

Modding Autonauts

This document contains all the information on how to start Modding Autonauts!

Autonauts uses Lua script for modding.

The modding system is continually being improved upon and added to.

If you don't see what you are looking for or are not sure if the functionality exists then head on over to the Official Autonauts Discord and give us a shout in the 'modding' channel.

Discord: <https://discord.gg/xXRfjsc>

This document is intended to get you up and running with an example as well as answering any main ideas.

If you are looking for the documentation/function reference, you can find that here:

<http://www.denki.co.uk/autonauts/modding/>

How do I create a Mod?

Simply locate to the 'StreamingAssets' folder in your Autonauts directory in your Steam library i.e.

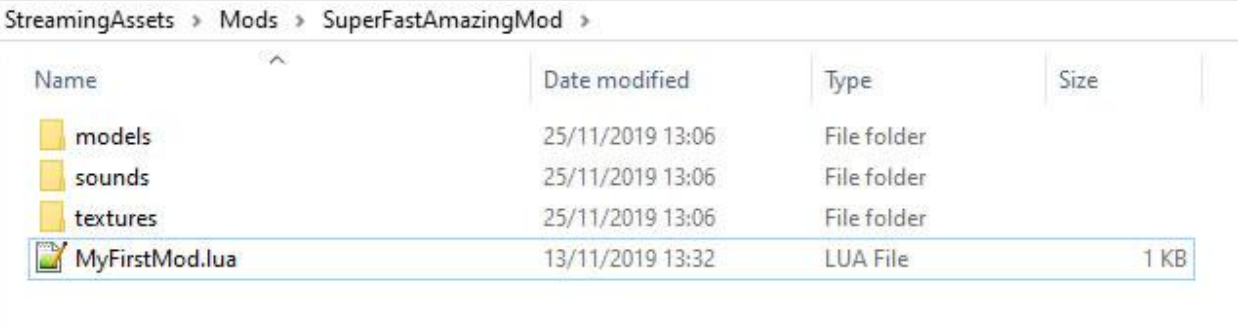
1. Open Autonauts and navigate to the Mod Menu (from Main Menu).
2. Click 'Open Folder' next to 'Local Mods'

In this folder (StreamingAssets), create a new folder called 'Mods' if it's not already there.

Inside the Mods folder you can now create your own mod by following these steps:

1. Create a folder with the name of your mod e.g. 'SuperFastAmazingMod'
2. Inside your folder, create three subfolders 'Sounds', 'Models' and 'Textures'
3. Create your LUA script e.g. 'MyFirstMod.lua'

It should look like this:



Name	Date modified	Type	Size
models	25/11/2019 13:06	File folder	
sounds	25/11/2019 13:06	File folder	
textures	25/11/2019 13:06	File folder	
MyFirstMod.lua	13/11/2019 13:32	LUA File	1 KB

Now all you have to do is decide what you want to do in your Mod!
Check the rest of the documentation for possibilities.

Lua and Structure:

The Lua file must conform to Lua 5.2, but above this there are key functions required by the game - ones you must use:

function SteamDetails - Used near exclusively for Steam Workshop and Mod information

function Expose - Here you can expose any variables to the game for settings

function Creation - Used to create any custom converters or buildings

function BeforeLoad - Initial load function by game

function AfterLoad - Once a game has loaded key functionality, this is called.

function AfterLoad_CreatedWorld - Only called when creating a game. [v134.23]

function AfterLoad_LoadedWorld - Only called on loading a game. [v134.23]

function OnUpdate(DeltaTime) - Called every frame, see also [Time.deltaTime](#)

NOTE: THE COLOURS OF THE FUNCTIONS

Throughout the documentation, the colours of the instructions will match the colours of the functions e.g. An **instruction this colour** means use in **function AfterLoad** in Lua.

Lua Resources:

As mentioned earlier the Lua script conforms to Lua 5.2 (with a few exceptions).

For more information/learning about Lua, here are some links:

Lua 5.2 reference:

<http://www.lua.org/manual/5.2/contents.html#contents>

Lua resources and great examples of code:

<http://lua-users.org/wiki/LuaDirectory>

Another great Lua resource:

<https://www.tutorialspoint.com/lua/>

If you are getting errors or want to check your Lua then try this site:

[F8 to run your code]

https://rextester.com/l/lua_online_compiler

Autonauts Lua allows:

GlobalConsts | TableIterators | String | Table | Basic | Math | Bit32 | Metatables | ErrorHandling |
Coroutine | OS_Time | Dynamic | Json

An Example Mod:

If you're looking for an example mod - head over to the Steam Workshop and search for '**BerryGoodMod**'. This example Mod shows how to make a new berry producing converter alongside a new recipe.

It also demonstrates spawning a berry and changing storage amount for sticks and berries.

The Mod will install into your steam workshop directory, to find it:

1. Open Autonauts and navigate to the Mod Menu (from Main Menu).
2. Click 'Open Folder' next to 'Subscribed Mods'

Let's take a look at this Lua file:

<http://www.denki.co.uk/autonauts/modding/Examples/DefaultExample.txt>

First setup the details about the Mod in the SteamDetails() function. This is used in game and for uploading to Steam.

function SteamDetails()

-- Setting of Steam details

ModBase.SetSteamWorkshopDetails("BerryGoodMod V2", "Mega Berry Maker. Adds converter for berries along with recipe.", {"berries", "berry converter"}, "BerryLogo.png")

end

The Creation() function is where new custom items like Food or Converters, Hats etc. are specified.

function Creation()

-- Creation of any new converters or buildings etc. go here

-- MOD - Create a new Berry Converter

ModConverter.CreateConverter("TheBerryMaker", {"Berries"}, {"Plank", "Pole"}, {4, 3}, "ObjCrates/wooden boxes pack", {-1,-1}, {1,0}, {0,1}, {1,1})

end

Next is the BeforeLoad() function, this is called when you try to load/create a new game before anything else.

function BeforeLoad()

-- Before Load Function - Pre-loading of map calls go here

***-- MOD - Create Berry Recipe - All Converters - 1 stick = 10 berries produced
ModVariable.SetIngredientsForRecipe("Berries", {"Stick"}, {1}, 10)***

***-- MOD - Set Storage for Sticks to 200
ModVariable.SetVariableForStorageAmount("Stick", 200)***

***-- MOD - Set Storage for Berries to 200
ModVariable.SetVariableForStorageAmount("Berries", 200)***

end

AfterLoad() function is called once all else is loaded/created when loading/creating a game.

function AfterLoad()

-- After Load Function - Anything after map creation goes here

***-- MOD - Spawn Berry at (50,50)
ModBase.SpawnItem("Berries", 50, 50)***

end

Uploading a Mod to Steam Workshop:

When a Mod is created it will show in the local mods section of the Mod Menu.

To upload a mod to Steam Workshop it's simply a case of:

1. Selecting the desired Mod from the list then clicking 'Upload to Steam'.
2. This opens a pop-up asking you to confirm, note at this point you must confirm you accept the steam agreement.
3. After upload is complete the steam overlay will show your Mod.

To be able to upload a mod to Steam Workshop certain details must be set.

You can set these in Lua, example:

```
function SteamDetails()  
    ModBase.SetSteamWorkshopDetails("MyFirstMod", "Description", {"tag1", "tag2"},  
    "ThisIsTheLogo.png")  
end
```

The image can be either 'png' or 'jpg' but must be less than 1MB in size.

Place the image in the 'textures' folder in your mod.

A title, description, image and tags must exist.

Disabling and Deleting a Mod

All mods, whether local or installed can be disabled.
In the mods menu, click the mod options and untick the 'Enabled' box to disable.

What does Disabled mean??

A disabled mod will mean you can no longer use the lua functions and mod creations (from the disabled Mod) in a game. However, if a game save already contains a mod custom creation - like a converter - this will still work. You will not be able to add another custom converter from a disabled mod but a pre-existing custom object will still function.

What if I delete/Uninstall the Mod?

If the mod is removed (either by deleting (locally) or uninstalling/unsubscribing (Steam)) then none of the functionality will work. If you have a custom mod item in your map then with the Mod removed, that item will be deleted from your map.

Reloading Scripts:

The Mods menu allows for (Lua) scripts to be reloaded during runtime.
All standard functions will be reloaded and work as expected apart from any custom creations from the Creation() function.
Custom created items will always require the game to be restarted due to model loading and initiation processes.

Game Types: Objects, Tiles etc.

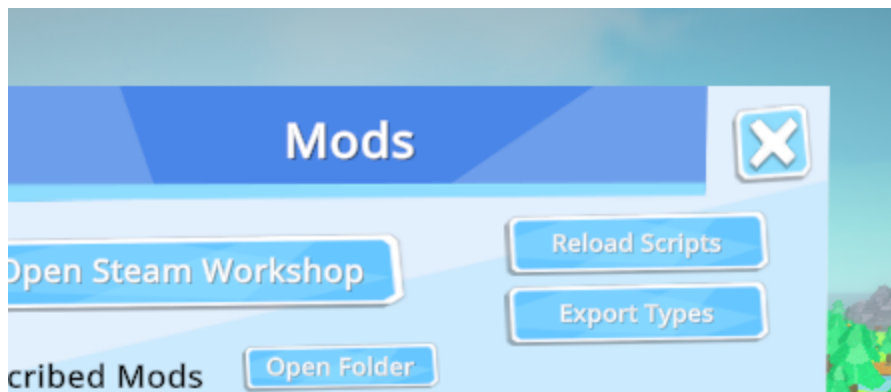
A list of the in game models, object types, tile types, states, sounds etc. can be generated by the game at any point.

