, &sectnum);
646     This function updates the sector number according to the x and y values
647     passed to it.  Be careful when you use this function with sprites because
648     remember that the sprite's sector number should not be modified directly.
649     If you want to update a sprite's sector, I recomment using the setsprite
650     function described below.
651
652  inside(long x, long y, short sectnum);
653     Tests to see whether the overhead point (x, y) is inside sector (sectnum)
654     Returns either 0 or 1, where 1 means it is inside, and 0 means it is not.
655
656  clipinsidebox(long x, long y, short wallnum, long walldist)
```

```
   Returns TRUE only if the given line (wallnum) intersects the square with
   center (x,y) and radius, walldist.

dragpoint(short wallnum, long newx, long newy);
   This function will drag a point in the exact same way a point is dragged
   in 2D EDIT MODE using the left mouse button.  Simply pass it which wall
   to drag and then pass the new x and y coordinates for that point.
   Please use this function because if you don't and try to drag points
   yourself, I can guarantee that it won't work as well as mine and you
   will get confused.  Note:  Every wall of course has 2 points.  When you
   pass a wall number to this function, you are actually passing 1 point,
   the left side of the wall (given that you are in the sector of that wall)
   Got it?

------------------------------------------------------------------------------
|                          MATH HELPER FUNCTIONS:                            |
------------------------------------------------------------------------------

krand()
   Random number function - returns numbers from 0-65535

ksqrt(long num)
   Returns the integer square root of the number.

getangle(long xvect, long yvect)
   Gets the angle of a vector (xvect,yvect)
   These are 2048 possible angles starting from the right, going clockwise

rotatepoint(long xpivot, long ypivot, long x, long y,
            short daang, long *x2, long *y2);
      This function is a very convenient and fast math helper function.
   Rotate points easily with this function without having to juggle your
   cosines and sines.  Simply pass it:

       Input:   1. Pivot point     (xpivot,ypivot)
                2. Original point  (x,y)
                3. Angle to rotate (0 = nothing, 512 = 90ø CW, etc.)
       Output:  4. Rotated point   (*x2,*y2)

lastwall(short point);
      Use this function as a reverse function of wall[].point2.  In order
   to save memory, my walls are only on a single linked list.

nextsectorneighborz(short sectnum, long thez, short topbottom, short direction)
   This function is used to tell where elevators should stop.  It searches
   nearby sectors for the next closest ceilingz or floorz it should stop at.
   sectnum - elevator sector
   thez - current z to start search from
   topbottom - search ceilingz's/floorz's only
   direction - search upwards/downwards

getceilzofslope(short sectnum, long x, long y)
getflorzofslope(short sectnum, long x, long y)
getzsofslope(short sectnum, long x, long y, long *ceilz, long *florz)
   These 3 functions get the height of a ceiling and/or floor in a sector
   at any (x,y) location.  Use getzsofslope only if you need both the ceiling
   and floor.

alignceilslope(short sectnum, long x, long y, long z)
alignflorslope(short sectnum, long x, long y, long z)
   Given a sector and assuming it's first wall is the pivot wall of the slope,
   this function makes the slope pass through the x,y,z point.  One use of
   this function is used for sin-wave floors.

------------------------------------------------------------------------------
|                            SPRITE FUNCTIONS:                               |
------------------------------------------------------------------------------

insertsprite(short sectnum, short statnum);   //returns (short)spritenum;
   Whenever you insert a sprite, you must pass it the sector
   number, and a status number (statnum).  The status number can be any
   number from 0 to MAXSTATUS-1.  Insertsprite works like a memory
   allocation function and returns the sprite number.

deletesprite(short spritenum);
```

```
   Deletes the sprite.

changespritesect(short spritenum, short newsectnum);
   Changes the sector of sprite (spritenum) to the
   newsector (newsectnum).  This function may become
   internal to the engine in the movesprite function.  But
   this function is necessary since all the sectors have
   their own doubly-linked lists of sprites.

changespritestat(short spritenum, short newstatnum);
   Changes the status of sprite (spritenum) to status
   (newstatus).  Newstatus can be any number from 0 to MAXSTATUS-1.
   You can use this function to put a monster on a list of active sprites
   when it first sees you.

setsprite(short spritenum, long newx, long newy, long newz);
      This function simply sets the sprite's position to a specified
   coordinate (newx, newy, newz) without any checking to see
   whether the position is valid or not.  You could directly
   modify the sprite[].x, sprite[].y, and sprite[].z values, but
   if you use my function, the sprite is guaranteed to be in the
   right sector.

------------------------------------------------------------------------
|                           CACHE FUNCTIONS:                           |
------------------------------------------------------------------------
initcache(long dacachestart, long dacachesize)
   First allocate a really large buffer (as large as possible), then pass off
   the memory bufer the initcache
   dacachestart: 32-bit offset in memory of start of cache
   dacachesize: number of bytes that were allocated for the cache to use

allocache (long *bufptr, long bufsiz, char *lockptr)
   *bufptr = pointer to 4-byte pointer to buffer.  This
      allows allocache to remove previously allocated things
      from the cache safely by setting the 4-byte pointer to 0.
   bufsiz = number of bytes to allocate
   *lockptr = pointer to locking char which tells whether
      the region can be removed or not.  If *lockptr = 0 then
      the region is not locked else its locked.

