TaxReceipt v2 Handbook

By Rubberduckycooly

Contents

Introduction to Retro-Engine v2 & TaxReceipt v2

The Retro Engine Software Development Kit v2 (or Retro-Engine/RSDK for short) is a classic-styled 2D focused game engine with many “old school” graphics effects such as mode 7 and the use of manipulatable palettes. RSDKv2 was only used in the Sonic CD (2011) remaster, since the Retro-Engine was upgraded to RSDKvB for the Sonic 1 & Sonic 2 mobile remasters. RSDKv2 is powered by its proprietary scripting language: TaxReceipt (v2), through which all object logic is run. TaxReceipt has a syntax similar to that of C and/or visual basic.

TaxReceipt’s syntax foregoes semicolons & braces much like visual basic, instead opting to use line breaks to mark the end of expressions. Since TaxReceipt is a scripting language it offers some benefits such as:

- being able to compile a script when a stage is loaded or restarted
- making changes quick and easy to do, as it is also heavily woven into the RSDK’s development environment
- being specifically designed to create object code and nothing else.
- RSDKv2 is still a very sonic-centered engine so creating sonic objects is quite easy to accomplish

Though TaxReceipt is not without its limitations though, many of them will make it a challenge for some seasoned C-like programmers to get a handle on the language, such as:

- Custom variables cannot be defined. All variables used are part of the default ones, though there’s TempValue0-7 for temporary variable storage & Object.Value0-7 for more permanent variable storage
- There are no data types aside from integers. that means no floats or string storage, though certain functions do accept string constants as parameters, such as functions that load assets like animations
- User-defined functions cannot have parameters. All values carry over when a function is called though, so storing variables in TempValues and then using those in the called function works as semi-parameters
- Arithmetic expressions only have one value on the left and one on the right. So, doing “A = B + C” is illegal, however “A = B” then “A += C” is legal

Arithmetic

As mentioned before arithmetic in TaxReceipt is more limited than other languages, only allowing access to the following operators:

"==+==-==++=--=*==/==>>==<<==&==|==^==%=" and “-, so longer expressions have to be broken down into multiple lines.

Eg:

“A = 1 + 2 * 3”

Would be something like:

“TempValue0 = 2”

“TempValue0 *= 3”

“A = 1”

“A += TempValue0”

Both expressions will end with *A* being equal to 7.

Another limitation to keep in mind is that functions do not count as variables & as such never return any value, instead opting to set another variable (see function information for more details), this also means that using functions in an expression is illegal, however using the value functions set is not.

Eg:

(function “getA” returns A and does not exist in TaxReceipt, it is psuedocode that’s purely for demonstration purposes)

“A = 5;”

“B = getA(A);”

Would be something like

“A = 5”

“getA(C, A)”

“B = C”

Because in this case *getA* sets parameter 1 (C) to the return value instead of returning it directly

Conditionals, Statements & Scopes

TaxReceipt offers standard statements such as “if=else=while” & “switch”

Note: TaxReceipt does **not** support the use of the “for” or “else if” statements

Note2: Unlike functions, statements do **not** require parenthesis: “if A == B” is the correct way to format a statement.

Statements must only have one conditional, as such “||” and “&&” operators are not supported in TaxReceipt, however regular conditional operators such as “==,>,=>,<=,<,!=” are supported in statement use only, using them in conjunction with arithmetic is illegal and will cause the compiler to throw an error

Since TaxReceipt does not use braces, there are special keywords used to end a statement they are as follows:

if/else – endif

while – loop

switch – endswitch

An example for correct formatting is shown below:

<u>If statement:</u>

```
If A != B
        FuncA()
else
```

```
    FuncB()
endif
```

<u>While Statement:</u>

```
while A < B
    A++
loop
```

<u>Switch Statement:</u>

```
switch A
case 1
        FunctionA()
    break
case 2
    FunctionB()
    break
case 3
case 4
    FunctionC()
default
    FunctionD()
    break
endswitch
```

Note: break, case & default keywords work exactly as they would in C: integers only, fallthrough is allowed, default is optional, etc

Subs & User-Defined Functions

TaxReceipt v2 has 4 default functions, known as “subs” which are called at periodic instances during gameplay. These subs are:

sub ObjectMain: called once every frame, if the object’s priority allows for it (see priority notes below). Used for the majority of code

sub ObjectPlayerInteraction: called once every frame, if the object’s priority allows for it, after ObjectMain and only if the current player has ObjectInteractions enabled, do any player collision code here.

sub ObjectDraw: called once every frame when object is about to be drawn to the screen, the ordering is based on the drawList (see below), only called if the object was added to the drawList (if ObjectMain wasn’t called, its likely ObjectDraw won’t be either, due to priority). Draw Graphics here, anywhere else and they won't be shown.

sub ObjectStartup: called **once per object type**. Objects haven’t been loaded yet so using anything like Object.Value is pointless. Setup & load assets here

User-defined functions can be definded using the syntax “function [name]” and called by using either “callFunction([functionName])” or using “callFunction([functionID])”

Function names are essentially ints under the hood so function “pointers” are not only possible but used commonly. To use it its as simple as something like:

TempValue0=MyFunction

CallFunction(TempValue0)

That will call “MyFunction” or whatever function tempValue is currently assigned to

When ending a sub, the keyword “end sub” is required, likewise the keyword “end function” is required when ending functions. A basic example is shown below:

```
sub ObjectMain
        FunctionA()
        TempValue0 += 4
end sub


function FunctionA
TempValue0 = 3
end function
```

Aliases

Since TaxReceipt lacks the ability to define custom variables and all default variables have names like “Object.Value0, TaxReceipt’s solution to this is to introduce the alias system, where by using the format **#alias [original]:[aliasName]** in scripts variables & values can be given unique names to help readability. Two examples are shown below.

- #alias 2: TYPE_STAGESETUP. This makes “TYPE_STAGESETUP” have a value of 2 so if it was assigned to a value it would assign 2. so TempValue0=TYPE_STAGESETUP is the same as TempValue0=2
- #alias Player.Value0:Player.Rings. This makes “Player.Rings” use the underlying value as “Player.Value0” so TempValue0=Player.Rings is the same as TempValue0=Player.Value0

In addition to user-defined aliases, RSDKv2 also has some default aliases that can be used for convenience. These are shown below in the format of [Name]=[Value]

true=1, used as it would be in any other language

false=0, used as it would be in any other language

FX_SCALE=0, used for DrawSpriteFX or DrawSpriteScreenFX, as the FX type to draw the sprite using scaling

FX_ROTATE=1, used for DrawSpriteFX or DrawSpriteScreenFX, as the FX type to draw the sprite using rotation

FX_ROTOZOOM=2, used for DrawSpriteFX or DrawSpriteScreenFX, as the FX type to draw the sprite using scaling & rotation

FX_INK=3, used for DrawSpriteFX or DrawSpriteScreenFX, as the FX type to draw the sprite using blending effects (none,blend,alpha,additive & subtractive)

