

# RetroScript

# Handbook v4

*By Rubberduckycooly + RMGRich*

*Last updated: Feb 2nd 2021*

# Table of Contents

- [Introduction to RSDK and RetroScript](#)
  - o [About RSDK](#)
  - o [About RetroScript](#)
- [Arithmetic](#)
  - o [Mathematics](#)
- [Conditionals and Statements](#)
  - o [Boolean Logic](#)
  - o [Control Statements](#)
- [Subs and Functions](#)
  - o [Subs](#)
  - o [Functions](#)
- [Preprocessor Directives](#)
- [Built-ins](#)
  - o [Audio](#)
  - o [Drawing](#)
  - o [Palettes](#)
  - o [Object](#)
  - o [Stages](#)
  - o [Input](#)
  - o [Math](#)
  - o [3D](#)
  - o [Menus](#)
  - o [Engine](#)
- [Further Assistance](#)

# Introduction to RSDK and RetroScript

## About RSDK

The Retro Engine Software Development Kit (Retro-Engine or RSDK) is a primarily 2D game engine with many “old school” graphics effects, including functionality akin to “Mode 7” on an SNES and palette-based graphics. RSDKv3 (previously thought to be RSDKv2<sup>1</sup>), the 3rd version, was only used in the Sonic CD (2011) remaster (with a slight update for the mobile port of which will be addressed later) and was then upgraded to RSDKv4 (previously thought to be RSDKvB<sup>1</sup>) for the Sonic 1 and 2 mobile remasters (and likely [the Sonic 3 proof-of-concept](#)), using an updated version of RetroScript with more built-ins. Mania uses RSDKv5, the latest officially used version of RSDK, which uses a transpilable version of RetroScript<sup>2</sup>. Versioning for RSDK has followed the editor’s version since v3<sup>3</sup>. RetroScript remains officially unnamed, though it was previously confused with TaxReciept<sup>1</sup>.

---

<sup>1</sup> Christian Whitehead’s reply to RDC’s tweet: <https://twitter.com/CFWhitehead/status/1341701486657433601>

<sup>2</sup> CW has stated that v5 scripts get transpiled into C for use in the Game.dll file.

<sup>3</sup> When asked why Nexus and CD was named v3, CW stated that as of v3, the engine versions began to match the editor’s.

## About RetroScript

RetroScript's syntax is like that of Visual Basic. It does not use semicolons or braces and instead uses line breaks to mark expression endings. Because of it being a scripting language, it offers many benefits compared to a typical language such like C:

- Scripts are recompiled when a stage is loaded/restarted
  - Changes are incredibly easy to make and test almost instantly
- Specifically designed to create object code, making it easy to create objects

However, because of this, there are also many drawbacks which add a challenge to more experienced programmers:

- Custom variables cannot be defined. One must use the temporary built-in variables (discussed later in the handbook.)
- There are no data types other than integers. No decimal places (floats) or strings can be stored, except for passing some string constants to some built-in functions.
- User-defined functions cannot be passed any parameters. All variables are however kept the same, so it is possible to use the built-in variables as a “passing” method.
- You cannot have multiple expressions on one line. For example,  $A = B + C$  is invalid, but  $A = B$  then  $A += C$  is valid (discussed more in the next page).

# Arithmetic

## Mathematics

As previously mentioned, you cannot have more than 1 arithmetic expression in one line and they all must be done one by one. There can only ever be 1 variable on the right and another on the left. Because of this, the list of mathematical arithmetic operators is limited to the following assignment operators:

- = - regular assignment
- 4-function
  - +=
  - -=
  - \*=
  - /= - division rounds *down* (flooring)
  - %= - modulo (used for remainder of division)
- Bit math
  - <<= - shift left
  - >>= - shift right
  - &= - AND
  - |= - OR
  - ^= - XOR
- Unary
  - ++ - used as `Variable++`, equivalent to `Variable += 1`
  - -- - used as `Variable--`, equivalent to `Variable -= 1`

## Examples

Pseudo-code	RetroScript (with custom variables)
<pre>i = 0; j = 15; i++;      //i is 1 i = j + 2; //i is 17</pre>	<pre>i = 0 j = 15 i++      //i is 1 i = j    //i is 15 i += 2   //i is 17</pre>
<pre>x = 19; y = 3; d = 5; x -= --d; //x subtracted by 4 y -= d--; //y subtracted by 4           //d is already 3</pre>	<pre>x = 19 y = 3 d = 5 d-- x -= d //x subtracted by 4 y -= d //y subtracted by 4 d--    //d is now 3</pre>
<pre>i = 2; i = i + 0.5; //i is 2.5</pre>	<pre>i = 2 i += 0.5 //oops! compiler error!           //if decimals were allowed,           //i would be 2.5</pre>

# Conditionals and Statements

## Boolean Logic

Boolean operation is also possible but can only be used in control statements, and thus why they are in this section. There is no such boolean “or” or boolean “and” operator ( | | and && respectively). The list of operators are as follows:

- == - equal to (not = on its own)
- >
- >=
- <
- <=
- !=

There are, however, some functions that you can use to assign variables boolean expressions:

- CheckEqual(A, B)
- CheckLower(A, B)
- CheckGreater(A, B)
- CheckNotEquals(A, B)

All these set `CheckResult` to either 0 or 1 based on the result of the function, which can later be checked and ORed/ANDed with.

