Let's Make a Mod (From Scratch!)

Contents

INTRO

Since this doesn't really exist I thought it would be a good intro point for people looking to make their first Rimworld mod. We're not going to be writing any code for this tutorial, instead we'll be working with a data format called **XML** (More on that later), which will tell the game how to interpret what we're trying to create.

One quick note before we begin, the tutorial is designed to be read **sequentially**. Read it **sequentially**. Unless you already know what you're doing, in which case READ IT **SEQUENTIALLY**... But skip past the sections when you see a **red line** of text that tells you you can skip past it.

The doc is also **really long**, but I promise you, *there are a lot of pictures*.

So without further ado…

WHAT ARE WE TRYING TO CREATE?

For this guide we're going to create a new drug in the game, called "Euphorium." It will be made at a **Drug Lab** from **Ambrosia, Insect Jelly and Neutroamine**; and give an overall buff to a pawn’s stats when taken, at the moderate risk of Ambrosia addiction.

Later on, we'll add the **research node** for Euphorium so that we can unlock it in the game. I'll also provide a link to the finished mod on the steam workshop so that you can see it in action.

Let's cook, Rimmy!

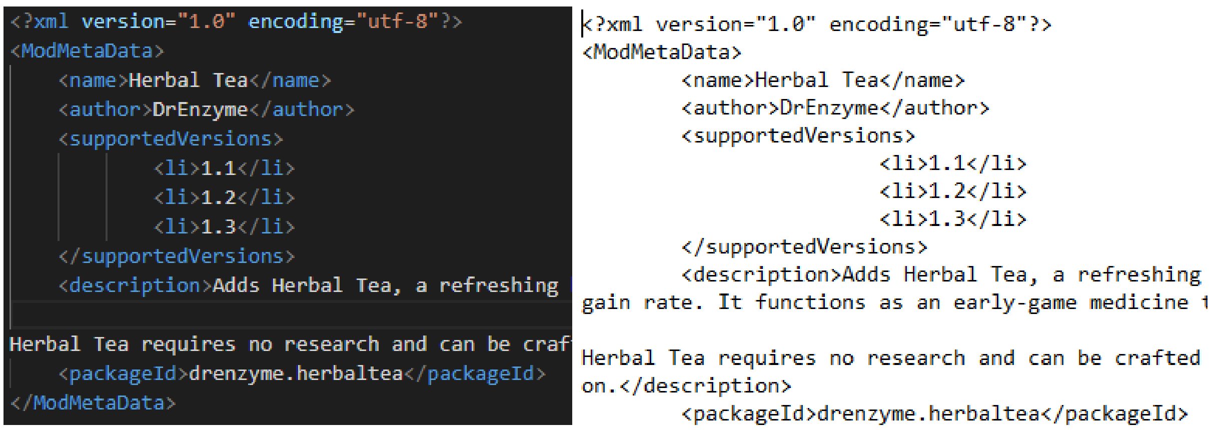
WHAT YOU'LL NEED

Two things:

- Rimworld. No expansions needed, just have the base game installed. I’ll be working on version 1.3, but this tutorial can apply to any version from 1.1 onward.
- A **GOOD** text editor. Notepad doesn’t cut it on The Rim. I'll be using **Visual Studio Code** (don't panic, we're not writing code), and I'll be providing a little guide on that right about now.

INSTALLING VSCODE

Skip this if you've already got a text editor you enjoy using, and it can make XML look pretty:



```
<?xml version="1.0" encoding="utf-8"?>
<ModMetaData>
    <name>Herbal Tea</name>
    <author>DrEnzyme</author>
    <supportedVersions>
        <li>1.1</li>
        <li>1.2</li>
        <li>1.3</li>
    </supportedVersions>
    <description>Adds Herbal Tea, a refreshing

Herbal Tea requires no research and can be craf
    <packageId>drenzyme.herbaltea</packageId>
</ModMetaData>
```

```
<?xml version="1.0" encoding="utf-8"?>
<ModMetaData>
        <name>Herbal Tea</name>
        <author>DrEnzyme</author>
        <supportedVersions>
                        <li>1.1</li>
                        <li>1.2</li>
                        <li>1.3</li>
        </supportedVersions>
        <description>Adds Herbal Tea, a refreshing
gain rate. It functions as an early-game medicine i

Herbal Tea requires no research and can be crafted
on.</description>
        <packageId>drenzyme.herbaltea</packageId>
```

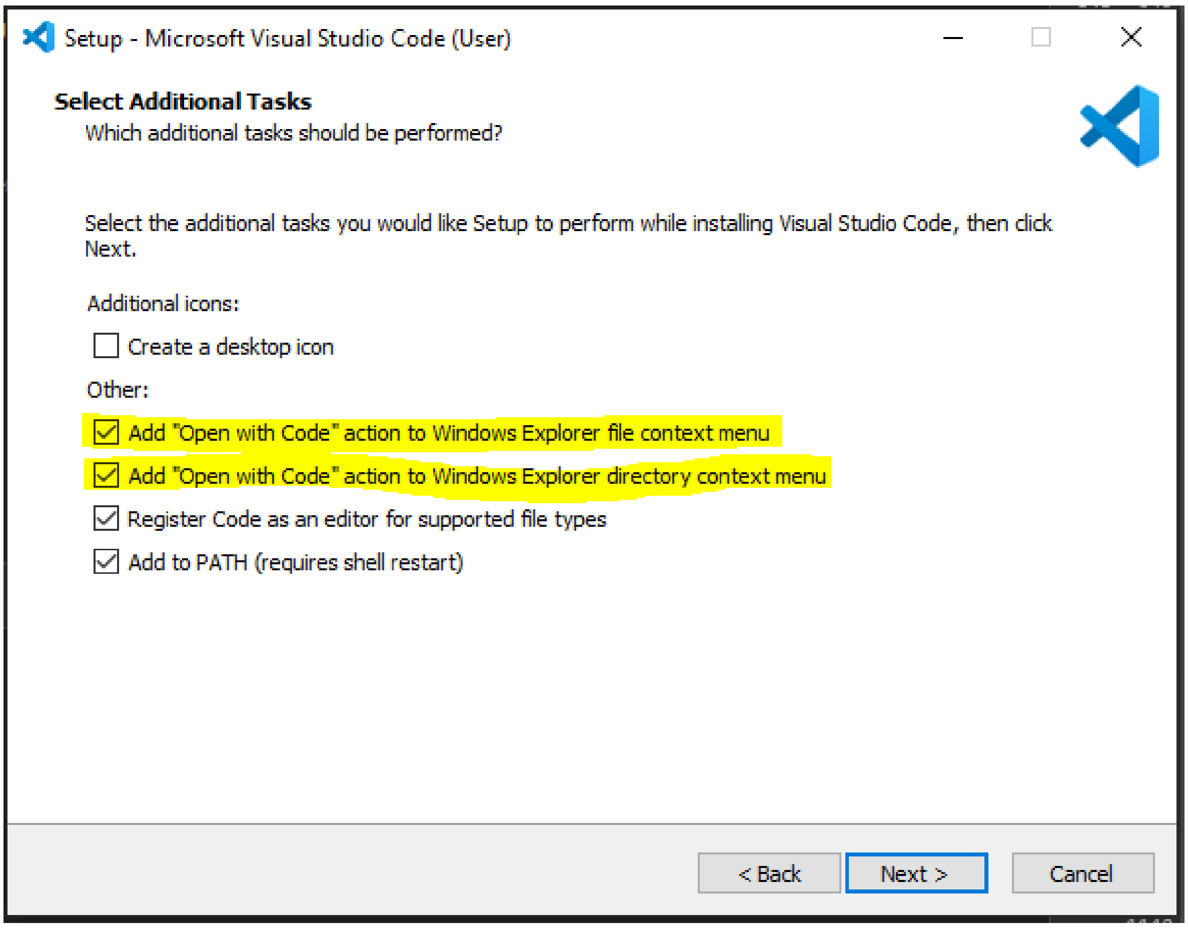
Left: Pretty XML. Right, GODAWFUL NOTEPAD XML.