FX_FLIP=5, used for DrawSpriteFX or DrawSpriteScreenFX, as the FX type to draw the sprite using flipping

PRESENTATION_STAGE=0, used for StageList operations. Is the ID for “Presentation Stages”

REGULAR_STAGE=1, used for StageList operations. Is the ID for “Regular Stages”

BONUS_STAGE=2, used for StageList operations. Is the ID for “Bonus Stages”

SPECIAL_STAGE=3, used for StageList operations. Is the ID for “Special Stages”

MENU_1=0, Used for TextMenu functions, is the ID for menu1

MENU_2=1, Used for TextMenu functions, is the ID for menu2

C_TOUCH=0, used for CheckPlayerCollisions, if used, collision will check if there was a collision and set CheckResult to true if so, no collision response is done

C_BOX=1, used for CheckPlayerCollisions, if used, collision will check if there was a collision and set CheckResult accordingly (0 = none 1 = top, 2 = left, 3 = right, 4 = bottom), box collision response is done here

C_BOX2=2, used for CheckPlayerCollisions, if used, collision will check if there was a collision and set CheckResult accordingly (0 = none 1 = top, 2 = left, 3 = right, 4 = bottom), box collision response is done here, mostly the same as C_BOX with a few tweaks, use C_BOX if unsure which one to use, this collision type is not available in the PC port of sonic CD

C_PLATFORM=3, used for CheckPlayerCollisions, if used, collision will check if there was a collision and set CheckResult to true if there was a collision, box collision response is done here, this collision type means the collision will only return true if the player was on top of it

MAT_WORLD=0, matrix ID used for matrix functions, this is the ID of the world matrix

MAT_VIEW=1, matrix ID used for matrix functions, this is the ID of the view matrix

MAT_TEMP=2, matrix ID used for matrix functions, this is the ID of the temp matrix, used for storing and holding values before they’re used in conjunction with the above 2

FACING_LEFT=1, aka flipX

FACING_RIGHT=0, aka no flip

STAGE_PAUSED=2, used with Stage.State, will set the stage to be paused

STAGE_RUNNING=1, used with Stage.State, will set the stage to be running (unpaused)

RESET_GAME=2, used with Engine.State, will reset the game

RETRO_WIN=0, used with Engine.PlatformID, this is the value it'll be if playing on windows

RETRO_OSX=1, used with Engine.PlatformID, this is the value it'll be if playing on Mac OSX

RETRO_XBOX_360=2, used with Engine.PlatformID, this is the value it'll be if playing on Xbox 360

RETRO_PS3=3, used with Engine.PlatformID, this is the value it'll be if playing on Playstation 3

RETRO_iOS=4, used with Engine.PlatformID, this is the value it'll be if playing on IOS

RETRO_ANDROID=5, used with Engine.PlatformID, this is the value it'll be if playing on android

RETRO_WP7=6, used with Engine.PlatformID, this is the value it'll be if playing on windows phone 7

Built-in Functions

TaxReceipt has a whole bunch of built-in function to assist with engine related calls/general help that would otherwise be a pain to setup, they are all listed below with variable names, types and a description of what they do.

Sin(int store, int angle), sets *store* to the 512-based sine value using *angle*

Cos(int store, int angle), sets *store* to the 512-based cosine value using *angle*

Sin256(int store, int angle), sets *store* to the 256-based sine value using *angle*

Cos256(int store, int angle), sets *store* to the 256-based cosine value using *angle*

SinChange(int store, int unknownA, int unknownB, int unknownC, int unknownD), functionality currently unknown

CosChange(int store, int unknownA, int unknownB, int unknownC, int unknownD), functionality currently unknown

ATan2(int store, int X, int Y), sets *store* to the arcTan value of *X* and *Y*

Interpolate(int store, int unknownA, int unknownB, int unknownC), functionality currently unknown

InterpolateXY(int storeX, int storeY, int unknownX1, int unknownX2, int unknownY1, int unknownY2, int unknown3), functionality currently unknown

LoadSpriteSheet(string sheetPath), loads the obejct spritesheet from “Data/Sprites/[s*heetPath*]” and sets *Object.Spritesheet* to the id of the loaded sheet, valid extensions are: .gif, .bmp & .gfx (Retro-Sonic gfx files), only one spritesheet can be loaded per object

RemoveSpriteSheet(string sheetPath), unloads the spritesheet *sheetPath*

DrawSprite(int frame), draws a sprite using *frame* at the object’s XY position, uses 16 bit bit shifted co-ordinates for XYPos, so 0x10000 == 1.0, 0x8000 == 0.5, etc

DrawSpriteXY(int frame, int XPos, int YPos), draws a sprite using *frame* at XPos & YPos, uses 16 bit bit shifted co-ordinates for XYPos

DrawSpriteScreenXY(int frame, int XPos, int YPos), draws a sprite using *frame* at XPos & YPos, relative to the screen. Uses regular co-ordinates (so 5,5 = 5 pixels right and 5 pixels down on screen)

DrawTintRect(int XPos, int YPos, int width, int height), draws a tint rect based on *width* and *height* at *XPos* & *YPos*

DrawNumbers(int startingFrame, int XPos, int YPos, int value, int digitCnt, int spacing, int showAllDigits), draws *value* using 10 consecutive frames for digits, starting from *startingFrame*, at *XPos* & *YPos (*Uses regular co-ordinates), with *spacing* pixels between each frame. Will only draw valid digits (or *digitCnt* digits if numer is exceeded) if *showAllDigits* is not enabled, otherwise *digitCnt* digits will be drawn, with extras being 0.

DrawActName(int startingFrame, int XPos, int YPos, int align, int unknown, int unknown2, int spacing), draws the loaded stage's act name using 26 frames starting from *startingFrame* ending at *startingFrame* + 26 (only english **and** uppercase text is supported), at *XPos* & *YPos* (Uses regular co-ordinates)*,* using *alignment* to determine where the text center is (1 = middle, 2 = right), with *spacing* pixels between each letter

DrawMenu(int menuID, int XPos, int YPos), draws the menu entries contained in the menu *menuID*, at *XPos* & *YPos*. it is recommended to use *MENU_1* & *MENU_2* for *menuID*

SpriteFrame(int pivotX, int pivotY, int width, int height, int sprX, int sprY), adds a spriteframe to the object using *sprX* and *sprY* to determine sheet locations, *width* and *height* to determine sprite size, and *pivotX* and *pivotY* to determine their drawing offsets.

EditFrame(int frameID, int pivotX, int pivotY, int width, int height, int sprX, int sprY), (not available in PC port), recreates spriteframe *frameID*, using *sprX*, *sprY*, *width*, *height*, *pivotX* & *pivotY*.

