N7 PROGRAMMING LANGUAGE EXTENSION

EngineA (enginea.n7) – API Reference

Introduction

EngineA (`EA`) is a sector- and portal-based 3D game engine for N7. A level consists of sectors (rooms) connected by portals (openings and doorways). By calculating portal visibility from the camera's current sector and clipping rendering to visible doorways, the engine avoids drawing hidden geometry—providing crucial performance gains for an interpreted language.

- This reference documents the functions and object interfaces in `enginea.n7`.
- For map editing (`enginea_editor.exe`) and step-by-step usage, see `enginea_tutorial.pdf` and the examples in `examples/enginea_library`.
- **Units:** Unless otherwise noted, all angles are specified in **radians**.

Setup & Engine Configuration

> **`EA_SetView(`**`(number)` **`target,`** `(number)` **`fov,`** `(number)` **`z_min,`** `(number)` **`z_max)`**

Configures the rendering viewport and view frustum.

- `target` – Destination image to render to (usually `primary`).
- `fov` – Field of view in radians.
- `z_min` / `z_max` – Near and far clipping planes. `z_max` also defines the distance at which distance fog reaches full opacity.

> **`EA_SetFog(`**`(number)` **`mode,`** `(number)` **`r,`** `(number)` **`g,`** `(number)` **`b)`**

Configures distance fog (disabled by default) using RGB color `(r, g, b)`.

Constant	Description
`EA_NONE` *(default)*	Disables fog.
`EA_NORMAL`	Smooth distance fog fading to `(r, g, b)` at `z_max`.
`EA_RETRO`	Quantized, sharp-stepped fog bands for a retro look.

> **`EA_SetDoorMode(`**`(number)` **`mode)`**

Sets how portal doors animate when opening and closing.

Constant	Description
`EA_SLIDE_UP` *(default)*	Doors slide upward into the ceiling.
`EA_SLIDE_DOWN`	Doors slide downward into the floor.
`EA_SLIDE_SIDE`	Doors slide horizontally into the adjacent wall.

> **`EA_SetDoorSpeed(`**`(number)` **`speed)`**

Sets door open/close speed. The default is `2.5` (meaning a door takes `1 / 2.5 = 0.4 s` to open or close). High values like `2.5` are recommended because entities cannot pass through a door until it is completely open.

> **`EA_SetFpsCap(`**`(number)` **`fps)`**

Caps the frame rate by calling `fwait` after each update/render cycle.

- `fps` – Target frame rate when `> 0`, or `0` *(default)* to disable capping. Limiting FPS (e.g., to `60`) is recommended because relative mouse look in FPS games behaves more reliably when the loop sleeps briefly.

> **`EA_SetDebugOutput(`**`(number)` **`value)`**

Enables on-screen debug output (including the current FPS counter) when set to `true` (`false` by default).

> **`EA_SetSoundMaxDist(`**`(number)` **`d)`**

Sets the maximum audible distance `d` (default `8`) for 3D positional sounds played with `EA_PlaySound()` or `obj.PlaySound()`.

> **`EA_PlaySound(`**`(number)` **`snd,`** `(number)` **`vol,`** `(number)` **`x,`** `(number)` **`y,`** `(number)` **`z)`**

Plays sound `snd` at volume `vol` from 3D world coordinates `(x, y, z)` with stereo panning and distance attenuation relative to the camera. Sounds at or beyond the max distance set by `EA_SetSoundMaxDist()` are not played.

Maps

> Levels must be built in the EngineA map editor (`enginea_editor.exe`); maps cannot be constructed procedurally from scratch in code.

> **(array) EA_LoadMap(**(string) **filename)**

Loads a map file created with the EngineA editor (automatically freeing any previously loaded map). Displays a runtime error if loading fails.

- `filename` – Path to the map file.
- **Returns:** An array of flag tables placed in the editor, each containing:
  - `(string) flag` – Flag name.
  - `(string) value` – Flag value.
  - `(number) x`, `(number) z` – World X and Z coordinates.
  - `(number) floorY` – Floor Y coordinate at `(x, z)`, or `unset` if outside any sector.
  - `(number) ceilingY` – Ceiling Y coordinate at `(x, z)`, or `unset` if outside any sector.

> **EA_FreeMap()**

Frees all resources for the currently loaded map. Called automatically by `EA_LoadMap()`, so manual calls are rarely needed.

Callbacks & Game Loop

Once your view, map, camera, and objects are initialized, call `EA_Run()` to start the main loop.