Don't know what that means? Read on!

For this tutorial I'll be using VSCODE to write all the XML. VSCode is a free text editor with a bunch of fancy features that we won't be using. The only two we care about are:

- XML formatting: VSCode turns XML from a painful mess into something semi- readable (see above).
- Find in Files: Find in files lets you search a whole folder structure for your search term. Want to know where luciferium is defined in the game? Type Luciferium into the search bar and it'll find all the references to it in the game. Since we're creating our mod from scratch, we don’t *technically* need this, but as soon as you venture out into the jungle of XML that the game provides, you're going to want it. Trust me.

So go download VSCode. Install it. Love it. ~~Worship it.~~ Default settings are mostly fine, but we're going to want to tick these features here:


Setup - Microsoft Visual Studio Code (User)

Select Additional Tasks

Which additional tasks should be performed?

Select the additional tasks you would like Setup to perform while installing Visual Studio Code, then click Next.

Additional icons:

- [ ] Create a desktop icon

Other:

- [✓] Add "Open with Code" action to Windows Explorer file context menu
- [✓] Add "Open with Code" action to Windows Explorer directory context menu
- [✓] Register Code as an editor for supported file types
- [✓] Add to PATH (requires shell restart)

The second one is the important one. This will let us open VSCode from whatever folder we're currently in, **so we can open it up in the base Rimworld folder** and get access to that beautiful "Find in Files" feature I was talking about.

So go install it, or use whatever text editor you're comfortable with. Then come back and we can begin with…

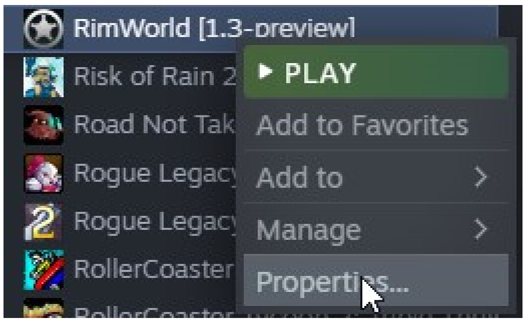
A BLANK MOD

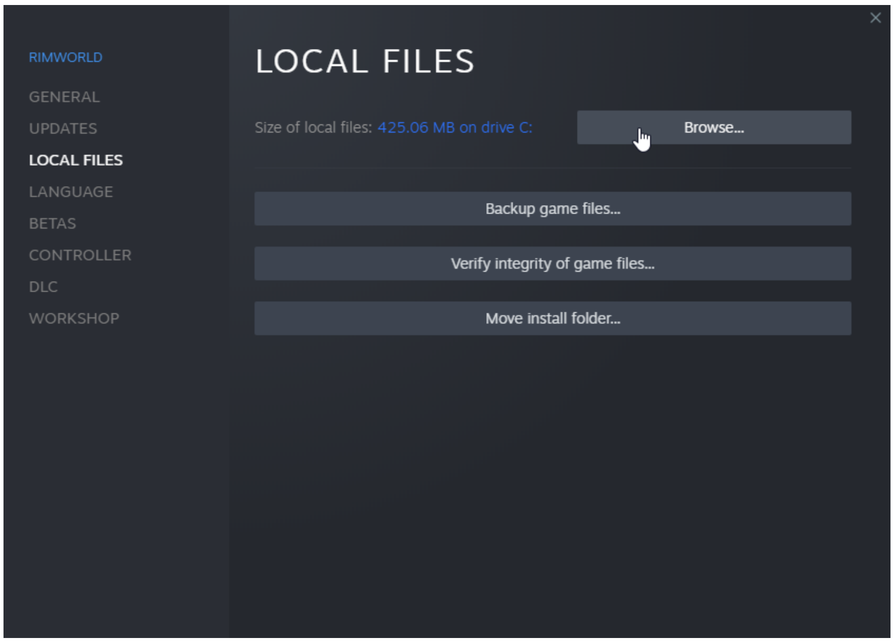
Tabula Rasa; a blank slate. None of this "Hello World" garbage just yet, there's some things I gotta *teach into you* before we actually write some XML.

First, where is Rimworld on your hard drive? It's usually wherever you installed Steam, it might even be in this folder here:

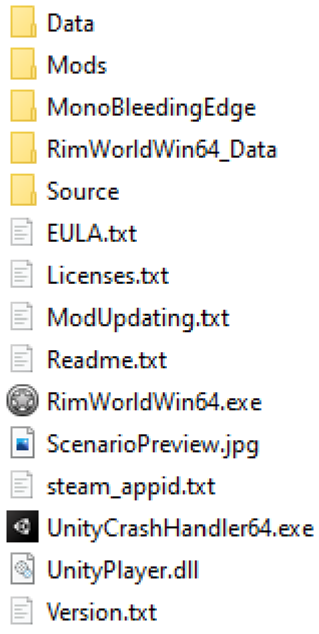
"C:\Program Files (x86)\Steam\steamapps\common\RimWorld"

If it isn't, or if you're having trouble finding it, you can always find it from within Steam itself, by right-clicking on rimworld in steam, going to "properties," going to "LOCAL FILES" then clicking "Browse":



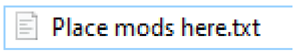


Size of local files: 425.06 MB on drive C:



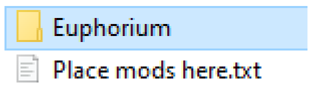
Data
Mods
MonoBleedingEdge
RimWorldWin64_Data
Source
EULA.txt
Licenses.txt
ModUpdating.txt
Readme.txt
RimWorldWin64.exe
ScenarioPreview.jpg
steam_appid.txt
UnityCrashHandler64.exe
UnityPlayer.dll
Version.txt

If you jump into that folder you'll find a folder called "Mods" and inside that there's a file that says "place mods here."



Place mods here.txt

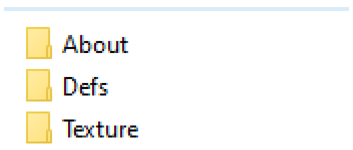
Awesome, this is where our mod has got to go. So let's create a new folder here and call it "Euphorium," that's the name of our mod:


Euphorium
Place mods here.txt

This is the base folder for our mod and all the data we create is going to go somewhere in here. Where? Well, that's What This Next Section Is For.

WHAT THIS NEXT SECTION IS FOR

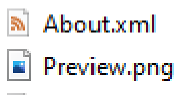
Create three folders in the "Euphorium" folder, called "About, Defs and Texture." Each of these folders has a different purpose:


About
Defs
Texture

- **About:** Contains the information **about** your mod, not the data for the mod itself. This is called "metadata" because it's data about data. This is going to tell Rimworld what your mod is called, give it a description, a preview picture, and a bit of info on which versions of Rimworld this mod can affect.
- **Defs:** Defs are the meat and potatoes of your mod. All the **data** for your mod lives here. This is the last thing we'll set up, as it is easily the most complicated.
- **Texture**: This is where all the **pictures** go that your mod will use. If you have a mod that adds new hair styles, the hair styles go here. If you have a mod that adds new guns, the images of the guns go here.

