FzxNGN Documentation

August 11,2023

Overview: fzxNGN is a 2D physics engine library that was ported to QB64 from the Impulse engine written by Randy Gaul https://github.com/RandyGaul/ImpulseEngine.

Features:

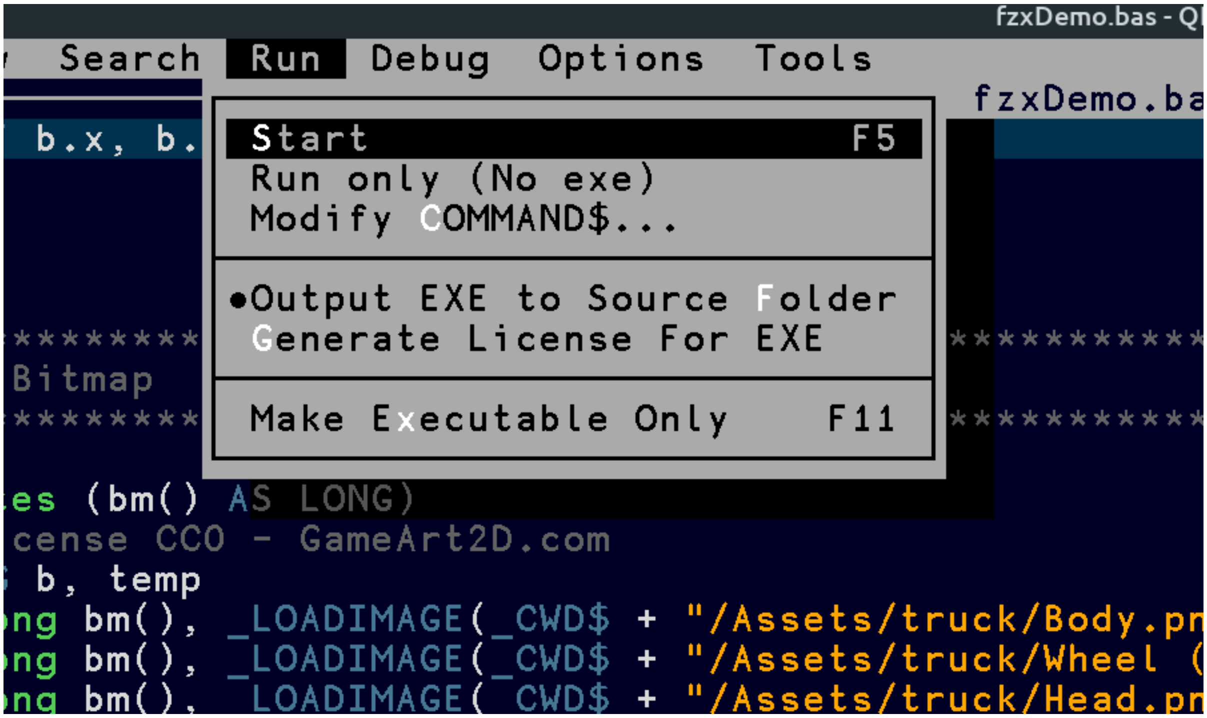
- Rigid body simulation
- Circle and polygon primitives, concave objects not supported yet
- Joint simulation
- Camera Library
- Input Library
- Finite State Machine helper functions
- Perlin noise library functions
- XML parsing (WIP)
- LERP functions
- FPS helper functions
- Tons of vector and matrix math functions.
- Units are arbitrary, Its up to the user to decide which units of length to use.

What it is and what it is not:

- A project to help QB64 programmers such as myself make more interesting demos and mini-games.
- It is a collection of subroutines and functions that I've made every attempt to generalize. They can be used outside the simulation.
- Not a core engine for the next AAA game.
- Not for serious engineering use.

Compiling examples:

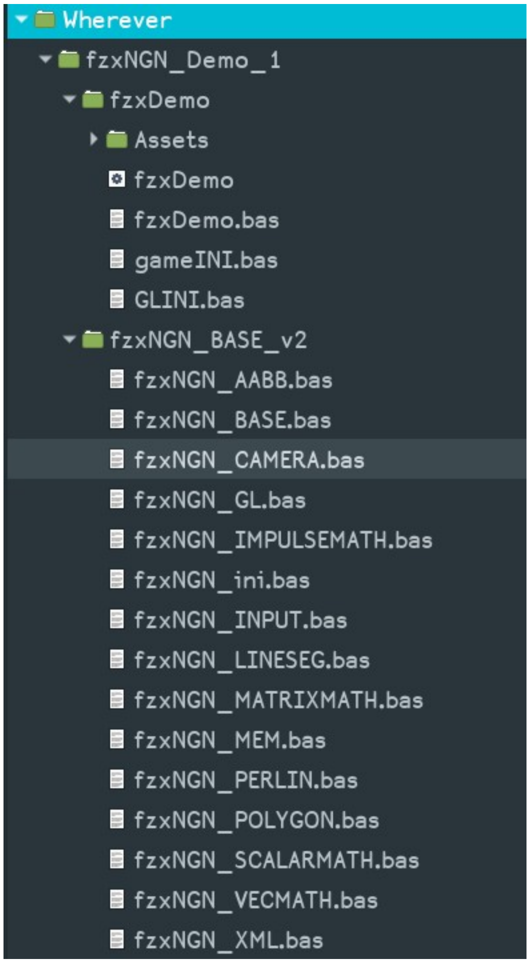
- Compiling should be straightforward. Load the example file and hit F5 or Start.
- Make sure the "Output EXE to Source Folder" is selected. This will ensure that the file structure is not broken.



```
 b.x, b.
*********                                        ***********
Bitmap
*********                                        ***********
es (bm() AS LONG)
cense CC0 - GameArt2D.com
 b, temp
ng bm(), _LOADIMAGE(_CWD$ + "/Assets/truck/Body.pn
ng bm(), _LOADIMAGE(_CWD$ + "/Assets/truck/Wheel (
ng bm(), _LOADIMAGE(_CWD$ + "/Assets/truck/Head.pn
```

FzxNGN File structure:

- The filenames may vary but the same general file structure should be maintained.



```
Wherever
  fzxNGN_Demo_1
    fzxDemo
      Assets
      fzxDemo
      fzxDemo.bas
      gameINI.bas
      GLINI.bas
    fzxNGN_BASE_v2
      fzxNGN_AABB.bas
      fzxNGN_BASE.bas
      fzxNGN_CAMERA.bas
      fzxNGN_GL.bas
      fzxNGN_IMPULSEMATH.bas
      fzxNGN_ini.bas
      fzxNGN_INPUT.bas
      fzxNGN_LINESEG.bas
      fzxNGN_MATRIXMATH.bas
      fzxNGN_MEM.bas
      fzxNGN_PERLIN.bas
      fzxNGN_POLYGON.bas
      fzxNGN_SCALARMATH.bas
      fzxNGN_VECMATH.bas
      fzxNGN_XML.bas
```

FzxNGN Globals (fzxNGN_ini.bas):

These variables that are global to the engine. Double underscores are used so that there little chance of interfering with the project it is being used with.

__fzxBody() : Contains all the data pertaining to each rigid body.
__fzxJoints() : Contains all the data pertaining to the joints.
__fzxHits() : Collision information
__fzxCamera : Camera data
__fzxWorld : World data
__fzxFPSCount : FPS counting
__fzxInputDevice : Mouse and keyboard
__fzxSettings : Generalized settings. (Currently only mouse double click timing)
__fzxStats : Contains general statistics about the simulation (fps, body and joint counts)

A Bare Bone Implementation:

This is a simple example of what it takes for a small Implementation of the engine.

- **Initialization include**
    - '$include:'..\fzxNGN_BASE_v2\fzxNGN_ini.bas'
    - sets up the types(UDT), global variables, and constants
- Call to **Build scene.**
- **Main Loop**
    - Call to **Animate scene**
        - Player interaction happens here
    - Call to fzxNGN to calculate the next step
        - **fzxImpulseStep**
    - Call to **Render scene**
- **Include core code**
    - '$include:'..\fzxNGN_BASE_v2\fzxNGN_BASE.bas'
    - All the core functionality is contained here
- **Build Scene**
    - Initial setup of the camera.
    - Set some limits on the world.
    - Setup gravity.
    - Add your bodies to the simulation.
- **Animate Scene**
    - This where you will interact with the bodies.
- **Render Scene**
    - Draw the bodies in the scene.