------------------------------------------------------------------------
|                         GROUP FILE FUNCTIONS:                        |
------------------------------------------------------------------------
initgroupfile(char *filename)
   Tells the engine what the group file name is.
   You should call this before any of the following group file functions.
uninitgroupfile()
   Frees buffers.  You should call this once at the end of the program
   before quitting to dos.

kopen4load(char *filename, char searchfirst)
   Open a file.  First tries to open a stand alone file.  Then searches for
   it in the group file.  If searchfirst is nonzero, it will check the group
   file only.

kread(long handle, void *buffer, long leng)
klseek(long handle, long offset, long whence)
kfilelength(long handle)
kclose(long handle)
   These 4 functions simply shadow the dos file functions - they
   can do file I/O on the group file in addition to stand-along files.

------------------------------------------------------------------------
|                         COMMUNICATIONS FUNCTIONS:                    |
------------------------------------------------------------------------
   Much of the following code is to keep compatibity with older network code:

initmultiplayers(char damultioption, char dacomrateoption, char dapriority)
   The parameters are ignored - just pass 3 0's
uninitmultiplayers()    Does nothing

sendpacket(long other, char *bufptr, long messleng)
   other - who to send the packet to
   bufptr - pointer to message to send
```

```
   messleng - length of message
short getpacket (short *other, char *bufptr)
   returns the number of bytes of the packet received, 0 if no packet
   other - who the packet was received from
   bufptr - pointer to message that was received

sendlogon()             Does nothing
sendlogoff()
   Sends a packet to everyone else where the
   first byte is 255, and the
   second byte is myconnectindex

getoutputcirclesize()  Does nothing - just a stub function, returns 0
setsocket(short newsocket) Does nothing

flushpackets()
   Clears all packet buffers
genericmultifunction(long other, char *bufptr, long messleng, long command)
   Passes a buffer to the commit driver.  This command provides a gateway
   for game programmer to access COMMIT directly.

------------------------------------------------------------------------------
|                            PALETTE FUNCTIONS:                              |
------------------------------------------------------------------------------
VBE_setPalette(long start, long num, char *palettebuffer)
VBE_getPalette(long start, long num, char *palettebuffer)
   Set (num) palette palette entries starting at (start)
   palette entries are in a 4-byte format in this order:
      0: Blue (0-63)
      1: Green (0-63)
      2: Red (0-63)
      3: Reserved

makepalookup(long palnum, char *remapbuf,
             signed char r, signed char g, signed char b,
             char dastat)
   This function allows different shirt colors for sprites.  First prepare
   remapbuf, which is a 256 byte buffer of chars which the colors to remap.
   Palnum can be anywhere from 1-15.  Since 0 is where the normal palette is
   stored, it is a bad idea to call this function with palnum=0.
   In BUILD.H notice I added a new variable, spritepal[MAXSPRITES].
   Usually the value of this is 0 for the default palette.  But if you
   change it to the palnum in the code between drawrooms() and drawmasks
   then the sprite will be drawn with that remapped palette.  The last 3
   parameters are the color that the palette fades to as you get further
   away.  This color is normally black (0,0,0).  White would be (63,63,63).
   if ((dastat&1) == 0) then makepalookup will allocate & deallocate
   the memory block for use but will not waste the time creating a palookup
   table (assuming you will create one yourself)

setbrightness(char gammalevel, char *dapal)
   Use this function to adjust for gamma correction.
   Gammalevel - ranges from 0-15, 0 is darkest, 15 brightest. Default: 0
   dapal: standard VGA palette (768 bytes)

------------------------------------------------------------------------------
|                     This document brought to you by:                       |
|                                                                            |
|          @@@@@@        @@@@@@  @@@@@@@@@@@@@@@@@@  @@@@@@@@         @@@@@@ |
|          @@@@@@       @@@@@@   @@@@@@@@@@@@@@@@@@  @@@@@@@@@        @@@@@@ |
|          @@@@@@     @@@@@@@    @@@@@@@@@@@@@@@@@@  @@@@@@@@@@       @@@@@@ |
|          @@@@@@    @@@@@@@     @@@@@@              @@@@@@@@@@@      @@@@@@ |
|          @@@@@@  @@@@@@@       @@@@@@              @@@@@@@@@@@@     @@@@@@ |
|          @@@@@@ @@@@@@@        @@@@@@              @@@@@@@@@@@@@    @@@@@@ |
|          @@@@@@@@@@@@          @@@@@@@@@@@@@@@@@@  @@@@@@ @@@@@@@   @@@@@@ |
| @@@@@@@  @@@@@@@@@@@@          @@@@@@@@@@@@@@@@@@  @@@@@@  @@@@@@@  @@@@@@ |
|          @@@@@@@@@@@@          @@@@@@@@@@@@@@@@@@  @@@@@@   @@@@@@@ @@@@@@ |
|          @@@@@@ @@@@@@@        @@@@@@              @@@@@@    @@@@@@@@@@@@ |
|          @@@@@@  @@@@@@@       @@@@@@              @@@@@@     @@@@@@@@@@@ |
|          @@@@@@    @@@@@@@     @@@@@@              @@@@@@      @@@@@@@@@@ |
|          @@@@@@     @@@@@@@    @@@@@@@@@@@@@@@@@@  @@@@@@       @@@@@@@@@ |
|          @@@@@@       @@@@@@   @@@@@@@@@@@@@@@@@@  @@@@@@        @@@@@@@@ |
|          @@@@@@        @@@@@@  @@@@@@@@@@@@@@@@@@  @@@@@@         @@@@@@@ |
|                                                                            |
|                  Ken Silverman of East Greenwich, RI USA                   |
```

```
882 ------------------------------------------------------------------------
```