# Control Statements

Since RetroScript does not use braces, there are specific keyword pairs that get used, along with small specifics for each:

- If statements:
  - `if [statement]` - `[statement]` is a single boolean expression as shown above
  - `else`
  - `endif` - use as the “ending brace”
  - **There is no such thing as a direct else-if in RetroScript.** To achieve an else-if, one must make a new if statement on a new line and close it properly.
- While statements:
  - `while [statement]` - `[statement]` is a single boolean expression as shown above
  - `loop` - use as the “ending brace”
- Switch statements:
  - `switch [variable]` - `[variable]` is the variable to check for
  - `case [int/alias]` - `[int/alias]` is an integer or alias to check if the variable is equal to
  - `endswitch` - use as the “ending brace”
  - Switches behave similarly as they do in C: `default` is optional and `break` is used in cases to stop fallthrough.
- Foreach statements:
  - `foreach (TypeName[objectName], store, type)`
    - iterates every object of type `TypeName[objectName]` and sets `store` to the object's slotID.
    - `type` is either `ALL_ENTITIES` or `ACTIVE_ENTITIES`
  - `next` - use as the “ending brace”



## Examples

### Pseudo-code

```
if (i == 0) {  
    x++;  
    y++;  
} else if (i == 1) {  
    x--;  
}  
}
```

```
while (x < 10) {  
    x++;  
    if (y == 5) break;  
}
```

```
switch (x) {  
    case 1:  
    case 2:  
        y++;  
    case 3:  
        x++;  
        break;  
    default:  
        z++;  
        break;  
}
```

### RetroScript (with custom variables)

```
if i == 0  
    x++  
    y++  
else  
    if i == 1  
        x--  
    endif  
endif
```

```
while x < 10  
    x++  
    if y == 5  
        break  
    endif  
loop
```

```
switch x  
case 1  
case 2  
    y++  
case 3  
    x++  
    break  
default  
    z++  
    break  
endswitch
```

### Pseudo-code

```
foreach (ArrayPos2 in TypeName[Ring]) {  
    Object[ArrayPos2].Value0 = 1;  
}  
  
foreach (ArrayPos2 in TypeGroup[256]) {  
    Object[ArrayPos2].XPos = Object.XPos;  
    Object[ArrayPos2].YPos = Object.YPos;  
}
```

### RetroScript (with custom variables)

```
foreach TypeName[Ring], ArrayPos2  
    Object[ArrayPos2].Value0 = 1  
floop  
  
foreach TypeGroup[256], ArrayPos2  
    Object[ArrayPos2].XPos = Object.XPos  
    Object[ArrayPos2].YPos = Object.YPos  
floop
```

# Events and Functions

## Events

Events are easily thought of as “default functions,” and are all called periodically during gameplay. To define events, you use event [name] as the start and end event as the “closing brace”. The definable events are as follows:

Sub	Description
ObjectMain	Called once every frame per object if priority allows for it [see priority notes]
ObjectDraw	Called once every frame per object if priority allows for it [see priority notes]. The ordering is based the value of <code>Object.DrawOrder</code>
ObjectStartup	Called once per object type and once when the stage loads. Used for loading assets and spriteFrames
RSDKDraw	similar to <code>ObjectDraw</code> , though only called by the editor (RetroED v2), called once a frame for each object
RSDKLoad	similar to <code>ObjectStartup</code> , though only called by the editor (RetroED v2), used to load any assets/sprite frames needed in <code>RSDKDraw</code>

# Functions

Users can define functions by using `function [name]` to start a function and `end function` as the “closing brace.” Functions can be forward declared using the preprocessor directive `reserve function [name]`. To call functions, you use the built in function `CallFunction(function)`, which means functions cannot have built in parameters, but there are ways to get around it in the example below. `return` can be used to preemptively end a function.

## Examples

Pseudo-code	RetroScript (with custom variables)
<pre>MyFunc(y);  ObjectMain() {     x += 5; //x is 5     MyFunc(x) //pass x (not it's value)     //x is 7 }  MyFunc(y) {     y += 2; //increment x     return;     y += 5; //this line doesn't hit }</pre>	<pre>reserve function MyFunc  event ObjectMain     x += 5     y = x     CallFunction(MyFunc) end event  function MyFunc     y += 2     return     y += 5 //this line doesn't hit end function</pre>

# Preprocessor Directives

RetroScript v4 has 1 preprocessor directive that is available to use. this preprocessor directives are as follows:

Directive	Description
<code>#platform: [type]</code> <code>#endplatform</code>	Skips over lines of code if type does not match with what the bytecode is being compiled for. type can be: <ul style="list-style-type: none"><li>• STANDARD or MOBILE</li><li>• SW_RENDERING or HW_RENDERING</li><li>• USE_F_FEEDBACK or NO_F_FEEDBACK</li></ul>

## Variables

RetroScript v4 has 3 formats for extra variables that are available to use. These use the keywords `public` and `private`. `public` means this variable can be accessed by any script compiled after the current one, while `private` means the variable can only be accessed by the script it was created in. the formats for the variables are as follows:

Directive	Description
<pre>[public]/[private] alias [val] : [name]</pre>	<p>Creates a new alias that gets replaced by <code>val</code> on compile time.</p> <p>example:</p> <pre>private alias 1 : myAlias</pre>
<pre>[public]/[private] value [name] = [val];</pre>	<p>Creates a new static variable with the value of <code>val</code>.</p> <p>example:</p> <pre>public value myValue = 0;</pre> <p>static variables are not tied to an object and thus should not be used when a value is needed for every instance of an object. they are regular values that can be accessed the same as any other built-in one</p>
<pre>[public]/[private] table [name] [values] end table</pre>	<p>Creates a new table, fills the table with any values it reads until it hits the <code>`end table`</code> keyword. Values should be separated by <code>``,`</code> character, unless there is a newline, then there should not be a <code>``,`</code> separator.</p> <p>example:</p> <pre>public table colourTable     0x600020, 0xC00040, 0xE04080     0x802040, 0xE04060, 0xE060A0     0xA04060, 0xE06080, 0xE080C0</pre>

```
end table
```