Code Example: fzxNGNBareBones.bas

I think the best way to learn the engine is by example. So the example is a fairly stripped down version of a working prototype. It can be scaled down further depending on its being used for. For instance, if you are not using gravity or only using circles, then those parts can be left out.

```
'$DYNAMIC
'$include:'..\fzxNGN_BASE_v2\fzxNGN_ini.bas'
SCREEN _NEWIMAGE(1024, 768, 32)
` Iterations and deltaTime are now global values
` These are the default Values
`__fzxWorld.deltaTime = 1 / 60
`__fzxWorld.iterations = 100
~~`DIM AS LONG iterations: iterations = 2~~
~~`DIM SHARED AS DOUBLE dt: dt = 1 / 60~~
```

- **'$DYNAMIC**
    - Some of the Global variable are dynamic and this must be declared
- **'$include:'..\fzxNGN_BASE_v2\fzxNGN_ini.bas'**
    - Include the types, constants, and the global variables
- **SCREEN _NEWIMAGE(1024, 768, 32)**
    - Screen size can be whatever you desire, because the camera functionality will allow the user to view any portion of the world.
- ~~**DIM AS LONG iterations: iterations = 2**~~
    - No longer necessary. Defaults are set in fzxNGN_ini.bas to 100 iterations
    - If you wish to change it, the new global variable is '__fzxWorld.iterations'
- ~~**DIM SHARED AS DOUBLE dt: dt = 1 / 60**~~
    - No longer necessary. Defaults are set in fzxNGN_ini.bas to 1/60 seconds
    - If you wish to change it, the new global variable is '__fzxWorld.deltaTime'

```
buildScene
DO
  CLS: LOCATE 1: PRINT "Click the mouse on the play field to spawn an object"
  fzxHandleInputDevice
  animatescene
  ` No longer need to pass ‘dt’ and ‘iterations’ as they are now global variables
  ~~fzxImpulseStep dt, iterations~~
  fzxImpulseStep
  renderBodies
_DISPLAY
LOOP UNTIL INKEY$ = CHR$(27)
SYSTEM
'$include:'..\fzxNGN_BASE_v2\fzxNGN_BASE.bas'
```

- **buildScene**
    - Call to the subroutine that will setup the world and build your initial scene
- **DO**
    - This is the Main loop of the program
- **CLS: LOCATE 1: PRINT "Click the mouse on the play field to spawn an object"**
- **fzxHandleInputDevice**
    - A subroutine that will manage some extra routines for the mouse and keyboard
    - Later in the program, **“__fzxInputDevice.mouse.b1.NegEdge”** flag will be queried, It will return a true if the mouse button has been released.
- **Animatescene**
    - This is a call to the subroutine that handles all of the frame to frame activities.
    - User interaction
    - Level animation and logic
- ~~**fzxImpulseStep dt, iterations**~~
    - No longer required to pass ‘dt’ and ‘iterations’ as they are global variables
- **fzxImpulseStep**
    - Run the simulation one step
- **renderBodies**
    - Draw the level
    - The renderer is up to the user to implement depending on their need.
- **_DISPLAY**
- **LOOP UNTIL INKEY$ = CHR$(27)**
- **SYSTEM**
- **'$include:'..\fzxNGN_BASE_v2\fzxNGN_BASE.bas'**
    - Provides all of the fzxNGN functionality

```
SUB animateScene
  DIM AS LONG temp
  ' Create a object on mouse click
  IF __fzxInputDevice.mouse.b1.NegEdge THEN
    ' Drop a ball or a polygon, flip a coin
    IF RND > .5 THEN
      temp = fzxCreateCircleBodyEx("b" + _TRIM$(STR$(RND * 1000000000)), 10 + RND * 10)
    ELSE
      temp = fzxCreatePolyBodyEx("b" + _TRIM$(STR$(RND * 1000000000)), 10 + RND * 10, 10 + RND * 10, 3
+ INT(RND * 5))
    END IF
    ' Set the bodies parameters
    ' Put the body where the mouse is on the screen
    fzxSetBody cFZX_PARAMETER_POSITION, temp, __fzxInputDevice.mouse.worldPosition.x,
__fzxInputDevice.mouse.worldPosition.y
    ' Give it the mouse's velocity, so you can throw it
    fzxSetBody cFZX_PARAMETER_VELOCITY, temp, __fzxInputDevice.mouse.velocity.x * 10,
__fzxInputDevice.mouse.velocity.y * 10
    ' Change its orientation or angle
    fzxSetBody cFZX_PARAMETER_ORIENT, temp, _D2R(RND * 360), 0
    ' Set the bouncyness
    fzxSetBody cFZX_PARAMETER_RESTITUTION, temp, .8, 0 ' Bounce
    ' Set the friction values of the body
    fzxSetBody cFZX_PARAMETER_STATICFRICTION, temp, .9, 0
    fzxSetBody cFZX_PARAMETER_DYNAMICFRICTION, temp, .25, 0
    ' Bodies wont live forever
    fzxSetBody cFZX_PARAMETER_LIFETIME, temp, RND * 20 + 10, 0
    ' Set the color
    fzxSetBody cFZX_PARAMETER_COLOR, temp, _RGB32(RND * 200, RND * 200, RND * 200), 0
END SUB
```

- **SUB animateScene**
- **DIM AS LONG temp**
  - temp is a variable we will use, because after the body is setup, we don’t care about it anymore.
- **IF __fzxInputDevice.mouse.b1.NegEdge THEN**
  - This condition will be true when the user releases the mouse button
- **IF RND > .5 THEN**
  - Flip a coin
- **temp = fzxCreateCircleBodyEx("b" + _TRIM$(STR$(RND * 1000000000)), 10)**
  - Create a circle body with some unique name with a radius of 10.
- **ELSE**
- **temp = fzxCreateBoxBodyEx("b" + _TRIM$(STR$(RND * 1000000000)), 10, 10)**
  - Create a box body with as unique name with a dimension of 10 by 10.
- **END IF**
- **fzxSetBody cFZX_PARAMETER_POSITION, temp, __fzxInputDevice.mouse.worldPosition.x, __fzxInputDevice.mouse.worldPosition.y**
  - Move the previously created body to the position in the game world pointed to by the mouse.
  - “**__fzxInputDevice.mouse.worldPosition**” is not the same as screen position, it calculated based on camera position and camera zoom.
- **fzxSetBody cFZX_PARAMETER_VELOCITY, temp, __fzxInputDevice.mouse.velocity.x, __fzxInputDevice.mouse.velocity.y**
  - Give the body some velocity based on how fast the mouse was moving when the user released the mouse button.

- **fzxSetBody cFZX_PARAMETER_ORIENT, temp, _D2R(RND * 360), 0**
    - Give the body an arbitrary angle
- **fzxSetBody cFZX_PARAMETER_RESTITUTION, temp, .5, 0 ' Bounce**
    - Set the bounce of the body.
    - Value should be between 0 to 1. Zero is no bounce at all and one is a very hyper super ball.
- **fzxSetBody cFZX_PARAMETER_STATICFRICTION, temp, .1, 0**
- **fzxSetBody cFZX_PARAMETER_DYNAMICFRICTION, temp, .85, 0**
    - Static and dynamic(kinetic) friction can be best described in this article on wikipedia.
    - https://en.wikipedia.org/wiki/Friction
- **fzxSetBody cFZX_PARAMETER_LIFETIME, temp, RND * 20 + 10, 0**
    - Delete the body after a random number of seconds.
- **END IF**
- **END SUB**

```
SUB buildScene
  DIM AS LONG temp
  __fzxCamera.zoom = 1
  fzxCalculateFOV
  fzxVector2DSet __fzxWorld.minusLimit, -200000, -200000
  fzxVector2DSet __fzxWorld.plusLimit, 200000, 200000
  fzxVector2DSet __fzxWorld.spawn, 0, 0
  fzxVector2DSet __fzxWorld.gravity, 0.0, 10.0
  fzxVector2DSet __fzxCamera.position, __fzxWorld.spawn.x, __fzxWorld.spawn.y - 300
  DIM o AS tFZX_VECTOR2d
  fzxVector2DMultiplyScalarND o, __fzxWorld.gravity, dt
  __fzxWorld.resting = fzxVector2DLengthSq(o) + cFZX_EPSILON
  temp = fzxCreateBoxBodyEx("floor", 800, 10)
  fzxSetBody cFZX_PARAMETER_POSITION, temp, __fzxWorld.spawn.x, __fzxWorld.spawn.y
  fzxSetBody cFZX_PARAMETER_STATIC, temp, 0, 0
END SUB
```