SETTING UP THE ABOUT FOLDER

To start with, we're going to fill in the "About" folder. We need two files in here, an "About.xml" and a "Preview.png:"

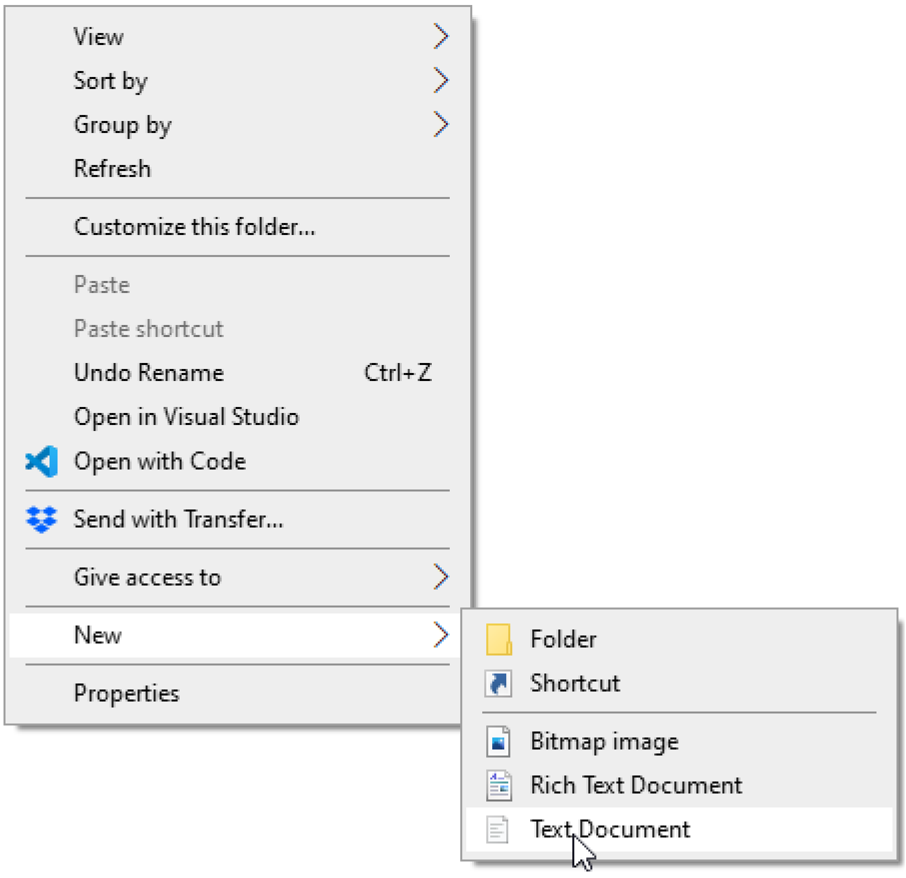

About.xml
Preview.png

Preview.png is the image that will get displayed when you view your mod in the mods menu in game, and is also the default image that gets uploaded to the steam workshop if you upload your mod there. Here's a Preview.png I prepared earlier, you can use this in your project, or boot up your favourite paint program and draw your own:

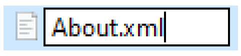
Use the link, don't try and grab this one, its so tiny.

Stick that in the About folder.

Now create a new file called “About.xml.” Right click in the folder, go to New->Text Document”



Now rename your file to “About.xml:”



NOTE FROM THE FUTURE: Windows hides file extensions by default. If you open up the file and it’s called “About.xml.txt” it means you’ve got file extensions hidden. Here’s how to turn them back on.

Open this file in your text editor of choice and prepare yourself carefully. You are now ready to jump into…

THE WILD WORLD OF XML

Skip this section if you know what XML is. If you don’t, keep reading.

Rimworld stores all of its data in XML. I’m not going to go into the history of XML or break down why it was the hottest file format of the 1990s. There’s a great article here that goes into a bunch of detail and I’d suggest reading it if you really want to understand what you’re doing. Instead I’m going to show you a little XML and try to explain it. **Don’t type it into the file you’ve got open**, this is just for reference:

```xml
<name>John Doe</name>
<occupation>game designer</occupation>
<age>28</age>
<weight>70</weight>
<!--This is a comment-->
```

What we are seeing here is XML. XML is a collection of `tags` and `values`. `Tags` (The things in blue) are "categories" of some kind that can have different `values` (the things in white). A good way to think about tags is that they're like the questions on a form (name, age, occupation etc). Everyone has a name, but the individual names (the values) can be different from one-another. Everyone has an age, but those numbers can be different from one-another. Similarly, every mod in Rimworld has a name, description and version numbers, but those values differ from mod to mod…

Basically, what I'm getting at is that making a mod is like filling in a big form.

Tags need to be formatted correctly for the game to read the XML file properly. Each tag needs to be surrounded with `<>` symbols, and we need to tell the game where the tag starts and ends. We do this by starting a tag with `<[TAG_NAME]>` and ending it with `</[TAG_NAME]>`. The latter tells the game that we are done with a tag and can move on to the next one. When writing XML, don't forget the "`/`" when you're closing a tag, it's very easy to do.

So what about `values`, what sorts of data can we stick in there? Well, as you can see from the above snippet, we can make values that are words(`John Doe, game designer `, and we can make values that are numbers (`28, 70`). Programmers call "word values" strings, because they have nothing to do with strings. There are a few other `values` that we can put in tags, which I will *cryptically allude to* now and explain later.

Finally The bottom line of this snippet says `<!--This is a comment-->`. Comments are just bits of text that we can leave in our XML file that the game will **completely ignore**. We can leave whatever *string* we want in a comment, we could remind ourselves what a particular tag does, we could leave a message to other mod developers that says `<!--DON'T STEAL MY MOD-->`, we could copy the entire Bee Movie script into a comment. You get the idea.

Awesome, so now XML is completely demystified and we can move onto...

ACTUALLY TYPING SOME XML

Skip this If you already know XML and just want to view the completed About.xml.

Okay, so XML is not *completely* demystified, but the best way to learn is by doing, right? So let's Do.

Back to our About.xml file. Remember, this is the file that will tell the game the name of our mod and what versions of Rimworld it is compatible with. So let's start typing…

```
<?xml version="1.0" encoding="utf-8"?>
<!--The above line is the start of every XML file. Just ignore it-->
```

NOTE FROM THE FUTURE: As of Rimworld 1.0, you don't need to create an XML declaration at the top of your file. You can skip this line if you want.

And we've already run into something weird. The above line is an XML declaration, it's a line that tells the game what version of XML we're using. In other pieces of software, this might be required, but in Rimworld it's optional. For now, let's continue writing some more XML on the line below the comment:

```
<ModMetaData>
    <name>Euphorium</name>
<description>This mod adds Euphorium to the game, a potent drug that will boost your pawn's stats at the risk of Ambrosia addiction</description>
    <author>DrEnzyme</author>
    <supportedVersions>
        <li>1.1</li>
        <li>1.2</li>
        <li>1.3</li>
    </supportedVersions>
    <packageId>drenzyme.euphorium</packageId>
</ModMetaData>
```

Ah gosh, That's a lot of stuff. What does it do? Let's go line by line.

`<ModMetaData>` is a tag that tells the game that the following info is all of the information related to our mod, and all of it is nicely packaged together. The weird part (that I *cryptically alluded to* before) is that this tag **CONTAINS OTHER TAGS**. Yes, you can have tags inside tags (programmers call this "nesting" because it has absolutely nothing to do with nests). Just like other tags, they start with `<[TAG_NAME]>` and end with `</[TAG_NAME>` but in the middle we can stick even more tags. Oh, and it doesn't matter that the `</ModMetaData>` bit is on a different line to `<ModMetaData>`, we sometimes stick things on different lines to make it look good. Programmers call sticking things on different lines "white space," because the gaps in between words or   tabs are white. You'll also notice that some of the lines in our snippet are indented (using the tab key), this is another example of white space. In XML, this white space is purely cosmetic, we're using it here to keep track of the *nested* tags, but in some languages (looking at you, python) white space has a big effect on how your code is interpreted.

All right, next line. This looks more familiar. `<name>` is a tag that contains the name of our mod. The value of it is `Euphorium`. It ends with a `</name>`

Similarly, `<description>` is a tag that contains the description for our mod, and `<author>` is a tag that contains your name… Or your username. It's probably best if you don't give out your real name *JAMES*…

If I get one person with that I'll feel validated.

Next line! Now we get to the `<supportedVersions>` tag, this is another tag that has tags in it. The tags in the tags (called child tags, because they have nothing to do with children) are `<li>`'s… Another new tag type! `li` means "list item". You can have a list of things in a tag too. In this case we have a list of supported version numbers for our mod. When a new version comes out, we can add that version number to the list (In a <li> tag) and Rimworld will know that our mod is compatible with that version number.