tables are closer to functions than variables, as values from them are accessed via ``GetTableValue(store, index, table)`` and can be set via ``SetTableValue(value, index, table)`` like functions their ids can be assigned to other variables for “pointer-like” functionality

example:

```
GetTableValue(temp0, temp1, myTable)
```

```
temp0 += 0x10
```

```
SetTableValue(temp0, temp1, myTable)
```

example 2:

```
switch (temp1)
```

```
case 0
```

```
    temp0 = myTable1
```

```
    break
```

```
case 1
```

```
    temp0 = myTable2
```

```
    break
```

```
end switch
```

```
GetTableValue(temp2, 2, temp0)
```





# Built-ins

## Audio

Function/Variable/Alias	Description
<code>music.volume</code>	Current master volume for music
<code>music.currentTrack</code>	Currently playing music track ID
<code>music.position</code>	Position of currently playing music
<code>engine.bgmVolume</code>	Sound FX Volume (ranges from 0-100)
<code>engine.sfxVolume</code>	BGM volume (ranges from 0-100), combined with <code>Music.Volume</code> to get the final output volume
<code>SetMusicTrack(string filePath, int trackID, int loopPoint)</code>	Loads the music file (has to be ogg format) from <code>Data/Music/[filePath]</code> into the <code>trackList</code> slot <code>trackID</code> , with a loop point of <code>loopPoint</code> (0 = no loop, 1 = loop from start, anything else is the sample to loop from)
<code>PlayMusic(int trackID)</code>	Plays the music track loaded into the slot <code>trackID</code>
<code>StopMusic()</code>	Stops the currently playing music track

<code>PauseMusic()</code>	Pauses the currently playing music track
<code>ResumeMusic()</code>	Resumes the music track that was paused using <code>PauseMusic()</code>
<code>SwapMusicTrack(string filePath, int trackID, int loopPoint, int ratio)</code>	Works similar to <code>SetMusicTrack()</code> & <code>PlayMusic()</code> but starts at a position based on ratio. ratio is using an 8000-based value, so 8000 = 1.0 music speed, 4000 = 0.5, etc. Used more commonly with speed shoes.
<code>SfxName[name]</code>	Use this to get the ID of an SFX based on it's name. (e.x Jump.wav has an sfxID of 0, so using <code>SfxName[Jump]</code> would be the same as using 0
<code>PlaySfx(int sfx, int loopCnt)</code>	Plays the sfx with index of sfx in gameconfig + stageconfig loopCnt times (recommended to use <code>SfxName[]</code> )
<code>StopSfx(int sfx)</code>	Stops the sfx with index of sfx in gameconfig + stageconfig (recommended to use <code>SfxName[]</code> )
<code>SetSfxAttributes(int sfx, int loopCnt, int pan)</code>	Sets the amount of times for sfx to loop to loopCnt (-1 to leave it unchanged) and the panning of sfx to pan (-100 to 100 for left to right, with 0 being balanced)

# Drawing

Function/Variable/Alias	Description
<code>LoadSpriteSheet(string path)</code>	Loads a spritesheet from <code>Data/Sprites/[path]</code> and sets <code>Object.SpriteSheet</code> to the sheet's ID
<code>RemoveSpriteSheet(string path)</code>	Removes a sheet that matches path if it exists
<code>SpriteFrame(int pivotX, int pivot, int width, int height, int sprX, int sprY)</code>	Creates a spriteframe with the specified values
<code>EditFrame(int frame, int pivotX, int pivot, int width, int height, int sprX, int sprY)</code>	Sets spriteframe frame to the new values
<code>DrawSprite(int frame)</code>	Draws sprite frame at the object's X and Y position
<code>DrawSpriteXY(int frame, int XPos, int YPos)</code>	Draws sprite frame to the specified X and Y position If using <code>DrawSpriteScreenXY</code> , the position is in screen-space (0, 0 is top left, 0, 1 is 1 px to the right, etc), otherwise the position is in world-space (0, 0 is top left, but 0, 0x10000 is 1px to the right)

<code>DrawSpriteScreenXY(int frame, int XPos, int YPos)</code>	
<code>FX_SCALE</code> <code>FX_ROTATE</code> <code>FX_ROTOTOZOOM</code> <code>FX_INK</code> <code>FX_FLIP</code>	IDs to be used for <code>DrawSpriteFX</code> and <code>DrawSpriteScreenFX</code> below. For <code>FX_INK</code> , see the IDs near
<code>DrawSpriteFX(int frame, int fx, int XPos, int YPos)</code>  <code>DrawSpriteScreenFX(int frame, int fx, int XPos, int YPos)</code>	Draws sprite frame to the specified X and Y position using the specified FX mode If using <code>DrawSpriteScreenFX</code> , the position is in screen-space (0, 0 is top left, 0, 1 is 1 px to the right, etc), otherwise the position is in world-space (0, 0 is top left, but 0, 0x10000 is 1px to the right)
<code>DrawTintRect(int XPos, int YPos, int width, int height)</code>	Draws a tint rect with a size of width, height at XPos & YPos relative to screen-space
<code>DrawNumbers(int startingFrame, int XPos, int YPos, int value, int digitCnt, int spacing, int showAllDigits)</code>	Draws values using startingFrame as the starting point at XPos & YPos (screen-space), with spacing pixels between each frame. Will only draw valid digits (or digitCnt digits if number is exceeded) if showAllDigits is 0, otherwise digitCnt digits will be drawn, with extras being 0

