

THE SCOURING SCRIPTING BASIC GUIDE
30 September 2025
Please note: this is not a LUA programming Guide!

You can download Scouring Dev Tools here:

[Link To Google Drive](#)

Actual version: SCOURING_DEVTOOLS_250930.zip (migration guide at end)

Note: make sure to follow us at Steam

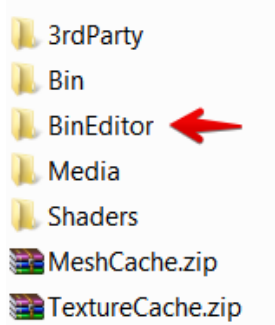
https://store.steampowered.com/app/3338950/The_Scouring/

and Discord

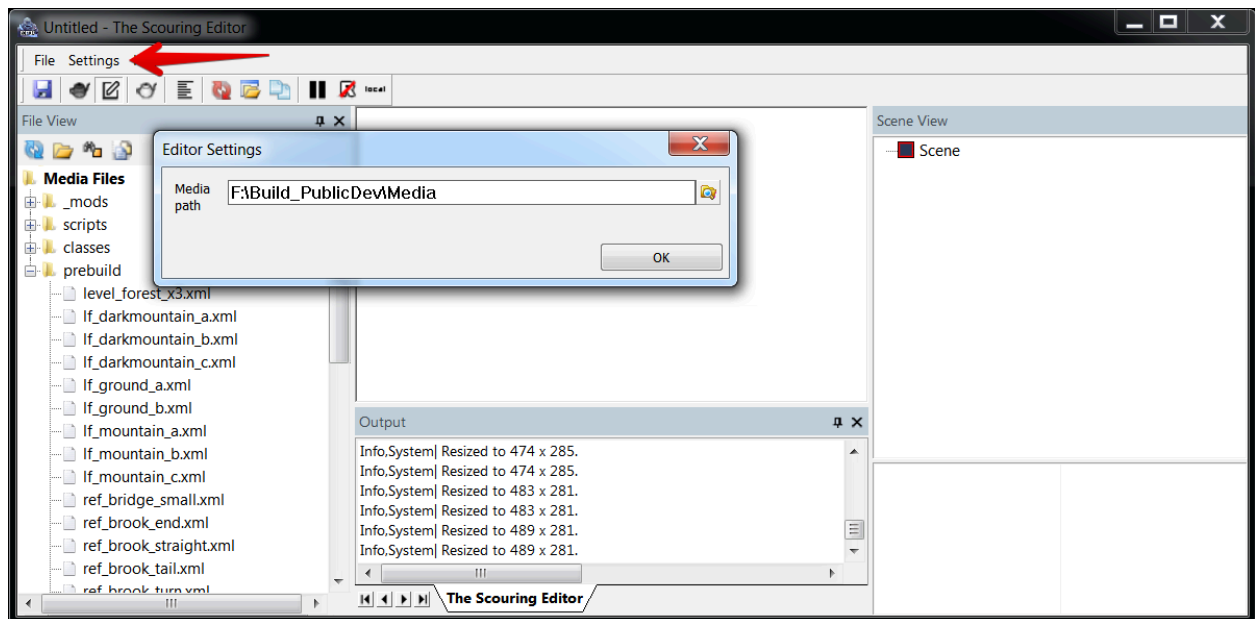
<https://discord.gg/9eSrACUgcd>

To keep track of updates!

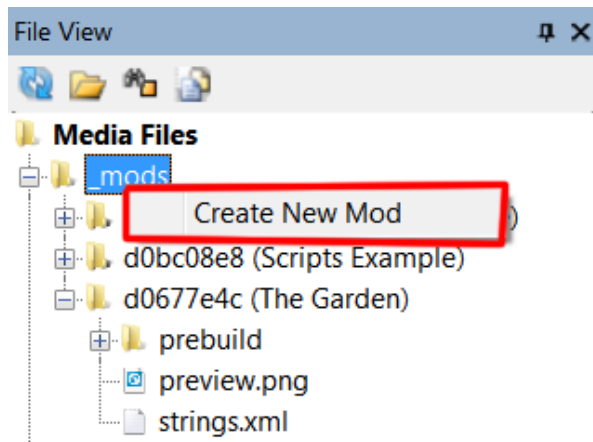
Start the editor by launching **TheScouringEditor.exe**



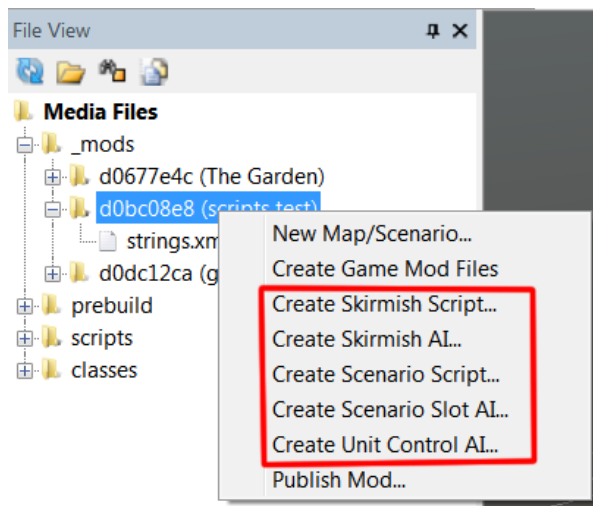
It automatically searches for the game files (located in the 'Media' folder), but you can change that path later.



Create new Steam Workshop mod:



Then create a new script:

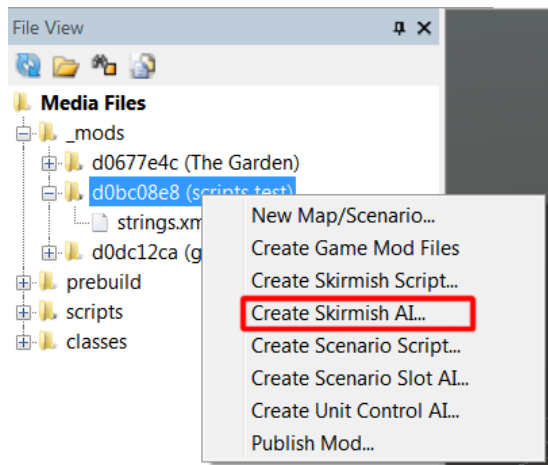


There are 5 types of scripts in the The Scouring:

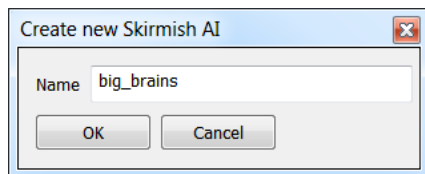
1. Skirmish Script - controls the background behaviour of a skirmish game. A default script is called "Curse of the Undead" and spawns aggressive skeletons at nighttime.
2. Skirmish AI - controls a bot in a skirmish game. Can also be used in a single-player scenario game to fill up player slots.
In "Heroes" mode, for a player-occupied slot, this script type is responsible for controlling the player base/units (but not the hero) and responding to hero commands.
3. Scenario Script - same as Skirmish Script, but is used in scenario levels.
4. Scenario Slot AI - same as Skirmish AI, but scenario levels.
5. Unit Control AI - controls particular unit type.
Use this to create custom units behaviour for new units.

Note: stock game scripts are all included in the Dev Tools package and you can study them ("Media\scripts\ai" folder)

To create new Skirmish AI, select “Create Skirmish AI...”:

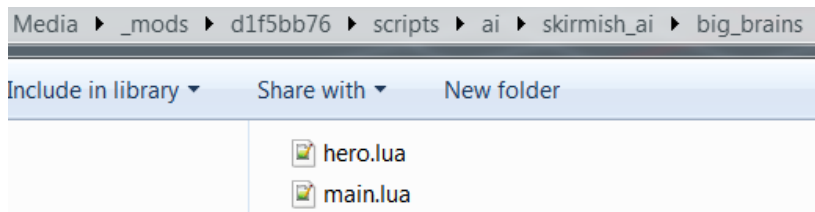


Then enter script name, which should be different to all other names (stock scripts included):



Important: Skirmish Scripts' names should start with “SkirmishScriptType” which is specified in the terrain preset. Currently, all levels have this set to “nighttime”. I.e. all skirmish scripts should start with “nighttime_”. This is subject to change.

A folder with new scripts' files will be opened:



main.lua is a required file for all script types. It defines functions which The Scouring would be calling to run this script (described in more detail later)

hero.lua is required for Skirmish AI and Scenario Slot AI script types.

You can add other .lua files too and include them in main.lua (or hero.lua)

The Scouring uses LuaJIT 2.1

<http://luajit.org/luajit.html>

Which in turn is based on Lua 5.1

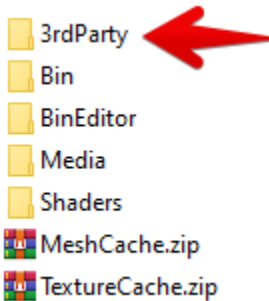
<https://www.lua.org/manual/5.1/>

Important: modules that can affect host PC, like sys, io, loadfile are available in Dev Tools version, but are disabled in Retail (Steam) version. Be careful when using Dev Tools to test unpacked versions of untrusted mods, as they can be dangerous!

For LUA editing, we recommend using ZeroBrane Studio

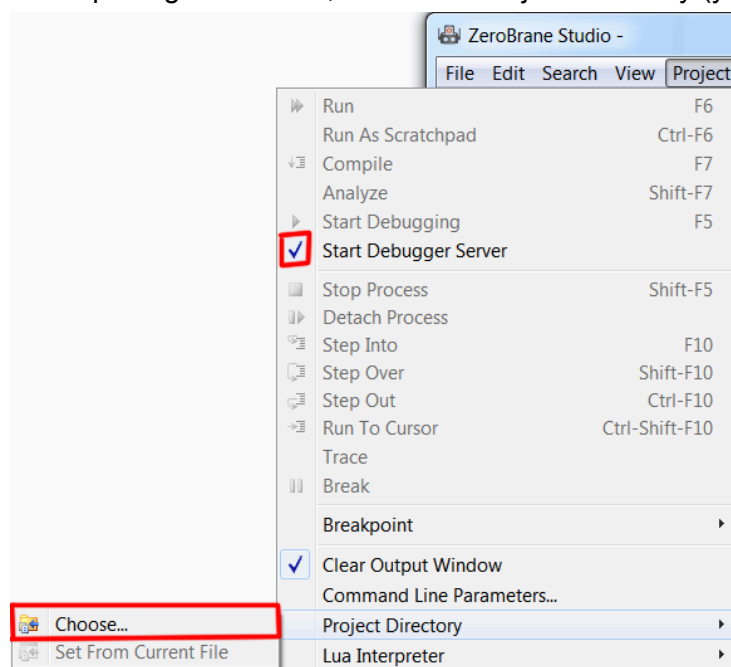
<https://studio.zerobrane.com/>

Pre-configured distribution is included in Dev Tools Package:



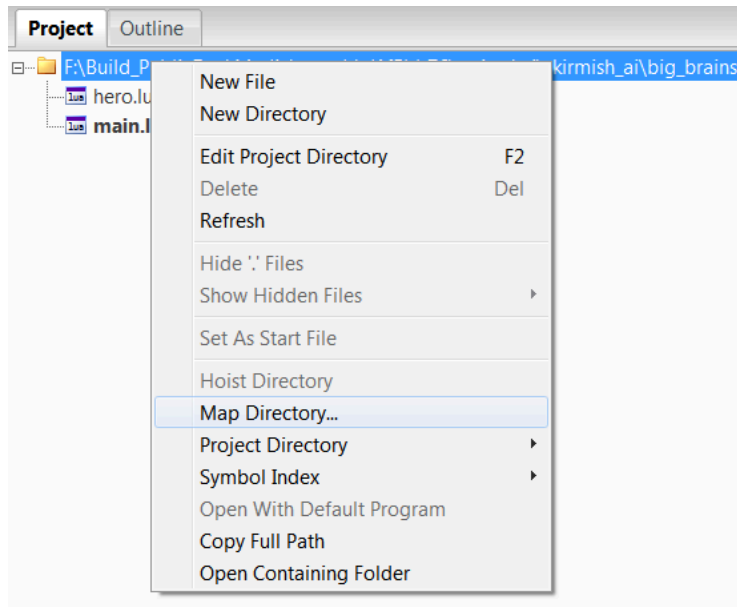
Note: you can use another/newer version of ZeroBrane Studio, just make sure to copy the scouring_output.lua plugin, which makes messages sent by The Scouring scripts appear in ZeroBrane Studio's output window.

After opening ZeroBrane, select the Project Directory (you don't have too, but that's convenient)

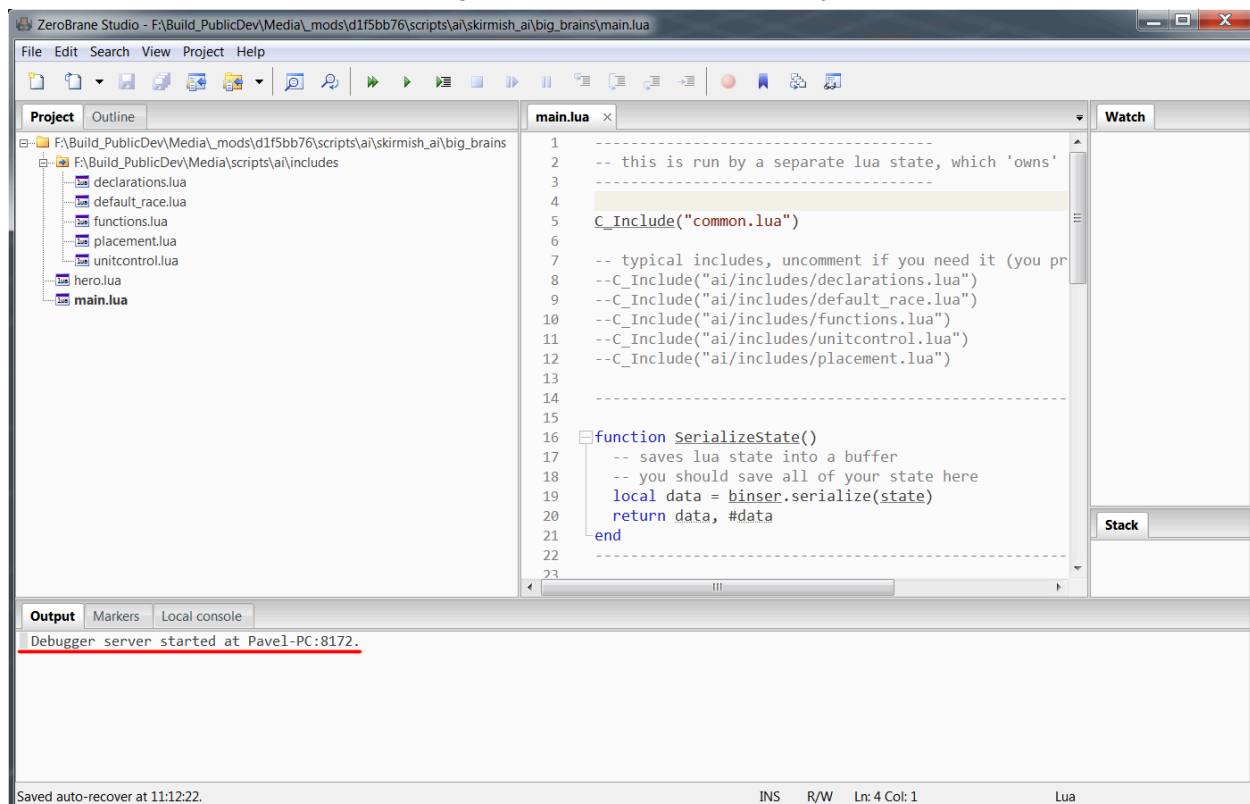


Also, make sure **Start Debugger Server** is on.