- **SUB buildScene**
- **DIM AS LONG temp**
    - We will be using this to create static objects that we wont need later
- **__fzxCamera.zoom = 1**
- **fzxCalculateFOV**
    - Set up Camera zoom. Note “fzxCalculateFOV” needs to be called every time the zoom is changed.
- **fzxVector2DSet __fzxWorld.minusLimit, -200000, -200000**
- **fzxVector2DSet __fzxWorld.plusLimit, 200000, 200000**
    - Set world limits. Objects outside of this are will be deleted.
- **fzxVector2DSet __fzxWorld.spawn, 0, 0**
    - This is a position that can be used how the user likes, but I use it to set the starting position of everything
- **fzxVector2DSet __fzxWorld.gravity, 0.0, 10.0**
    - Set the gravity vector
- **fzxVector2DSet __fzxCamera.position, __fzxWorld.spawn.x, __fzxWorld.spawn.y - 300**
    - Set the camera position
- **DIM o AS tFZX_VECTOR2d**
- **fzxVector2DMultiplyScalarND o, __fzxWorld.gravity, dt**
- **__fzxWorld.resting = fzxVector2DLengthSq(o) + cFZX_EPSILON**
    - Setting up some simulation related values
    - ToDo: Do this automatically
    - If your not using gravity then this can be omitted
- **temp = fzxCreateBoxBodyEx("floor", 800, 10)**
    - Create a floor body
- **fzxSetBody cFZX_PARAMETER_POSITION, temp, __fzxWorld.spawn.x, __fzxWorld.spawn.y**
    - Set it at the spawn point
- **fzxSetBody cFZX_PARAMETER_STATIC, temp, 0, 0**
    - Set it as a static object
- **END SUB**

```
SUB renderBodies STATIC
  DIM i AS LONG
  DIM AS tFZX_VECTOR2d scSize, scMid, scUpperLeft, camUpperLeft, aabbUpperLeft, aabbSize, aabbHalfSize
  DIM AS LONG ub: ub = UBOUND(__fzxBody)

  fzxVector2DSet aabbSize, 40000, 40000
  fzxVector2DSet aabbHalfSize, aabbSize.x / 2, aabbSize.y / 2

  fzxVector2DSet scUpperLeft, 0, 0
  fzxVector2DSet scSize, _WIDTH, _HEIGHT

  fzxVector2DDivideScalarND scMid, scSize, 2
  fzxVector2DSubVectorND camUpperLeft, __fzxCamera.position, scMid

  i = 0: DO WHILE i < ub
    IF __fzxBody(i).enable THEN
      'fzxAABB to cut down on rendering objects out of camera view
      fzxVector2DSubVectorND aabbUpperLeft, __fzxBody(i).fzx.position, aabbHalfSize
      IF fzxAABBOverlap(camUpperLeft.x, camUpperLeft.y, scSize.x, scSize.y, aabbUpperLeft.x,
aabbUpperLeft.y, aabbSize.x, aabbSize.y) THEN
        IF __fzxBody(i).shape.ty = cFZX_SHAPE_CIRCLE THEN
          renderWireFrameCircle i, _RGB32(0, 255, 0)
        ELSE IF __fzxBody(i).shape.ty = cFZX_SHAPE_POLYGON THEN
            renderWireFramePoly i
          END IF
        END IF
      END IF
    END IF
    i = i + 1
  LOOP
END SUB
```

- **SUB renderBodies STATIC**
- **DIM i AS LONG**
- **DIM AS tFZX_VECTOR2d scSize, scMid, scUpperLeft, camUpperLeft, aabbUpperLeft, aabbSize, aabbHalfSize**
- **DIM AS LONG ub: ub = UBOUND(__fzxBody)**
  - Assign **ub** to the size of the body array.
- **fzxVector2DSet aabbSize, 40000, 40000**
  - This size is set rather large, if speed of rendering is an issue then you might tighten it to closer to the play area that is visible on the screen.
- **fzxVector2DSet aabbHalfSize, aabbSize.x / 2, aabbSize.y / 2**
- **fzxVector2DSet scUpperLeft, 0, 0**
- **fzxVector2DSet scSize, _WIDTH, _HEIGHT**
- **fzxVector2DDivideScalarND scMid, scSize, 2**
- **fzxVector2DSubVectorND camUpperLeft, __fzxCamera.position, scMid**
  - If your play area is limited in size then AABB can be omitted, It is here to demonstrate that you avoid rendering object that are not on the screen
- **i = 0: DO WHILE i < ub**
  - Loop through all bodies in the array
  - I use **DO..LOOP**s a lot, because I understood that they are faster than **FOR..NEXT**
- **IF __fzxBody(i).enable THEN**
  - Don’t bother rendering objects that are not enabled.
  - Disabling an object can be temporary, in case you need it later.
  - It will not be collided with nor have any forces applied to it.
- **'fzxAABB to cut down on rendering objects out of camera view**

- **fzxVector2DSubVectorND aabbUpperLeft, __fzxBody(i).fzx.position, aabbHalfSize**
  - Calculate visible play area
- **IF fzxAABBOverlap(camUpperLeft.x, camUpperLeft.y, scSize.x, scSize.y, aabbUpperLeft.x, aabbUpperLeft.y, aabbSize.x, aabbSize.y) THEN**
  - Cull out objects that are not visible in the AABB
- **IF __fzxBody(i).shape.ty = cFZX_SHAPE_CIRCLE THEN**
  - **'__fzxBody(i).shape.ty'** contains the shape of the body
  - Currently only two shapes are available
    - **'cFZX_SHAPE_CIRCLE'**
    - **'cFZX_SHAPE_POLYGON'**
- **renderWireFrameCircle i, _RGB32(0, 255, 0)**
- **ELSE IF __fzxBody(i).shape.ty = cFZX_SHAPE_POLYGON THEN**
- **renderWireFramePoly i**
- **END IF**
- **END IF**
- **END IF**
- **END IF**
- **i = i + 1**
- **LOOP**
- **END SUB**

```
SUB renderWireFrameCircle (index AS LONG, c AS LONG)
  DIM AS tFZX_VECTOR2d o1, o2
  fzxWorldToCameraEx __fzxBody(index).fzx.position, o1
  CIRCLE (o1.x, o1.y), __fzxBody(index).shape.radius * __fzxCamera.zoom, c
  o2.x = o1.x + (__fzxBody(index).shape.radius * __fzxCamera.zoom) * COS(__fzxBody(index).fzx.orient)
  o2.y = o1.y + (__fzxBody(index).shape.radius * __fzxCamera.zoom) * SIN(__fzxBody(index).fzx.orient)
  LINE (o1.x, o1.y)-(o2.x, o2.y), c
END SUB
```

- **SUB renderWireFrameCircle (index AS LONG, c AS LONG)**
- **DIM AS tFZX_VECTOR2d o1, o2**
- **fzxWorldToCameraEx __fzxBody(index).fzx.position, o1**
  - Convert **'o1'** to screen coordinates
- **CIRCLE (o1.x, o1.y), __fzxBody(index).shape.radius * __fzxCamera.zoom, c**
- **o2.x = o1.x + (__fzxBody(index).shape.radius * __fzxCamera.zoom) * COS(__fzxBody(index).fzx.orient)**
- **o2.y = o1.y + (__fzxBody(index).shape.radius * __fzxCamera.zoom) * SIN(__fzxBody(index).fzx.orient)**
  - The previous two line calculate the line that extends from the center of the circle to the radius
  - Its a visual aid to see the rotation of the circle.
- **LINE (o1.x, o1.y)-(o2.x, o2.y), c**
- **END SUB**

```
SUB renderWireFramePoly (index AS LONG) STATIC
  DIM AS LONG polyCount, i
  polyCount = fzxGetBodyD(CFZX_PARAMETER_POLYCOUNT, index, 0)
  DIM AS tFZX_VECTOR2d vert1, vert2
  i = 0: DO WHILE i <= polyCount
    fzxGetBodyVert index, i, vert1
    fzxGetBodyVert index, fzxArrayNextIndex(i, polyCount), vert2

    fzxWorldToCamera index, vert1
    fzxWorldToCamera index, vert2
    LINE (vert1.x, vert1.y)-(vert2.x, vert2.y), _RGB(0, 255, 0)
  i = i + 1: LOOP
END SUB
```

- **SUB renderWireFramePoly (index AS LONG) STATIC**
- **DIM AS LONG polyCount, i**
- **polyCount = fzxGetBodyD(CFZX_PARAMETER_POLYCOUNT, index, 0)**
    - Query the number of vertices in a body
- **DIM AS tFZX_VECTOR2d vert1, vert2**
- **i = 0: DO WHILE i <= polyCount**
    - Loop through the vertices
- **fzxGetBodyVert index, i, vert1**
    - Helper function that retrieves vertex **'vert1'** from body **'index'** at position **'i'**
    - The vertices are in a _MEM array.
    - The array has a max size set by cMAXVERTSPERBODY set in fzxNGN_ini
- **fzxGetBodyVert index, fzxArrayNextIndex(i, polyCount), vert2**
    - **'fzxArrayNextIndex(i, polyCount)'** will retrieve next vertex in the array and loop back around if past the end.
- **fzxWorldToCamera index, vert1**
    - Convert vertex to screen coordinates.
- **fzxWorldToCamera index, vert2**
    - Convert vertex to screen coordinates.
- **LINE (vert1.x, vert1.y)-(vert2.x, vert2.y), _RGB(0, 255, 0)**
- **i = i + 1: LOOP**
- **END SUB**

Body Creation:

The following function create bodies for the simulation.

fzxCreateCircleBodyEx (“unique name for object”, radius)

- Adds a circle body to simulation
- returns an index to the body in the __fzxBody() array

fzxCreateBoxBodyEx (“unique name for object”, Width, Height)