Supports:

- Object Types e.g. 'Berries'
- Tile Types e.g. 'Soil'
- Game States e.g. 'Normal'
- Game Models e.g. 'Models/Other/FarmerPlayer'
- Farmer States e.g. 'Moving'
- Audio Events e.g. 'AmbienceRain'

How to generate the list:

1. Open the Mods Panel
2. Click on 'Export Types'
3. The output folder will be opened upon completion.



Error Output:

When you have created a local mod it will show in the Mods panel (under local mods).

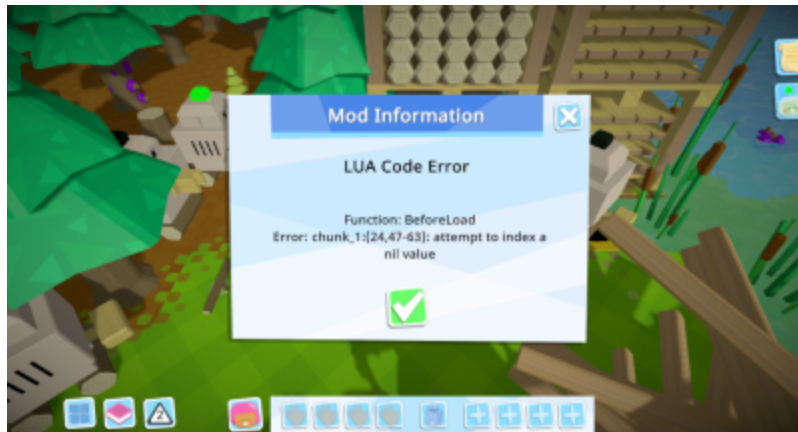
This means it's automatically enabled for playing Autonauts.

Important: If you make a script change while Autonauts is open, enable "Dev Mode" and launch a game from the Main Menu. Changes to Creation() will always require a restart.

If at any point your script has an error (syntax) or causes a logic error in the game, a pop-up will show when you start playing in game.

The error is also dumped to a text file, located in the 'Mods' folder (under 'StreamingAssets')

Below is an example of the error popup.



The example shows the function in Lua (which is BeforeLoad function in this case), the line 24 and the chars on the line (47-63) and the error itself.

Debug/Print Logging

If at any point you want to print out what is going on in your Lua script (log to file) then this can be done via the command:

- ***ModDebug.Log("Hello World!")***

It can also be used to check variables as any other print/debug log functionality would do. IMPORTANT NOTE: it does not adhere to a typical style of 'print("var is %d",intvar)' which Lua does. Instead it uses a much more user friendly style (akin to Unity) like:

- ***local amazingVar = 10***
- ***ModDebug.Log("My var is ", amazingVar)***

This would log:

"My var is 10"

To the file, which can be found at **\StreamingAssets\Mods\ModLog.txt**

Every variable or string statement must feature a comma between them.

Like so:

- ***Local basic = 0***
- ***ModDebug.Log(basic," Anyone there? ", basic, " hello? ", basic)***

If you ever want to clear the log manually at a certain point in your script, you can do so via:

- ***ModDebug.ClearLog()***

NOTE: It's cleared automatically on each startup.

Settings & Exposing Variables

Enabling/Disabling Mods:

All Mods by default contain an enable/disable flag. Naturally, this enables/disables the mod.

Exposing Variables from Lua:

Variables can be exposed to the Mod settings/options screen like so:



The image shows the default 'Enabled' setting but also four other settings, straight from the Lua file.

Every exposed variable must be handled within the **function Expose()**.

Function calls:

- ModBase.ExposeVariable(Name, Default Value, Callback)
- ModBase.ExposeVariable(Name, Default Value, Callback, Min, Max)

Where:

Name is the unique name of your variable.

Default Value is the default value of the variable, you can pass the actual variable.

Callback is the function to call when value is changed.

Min and Max are only applicable to numbers and can be ignored (e.g. if using a bool).

You must specify the min/max if using a number. The range of allowance.

A Lua example of setting some exposed variables and also getting them back from the game:

```
local Speed = 10
```

```
local Win = false
```

```
function Expose()
```

```
    ModBase.ExposeVariable("SpeedVar", Speed, SpeedCallback, 0, 20)
```

```
    ModBase.ExposeVariable("Instant Win", Win, AnotherCallback)
```

```
end
```

```
function SpeedCallback( param )
```

```
    Speed = param
```

```
end
```

```
function AnotherCallback( param )
```

```
    ModDebug.Log( "Returned: ", param )
```

```
end
```

The Mod Options menu is accessible from the Mods Menu itself. Select a local or subscribed mod and click the options button.

Custom 3D Models:

If you are looking to create something more custom like a converter or decorative building then a model will be required.

All models must be in .OBJ format and be exported with/contain a .MTL file for the material creation.

Any 3D art package should be able to produce an .OBJ format export. Blender is a great example application - it's free or you can download an .OBJ model from the internet.

The game uses a flat style of shading, one without any specular or emissive. Obviously no PBR either. To achieve a similar feel, set any 'Ks' or 'Ke' to near 0 in the .MTL file, unless you want to achieve that look. The diffuse and ambient ('Kd' and 'Ka') are better suited.

Guide to Autonauts with Blender:

A fantastic guide to using blender for Autonauts written by community member Psylem can be found here:

https://docs.google.com/document/d/18YwQpZD8awG-p7caCakf84K_UZTIYoDcINPoxu3scDA/edit

Example Blender Files:

Download some of the in game Blender files to help show sizes, rotations etc.

<https://www.dropbox.com/s/yh9g4v6mr4v8js3/ExampleModels.zip?dl=1>

Example Mod:

Need an example? Subscribe to the 'BerryGoodMod' and locate the model file.

The BerryGoodMod will download to:

-Your Steam Library- \steamapps\workshop\content\979120\1977627950

Inside you will see the 'models' folder and the .LUA file; these provide a good example.

Model Functions:

When using a custom model to make a custom object like a building, converter, decorative, holdable or a food item then you may want to adjust it's parameters without having to tweak in an art package.

You can change the scale, rotation and translation (together or separately):

Example of changing a model used to create a food item:

- ***ModFood.UpdateModelParameters("Cupcake", 0.2, -90,0,0, 0,0,1)***

The above is altering a model used to create a food item called 'Cupcake' - hence the use of 'ModFood' calling it.

The parameters are: (after the Item name)

float Scale,

float RotationX, float RotationY, float RotationZ,

float TranslationX, float TranslationY, float TranslationZ

If only one of the adjustments is required it can be called like so:

- ***ModFood.UpdateModelScale("Cupcake", 0.2)***
- ***ModFood.UpdateModelScaleSplit("Cupcake", 0.2,1,1)***
- ***ModFood.UpdateModelRotation("Cupcake", -90,0,0)***
- ***ModFood.UpdateModelTranslation("Cupcake", 0,0,1)***

NOTE: It must be called after the item is created and also in the Creation() function only.

Sounds & Audio

Game Event Sounds:

Make sure you have a folder in your Mod folder called 'sounds'
 Inside there, place your new .WAV file (e.g. Monster.wav)
 Now let's try to change the Cow eating sound...
 All sound changes can be done in the 'OnStart' Lua function.

In lua script:

```
function BeforeLoad()
  ModSound.ChangeSound("AnimalCowEating", "Monster")
  ModSound.ChangeVolume("AnimalCowEating", 20)
  ModSound.ChangePitch("AnimalCowEating", 2)
end
```

For a list of Audio Events - [see here](#)

Game Music:

Similar to event sounds you must place your audio WAV file in the 'sounds' folder in your Mod.
 Let's change the game audio to be using 'Halloween.wav'.
 Here is an example:

- ***ModSound.ChangeAllGameMusic("Halloween")***

You can also change the day music or the night music specifically:

- ***ModSound.ChangeDayGameMusic("Halloween")***
- ***ModSound.ChangeNightGameMusic("Halloween")***

Even the Loading, Menu and About screen music:

- ***ModSound.ChangeLoadingMusic("Halloween")***
- ***ModSound.ChangeAboutMusic("Halloween")***

Volume of the music can be changed also:

- ***ModSound.ChangeMusicVolume(0.0)***

Custom Sounds: [v134.32]

You can now play custom .wav files at any point.

Example:

- ***ModSound.PlayCustomSound("HalloweenBoom")***

Altering Variables Of The Game

Modifying the Main Character's variables:

Note: The main character you play as is referred to as the 'FarmerPlayer' in code, and variables that are shared by both the player and bots refer to a 'Farmer'.

Setting the Farmer's 'Action' on an '**Object**' with an 'ObjectTool' for 'X' amount

Example: Wanting the Farmer to chop a pine tree with a stone axe in 2 motions. (Default is 24).

- `ModVariable.SetVariableFarmerAction("Chopping", "TreePine", "ToolAxeStone", 2)`

Setting the Farmer's 'Action' on a '**Tile**' with an 'ObjectTool' for 'X' amount

Example: Wanting the Farmer to mine an Iron tile with a StonePick in 2 motions. (Default is 20).