You can also “map” **Media\scripts\ai\includes** (from the root of Dev Tools), which contains stock lua functions:



So that you would be operating with only the files that affect your current script (i.e. operations like Find will be returning results from those files only):

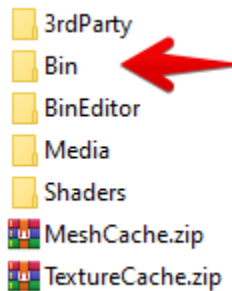


Note: make sure “Debugger server started” is printed in Output when you start ZeroBrane Studio (Start Debugger Server check is ON)

The **GetStats()** function returns scripts' debug information as a string (only used by Dev Tools).
Add a little test output:

```
function GetStats()  
-- add debug stats here  
-- called when appropriate 'Dev Tools' check box is set in the game  
-- not used in retail game  
if (state == nil) then  
    return "invalid state"  
end  
stats = "this is big brains AI!"  
return stats  
end
```

At this point, you can already test your script in the Dev version of the game.
Start it by launching **TheScouringRelease.exe**

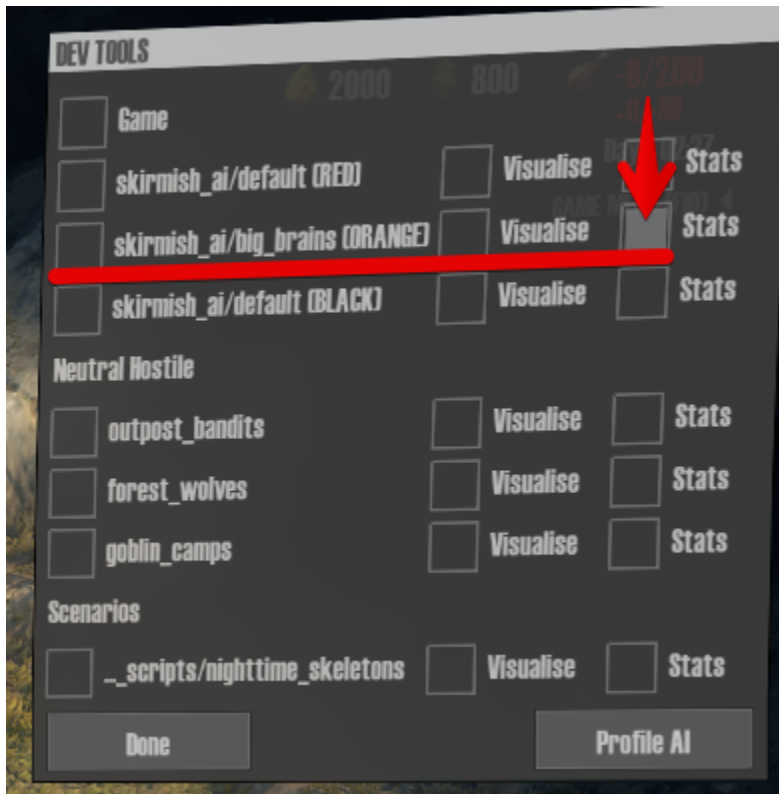


Skirmish AI will be available in a list of available bot AIs:



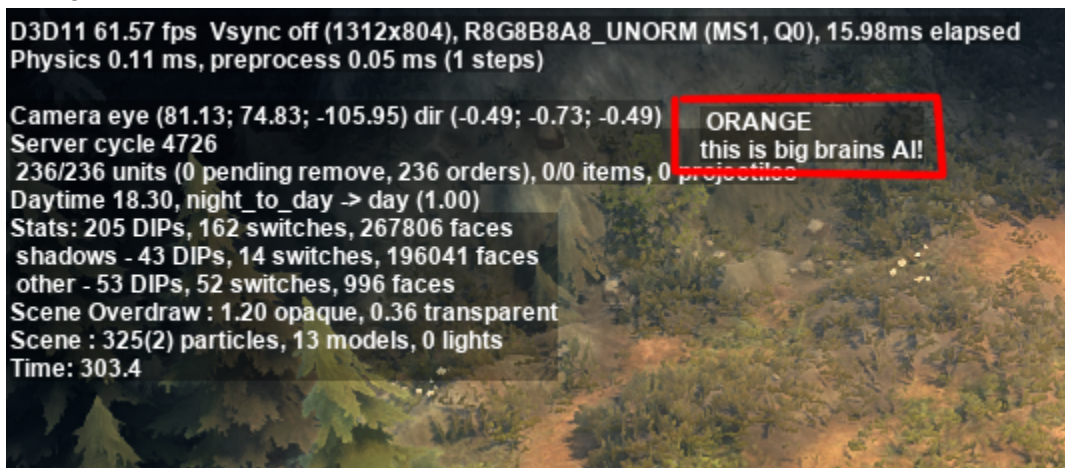
Note: scripts from your local mod files (not the mods downloaded from workshop) are tagged "(unpacked)". (here we only see "(unpa" because of text cropping)

After you start the game, press **CTRL-L** to activate the Dev Tools menu, which allows you to select which scripts would be calling their GetStats() function:



Note: the same way GetStats() is called for scripts that have their “Stats” check on, **Visualize()** is called for scripts that have their “Visualize” check on.

Then press **CTRL-D** to activate debug visualizations and output, and you will see the script’s debug information:



Note: Dev Tools’ menu state is saved and restored after you close and restart the game.

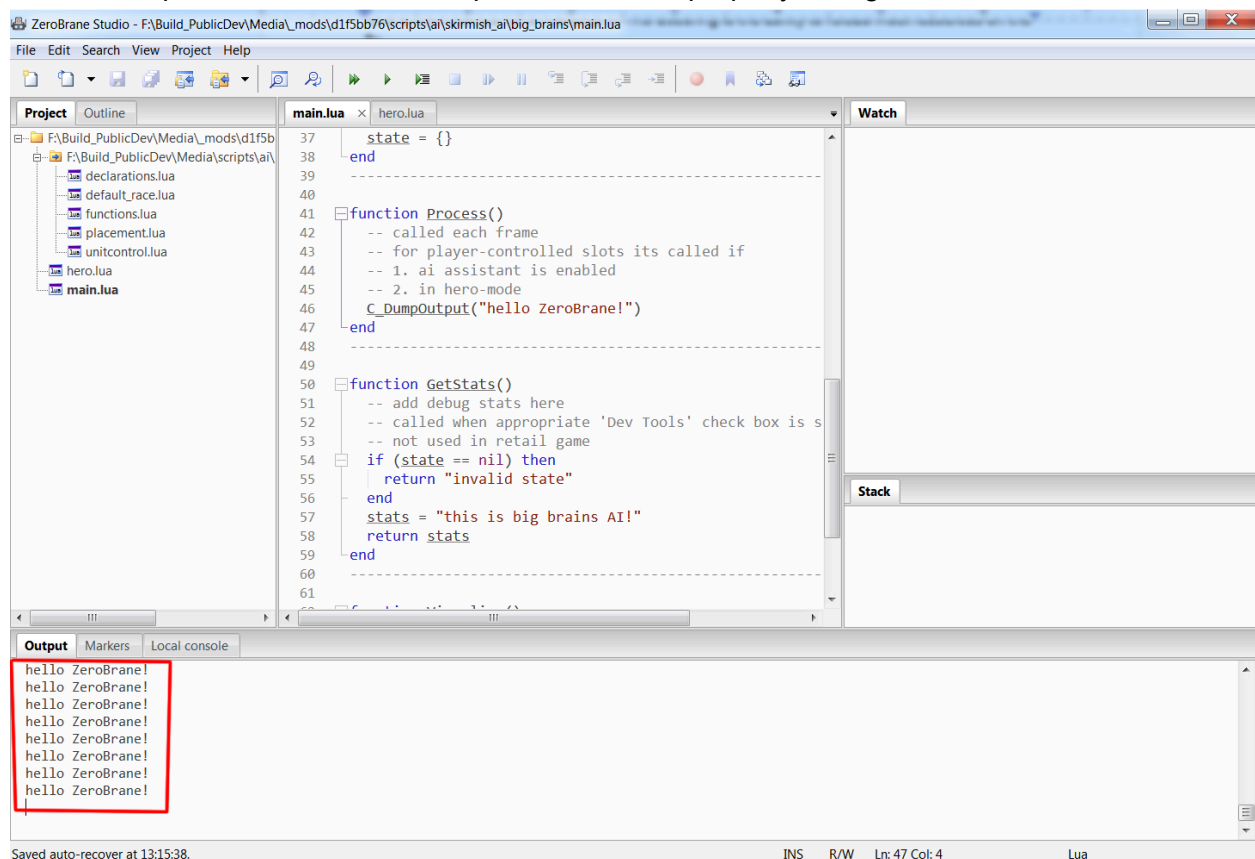
You can print logs from scripts too. To do so, use function **C_DumpOutput**:

```
function Process()  
    -- called each frame  
    -- for player-controlled slots its called if  
    -- 1. ai assistant is enabled  
    -- 2. in hero-mode  
    C_DumpOutput("hello ZeroBrane!")  
end
```

Note: all Scouring functions imported to LUA start with "C_".

For a complete list of imported functions, refer to [Functions Documentation](#)

This functions dumps text to %AppData%\TheScouring Dev\LuaOutput.txt, contents of which would be copied to ZeroBrane's Output window if its properly configured:

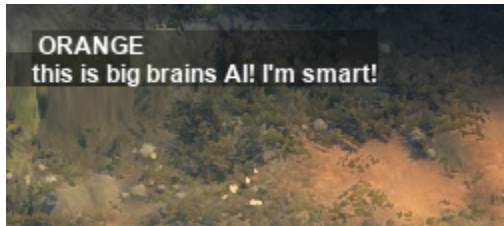


Note: LuaOutput.txt is only copied when you activate ZeroBrane, so you need to switch active windows for it to update.

One of the most powerful features of The Scouring is the ability to reload data on the fly. Without closing the game, go back to ZeroBrane and change the scripts' output:

```
function GetStats()  
    -- add debug stats here  
    -- called when appropriate 'Dev Tools' check box is set in the game  
    -- not used in retail game  
    if (state == nil) then  
        return "invalid state"  
    end  
    stats = "this is big brains AI!"  
    stats = stats.." I'm smart!"  
    return stats  
end
```

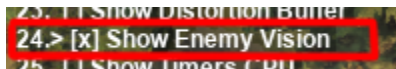
Press **CTRL-R** in the game to reload the script:



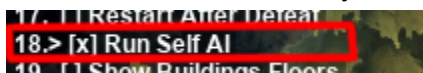
Note: reloading only works with your local files ("unpacked" files). To reload contents of a published mod, you need to reupload it to Steam Workshop (described later).

You would probably want to trigger "Show Enemy Vision" when testing scripts:

(list of triggers is activated by pressing '~', navigate it with Up/Down and trigger with Left/Right)

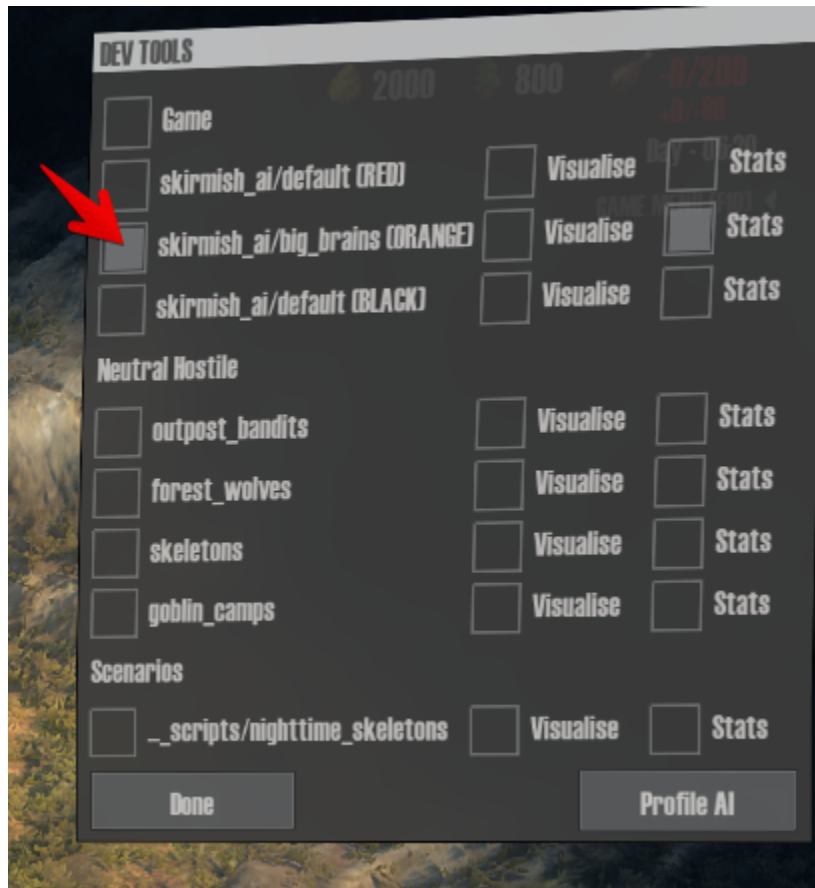


As well as "Run Self AI" if you want a script to control current player slot:



Note: refer to [The Scouring Modding Guide](#) for additional information about debug triggers.