LoadPalette(string filePath, int paletteID, int startPaletteIndex, int startIndex, int endIndex), loads a palette from "Data/Palettes/[*filePath*]" into the internal palette *paletteID*. The colours *startIndex* through *endIndex* will be loaded from the palette file into the internal palette sequentially starting from index *startPaletteIndex*

RotatePalette(int startIndex, int endIndex, int right), shifts the colour indexes *startIndex* through *endIndex* to the left or right (depending on if *right* is true or not) one index, with the end colour going into *startIndex*

SetScreenFade(int R, int G, int B, int A), sets the screen fade effect. *R*, *G* & *B* are used for colour (0-255 for each) and *A* is used for how transparent it is (0-255, though it can be higher it internally caps at 255)

SetActivePalette(int palID, int startLine, int endLine), sets the active palette for *startLine* through to *endLine* to *palID*, this is used for effects like water, which usually use a second palette that is active on any lines drawn below Stage.WaterLevel

SetPaletteFade(int paletteID, int R, int G, int B, int A, int startIndex, int endIndex), functionality currently unknown

CopyPalette(int srcPal, int destPal), copies all colours from *destPal* into *srcPal*

ClearScreen(int paletteIndex), clears the screen using *paletteindex* as a colour

DrawSpriteFX(int frame, int FXType, int XPos, int YPos), draws a sprite using *frame* with a graphic effect *FXType* at *XPos* & *YPos*, uses 16 bit bitshifted co-ordinates for XYPos. It is recommended to use *FX_SCALE*, *FX_ROTATE*, *FX_ROTOZOOM* & *FX_FLIP* for *FXType*.

DrawSpriteScreenFX(int frame, int FXType, int XPos, int YPos), draws a sprite using *frame* with a graphic effect *FXType* at *XPos* & *YPos*, relative to the camera. uses regular co-ordinates for XYPos. It is recommended to use *FX_SCALE*, *FX_ROTATE*, *FX_ROTOZOOM* & *FX_FLIP* for *FXType*.

LoadAnimation(string animationPath), Loads an animation file from “Data/Animations/[animationPath]”, **Note:** only one animation file can be loaded per object **and** animations MUST be drawn using DrawObjectAnimation() (or DrawPlayerAnimation() for player object) otherwise sprite frames will be drawn instead, and yes, an object may have an animation loaded as well as separate spriteframes

SetupMenu(int menuID, int rowCount, int selectionCount, int alignment), sets up menu *menuID*, with *rowCount* rows, using *selectionCount* selections and with *alignment* alignment for entries. it is recommended to use *MENU_1* & *MENU_2* for *menuID*

AddMenuEntry(int menuID, string entryText, int highlightEntry), adds a text entry to menu *menuID* using *entryText*, if *highlightEntry* is set it’ll be highlighted. it is recommended to use *MENU_1* & *MENU_2* for *menuID*

EditMenuEntry(int menuID, string entryText, int entryID, int highlightEntry), edits text entry *entryID* in menu *menuID*, setting the text to entry text and the highlight to *highlightEntry*. it is recommended to use *MENU_1* & *MENU_2* for *menuID*

LoadStage(), loads a stage. The loaded stage is determined by the values of Stage.ActiveStageList for category and Stage.StageListPosition for scene

DrawRect(int XPos, int YPos, int width, int height, int R, int G, int B, int A), draws a rectangle to the screen at *XPos* & *YPos* (Uses regular co-ordinates), with a width of *width* pixels and a height of *height* pixels, using *R*, *G* & *B* (0-255) for colour and *A* for transparency (0-255, can be greater but internally caps at 255)

ResetObjectEntity(int slot, int type, int propertyValue, int XPos, int YPos), resets the object in *slot*, to object type *type*, with a propertyValue (subtype) of *propertyValue*, and sets its XYPos to *XPos* & *YPos* uses 16 bit bitshifted co-ordinates

PlayerObjectCollision(int colType, int left, int top, int right, int bottom), checks to see if there was a collision with the player object using left, top, right & bottom as the object hitbox (see RSDK Animation editor for more on how those co-ordinates work). It is recommended to use *C_TOUCH*, *C_BOX*, *C_BOX2* (mobile only) and *C_PLATFORM* for colType

CreateTempObject(int type, int propertyValue, int XPos, int YPos), creates a temporary object in the temp object slots with an object type *type*, with a propertyValue (subtype) of *propertyValue*, and sets its XYPos to *XPos* & *YPos* uses 16 bit bitshifted co-ordinates. Temp objects should be stuff like smoke and other misc effects, if longer lasting objects are required, either use ResetObjectEntity and manually select an object slot or set an object's type manually. The created object's slotID is stored in *TempObjectPos*.

BindPlayerToObject(int playerID, int objectID), binds the player object *playerID* to an object slot *objectID*, this is almost certainly a requirement as without this the game will likely crash or at the very least not function well at all

PlayerTileCollision(), performs player tile collision checks using the active hitbox in the player's animation file

ProcessPlayerControl(), processes the player's inputs & input state, sets the resulting input flags to Player.Up, Player.Down, Player.Left & Player.Right for movement inputs, and Player.JumpPress & Player.JumpHold for jump inputs

ProcessAnimation(), processes the active animation, as set by Object.Animation, after loading an animation

DrawObjectAnimation(), Draws the object at the object's XY position, using Object.Animation, Object.Frame & the loaded animation as data to determine what frame to draw, fx flags are based on the animation's rotation flags (See: RSDK Animation editor for more info), only enabled fx are rotation & flip

DrawPlayerAnimation(), mostly the same as above but called if the player object is used rather than regular objects

SetMusicTrack(string filePath, int trackID, int loopPoint), loads the music file (has to be .ogg) from "Data/Music/[*filePath*]" into track slot *trackID*, with a loopPoint of *loopPoint* (0 = no loop & 1 = always loop, any other value is a sample index to restart from)

PlayMusic(int trackID), plays the music loaded in *trackID*

StopMusic(), stops the currently playing music

PlaySfx(int sfxID, int loopCnt), plays the global soundFX *sfxID*, and repeats it *loopCnt* times

StopSfx(int sfxID), stops playing the global soundFX *sfxID* if its playing

SetSfxAttributes(int sfx, int loopCnt, int pan), sets the loopCount to loopCnt and the panning to pan (-100 = left, 100 = right, 0 = balanced) for the soundFX *sfx*, stageSFX ids are stage sfx ID + globalSFXCount, so if there were 5 global Sfx and 2 stage sfx and you wanted to play stage sfx 2, you'd have *sfx* 7 (5 + 2)

ObjectTileCollision(int cSide, int xOffset, int yOffset, int cPlane), check and respond to tile collisions on collision plane cPlane for objects, not as versatile as player collisions, this will only check if a collision has been hit on side cSide and respond, no other physics or anything will be done.