<code>DrawActName(int startingFrame, int XPos, int YPos, int align, int unknown, int unknown2, int spacing)</code>	Draws the loaded stage's act name using 26 frames starting from startingFrame (only uppercase english letters are supported), at XPos & YPos (screen-space), using alignment to determine where the text center is (0 = left, 1 = middle, 2 = right), with spacing pixels between each letter
<code>DrawRect(int XPos, int YPos, int width, int height, int R, int G, int B, int A)</code>	Draws a rect with a size of width, height at XPos & YPos (screen-space), with a colour of R, G, B and with an alpha of A
<code>LoadAnimation(string filePath)</code>	Loads an animation from Data/Animations/[filePath] for the object to use
<code>DrawObjectAnimation()</code>	Draws the object at its X and Y position, based on the loaded animation and <code>Object.Frame/Object.Animation</code>
<code>ClearDrawList(int layer)</code>	Removes all entries in drawList layer
<code>AddDrawListEntityRef(int layer, int objectPos)</code>	Adds objectPos to the drawList layer
<code>GetDrawListEntityRef(var store, int layer, int objectPos)</code>	Gets the value in drawList layer at objectPos and stores it in store
<code>SetDrawListEntityRef(int value, int layer, int objectPos)</code>	Sets the value in drawList layer at objectPos to the value of value

# Palettes

Function/Variable/Alias	Description
<code>LoadPalette(string filePath, int palID, int startPalIndex, int startIndex, int endIndex)</code>	Loads a palette from Data/Palettes/[filePath] into palID starting from startPalIndex, with a file offset of startIndex and reading all colors through to endIndex
<code>RotatePalette(int palID, int startIndex, int endIndex, int right)</code>	Rotates all colours in palID starting from startIndex through to endIndex left one index if right is 0, else rotates one right
<code>SetScreenFade(int r, int g, int b, in a)</code>	Sets the fade out effect based on r, g, b and a
<code>SetActivePalette(int palID, int startLine, int endLine)</code>	Sets the active palette to palID for all lines from startLine through to endLine
<code>SetPaletteEntry(int palID, int index, int colour)</code>	Sets the palette entry in palID at index to the value of `colour`

GetPaletteEntry(int palID, int index, int store)	Gets the palette entry from palID at index and stores it in store
SetPaletteFade(int dstPal, int srcPalA, int srcPalB, int blendAmount, int startIndex, int endIndex)	Blends srcPalA with srcPalB by blendAmount amount, starting at palette index startIndex and continuing through to endIndex and stores the resulting colours in dstPal
CopyPalette(int srcPal, int srcPalStart, int dstPal, int dstPalStart, int count)	Copies count colours from srcPal, starting at srcPalStart, to dstPal, starting at dstPalStart
ClearScreen(int clrIndex)	Clears all pixels on screen with the colour from clrIndex in the active palette

# Object

**A NOTE ABOUT index:** appending a + or - to an array value or a constant will offset it + or - from that value or constant from the object's object position. [index] is also optional, and not including it will reference the current object.

Function/Variable/Alias	Description
temp0 temp1 temp2 ... temp7	Temporary values used to store values during arithmetic or other similar operations
arrayPos0 arrayPos1 ... arrayPos5 arrayPos6 arrayPos7	Variables used for storing indexes to be used with arrays.
tempObjectPos	Set when CreateTempObject() is called, can only be used as an arrayPos
CreateTempObject(int type, int propertyValue, int XPos, int YPos)	Creates a temporary object specified by type, propertyValue, XPos and YPos near the end of the object list and sets TempObjectPos to the created object's slotID. This should only be used for misc objects like FX and objects that are destroyed quickly



<code>ResetObjectEntity(int slot, int type, int propertyValue, int XPos, int YPos)</code>	Resets the object at slot to the type and position specified by type, propertyValue, XPos and YPos
<code>checkResult</code>	A value that some functions set as the resulting value. Can be used with all sorts of arithmetic
<code>object[index].value0</code> <code>object[index].value1</code> <code>object[index].value2</code> <code>...</code> <code>object[index].value47</code>	Integer values used for long-term storage. What they are used for varies on an object-by-object basis.
<code>object[index].entityPos</code>	The object's slot in the object list
<code>object[index].groupID</code>	The object's typeGroup. By default, it matches its type, but can be set to another one (0x100, 0x101 & 0x102 are never assigned by default so they're good for using for custom groups)
<code>object[index].type</code>	The object's type
<code>object[index].propertyValue</code>	The object's propertyValue (subtype)
<code>object[index].xpos</code> <code>object[index].ypos</code>	The object's position in world-space (0x10000 (65536) == 1.0)

object[index].ixpos object[index].iypos	The object's position in screen-space, truncated down from XPos (1 == 1)
object[index].xvel object[index].yvel	The object's speed on the X & Y axis (world-space)
object[index].speed	The object's general speed (world-space)
object[index].state	The object's state. Can be used any way the objects needs
object[index].rotation	The object's rotation, generally used with DrawSpriteFX and FX_ROTATE or FX_ROTOTOZOOM (ranges from 0-511)
object[index].scale	The object's scale, generally used with generally used with DrawSpriteFX and FX_ROTATE or FX_ROTOTOZOOM Uses a 9-bit bitshifted value, so $0x200 \ (512) == 1.0$
PRIORITY_ACTIVE_BOUNDS PRIORITY_ACTIVE PRIORITY_ACTIVE_PAUSED PRIORITY_XBOUNDS PRIORITY_XBOUNDS_DESTROY PRIORITY_INACTIVE PRIORITY_BOUNDS_SMALL PRIORITY_UNKNOWN	IDs for Object.Priority. PRIORITY_ACTIVE_BOUNDS: object will update as long as it's within 0x80 pixels of the screen border left/right and 0x100 pixels up/down PRIORITY_ACTIVE: object will always update, unless paused (or frozen) PRIORITY_ACTIVE_PAUSED: object will always update, even if paused (or frozen) PRIORITY_XBOUNDS: same as PRIORITY_ACTIVE_BOUNDS however there's no Y check so as long as it's within XBounds it'll update