Another crucial feature of The Scouring is the ability to debug scripts. All you need to do is trigger the checkbox next to a script that you want to debug:



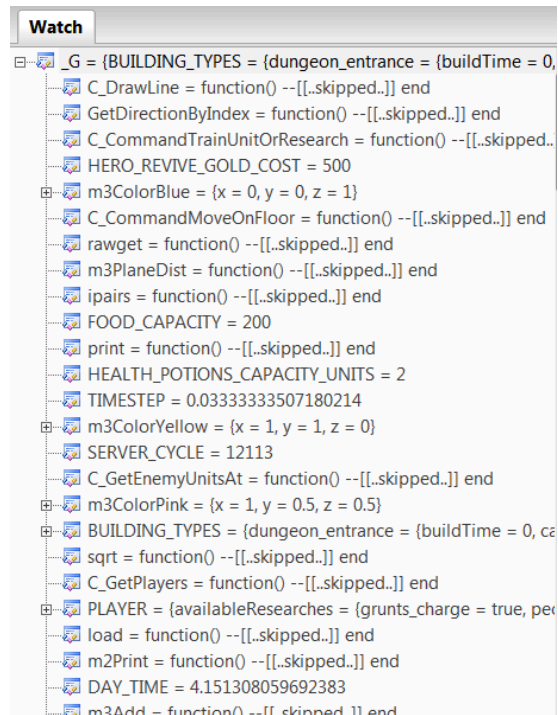
After that you can set Breakpoints in ZeroBrane (hotkey is CTRL-F9):



Just make sure that the file you have opened in ZeroBrane is the actual file of the script you are debugging. Also, if you have made changes to the script, make sure it's in sync with the game by pressing CTRL-R.

Important: ZeroBrane must be launched and the project directory must be set BEFORE launching the game for debugging to work properly. Sometimes the debugging connection might break too. In this case you should relaunch both the ZeroBrane and the game.

Note: when breakpoint is hit, you can use “Watch” window to familiarize yourself with what you can play with in the scripts (by watching global variable `_G`):



Your scripts must save (`SerializeState`) and load (`DeserializeState`) its full state. This is required when the game is terminated before the game session is complete (and then resumed) - but also when the multiplayer host migrates to transfer scripts state to the new host.

You can use built-in serializer (included to all scripts automatically):

```
function SerializeState()
    -- saves lua state into a buffer
    -- you should save all of your state here
    local data = binser.serialize(state)
    return data, #data
end

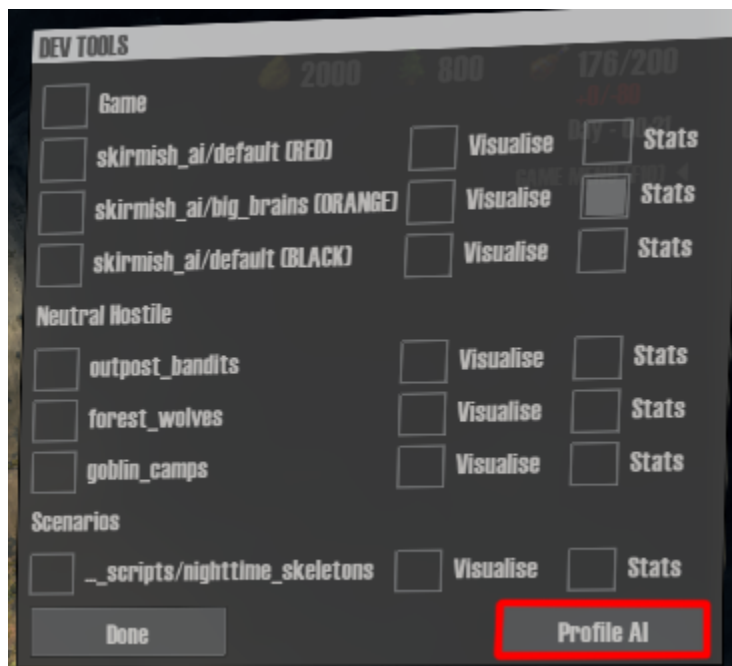
-----

function DeserializeState(data)
    -- loads lua state from buffer
    local results, len = binser.deserialize(data)
    if (results ~= nil) then
        local n = 1
        state = results[n]
        n = n + 1
    end
end
```

For more info, refer to <https://github.com/bakpakin/binser>

You can also profile the script with The Scouring's built-in profiler.

<https://github.com/2dengine/profile.lua>



Press **Profile AI**, give it some time to work, then press **Stop Profiling**.

Results would be printed to %AppData%\TheScouring Dev\LuaOutput.txt

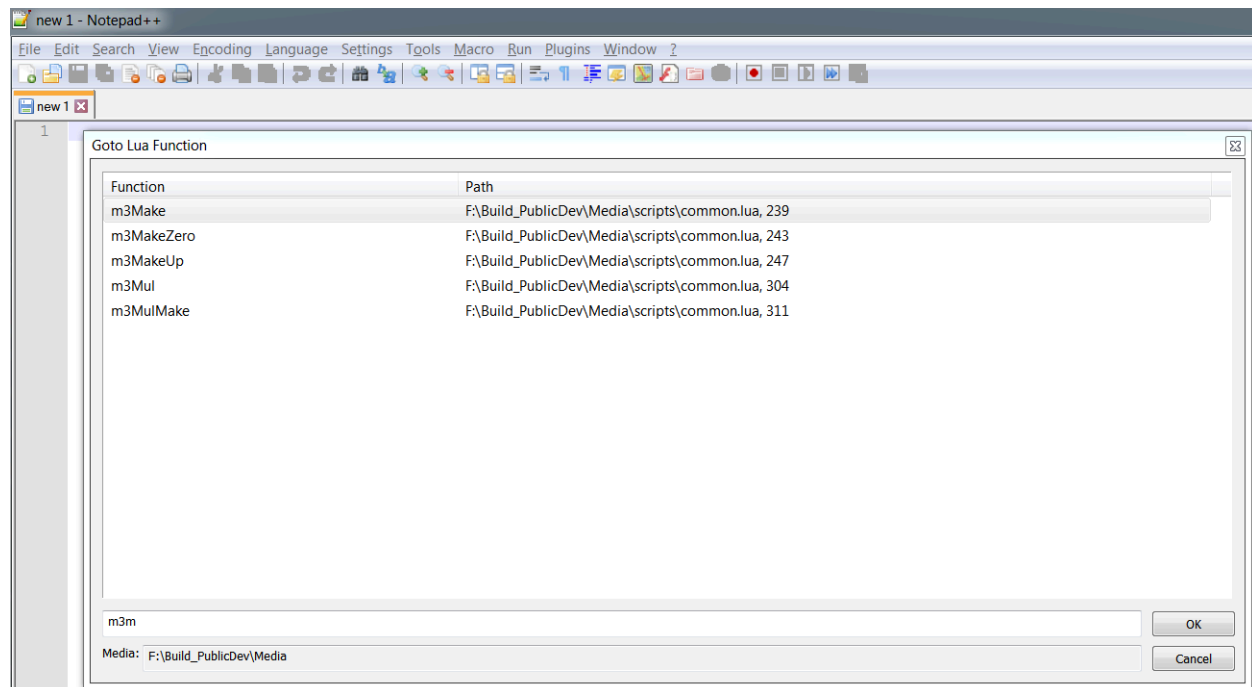
You can open this folder with F3 hotkey when the game is running.

Note: if you have ZeroBrane launched and properly configured, LuaOutput.txt would be copied to ZeroBrane's log.

To navigate your files, use ZeroBrane's Outline and Find Function (CTRL-B)



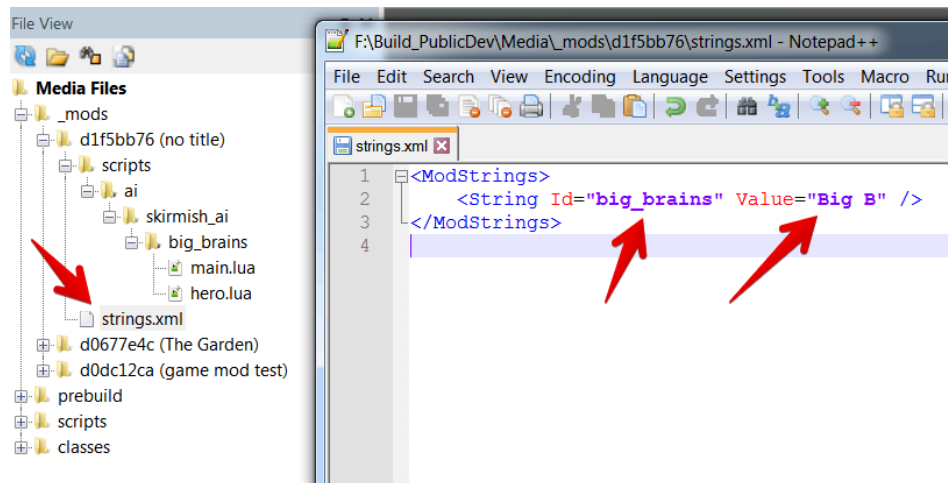
You can also use Notepad++ with The Scouring plugin's (which is also found in 3rdParty folder) **Goto Lua Function** (ALT-SHIFT-S):



Which is sometimes more convenient, especially if you need to look at files outside of your script's project directory. More specifically, the **common.lua** file which contains The Scourings' built-in math library and other utility functions.

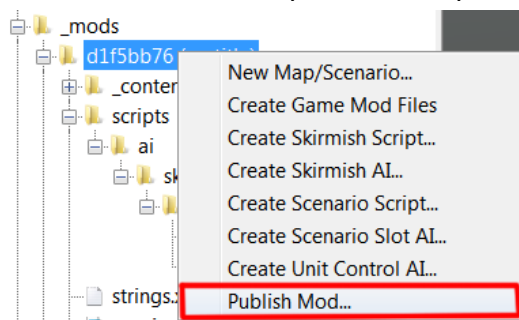
For more info about Notepad++, refer to [The Scouring Modding Guide](#).

Before publishing your script, add its name to **strings.xml**:

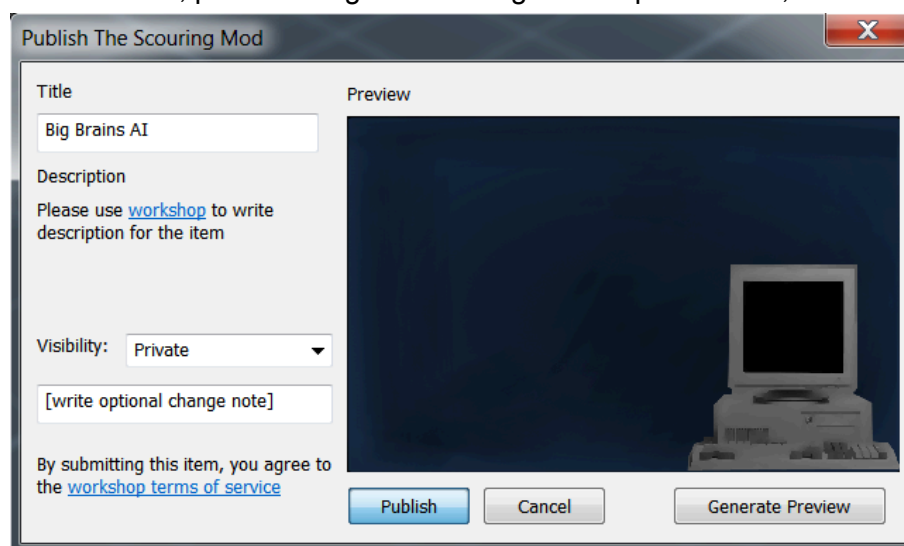


Note: this is not required, but otherwise your script's name will appear as an abbreviation of your mods' name after you download it from Steam Workshop.

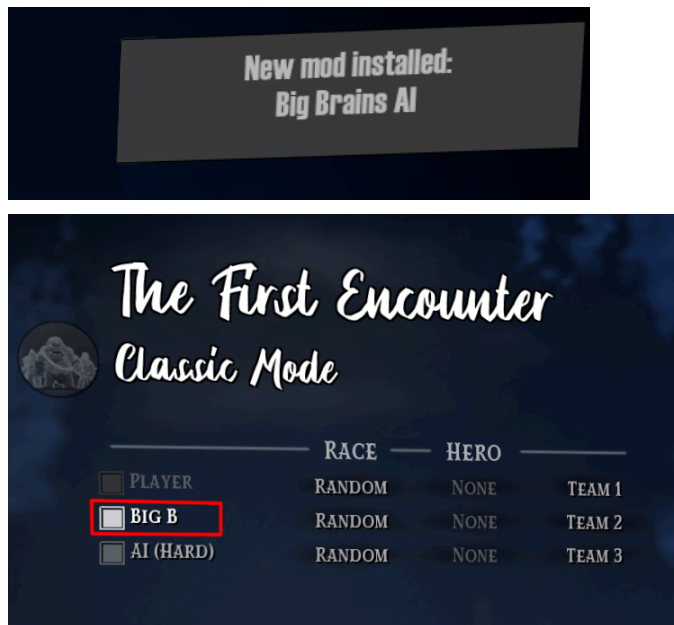
Select Publish Mod to pack it and upload to Steam Workshop:



Enter the title, press "Autogenerate" to generate preview file, and Publish the mod:



After publishing the mod, subscribe to it (you can use the in-game menu or Steam Workshop web interface), and it will become available in the list of available AIs (along with your local “unpacked” version):



Name of a script from a packed mod would be shown as an abbreviation of mods name in Dev Tools menu:



Note: scripts from packed mods (downloaded from Workshop) can't be connected to by ZeroBrane for debugging.

When developing a Skirmish AI script, the primary source of information about the world is stored in global variables `OWN_UNITS` and `OWN_BUILDINGS`.
(for more information, refer to [Structures Documentation](#)).