- `ModVariable.SetVariableFarmerActionOnTiles("Mining", "Iron", "ToolPickStone", 2)`

All 'Farmer Action' changes can be done in the 'BeforeLoad' Lua function.

NOTE: Obviously not all combinations work with all combinations - common sense required!

For a list of Types - [see here](#)

Setting the player speed

The speed is considered the delay in movement (i.e. smaller = faster)

Example:

- `ModVariable.SetVariableForObjectAsFloat("FarmerPlayer", "BaseDelay", 0.2)`

Carry Size/Inventory size:

- `ModVariable.SetVariableForObjectAsInt("FarmerPlayer", "CarrySize", 4)`
- `ModVariable.SetVariableForObjectAsInt("FarmerPlayer", "InventoryCapacity", 1)`

Setting the main character's upgrades

Examples:

- Set "Capacity" of Player Inventory at Crude level.
ModVariable.SetVariableForObjectAsInt("UpgradePlayerInventoryCrude", "Capacity", 1)
- Set "Capacity" of Player Inventory at Good level.
ModVariable.SetVariableForObjectAsInt("UpgradePlayerInventoryGood", "Capacity", 3)
- Set "Capacity" of Player Inventory at Super level.
ModVariable.SetVariableForObjectAsInt("UpgradePlayerInventorySuper", "Capacity", 5)
- Set "Delay" of Player Movement at Crude level.
ModVariable.SetVariableForObjectAsFloat("UpgradePlayerMovementCrude", "Delay", 0.05)
- Set "Delay" of Player Movement at Good level.
ModVariable.SetVariableForObjectAsFloat("UpgradePlayerMovementGood", "Delay", 0.075)
- Set "Delay" of Player Movement at Super level.
ModVariable.SetVariableForObjectAsFloat("UpgradePlayerMovementSuper", "Delay", 0.1)

Modifying a Tool's variables:

Tools are classified as 'Objects' within the game.

You can set the variable for any tool by using one of three types of calls:

Set as Integer:

- ***ModVariable.SetVariableForObjectAsInt(string Object, string Variable, int Value)***

Set as Float:

- ***ModVariable.SetVariableForObjectAsFloat(string Object, string Variable, float Value)***

Set as String:

- ***ModVariable.SetVariableForObjectAsString(string Object, string Variable, string Value)***

Setting a Tool's Life Example:

- ***ModVariable.SetVariableForObjectAsInt("ToolShovelStone", "MaxUsage", 20)***

Setting a Tool's Capacity Example:

- ***ModVariable.SetVariableForObjectAsInt("ToolBucketCrude", "Capacity", 1)***

Setting a Tool's Scoop Animation Requirements Example:

- ***ModVariable.SetVariableForObjectAsInt("ToolBucketCrude", "FillScoops", 7)***

Modifying a Bot's variables:

Note: A Bot is referred to as a 'Worker' in code.

Every bot can be considered to have a 'HEAD', a 'FRAME' and a 'DRIVE' part.

When using Lua we need to pass 3 pieces of information to set a bot variable; the 'Part' (from above), the variable to change and the value desired.

The 'Part' e.g. "WorkerHeadMk0", is for a Mk0's (first bot) head part. Whereas "WorkerFrameMk1" is for a Mk1's frame part.

HEAD variables with examples:

- "Max Instructions" - How many lines of 'code' a bot can allow (12 is base).
ModVariable.SetVariableForObjectAsInt("WorkerHeadMk0", "MaxInstructions", 12)
- "SerialPrefix" - Like a serial number e.g. "Mk4".
ModVariable.SetVariableForObjectAsString("WorkerHeadMk1", "SerialPrefix", "Mk1")
- "FindNearestRange" - The range in which this bot can find objects.
ModVariable.SetVariableForObjectAsInt("WorkerHeadMk3", "FindNearestRange", 45)
- "FindNearestDelay" - The delay before the bot does this, larger number = slower.
ModVariable.SetVariableForObjectAsInt("WorkerHeadMk1", "FindNearestDelay", 30)
- "HeadScale" - The Scale of the head.
ModVariable.SetVariableForObjectAsFloat("WorkerHeadMk1", "HeadScale", 1.0)

FRAME variables with examples:

- "WorkingSoundName" - The sound the bot makes when working.
ModVariable.SetVariableForObjectAsString("WorkerFrameMk0", "WorkingSoundName", "WorkerWorkingClockwork")
NOTE: This can be used in conjunction with a sound mod.
- "CarrySize" - How many items this Bot can carry at once.
ModVariable.SetVariableForObjectAsString("WorkerFrameMk0", "CarrySize", 3)
- "InventorySize" - The inventory size of this Bot.
ModVariable.SetVariableForObjectAsInt("WorkerFrameMk0", "InventorySize", 1)
- "UpgradeSize" - The amount of upgrade slots for this Bot.
ModVariable.SetVariableForObjectAsInt("WorkerFrameMk0", "UpgradeSize", 1)
- "FrameScale" - The Scale of the frame.
ModVariable.SetVariableForObjectAsFloat("WorkerFrameMk0", "FrameScale", 1.0)

DRIVE variables with examples:

- "SpeedScale" - How fast this Bot moves.
ModVariable.SetVariableForObjectAsFloat("WorkerDriveMk0", "SpeedScale", 0.5)
- "MoveInitialDelay" - Delay before initial move is made.
ModVariable.SetVariableForObjectAsInt("WorkerDriveMk0", "MoveInitialDelay", 60)
- "RechargeDelay" - Time taken to recharge another Bot.
ModVariable.SetVariableForObjectAsFloat("WorkerDriveMk0", "RechargeDelay", 3.0)
- "DriveEnergy" - Energy supply of a Bot (before recharge).
ModVariable.SetVariableForObjectAsFloat("WorkerDriveMk0", "DriveEnergy", 90.0)
- "MoveSoundName" - Sound when moving.
ModVariable.SetVariableForObjectAsString("WorkerDriveMk0", "MoveSoundName", "WorkerCrudeMove")
- "MoveStoneSoundName" - Sound when moving stone.
ModVariable.SetVariableForObjectAsString("WorkerDriveMk0", "MoveStoneSoundName", "WorkerCrudeStoneMove")
- "MoveClaySoundName" - Sound when moving clay.
ModVariable.SetVariableForObjectAsString("WorkerDriveMk0", "MoveClaySoundName", "WorkerCrudeClayMove")
- "DriveScale" - The Scale of the drive.
ModVariable.SetVariableForObjectAsFloat("WorkerDriveMk0", "DriveScale", 1.0)

Setting the Bot's upgrade examples:

- Set 'Capacity' of the Bot's inventory upgrade at Crude level.
ModVariable.SetVariableForObjectAsInt("UpgradeWorkerInventoryCrude", "Capacity", 1)
- Set 'Capacity' of the Bot's inventory upgrade at Good level.
ModVariable.SetVariableForObjectAsInt("UpgradeWorkerInventoryGood", "Capacity", 2)
- Set 'Capacity' of the Bot's inventory upgrade at Super level.
ModVariable.SetVariableForObjectAsInt("UpgradeWorkerInventorySuper", "Capacity", 4)
- Set 'Capacity' of the Bot's carry amount upgrade at Crude level.
ModVariable.SetVariableForObjectAsInt("UpgradeWorkerCarryCrude", "Capacity", 1)
- Set 'Capacity' of the Bot's carry amount upgrade at Good level.
ModVariable.SetVariableForObjectAsInt("UpgradeWorkerCarryGood", "Capacity", 2)

- Set 'Capacity' of the Bot's carry amount upgrade at Super level.
ModVariable.SetVariableForObjectAsInt("UpgradeWorkerCarrySuper", "Capacity", 3)
- Set reduced Energy consumption (In Energy/Secs) upgrade of the Bot at Crude level.
ModVariable.SetVariableForObjectAsFloat("UpgradeWorkerEnergyCrude", "DriveEnergy", 0.2)
- Set reduced Energy consumption (In Energy/Secs) upgrade of the Bot at Good level.
ModVariable.SetVariableForObjectAsFloat("UpgradeWorkerEnergyGood", "DriveEnergy", 0.4)
- Set reduced Energy consumption (In Energy/Secs) upgrade of the Bot at Super level.
ModVariable.SetVariableForObjectAsFloat("UpgradeWorkerEnergySuper", "DriveEnergy", 0.6)
- Set Memory upgrade of the Bot's memory at Crude level.
ModVariable.SetVariableForObjectAsInt("UpgradeWorkerMemoryCrude", "Size", 4)
- Set Memory upgrade of the Bot's memory at Good level.
ModVariable.SetVariableForObjectAsInt("UpgradeWorkerMemoryGood", "Size", 8)
- Set Memory upgrade of the Bot's memory at Super level.
ModVariable.SetVariableForObjectAsInt("UpgradeWorkerMemorySuper", "Size", 32)
- Set 'Initial Delay' (time before move) upgrade of the Bot at Crude level.
ModVariable.SetVariableForObjectAsInt("UpgradeWorkerMovementCrude", "InitialDelay", 20)
- Set 'Move Scale' (movement multiplier) upgrade of the Bot at Crude level.
ModVariable.SetVariableForObjectAsFloat("UpgradeWorkerMovementCrude", "MoveScale", 0.1)
- Set 'Initial Delay' (time before move) upgrade of the Bot at Good level.
ModVariable.SetVariableForObjectAsInt("UpgradeWorkerMovementGood", "InitialDelay", 40)
- Set 'Move Scale' (movement multiplier) upgrade of the Bot at Good level.
ModVariable.SetVariableForObjectAsFloat("UpgradeWorkerMovementGood", "MoveScale", 0.2)
- Set 'Initial Delay' (time before move) upgrade of the Bot at Super level.
ModVariable.SetVariableForObjectAsInt("UpgradeWorkerMovementSuper", "InitialDelay", 60)
- Set 'Move Scale' (movement multiplier) upgrade of the Bot at Super level.
ModVariable.SetVariableForObjectAsFloat("UpgradeWorkerMovementSuper", "MoveScale", 0.3)
- Set 'Initial Delay' (time before search) upgrade of the Bot at Crude level.
ModVariable.SetVariableForObjectAsInt("UpgradeWorkerSearchCrude", "InitialDelay", 10)
- Set 'Range' (search range) upgrade of the Bot at Crude level.
ModVariable.SetVariableForObjectAsInt("UpgradeWorkerSearchCrude", "Range", 5)