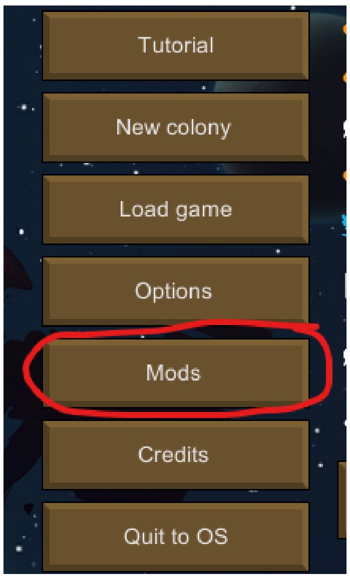
`<packageId>` is used by the steam workshop. It is a unique identifier that separates your mod from anybody else's. Some people format their packageId with the value `[username].[modname]`, like I've done here.

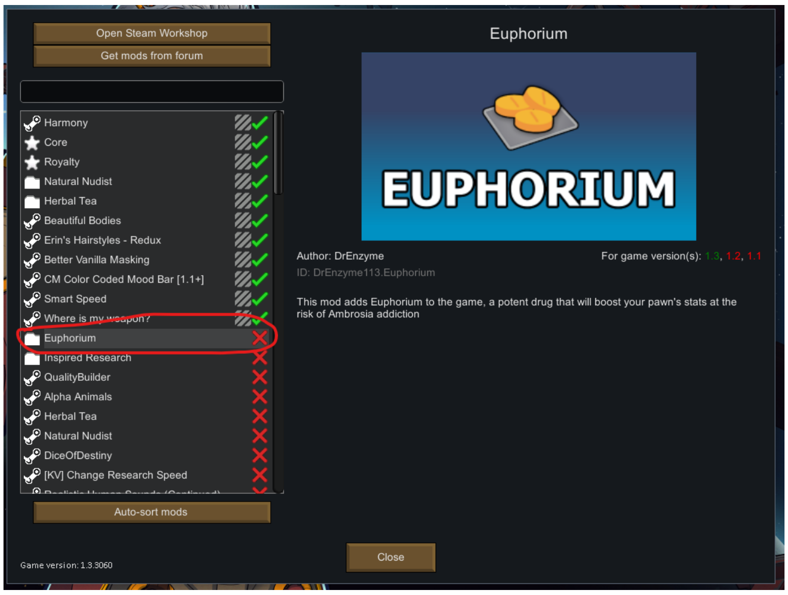
All right, sticking all of that together, your About.xml file should look like this:

THE COMPLETED ABOUT.XML

```xml
<?xml version="1.0" encoding="utf-8"?>
<!--The above line is the start of every XML file. Just ignore it-->
<ModMetaData>
    <name>Euphorium</name>
<description>This mod adds Euphorium to the game, a potent drug that will boost your pawn's stats at the risk of Ambrosia addiction</description>
    <author>DrEnzyme</author>
    <supportedVersions>
        <li>1.1</li>
        <li>1.2</li>
        <li>1.3</li>
    </supportedVersions>
    <packageId>drenzyme.euphorium</packageId>
</ModMetaData>
```

Awesome. Save your About.XML file and boot up Rimworld and see what your mod looks like in the mod list!:



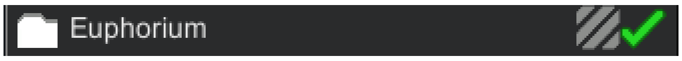


Author: DrEnzyme
For game version(s): 1.3, 1.2, 1.1
ID: DrEnzyme113.Euphorium

This mod adds Euphorium to the game, a potent drug that will boost your pawn's stats at the risk of Ambrosia addiction

Game version: 1.3.3060

It's there and it will do nothing, hooray! Now let's activate it:



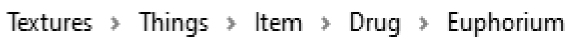
It still won't do anything, but it will do things once we add in the other bits.

Awesome! We've created our first XML file and we're *almost* ready to create the data for our mod!

Before we do that though, let's switch gears for a second and create the textures that the game will use for our drug.

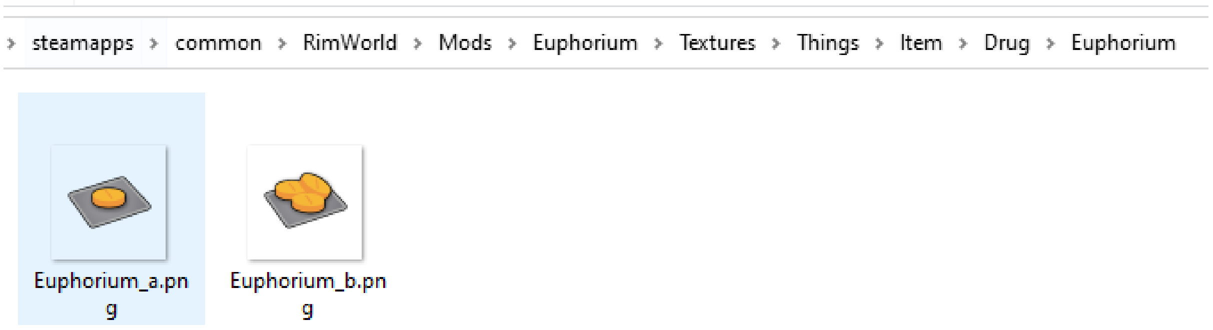
SETTING UP THE TEXTURES

This is a relatively easy part of making our mod. First we'll create a bunch of subfolders under the Textures folder. The folder structure should look like this:



Textures › Things › Item › Drug › Euphorium

Then we'll paste the images of our drug into the Euphorium folder:



› steamapps › common › RimWorld › Mods › Euphorium › Textures › Things › Item › Drug › Euphorium

Unfortunately this isn't an art tutorial, so I've already prepared some files for you to use. If you want to, you can create your own images to customise the look of your drug, otherwise, you can use these. Make sure you stick them in the path we've just set up and that they're named correctly:

Euphorium_a.png

Euphorium_b.png

As you can see, we have multiple images for our item. Rimworld has a "stack system," and the image of an item updates depending on how many items are in the stack. You might have noticed this with potatoes 🥔. In Rimworld, If there is one potato on the ground, it only shows you 1 potato. If there are a few more potatoes, the game shows you a box of potatoes. The same is true of steel, beer, silver, and of course, drugs. That's why our images have an "_a" or "_b" at the end, _a is used for the smallest stack size, and _b will be used for larger stack sizes. We could even add a "_c" image for even larger stack sizes, but I'm lazy, so you get two.

Anyway, that's all we need for the Textures folder, now we can move onto the main event…

SETTING UP THE DEF

WHAT IS A DEF

Def is short for "definition," in XML we will **define** what our item is and how it behaves. This is the final XML file we will need to write. It's a big one, so bear with me. Rimworld items have a lot of properties, and each property needs to be defined with tags and values. For example, our drug has a **recipe**, it has a **technology prerequisite**, it can be **manufactured** somewhere, it has a **work cost** to create it, it has **effects** when consumed, the list goes on. All of this needs to be written down somewhere or the game won't know what to do with our item.

Because of the huge number of properties an item can have, if you were looking to start creating an item outside of this tutorial I would recommend copying one that already exists and modifying it, rather than writing the XML from scratch. This is honestly the best way to start exploring other item types, and I'll have a section on this at the end of the tutorial. For now, we're going to go line by line again, this time even slower, and we're going to explain each thing we're adding. So let's roll up our ESleeves and get into it.

CREATING THE DEF

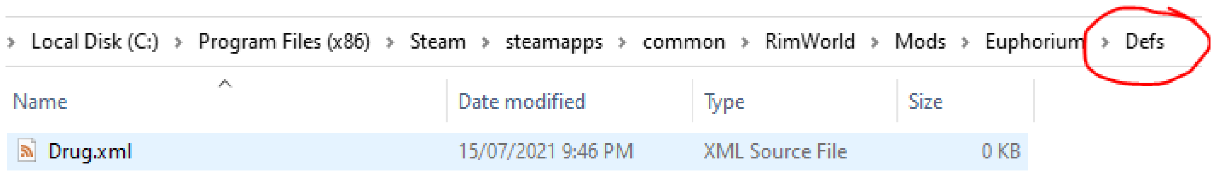
This is a pretty long and wild section, you might want to go make yourself a cup of coffee or something and have little breaks between the sections to make this easier to digest.