A typical processing loop would look something like that:

```
-- invalidate unit pointers
for _,assignment in pairs(assignments) do
    assignment.unit = nil
end

-- iterate units and update list
for _,unit in pairs(OWN_UNITS) do
    if (unit.type == UNIT_TROLL) then
        local assignment = assignments[unit.uid]
        if (assignment == nil) then
            AssignUnit(unit)
        else
            -- update unit pointer
            assignment.unit = unit
        end
    end
end
end
```

```
-- process assignments
for uid,assignment in pairs(assignments) do
    if (assignment.unit ~= nil) then
        ProcessUnit(assignment)
    else
        assignments[uid] = nil
    end
end
end
```

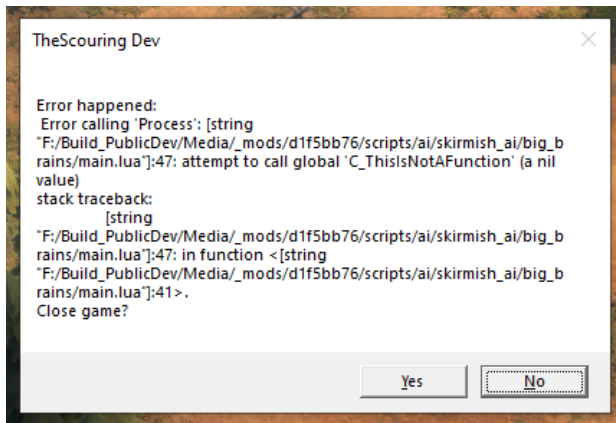
You would then put the unit's logic into `ProcessUnit()`. Refer to [Functions Documentation](#) for a list of available functions.

Note: don't forget to serialize and deserialize all of your state's variables!

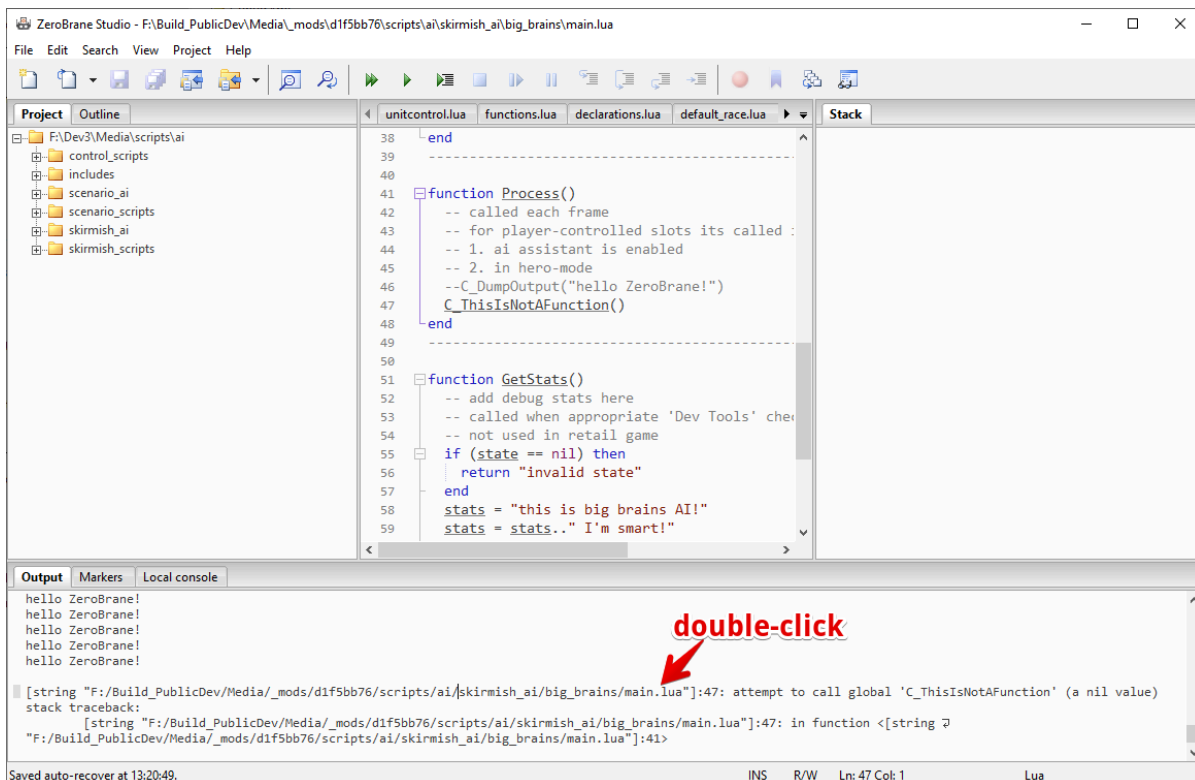
However, errors might happen when coding. Let's say we make a syntax Lua error like that:

```
function Process()  
    -- called each frame  
    -- for player-controlled slots its called if  
    -- 1. ai assistant is enabled  
    -- 2. in hero-mode  
    C_DumpOutput("hello ZeroBrane!")  
    C_ThisIsNotAFunction()  
end
```

After pressing CTRL-R or restarting the game, an error message would be printed:



However, this is not a critical error. We can skip it by pressing "No". If ZeroBrane Studio is running, this message would be copied to its Output Window:



Double-click the path in the Output window to navigate to the line with the error!

Now, there are also logical errors. Let's say we do something like:

```
for _,unit in pairs(OWN_UNITS) do
    C_CommandAction(unit.uid, ACTION_USE_HEALTH_POTION)
end
```

Trigger Print Command Errors:

```
15. [[Players Stats
16.> [x] Print Command Errors
17. [[Print Localisation Ids
```



Since we haven't checked that units have health potions, commands have failed.

If there is a mistake in the command declaration, i.e. we pass wrong unit UID like this:

```
C_CommandAction(123123, ACTION_USE_HEALTH_POTION)
```

The game will print indication like this:



Dev Tools Migration Guide

To update Dev Tools, If you have the previous version installed, simply unpack the new .ZIP archive to your folder replacing all files.

Note: this will overwrite any changes to the “stock” files that you made, so it's up to you to back them up yourself. Your Mod files will remain in the “_mods” subfolder untouched.

We will do our best to implement Dev Tools auto-updater later on.

For Skirmish Script, Scenario Script, or Unit Control AI, use `C_GetAllUnits` and `C_GetAllBuildings` to get information about the current state of the game.

For Skirmish AI, Scenario Slot AI those functions are not available. Instead, those script types perceive the world via “Metadata Grid”. This is a Lua table of tables (matrix), where each cell describes the state of a particular (square) section of the map.

The table itself is stored in the `METADATA` variable.

Information about its structure is stored in `METADATA_STEP_X`, `METADATA_STEP_Z`, `GRID_MIN_XZ`, `GRID_SIZE_XZ` variables.

Note: for optional parameters, if default value is not explicitly specified, it defaults to 0, nil, empty string, etc.

UID (unique identifiers) are used to retrieve following entities in The Scouring:

- Units
- Buildings
- Ships
- Floors (floor is an area on the roof of a building or a static model on the map. Dungeons are special floors that are isolated from the rest of the game scene)
- Plants (plants store resources like gold and lumber)

UIDs are integers.

Items don't have UIDs and are identified by their position (they can't move) and type.

Note: some unit types that are static, like farm houses, barricades, health potions stores behave like buildings in some ways (i.e. constructed with command `C_CommandPlaceUnit`), but are not buildings.

System Functions

C_Include

Includes a file with a path relative to current file

Inputs:

(String) Filename

C_DumpOutput

Dumps text to “%AppData%\TheScouring Dev\LuaOutput.txt” (which would be copied to ZeroBane Studio's Output window if its running and correctly configured)

Inputs:

(String) Text

World Access

C_GetPlayerByIndex

Inputs:

(Integer) Player index

Returns:

Player (table). See Lua Structures / Player.

C_GetPlayers

Player tags are used in scenario maps to give player slots specific behaviour.

Inputs:

(Comma separated list of strings) tagsRequired

(Comma separated list of strings) tagsExclude

Returns:

List of Players. See Lua Structures / Player.

C_GetGoldAmountAtTarget

Inputs:

(Integer) Target plant uid

Returns:

(Integer) Gold amount

C_GetLumberAmountAtTarget

Inputs:

(Integer) Target plant uid

Returns:

(Integer) Lumber amount

C_GetClosestGoldTarget

Inputs:

(Vec2) Search pos

(Float) Search radius

Returns:

A table containing:

- (Vec2) **pos**, position of a plant
- (Integer) **uid**, plant uid

Or nil if no plant was found within range.

C_GetClosestLumberTarget

Inputs:

(Vec2) Search pos

(Float) Search radius

Returns:

A table containing:

- (Vec2) **pos**, position of a plant
- (Integer) **uid**, plant uid

Or nil if no plant was found within range.

C_GetClosestLumberTargetWithDestination

Searches for a plant along a given direction, within a given “clearing width”.

Use this function if you need to chop through a forest to reach a particular destination, and have a corridor of given “width”.

Inputs:

(Vec2) Search pos

(Vec2) Destination pos

(Float) Clearing width

Returns:

A table containing:

- (Vec2) **pos**, position of a plant
- (Integer) **uid**, plant uid

Or nil if no plant was found.

C_GetClosestItem

Inputs:

(Vec2) Search pos

(Float) Search radius

(Integer, Optional) Floor uid

(String, Optional) Item type. One of: HealthPotion, ManaPotion, HeroItem

Returns:

(Vec2) Item pos

(String) Item type. One of: HealthPotion, ManaPotion, HeroItem

or

nil if no item was found within search radius.

C_GetAvailableFloorsAt

Inputs:

(Vec2) Search pos

(Float) Search radius

(Bool, Optional) isAvailableOnly (defaults to true)

(Bool, Optional) isLookForUnderground

Returns:

List of available Floor tables (see Lua Structures / Floor)

Note: floor is considered available if it belongs to a friendly building (doesn't have to be climbed to).

C_GetFloor

Inputs:

(Integer) Floor uid

Returns:

Floor table (see Lua Structures / Floor) or nil

C_GetUnit

Inputs:

(Integer) Unit uid

Returns:

Unit table (see Lua Structures / Unit Instance) or nil

C_GetBuilding

Inputs:

(Integer) Building uid

Returns:

Building table (see Lua Structures / Building Instance) or nil

C_FindRegionByName

Inputs:

(String) Region name

Returns:

A table containing:

- (Vec2) **center**, center of a region

Or nil if the region is not found.

C_GetBuildingByFloor

Searches for a building that floor belongs to.

Inputs:

(Integer) Floor uid

Returns:

Building table (see Lua Structures / Building Instance) or nil

C_GetFloorsByBuilding

Searches for floors that belong to the building.

Inputs:

(Integer) Building uid

Returns:

List of Floor tables.

C_GetEnemyUnitsAt

This function returns units that are seen by the current player.

Inputs:

(Vec2) Search pos

(Float) Search radius

(Integer, Optional) Underground floor uid

(Bool, Optional) Search for Heroes only

Returns:

List of Unit tables.

Note: if a unit is inside a dungeon, he can interact with units inside the same dungeon only.

C_GetFriendlyUnitsAt

Inputs:

(Vec2) Search pos

(Float) Search radius

(Integer, Optional) Underground floor uid

Returns:

List of Unit tables.

C_GetUnitsAt

This function returns units that are seen by the current player.

Inputs:

(Vec2) Search pos

(Float) Search radius

(Integer, Optional) Underground floor uid

Returns:

List of Unit tables.

C_GetBuildingsAt

Inputs:

(Vec2) Search pos

(Float) Search radius

Returns:

List of Building tables.

C_GetLumberAmountAt

Returns total amount of lumber at given location

Inputs:

(Vec2) Search pos

(Float) Radius

Returns:

(Integer) Lumber amount

C_GetGoldDepositsAt

Gold deposit is a location where one or more gold veins are located.

After a gold deposit is detected, you need to look for a particular gold target.

Inputs:

(Vec2) Search pos

(Float) Search radius

Returns:

List of tables. Each table containing:

- (Vec2) **pos**, position of a gold deposit
- (Integer) **goldAmount**

C_GetGridDistanceTo

Calculates the walking distance from one position to another

Inputs:

(Vec2) Source pos

(Vec2) Destination pos

(Bool, Optional) isWantMetadataCoords

Returns:

(Float) Distance

(Vec2) Initial movement direction

If isWantMetadataCoords was true,

List of tables, each containing

- (Integer) **x**, metadata grid coord X
- (Integer) **y**, metadata grid coord Z

Note: returns negative distance if destination can't be reached from source.

C_GetHeroItemDesc

Inputs:

(String) Item type

Returns:

Table describing item:

- (Integer) **buyCost**
- (Integer) **sellCost**

C_GetMetaCoordPos

Inputs:

(Integer) Metadata grid coord X

(Integer) Metadata grid coord Z

Returns:

(Vec2) Center of metadata grid cell.

Note: unlike the centers of cell of the METADATA global variable, this center is always the same.

C_IsGridWalkableAt

Inputs:

(Vec2) Check pos

(Float, Optional) Check radius

Returns:

(Bool) Walkable or not

C_IsDirectShotAvailable

Inputs:

(Integer) Unit uid

(Integer) Target unit uid

(Bool) isAllowThreshold. Is threshold is allowed (default to false), a “soft” check if done against buildings. This should be used for weapons that can’t hit buildings, because projectiles won’t damage the buildings even if they collide.

Returns:

(Bool) Available or not

C_IsCanSiegeAttackUnit

Determines if the target unit can be attacked by units in siege-mode.

Inputs:

(Integer) Target unit uid

Returns:

(Bool) Can attack or not

C_IsBuildingPlaceAvailable

Inputs:

(Vec2) Check pos

(String) Building type

Returns:

(Bool) Available or not

C_IsUnitPlaceAvailable

Inputs:

(Vec2) Check pos

(Float) Check radius

Returns:

(Bool) Available or not

C_IsCanSeeAt

Checks if a given player (or any player if Player index is not provided) has vision at a particular spot (overground).

Inputs:

(Vec2) Check pos

(Integer, Optional) Player index

Returns:

(Bool) Can see or not

C_IsDungeonHaveCreeps

Inputs:

(Integer) Dungeon floor uid

Returns:

(Bool) Have creeps or not

C_IsDungeonHaveBosses

Inputs:

(Integer) Dungeon floor uid

Returns:

(Bool) Have bosses or not

C_IsDungeonHavePlayerHero

Inputs:

(Integer) Dungeon floor uid

Returns:

(Bool) Have hero or not

C_IsHeroCanBuyConsumables

Hero can buy consumables when he is close to one of the Main buildings or Altars.

Inputs:

(Integer) Unit uid

Returns:

(Bool) Can buy consumables or not

C_IsUnitCanUseAbilityAtUnit

Inputs:

(Integer) Unit uid

(Integer) Target unit uid

(Integer) Ability alias

Returns:

(Bool) If target unit can be affected by the ability

C_IsUnitCanUseAbilityAtBuilding

Inputs:

(Integer) Unit uid

(Integer) Target building uid

(Integer) Ability alias

Returns:

(Bool) If target building can be affected by the ability

Following functions are available for non-player slot* (Skirmish Script, Scenario Script, or Unit Control Script)

C_GetAllUnits*

Retrieves full list of units for a given player (or all players if Player index is not provided)

Inputs:

(Integer, Optional) Player index

Returns:

List of Unit tables.