	<p>PRIORITY_XBOUNDS_DESTROY: same as PRIORITY_ACTIVE_XBOUNDS, except if the check fails the object's type will be set to blank object</p> <p>PRIORITY_INACTIVE: never updates or draws</p> <p>PRIORITY_BOUNDS_SMALL: object will update as long as it's within 0x20 pixels of the screen border left/right and 0x80 pixels up/down</p> <p>PRIORITY_UNKNOWN: object will always update, unless paused (or frozen), not entirely sure the difference between this and PRIORITY_ACTIVE</p>
<code>object[index].priority</code>	The object's priority value, determines how the engine handles object activity, by default it's set to PRIORITY_ACTIVE_BOUNDS
<code>object[index].drawOrder</code>	The object's drawing layer: is 3 by default. Manages what drawList the object is placed in after ObjectMain
FLIP_NONE FLIP_X FLIP_Y FLIP_XY	IDs for Object.Direction.
<code>object[index].direction</code>	determines the flip of the sprites when drawing
INK_NONE INK_BLEND INK_ALPHA INK_ADD INK_SUB	IDs for Object.inkEffect

<code>object[index].inkEffect</code>	Determines the blending mode used with <code>DrawSpriteFX</code> & <code>FX_INK</code>
<code>object[index].alpha</code>	The object's transparency from 0 to 255.
<code>object[index].frame</code>	The object's frame ID
<code>object[index].animation</code>	The object's animation ID
<code>object[index].prevAnimation</code>	The last animation the object was processing during <code>ProcessAnimation()</code>
<code>object[index].animationSpeed</code>	The object's animation processing speed
<code>object[index].animationTimer</code>	The timer used to process the animations
<code>object[index].lookPosX</code> <code>object[index].lookPosY</code>	The camera offset from the player's position.
<code>object[index].outOfBounds</code>	Read-only value that is true if the object is out of the camera bounds
<code>object[index].spriteSheet</code>	The <code>spriteSheetID</code> of the active object
<code>ProcessObjectControl()</code>	Handles control inputs
<code>ProcessObjectMovement()</code>	Handles all of object tile collisions (used almost only for player)

C_TOUCH C_BOX C_BOX2 C_PLATFORM	IDs for collision type below
BoxCollisionTest( int type, int thisObject, int thisLeft, int thisTop, int thisRight, int thisBottom, int otherObject, int otherLeft, int otherTop, int otherRight, int otherBottom)	Checks for a collision between thisObject and otherObject using the hitbox values passed. Values can be set to 0x10000 and they will instead be loaded from the object's active hitbox. Sets CheckResult to 0 if there wasn't a collision, otherwise it's set to 1 (floor), 2 (LWall), 3 (RWall) or 4 (Roof)
CSIDE_FLOOR CSIDE_LWALL CSIDE_RWALL CSIDE_ROOF	IDs for cSide for the functions below
ObjectTileCollision(int cSide, int xOffset, int yOffset, int cPlane)	<p>Tries to collide with the FG layer based on the position of iXPos + xOffset, iYPos + yOffset.</p> <p>Sets CheckResult to true if there was a collision, false if there wasn't.</p> <p>This function is best used to check if a tile is there, not to move along it</p>

<code>ObjectTileGrip(int cSide, int xOffset, int yOffset, int cPlane)</code>	<p>Tries to collide with the FG layer based on the position of <code>iXPos + xOffset</code>, <code>iYPos + yOffset</code>.</p> <p>Sets <code>CheckResult</code> to true if there was a collision, false if there wasn't.</p> <p>This function is better used to handle moving along surfaces</p>
<code>object[index].angle</code>	Object's tile angle. Usually set via <code>ProcessObjectMovement()</code>
<code>object[index].collisionPlane</code>	Object collision plane (only 0 or 1)
<code>CMODE_FLOOR</code> <code>CMODE_LWALL</code> <code>CMODE_ROOF</code> <code>CMODE_RWALL</code>	IDs for CollisionMode, not to be confused with <code>CSIDE</code>
<code>object[index].collisionMode</code>	Object's active collision mode
<code>object[index].controlMode</code>	Object control mode (0 for normal)
<code>object[index].controlLock</code>	Object control lock timer
<code>object[index].pushing</code>	Object pushing flag usually set via collision functions
<code>object[index].visible</code>	Determines if the object is visible or not
<code>object[index].tileCollisions</code>	Determines if the object will interact with tiles or not

<code>object[index].interactions</code>	Determines if the object will interact with other objects or not
<code>object[index].gravity</code>	The object's gravity state. True if gravity is being applied (falling)
<code>object[index].up</code> <code>object[index].down</code> <code>object[index].left</code> <code>object[index].right</code> <code>object[index].jumpPress</code> <code>object[index].jumpHold</code>	Object input buffer values, generally set via <code>ProcessPlayerControl()</code>
<code>object[index].scrollTracking</code>	Determines if the camera will track the object's position or just follow it
<code>object[index].floorSensorL</code> <code>object[index].floorSensorC</code> <code>object[index].floorSensorR</code> <code>object[index].floorSensorLC</code> <code>object[index].floorSensorRC</code>	Collision sensor result values when on floor. True if there was no collision, false if there was
<code>object[index].collisionLeft</code> <code>object[index].collisionTop</code> <code>object[index].collisionRight</code> <code>object[index].collisionBottom</code>	The object's active hitbox values based on the loaded animation and <code>Object.Animation/Object.Frame</code> values
<code>GetObjectValue(int store, int index, int objectPos)</code>	Gets <code>Object[objectPos].Value[index]</code> and stores it in store
<code>SetObjectValue(int value, int index, int objectPos)</code>	Sets <code>Object[objectPos].Value[index]</code> to the value of value

`SetObjectRange(int range)`

sets the update ranges for all objects, range is how wide the “screen” is. the default values are the same as `SetObjectRange(424)`

`CopyObject(int  
destSlot, int srcSlot,  
int count)`

Copies (an) object(s) to another slot in the entity list.