We're also going to be using a lot of tags that you've never seen before. You might start to wonder where they came from or how I knew that they needed to go in the def file. To make this mod I referenced the existing def files that the game provides, and there's a section on that at the end here. You might want to go take a look at some of the existing Defs while you're following along (the one for Ambrosia is pretty good).

Finally, if you don't want to write the XML yourself, or you want the completed file for this section, it can be found here. I'll also be linking the XML for the other sections at the end of those sections.

Anyway, enough chat, let's do this.

Go to the Defs folder in your mod, right click->new->text document and create an XML file called "Drug.xml":



› Local Disk (C:) › Program Files (x86) › Steam › steamapps › common › RimWorld › Mods › Euphorium › Defs

Name	Date modified	Type	Size
Drug.xml	15/07/2021 9:46 PM	XML Source File	0 KB

This can be called anything we want it to be. We could call it Euphorium.xml, CoolMod.xml, doesn't matter. I called it "Drug.xml" to prove that you can call it anything. Now we're going to open it in VSCode and start typing:

```
<?xml version="1.0" encoding="utf-8"?>
```

Remember this line from before? It's still optional, but I'm putting it in. Now we'll add our first tag beneath it:

```xml
<?xml version="1.0" encoding="utf-8"?>
<Defs>
</Defs>
```

First we add a `<Defs>` tag. This will contain ALL the definitions for our mod (you can have more than one). Inside the defs tag we'll add a *child tag* for the item we're creating, called "ThingDef," and inside that we'll add a couple more tags:

```xml
<?xml version="1.0" encoding="utf-8"?>
<Defs>
    <!--Euphorium is a ThingDef, and we are telling-->
    <!--the game that we want to make it a Drug-->
    <ThingDef ParentName="MakeableDrugBase">
        <!--What XML calls our item-->
        <defName>Euphorium</defName>
        <!--What Rimworld calls our item-->
        <label>Euphorium</label>
    </ThingDef>
</Defs>
```

We've just added four new tags `<ThingDef>`, `<ParentName>`,`<defName>` and `<label>.` `<ThingDef>` tells Rimworld that what we're creating is a physical object, a "thing" if you will. We can create definitions for stuff that isn't a physical object, like Research Nodes or Drug Effects. Those are different types of "Defs." `ParentName` is hard to explain, but it tells Rimworld that the code we want to run in association with this "thing" is the code for "Makeable Drugs", and not the code for Beds or Plants or something weird like that. It also uses a slightly different format when it comes to declaring itself, in XML we call this an "`attribute`," not a tag.
`<defName>` is the name of our item when we refer to it in XML, and `<label>` is the name that Rimworld will call our item. To quickly explain the difference, let's say we decided that "Euphorium" was a stupid name for our drug, and we wanted to call it "Floor Pills," we can change value of the `<label>` tag and Rimworld will start calling our drug "Floor Pills" without us having to change the name "Euphorium" everywhere in the XML. Anyway, now that most of you have changed the name of the drug to "Floor Pills," let's move on and write some more XML underneath `<label>` inside the `ThingDef` tag:

```xml
    <ThingDef ParentName="MakeableDrugBase">
        <!--What XML calls our item-->
        <defName>Euphorium</defName>
        <!--What Rimworld calls our item-->
        <label>Euphorium</label>
        <!--Description of our item when you click the "i" button-->
        <description>a potent drug that will heavily boost stats at the risk of
Ambrosia addiction.</description>
        <!--This stuff loads the textures for our drug-->
        <graphicData>
            <texPath>Things/Item/Drug/Euphorium</texPath>
            <graphicClass>Graphic_StackCount</graphicClass>
        </graphicData>
    </ThingDef>
```

The next tags we've added are `<description>` and `<graphicData>`. Description is a nice self-explanatory tag, it's the text that appears when you open the information window for an object in the game. `<graphicData>` is a little harder to explain.
We have two *child* tags inside `<graphicData>`, the first is the texture path `<texPath>` which tells the game where to find the graphics we set up before. Make sure that the folders are named **exactly the same as they are in the texture folder**. This means no "Drugs" instead or "Drug" or "Items" instead of "Item." They have to match.
The next tag is the `<graphicClass>` Remember how I said that some items in Rimworld change appearance depending on the stack size? Well some items don't, like clothing and guns, so we need to be specific. Here we're telling Rimworld that we want the item to change appearance based on how many items are in the stack.

All right, let's add some more tags beneath `<graphicData>`:

```
	<ThingDef ParentName="MakeableDrugBase">
		<!--What XML calls our item-->
		<defName>Euphorium</defName>
		<!--What Rimworld calls our item-->
		<label>Euphorium</label>
		<!--Description of our item when you click the "i" button-->
		<description>a potent drug that will heavily boost stats at the risk of 
Ambrosia addiction.</description>
		<!--This stuff loads the textures for our drug-->
		<graphicData>
			<texPath>Things/Item/Drug/Euphorium</texPath>
			<graphicClass>Graphic_StackCount</graphicClass>
		</graphicData>
		<statBases>
			<!--How much work does it take to make this item-->
			<WorkToMake>500</WorkToMake>
			<!--How much is this item worth to traders?-->
			<MarketValue>10</MarketValue>
			<!--How much does the item weigh, in KG I think?-->
			<Mass>0.01</Mass>
		</statBases>
	</ThingDef>
```

Here we've added `<statBases>`. There are a bunch of these that you can define for your item here (another one would be *flammability,* another would be *nutrition)*, but we're going to stick with these three tags:.
`<WorkToMake>` is how much work it takes to make the item. If you've tailored a duster or made a sculpture in the game, you'll have noticed the amount of work ticking down as your pawns work on the item. This is that value.
`<MarketValue>` is how much the item is worth in silver.
`<Mass>` is how much the item weighs in kg.

Nice and simple, right? Riiigghhhttt...

Next we'll add in the recipe for our item below statBases:

```
        <statBases>
            <!--How much work does it take to make this item-->
            <WorkToMake>500</WorkToMake>
            <!--How much is this item worth to traders?-->
            <MarketValue>10</MarketValue>
            <!--How much does the item weigh, in KG I think?-->
            <Mass>0.01</Mass>
        </statBases>
        <!--This is the recipe for our item-->
        <!--I.e what it takes to craft it-->
        <costList>
            <Ambrosia>2</Ambrosia>
            <Neutroamine>2</Neutroamine>
            <InsectJelly>5</InsectJelly>
        </costList>
    </ThingDef>
</Def>
```

The comment should clarify what we've just added, it's the recipe for our item. I've gone with **2 Ambrosia**, **2 Neutroamine** and **5 Insect Jelly**, but you can craft it out of whatever you like. You can craft your floor pills out of potatoes, if that's what you'd like to do.

We're almost there, two more sections and we'll actually be able to see what we've created in the game. Underneath the cost list we're going to specify the ingestible values of our drug:

```
            <Ambrosia>2</Ambrosia>
            <Neutroamine>2</Neutroamine>
            <InsectJelly>5</InsectJelly>
        </costList>
        <!--All the values we need for an ingestible drug-->
        <ingestible>
            <foodType>Processed</foodType>
            <joyKind>Chemical</joyKind>
            <joy>0.30</joy>
            <drugCategory>Hard</drugCategory>
            <baseIngestTicks>150</baseIngestTicks>
            <chairSearchRadius>4</chairSearchRadius>
            <!--Can someone feed you this drug-->
            <nurseable>true</nurseable>
            <!--Put outcomedoers here later-->
        </ingestible>
```

This is another hefty chunk of XML, so let's go line by line.
The `<ingestible>` section we've just added describes what happens when we take our drug.
The first child tag, `<foodType>`, tells the game what type of food this is. Other values here could be "`VegetableOrFruit`," "`Fluid`" etc.
`<joyKind>`is chemical. That should make sense. I think this is mainly used to determine when pawns are "seeking recreation variety." It might also be used to satisfy pawns that have a "chemical fascination," but don't hold me to that.
`<joy>` is the amount of joy that will be added to a pawn's recreation bar when they take this drug. Its max

value is 1, which would fill the recreation meter completely. I've set it to 0.3, but you can decide how fun your drug is.
`<drugCategory>` is set to `Hard`. There are `Social` drugs in the game too, beer, ambrosia or smokeleaf fall into that category.
`<baseIngestTicks>` is how long it'll take a pawn to "eat" our drug. The lower this value is, the less time it'll take.
`<nurseable>` is our first `true/false` tag! Programmers use "true" or "false" instead of "yes" or "no," because underneath their false skin they're soulless automatons. If this tag is set to true, pawns will be able to feed the drug to other pawns. This is super useful if the pawn we want to take the drug is downed and we want to use Euphorium to fix them up.