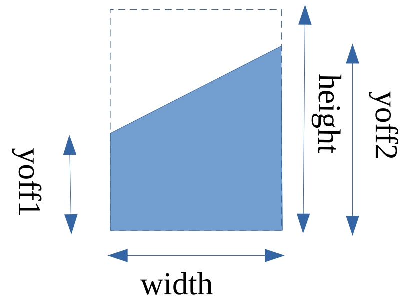
- Adds a Rectangle to the simulation
- returns an index to the body in the __fzxBody() array

fzxCreatePolyBodyEx(“unique name for object”, Width, Height, Number of Sides)

- Adds a Polygon to the simulation.
- Can have arbitrary number of sides (3 to cMAXVERTSPERBODY).
- returns an index to the body in the __fzxBody() array

fzxCreateTrapBodyEx (“unique name for object”, Width, Height, yoff1, yoff2)

- Adds a trapezoid to the simulation
- returns an index to the body in the __fzxBody() array



An example of a body creation would be like:

```
1.    box = fzxCreateBoxBodyEx("box", 100, 100)
2.    fzxSetBody cFZX_PARAMETER_POSITION, box, 100, 100
3.    fzxSetBody cFZX_PARAMETER_STATIC, box, 0, 0
4.    fzxSetBodyEx cFZX_PARAMETER_ORIENT, "box", _D2R(90), 0
```

Line 1 creates a box named “box” that is 100 units long by 100 units wide. As stated earlier units are arbitrary, so it can be 100 miles or 100 millimeters. Its up to the user to decide.

Line 2 the body is moved to a position of 100, 100. Again units are arbitrary.

Line 3 the body is set to static, and acts as a wall or a solid obstacle. You can still move it or arrange it as you see fit.

Line 4 the body is now addressed by its name instead of index and the orientation is set to 90 degrees.

Body Parameters:

The following parameters can be set by the fzxSetBody subroutine.

- fzxSetBody (Parameter, Index, argument 1, argument 2)
  - Index in the body in the __fzxBody() array you are changing
  - The arguments are the new values.
    - Argument 2 may not always be necessary. Just leave it 0.
  - Parameter Contants
    - cFZX_PARAMETER_POSITION
      - Argument 1 – X position in the world
      - Argument 2 – Y position in the world
    - cFZX_PARAMETER_VELOCITY
      - Argument 1 – X velocity in the world
      - Argument 2 – Y velocity in the world
    - cFZX_PARAMETER_FORCE
      - Argument 1 – X force applied to body
      - Argument 2 – Y force applied to body
    - cFZX_PARAMETER_ANGULARVELOCITY
      - Argument 1 – angular velocity in the world
      - Argument 2 – not used
    - cFZX_PARAMETER_TORQUE
      - Argument 1 – torque force applied to body
      - Argument 2 – not used
    - cFZX_PARAMETER_ORIENT
      - Argument 1 – body angle in radians
      - Argument 2 – not used
    - cFZX_PARAMETER_STATICFRICTION
      - Argument 1 – static friction on the body surface
      - Argument 2 – not used
      - More info https://en.wikipedia.org/wiki/Friction
    - cFZX_PARAMETER_DYNAMICFRICTION
      - Argument 1 – dynamic/kinetic friction on the body surface
      - Argument 2 – not used
      - More info https://en.wikipedia.org/wiki/Friction
    - cFZX_PARAMETER_RESTITUTION
      - Argument 1 – bounciness of the body surface
      - Argument 2 – not used
    - cFZX_PARAMETER_COLOR
      - Argument 1 – color used in wire frame, depends on renderer to implement.
      - Argument 2 – not used
    - cFZX_PARAMETER_ENABLE
      - Argument 1 – 0 or non zero
        - Removes body from simulation
        - can be reenabled
      - Argument 2 – not used

- cFZX_PARAMETER_STATIC
  - Sets the object as static and object act like a wall or permanent fixture
  - Argument 1 – not used
  - Argument 2 – not used
- cFZX_PARAMETER_TEXTURE
  - Sets the texture for the body, depends on renderer to implement.
  - Argument 1 – valid texture handle.
  - Argument 2 – not used
- cFZX_PARAMETER_FLIPTEXTURE
  - Flip texture flag, depends on renderer to implement.
  - Argument 1 – 0 or non zero
  - Argument 2 – not used
- cFZX_PARAMETER_SCALETEXTURE
  - Scale texture multiplier, depends on renderer to implement.
  - Argument 1 – X axis, positive non zero number
  - Argument 2 – Y axis, positive non zero number
- cFZX_PARAMETER_OFFSETTEXTURE
  - Shift texture by offset, depends on renderer to implement.
  - Argument 1 – X axis
  - Argument 2 – Y axis
- cFZX_PARAMETER_COLLISIONMASK
  - Used to selectively allow collisions between bodies
    - A value of &B00000001 on one body and value &B00000001 on another body will collide.
    - A value of &B00000010 on one body and &B00000001 on another body will not collide.
    - The default is &B11111111.
    - They essentially logically ANDed together.
  - Argument 1 – unsigned integer
  - Argument 2 – not used
- cFZX_PARAMETER_INVERTNORMALS
  - Experimental feature (I don’t recommend using it)
  - Argument 1 – unsigned integer
  - Argument 2 – not used
- cFZX_PARAMETER_NOPHYSICS
  - Used for sensors. Similar to cFZX_PARAMETER_ENABLE, but body still picks up collisions, but wont react to them.
  - Argument 1 – 0 or non zero
  - Argument 2 – not used
- cFZX_PARAMETER_SPECIALFUNCTION
  - User functionality, can be used for whatever the user needs
  - Argument 1 – any value
  - Argument 2 – any value
- cFZX_PARAMETER_RENDERORDER
  - Depreciated – left for compatibility
  - Argument 1 – any value
  - Argument 2 – unused

- cFZX_PARAMETER_ENTITYID
  - User functionality, can be used for whatever the user needs
  - Argument 1 – any value
  - Argument 2 – unused
- cFZX_PARAMETER_LIFETIME
  - Give the body a finite lifetime
  - Argument 1 – time in seconds
  - Argument 2 – unused
- cFZX_PARAMETER_REPEATTEXTURE
  - Repeat texture multiplier, depends on renderer to implement.
  - Argument 1 – X axis, positive non zero number
  - Argument 2 – Y axis, positive non zero number
- cFZX_PARAMETER_ZPOSITION
  - Sets the body render order, depends on renderer to implement.
  - Argument 1 – Z axis
  - Argument 2 – unused
- cFZX_PARAMETER_UV0
  - Texture Coordinates, depends on renderer to implement.
  - Argument 1 – X axis, positive non zero number
  - Argument 2 – Y axis, positive non zero number
- cFZX_PARAMETER_UV1
  - Texture Coordinates, depends on renderer to implement.
  - Argument 1 – X axis, positive non zero number
  - Argument 2 – Y axis, positive non zero number
- cFZX_PARAMETER_UV2
  - Texture Coordinates, depends on renderer to implement.
  - Argument 1 – X axis, positive non zero number
  - Argument 2 – Y axis, positive non zero number
- cFZX_PARAMETER_UV3
  - Texture Coordinates, depends on renderer to implement.
  - Argument 1 – X axis, positive non zero number
  - Argument 2 – Y axis, positive non zero number

Querying Body Parameters:

~~Making this easier is on the To-Do list. All of the parameters that have been set can be read by looking at the __fzxBody() structure. The structure is defined in the fzxNGN_ini.bas file.~~

~~From the earlier example we can look at the current position~~

1. ~~PRINT __fzxBody(box).fzx.position.x~~
2. ~~PRINT __fzxBody(box).fzx.position.y~~

New function added:

fzxGetBodyD# (Parameter AS LONG, Index AS LONG, arg AS _BYTE)

Returns a double based on the parameter and argument supplied.

Example:

```
x = fzxGetBodyD(cFZX_PARAMETER_POSITION, nodeXB, cFZX_ARGUMENT_X)
y = fzxGetBodyD(cFZX_PARAMETER_POSITION, nodeXB, cFZX_ARGUMENT_Y)
```

Returns the x and y positions of body ‘nodeXB’

Appendix I: List of Subs and Functions

/fzxNGN_BASE_v2/fzxNGN_AABB.bas

fzxAABBOverlap

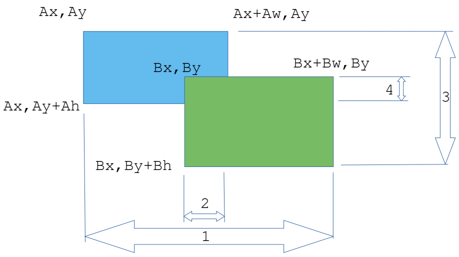
Usage:
ret = fzxAABBOverlap (Ax, Ay, Aw, Ah, Bx, By, Bw, Bh)

The fzxAABBOverlap function detects when two Axis Aligned Bounding Boxes over lap one another. In the fzxNGN, it is used as a fast pretest of two objects colliding, if the two objects are colliding further computation is done.