The collision check will be done at (iXPos + *xOffset*, iYPos + *yOffset*). This should be used for just checking if a surface is there and responding rather than being used when consistently moving along a surface. (valid cSide values: CSIDE_FLOOR = 0, CSIDE_LWALL = 1, CSIDE_RWALL = 2, CSIDE_ROOF = 3) these are not aliases the engine provides so the values will have to be provided instead, the aliases are just listed here for easier understanding

ObjectTileGrip(int cSide, int xOffset, int yOffset, int cPlane), check and respond to tile collisions on collision plane cPlane for objects, not as versatile as player collisions, this will only check if a collision has been hit on side cSide and respond, no other physics or anything will be done. The collision check will be done at (iXPos + *xOffset*, iYPos + *yOffset*). This differs from ObjectTileCollision as it is a more complex check and is meant for clinging onto surfaces rather than just checking for collisions, this should be used in instances where objects move along the ground for example. (valid cSide values: CSIDE_FLOOR = 0, CSIDE_LWALL = 1, CSIDE_RWALL = 2, CSIDE_ROOF = 3) these are not aliases the engine provides so the values will have to be provided instead, the aliases are just listed here for easier understanding

LoadVideo(string videoName), loads a video based on *videoName*, for .ogv files all that's needed is the filename without the extension (so "Opening" loads the opening (Videos/Opening.ogv) for example. If using the PC port, support is also added for loading RSV files using this method, these files are loaded using videoName as well, and will be loaded like: "Data/Sprites/[videoName]" ONLY if the extension contains ".rsv"

NextVideoFrame(), advances to the next video frame, only used if an RSV file is loaded (aka almost never)

PlayStageSfx(int sfxID, int loopCnt), plays the stage soundFX *sfxID*, and repeats it *loopCnt* times

StopStageSfx(int sfxID), stops playing the stage soundFX *sfxID* if its playing

Not(int value), performs a not operation on *value* (*value* = ~*value*)

Draw3DScene(), renders all faces in the faceBuffer into the scene after being sorted and transformed into screen space by MAT_WORLD & MAT_VIEW

SetIdentityMatrix(int matrixID), sets the identity matrix for *matrixID,* it is recommended to use MAT_WORLD, MAT_VIEW or MAT_TEMP for matrixID. **NOTE:** RSDKv2 uses an 8-bit shifted value system for matrices, so 1.0 on a standard matrix would be 0x100 in RSDKv2, aside from that matrix math is the same as it would be in any other environment

MatrixMultiply(int matrixA, int matrixB), multiplies *matrixA* by *matrixB,* it is recommended to use MAT_WORLD, MAT_VIEW or MAT_TEMP for matrixA & matrixB.

MatrixTranslateXYZ(int matrixID, int XPos, int YPos, int ZPos), translates *matrixID* to *XPos*, *YPos* & *ZPos,* it is recommended to use MAT_WORLD, MAT_VIEW or MAT_TEMP for *matrixID*.

MatrixScaleXYZ(int matrixID, int scaleX, int scaleY, int scaleZ), scales *matrixID* by *scaleX*, *scaleY* & *scaleZ,* it is recommented to use MAT_WORLD, MAT_VIEW or MAT_TEMP for *matrixID*.

MatrixRotateX(int matrixID, int rotationX), rotates *matrixID* on the X axis by *rotationX,* it is recommended to use MAT_WORLD, MAT_VIEW or MAT_TEMP for *matrixID*.

MatrixRotateY(int matrixID, int rotationY), rotates *matrixID* on the Y axis by *rotationY,* it is recommended to use MAT_WORLD, MAT_VIEW or MAT_TEMP for *matrixID*.

MatrixRotateZ(int matrixID, int rotationZ), rotates *matrixID* on the Z axis by *rotationZ,* it is recommended to use MAT_WORLD, MAT_VIEW or MAT_TEMP for *matrixID*.

MatrixRotateXYZ(int matrixID, int rotationX, int rotationY, int rotationZ), rotates *matrixID* on all 3 axes by *rotationX*, *rotationY* & *rotationZ* respectively*,* it is recommended to use MAT_WORLD, MAT_VIEW or MAT_TEMP for matrixID.

TransformVertices(int matrixID, int startVert, int endVert), transforms all verticies in the vertexBuffer starting from startVert all the way to endVert by *matrixID,* it is recommended to use MAT_WORLD, MAT_VIEW or MAT_TEMP for *matrixID*.

CallFunction(int functionID), calls function *functionID*, can be said function's name, as long as it was declaired before being called

endfunction, used instead of "endsub" if in a function instead of a sub

SetLayerDeformation(int deformID, int deformationA, int deformationB, int deformType, int deformOffset, int deformCount),

CheckTouchRect(int X1, int Y2, int Y2), checks if there has been a touch input inside the positions of *X1*,*Y1* & *X2*,*Y2* (co-ordinates are screen-relative) and if so CheckResult is set to the touchID, if there was no touch CheckResult is set to -1

GetTileLayerEntry(int store, int layer, int chunkX, int chunkY), gets the chunkID on layer *layer* at *chunkX*, *chunkY* (using chunk co-ordinates, top left chunk is 0,0, the one next to it is 0,1, the one below it is 0,1 etc) and sets it in *store*

SetTileLayerEntry(int value, int layer, int chunkX, int chunkY), sets the chunkID on layer *layer* at *chunkX*, *chunkY* (using chunk co-ordinates, top left chunk is 0,0, the one next to it is 0,1, the one below it is 0,1 etc) to *value*

GetBit(int store, int value, int bitID), gets the state of bit *bitID* in *value*, and sets store to the result (true or false)

SetBit(int value, int bitID, int set), sets the state of bit *bitID* in *value* to *set*, and updates *value*.

PauseMusic(), pauses the currently playing music

ResumeMusic(), resumes the currently playing music

ClearDrawList(int layerID), clears the draw list for layerID

AddDrawListEntityRef(int drawLayer, int entityID), adds *entityID* as an entry to the draw list for *drawLayer*

GetDrawListEntityRef(int store, int drawLayer, int entryID), gets the entry *entryID* in *drawList*, and sets it in *store*

SetDrawListEntityRef(int value, int drawLayer, int entryID), sets the entry *entryID* in *drawList* to *value* (drawList entries are entity slot IDs)

Get16x16TileInfo(int store, int tileX, int tileY, int infoType), gets info about the 16x16 tile at *tileX*, *tileY* (using tile co-ordinates, top left tile is 0,0, the one next to it is 0,1, the one below it is 0,1 etc) and sets it in *store*. Valid *infoType* values: TILEINFO_INDEX = 0, TILEINFO_DIRECTION = 1, TILEINFO_SOLIDITYA = 2, TILEINFO_SOLIDITYB = 3, TILEINFO_FLAGSA = 4, TILEINFO_ANGLEA = 5, TILEINFO_FLAGSB = 6, TILEINFO_ANGLEB = 7) these aliases aren't supported natively by the engine they are just here to help with understanding.

Copy16x16Tile(int destTileID, int srcTileID), copies the pixel data for *scrTileID* into *destTileID* (0-1023 for both), used for aniTiles.