`<!--Put outcomedoers here later-->` is an ominous portent of things to come. Remember it for later. Or don't, I'll remind you when we get there.

All right, last section and we can test it in game:

```
            <nurseable>true</nurseable>
            <!--Put outcomedoers here later-->
        </ingestible>
        <!--This comp is needed for the "drug policy" menu-->
        <!--It also defines a bunch of important properties about the drug-->
        <comps>
            <li Class="CompProperties_Drug">
                <chemical>Ambrosia</chemical>
                <addictiveness>0.030</addictiveness>
                <minToleranceToAddict>0.15</minToleranceToAddict>
                <listOrder>40</listOrder>
            </li>
        </comps>
    </ThingDef>
</Defs>
```

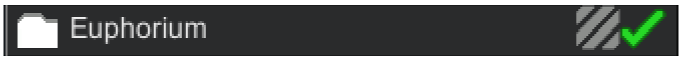
Here we're adding a "component" (using the `<comps>` tag) to our drug. This is another list, as our drug can have more than one type of component (`"CompProperties_Rottable"` is another type of component for instance, Ambrosia has that one because it can rot).
`<chemical>` is `Ambrosia`. Different drugs can share chemicals (Flake, Psychite Tea and Yayo all share the "Psychite" chemical for instance), so this lets us set up those kinds of drugs. We're going to share our chemical type with Ambrosia.
`<addictiveness>` is mostly self explanatory, it's how likely you are to get addicted to our drug when you take it. Normal Ambrosia has a value of `0.01`, so I've increased it here to reflect Euphorium being a "hard" drug.
`<minToleranceToAddict>` is what it sounds like. You can't get addicted to our drug unless you already have a bit of tolerance to it. So if you let the drug leave your system and take it sparingly, you're less likely to get addicted.
`<listOrder>` Relates to the drug policy menu. The higher it is, the lower down your drug will appear in the drug policy list.

Finally, with this last addition, we have enough XML to see our drug in the game. Just a reminder that the completed version of the XML file for this section is here, so if you have any problems or errors you can cross reference yours with that one.

TESTING YOUR MOD IN THE GAME

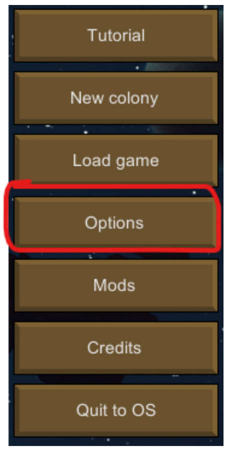
We've written enough XML that the game will know how to handle our "Euphorium" drug without errors, so let's boot up Rimworld and **check it out!**.

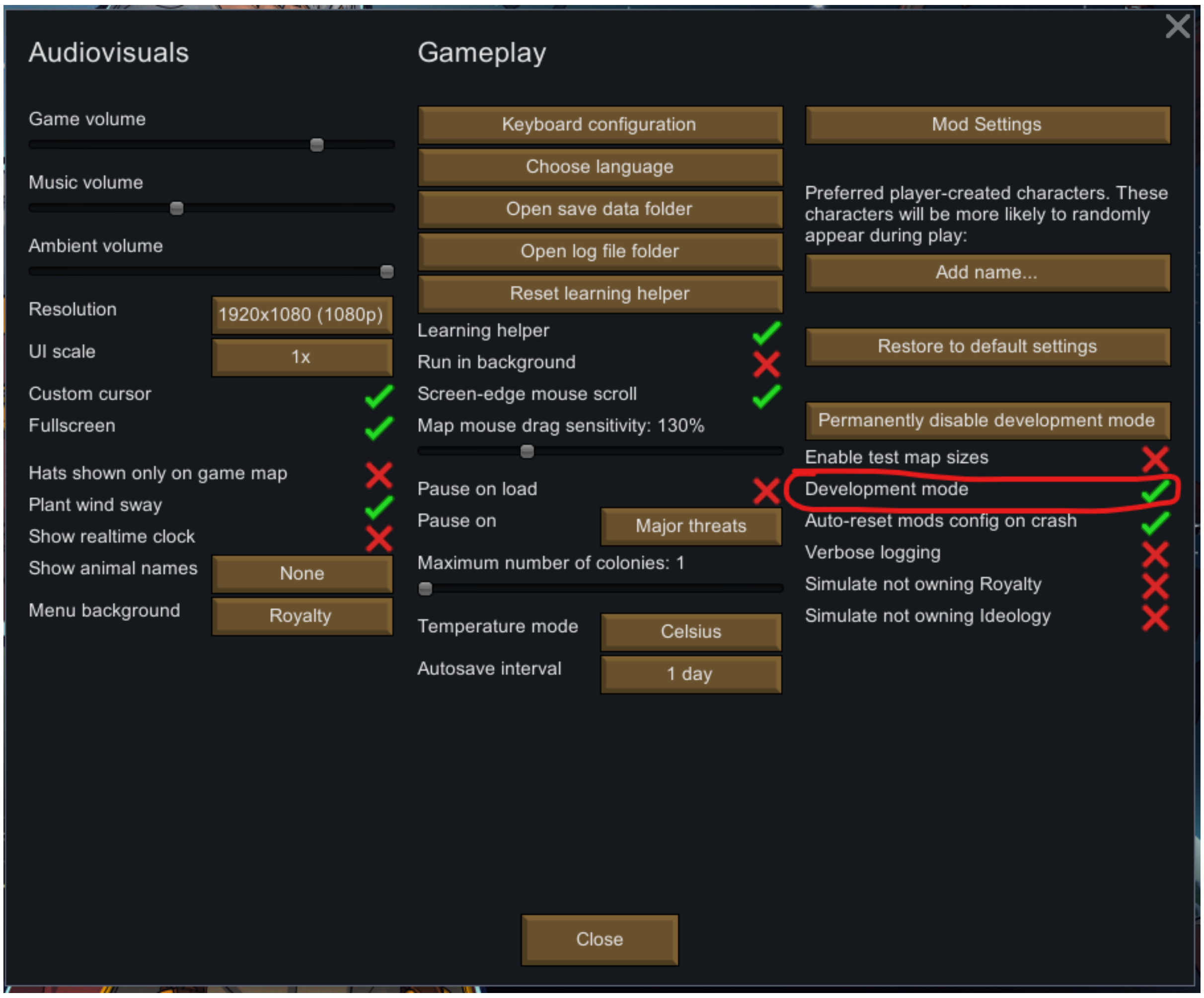
Before we go into the actual game, there are a couple of things to double check in the Rimworld menus. The first is that we've activated the mod in the mods menu (you did do that before, right?):



Euphorium ✓

The second thing we need to do is enable "Developer Mode" in the options menu. It's okay, I'm officially promoting you to developer status, my uncle works at Ludeon and he said I could so it's fine:





Audiovisuals

Game volume

Music volume

Ambient volume

Resolution: 1920x1080 (1080p)
UI scale: 1x
Custom cursor ✓
Fullscreen ✓
Hats shown only on game map ✗
Plant wind sway ✓
Show realtime clock ✗
Show animal names: None
Menu background: Royalty

Gameplay

Keyboard configuration
Choose language
Open save data folder
Open log file folder
Reset learning helper
Learning helper ✓
Run in background ✗
Screen-edge mouse scroll ✓
Map mouse drag sensitivity: 130%
Pause on load ✗
Pause on: Major threats
Maximum number of colonies: 1
Temperature mode: Celsius
Autosave interval: 1 day

Mod Settings

Preferred player-created characters. These characters will be more likely to randomly appear during play:

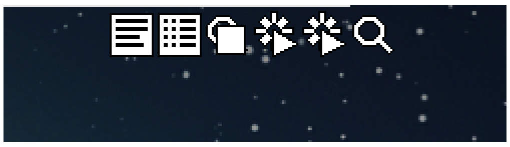
Add name...