Note: this function can be slow if called frequently, if there are many units in the game

C_GetAllBuildings*

Retrieves full list of buildings for a given player (or all players if Player index is not provided)

Inputs:

(Integer, Optional) Player index

Returns:

List of Building tables.

World Modification

C_SetPlayerSurrender

Makes the current player surrender.

C_SendHeroCommand

Hero commands are the way for Heroes to interact with the AI in heroes mode.

It is completely up to the AI script to react to any Hero commands it receives.

Inputs:

(Integer) Unit uid

(String) Command name

C_BuyRevivalAmulet

Note: you need to check that the hero can buy consumables and that the player has enough money for this command to succeed.

C_DoHeroItemOperation

Inputs:

(String) Item type

(String) Item operation. One of: Buy, Sell

Note: you need to check that the hero can buy consumables for this command to succeed.

If the operation is Sell, you need to check that the player owns an item of a particular type.

If the operation is Buy, you need to check that the player has enough money.

C_MarkUnitUnderHeroControl

Inputs:

(Integer) Unit uid

(Bool) Is under hero control

Note: you should mark units that are currently under Hero command (and not under the AI command) for human-readable indication.

Following functions are available for non-player slots* (Skirmish Script, Scenario Script, or Unit Control Script):

C_SpawnUnit*

Inputs:

(String) Unit type

(Vec2) Unit pos

(Vec2, Optional) Unit direction. If not provided, random direction is used

(Integer, Optional) Unit Owner, if not provided, current state's player index is used.

If the current state doesn't have a player index (Skirmish Script, Scenario Script), this function will fail.

(Integer, Optional) Unit Tier

(Integer, Optional) Unit Floor uid

C_SpawnUnitWithTag*

Inputs:

(String) Unit type

(String) Tag. Any string, will be passed on to the "OnUnitSpawned" callback

(Vec2) Unit pos

(Vec2, Optional) Unit direction. If not provided, random direction is used

(Integer, Optional) Unit Owner, if not provided, current state's player index is used.

If the current state doesn't have a player index (Skirmish Script, Scenario Script), this function will fail.

(Integer, Optional) Unit Tier

(Integer, Optional) Unit Floor uid

C_RemoveUnit*

Inputs:

(Integer) Unit uid

(Bool) isPlayDeathEffect

C_ChangeUnitOwner*

Inputs:

(Integer) Unit uid

(Integer) New owner player index

C_ApplyUnitDamage*

Inputs:

(Integer) Unit uid

(Integer) nDamage

C_SpawnDroppableItem*

Inputs:

(Vec2) Item pos

(Integer, Optional) Target Floor uid

Note: item type is determined automatically

C_AddPlayerResources*

Inputs:

(Integer) Player index

(Integer) Gold amount

(Integer) Lumber amount

C_SetPlayerDefeated*

Inputs:

(Integer) Player index

Visual Notifications

C_AddHudsupMessage

Inputs:

(String) Message id

(String) Message text

(Float, Optional) Progress bar (0 to 1)

(Bool, Optional) Is just added (triggers pop-up animation)

(Integer, Optional) Player index to show to. If not provided, shown to all players.

Note: if this function is not continuously called with the same id, the message would be removed automatically.

C_AddNotificationMessage

Inputs:

(String) Message text

(String) Sound name to be played when notification is shown

(Integer, Optional) Player index to show to. If not provided, show to all players.

(Vec2, Optional) Notification position

C_AddMinimapMarker

Inputs:

(String) Marker id

(Vec2) Marker position

(Integer, Optional) Player index to show to. If not provided, shown to all players.

C_AddUnitPopupMessage

Inputs:

(Integer) Unit uid

(String) Message text

(Bool, Optional) Is message aggressive or passive

Commands

C_CommandStand

Inputs:

(Integer) Unit uid

(Vec2, Optional) Target pos

(Bool, Optional) isHold

Note: if target pos is provided and isHold is false, the unit will restrict its movement to it. I.e. will not chase attackers beyond a certain distance.

C_CommandMove

Inputs:

(Integer) Unit uid

(Vec2) Target pos

(Bool) isAttackMove

Note: if issued while the unit is on a floor (or inside a dungeon), the unit will exit the floor.

C_CommandMoveOnFloor

Inputs:

(Integer) Unit uid

(Integer) Floor uid

(Vec2) Target pos

(Bool) isAttackMove

Note: if issued while the unit is outside the target floor, the unit will first try to get on it.

C_CommandMoveToGetItem

Inputs:

(Integer) Unit uid

(Vec2) Target item pos

(String, Optional) Target item type. One of: HealthPotion, ManaPotion, HeroItem

Note: if item type is provided, the unit will only move to get an item of specified type.

C_CommandFarm

Inputs:

(Integer) Unit uid

(Integer) Target building uid

C_CommandUseAbility

Inputs:

(Integer) Unit uid

(String) Ability Type (Alias)

C_CommandUseAbilityAt

Inputs:

(Integer) Unit uid

(Vec2) Target pos

(String) Ability Type (Alias)

(Integer, Optional) Target floor uid

C_CommandUseAbilityAtUnit

Inputs:

(Integer) Unit uid

(Integer) Target unit uid

(String) Ability Type (Alias)

C_CommandAttack

Inputs:

(Integer) Unit uid

(Integer) Target unit uid

(Bool, Optional) isAllowRetarget

C_CommandAttackShip

Inputs:

(Integer) Unit uid

(Integer) Target ship uid

(Bool, Optional) isAllowRetarget

C_CommandAttackBuilding

Inputs:

(Integer) Unit uid

(Integer) Target building uid

(Bool, Optional) isAllowRetarget

C_CommandFollow

Inputs:

(Integer) Unit uid

(Integer) Target unit uid

C_CommandBuyHealthPotion

Inputs:

(Integer) Unit uid

(Integer) Health potion store uid (unit uid)

Note: This function makes a unit buy health potions at particular health potions “store” (which is another unit with IsHealthPotionsStore="true" in XML). Heroes can also buy health potions by using C_CommandAction.

C_CommandSiege

Inputs:

(Integer) Unit uid

(Vec2) Target direction

C_CommandDefend

Inputs:

(Integer) Unit uid

(Bool) isDefend

Note: this command does not stop current units' order. It instantly changes the unit's current state (defending or not).

C_CommandRestockShots

Inputs:

(Integer) Unit uid

C_CommandAction

Inputs:

(Integer) Unit uid

(String) Action type. One of: BuyHealthPotion, BuyManaPotion, UseHealthPotion, UseManaPotion

Note: actions stop current units' order but don't cancel it. It is resumed after action is complete.

C_CommandEnterFloor

Inputs:

(Integer) Unit uid

(Integer) Floor uid

C_CommandLeaveFloor

Inputs:

(Integer) Unit uid

C_CommandHarvestGold

Inputs:

(Integer) Unit uid

(Integer) Plant uid

C_CommandHarvestLumber

Inputs:

(Integer) Unit uid

(Integer) Plant uid

C_CommandConstructBuilding

Inputs:

(Integer) Unit uid

(Integer) Building uid

C_CommandRepairBuilding

Inputs:

(Integer) Unit uid

(Integer) Building uid

C_CommandPlaceBuilding

Inputs:

(Integer) Unit uid

(String) Building type

(Vec2) Target pos

C_CommandPlaceUnit

Inputs:

(Integer) Unit uid

(String) Unit type

(Vec2) Target pos

(Vec2, Optional) Target direction

C_CommandTrainUnitOrResearch

Inputs:

(Integer) Building uid

(String) Unit or research type

(Bool, Optional) Is training research (otherwise, a unit)

C_CommandUpgradeBuilding

Inputs:

(Integer) Building uid

(String) Target building type

Visualization Functions

C_GetUnitPosInScene

Inputs:

(Integer) Unit uid

(Bool, Optional) VisibleOnly - return position of visible units only (default to true)

Returns:

(Vec3) Unit world position

or nil if VisibleOnly was true and the unit is not visible.

C_DrawLine

Inputs:

(Vec2) Start pos

(Vec2) End pos

(Bool, Optional) Enable depth test (defaults to true)

(Vec3, Optional) Line color. Defaults to current state's owner player color.

(Bool, Optional) Draw pointer at the end

Note: map height is used as line height

C_DrawRadius

Inputs:

(Vec2) Pos

(Float) Radius

(Vec3, Optional) Box color. Defaults to current state's owner player color.

Note: map height is used as radius height

C_DrawBox

Inputs:

(Vec3) World min position

(Vec3) World max position

(Bool, Optional) Enable depth test (defaults to true)

(Vec3, Optional) Box color. Defaults to current state's owner player color.

C_DrawTextAt

Inputs:

(Vec3) World position

(String) Text

Note: map height is used as the minimum for world position's height (z coordinate). I.e. set world positions' Z to 0 if you want to draw the text at "default" height.

Global Constants

DAY_DURATION

(Float) Time in seconds of full day/night cycle

GRID_STEP

(Float) World size of a step in a server “grid”.

GRID_MIN_XZ

(Vec2) World position of a “top left” corner of the current map.

GRID_SIZE_XZ

(Vec2) World size of the current map.

TIMESTEP

(Float) Time in seconds between server cycles.

OWNER_INDEX_NEUTRAL_FRIENDLY

(Integer) Player index of “Neutral Friendly” slot

OWNER_INDEX_NEUTRAL_HOSTILE

(Integer) Player index of “Neutral Hostile” slot

MANA_POTIONS_CAPACITY_HERO

(Integer) Max number of mana potions a Hero can carry

HEALTH_POTIONS_CAPACITY_HERO

(Integer) Max number of health potions a Hero can carry

HEALTH_POTIONS_CAPACITY_UNITS

(Integer) Max number of health potions a non-hero unit with inventory can carry.
Only defined if Health Potions are enabled for current session

UNIT_TYPES

(Table) All unit types. See Game Structures / Unit Type

BUILDING_TYPES

(Table) All building types. See Game Structures / Building Type

DIFFICULTY_LEVEL

(Integer) Difficulty level, -1 (easy), 0 (normal), 1 (hard), 2 (insane)
Only defined for scenario levels

Player state constants

Skirmish scripts, scenario scripts, neutral slots are not player states (they don't have an owner index)

PLAYER

See Game Structures / Player

(Bool) **isDontSpendResources**

Trigger by user in hero mode

(Bool) **isControlledByPlayer**

True if controlled by AI. false if controlled by human (both hero and classic mode)

(Bool) **isAllowSurrender**

UNIT_HERO

(String) Current player's hero type

METADATA

(Table of tables) Metadata grid. See Game Structures / Metadata Cell

METADATA_STEP_X

(Float) World size of metadata grid step along X axis

METADATA_STEP_Z

(Float) World size of metadata grid step along Z axis

OWN_UNITS

(Table) List of units owned by current player. See Game Structures / Unit Instance

OWN_BUILDINGS

(Table) List of units owned by current player. See Game Structures / Building Instance

Player with race state constants

(Player state might not have a race in some situations)

RACE

(String) Race name (id)

FARM_FOOD_PRODUCTION

(Integer) FarmFoodProduction (XML value)

FOOD_CAPACITY

(Integer) FoodCapacity (XML value)

PEASANTS_LIMIT

(Integer) PeasantsLimit (XML value)

POTION_COST

(Integer) PotionGoldCost (XML value)

HERO_REVIVE_GOLD_COST

(Integer) HeroParameters/ReviveGoldCost (XML value)

RESEARCH_TYPES

(Table) All research types for this race

Game Structures

Note: "type" data structures fields are copied from XML structures and are updated when CTRL-R is pressed in game.

Unit (Type)

(Float) **radius**

(Float) **speed**

separationType - One of: Default, Knockback, NoKnockback, Hero

(Integer) **costGold**

(Integer) **costLumber**

(Float) **trainTime** (seconds)

requiredBuildings - list of strings (building types ids)

(Integer) **hitpointsBase**

(Integer) **hitpointsPerTier**

(Float) **defaultAttackRange**

(Float) **defaultAttackRangeThreshold**

(Float) **floorAttackRange**

(Float) **floorAttackRangeThreshold**

(Float) **scoreValue** - score is a unit metadata used by scripts (not used by the game itself)

(Float) **scoreOnFloor**

scoreTags - list of strings

example of usage

```
UNIT_TYPES["unit_type_id"].scoreTags["Tag"]==false
```

(Bool) **isHero**

(Bool) **isCanClimb**

(Bool) **isMechanical**

(Bool) **isPretendAsBuilding** - true for units that can't move (speed = 0). Makes the unit act as a building in some interactions (i.e. units that can attack buildings, can't attack this unit)

(Bool) **isHealthPotionsStore**

Research (Type)

(Integer) **costGold**

(Integer) **costLumber**

Building (Type)

(Integer) **costGold**

(Integer) **costLumber**

(Float) **buildTime** (seconds)

(Float) **captureTime** (seconds)

trainOptions - list of strings (unit types ids)

heroOptions - list of strings (unit types ids)

researchOptions - list of strings (research types ids)

requiredBuildings - list of strings (building types ids)

(Bool) **isMain**

True for BuildingType="Main". Main building serves as a delivery destination for lumber and gold, plus heroes can buy items when they are near. When the hero dies, he revives at the closest Main building.

(Bool) **isAltar**

True for BuildingType="Altar". Default losing condition is when the player loses all of his buildings. If any of the players in a team have an "Altar" building, team members don't lose because of the default losing condition. Additionally, friendly heroes revive at the closest Altar building if they die.

Floor

(Integer) **uid**

(Integer) **buildingUID**

Defined for building floors. Floors can exist without a building.

(Vec2) **entrancePos**

(Bool) **isUnderground**

Underground (dungeon) is a special floor type. Underground units don't affect overground vision (can only see each other if at the same dungeon). Also, some abilities are not available underground.