Set16x16TileInfo(int value, int tileX, int tileY, int infoType), sets info about the 16x16 tile at *tileX*, *tileY* (using tile co-ordinates, top left tile is 0,0, the one next to it is 0,1, the one below it is 0,1 etc) to *store*. Valid *infoType* values: TILEINFO_INDEX = 0, TILEINFO_DIRECTION = 1, TILEINFO_SOLIDITYA = 2, TILEINFO_SOLIDITYB = 3, TILEINFO_FLAGSA = 4, TILEINFO_ANGLEA = 5) these aliases aren't supported natively by the engine they are just here to help with understanding.

GetAnimationByName(int store, string animName), attempts for find an animation with the name *animName* in the loaded animation and if found will be set in *store*. If not found *store* is set to 0.

ReadSaveRAM(), reads the save file into SaveRAM

WriteSaveRAM(), writes SaveRAM to file

LoadTextFont(string fontPath), loads a font file from *fontPath* for use with drawing text

LoadTextFile(int menuID, string filePath, int mapCode), loads a text file from *filePath* into *menuID*, if mapCode is set the text data will be mapped to the loaded font file

DrawText(int menuID, int XPos, int YPos, int scale, int spacing, int rowStart, int rowCount), draws text from *menuID* at *XPos* & *YPos* (uses regular co-ordinates) with a scale of *scale*, with *spacing* pixels between each character. Text will be drawn starting from *rowStart*, for *rowCount* rows

GetTextInfo(int store, int menuID, int infoType, int infoID, int infoOffset), gets info about the textMenu *menuID* and stores it in *store*. Valid *infoType* values: TEXTINFO_TEXTDATA = 0, TEXTINFO_TEXTSIZE = 1, TEXTINFO_ROWCOUNT = 2, these aliases aren't supported natively by the engine they are just here to help with understanding. *infoID* is used for textData and textSize to determine the text entry to get info from, its not used at all in rowCount. Likewise, neither is *infoOffset*, that's only used in textData to get the characterID in the text menu string.

GetVersionNumber(int menuID, int highlightText), adds a text entry to *menuID* with the text being the game's loaded version, it will be highlighted if *highlightText* is enabled

SetAchievement(int achievementID, int status), sets achievement *achievementID* status to *status*, common statuses are: 0 = not achieved, 100 = achieved

SetLeaderboard(int leaderboardID, int leaderboardEntry), sets the leaderboard entry in leaderboard *leaderboardID* to *leaderboardEntry*.

LoadOnlineMenu(int menuType), loads an online menu, 0 = achievements menu, 1 = leaderboards menu

EngineCallback(int callbackID), sends a callback to the engine, this mostly handles messages between hardcoded parts of the engine and scripted parts, callbackID 0-16 is used by the game, the rest aren't

HapticEffect(int unknownA, int unknownB, int unknownC, int unknownD), (not avaliable in the PC port) this is only in the mobile port and I have no clue how it works at all

Built-in Variables

Since TaxReceipt doesn’t support adding custom variables, it comes built in with a ton of variables to use that can get almost any job done, below is a list of variables and descriptions for each variable. If a variable contains “[index]” that means that it is an array variable is able to be indexed, all array variables besides object ones should never be used without specifically picking which array value to use, otherwise who knows what could happen. If Object values are used without an index, they will just default to the active object so they’re fine

TempValue0, Temporary Value 0, useful for anything, except storing long term values

TempValue1, Temporary Value 1, useful for anything, except storing long term values

TempValue2, Temporary Value 2, useful for anything, except storing long term values

TempValue3, Temporary Value 3, useful for anything, except storing long term values

TempValue4, Temporary Value 4, useful for anything, except storing long term values

TempValue5, Temporary Value 5, useful for anything, except storing long term values

TempValue6, Temporary Value 6, useful for anything, except storing long term values

TempValue7, Temporary Value 7, useful for anything, except storing long term values

CheckResult, Set based on various function results, only use this in arithmatic if you know what you’re doing otherwise hard to fix bugs may pop up

ArrayPos0, array position 0, since TaxReceipt can only use constants or ArrayPos0, ArrayPos1 or TempObjectPos for array indexes its advised to use this only when needed

ArrayPos1, array position 1, since TaxReceipt can only use constants or ArrayPos0, ArrayPos1 or TempObjectPos for array indexes its advised to use this only when needed

Global[index], global variable storage, it is highly recommended to never ever use this value in this form, instead add global variables to the gameconfig and they will be able to be used by using their names. Example: adding variable “Options.GameMode” to the gameconfig would be used as “Options.GameMode=1” but internally it’d be Global[0]=1

Object[index].EntityNo, the object’s slot ID

Object[index].Type, the object’s type

Object[index].PropertyValue, the object’s propertyValue (subtype)

Object[index].XPos, the object’s X position (shifted by 16 ,aka 1.0 = 0x10000)

Object[index].YPos, the object’s Y position (shifted by 16, aka 1.0 = 0x10000)

Object[index].iXPos, the object’s X position as a full number (so XPos: 0x20000 == iXPos: 2)

Object[index].iYPos, the object’s Y position as a full number (so YPos: 0x20000 == iYPos: 2)

Object[index].State, the object’s state, can be used for whatever, is managed on a per object basis

Object[index].Rotation, the object’s rotation value, goes from 0-511 to match the Sin function, used by drawing if fxType is FX_ROTATE

Object[index].Scale, the object’s scale value, scales both X & Y, scale is 9 bit shifted so 1.0 == 0x200, 0.5 == 0x100, etc

Object[index].Priority, the object’s priority state, determines how the object manages update priority, valid values are: PRIORITY_ACTIVE_BOUNDS = 0 (object will update if its within 128 pixels of the screen borders), PRIORITY_ACTIVE = 1 (object will always update, unless paused), PRIORITY_ACTIVE_PAUSED = 2 (object will always update, even if paused), PRIORITY_ACTIVE_XBOUNDS = 3 (object will update if its XPos is within 128 pixels of the horizontal screen borders, no matter where the YPos is), PRIORITY_ACTIVE_BOUNDS_REMOVE = 4 (the same as bounds, but if the bounds check fails, the object is destroyed, used for things like debug mode objects), PRIORITY_INACTIVE = 5 (object never updates). once again these aliases are not supported by the engine natively so the raw values will have to be used instead, they are provided here for ease of understanding

Object[index].DrawOrder, the object’s draw Order, this determines what drawList the object will go in after objectMain

Object[index].Direction, the object’s direction flag, used for FX_FLIP, and general direction management, usually 0-3 (FLIP_NONE, FLIP_X, FLIP_Y & FLIP_XY respectively)

Object[index].InkEffect, the object’s inkEffect flag, used with FX_INK. 0 = no ink effect, 1 = blending ink effect (think about 0.5 alpha), 2 = alpha blending ink effect, object transparency will be based on alpha value of 0-255, 3 = additive ink effect, works like alpha, but applies an additive blending instead of an alpha blending effect, 4 = subtractive, works like alpha, but applies a subtractive blending effect instead of an alpha one