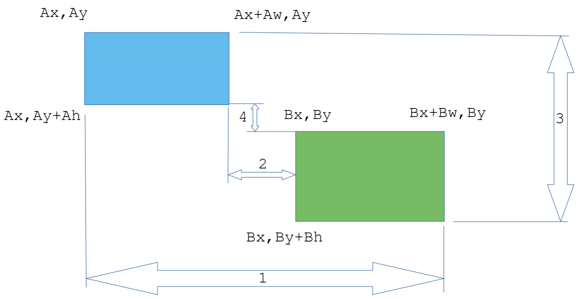
- Takes in two bounding boxes and returns a 0 or -1
- A bounding box is the x and y coordinate of the upper left corner and the height and width.
- All input arguments are typed DOUBLE.

Theory of operation:

```
fzxAABBOverlap = Ax < Bx + Bw AND Ax + Aw > Bx AND Ay < By + Bh AND Ay + Ah > By
```



1. Ax < Bx + Bw is true
2. Ax + Aw > Bx is true
3. Ay < By + Bh is true
4. Ay + Ah > By is true
5. Thus both bounding boxes overlap



1. Ax < Bx + Bw is true
2. Ax + Aw > Bx is false
3. Ay < By + Bh is true
4. Ay + Ah > By is false
5. Thus both bounding boxes do not overlap

Implementation:
FUNCTION fzxAABBOverlap (Ax AS DOUBLE, Ay AS DOUBLE, Aw AS DOUBLE, Ah AS DOUBLE, Bx AS DOUBLE, By AS DOUBLE, Bw AS DOUBLE, Bh AS DOUBLE)

---

fzxAABBOverlapVector

Usage:
ret = fzxAABBOverlapVector (A, Aw, Ah, B, Bw, Bh)

The fzxAABBOverlapVector function detects when two Axis Aligned Bounding Boxes over lap one another. In the fzxNGN, it is used as a fast pretest of two objects colliding, if the two objects are colliding further computation is done.

- Takes in two bounding boxes and returns a 0 or -1.
- A bounding box is the vector of the upper left corner and the height and width.
- The A and B arguments are typed tFZX_VECTOR2D, the rest are typed DOUBLES.

Theory of operation:
Same as fzxAABBOverlap

Implementation:
FUNCTION fzxAABBOverlapVector (A AS tFZX_VECTOR2d, Aw AS DOUBLE, Ah AS DOUBLE, B AS tFZX_VECTOR2d, Bw AS DOUBLE, Bh AS DOUBLE)

---

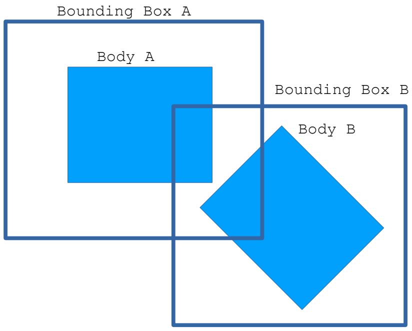
fzxAABBOverlapObjects

Usage:
ret = fzxAABBOverlapObjects (a, b)

The fzxAABBOverlapObjects function detects when two Axis Aligned Bounding Boxes overlap one another. These bounding boxes are directly centered around the fzxNGN bodies.

- Takes in the index of two bodies and performs a AABB test on them and returns a 0 or -1.
- To account for rotation of the body the bounding boxes are scaled in size by the value set by cFZX_AABB_TOLERANCE (default is 1.5)
- It is possible to have no collision and have the bounding boxes overlap, this case still require further computation.
- The AABB overlap is calculated the same ways as fzxAABBOverlap.
- Both input arguments are typed LONG.

Theory of operation:



Implementation:
FUNCTION fzxAABBOverlapObjects (a AS LONG, b AS LONG)

/fzxNGN_BASE_v2/fzxNGN_BASE.bas

fzxCircleInitialize (internal)

Usage:
fzxCircleInitialize indexOfBody
or
CALL fzxCircleInitialize indexOfBody

The fzxCircleInitialize subroutine is a shortcut to the subroutine fzxCircleComputeMass using the default density.

- The default density is set by cFZX_MASS_DENSITY (default is 0.00001)
- The argument is typed LONG, and is a value that was allocated by the body manager.
- This is called during body creation or possibly during body size change.
- Normally this would not be used by the end user.

Theory of operation:
N/A

Implementation:
SUB fzxCircleInitialize (index AS LONG)

---

fzxCircleComputeMass

Usage:
fzxCircleComputeMass indexOfBody, density
or
CALL fzxCircleComputeMass (indexOfBody, density)

The fzxCircleComputeMass subroutine calculates the mass of a 2d circle with a density value. For the sake of the mathematics, we assume all bodies in the simulation has a depth of 1 unit (units are arbitrary).

- The default density is set by cFZX_MASS_DENSITY (default is 0.00001)
- The indexOfBody argument is typed LONG, and the density is a DOUBLE.
- **DO NOT** use subroutine on a body that has not already been created or is not a circle body.
- Mass is calculated by mass = $\pi r^2 d$ (d is density)
- Inverse mass is calculated invMass = 1 / mass
- Inertia is calculated inertia = πr^2
- Inverse Inertia is calculated invInertia = 1 / inertia
- All values are stored in the __fzxBody UDT

Theory of operation:
N/A

Implementation:
SUB fzxCircleComputeMass (index AS LONG, density AS DOUBLE)

fzxPolygonInitialize (internal)

Usage:
fzxPolygonInitialize indexOfBody
or
CALL fzxPolygonInitialize indexOfBody

The subroutine fzxPolygonInitialize is a shortcut to the subroutine fzxPolygonComputeMass using the default density.

- The default density is set by cFZX_MASS_DENSITY (default is 0.00001)
- The argument is typed LONG, and is a value that was allocated by the body manager.
- This is called during body creation or possibly during body size change.
- Normally this would not be used by the end user.

Theory of operation:
N/A

Implementation:
SUB fzxPolygonInitialize (index AS LONG)

fzxPolygonComputeMass

Usage:
fzxPolygonComputeMass indexOfBody, density
or
CALL fzxPolygonComputeMass (indexOfBody, density)

The subroutine fzxPolygonComputeMass calculates the mass of a polygon with an arbitrary number of sides. For the sake of the mathematics, we assume all bodies in the simulation has a depth of 1 unit (units are arbitrary).

- The default density is set by cFZX_MASS_DENSITY (default is 0.00001)
- The indexOfBody argument is typed LONG, and the density is a DOUBLE.
- **DO NOT** use subroutine on a body that has not already been created or is not a circle body.
- Mass, Inverse Mass, Inertia, and Inverse Inertia are calculated and stored in __fzxBody UDT.
- The body is centered around the centroid.

Theory of operation:
This is mathematical sorcery that Randy Gaul could explain better than I ever could. Perhaps some bastardization of the Shoelace algorithm?

Implementation:
SUB fzxPolygonComputeMass (index AS LONG, density AS DOUBLE)

fzxCreateCircleBody (depreciated)

Usage:
ret = fzxCreateCircleBody (indexOfBody, radius)

The function fzxCreateCircleBody creates a circle body with a radius of ***radius***. It returns indexOfBody (yes, I know its redundant.)

- Depreciated do not use, use fzxCreateCircleBodyEx instead.
- This is a legacy command before the body manager existed, was left in because I forgot about it.

Theory of operation:
N/A

Implementation:
FUNCTION fzxCreateCircleBody (index AS LONG, radius AS DOUBLE)

---

fzxCreateCircleBodyEx

Usage:
ret = fzxCreateCircleBodyEx ("MyUniqueName", radius)

The function fzxCreateCircleBodyEx creates a circle body. A unique name should be assigned to the body so that it can be accessed after creation.

- The first argument is a string that can be no longer than 64 characters. The second is a typed DOUBLE.
- The return value is a LONG that contains the index of the body contained in __fzxBody UDT array.
- You can choose to keep track of the index or use the body management tools to access the body later in the simulation.
- A default set of parameters are applied to the body at creation.

Theory of operation:
N/A

Implementation:
FUNCTION fzxCreateCircleBodyEx (objName AS STRING, radius AS DOUBLE)

---

fzxCreateBoxBody(depreciated)

Usage:
ret = fzxCreateBoxBody (indexOfBody, xSize, ySize)

The function fzxCreateBoxBody creates a rectangular body with a size of xSize by ySize. It returns indexOfBody (yes, I know its redundant.)

- Depreciated do not use, use fzxCreateBoxBodyEx or fzxCreatePolyBodyEx instead.
- This is a legacy command before the body manager existed, was left in because I forgot about it.