(Float) **spaceAvailable**

Amount of floor space available

(Float) **occupancy**

0..1 percentage of occupied floor space

Unit (Instance)

(String) **type**

(Integer) **uid**

(Integer) **tier**

(Integer) **ownerIndex**

(Bool) **isEnemy**

Defined for player

(Float) **manaAmount**

Defined for units with abilities only

(Float) **manaCapacity**

Defined for units with abilities only

(Vec2) **pos**

(Vec2) **dir**

(Integer) **floorUID**

Only defined for units that are on floors

(Bool) **isUnderground**

True if unit is on floor which is marked as “dungeon”

(Bool) **isCantMove**

True if movement speed is zero

(Bool) **isRageBuff**

True if “rage” buff is active

(Bool) **isSlowdownDebuff**

True if “slowdown” debuff is active

(Bool) **isDeathCloudEffect**

True if “death cloud” effect is active

(Bool) **isMagicShield**

True if unit have been cast “magic shield” upon

(Bool) **isCharging**

True if unit is using “charge” ability

(Bool) **isConsumableDelay**

(Bool) **isAttacking**

(Bool) **isClimbing**

(Bool) **isInsideBuilding**

(Bool) **isCanDefend**

True if “defend” ability is available (required research, if any, is complete)

(Bool) **isDefended**

True if “defend” ability is active and in effect. Even if “defend” is active, ability is not in effect if unit is climbing, being knockbacked, or attacking (unless IsDefendedWhenAttacking is “true”)

(Bool) **isDefending**

True if “defend” ability is active

(Vec2) **siegeDirection**

Defined if the current unit is sieged.

(Bool) **isRanged**

(Bool) **isAggressive**

If IsNotAggressive="true" (in unit's XML), defined as true only if current order is “aggressive” (attack, patrol, siege, etc)

Otherwise, defined as true for all units that can attack.

Those fields are defined only for a player-controlled state:

(String) **currentOrder**

For a list of possible order names, see declarations.lua (stock script file provided as part of Dev Tools package)

(Vec2) **targetPos**

World position of a current order (i.e. move order, attack order, etc)

(Integer) **targetUnitUID**

Uid of current unit order's target unit, if any

(Integer) **targetBuildingUID**

Uid of current unit order's target building, if any

(String) **orderAbilityType**

Id of the ability if current order is "Use ability"

(String) **orderBuildingType**

Id of the building if current order is "Place/Construct/Repair Building"

(String) **orderUnitType**

Id of the building if current order is "Place Unit"

abilities - table of booleans per ability id. Defines a list of available abilities. Ability is "available" if required research (if any) is complete, tier requirement (if any) is met, and ability is not on cooldown.

attackedBy - table describing a unit that is currently attacking this unit, with following data fields:

- (Integer) **uid** - uid of the attacker unit
- (Vec2) **pos** - world position of the attacker
- (Bool) **isRanged** - if attacker uses ranged attack

(Integer) **healthPotions**

Number of current health potions in unit inventory, defined for units that can carry health potions

(Integer) **manaPotions**

Number of current mana potions in unit inventory, defined for units that can carry mana potions

isUnderControl

False in AI-assistant mode, after a user issues an order to this unit (script temporarily loses control of it), or if a player has assigned this unit to a control group (script permanently loses control of it).

Building (Instance)

(String) **type**

(Integer) **uid**

(Float) **health**

0 to 1 percentage of buildings' health

(Vec2) **pos**

(Integer) **ownerIndex**

Only defined for buildings that are owned by someone.

(Vec2) **entrancePos**

(Vec2) **entranceDir**

(Bool) **isBuilt**

(Bool) **isTraining**

(Bool) **isBeingConstructed**

(Bool) **isBeingUpgraded**

(Bool) **isBeingCaptured**

Those fields are defined only for a player-controlled state:

(Bool) **isAvailableToOccupy**

(Bool) **isAvailableToConstruct**

(Bool) **isAvailableToRepair**

(Bool) **isAvailableToCapture**

(Bool) **isOwnedByEnemy**

(Bool) **isOwnedByAlly**

(Bool) **isUnderControl**

False in AI-assistant mode, after a user issues an order to this building (script temporarily loses control of it)

Player

(Integer) **playerIndex**

(Integer) **teamIndex**

(Integer) **goldBalance**

(Integer) **goldIncome**

(Integer) **lumberBalance**

(Integer) **lumberIncome**

(Integer) **foodAmount**

(Integer) **foodProduction**

(Integer) **foodConsumption**

(String) **race** - race id

slotTags - list of strings

Slot tags are used in scenario levels to customize AI behaviour for each slot.

completedResearches - list of strings (research types ids)

pendingResearches - list of strings (research types ids)

List of researches that are currently being trained.

availableResearches - list of strings (research types ids)

Research is available if its requirements (buildings, other researches etc) are all met.

(Bool) **isHeroesMode**

(Bool) **isDefeated**

(Bool) **isNotSpendingResources**

heroItemsInventory - list of strings (item types ids)

(Integer) **heroRevivalsRemaining**

Metadata Cell

(Vec2) **pos**

A world position close to the center of the grid cell. Note: this position changes each few frames, but it's always close to the center of the grid.

Values below reflect aggregated/estimated amount of entities of given type in given section of the map (metagrid cell):

(Float) **goldSum**

(Float) **lumberSum**

(Float) **alliedBuildings**

(Float) **enemyBuildings**

(Float) **capturableBuildings**

(Float) **alliedPassiveUnits**

(Float) **alliedAggressiveUnits**

(Float) **enemyPassiveUnits**

(Float) **enemyAggressiveUnits**

(Float) **neutralAggressive**

Refer to stock Lua functions like `GetMetaScoreAt` (included in Dev Tools package scripts) to get an idea on how to access Metadata Grid.

Scripts Moments

1) Update structure data

When you use structure, and save it like structure, for example:

```
local reg=C_FindRegionByName("hero")
local units=C_GetUnitsAt(reg.center,8)
for _,unit in pairs(units) do
    hero=unit
end
```

You will have structure **hero** with data that exist at a moment of command **hero=unit**. And if that **unit** changes their position, **hero.pos** will have an old position.

To actualize data use **hero=C_GetUnit(hero.uid)**.

CODE EDITOR INTEGRATION

Follow this guide to enhance your experience when writing Lua scripts for The Scouring. You will need to download an API file which can be accessed [here](#) (click).

What is API

API is an *application programming interface*. It describes how a person should interact with an application through code. The API provided to the code editor is only an API stub. This means that the API file doesn't implement any functionality; it contains only declarations and descriptions. In other words, the provided API file is not a function library, but rather its documentation.

What API provides

API can be given to a code editor, such as **Visual Studio Code** or **ZeroBrane Studio**, which can help you write code more easily by letting you know what tools are available. Typical features are:

- ***Generating choices for code completion;***
- ***Showing which functions and global variables the game provides;***
- ***Showing tooltips with function descriptions;***
- ***Describing expected input & output data types;***
- ***Describing what the game's Lua structures are made of;***
- ***Inferring datatypes for values returned from functions;***
- ****Showing warnings when given variables are incompatible with the API definitions.***

**Please, note that ZeroBrane's hinting features are limited compared to VS Code. Features marked with an asterisk are not supported in ZeroBrane. On the other hand, code debugging is currently supported only in the ZeroBrane Studio.*

Table of contents

Samples of API usage in ZeroBrane Studio.....	2
Samples of API usage in Visual Studio Code.....	2
How to import API to ZeroBrane Studio.....	3
Adding API file into Lua API folder.....	3
Static API integration.....	3
Dynamic API integration.....	4
Using API in ZeroBrane Studio.....	4
How to import API into Visual Studio Code.....	6
Installing Lua Language Server extension.....	6
Setting path to API.....	6
Disabling third party file check of API implementation.....	8
Using API in Visual Studio Code.....	9
Coding best practices.....	10

Samples of API usage in ZeroBrane Studio

```
83 function OnUnitSpawned(uid, tag)
84     -- called when any unit is spawned
85     if tag == "colossus" then
86         local unit = C_GetU
87     end
88 end
89 -----
90
91
92
```

C_GetUnit

C_GetUnitPosInScene

C_GetUnitsAt

C_GetBuilding

C_GetBuildingByFloor

ZB Studio - API sample 1

```
83 function OnUnitSpawned(uid, tag)
84     -- called when any unit is spawned
85     if tag == "colossus" then
86         local unit = C_GetUnit(
87     end
88 end
89 -----
90
91
92
```

(UnitInstance) C_GetUnit (uid: integer)

Parameters:

- uid: integer -- Unit UID

Returns:

- unit: UnitInstance -- Unit table; Or nil

ZB Studio - API sample 2

Samples of API usage in Visual Studio Code

```
function OnUnitSpawned(uid, tag)
    -- called when any unit is spawned
    if tag == "Colossus" then
        local unit = C_GetU
    end
end
-----
```

C_GetUnit(uid)

C_GetUnitPosInScene(unitUID, visibleOnly)

C_GetUnitsAt(pos, r, floorUID)

C_GetAllUnits(playerIndex)

C_GetEnemyUnitsAt(pos, r, floorUID, heroesOnly)

C_GetFriendlyUnitsAt(pos, r, floorUID)

C_GetBuildingsAt(pos, r)

C_GetLumberAmountAt(pos, r)

C_GetPurchasableHeroItems()

C_GetBuilding(uid)

C_GetAllBuildings(playerIndex)

C_GetClosestLumberTarget(pos, r)

function

function C_GetUnit(uid: integer)

-> unit: UnitInstance

@param uid — Unit uid

@return unit — Unit table (See Lua Structures / Unit Instance); Or nil

VS Code - API sample 1

```
function OnUnitSpawned(uid, tag)
    -- called when any unit is spawned
    if tag == "Colossus" then
        local unit = C_GetUnit(uid)
        local unitType = unit.
    end
end
-----
```

orderBuildingType

orderUnitType

ownerIndex

pos

siegeDirection

targetBuildingUID

targetPos

targetShipUID

targetUnitUID

tier

type

uid

string

(field) UnitInstance.type: string

VS Code - API sample 2

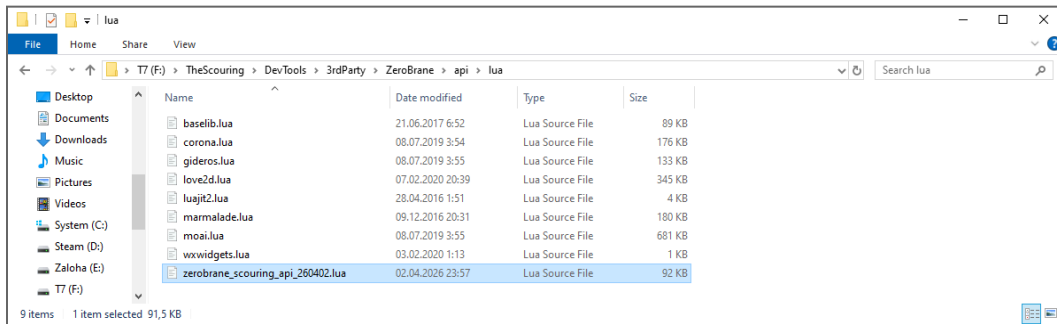
How to import API to ZeroBrane Studio

Integrating the API in *ZB Studio* can be done in two ways. Either way, the API file must be placed in the appropriate directory in the *ZeroBrane* folder structure. The first method is quick and easy, but updating the API file may get tedious over time.

The second, and faster, way to integrate the API is to use the provided *loader* script that automatically loads the latest API version available. The script loads the API on startup, if any changes are made to the settings or the API file, the *ZB Studio* must be restarted. Both ways of integrating the API will be described below.

Adding API file into Lua API folder

Download the API file for *ZB Studio* and put it into the `ZeroBrane/api/lua` folder. The API file must be in this folder no matter what following integration you choose.

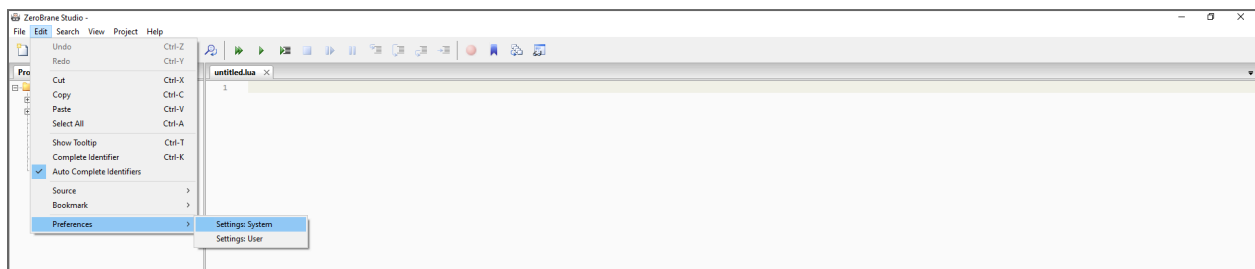


Windows - Adding API file to ZeroBrane API folder

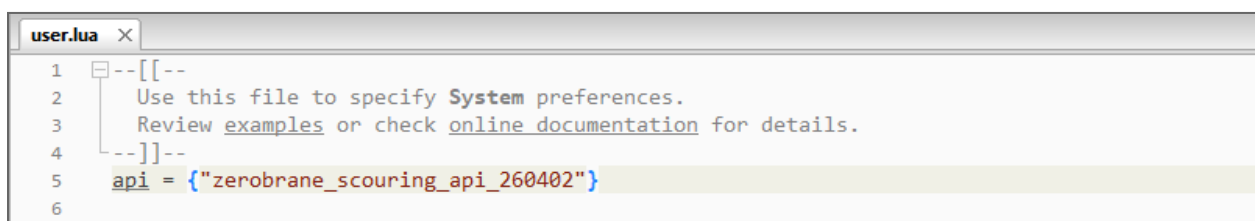
Static API integration

Start the *ZeroBrane Studio* and open the *Edit* pane from the top bar. Navigate to *Preferences* and choose either *System* or *User* settings.

Note, system settings will create the `user.lua` profile in the ZeroBrane package, while the user settings will create the profile in your Windows user documents. System settings are not tied to your PC, but only to the ZeroBrane package. You may need to set up the API again when changing to a new Dev Tools package. User settings will persist on your PC, which is desired for common developer purposes.