- Set 'Initial Delay' (time before search) upgrade of the Bot at Good level.
ModVariable.SetVariableForObjectAsInt("UpgradeWorkerSearchGood", "InitialDelay", 20)
- Set 'Range' (search range) upgrade of the Bot at Good level.
ModVariable.SetVariableForObjectAsInt("UpgradeWorkerSearchGood", "Range", 10)
- Set 'Initial Delay' (time before search) upgrade of the Bot at Super level.
ModVariable.SetVariableForObjectAsInt("UpgradeWorkerSearchSuper", "InitialDelay", 30)
- Set 'Range' (search range) upgrade of the Bot at Super level.
ModVariable.SetVariableForObjectAsInt("UpgradeWorkerSearchSuper", "Range", 15)

Setting the bot speed

The speed is considered the delay in movement (i.e. smaller = faster)

Example:

- ***ModVariable.SetVariableForObjectAsFloat("Worker", "BaseDelay", 0.02)***

Modifying Crop variables:

Bushes ["Bush"]:

- "GrowSeedDelay" - How long before it becomes a bush.
ModVariable.SetVariableForObjectAsFloat("Bush", "GrowSeedDelay", 5.0)
- "GrowBushDelay" - How long before it starts growing berries.
ModVariable.SetVariableForObjectAsFloat("Bush", "GrowBushDelay", 20.0)
- "GrowBerriesDelay" - How long before the berries are completely grown
ModVariable.SetVariableForObjectAsFloat("Bush", "GrowBerriesDelay", 120.0)

Fruit Trees ["TreeApple"]:

- "GrowFruitDelay" - How long before Apples are grown.
ModVariable.SetVariableForObjectAsFloat("TreeApple", "GrowFruitDelay", 120.0)

Crops ["CropCotton", "CropWheat", "CropCarrot"] :

- "GrowDelay" - Time before fully grown.
ModVariable.SetVariableForObjectAsFloat("CropWheat", "GrowDelay", 180.0)
- "MaxYield" - Maximum possible yield from the crop.
ModVariable.SetVariableForObjectAsInt("CropWheat", "MaxYield", 5)
- "TilledBoost" - Boost from the soil being tilled.
ModVariable.SetVariableForObjectAsInt("CropWheat", "TilledBoost", 1)

- "FertiliserBoost" - Boost from the soil being fertilised.
ModVariable.SetVariableForObjectAsInt("CropWheat", "FertiliserBoost", 1)
- "WaterBoost" - Boost from watering the crop.
ModVariable.SetVariableForObjectAsInt("CropWheat", "WaterBoost", 1)
- "MaxHeight" - Height setting of the crop.
ModVariable.SetVariableForObjectAsFloat("CropWheat", "MaxHeight", 1.0)

Growables ["Grass", "Weed", "Mushroom", "Bullrushes"]:

- "GrowDelay" - Time before fully grown.
ModVariable.SetVariableForObjectAsFloat("Grass", "GrowDelay", 45.0)

Modifying Mining variables:

Note: The number of swings required to obtain a resource from a tile = Base * 100/Chance:

Chance for digging CLAY on Clay Deposits (first layer):

- ***ModVariable.SetVariable("ClaySoilChance", 50)***

Chance for digging CLAY on Rich Clay Deposits (second layer):

- ***ModVariable.SetVariableFarmerActionOnTiles("Shovel", "Clay", "ToolShovelStone", 20)***
- ***ModVariable.SetVariableFarmerActionOnTiles("Shovel", "Clay", "ToolShovel", 12)***

Base number of swings required for digging tiles:

- ***ModVariable.SetVariableFarmerAction("Shovel", "Plot", "ToolShovelStone", 16)***
- ***ModVariable.SetVariableFarmerAction("Shovel", "Plot", "ToolShovel", 8)***

Chance for mining IRON on Trace Metal Ore Deposits (first layer):

- ***ModVariable.SetVariable("IronSoilChance", 20)***

Chance for mining IRON on Metal Ore Deposits (second layer):

- ***ModVariable.SetVariable("IronSoil2Chance", 50)***

Number of swings required for IRON on Rich Metal Ore Deposits (third layer):

- ***ModVariable.SetVariableFarmerActionOnTiles("Mining", "Iron", "ToolPickStone", 20)***
- ***ModVariable.SetVariableFarmerActionOnTiles("Mining", "Iron", "ToolPick", 12)***

Base number of swings required for mining tiles:

- ***ModVariable.SetVariableFarmerAction("Mining", "Plot", "ToolPickStone", 20)***
- ***ModVariable.SetVariableFarmerAction("Mining", "Plot", "ToolPick", 12)***

Chance for mining COAL on Trace Coal Deposits (first layer):

- ***ModVariable.SetVariable("CoalSoilChance", 20)***

Chance for mining COAL on Coal Deposits (second layer):

- ***ModVariable.SetVariable("CoalSoil2Chance", 40)***

Chance for mining COAL on Rich Coal Deposits (third layer):

- ***ModVariable.SetVariable("CoalSoil3Chance", 60)***

Number of swings required for COAL on Pure Coal Deposits (fourth layer):

- ***ModVariable.SetVariableFarmerActionOnTiles("Mining", "Coal", "ToolPickStone", 20)***
- ***ModVariable.SetVariableFarmerActionOnTiles("Mining", "Coal", "ToolPick", 12)***

Base number of swings required for mining tiles:

- ***ModVariable.SetVariableFarmerAction("Mining", "Plot", "ToolPickStone", 20)***
- ***ModVariable.SetVariableFarmerAction("Mining", "Plot", "ToolPick", 12)***

Chance for mining STONE on Stone Deposits (first layer):

- ***ModVariable.SetVariable("StoneSoilChance", 50)***

Number of swings required for STONE on Rich Stone Deposits (second layer):

- ***ModVariable.SetVariableFarmerActionOnTiles("Mining", "Stone", "ToolPickStone", 20)***
- ***ModVariable.SetVariableFarmerActionOnTiles("Mining", "Stone", "ToolPick", 12)***

Base number of swings required for mining tiles:

- ***ModVariable.SetVariableFarmerAction("Mining", "Plot", "ToolPickStone", 20)***
- ***ModVariable.SetVariableFarmerAction("Mining", "Plot", "ToolPick", 12)***

Chance of digging two turfs rather than one:

- ***ModVariable.SetVariable("TurfChance", 30)***

Modifying Colonists variables:

Note: The colonists are referred to as 'Folk' in code.

Colonists ["Folk"]:

- "MaxEnergy" - Largest amount of time a colonist can be fed.
ModVariable.SetVariableForObjectAsFloat("Folk", "MaxEnergy", 180.0)
- "HappinessDelay" - Time between each heart generated when happy
ModVariable.SetVariableForObjectAsFloat("Folk", "HappinessDelay", 15.0)
- "Weight0"- "Weight7" - Weight at different tiers
ModVariable.SetVariableForObjectAsInt("Folk", "Weight0", 1)

Colonists' Hearts ["FolkHeart" - "FolkHeart8"]:

- "Value" - The value/worth of each heart type.
ModVariable.SetVariableForObjectAsInt("FolkHeart", "Value", 1)
ModVariable.SetVariableForObjectAsInt("FolkHeart5", "Value", 10000)

- "Usage" - How much 'Usage'/Damage each heart type does to tops/housing/toys when Colonists make a heart.

ModVariable.SetVariableForObjectAsInt("FolkHeart", "Usage", 1)

ModVariable.SetVariableForObjectAsInt("FolkHeart5", "Usage", 10000)

Modifying Fuel variables:

Setting the 'Burnable Fuel Required' (To make recipe) Example:

- ***ModVariable.SetVariableForObjectAsFloat("Porridge", "FuelRequired", 20.0)***

Works for:

"MushroomSoup", "MushroomStew", "MushroomPie", "BerriesStew", "BerriesJam", "BerriesPie", "Porridge", "BreadCrude", "Bread", "FruitLoaf", "SeededLoaf", "MilkPorridge", "FruitPorridge", "HoneyPorridge", "BoiledEgg", "PumpkinSoup", "PumpkinStew", "PumpkinPie", "AppleStew", "AppleJam", "ApplePie", "FishSoup", "FishStew", "FishPie", "Charcoal", "IronCrude", "Iron", "PotClay", "LargeBowlClay", "FlowerPot", "BricksCrude", "Gnome", "Seedling".