> **EA_SetUpdateAction(**(function) **func)**

Registers a global update callback invoked once per frame with `dt` (elapsed seconds since the previous frame, e.g., `~0.0167` at 60 FPS):

```
function MyUpdateFunc(dt)
    if keydown(KEY_ESCAPE)  EA_Stop()
endfunc

EA_SetUpdateAction(MyUpdateFunc)
```

> **EA_SetDrawAction(**(function) **func)**

Registers a 2D/3D overlay draw callback invoked every frame after the 3D world has rendered, but before `redraw` is called. Ideal for HUD elements (health, score) or first-person weapons rendered via `s3d`:

```
function MyDrawFunc()
    set color 255, 255, 255
    set caret 0, 0
    wln "Hello world!"
endfunc

EA_SetDrawAction(MyDrawFunc)
```

`EA_Run()`

Enters the engine's main game loop. Blocks until `EA_Stop()` is called. A runtime error occurs if `EA_SetCamera()` has not been called beforehand.

`EA_Stop()`

Signals the game loop to exit and return from `EA_Run()`.

`EA_Pause()`

Pauses internal engine timers and updates so you can run custom logic/rendering (such as a pause or settings menu). Must be paired with `EA_Resume()`:

```
function MyUpdateFunc(dt)
    if keydown(KEY_ESCAPE, true)
        EA_Pause()
        set color 0, 0, 0, 128
        cls
        set color 255, 255, 255
        set caret width(primary)/2, height(primary)/2
        center "Game paused..."
        redraw
        while not keydown(KEY_ESCAPE, true)
        wend
        fwait 60
        EA_Resume()
    endif
endfunc
```

`EA_Resume()`

Resumes engine timers and execution after `EA_Pause()`.

Camera & FPS Player Object

```
EA_SetCamera((table) obj)
```

Assigns an object derived from `EA_Object()` (such as an `EA_FpsPlayer()`) to act as the active camera.

```
(table) EA_Camera()
```

Returns the current camera object.

```
(table) EA_FpsPlayer()
```

Creates a preconfigured first-person player/camera object extending `EA_Object()`. By default, it uses mouse look, `WASD` movement, and `Space` to jump. Remember to register it with `EA_AddObject()` and `EA_SetCamera()`.

`EA_FpsPlayer` Key Bindings

- `SetForwardKey((number) key_code)` / `(number) ForwardKey()` – Move forward *(default `KEY_W`)*.
- `SetBackwardKey((number) key_code)` / `(number) BackwardKey()` – Move backward *(default `KEY_S`)*.
- `SetStrafeLeftKey((number) key_code)` / `(number) StrafeLeftKey()` – Strafe left *(default `KEY_A`)*.
- `SetStrafeRightKey((number) key_code)` / `(number) StrafeRightKey()` – Strafe right *(default `KEY_D`)*.
- `SetStrafeKey((number) key_code)` / `(number) StrafeKey()` – Modifier key that turns rotation keys into strafing *(default `unset`)*.
- `SetRotateLeftKey((number) key_code)` / `(number) RotateLeftKey()` – Turn left *(default `unset`)*.
- `SetRotateRightKey((number) key_code)` / `(number) RotateRightKey()` – Turn right *(default `unset`)*.
- `SetJumpKey((number) key_code)` / `(number) JumpKey()` – Jump *(default `KEY_SPACE`)*.

`EA_FpsPlayer` Mouse, Movement & Jumping

- `SetMouseSens((number) value)` / `(number) MouseSens()` – Mouse sensitivity *(default `1`; set to `unset` to disable mouse look)*.
- `SetUsePitch((number) value)` / `(number) UsePitch()` – Enables/disables vertical mouse look *(default `true`)*.
- `SetMaxPitch((number) angle)` / `(number) MaxPitch()` – Maximum vertical look angle in radians *(default `0.35 * PI`)*.
- `SetMoveSpeed((number) speed)` / `(number) MoveSpeed()` – Movement speed in units/sec *(default `1`)*.
- `SetRotateSpeed((number) speed)` / `(number) RotateSpeed()` – Keyboard turn speed in radians/sec *(default `PI / 2`)*.
- `SetLeap((number) leap_height)` / `(number) Leap()` – Max step height the player can walk up without jumping *(default `0`)*.
- `SetJumpForce((number) force)` / `(number) JumpForce()` – Initial vertical jump velocity in units/sec *(default `5`; gravity is `5` units/sec^2)*.
- `Jump((number) force)` – Triggers a jump with initial vertical velocity `force`.
- `(number) Jumping((number) unflag)` – Returns `true` if jumping; if `unflag` is `true`, clears the flag until the next jump.
- `(number) Walking()` – Returns `true` if the player is moving.
- `(number) OnGround()` – Returns `true` if the player is standing on solid ground.