Object[index].Alpha, the object’s transparency value, used with the correct inkEffect modes, goes from 0-255, can be higher but the engine caps most alpha effects at 255

Object[index].Frame, the object’s frame ID, used with loaded animations, but also used to store spriteframe ids

Object[index].Animation, the object’s animation id, only really used when an animation is loaded

Object[index].PrevAnimation, the object’s previous animation, set and managed by ProcessAnimation()

Object[index].AnimationSpeed, the object’s animation speed, used and managed by ProcessAnimation(), though some objects use it to handle their own anims

Object[index].AnimationTimer, the object’s animation timer, also used and managed by ProcessAnimation(), though again some objects use and manage their animations with this themselves

Object[index].Value0, object storage value 0, used for storing values long term in object slots, can be used for whatever, managed on an object-by-object basis

Object[index].Value1, object storage value 1, used for storing values long term in object slots, can be used for whatever, managed on an object-by-object basis

Object[index].Value2, object storage value 2, used for storing values long term in object slots, can be used for whatever, managed on an object-by-object basis

Object[index].Value3, object storage value 3, used for storing values long term in object slots, can be used for whatever, managed on an object-by-object basis

Object[index].Value4, object storage value 4, used for storing values long term in object slots, can be used for whatever, managed on an object-by-object basis

Object[index].Value5, object storage value 5, used for storing values long term in object slots, can be used for whatever, managed on an object-by-object basis

Object[index].Value6, object storage value 6, used for storing values long term in object slots, can be used for whatever, managed on an object-by-object basis

Object[index].Value7, object storage value 7, used for storing values long term in object slots, can be used for whatever, managed on an object-by-object basis

Object[index].OutOfBounds, the object’s out of bounds flag, will be true if the object is not on screen

Player.State, the player’s state, can be used for whatever, is managed on a per player basis

Player.ControlMode, the player’s control mode. Manages how ProcessPlayerControl() works, if –1 the player’s control is taken away, if 0 the player’s control is completely normal, if 1 the player’s control is locked to the last input

Player.ControlLock, determines how many frames the control lock lasts for

Player.CollisionMode, what collisionMode the player is on right now, valid values are: CMODE_FLOOR = 0, CMODE_LWALL = 1, CMODE_ROOF = 2, CMODE_RWALL = 3, aliases not available in engine, used here purely for understanding easier, not to be confused with CSIDE

Player.CollisionPlane, the player’s active collision plane (0 = PlaneA, 1 = PlaneB)

Player.XPos, the player's X position (shifted by 16, aka 1.0 = 0x10000)

Player.YPos, the player's Y position (shifted by 16, aka 1.0 = 0x10000)

Player.iXPos, the player's X position as a full number (so XPos: 0x20000 == iXPos: 2)

Player.iYPos, the player's Y position as a full number (so YPos: 0x20000 == iYPos: 2)

Player.ScreenXPos, the player’s current X position relative to the screen screen as a full number

Player.ScreenYPos, the player’s current Y position relative to the screen screen as a full number

Player.Speed, the player’s current speed (aka gspeed) (shifted by 16, aka 1.0 = 0x10000)

Player.XVelocity, the player’s current X Velocity (aka xspd) (shifted by 16, aka 1.0 = 0x10000)

Player.YVelocity, the player’s current Y Velocity (aka yspd) (shifted by 16, aka 1.0 = 0x10000)

Player.Gravity, if the player is in the air or not, 0 = on ground, 1 = in air

Player.Angle, the angle the player is on, based on tile angles so values range from 0-255

Player.Skidding, the player’s skidding timer

Player.Pushing, the player’s pushing flag, set to 2 if the player should be in pushing state, set and managed by ProcessPlayerTileCollision()

Player.TrackScroll, track scroll flag, if set the camera will ignore the 16-pixel max scroll and track the player, rather than follow it

Player.Up, set to true if Up input was held

Player.Down, set to true if Down input was held

Player.Left, set to true if Left input was held

Player.Right, set to true if Right input was held

Player.JumpPress, set to true if any jump buttons were pressed on this frame

Player.JumpHold, set to true if any jump buttons were held down on this frame

Player.FollowPlayer1, set to true if the player should follow player 1 (never used in CD)

Player.LookPos, player look pos, used for looking up/down, camera will be offset by lookPos when following the player

Player.Water, set if the player enters the water (never used in CD, since it checks gravityStrength instead)

Player.TopSpeed, the player’s top speed value

Player.Acceleration, the player’s acceleration value

Player.Deceleration, the player’s deceleration value

Player.AirAcceleration, the player’s air acceleration value

Player.AirDeceleration, the player’s air deceleration value

Player.GravityStrength, the player’s gravity strength (how fast the player will fall)

Player.JumpStrength, the player’s jump strength

Player.JumpCap, the max the player can jump

Player.RollingAcceleration, the player’s rolling acceleration

Player.RollingDeceleration, the players rolling deceleration

Player.EntityNo, the player’s slot id

Player.CollisionLeft, the player’s collision left (based on the active frame’s hitbox)

Player.CollisionTop, the player’s collision top (based on the active frame’s hitbox)

Player.CollisionRight, the player’s collision right (based on the active frame’s hitbox)

Player.CollisionBottom, the player’s collision bottom (based on the active frame’s hitbox)

Player.Flailing[index], the player’s flailing flags, indexed 0-2, set to true if there wasn’t a collision on the surface the player is on, used for balance anims

Player.Timer, the player’s timer, used for anything a timer is needed for

Player.TileCollisions, the player’s tile collision flag, if false ProcessPlayerTileCollisions() won’t interact with any tiles

Player.ObjectInteraction, the player’s object interaction flag, if false other object’s wont run the PlayerInteraction sub

Player.Visible, player visibility flag, if false the player wont be drawn

Player.Rotation, the player’s rotation value, goes from 0-511 to match the Sin function, used by drawing if fxType is FX_ROTATE

Player.Scale, the player’s scale value, scales both X & Y, scale is 9 bit shifted so 1.0 == 0x200, 0.5 == 0x100, etc

Player.Priority, the player’s priority state, determines how the player manages update priority, valid values are: PRIORITY_ACTIVE_BOUNDS = 0 (object will update if its within 128 pixels of the screen borders), PRIORITY_ACTIVE = 1 (object will always update, unless paused), PRIORITY_ACTIVE_PAUSED = 2 (object will always update, even if paused), PRIORITY_ACTIVE_XBOUNDS = 3 (object will update if its XPos is within 128 pixels of the horizontal screen borders, no matter where the YPos is), PRIORITY_ACTIVE_BOUNDS_REMOVE = 4 (the same as bounds, but if the bounds check fails, the object is destroyed, used for things like debug mode objects), PRIORITY_INACTIVE = 5 (object never updates).