Note: the size of the object list is 1184 entities, however there is another 1184 slots beyond that to be used for storage

Example: `CopyObject(1184, 0, 2)` copies the objects in slot 0 & slot 1 into slot 1184 & 1185 respectively



# Stages

Function/Variable/Alias	Description
LoadStage()	Loads a stage based on <code>stage.ListPos</code> & <code>stage.ActiveList</code>
<code>stage.listPos</code>	The stage index in the active stage list
<code>stage.activeList</code>	The active stage list to load stages from
<code>stage.listSize[index]</code>	The amount of stages that are in stage list <code>index</code>
PRESENTATION_STAGE REGULAR_STAGE BONUS_STAGE SPECIAL_STAGE	IDs for the 4 stage lists that can be used to store stages in RSDKv4
<code>stage.minutes</code> <code>stage.seconds</code> <code>stage.milliseconds</code>	The timer values for the current stage. These are automatically set for you as long as <code>stage.TimeEnabled</code> is true
<code>stage.timeEnabled</code>	Determines if the timer should increase or not
<code>stage.pauseEnabled</code>	Determines whether or not the player is allowed to pause the game
<code>stage.actNum</code>	The stage's current act ID

<code>stage.curXBoundary1</code> <code>stage.curXBoundary2</code> <code>stage.curYBoundary1</code> <code>stage.curYBoundary2</code>	The stage's main camera boundaries, the camera will not go beyond these
<code>stage.newXBoundary1</code> <code>stage.newXBoundary2</code> <code>stage.newYBoundary1</code> <code>stage.newYBoundary2</code>	The stage's other camera boundaries, the camera will not go beyond these, however these are used when setting new camera boundaries
<code>stage.deformationData0[index]</code> <code>stage.deformationData1[index]</code> <code>stage.deformationData2[index]</code> <code>stage.deformationData3[index]</code>	<p>The layer deformation data arrays.</p> <p>0 &amp; 1 are used for the FG Layer (0 being for above water, 1 being for below water), while 2 &amp; 3 are used for BG Layers (2 being for above water, 3 being for below water)</p>
<code>SetLayerDeformation(int deformID, int deformA, int deformB, int type, int offset, int count)</code>	Sets the deformation of the deformation data array of <code>deformID</code> based on the deform values
<code>stage.activeLayer[index]</code>	<p>Drawable layer IDs, with index 0 being the lowest and index 3 being the highest.</p> <p>Any layers that are not set with this array or are set to 9 will not be drawn.</p>
<code>stage.midpoint</code>	Any active layers above this value will draw only tiles on the high Visual Plane, otherwise they will only draw tiles on the low Visual Plane
<code>stage.waterLevel</code>	The height of the water relative to 0 in the stage layout

<pre> STAGE_RUNNING = 1 STAGE_PAUSED = 2 </pre>	Stage state IDs
<code>stage.state</code>	The stage's current activity state
<code>stage.playerListPos</code>	The current player ID, based on the gameconfig's player list
<code>stage.debugMode</code>	Determines if debugMode is active or not
<code>stage.entityPos</code>	The current slotID of the object being run
<code>GetTileLayerEntry(var store, int layer, int chunkX, int chunkY)</code>	Gets the chunkID of the chunk at chunkX, chunkY on tileLayer layer and stores it in store
<code>SetTileLayerEntry(int value, int layer, int chunkX, int chunkY)</code>	Sets the chunkID of the chunk at chunkX, chunkY on tileLayer layer and sets the index to value
<pre> TILEINFO_INDEX TILEINFO_DIRECTION TILEINFO_VISUALPLANE TILEINFO_SOLIDITYA TILEINFO_SOLIDITYB TILEINFO_FLAGSA TILEINFO_ANGLEA TILEINFO_FLAGSB TILEINFO_ANGLEB </pre>	<p>IDs for infoType for Get/Set16x16TileInfo</p> <p>TILEINFO_FLAGSB &amp; TILEINFO_ANGLEB can only be used with Get16x16TileInfo() as they are read-only</p>