You can attach custom per-frame logic to an `EA_FpsPlayer` by defining its `Update(dt)` function:

```
player = EA_FpsPlayer()
player.maySpaceJump = false
EA_AddObject(player)
EA_SetCamera(player)

player.Update = function(dt)
    if this.OnGround()
        this.maySpaceJump = true
    else
        if this.maySpaceJump
            if keydown(this.JumpKey(), true)
                this.maySpaceJump = false
                this.Jump(this.JumpForce())
            endif
        endif
    endif
endfunc
```

World Objects (`EA_Object` & `EA_StaticObject`)

Objects represent all dynamic or decorative entities in the world (sprites, 3D meshes, or invisible triggers/cameras). Each object has a 3D position `(x, y, z)` (where `y` is the **bottom** of the object), orientation angles `yaw` (around Y) and `pitch` (around X), and a bounding cylinder defined by `height` and `radius`.

```
EA_AddObject((table) obj)
```

Adds a dynamic object to the world.

```
EA_AddStaticObject((table) obj)
```

Adds a static object to the world (cannot move once the game loop has started).

```
EA_RemoveObject((table) obj)
```

Removes any dynamic or static object from the world.

```
(array) EA_Objects()
```

Returns an array of all active non-static objects.

```
(table) EA_Object()
```

Creates a new dynamic game object with the following methods:

Position, Orientation & Dimensions

- `SetPos((number) x, (number) y, (number) z)` – Sets position (`y` = bottom). Use during setup; use `Move()` for collision-checked movement.
- `(number) X()`, `(number) Y()`, `(number) Z()` – Return individual coordinates.
- `(array) Pos()` – Returns `[x, y, z]` *(do not modify this array)*.
- `SetYaw((number) angle)` / `(number) Yaw()` – Sets/gets yaw rotation in radians.
- `SetPitch((number) angle)` / `(number) Pitch()` – Sets/gets pitch rotation in radians.
- `(number) DX()`, `(number) DY()`, `(number) DZ()` – Return unit direction vector components derived from yaw and pitch.
- `(array) Dir()` – Returns `[dx, dy, dz]` *(do not modify this array)*.
- `SetHeight((number) h)` / `(number) Height()` – Sets/gets bounding cylinder and sprite height.
- `SetRadius((number) r)` / `(number) Radius()` – Sets/gets bounding cylinder radius (sprite width = `2 * radius`).

- `SetEye((number) h)` / `(number) Eye()` – Sets/gets vertical eye offset from `y` (used for camera height and `Facing()`).

Rendering (Sprites, Meshes & Lighting)

- `SetSprite((number) img, (number) cel, (number) only_yaw)` – Renders the object as a 2D sprite using image `img` and grid cell `cel` (`0` if ungridded). If `only_yaw` is `true`, the billboard rotates only around the Y-axis; if `false`, it faces the camera on all axes.
- `(number) Sprite()`, `SetCel((number) cel)`, `(number) Cel()` – Get/set sprite image and cell.
- `SetMesh((number) mesh)` / `(number) Mesh()` – Renders the object using an `s3d` mesh ID (rotated by yaw/pitch; scaled at load time via `S3D_LoadMesh`).
- `SetFrame((number) frame)` / `(number) Frame()` – Sets/gets active mesh keyframe.
- `SetFrames((number) frame_1, (number) frame_2, (number) blend)` – Blends between two mesh keyframes (`blend` in `[0..1]`).
- `(number) Frame2()` / `(number) Blend()` – Return secondary keyframe and blend factor.
- `SetUseSectorLight((number) value)` / `(number) UseSectorLight()` – Enables or disables sector ambient lighting for this object *(default `true`)*.