once again these aliases are not supported by the engine natively so the raw values will have to be used instead, they are provided here for ease of understanding

Player.DrawOrder, the player's draw Order, this determines what drawList the player will go in after objectMain

Player.Direction, the player's direction flag, used for FX_FLIP, and general direction management, usually 0-3 (FLIP_NONE, FLIP_X, FLIP_Y & FLIP_XY respectively)

Player.InkEffect, the player's inkEffect flag, used with FX_INK. 0 = no ink effect, 1 = blending ink effect (think about 0.5 alpha), 2 = alpha blending ink effect, object transparency will be based on alpha value of 0-255, 3 = additive ink effect, works like alpha, but applies an additive blending instead of an alpha blending effect, 4 = subtractive, works like alpha, but applies a subtractive blending effect instead of an alpha one

Player.Alpha, the player's transparency value, used with the correct inkEffect modes, goes from 0-255, can be higher but the engine caps most alpha effects at 255

Player.Frame, the player's frame ID, used with loaded animations, but also used to store spriteframe ids

Player.Animation, the player's animation id, only really used when an animation is loaded

Player.PrevAnimation, the player's previous animation, set and managed by ProcessAnimation()

Player.AnimationSpeed, the player's animation speed, used and managed by ProcessAnimation(), though some objects use it to handle their own anims

Player.AnimationTimer, the object's animation timer, also used and managed by ProcessAnimation(), though again some objects use and manage their animations with this themselves

Player.Value0, player storage value 0, used for storing values long term in player data, can be used for whatever, managed on a player-by-player basis

Player.Value1, player storage value 1, used for storing values long term in player data, can be used for whatever, managed on a player-by-player basis

Player.Value2, player storage value 2, used for storing values long term in player data, can be used for whatever, managed on a player-by-player basis

Player.Value3, player storage value 3, used for storing values long term in player data, can be used for whatever, managed on a player-by-player basis

Player.Value4, player storage value 4, used for storing values long term in player data, can be used for whatever, managed on a player-by-player basis

Player.Value5, player storage value 5, used for storing values long term in player data, can be used for whatever, managed on a player-by-player basis

Player.Value6, player storage value 6, used for storing values long term in player data, can be used for whatever, managed on a player-by-player basis

Player.Value7, player storage value 7, used for storing values long term in player data, can be used for whatever, managed on a player-by-player basis

Player.Value8, player storage value 8, used for storing values long term in player data, can be used for whatever, managed on a player-by-player basis

Player.Value9, player storage value 9, used for storing values long term in player data, can be used for whatever, managed on a player-by-player basis

Player.Value10, player storage value 10, used for storing values long term in player data, can be used for whatever, managed on a player-by-player basis

Player.Value11, player storage value 11, used for storing values long term in player data, can be used for whatever, managed on a player-by-player basis

Player.Value12, player storage value 12, used for storing values long term in player data, can be used for whatever, managed on a player-by-player basis

Player.Value13, player storage value 13, used for storing values long term in player data, can be used for whatever, managed on a player-by-player basis

Player.Value14, player storage value 14, used for storing values long term in player data, can be used for whatever, managed on a player-by-player basis

Player.Value15, player storage value 15, used for storing values long term in player data, can be used for whatever, managed on a player-by-player basis

Player.OutOfBounds, the player’s out of bounds flag, will be true if the player is not on screen

Stage.State, the stage’s state, valid values are: STAGE_RUNNING = 1, STAGE_PAUSED = 2

Stage.ActiveList, Active Category, values 0-3 are valid

Stage.ListPos, Active scene id

Stage.TimeEnabled, if true, time is enabled and Stage.Milliseconds/Seconds/Minutes will hold the time since stage was started

Stage.MilliSeconds, how many milliseconds since the last second

Stage.Seconds, how many seconds since the last minute

Stage.Minutes, how many minutes have elapsed

Stage.ActNo, stage’s act ID, determined by the “Scene ID” flag for the stage in the gameconfig,

Stage.PauseEnabled, if pausing is allowed or not

Stage.ListSize, how many scenes are in the active stage list

Stage.NewXBoundary1, the new stage boundary to the left regular boundary to the left will be based on

Stage.NewXBoundary2, the new stage boundary to the right regular boundary to the right will be based on

Stage.NewYBoundary1, the new stage boundary upwards regular boundary upwards will be based on

Stage.NewYBoundary2, the new stage boundary downwards regular boundary downwards will be based on

Stage.XBoundary1, the stage’s boundary to the left, in pixels, relative to 0,0 in the stage

Stage.XBoundary2, the stage’s boundary to the right, in pixels, relative to 0,0 in the stage

Stage.YBoundary1, the stages boundary upwards, in pixels, relative to 0,0 in the stage

Stage.YBoundary2, the stages boundary downwards, in pixels, relative to 0,0 in the stage

Stage.DeformationData0[index], deformation data for layer 0 (FG) above the water level

Stage.DeformationData1[index], deformation data for layer 0 (FG) below the water level

Stage.DeformationData2[index], deformation data for layers 1-8 (BG) above the water level

Stage.DeformationData3[index], deformation data for layer 1-8 (BG) below the water level

Stage.WaterLevel, the water level’s height in pixels, relative to 0,0 in the stage

Stage.ActiveLayer[index], indexed 0-3, what tile layer IDs are active draw layers

Stage.MidPoint, the tile layer midPoint (what activeLayer and above tiles count as being on the “high plane”

Stage.PlayerListPos, player list position (playerID, based on gameconfig list)

Stage.ActivePlayer, active player object ID (almost always 0)

Screen.CameraEnabled, if set the camera is active and will follow the target

Screen.CameraTarget, camera’s target object ID

Screen.CameraStyle, camera follow style (0 = regular sonic style, 1,2 & 3 = extended camera, 4 = horizontally locked camera)

Screen.DrawListSize[index], how many entries are in the drawList *index*

Screen.CenterX, screen center on the x axis

Screen.CenterY, screen center on the y axis (almost always 120)

Screen.XSize, screen width

Screen.YSize, screen height (almost always 240)

Screen.XOffset, how far the camera has scrolled on the X axis

Screen.YOffset, how far the carmera has scrolled on the Y Axis

Screen.ShakeX, camera shake X amount

Screen.ShakeY, camera shake Y amount

Screen.AdjustCameraY, camera adjust on the y axis

TouchScreen[index].Down, is the selected touch input holding down

TouchScreen[index].XPos, the XPos of the selected touch input relative to the screen

TouchScreen[index].YPos, the YPos of the selected touch input relative to the screen