Theory of operation:
N/A

Implementation:
FUNCTION fzxCreateBoxBody (index AS LONG, xs AS DOUBLE, ys AS DOUBLE)

---

fzxCreateBoxBodyEx

Usage:
ret = fzxCreateBoxBodyEx ("MyUniqueName", xSize, ySize)

The function fzxCreateBoxBodyEx creates a rectangluar body with a size of xSize by ySize. A unique name should be assigned to the body so that it can be accessed after creation.

- The first argument is a string that can be no longer than 64 characters. The second is a typed DOUBLE.
- The return value is a LONG that contains the index of the body contained in __fzxBody UDT array.
- You can choose to keep track of the index or use the body management tools to access the body later in the simulation.
- A default set of parameters are applied to the body at creation.

Theory of operation:
N/A

Implementation:
FUNCTION fzxCreateBoxBodyEx (objName AS STRING, xs AS DOUBLE, ys AS DOUBLE)

---

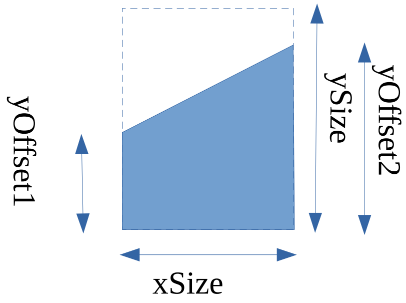
fzxCreateTrapBody (depreciated)

Usage:
fzxCreateTrapBody indexOfBody, xSize, ySize, yOffset1, yOffset2
or
CALL fzxCreateTrapBody (indexOfBody, xSize, ySize, yOffset1, yOffset2)

The subroutine fzxCreateTrapBody creates a trapezoid generally for terrain creation usage.

- Do not use, use fzxCreateTrapBodyEx instead.
- IndexOfBody is typed long and should be already allocated by the body manager.
- xSize and ySize are typed DOUBLE and are the overall dimensions of the trapezoid.
- YOffset1 and yOffset2 are also typed DOUBLE and are negative offsets from the ySize.
- Typical use is for simple terrain generation and making them static objects.
- ToDo: optimize collision handler to only process the upper surface reducing processing by 75%
- Although typical use is for static terrain, They are just like any other body and can be used for any purpose.

Theory of operation:



Implementation:
SUB fzxCreateTrapBody (index AS LONG, xs AS DOUBLE, ys AS DOUBLE, yoff1 AS DOUBLE, yoff2 AS DOUBLE)

fzxCreateTrapBodyEx

Usage:
ret = fzxCreateTrapBodyEx ("MyUniqueName", xSize, ySize, yOffset1, yOffset2)

The fzxCreateTrapBodyEx function creates a trapezoid generally for terrain creation usage.

- The first argument is a unique name so that it may be referenced later using the body manager tools.
- xSize and ySize are typed DOUBLE and are the overall dimensions of the trapezoid.
- YOffset1 and yOffset2 are also typed DOUBLE and are negative offsets from the ySize.
- Typical use is for simple terrain generation and making them static objects.
- ToDo: optimize collision handler to only process the upper surface reducing processing by 75%
- Although typical use is for static terrain, They are just like any other body and can be used for any purpose.

Theory of operation:
Same as fzxCreateTrapBody.

Implementation:
FUNCTION fzxCreateTrapBodyEx (objName AS STRING, xs AS DOUBLE, ys AS DOUBLE, yoff1 AS DOUBLE, yoff2 AS DOUBLE)

fzxCreatePolyBody(depreciated)

Usage:
fzxCreatePolyBody indexOfBody, xSize, ySize, numberOfSides
or
CALL fzxCreatePolyBody (indexOfBody, xSize, ySize, numberOfSides)

The fzxCreatePolyBody subroutine creates a polygon with the size of xSize by ySize. It will have the number of sides dictated by numberOfSides argument.

- Do not use this command use fzxCreatePolyBodyEx instead.
- xSize and ySize are typed DOUBLE and are the radius of the polygon.
- The minimum number of sides is three.
- The maximum number of sides is dictated by the constant cMAXVERTSPERBODY.

Theory of operation:
Refer to fzxPolyCreate for polygon creation.

Implementation:
SUB fzxCreatePolyBody (index AS LONG, xs AS DOUBLE, ys AS DOUBLE, sides AS LONG)

fzxCreatePolyBodyEx

Usage:
ret = fzxCreatePolyBodyEx ("MyUniqueName", xSize, ySize, numberOfSides)

The fzxCreatePolyBodyEx subroutine creates a polygon with the size of xSize by ySize. It will have the number of sides dictated by numberOfSides argument.

- xSize and ySize are typed DOUBLE and are ½ the size of the polygon.
- The minimum number of sides is three.
- The maximum number of sides is dictated by the constant cMAXVERTSPERBODY.
- If the poly has a high number of sides then it is best to use circle body, because it computes much faster.

Theory of operation:
Refer to fzxPolyCreate for polygon creation.

Implementation:
FUNCTION fzxCreatePolyBodyEx (objName AS STRING, xs AS DOUBLE, ys AS DOUBLE, sides AS LONG)

---

fzxCreatePolyBodyTest(experimental)

Usage:
ret = fzxCreatePolyBodyTest ("MyUniqueName", xSize, ySize, numberOfSides)

The fzxCreatePolyBodyTest function is a experimental function to explore concave polygons.

- Currently all non circle bodies run a subroutine fzxVetexSet. fzxVertexSet reorders the vertices to set the normals in the correct direction, and removes concave vertices from the polygon. fzxCreatePolyBodyTest does not perform these operations and leaves the polygon as-is.
- Aside from the vertex manipulation being removed it works the same as fzxCreatePolyBodyEx.

Theory of operation:
Refer to fzxPolyCreate for polygon creation.

Implementation:
FUNCTION fzxCreatePolyBodyTest (objName AS STRING, xs AS DOUBLE, ys AS DOUBLE, sides AS LONG)

---

fzxPolyCreate (internal)

Usage:
fzxPolyCreate indexOfBody, xSize, ySize, numberOfSides
or
CALL fzxPolyCreate (indexOfBody, xSize, ySize, numberOfSides)

The fzxPolyCreate subroutine creates the vertices for the desired polygon and stores them in the __fzxBody() array.

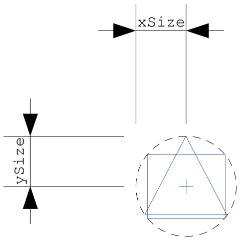
- This is an internal subroutine and should not be used by the end user.

Theory of operation:

The polygon is created by this section of code.

```
  ' verts() is an array vertices for the body
  ' vertCount is the index in the verts() array
  ' theta is the current degree
  ' The step divides 360 degrees by the number of sides

  FOR theta = 0 TO 359 STEP (360 / numberOfSides)
  ' calculate the points in the radius of a circle and add that to the verts() array
    fzxVector2DSet verts(vertCount), xSize * COS(_D2R(theta)), ySize * SIN(_D2R(theta))
    ' increment verts array index
    vertCount = vertCount + 1
  NEXT
```



Implementation:
SUB fzxPolyCreate (index AS LONG, sizex AS DOUBLE, sizey AS DOUBLE, sides AS LONG)

---

fzxPolyCreateTest (internal/experimental)

Usage:
fzxPolyCreateTest indexOfBody, xSize, ySize, numberOfSides
or
CALL fzxPolyCreateTest (indexOfBody, xSize, ySize, numberOfSides)

The fzxPolyCreateTest subroutine creates a polygon with ***numberOfSides*** number of sides. The polygon will be ½ ***xSize*** by ½ ***ySize.*** The polygon will not have had vertices prepared for simulation may not give desired results.

- This is an internal subroutine and should not be used by the end user.

Theory of operation:
Same as fzxPolyCreate

Implementation:
SUB fzxPolyCreateTest (index AS LONG, sizex AS DOUBLE, sizey AS DOUBLE, sides AS LONG)

---

fzxBodyCreate (internal)

Usage:
fzxBodyCreate indexOfBody, shape
or
CALL fzxBodyCreate (indexOfBody, shape)

The fzxBodyCreate subroutine initializes the body with a set of default values. It is an internal subroutine used in the initial body creation.