<code>Get16x16TileInfo(int store, int tileX, int tileY, int infoType)</code>	Gets the info of <code>infoType</code> of the tile at <code>tileX</code> , <code>tileY</code> and stores it in <code>store</code>
<code>Set16x16TileInfo(int value, int tileX, int tileY, int infoType)</code>	Sets the info of <code>infoType</code> of the tile at <code>tileX</code> , <code>tileY</code> and sets it based on <code>value</code>
<code>Copy16x16Tile(int dst, int src)</code>	Copies the tileset image data of <code>src</code> into <code>dst</code> , used for animated tiles
<code>CheckCurrentStageFolder(string folder)</code>	If the loaded stage's folder matches <code>folder</code> , <code>CheckResult</code> is set to <code>true</code> , else it is set to <code>false</code>
<code>tileLayer[index].xsize</code> <code>tileLayer[index].ysize</code>	The width/height of the <code>tileLayer</code> in chunks
<code>TILELAYER_NOSCROLL</code> <code>TILELAYER_HSCROLL</code> <code>TILELAYER_VSCROLL</code> <code>TILELAYER_3DFLOOR</code> <code>TILELAYER_3DSKY</code>	IDs for <code>TileLayer.Type</code>
<code>tileLayer[index].type</code>	The type of rendering that the <code>tileLayer</code> uses
<code>tileLayer[index].angle</code>	The angle of the <code>tileLayer</code> (used for 3DFloor & 3DSky rotations)
<code>tileLayer[index].xpos</code> <code>tileLayer[index].ypos</code>	The position of the <code>tileLayer</code> (used for 3DFloor & 3DSky rotations)

<code>tileLayer[index].zpos</code>	
<code>tileLayer[index].parallaxFactor</code> <code>tileLayer[index].scrollSpeed</code> <code>tileLayer[index].scrollPos</code>	The parallax values of the tileLayer (see parallax below for more info)
<code>tileLayer[index].deformationOffset</code> <code>tileLayer[index].deformationOffsetW</code>	The offset for the deformation data arrays when rendering (0,1 for FG & 2,3 for BG)
<code>hParallax[index].parallaxFactor</code> <code>vParallax[index].parallaxFactor</code>	The scroll info's parallax factor (relative speed), which determines how many pixels the parallax moves per pixel move of the camera
<code>hParallax[index].scrollSpeed</code> <code>vParallax[index].scrollSpeed</code>	The scroll info's scroll speed (constant speed), which determines how many pixels the parallax moves per frame
<code>hParallax[index].scrollPos</code> <code>vParallax[index].scrollPos</code>	The scroll info's scroll position, which is how many pixels the parallax is offset from the starting pos

# Input

Function/Variable/Alias	Description
<code>inputDown.up</code> <code>inputDown.down</code> <code>inputDown.left</code> <code>inputDown.right</code> <code>inputDown.buttonA</code> <code>inputDown.buttonB</code> <code>inputDown.buttonC</code> <code>inputDown.buttonX</code> <code>inputDown.buttonY</code> <code>inputDown.buttonZ</code> <code>inputDown.buttonL</code> <code>inputDown.buttonR</code> <code>inputDown.start</code> <code>inputDown.select</code>	True if the corresponding button/key has been held. <code>inputDown.buttonX</code> through <code>Z</code> and <code>L/R</code> are both mapped to <code>A/B/C</code>
<code>inputPress.up</code> <code>inputPress.down</code> <code>inputPress.left</code> <code>inputPress.right</code> <code>inputPress.buttonA</code> <code>inputPress.buttonB</code> <code>inputPress.buttonC</code> <code>inputPress.buttonX</code> <code>inputPress.buttonY</code> <code>inputPress.buttonZ</code> <code>inputPress.buttonL</code> <code>inputPress.buttonR</code> <code>inputPress.start</code> <code>inputPress.select</code>	True if the corresponding button/key was pressed on this frame. Same note as above.

menu1.Selection  
menu2.Selection

the current row selected by MENU\_1/MENU\_2

CheckTouchRect(int x1, int  
y1, int x2, int y2)

Checks if a touch input was detected between the inputted coordinates (based on screen)

# Math

Function/Variable/Alias	Description
<code>Sin(int store, int angle)</code> <code>Cos(int store, int angle)</code>	Gets the value from the sin/cos512 lookup table based on <code>angle</code> and sets it in <code>store</code>
<code>Sin256(int store, int angle)</code> <code>Cos256(int store, int angle)</code>	Gets the value from the sin/cos256 lookup table based on <code>angle</code> and sets it in <code>store</code>
<code>ATan2(int store, int x, int y)</code>	Performs an arctan operation using <code>x</code> and <code>y</code> and stores the result in <code>store</code>
<code>GetBit(var store, int value, int pos)</code>	Gets bit at index <code>pos</code> from <code>value</code> and stores it in <code>store</code>
<code>SetBit(int value, int pos, int set)</code>	Sets bit at index <code>pos</code> to <code>set</code> and updates <code>value</code> accordingly
<code>Rand(var store, int max)</code>	Gets a random value from 0 to <code>max</code> and stores it in <code>store</code>
<code>Not(var value)</code>	Performs a NOT operation on <code>value</code> and updates it ( <code>value = ~value</code> )
<code>Abs(var value)</code>	Gets the absolute number of <code>value</code> and updates <code>value</code> with the new number
<code>GetTableValue(var store, int index, arr array)</code>	Gets a value from <code>array</code> at <code>index</code> and stores it in <code>store</code>



```
SetTableValue(int  
value, int index, arr  
array)
```

Sets the value in array at index to value

```
Interpolate(var store, int  
x, int y, int percent)
```

Linearly interpolates (LERPs) x and y by percent and stores the result in store.  
percent is 0 through 256.

```
InterpolateXY(var storeX,  
var storeY, int aX, int aY,  
int bX, int bY, int percent)
```

InterpolateXY does 2 at once for points (aX, aY) and (bX, bY)

# 3D

Function/Variable/Alias	Description
MAT_WORLD MAT_VIEW MAT_TEMP	RSDKv4 only allow use of 3 matrices: world, view & temp. Passing these should only be done to parameters of type mat. RSDK matrix values are shifted 8 bits, so 0x100 (starting vals) is 1.0
scene3D.vertexCount scene3D.faceCount	Amount of active faces/vertices in each buffer respectively (max of 1024 faces and 4096 vertices)
scene3D.projectionX scene3D.projectionY	The width (X) and height (Y) of the 3DScene draw buffer. These values determine what base resolution to use for drawing functions.
scene3D.fogColor scene3D.fogStrength	The colour of the fog in RGB format and the strength of the fog (0-255). Used with FADE_FADED flag
faceBuffer[index].a faceBuffer[index].b faceBuffer[index].c faceBuffer[index].d	The vertex indices to use to control this face's drawing
FACE_TEXTURED_3D FACE_TEXTURED_2D FACE_COLOURED_3D FACE_COLOURED_2D FACE_FADED FACE_TEXTURED_C FADE_TEXTURED_D FACE_SPRITE3D	The different face drawing flags that can be used with faceBuffer.flag.