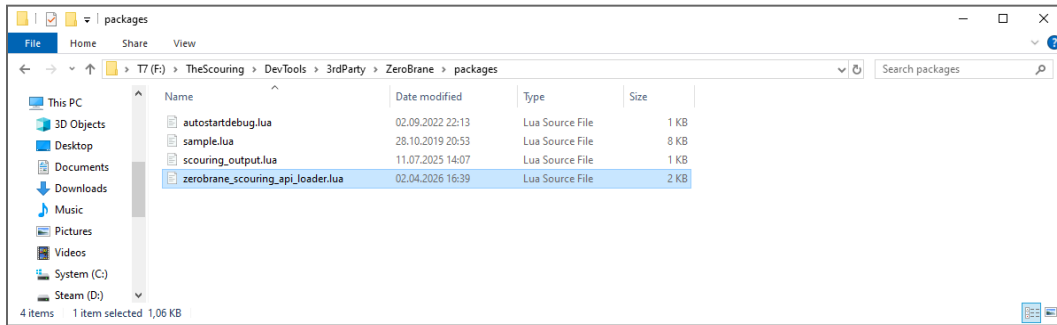
ZB Studio - Opening system preferences settings



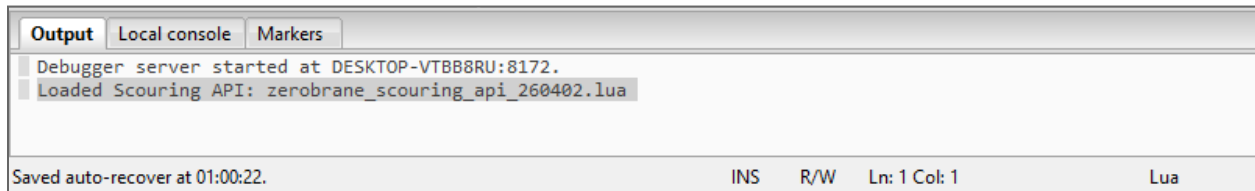
ZB Studio - Static API declaration in preferences settings

Dynamic API integration

For the purposes of easily updating the API file, a simple API loader script was created. You can download this script together with the API file. This script will search the `ZeroBrane/api/lua` folder for files starting with `zerobrane_scouring_api_`. If multiple files are present, the one with the latest date suffix will be chosen. The API loader script must be placed in the `ZeroBrane/packages` folder, so it can execute automatically on editor startup. The script will print a message to the *Output* window describing the loading result.



Windows - Adding API loader script to ZeroBrane packages folder



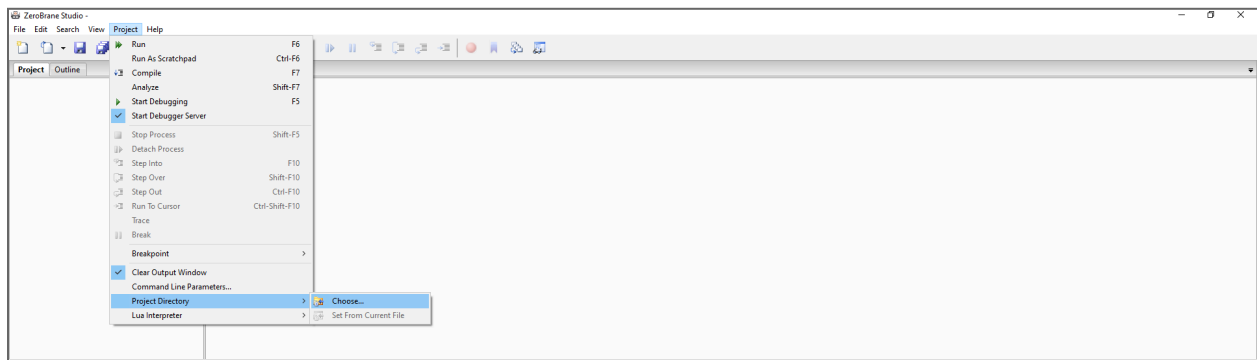
ZB Studio - Example message from API loader script

Using API in ZeroBrane Studio

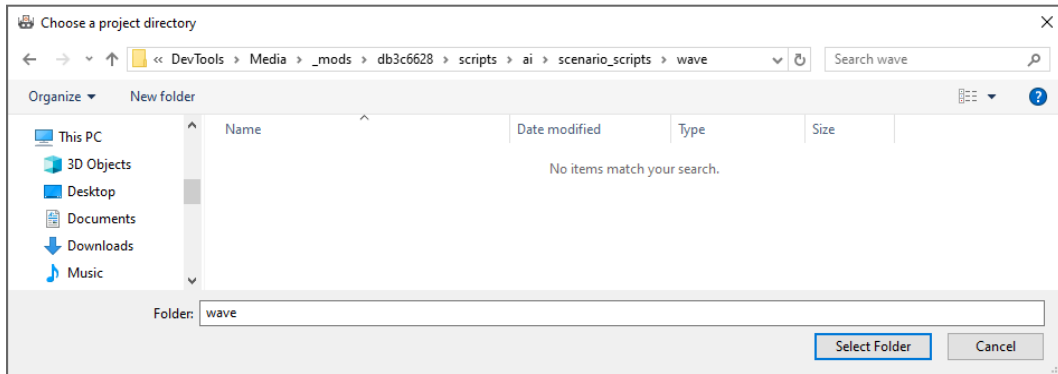
When you are starting to work on your scripts, don't open each script individually. Instead, open a folder with all of the scripts, so it becomes your working directory in the editor. You can also map a second directory to the project so you get some additional information about the functions declared here.

To set the current working directory, select *Project Directory* from the *Project* pane in the top toolbar. Choose a folder containing your script or one of its parent folders for broader context. Then, right-click the project folder in the *Project* view on the left. Select *Map Directory* and choose the `DevTools/Media/scripts` folder.

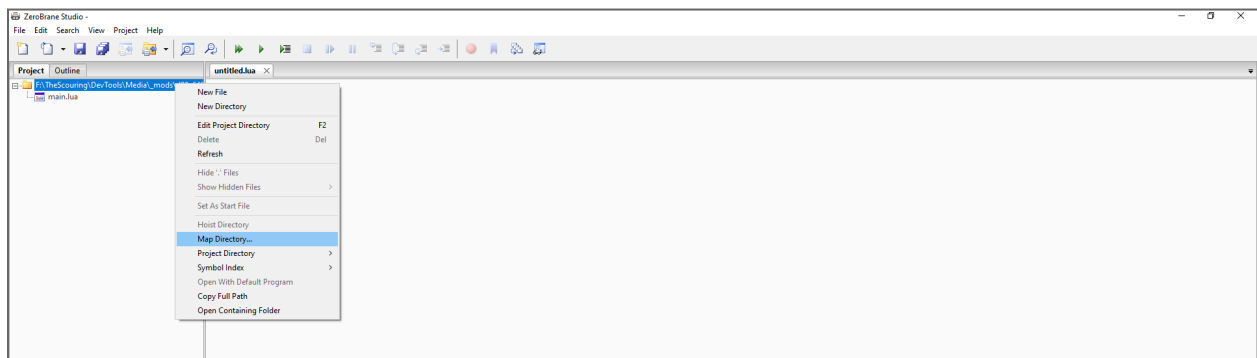
Auto-complete options are shown automatically after typing 2 characters, but can also be triggered by pressing **Ctrl+K**. Tooltips can be displayed by hovering your mouse over a function, or by pressing **Ctrl+T** during writing.



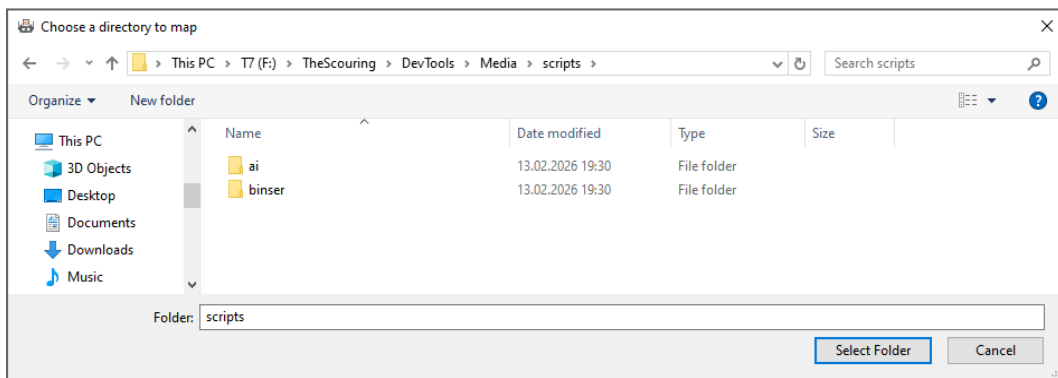
ZB Studio - Opening project directory



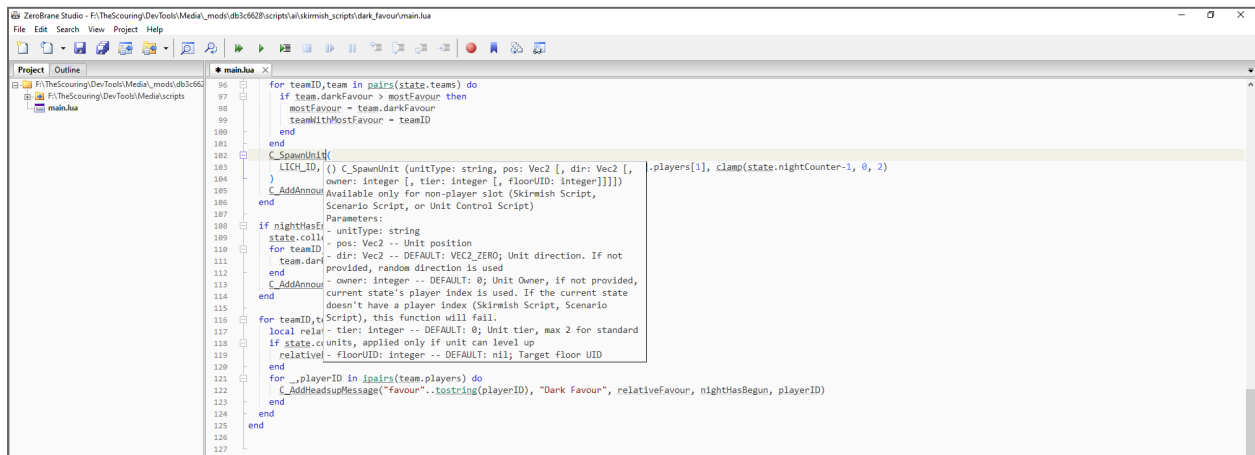
Windows - Selecting project directory



ZB Studio - Mapping directory to project scope



Windows - Selecting mapped directory



ZB Studio - Sample script with API tooltip

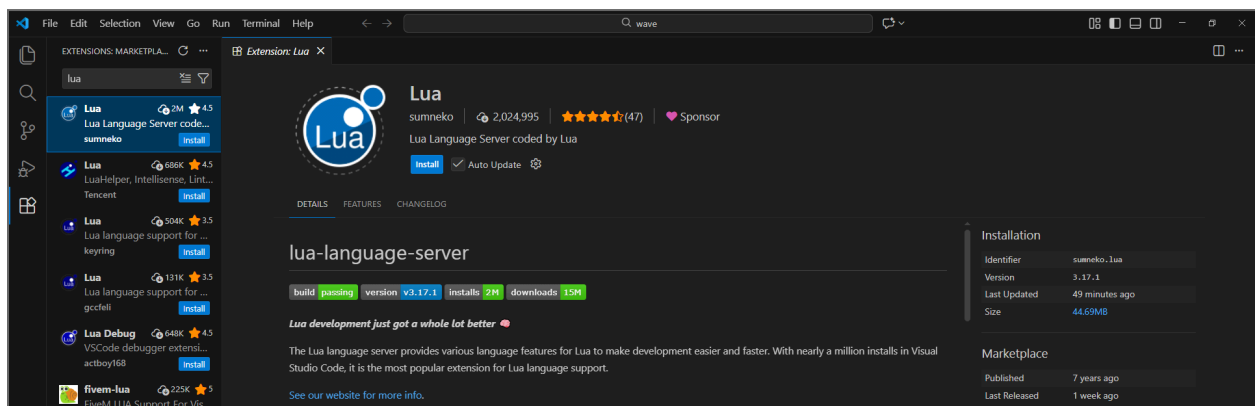
How to import API into Visual Studio Code

Integrating the API in VS Code can be done in two ways. The first way is to put the API file into the working directory of the editor. This is a quick and easy method, but copying the file to every project may get tedious over time. You would also upload the API file on *Steam* this way, even though this is unnecessary since it doesn't contain any functionality.

The second, and more scalable way to integrate the API is to set up a folder with third party libraries. This way the editor pre-loads the API file on startup, and then it works all the time. How to pre-load the API on start up will be described below.

Installing Lua Language Server extension

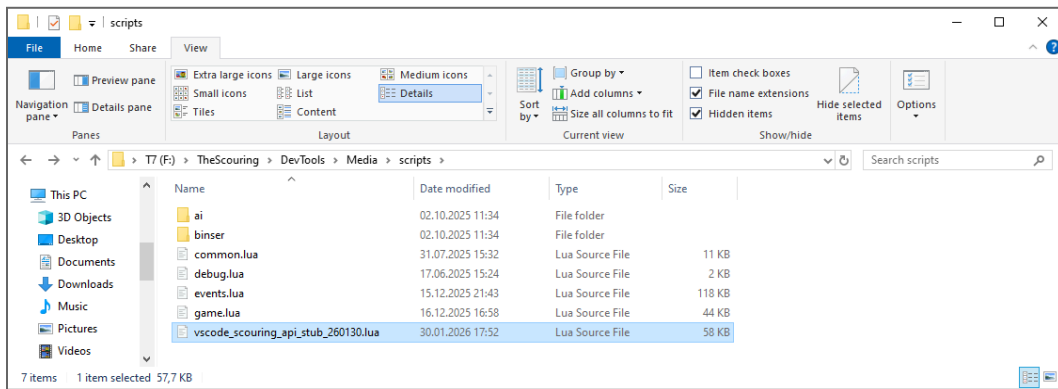
First, you will need to install a *Lua language server* extension into the editor. Start the *VS Code* and open the extension pane from the side bar. Search for *Lua*, and find the one created by *sumneko*. Install the extension.