Theory of operation:
N/A

Implementation:
SUB fzxBodyCreate (index AS LONG, shape AS tFZX_SHAPE)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxBodyCreateEx (objname AS STRING, shape AS tFZX_SHAPE, index AS LONG)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxBoxCreate (index AS LONG, sizex AS DOUBLE, sizey AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxTrapCreate (index AS LONG, sizex AS DOUBLE, sizey AS DOUBLE, yOff1 AS DOUBLE, yOff2 AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxCreateTerrainBody (index AS LONG, slices AS LONG, sliceWidth AS DOUBLE, nominalHeight AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxCreateTerrainBodyEx (objName AS STRING, elevation() AS DOUBLE, slices AS LONG, sliceWidth AS DOUBLE, nominalHeight AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxTerrainCreate (index AS LONG, ele1 AS DOUBLE, ele2 AS DOUBLE, sliceWidth AS DOUBLE, nominalHeight AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxVShapeCreate (index AS LONG, sizex AS DOUBLE, sizey AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxVertexSet (index AS LONG, verts() AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxVertexSetTest (index AS LONG, verts() AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxBodyClear

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxBodyDelete (index AS LONG, perm AS _BYTE)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxFindRightMostVert (verts() AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxVector2DGetSupport (index AS LONG, dir AS tFZX_VECTOR2d, bestVertex AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxShapeCreate (sh AS tFZX_SHAPE, ty AS LONG, radius AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxSetBody (Parameter AS LONG, Index AS LONG, arg1 AS DOUBLE, arg2 AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxSetBodyEx (parameter AS LONG, objName AS STRING, arg1 AS DOUBLE, arg2 AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxGetBodyD# (Parameter AS LONG, Index AS LONG, arg AS _BYTE)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxBodyStop (index AS LONG)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxBodyOffset (index AS LONG, vec AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxBodySetStatic (index AS LONG, arg AS LONG)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxBodyAtRest (index AS LONG, minVel AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxCopyBodies (body() AS tFZX_BODY, newBody() AS tFZX_BODY)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxArrayNextIndex (i AS LONG, count AS LONG)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxCollisionCCHandle (m AS tFZX_MANIFOLD, contacts() AS tFZX_VECTOR2d, A AS LONG, B AS LONG)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxCollisionPCHandle (m AS tFZX_MANIFOLD, contacts() AS tFZX_VECTOR2d, A AS LONG, B AS LONG)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxCollisionCPHandle (m AS tFZX_MANIFOLD, contacts() AS tFZX_VECTOR2d, A AS LONG, B AS LONG)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxCollisionPPClip (n AS tFZX_VECTOR2d, c AS DOUBLE, face() AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxCollisionPPFindIncidentFace (v() AS tFZX_VECTOR2d, RefPoly AS LONG, IncPoly AS LONG, referenceIndex AS LONG)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxCollisionPPHandle (m AS tFZX_MANIFOLD, contacts() AS tFZX_VECTOR2d, A AS LONG, B AS LONG)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxCollisionPPFindAxisLeastPenetration (faceIndex() AS LONG, A AS LONG, B AS LONG)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxImpulseIntegrateForces (index AS LONG)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxImpulseIntegrateVelocity (index AS LONG)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxImpulseStep

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxBodyApplyImpulse (index AS LONG, fzxImpulse AS tFZX_VECTOR2d, contactVector AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxManifoldInit (m AS tFZX_MANIFOLD, contacts() AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxManifoldApplyImpulse (m AS tFZX_MANIFOLD, contacts() AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxManifoldPositionalCorrection (m AS tFZX_MANIFOLD)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxManifoldInfiniteMassCorrection (A AS LONG, B AS LONG)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxJointAllocate

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxJointCreate (b1 AS LONG, b2 AS LONG, x AS DOUBLE, y AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxJointCreateEx (b1 AS LONG, b2 AS LONG, anchor1 AS tFZX_VECTOR2d, anchor2 AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxJointClear

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxJointDelete (d AS LONG)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxJointSet (index AS LONG, b1 AS LONG, b2 AS LONG, x AS DOUBLE, y AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxJointSetEx (index AS LONG, b1 AS LONG, b2 AS LONG, anchor1 AS tFZX_VECTOR2d, anchor2 AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxJointPrestep (index AS LONG)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxJointApplyImpulse (index AS LONG)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxIsBodyTouchingBody (A AS LONG, B AS LONG)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxIsBodyTouchingStatic (A AS LONG)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxIsBodyTouching (A AS LONG)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxHighestCollisionVelocity (hits() AS tFZX_HIT, A AS LONG) ' this function is a bit dubious and may not do as you think

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxBodyManagerAdd ()

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxBodyWithHash (hash AS _INTEGER64)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxBodyWithHashMask (hash AS _INTEGER64, mask AS LONG)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxBodyManagerID (objName AS STRING)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxBodyContainsString (start AS LONG, s AS STRING)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxComputeHash&& (s AS STRING)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxHandleNetwork (net AS tFZX_NETWORK)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxNetworkStartHost (net AS tFZX_NETWORK)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxNetworkReceiveFromHost (net AS tFZX_NETWORK)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxNetworkTransmit (net AS tFZX_NETWORK)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxNetworkClose (net AS tFZX_NETWORK)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxFSMChangeState (fsm AS tFZX_FSM, newState AS LONG)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxFSMChangeStateEx (fsm AS tFZX_FSM, newState AS LONG, arg1 AS tFZX_VECTOR2d, arg2 AS tFZX_VECTOR2d, arg3 AS LONG)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxFSMChangeStateOnTimer (fsm AS tFZX_FSM, newstate AS LONG)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxReadArrayLong& (s AS STRING, p AS LONG)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxSetArrayLong (s AS STRING, p AS LONG, v AS LONG)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxReadArraySingle! (s AS STRING, p AS LONG)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxSetArraySingle (s AS STRING, p AS LONG, v AS SINGLE)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxReadArrayInteger% (s AS STRING, p AS LONG)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxSetArrayInteger (s AS STRING, p AS LONG, v AS INTEGER)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxReadArrayDouble# (s AS STRING, p AS LONG)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxSetArrayDouble (s AS STRING, p AS LONG, v AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxInitFPS

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxFPS

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxFPSMain

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxFPSdt

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxHandleFPSMain

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxHandleFPSGL
/fzxNGN_BASE_v2/fzxNGN_CAMERA.bas

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxWorldToCamera (index AS INTEGER, vert AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxWorldToCameraEx (posVert AS tFZX_VECTOR2d, vert AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxCalculateFOV

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxCameraToWorld (oVec AS tFZX_VECTOR2d, iVec AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxCameratoWorldEx (iVec AS tFZX_VECTOR2d, oVec AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxCameratoWorldScEx (iVec AS tFZX_VECTOR2d, oVec AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxWorldToCameraBody (index AS LONG, vert AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxWorldToCameraBodyNR (index AS LONG, vert AS tFZX_VECTOR2d)
/fzxNGN_BASE_v2/fzxNGN_IMPULSEMATH.bas

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxImpulseEqual (a AS DOUBLE, b AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxImpulseClamp# (min AS DOUBLE, max AS DOUBLE, a AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxImpulseRound# (a AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxImpulseRandomFloat## (min AS _FLOAT, max AS _FLOAT)

---

fzx

Usage:

Theory of operation:

Implementation:

FUNCTION fzxImpulseRandomInteger% (min AS INTEGER, max AS INTEGER)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxImpulseRandomdouble# (min AS DOUBLE, max AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxImpulseGT (a AS DOUBLE, b AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxImpulseWithin (v AS DOUBLE, low AS DOUBLE, high AS DOUBLE)
/fzxNGN_BASE_v2/fzxNGN_INPUT.bas

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxHandleInputDevice
/fzxNGN_BASE_v2/fzxNGN_LINESEG.bas

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxLineIntersection (l1 AS tFZX_LINE2d, l2 AS tFZX_LINE2d, o AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxLineSegmentsIntersect (l1 AS tFZX_LINE2d, l2 AS tFZX_LINE2d)
/fzxNGN_BASE_v2/fzxNGN_MATRIXMATH.bas

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxMatrix2x2SetRadians (m AS tFZX_MATRIX2D, radians AS DOUBLE)

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxMatrix2x2SetScalar (m AS tFZX_MATRIX2D, a AS DOUBLE, b AS DOUBLE, c AS DOUBLE, d AS DOUBLE)

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxMatrix2x2Abs (m AS tFZX_MATRIX2D, o AS tFZX_MATRIX2D)

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxMatrix2x2GetAxisX (m AS tFZX_MATRIX2D, o AS tFZX_VECTOR2d)

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxMatrix2x2GetAxisY (m AS tFZX_MATRIX2D, o AS tFZX_VECTOR2d)

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxMatrix2x2TransposeI (m AS tFZX_MATRIX2D)

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxMatrix2x2Transpose (m AS tFZX_MATRIX2D, o AS tFZX_MATRIX2D)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxMatrix2x2Invert (m AS tFZX_MATRIX2D, o AS tFZX_MATRIX2D)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxMatrix2x2MultiplyVector (m AS tFZX_MATRIX2D, v AS tFZX_VECTOR2d, o AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxMatrix2x2AddMatrix (m AS tFZX_MATRIX2D, x AS tFZX_MATRIX2D, o AS tFZX_MATRIX2D)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxMatrix2x2MultiplyMatrix (m AS tFZX_MATRIX2D, x AS tFZX_MATRIX2D, o AS tFZX_MATRIX2D)
/fzxNGN_BASE_v2/fzxNGN_MEM.bas