Modifying Food variables:

Setting the 'Food Energy' (The amount of energy received by the colonists) Example:

- ***ModVariable.SetVariableForObjectAsFloat("MushroomDug", "Energy", 20.0)***

Works for:

"MushroomDug", "MushroomHerb", "MushroomSoup", "MushroomStew", "MushroomPie", "Berries", "BerriesSpice", "BerriesStew", "BerriesJam", "BerriesPie", "MilkPorridge", "FruitPorridge", "HoneyPorridge", "Porridge", "BreadCrude", "Bread", "Bread", "BreadButtered", "FruitLoaf", "SeededLoaf", "PumpkinRaw", "PumpkinSeeds", "PumpkinHerb", "PumpkinSoup", "PumpkinStew", "PumpkinPie", "FishRaw", "FishHerb", "FishSoup", "FishStew", "FishPie", "Apple", "AppleSpice", "AppleStew", "AppleJam", "ApplePie", "Carrot", "CarrotSalad", "CarrotStirFry", "CarrotHoney", "CarrotCurry", "Water", "Milk", "Egg", "BoiledEgg", "Butter", "Honey".

Modifying Unlocked/Locked variables:

Any object can be unlocked at the start of play by using this example:

- ***ModVariable.SetVariableForObjectAsInt("Plank", "Unlocked", 1)***

Where '1' must be passed in.

NOTE: Many items are already unlocked (obviously). Also, items below may be unlocked but are only available at certain unlock stages of the game.

These Include:

"Plank", "Pole", "Stick", "Rock", "Log", "Clay", "IronOre", "Coal", "BeesNest", "Apple", "TreeSeed", "Pumpkin", "PumpkinSeeds", "PumpkinRaw", "Berries", "GrassCut", "AnimalBee", "AnimalBird", "AnimalChicken", "AnimalCow", "AnimalSheep", "FishSalmon", "Egg", "MushroomDug", "Water", "Milk", "SeaWater", "Sand", "Soil", "Manure", "WheatSeed", "Wheat", "Straw", "Wool", "WeedDug", "Turf", "FlowerSeeds01", "FlowerSeeds02", "FlowerSeeds03", "FlowerSeeds04", "FlowerSeeds05", "FlowerSeeds06", "FlowerSeeds07", "Folk", "FolkHeart", "FolkHeart2", "FolkHeart3", "FolkHeart4", "FolkHeart5", "FolkHeart6", "FolkHeart7", "FolkHeart8", "FishAny", "HatAny", "TopAny", "WorkerFrameAny", "WorkerHeadAny", "WorkerDriveAny", "Fuel", "StoneBlockCrude", "Fleece", "FishRaw", "CottonSeeds", "CottonBall", "CottonLint", "BullrushesSeeds", "BullrushesStems", "BullrushesFibre", "Weed", "Grass", "CropWheat", "CropCotton", "TreePine", "TreeApple", "TreeStump", "CropPumpkin", "Bush", "Hedge", "Mushroom", "FlowerWild", "Bullrushes", "Honey".

Modifying Weights of Objects variables:

Every object has a 'weight', much like real life.

You can set the weight of an object using this example:

- ***ModVariable.SetVariableForObjectAsInt("Charcoal", "Weight", 2)***

Works with every object type.

Modifying Multiple Carry Objects variables:

The player and the bots can carry multiple of any object unless specified that they cannot.

This can be altered by using:

- ***ModVariable.SetVariableForObjectAsInt("AnimalSheep", "NoMultiple", 1)***

Works with every object type.

NOTE: Some objects are forced to one item regardless of the above. These are the Bots and any Tools.

Modifying the Day/Night Time variables:

If you want to change the amount of time between 'stages' of day/night then here is an example:

- Sunrise (Time in mins)
ModVariable.SetVariable("SunRiseDelay", 3)
- Day (Time in mins)
ModVariable.SetVariable("DayTimeDelay", 12)

- Sunset (Time in mins)
ModVariable.SetVariable("SunSetDelay", 3)
- Night (Time in mins)
ModVariable.SetVariable("NightTimeDelay", 3)

Modifying Vehicle variables:

Any vehicle has a 'weight capacity' (much like real life) which can be set in code, here is an example:

- ***ModVariable.SetVariableForObjectAsInt("WheelBarrow", "WeightCapacity", 20)***

Works with: [Any Storage Vehicles]

"WheelBarrow", "Cart", "CartLiquid", "Carriage", "CarriageLiquid", "CarriageTrain".

Modifying Fertiliser variables:

All food can be used as fertiliser, set the value like this:

- ***ModVariable.SetVariable("FertiliserValueFood", 0.25)***

All other types of Fertiliser object can be specifically set, for example:

- ***ModVariable.SetVariableForObjectAsInt("Fertiliser", "FertiliserValue", 1)***

Works with:

"Fertiliser", "Manure", "Straw", "TreeSeed", "WheatSeed", "CarrotSeed", "CottonSeeds", "BullrushesSeeds", "WeedDug", "Pumpkin", "GrassCut", "Turf".

Modifying Fueller variables:

Any type of burnable fuel object can have its value set (for how much it provides), for example:

- ***ModVariable.SetVariableForObjectAsInt("Stick", "Fuel", 10)***

Works with:

"Stick", "Pole", "Plank", "Log", "Charcoal", "Coal", "HayBale", "Fertiliser".

Modifying Misc. variables:

- Number of times a 'Grazer' can eat a hay bale.
ModVariable.SetVariableForObjectAsInt("HayBale", "MaxEats", 10)
- Max number of times a Boulder can be mined for stone.
ModVariable.SetVariableForObjectAsInt("Boulder", "MaxMined", 20)
- Cost (in stones) for a Rough Stone Block.
ModVariable.SetVariable("CrudeStoneBlockCost", 10)
- Max number of times a Tall Boulder (size 1) can be mined is $\text{MaxMined1} \div \text{CrudeStoneBlockCost}$.
ModVariable.SetVariableForObjectAsInt("TallBoulder", "MaxMined1", 20)
- Max number of times a Tall Boulder (size 2) can be mined is $\text{MaxMined2} \div \text{CrudeStoneBlockCost}$.
ModVariable.SetVariableForObjectAsInt("TallBoulder", "MaxMined2", 30)
- Max number of times a Tall Boulder (size 3) can be mined is $\text{MaxMined3} \div \text{CrudeStoneBlockCost}$.
ModVariable.SetVariableForObjectAsInt("TallBoulder", "MaxMined3", 40)
- Max amount of times a Clay Pot can be used.
ModVariable.SetVariableForObjectAsInt("PotClay", "MaxUsage", 5)
- Max amount of times a Large Clay Pot can be used.
ModVariable.SetVariableForObjectAsInt("LargeBowlClay", "MaxUsage", 5)
- The Range of a Sign's area.
ModVariable.SetVariableForObjectAsInt("Sign", "Range", 16)
- The Range of a Direction Sign's area.
ModVariable.SetVariableForObjectAsInt("Sign2", "Range", 16)
- The Range of a Billboard's area.
ModVariable.SetVariableForObjectAsInt("Billboard", "Range", 20)

Modifying a Building's variables:

Buildings can vary from static decorative 'buildings' to storage facilities to 'converters' that take ingredients and produce a product.

The time taken for a 'converter' building to convert something (in seconds) can be set. Example of it being set to 1 second (from the original time of 5 seconds):

- ***ModVariable.SetVariableForObjectAsFloat("ChoppingBlock", "ConversionDelay", 1.0)***

The time taken for a building blueprint to be constructed (in seconds) can also be set.

Example of it being set to 1 second (from the original time of 3 seconds):

- ***ModVariable.SetVariableForObjectAsFloat("StorageFertiliser", "BuildDelay", 1.0)***

Some buildings feature walls or floors - to enable this use:

- ***ModVariable.SetVariableForObjectAsInt("KitchenTable", "Walls", 1)***
- ***ModVariable.SetVariableForObjectAsInt("KitchenTable", "Floors", 1)***

A few specific buildings use 'energy' to create the product (in seconds)(Like the windmill or waterwheel).

Example of changing the energy required to 2 seconds (from original 15):

- ***ModVariable.SetVariableForObjectAsFloat("Windmill", "RechargeTime", 2.0)***

Houses for colonists degrade over use, this can be modified by:

- ***ModVariable.SetVariableForObjectAsInt("Hut", "MaxUsage", 100)***

Can be used with "Hut", "LogCabin", "StoneCottage", "BrickHut", "Mansion".

Modifying Building Repair:

Set the Type required to repair a building

- ***ModVariable.SetVariableForObjectAsIntFromString("Hut", "RepairObject", "Log")***
- ***ModVariable.SetVariableForObjectAsInt("Hut", "RepairAmount", 2)***

Modifying Storage allowances:

Any storage box or palette has a limited amount specified, it's easy to change this. For example, changing the amount a storage can hold for sticks (from 25 max to 200 max):

- ***ModVariable.SetVariableForStorageAmount("Stick", 200)***

Note: The bonus of stacking two (2.4x) or three (4x) still applies.