Movement, Collision & Spatial Queries

- `SetCollisionMode((number) mode)` – Sets collision accuracy for `Move()`: `EA_NORMAL` *(recommended for players)* or `EA_FAST` *(recommended for enemies/projectiles)*.
- `(table) Move((number) dx, (number) dy, (number) dz, (number) leap)` – Moves the object by `(dx, dy, dz)` with collision detection, allowing step-ups up to `leap` units. Returns a collision result table:
  - `(number) w`, `(number) g`, `(number) c`, `(number) any` – Booleans indicating collision with wall, ground, ceiling, or any surface.
  - `(number) dx`, `(number) dz` – X and Z components of the hit wall's surface normal.
  - `(number) dy` – `-1` on floor collision, `+1` on ceiling collision.
  - `(table) info` – Details about the hit wall (same structure as `Facing()`, or `unset` if `w` is `false`).
- `(table) Facing()` – Raycasts along the object's look direction to find the closest wall, door, or static object (returns `unset` if looking at a floor or ceiling):
  - `(number) dist` – Distance to the hit surface.
  - `(number) type` – `EA_WALL`, `EA_DOOR`, or `EA_OBJECT`.
  - `(number/table) data` – Wall texture index (`EA_WALL`), door object (`EA_DOOR`), or static object (`EA_OBJECT`).
- `(table) Sector()` – Returns the sector table the object currently occupies *(do not modify; useful for comparing if two objects share a sector)*.

- `(array) SectorObjects()` – Returns an array of all objects in the same sector, including this object *(do not modify)*.
- `(number) Visible((table) obj)` – Returns `true` if there is an unobstructed line of sight between this object's center and `obj`.
- `(number) CollidesWith((table) obj)` – Returns `true` if this object's bounding cylinder overlaps `obj`'s bounding cylinder.
- `(number) DistanceTo((table) obj)` – Returns Euclidean distance to `obj`.
- `(number) SqrDistanceTo((table) obj)` – Returns squared distance to `obj` (faster for distance comparisons).
- `PlaySound((number) snd, (number) vol)` – Plays sound `snd` at volume `vol` from this object's position.

Object Lifecycle Callbacks

You can assign custom callback functions to any `EA_Object`, `EA_StaticObject`, or `EA_FpsPlayer` instance:

- `Update((number) dt)` – Called every frame while the loop is running.
- `Run()` – Called once when `EA_Run()` starts.
- `Stop()` – Called once when `EA_Stop()` is triggered.
- `Pause()` – Called whenever `EA_Pause()` is invoked.
- `Resume()` – Called whenever `EA_Resume()` is invoked.
- `Render()` – Called during rendering if you want to draw custom `s3d` geometry for the object.

> **`(table) EA_StaticObject()`**

Creates a static world object (e.g., crates, pillars, barrels). Supports all `EA_Object()` methods **except** `Move()` and `SetCollisionMode()`, plus:

- `SetColPoly((array) p)` – Defines an invisible collision prism around the object using a closed 2D XZ polygon `[x0, z0, x1, z1, ..., xn, zn]` relative to the object's center (do not repeat the first vertex at the end). Walls extend to the object's `Height()`, and a walkable top surface is created unless `Height()` is `unset`.

Sectors & Doors

> **`(array) EA_Sectors()`**

Returns an array of all sectors in the loaded map. Sector tables expose the following methods:

- `(string) Name()` / `SetName((string) name)` – Gets/sets the sector name defined in the editor.
- `(array) Light()` – Returns the sector's RGB light color as `[r, g, b]`.
- `SetLight((number) r, (number) g, (number) b)` – Sets the sector's RGB light color.
- `(array) Objects()` – Returns an array of all objects currently inside the sector.
- `(array) Doors()` – Returns an array of all doors connected to the sector.

`(array)` EA_RenderedSectors()

Returns an array of sectors rendered during the most recent frame (terminated by an `unset` element, as the buffer array may exceed the rendered count).

`(number)` EA_RenderedSectorCount()

Returns the exact number of sectors rendered during the most recent frame.

`(array)` EA_Doors()

Returns an array of all doors in the loaded map. Door objects expose the following methods:

- `(string) Id()` / `SetId((string) id)` – Gets/sets the door ID assigned in the editor.
- `(number) Texture()` / `SetTexture((number) index)` – Gets/sets the door's wall texture index.
- `(number) UseSectorLight()` / `SetUseSectorLight((number) value)` – Gets/sets whether the door is lit by sector lighting *(default `true`)*.
- `(number) X()`, `(number) Y()`, `(number) Z()` – Return the center world coordinates of the door.
- `(number) Open()` – Triggers the door to open; returns `true` if it was not already open or opening.
- `(number) Close()` – Triggers the door to close; returns `true` if the door was not already closed or closing.