<code>faceBuffer[index].flag</code>	The active drawing flag for this face
<code>faceBuffer[index].color</code>	The colour to draw the face when drawing with <code>FACE_COLOURED_2D</code> or <code>FACE_COLOURED_3D</code> flags
<code>vertexBuffer[index].x</code> <code>vertexBuffer[index].y</code> <code>vertexBuffer[index].z</code> <code>vertexBuffer[index].u</code> <code>vertexBuffer[index].v</code>	The vertex coordinates for the specified vertex
<code>SetIdentityMatrix(mat matrix)</code>	Sets the matrix of <code>matID</code> to the identity state
<code>MatrixMultiply(mat matrixA, mat matrixB)</code>	Multiplies <code>matrixA</code> by <code>matrixB</code> and stores the result in <code>matrixA</code>
<code>MatrixTranslateXYZ(mat matrix, int x, int y, int z)</code>	Translates <code>matrix</code> to <code>x, y, z</code> , all shifted 8 bits ( <code>0x100 = 1.0</code> )
<code>MatrixScaleXYZ(int matrix, int x, int y, int z)</code>	Scales <code>matrix</code> by <code>x, y, z</code> , all shifted 8 bits ( <code>0x100 = 1.0</code> )
<code>MatrixRotateX(mat matrix, int angle)</code> <code>MatrixRotateY(mat matrix, int angle)</code> <code>MatrixRotateZ(mat matrix, int angle)</code> <code>MatrixRotateXYZ(mat matrix, int x, int y, int z)</code>	Rotates <code>matrix</code> to <code>angle</code> on the specified axis, or all if using <code>MatrixRotateXYZ</code> . Angles are 512-based, similar to sin/cos

<code>MatrixInverse(int matrix)</code>	Performs an inversion on the values of <code>matrix</code>
<code>TransformVertices(mat matrix, int startIndex, int endIndex)</code>	Transforms all vertices from <code>startIndex</code> to <code>endIndex</code> using <code>matrix</code>
<code>Draw3DScene()</code>	Draws the active 3DScene data to the screen

# Menus

Function/Variable/Alias	Description
MENU_1 MENU_2	Menu IDs for menu parameters
LoadTextFile(int menu, string filePath)	Loads a menu based on the file loaded from filePath
SetupMenu(int menu, int rowCount, int selectionCount, int alignment)	Sets up menu with rowCount rows, selectionCount active selections and aligning to alignment
AddTextMenuEntry(int menu, string text, int highlightEntry)  EditTextMenuEntry(int menu, string text, int rowID, int highlightEntry)	Adds or edits an entry to menu with the contents of text, and highlighted if highlightEntry is set to true
TEXTINFO_TEXTDATA TEXTINFO_TEXTSIZE TEXTINFO_ROWCOUNT	Types of data that can be fetched via GetTextInfo().

GetTextInfo(var store, int menu, int type, int index, int offset)	Gets the data of type from menu using index, using offset if the type is TEXTINFO_TEXTDATA
DrawMenu(int menu, int XPos, int YPos)	Draws menu to XPos & YPos relative to the screen
GetVersionNumber(int menu, int highlight)	Adds a text entry with the game's version as the text, highlighted if highlight is set

# Engine

Function/Variable/Alias	Description
<code>engine.state</code>	The current engine game loop state, can be set to 2 or <code>RESET_GAME</code> to restart the game
<code>engine.language</code>	The language the engine is actively using
<code>engine.onlineActive</code>	Whether or not online functionality is enabled for the engine
<code>engine.trialMode</code>	Whether or not the game is built as a “trial version” (basically always false)
<code>RETRO_STANDARD</code> <code>RETRO_MOBILE</code>	Names for the values of <code>engine.deviceType</code>
<code>engine.deviceType</code>	The current device type the game is currently running on
<code>CallNativeFunction(int functionID)</code>  <code>CallNativeFunction2(int functionID, int param1, int param2)</code>  <code>CallNativeFunction4(int functionID, int param1, int param2, int param3, int param4)</code>	Calls the native engine function with the ID of <code>callbackFuncID</code> using no params, 2 params, or 4 params respectively

<code>Print(message, bool isInt, bool addNewLine)</code>	Prints a message to the console & the log, if <code>isInt</code> is set then <code>message</code> will be treated as an int, otherwise it will be treated as a string. If <code>addNewLine</code> is set a <code>\n</code> character will be added to the end of the message when printed.
<code>saveRAM[index]</code>	an array of data capable of being written/read from file via <code>ReadSaveRAM()</code> / <code>WriteSaveRAM()</code>
<code>ReadSaveRAM()</code>	reads the contents of the save file on disk into SaveRAM (overwrites any existing values)
<code>WriteSaveRAM()</code>	writes the contents of SaveRAM to the save file on disk



# Further Assistance

For any further questions relating to RetroScript or RSDK modding in general, [join the Retro Engine Modding Server: your one stop for all RSDK modding!](#)