Can be used with any object type already storable.

Getting Variables of the game

Sometimes it's useful to see the original values of certain game variables.

Here's examples of getting a float, int or string:

- ***ModVariable.GetVariableForObjectAsFloat("UpgradePlayerMovementCrude", "Delay")***

- ***ModVariable.GetVariableForObjectAsString("WorkerDriveMk3", "MoveSoundName")***
- ***ModVariable.GetVariableForObjectAsInt("UpgradePlayerInventoryCrude", "Capacity")***

Ingredients and Converters

Everything produced in Autonauts will consist of ingredients.

For example, to make a 'Basic Bot' it requires 4 types of ingredients: 1xLog, 3xPlanks, 1xPole and 1xTreeSeed.

This is classed as a recipe. Whether you're making a bot, a blanket or mushroom soup, it's a recipe.

An example of changing a recipe for 1x 'Basic Bot' for any converter with ingredients: 4xBerries, 2xBridges and 1xAnimalCow is done like this:

- ***ModVariable.SetIngredientsForRecipe("BasicWorker", {"Berries", "Bridge", "AnimalCow"}, {4,2,1}, 1)***

An example of changing a recipe for 1x 'Basic Bot' changing the recipe ONLY for the 'Bot Assembly Unit' converter with ingredients: 4xBerries, 2xBridges and 1xAnimalCow is done like this:

- ***ModVariable.SetIngredientsForRecipeSpecific("WorkerAssembler", "BasicWorker", {"Berries", "Bridge", "AnimalCow"}, {4,2,1}, 1)***

NOTE: The final '1' is the amount of BasicWorkers produced.

The Ingredients for any recipe can be found in a similar fashion. For example, to find what ingredients are in the recipe for 'Berries':

- ***AllIngredients = ModVariable.GetIngredientsForRecipe("Berries")***

The above returns a table of objects results.

To find the quantities (in order of ingredients):

- ***AllIngredientAmounts = ModVariable.GetIngredientsAmountForRecipe("Berries")***

Creating a Converter

Function:

- ***ModConverter.CreateConverter(string Name, string array RecipeIngredients, number array RecipeIngredientsAmount, string ModelName, int array TopLeft, int array BottomRight, int array AccessPoint, int array SpawnPoint, bool UsingCustomModel)***

Creating a custom converter is fairly simple and can be thought of as two steps or two lines of code.

First you need to think about what recipe is required to construct the converter.

A worker assembler requires 2xLog and 3xPlank to build from a blueprint.

For this example we will focus on making a new converter called 'BerryMaker'.

To construct the 'BerryMaker' it will require 1xPole and it will ultimately produce berries.

Example:

- ***ModConverter.CreateConverter("BerryMaker", {"Berries"}, {"Pole"}, {1})***

To explain the above: We are creating a new converter called 'BerryMaker' that will create the game object Berries (the ones that normally come from bushes), to construct this from a blueprint it takes 1xPole.

At this point when you run the game it will now show a new custom converter in the blueprint menu. If you were to place it and construct it with the 1xPole required it will do nothing. Why? Simple, because the "Berries" recipe doesn't exist, if it was e.g. "BasicWorker" then the converter would let you produce a basic worker/bot.

So how do we set the berry recipe for all converters using it? Using the earlier concept of 'SetIngredientsForRecipe'. Let's create the recipe using what we learned from earlier:

Example:

- ***ModVariable.SetIngredientsForRecipe("Berries", {"Plank"}, {3}, 10)***

The above "BerryMaker" converter now uses the "Berries" recipe which requires 3xPlanks to produce 10xBerry.

You may notice earlier that the converter has "Berries" in an array (like this {"Berries"}), this allows you to add multiple recipes (which can be set in game by having the player click on the constructed converter). For example, if the converter featured recipes for creation of "Berries" and "BasicWorker" you could set it like so:

Example:

- ***ModConverter.CreateConverter("BerryMaker", {"Berries", "BasicWorker"}, {"Pole"}, {1})***

Advanced Converter Creation:

Above the creation of a custom converter, a model and dimensions can be set.

The model must reside within the 'models' folder in your mod.

Let's say the model is stored in 'Mods/Your Mod/models/fastfood' and called 'myfoodbuildmodel', this is how to use it:

- ***ModConverter.CreateConverter("New1", {"Berries"}, {"Pole"}, {1}, "fastfood/myfoodbuildmodel")***

The last parameter is the model desired. You must state the full path from anything after the 'models' folder.

All models must be in '.obj' format.

Once you have a custom model set, you can then improve the dimensions (and where the in/out pads are placed).

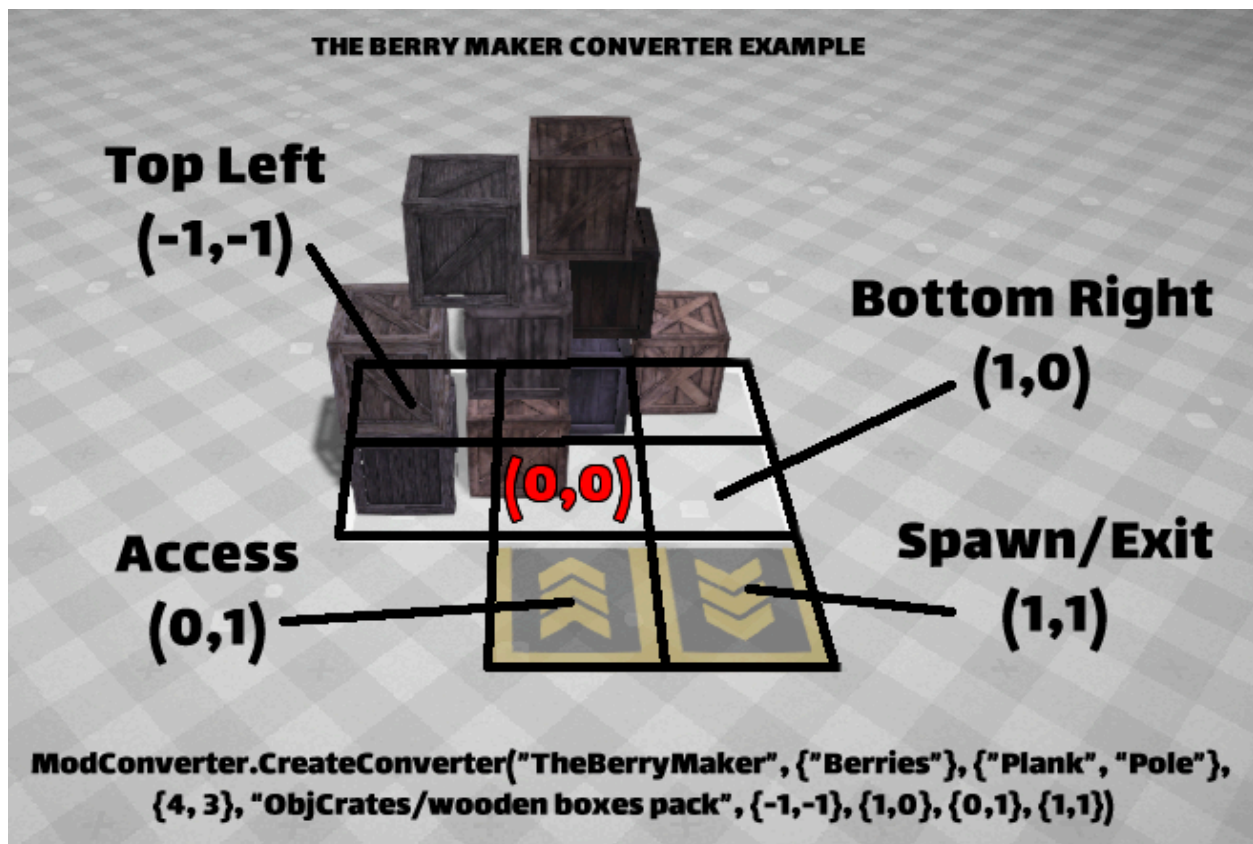
The top left and the bottom right of the tile space identify the 'space' where the converter is placed. The 'in' and 'out' must be kept outside of this.

NOTE: Parameters (for dimensions) are Top Left, Bottom Right, Access Point, Spawn Point.

Example of changing dimensions with a custom model:

- `ModConverter.CreateConverter("New1", {"Berries"}, {"Pole"}, {1}, "myfoodbuildmodel", {-1,-1}, {1,0}, {0,1}, {1,1})`

Here is an example from the 'BerryGoodMod' which should help with understanding the dimension parameters.



Converter Examples:

Custom Converter, using default game model:

- *ModConverter.CreateConverter("BerryMaker", {"Berries"}, {"Pole"}, {1})*

Custom Converter, using specified game model, specifying dimensions and producing a custom created ingredient ('Colonist Cupcake'): [v134.29]

- *ModConverter.CreateConverter("Colonist Cupcakes Maker2", {"Colonist Cupcake"}, {"Folk"}, {4}, "Models/Buildings/Converters/Workbench", {-1,-1}, {1,0}, {0,1}, {1,1}, false)*

Custom Converter, using specified user model, specifying dimensions and requiring a custom created ingredient ('Crate'): [v134.29]

- *ModConverter.CreateConverter("Mega Maker", {"Colonist Cupcake", "Berries"}, {"Crate"}, {4}, "ObjCrates/wooden boxes pack", {-1,-1}, {1,0}, {0,1}, {1,1})*

Creating a Building:

Function:

- *ModBuilding.CreateBuilding(string Name, string array RecipeIngredients, number array RecipeIngredientsAmount, string ModelName, int array TopLeft, int array BottomRight, int array AccessPoint, bool UsingCustomModel)*

Buildings can be created in a similar fashion to converters. However, they must always contain a model reference as a building can be thought of as a static built item.