VS Code - Lua extension

Setting path to API

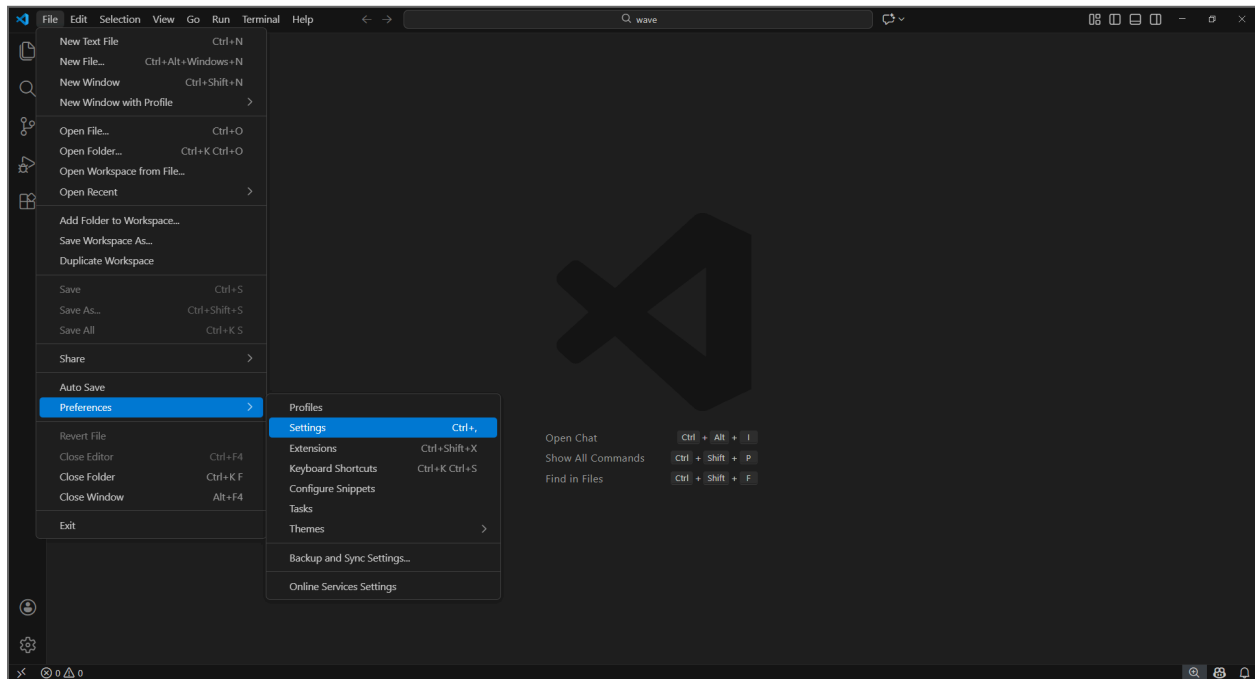
Download the API file for VS Code and put it into the `DevTools/Media/scripts` folder in the *Dev Tools* package. This folder will be used as the third party library for *VS Code*. The advantage of using this folder is that, apart from documented API functions, you will also get code completion for other Lua functions shipped with the *Dev Tools*.



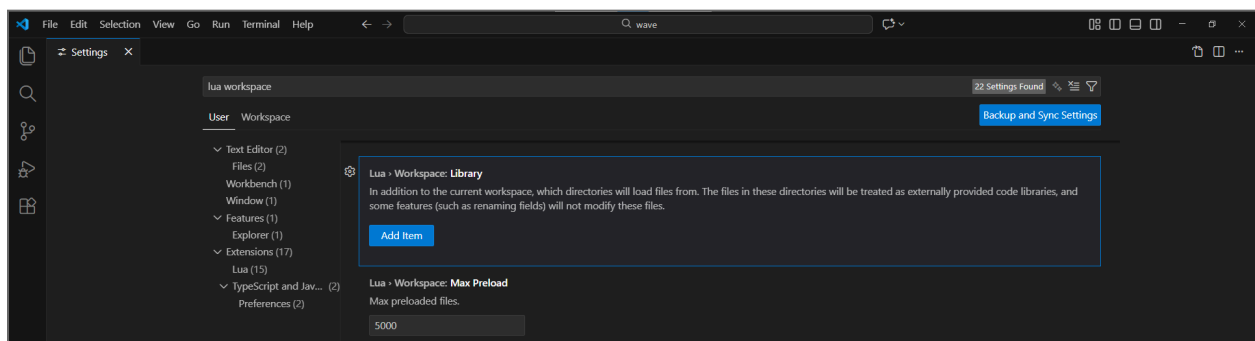
Windows - Adding API file into Developer Tools script folder

Note, that before you proceed, there are two options for saving the editor settings: User and Workspace. Choose user settings so that changes always remain relevant for any project.

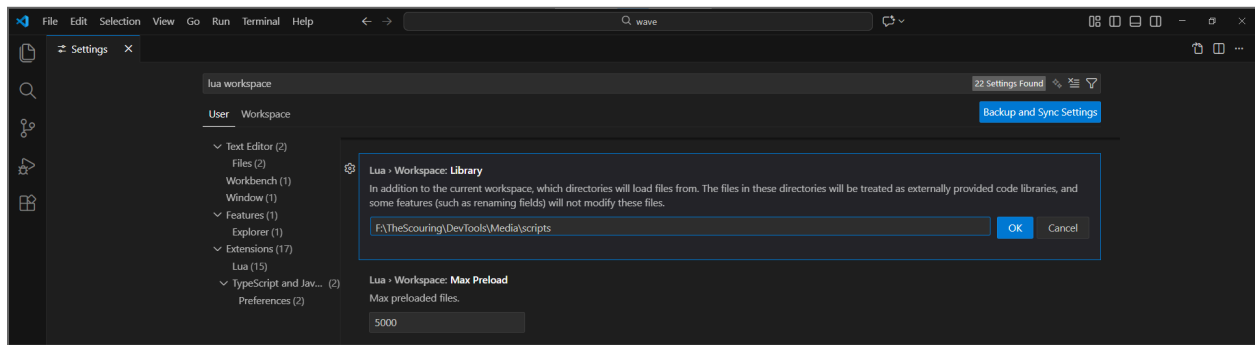
The next step is to tell the editor where the API is. Open editor **Settings** from a toolbar at the top. Type **lua workspace** into the search bar and find the **Library** settings. Press **Add Item** and insert a path that leads to the folder with scripts.



VS Code - Settings location in toolbar



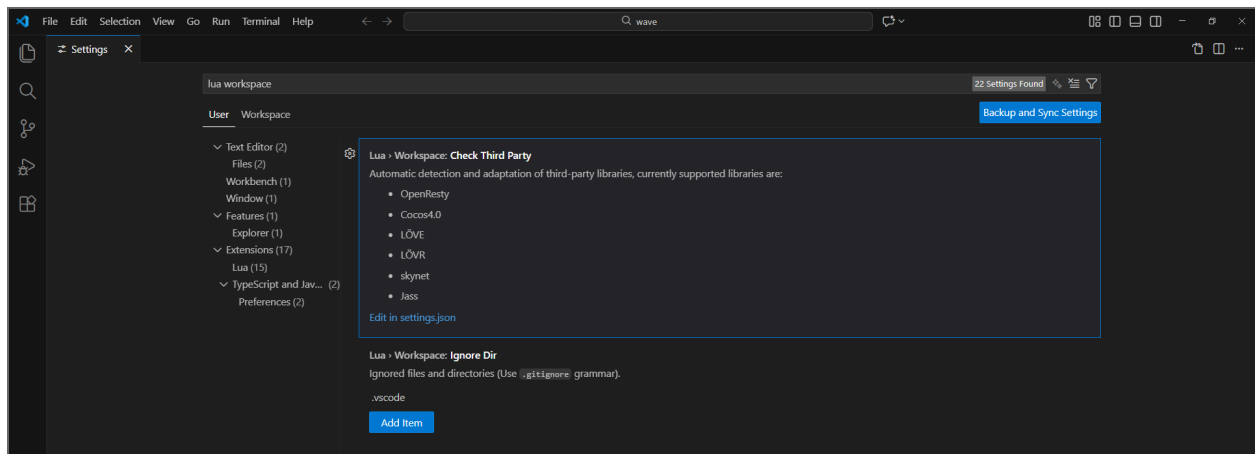
VS Code - Workspace library



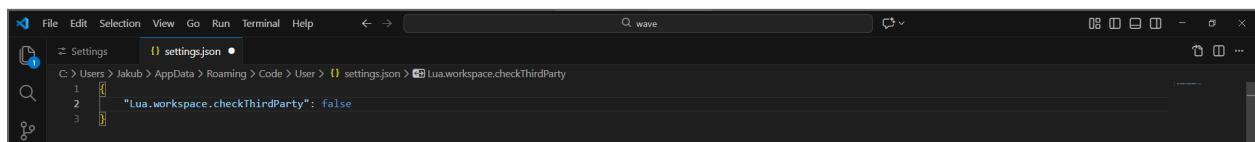
VS Code - Adding path to library

Disabling third party file check of API implementation

Unfortunately, files in the third party library are treated as fully implemented, but our API stub isn't a full API. The editor will warn you that a function lacks implementation whenever you use one from the API. To suppress the annoying warning, we can disable the option for checking third party files' integrity. Type again `lua workspace` into the search bar and find the *Check Third Party* option. Click on the *Edit in settings.json* and change the value of the setting to `false`.



VS Code - Automatic check of third party files



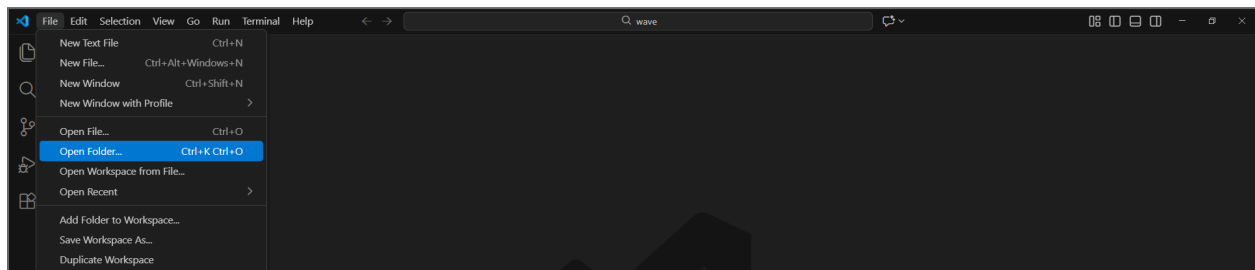
VS Code - Disabling third party files check

Using API in Visual Studio Code

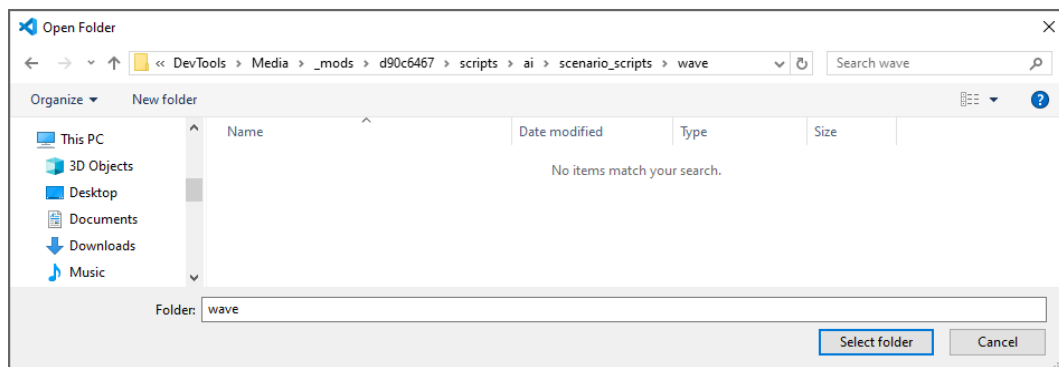
When you are starting to work on your scripts, don't open each script individually. Instead, open a folder with all of the scripts, so it becomes your working directory in the editor. This way the editor will also be aware of the context of any Lua scripts present in the working directory.

To set the current working directory, select *Open Folder* from the *File* pane in the top toolbar. Choose a folder containing your script or one of its parent folders for broader context. Then, open a script for editing, and you are ready to start scripting with intelligent code completion tools.

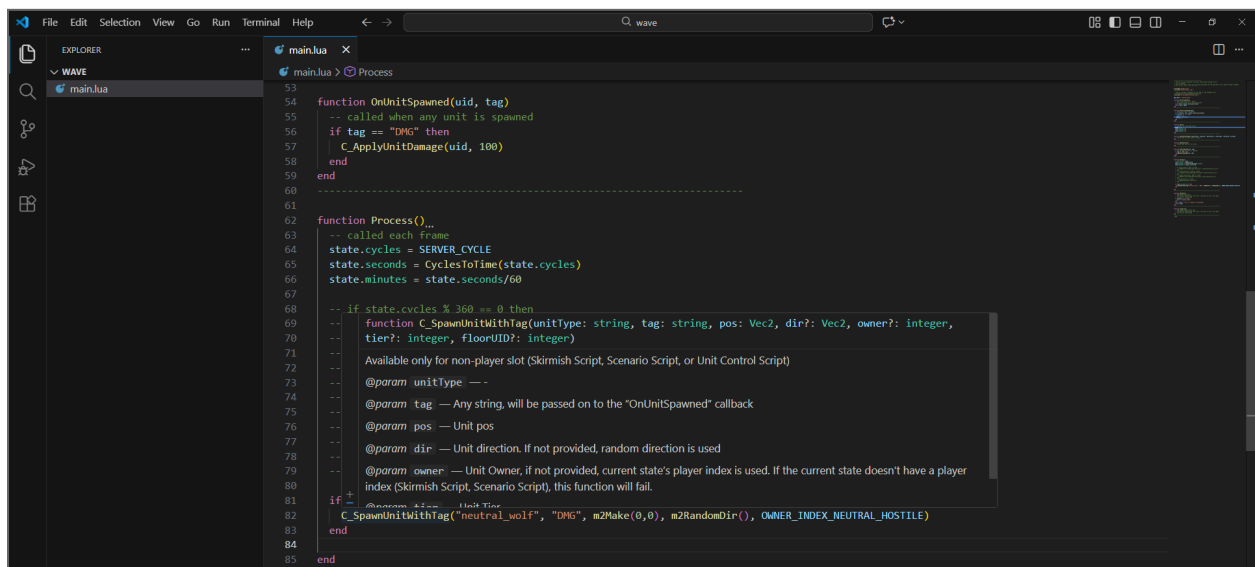
Auto-complete options are shown automatically when typing, but can also be triggered by pressing **Ctrl+Space**. Tooltips can be displayed by hovering your mouse over a function or a variable, or by pressing **Ctrl+Shift+Space** during writing. If you need to see a declaration of a function, then hold **Ctrl** and click on the function call in your script.



VS Code - Opening working directory



Windows - Select working directory



VS Code - Sample script with API tooltip

Coding best practices

After opening a pre-generated script by the game editor, you will notice that VS Code warns you about using lowercase for global variables. This is considered bad practice because it makes it difficult to distinguish between global and local variables. You can either rename the variable by pressing **F2**, suppress the warning, or declare the variable as local global (upvalue).

```
52 function Start()  
53 |   -- called once, when game starts  
54 end
```

*VS Code - Remove **state** declaration from **Start***

```
15 local state = {}  
16 -----  
17  
18 function SerializeState()  
19 |   -- saves lua state into a buffer  
20 |   -- you should save all of your state here  
21 |   local data = binser.serialize(state)  
22 |   return data, #data  
23 end
```

*VS Code - Declare **state** as local global variable*