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxSetBodyVertXY (indexBody AS LONG, indexVert AS LONG, x AS DOUBLE, y AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxSetBodyVert (indexBody AS LONG, indexVert AS LONG, v AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxGetBodyVertX# (indexBody AS LONG, indexVert AS LONG)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxGetBodyVertY# (indexBody AS LONG, indexVert AS LONG)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxGetBodyVert (indexBody AS LONG, indexVert AS LONG, vert AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxSetBodyNormXY (indexBody AS LONG, indexNorm AS LONG, x AS DOUBLE, y AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxSetBodyNorm (indexBody AS LONG, indexNorm AS LONG, v AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxGetBodyNormX# (indexBody AS LONG, indexNorm AS LONG)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxGetBodyNormY# (indexBody AS LONG, indexNorm AS LONG)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxGetBodyNorm (indexBody AS LONG, indexNorm AS LONG, norm AS tFZX_VECTOR2d)
/fzxNGN_BASE_v2/fzxNGN_PERLIN.bas

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxPerlinScaleOffset (p AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxPerlinInterpolate# (a0 AS DOUBLE, a1 AS DOUBLE, w AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxPerlinRandomGradient (seed AS DOUBLE, ix AS INTEGER, iy AS INTEGER, o AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxPerlinDotGridGradient# (seed AS DOUBLE, ix AS INTEGER, iy AS INTEGER, x AS DOUBLE, y AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxPerlin# (x AS DOUBLE, y AS DOUBLE, seed AS DOUBLE)
/fzxNGN_BASE_v2/fzxNGN_POLYGON.bas

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxPolygonMakeCCW (obj AS tFZX_TRIANGLE)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxPolygonIsReflex (t AS tFZX_TRIANGLE)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxPolygonSetOrient (index AS LONG, radians AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxPolygonInvertNormals (index AS LONG)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxPointInTriangle (a AS tFZX_VECTOR2d, b AS tFZX_VECTOR2d, c AS tFZX_VECTOR2d, p AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxAreaOfPolygon# (v() AS tFZX_VECTOR2d)
/fzxNGN_BASE_v2/fzxNGN_SCALARMATH.bas

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxScalarMin# (a AS DOUBLE, b AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxScalarMax# (a AS DOUBLE, b AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxScalarMap# (x AS DOUBLE, in_min AS DOUBLE, in_max AS DOUBLE, out_min AS DOUBLE, out_max AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxScalarLERP# (current AS DOUBLE, target AS DOUBLE, t AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxScalarLERPSmooth# (current AS DOUBLE, target AS DOUBLE, t AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxScalarLERPSmoother# (current AS DOUBLE, target AS DOUBLE, t AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxScalarLERPProgress# (startTime AS DOUBLE, endTime AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxScalarRoughEqual (a AS DOUBLE, b AS DOUBLE, tolerance AS DOUBLE)
/fzxNGN_BASE_v2/fzxNGN_VECMATH.bas

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxVector2DSet (v AS tFZX_VECTOR2d, x AS DOUBLE, y AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxVector2dSetVector (o AS tFZX_VECTOR2d, v AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxVector2dNeg (v AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxVector2DNegND (o AS tFZX_VECTOR2d, v AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxVector2DMultiplyScalar (v AS tFZX_VECTOR2d, s AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxVector2DMultiplyScalarND (o AS tFZX_VECTOR2d, v AS tFZX_VECTOR2d, s AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxVector2DDivideScalar (v AS tFZX_VECTOR2d, s AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxVector2DDivideScalarND (o AS tFZX_VECTOR2d, v AS tFZX_VECTOR2d, s AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxVector2DAddScalar (v AS tFZX_VECTOR2d, s AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxVector2DAddScalarND (o AS tFZX_VECTOR2d, v AS tFZX_VECTOR2d, s AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxVector2DMultiplyVector (v AS tFZX_VECTOR2d, m AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxVector2DMultiplyVectorND (o AS tFZX_VECTOR2d, v AS tFZX_VECTOR2d, m AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxVector2DDivideVector (v AS tFZX_VECTOR2d, m AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxVector2DDivideVectorND (o AS tFZX_VECTOR2d, v AS tFZX_VECTOR2d, m AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxVector2DAddVector (v AS tFZX_VECTOR2d, m AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxVector2DAddVectorND (o AS tFZX_VECTOR2d, v AS tFZX_VECTOR2d, m AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxVector2DAddVectorScalar (v AS tFZX_VECTOR2d, m AS tFZX_VECTOR2d, s AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxVector2DAddVectorScalarND (o AS tFZX_VECTOR2d, v AS tFZX_VECTOR2d, m AS tFZX_VECTOR2d, s AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxVector2DSubVector (v AS tFZX_VECTOR2d, m AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxVector2DSubVectorND (o AS tFZX_VECTOR2d, v AS tFZX_VECTOR2d, m AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxVector2DSwap (v1 AS tFZX_VECTOR2d, v2 AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxVector2DLengthSq# (v AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxVector2DLength# (v AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxVector2DRotate (v AS tFZX_VECTOR2d, radians AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxVector2DNormalize (v AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxVector2DMin (a AS tFZX_VECTOR2d, b AS tFZX_VECTOR2d, o AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxVector2DMax (a AS tFZX_VECTOR2d, b AS tFZX_VECTOR2d, o AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxVector2DDot# (a AS tFZX_VECTOR2d, b AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxVector2DSqDist# (a AS tFZX_VECTOR2d, b AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxVector2DDistance# (a AS tFZX_VECTOR2d, b AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxVector2DCross# (a AS tFZX_VECTOR2d, b AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxVector2DCrossScalar (o AS tFZX_VECTOR2d, v AS tFZX_VECTOR2d, a AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxVector2DArea# (a AS tFZX_VECTOR2d, b AS tFZX_VECTOR2d, c AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxVector2DLeft# (a AS tFZX_VECTOR2d, b AS tFZX_VECTOR2d, c AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxVector2DLeftOn# (a AS tFZX_VECTOR2d, b AS tFZX_VECTOR2d, c AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxVector2DRight# (a AS tFZX_VECTOR2d, b AS tFZX_VECTOR2d, c AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxVector2DRightOn# (a AS tFZX_VECTOR2d, b AS tFZX_VECTOR2d, c AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxVector2DCollinear# (a AS tFZX_VECTOR2d, b AS tFZX_VECTOR2d, c AS tFZX_VECTOR2d, thresholdAngle AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxVector2DLERP (curr AS tFZX_VECTOR2d, start AS tFZX_VECTOR2d, target AS tFZX_VECTOR2d, inc AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxVector2DLERPSmooth (curr AS tFZX_VECTOR2d, start AS tFZX_VECTOR2d, target AS tFZX_VECTOR2d, inc AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxVector2DLERPSmoother (curr AS tFZX_VECTOR2d, start AS tFZX_VECTOR2d, target AS tFZX_VECTOR2d, inc AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxVector2DOrbitVector (o AS tFZX_VECTOR2d, position AS tFZX_VECTOR2d, dist AS DOUBLE, angle AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxVector2DEqual (a AS tFZX_VECTOR2d, b AS tFZX_VECTOR2d, tolerance AS DOUBLE)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB fzxVector2DMid (o AS tFZX_VECTOR2d, v1 AS tFZX_VECTOR2d, v2 AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxGetAngleVector2d# (p1 AS tFZX_VECTOR2d, p2 AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION fzxGetAngle# (x1 AS DOUBLE, y1 AS DOUBLE, x2 AS DOUBLE, y2 AS DOUBLE) 'returns 0-359.99...
/fzxNGN_BASE_v2/fzxNGN_XML.bas

---

fzx

Usage:

Theory of operation:

Implementation:
SUB XMLparse (file AS STRING, con() AS tFZX_STRINGTUPLE)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION getXMLArgValue# (i AS STRING, s AS STRING)

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION getXMLArgString$ (i AS STRING, s AS STRING)

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION topStackString$ (stack() AS STRING)

fzx

Usage:

Theory of operation:

Implementation:
SUB pushStackString (stack() AS STRING, element AS STRING)

fzx

Usage:

Theory of operation:

Implementation:
SUB popStackString (stack() AS STRING)

fzx

Usage:

Theory of operation:

Implementation:
SUB pushStackContextArg (stack() AS tFZX_STRINGTUPLE, element_name AS STRING, element AS STRING)

fzx

Usage:

Theory of operation:

Implementation:
SUB pushStackContext (stack() AS tFZX_STRINGTUPLE, element AS tFZX_STRINGTUPLE)

fzx

Usage:

Theory of operation:

Implementation:
SUB popStackContext (stack() AS tFZX_STRINGTUPLE)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB pushStackVector (stack() AS tFZX_VECTOR2d, element AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB popStackVector (stack() AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB topStackVector (o AS tFZX_VECTOR2d, stack() AS tFZX_VECTOR2d)

---

fzx

Usage:

Theory of operation:

Implementation:
SUB loadFilebyLine (fl AS STRING, filetext() AS STRING)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION trim$ (s AS STRING)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION isAlpha (c AS STRING)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION isDigit (c AS STRING)

---

fzx

Usage:

Theory of operation:

Implementation:
FUNCTION isSymbol (c AS STRING)