If you wanted to create a building called 'CrateStuff', constructed using '2 poles' and featured the model 'MyCrate' (stored in the 'models' folder), it would be done like so:

- *ModBuilding.CreateBuilding("CrateStuff", {"Pole"}, {2}, "MyCrate")*

Above this, you can specify the dimensions of the building to create the correct outline size.

Example:

- *ModBuilding.CreateBuilding("Crate2", {"Pole"}, {2}, "MyCrate", {-1,-1}, {1,0})*

Where (-1,-1) are the top left (x,y) coordinates and (1,0) are the bottom right coordinates.

Creating a Decorative Item:

Function:

- *ModDecorative.CreateDecorative(string Name, string array RecipeIngredients, number array RecipeIngredientsAmount, string ModelName, bool UsingCustomModel)*

If you want a holdable spawned item, or rather a decorative item. Then ModDecorative is the answer. These items can be spawned and moved about.

In a very similar syntax to the ModBuildings, you can create a decorative item called 'Crate', constructed from '2 planks' and uses the model 'Crates/Crate1' like so:

- *ModDecorative.CreateDecorative("Crate", {"Plank"}, {2}, "Crates/Crate1")*

Creating a Holdable Item:

Function:

- *ModHoldable.CreateHoldable(string Name, string array RecipeIngredients, number array RecipeIngredientsAmount, string ModelName, bool UsingCustomModel)*

Want to create a new in-game item? One that can be picked up, spawned and used by (and produced from) a converter? Then this is for you.

Example:

- *ModHoldable.CreateHoldable("Crate", {"Stick"}, {1}, "ObjCrates/wooden boxes pack")*

The above makes a new object called 'Crate'. This object model is located at "ObjCrates/wooden boxes pack" under the models folder in your mod.

The recipe to make the 'Crate' is 1xStick. This is used when a converter produces this item.

NOTE: This must be called in the function Creation but it must be specified before any of your custom converters/building etc. that use it.

Example of a converter producing the item:

- *ModConverter.CreateConverter("TheCrater", {"Crate"}, {"Plank"}, {4}, "ObjCrates/wooden boxes pack", {-1,-1}, {1,0}, {0,1}, {1,1})*

Example of a converter using it as a construction ingredient:

- *ModConverter.CreateConverter("BerryMaker", {"Berries"}, {"Plank", "Crate"}, {4, 3}, "ObjCrates/wooden boxes pack", {-1,-1}, {1,0}, {0,1}, {1,1})*

Example of changing a recipe for the item: (to 8x sticks required, making 1 'Crate')

- *ModVariable.SetIngredientsForRecipe("Crate", {"Stick"}, {8}, 1)*

Example of changing another recipe to use the item as an ingredient: (4x 'Crates' makes 10x Berries)

- *ModVariable.SetIngredientsForRecipe("Berries", {"Crate"}, {4}, 10)*

Holdable Examples:

Custom Holdable using default model:

- *ModHoldable.CreateHoldable("Crate1")*

Custom Holdable using default model and specified recipe ingredients to make it:

- *ModHoldable.CreateHoldable("Crate2", {"Stick"}, {1})*

Custom Holdable using specified in-game model (and recipe ingredients): [v134.29]

- *ModHoldable.CreateHoldable("Crate3", {"Stick"}, {1}, "Models/Food/AppleJam", false)*

Custom Holdable using specified user model (and recipe ingredients): [v134.29]

- *ModHoldable.CreateHoldable("Crate4", {"Stick"}, {1}, "ObjCrates/wooden boxes pack", true)*

Creating a Food Item:

Function:

- *ModFood.CreateFood(string Name, string array RecipeIngredients, number array RecipeIngredientsAmount, string ModelName, bool UsingCustomModel)*

Much like a holdable item (see above), you can create a food item in a similar fashion. A food item can be picked up like a holdable, used as an ingredient but is also usable as food for colonists.

Food Creation Examples:

Custom Holdable using default model:

- ***ModFood.CreateFood("Colonist Cupcake")***

Custom Holdable using default model and specified recipe ingredients to make it:

- ***ModFood.CreateFood("Colonist Cupcake", {"Folk"}, {1})***

Custom Holdable using specified in-game model (and recipe ingredients): [v134.29]

- ***ModFood.CreateFood("Colonist Cupcake", {"Folk"}, {1}, "Models/Food/AppleJam", false)***

Custom Holdable using specified user model (and recipe ingredients): [v134.29]

- ***ModFood.CreateFood("Colonist Cupcake", {"Folk"}, {1}, "Cupcake/Cupcake_v1", true)***

Spawning any Object:

If you wish to spawn any object at a certain place in the map, it's as simple as:

- ***ModBase.SpawnItem("Berries", 0, 0)***

The above will spawn some berries at map tile (0,0).

NOTE - you must place the call in the 'AfterLoad' function or later (i.e. OnUpdate()).

Can be used with any object type item.

Any Mod Decorative item can be spawned using this system also. For example, if you created a decorative item like this:

- ***ModDecorative.CreateDecorative("CrateDeco", {"Pole"}, {1}, "myModel")***

You would spawn it like this:

- ***ModBase.SpawnItem("CrateDeco", 10, 10)***

As default the mod will spawn the item/s on each load, if you wish to make it spawn once and not on each load to the game, change it to this:

- ***ModBase.SpawnItem("Berries", 0, 0, true)***

NOTE: Setting the last flag to 'true' CANNOT BE USED inside loops.

Spawning special items like buildings, roads and bridges etc. use the same function and are fully supported now.

Spawning as Blueprint or Fully Built:

Buildings can now be set to spawn in 'blueprint' or 'fully built' state.
Two new parameters have been added, making the function call like so:

- *ModBase.SpawnItem(Item, x, y, DoOnce, Instant, ForceBlueprint)*

Where:

DoOnce - This forces spawning to happen once only, regardless of a new load. Do not use inside large loops! Performance will be very very slow.

Instant - If enabled this fully builds the building. If disabled, the status of built depends on if it was built previously.

ForceBlueprint - If enabled forces the blueprint version of building regardless of Instant or previously built status.

NOTE: All three parameters default to false. You don't need to define them in a call.

This includes:

FlooringCrude, Workshop, FlooringBrick, FlooringFlagstone, FlooringParquet, SandPath, StonePath, RoadCrude, RoadGood, Bridge, BridgeStone, FencePost, Gate, FencePicket, GatePicket, StoneWall, BrickWall, LogWall, BlockWall, BlockDoor, StoneArch, StoneArchDoor, LogArch, Door, Window, WindowStone, TrainTrack, TrainTrackCurve, TrainTrackPointsLeft, TrainTrackPointsRight, TrainTrackBuffer, TrainTrackBridge.

Clearing & Destruction:

If you ever want to clear or 'reset' a tile or even large area, then this section is for you.
You must call these in the 'AfterLoad' function in LUA.

To clear a single tile back to grass/soil:

- *ModTiles.ClearEverythingOnSingleTile(20, 24)*

Where (20,24) is the (x,y) coordinate.

A large section can be erased by:

- *ModTiles.ClearEverythingInArea(0,0, 10, 10)*

Where (0,0) is the start and (10,10) is the end of the area to be wiped.

Clearing an entire map can be done like so:

- *ModTiles.ClearEverythingInArea(0,0, ModTiles.GetTilesWide() - 1, ModTiles.GetTilesHigh() - 1)*

Sometimes you want to be specific about what is to be cleared. In this you can choose from whether to clear 'Buildings', 'Static Objects', 'Holdable Objects' and 'Tiles'.

Example:

- ***ModTiles.ClearSpecificsInArea(20,20, 40, 40, true, true, true, true)***

The above clears everything between (20,20) to (40,40). The last four parameters are 'Buildings', 'Static Objects', 'Holdable Objects' and 'Tiles' (in that order).

Setting Buildings to true removes all buildings, walls, roads, bridges etc.

Setting Static Objects to true removes trees, bushes (anything that is a static object).

Setting Holdable Objects to true removes everything holdable (most objects like berries etc.)

Setting Tiles to true will force each tile back to soil ("Soil") or grass ("Empty").

NOTE: The Player (Farmer), the tutorial bot (Quest stack), all robots (Workers) and all colonists (Folk) will not be removed regardless.

Disabling Safety Features:

If you want to disable the safety logic which makes sure to provide a minimum number of items on the map e.g. trees, sheep, cows, chickens, flowers etc.

You can use this:

- ***ModBase.DisableSafety(true)***

(Where true/false is the expected parameter)(true to disable, false to enable safety)

[Call in the BeforeLoad() function only]

Various/Misc. Gets

Is it Night Time?

- ***ModBase.GetIsNightTime()***

Get Game Version

- ***ModBase.GetGameVersion()***

Get Current Game State

- ***ModBase.GetGameState()***

Input Callbacks:

The Mod system now supports two styles of input callback; the first is simply a registration callback that calls every time a key is pressed, the second is an exposed defined key that allows the user to redefine the key as desired.