Music.Volume, current master volume for music

Music.CurrentTrack, currently playing music track ID

KeyDown.Up, set if the up button is held

KeyDown.Down, et if the down button is held

KeyDown.Left, set if the left button is held

KeyDown.Right, set if the right button is held

KeyDown.ButtonA, set if the A button is held

KeyDown.ButtonB, set if the B button is held

KeyDown.ButtonC, set if the C button is held

KeyDown.Start, set if the start button is held

KeyPress.Up, set if the up button is pressed on this frame

KeyPress.Down, set if the down button is pressed on this frame

KeyPress.Left, set if the left button is pressed on this frame

KeyPress.Right, set if the right button is pressed on this frame

KeyPress.ButtonA, set if the A button is pressed on this frame

KeyPress.ButtonB, set if the B button is pressed on this frame

KeyPress.ButtonC, set if the C button is pressed on this frame

KeyPress.Start, set if the start button is pressed on this frame

Menu1.Selection, the selection of MENU_1

Menu2.Selection, the selection of MENU_2

TileLayer[index].XSize, width of the tile layer in chunks

TileLayer[index].YSize, height of the tile layer in chunks

TileLayer[index].Type, tile layer type (0 = invalid, 1 = hScroll, 2 = vScroll, 3 = 3d Floor, 4 = 3d Sky)

TileLayer[index].Angle, tile layer angle (0-511)

TileLayer[index].XPos, tile layer X position (shifted by 16, aka 1.0 = 0x10000)

TileLayer[index].YPos, tile layer Y position (shifted by 16, aka 1.0 = 0x10000)

TileLayer[index].ZPos, tile layer Z position (shifted by 16, aka 1.0 = 0x10000)

TileLayer[index].ParallaxFactor, tile layer parallax factor (relative speed) (shifted by 16, aka 1.0 = 0x10000)

TileLayer[index].ScrollSpeed, tile layer scroll speed (constant speed) (shifted by 16, aka 1.0 = 0x10000)

TileLayer[index].ScrollPos, tile layer scroll offset (shifted by 16, aka 1.0 = 0x10000)

TileLayer[index].DeformationOffset, tile layer deformation offset (used with deformationData0 or 2)

TileLayer[index].DeformationOffsetW, tile layer deformation offset (used with deformationData1 or 3)

HParallax.ParallaxFactor[index], hParallax parallax factor (relative speed) (shifted by 16, aka 1.0 = 0x10000)

HParallax.ScrollSpeed[index], hParallax scroll speed (constant speed) (shifted by 16, aka 1.0 = 0x10000)

HParallax.ScrollPos[index], hParallax scroll offset (shifted by 16, aka 1.0 = 0x10000)

VParallax.ParallaxFactor[index], vParallax parallax factor (relative speed) (shifted by 16, aka 1.0 = 0x10000)

VParallax.ScrollSpeed[index], vParallax scroll speed (constant speed) (shifted by 16, aka 1.0 = 0x10000)

VParallax.ScrollPos[index], vParallax scroll offset (shifted by 16, aka 1.0 = 0x10000)

3DScene.NoVertices, number of active vertices in the vertesBuffer

3DScene.NoFaces, number of active faces in the faceBuffer

VertexBuffer[index].x, vertex x position (shifted by 8, aka 1.0 = 0x100)

VertexBuffer[index].y, vertex y position (shifted by 8, aka 1.0 = 0x100)

VertexBuffer[index].z, vertex z position (shifted by 8, aka 1.0 = 0x100)

VertexBuffer[index].u, vertex u position (shifted by 8, aka 1.0 = 0x100)

VertexBuffer[index].v, vertex v position (shifted by 8, aka 1.0 = 0x100)

FaceBuffer[index].a, vertex 1 index

FaceBuffer[index].b, vertex 2 index

FaceBuffer[index].c, vertex 3 index

FaceBuffer[index].d, vertex 4 index

FaceBuffer[index].Flag, vertex draw flag. Valid values (FLAG_TEXTURED_3D = 0 (textured and rendered in 3D), FLAG_TEXTURED_2D = 1 (textured and rendered in 2D), FLAG_COLOURED_3D = 2 (flat colour and rendered in 3D), FLAG_COLOURED_2D = 3 (flat colour and rendered in 2D), aliases not available in engine, here for understanding purposes

FaceBuffer[index].Color, vertex color (RGBA format)

3DScene.ProjectionX, projection X (aka fake resolution X)

3DScene.ProjectionY, projection Y (aka fake resolution Y)

Engine.State, engine's internal gameMode, should almost always be 1 unless something is terribly wrong

Stage.DebugMode, flag that determines if the stage is in debug mode or not

Engine.Message, engine's callback message (generally unknown what these values are)

SaveRAM[index], saveRAM array, anything stored in here will be written to "SData.bin" when WriteSaveRAM() is called, likewise it'll be overwritten when ReadSaveRam() is called

Engine.Language, the engine's current language, valid values are: RETRO_EN = 0, RETRO_FR = 1, RETRO_IT = 2, RETRO_DE = 3, RETRO_ES = 4, RETRO_JP = 5, aliases are not provided by engine and are here for understanding purposes only

Object[index].SpriteSheet, object's spritesheet ID

Engine.OnlineActive, flag that is set if online services are avaliable

Engine.FrameSkipTimer, the engine’s frame skip timer

Engine.FrameSkipSetting, the engine’s frame skip setting

Engine.SFXVolume, soundFX volume (0-100)

Engine.BGMVolume, bgm volume (0-100), combined with music.volume to get the final output volume

Engine.PlatformID, the game’s current platform ID. Valid values: RETRO_WIN = 0, RETRO_OSX = 1, RETRO_XBOX_360 = 2, RETRO_PS3 = 3, RETRO_iOS = 4, RETRO_ANDROID = 5, RETRO_WP7 = 6

Engine.TrialMode, flag that is set if the engine is in trial mode, instead of a full release

KeyPress.AnyStart, set if any key is pressed, only used on the title screen and possibly only available on title(?)

Engine.HapticsEnabled, (not available on PC port) set to true if haptics are enabled

Misc Notes/Tips:

- Windows calculator has a “programmer’s calculator” mode which is really helpful when dealing with large bit shifted positions/values
- If an object’s priority check fails, objectMain, objectPlayerInteraction & objectDraw all wont be called until the check succeeds
- TaxReceipt doesn’t have a very smart compiler so everything has to be declared above where its used, you cant use “Player_Func0” before its declared
- TaxReceipt also has TypeName[] as an alias, where using TypeName[objName], with the object name in the brackets will act the same as using the object’s type id, useful for when object types are getting shuffled around often
- It is recommended to read through CD’s existing scripts and play around with them before going and creating fully original content
- Restarting a stage via the dev menu will reload & recompile all scripts (as well as all other stage assets)
- TaxReceipt scripts can have any extension, however they absolutely **MUST** use ASCII formatting **AND** windows CRLF line ending styles otherwise they will not be parsed correctly

Further Help/Contact:

Join the Retro Engine Modding Server (REMS), for general questions and support: **https://dc.railgun.works/retroengine**

I have a discord account which I check fairly recently which I will answer any questions in case you can't join rems for some reason: Rubberduckycooly#6438