Let's take a look by example:

Register for all Input:

Function:

- *ModBase.RegisterForInputPress(Function CallbackFunction)*

Example:

```
function Expose()
    ModBase.RegisterForInputPress(InputCallback)
end
function InputCallback( param )
    ModDebug.Log("InputCallback ",param)
end
```

Each key press will return to the callback as 'string param'. Meaning if 'W' is pressed param will be 'W'.

Exposed Mod Keys:

Function:

- *ModBase.ExposeKeybinding(string Name, number KeyToUse, function Callback)*

Example:


```
function Expose()  
    ModBase.ExposeKeybinding("Explosion", 1, ExposedKeyCallback)  
end  
function ExposedKeyCallback( param )  
    ModDebug.Log("ExposedKeyCallback ",param)  
end
```

The above sets 'mod key 1' to the callback 'ExposedKeyCallback'.

In Mod Options, the player can select what 'mod key 1' is defined as (any key they choose).

When the key assigned to 'mod key 1' is pressed the callback will return with string param set to the name of the key, which in this case is 'Explosion'.

If the user selects the 'mod key 1' as '1' then when they press '1', 'Explosion' will be triggered.

NOTE: It will trigger in all states including the menu - check the game state first.

NOTE: There are a possible 10 (1-10) keys for mods, they can clash with other mods if all enabled.

Tile Management:

Getting Map Limits:

Finding the size of the map in tiles is simple:

- *local Width = ModTiles.GetTilesWide()*
- *local Height = ModTiles.GetTilesHigh()*

These can be used in all functions (BeforeLoad(), AfterLoad(), OnUpdate()).

Changing Tile Type:

All tiles can be changed in Lua using:

- *ModTiles.SetTile(0,0,"Soil")*

This takes the tile at coordinates (0,0) and changes it to type Soil.

NOTE - you must place the call in the 'AfterLoad' function (or afterwards).

Tile Types [Like Soil]:

Empty, Soil, SoilTilled, SoilHole, SoilUsed, SoilDung, WaterShallow, WaterDeep, SeaWaterShallow, SeaWaterDeep, Sand, Dredged, Swamp, IronHidden, IronSoil, IronSoil2, Iron, IronUsed, ClayHidden, ClaySoil, Clay, ClayUsed, CoalHidden, CoalSoil, CoalSoil2, CoalSoil3, Coal, CoalUsed, StoneHidden, StoneSoil, Stone, StoneUsed, Raised, Building.

Getting Tile Type:

If you ever need to know what tile type is at a certain tile, this can be achieved by:

- *local Type = ModTiles.GetTileType(0,0)*

This returns the string e.g. 'Soil' of what type of tile is located at (0,0).

NOTE - you must place the call in the 'AfterLoad' function (or afterwards).

Getting Object info on Tile:

A tile can contain many objects (e.g. 4 berries and 3 berry jams), to query any tile and return an array of the types can be done like so:

```
ObjectsOnTile = ModTiles.GetObjectTypeOnTile(52,52)
for key,value in ipairs(ObjectsOnTile)
do
  print(key, value) --Where key would be the iter and value is the object string (from
array)
end
```

The above example shows how to get the objects on a tile and how to loop through them (if desired). 'ObjectsOnTiles' is a returned string array of the types of objects. The command returns all object types at coordinates (52,52).

Getting Objects of Type in an Area:

If you are searching for a specific type of object within an area then this can be achieved in a similar fashion to the above call e.g.:

- **ObjectsOnTile = ModTiles.GetAmountObjectsOfTypeInArea("BerriesJam", 0,0, ModTiles.GetTilesWide() - 1, ModTiles.GetTilesHigh() - 1)**

In this example we search for type 'BerriesJam' in the entire map. It returns an int of how many exist.

Getting the UID of Objects of Type in an Area:

When searching and obtaining certain objects, you may want to grab the unique identifier (UID) to manipulate the object later.

- **ModTiles.GetObjectUIDsOfType("BerriesJam", 0,0, 20,20)**

The above line will return a list of UIDs (ints) of all the 'BerriesJam' in the area (0,0) to (20,20). These can be used to modify the objects in game.

Is a Building on a Tile:

If you need to enquire if a specified tile contains any type of building, you can use:

- *ModTiles.IsBuildingOnTile(42,48)*

It returns true/false based on the tile containing any sort of building.

The Player/Farmer:

Moving The Player:

You can move the farmer to a desired location (ideally this would be used in Update).

An example of moving the player to (10, 10):

- *ModPlayer.MoveTo(10,10)*

Is Busy?

You can check if the Player is currently moving or performing any other action (returns true/false):

- *ModPlayer.IsBusy()*

This is the same as checking player state isn't None:

- *if (ModPlayer.GetState() ~= "None") then*

...

Set Start Location On Map:

If you need to set the exact location where the player should start then this is for you:

- *ModPlayer.SetPlayerStartLocation(20, 20)*

Note: the function must be called in AfterLoad_CreatedWorld() function only.

Get Player Location:

The location (in tile coordinates) can be returned via use of a table, with the first entry being X and second entry being Y.

Example:

```
Pos = ModPlayer.GetLocation()
if(Pos[1] > -1)
```

```

then
    ModDebug.Log(Pos[1], Pos[2])
end

```

Move Object To

Objects like the Player(Farmer) can 'MoveTo' a tile coordinate but this is limited to 'GoTo' objects. If you want to move an object from A to B regardless of type and without pathfinding then this can be used:

First set the position and details: (NOTE: Overloaded method)

- `ModObject.StartMoveTo(number Object ID, number endX, number endY)`
- `ModObject.StartMoveTo(number Object ID, number endX, number endY, number Speed (10), number Height (0))`

Then call the Move function in the OnUpdate() function: (NOTE: Overloaded method)

- Returns `bool MoveCompleted = ModObject.UpdateMoveTo(number Object ID)`
- Returns `bool MoveCompleted = ModObject.UpdateMoveTo(number Object ID, bool Arc (false), bool Wobble (false))`

Where: Object ID is the unique identifier of an in game object

Where: Arc will force a trajectory from A to B (reaching highest point in middle distance)

Where: Wobble will cause a sine wave movement (like a Bee)

NOTE: If it's a 'GoTo' object (like Player Farmer) then use the MoveTo command instead.

Example Lua:

```

-- Hold the object ID
ObjId = -1
-- Hold if the move has completed
MoveComplete = false

```

```
function AfterLoad()
  ObjectIDs = {}
  -- Get all the 'Berries' objects on the map
  ObjectIDs = ModTiles.GetObjectUIDsOfType("Berries", 0,0, Width - 1, Height - 1)
  -- If at least 1 exists - set ObjId
  if(#ObjectIDs[1] >= 1)
    then
      ObjId = ObjectIDs[1]
      -- Setup the MoveTo call
      ModObject.StartMoveTo(ObjId, 0,0, 10, 0)
    end
  end
end
```

```
function OnUpdate()
  -- If the move isn't complete and we have an object...
  if( MoveComplete == false and ObjId ~= -1 )
    then
      -- Call the Move function and return if completed
      MoveComplete = ModObject.UpdateMoveTo(ObjId, true, true)
    end
  end
end
```

Managing Large Mod Scripts

There are a couple of options for dividing a large mod into separate lua files. The first option is good when you have lots of custom buildings/objects that don't really interact with each other via lua code. Simply break each building/object into its own separate lua file. Note, any functions and variables in one lua script can not see the functions and variables in another script.

The second option is a work in progress, there may be some conditions or quirks to this approach (current version must be 137.14.9 or higher). Create a folder with the suffix ".scope", all lua scripts found in such a folder will be assembled in alphabetical order of filename and loaded as a single lua script. This allows you to split your mod into separate files without losing access to any functions or variables between files. Current issue with this approach is that any error messages will refer to a line number based upon all the scripts combined.

To address the line number issue, the combined file should be output into the mods directory so the modder can correlate an error to a line of code (yet to be implemented). The game should ignore ".scope" directories unless we're in Dev Mode and perhaps ignored when submitting a mod to the workshop. This feature is experimental and may change or be dropped entirely, to ensure your published mods are forward compatible, please combine your scripts into one file and remove the ".scope" directory before uploading to the workshop.

Steam Workshop Issues/FAQS

"Mods menu is stuck on 'Downloading'"

This can be one of a few things.

- Steam is busy downloading in the background and it's taking a while but is working correctly. In this case wait longer.
- Steam downloaded and installed the mod but didn't inform the game. Restart Autonauts.
- Steam downloaded but didn't install the mod and is now in a bad state. Fixing this can be annoying. Firstly try restarting Steam then launch the game. If that doesn't work you will need to fix the workshop state in your Steam folder:

Close Steam.

Locate to your steam workshop folder I.e. "Steam\steamapps\workshop".

Delete the acf file e.g. "appworkshop_979120.acf"

Clear the content folder "Steam\steamapps\workshop\content\979120"
Also the downloading folder too.
Open Steam and launch Autonauts.

"Upload Mod is failing"

Two things to check here:

- You must accept the Steam T&Cs and be in good standing with Steam. The T&Cs link is in the upload mod box.
- The icon (image) for you mod is larger than 1MB. Reduce the size to fix.