Restore to default settings

Permanently disable development mode
Enable test map sizes ✗
Development mode ✓
Auto-reset mods config on crash ✓
Verbose logging ✗
Simulate not owning Royalty ✗
Simulate not owning Ideology ✗

Close

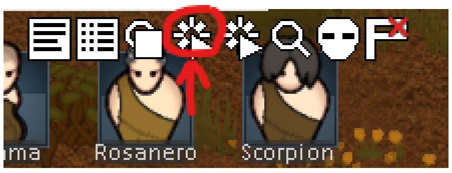
Developer mode adds a bunch of little buttons **at the top of the screen**, which we'll be using in a second:



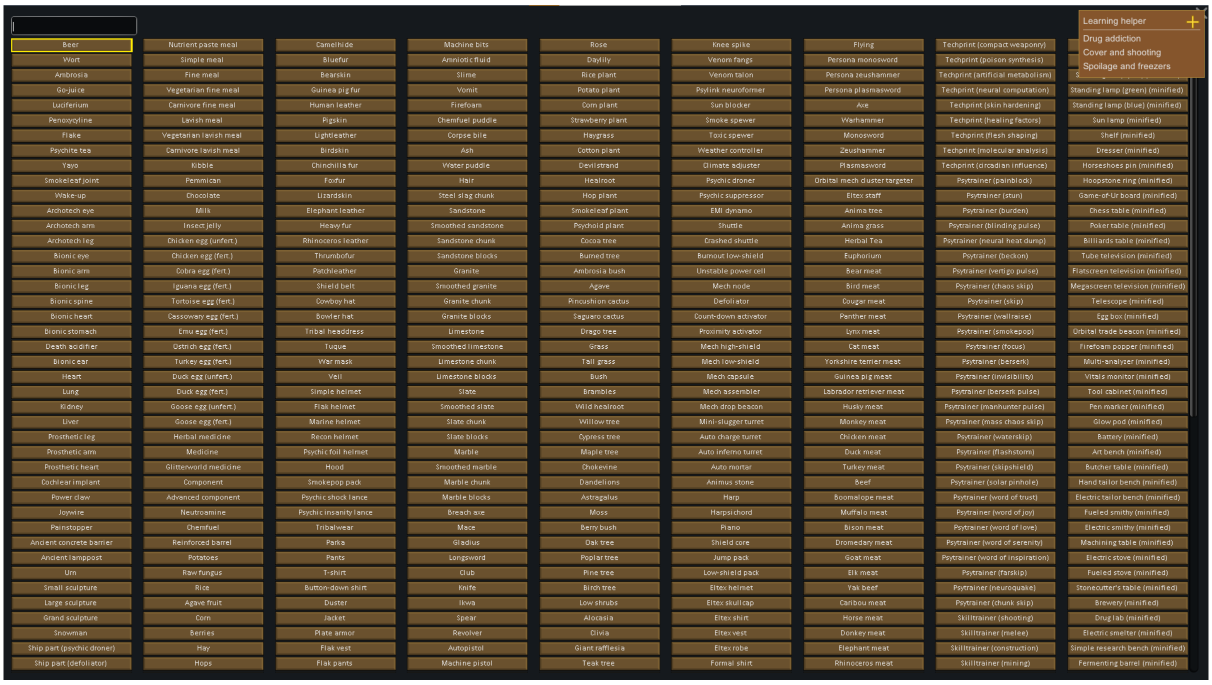
Next we need to set up a colony to test with. Do it the same way you usually would, go to New game, pick a storyteller, pawns, a starting location etc. You can choose whatever settings you like, but I'd recommend you **don't choose commitment mode** for your storyteller. We're going to want to be able to reload our game a bunch of times. After all that is set up and you're in the game, immediately save it. It's nice to have a fresh colony when testing stuff and we don't want to have to go through worldgen every time we make a change to our mod.

Great! Now we're going to use some of our developer mode buttons to spawn our item. Although we gave Euphorium a recipe, we never actually specified where it was made, so the only way to get it (for now) is from traders or the debug menu.

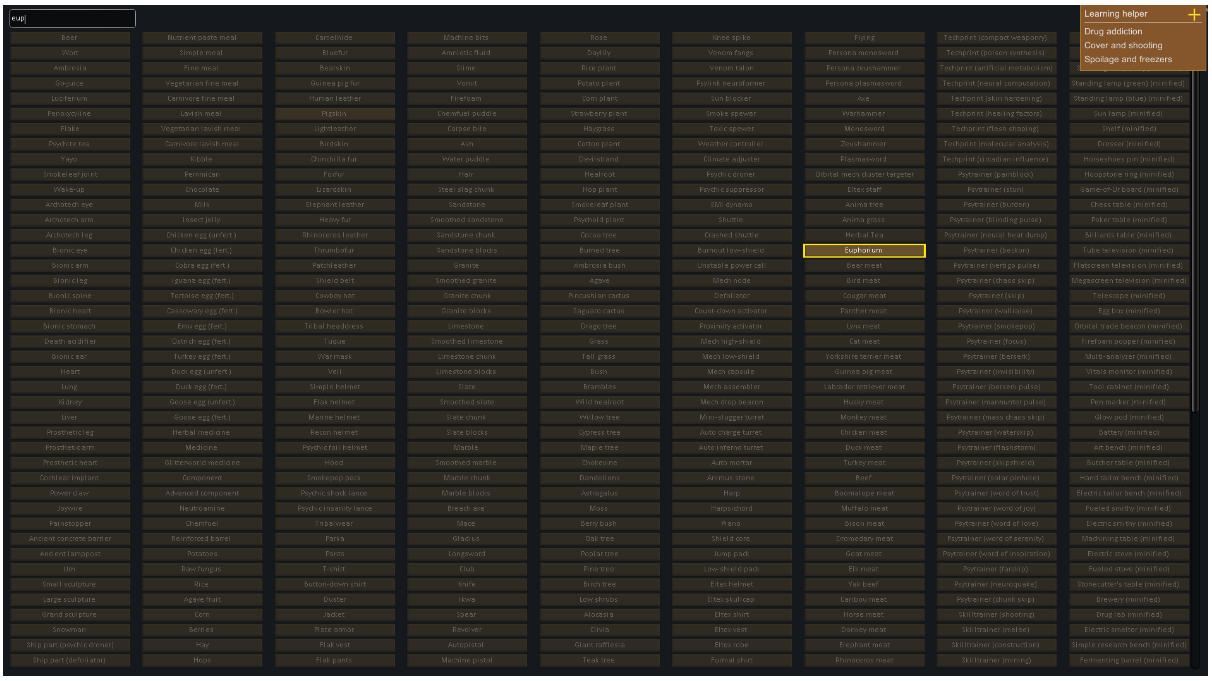
First, click on this button here at the top of the screen:



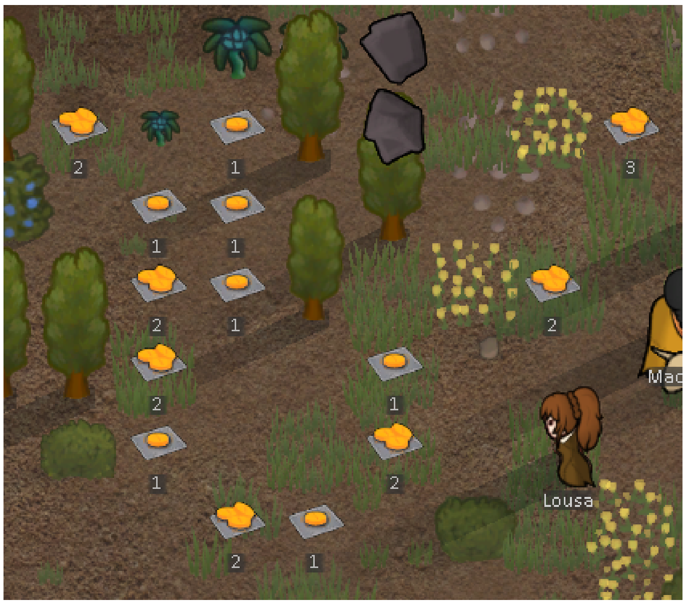
You'll see a huge array of options here. We're going to use the search bar to search for "try place" then select "try place near thing," That will bring up this menu with all the items in the game:



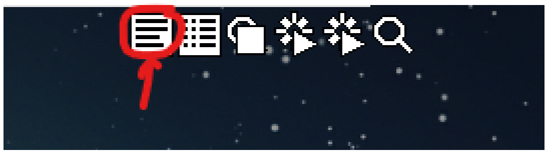
Once again, the search bar is our friend. Type "Eup" and the game will grey out everything except Euphorium. Click on it.



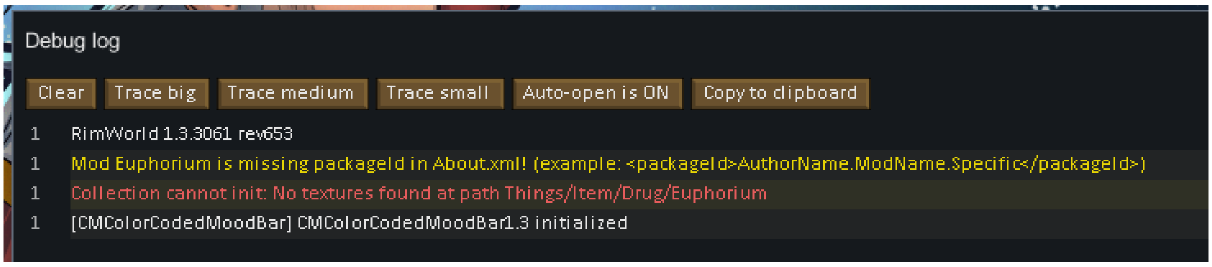
If everything has gone well, you can now click on the ground and spawn a bunch of Euphorium all over the place. Give yourself a pat on the back, because getting this far wasn't easy.



If you can't get Euphorium to spawn, it probably means you've got some errors. You can check to see your errors in the log using this button:



The red ones are errors. Here's an example one, which you'd get if you didn't set up the textures properly:

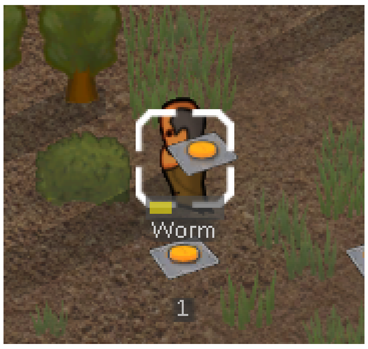


```
1   RimWorld 1.3.3061 rev653
1   Mod Euphorium is missing packageId in About.xml! (example: <packageId>AuthorName.ModName.Specific</packageId>)
1   Collection cannot init: No textures found at path Things/Item/Drug/Euphorium
1   [CMColorCodedMoodBar] CMColorCodedMoodBar1.3 initialized
```

If you see a bunch of errors, it's best to look at the top of the list and go from there. Subsequent errors can be caused by previous errors, so when you fix the first one, the rest might disappear!

Unfortunately, I can't see what you see, so I won't be able to solve any errors that have occurred. You might just have to use the pre-made XML if you're having trouble, or cross reference your data with it.

Back to our game, If you select a pawn, you'll be able to tell them to consume Euphorium. Good times...



Except it's not good times because Euphorium isn't doing anything! Ah dang, we did all that XML and we still haven't added the effects of the drug itself!

ADDING THE DRUG EFFECTS

Let's exit Rimworld and head back to the XML. There are **two parts** we'll need to add to our XML to tell Rimworld what effects our drug has. The first part is to tell the game what effects to give a pawn when the drug is consumed, the second is to **define** the effect itself.

Remember "`<!--Put outcomedoers here later-->`"? Well now we're going to add those "outcome doers" into the XML, right under that comment:

```
            <outcomeDoers>
                <li Class="IngestionOutcomeDoer_GiveHediff">
                    <hediffDef>EuphoriumHigh</hediffDef>
                    <severity>0.50</severity>
                    <toleranceChemical>Ambrosia</toleranceChemical>
                </li>
                <li Class="IngestionOutcomeDoer_GiveHediff">
                    <hediffDef>AmbrosiaTolerance</hediffDef>
                    <severity>0.032</severity>
                </li>
            </outcomeDoers>
```

There's a bit going on here. `<outcomeDoers>` are the things that are going to happen when we take our drug. It's a list `<li>` of effects that get applied, and each effect has the following tags:
`<hediffDef>` is the name of the **definition** for the effect that we want to apply. Remember how I said definitions don't need to be "things," they can be research nodes, thoughts, etc. Well "drug effects" have their own definition type too. In fact we're going to be creating one shortly called "`EuphoriumHigh`"
`<severity>` is "how much" of this effect we want to apply. Severity slowly ticks down over time and eventually wears off completely, and if we want our drug to stay in a pawn's system longer, we can increase this value.
`<toleranceChemical>` If you have a tolerance to this chemical (in our case, `Ambrosia`), the effect is not going to be as strong as if you were taking it for the first time. If no tolerance chemical is specified, you'll

get the full effect of the drug each time you take it.

All right, with that set up, we can now create the **definition** for the effect of the drug. This bit of XML needs to go after the `</ThingDef>` for the Euphorium item, but before the `</Def>` tag at the end of the file:

```xml
    </ThingDef>

    <!--This is the definition for our drug's effect-->
    <HediffDef>
        <!--This is used by other defs when referring to this def-->
        <defName>EuphoriumHigh</defName>
        <!--Strings that the game uses when describing our drug effect-->
        <label>high on Euphorium</label>
        <labelNoun>a Euphorium high</labelNoun>
        <description>An amazingly potent drug that gives a boost to many stats at
once.</description>
        <hediffClass>Hediff_High</hediffClass>
        <defaultLabelColor>(1,0,0.5)</defaultLabelColor>
        <maxSeverity>1.0</maxSeverity>
        <isBad>false</isBad>
        <comps>
            <li Class="HediffCompProperties_SeverityPerDay">
                <severityPerDay>-3.0</severityPerDay>
                <showHoursToRecover>true</showHoursToRecover>
            </li>
        </comps>
        <stages>
            <li>
                <painFactor>0.5</painFactor>
                <restFallFactor>0.33</restFallFactor>
                <statOffsets>
                    <ImmunityGainSpeed>0.1</ImmunityGainSpeed>
                </statOffsets>
                <capMods>
                    <li>
                        <capacity>Consciousness</capacity>
                        <offset>0.20</offset>
                    </li>
                    <li>
                        <capacity>Sight</capacity>
                        <offset>0.35</offset>
                    </li>
                    <li>
                        <capacity>Moving</capacity>
                        <offset>1.0</offset>
                    </li>
                </capMods>
            </li>
        </stages>
    </HediffDef>
</Defs>
```

Boy, that's a lot of XML, isn't it? But you might notice a few similarities between this **def** and the first one we made. Both **definitions** have a `<defName>`, `<label>` and a `<description>`. `<HediffDef>` is a new type of **definition** (A Health-Difference-Definition)**,** we created a `<ThingDef>` for the physical item of our drug before, but this isn't a physical item, which is why we're using this new type of Def.

Moving through the XML, we see a `<labelNoun>`, this is just another way for the game to refer to "A Euphorium High" in a different context (as a noun). It's just another way of describing the drug effect.

Further down we see `<hediffClass>`, which tells us what "class" our Hediff falls into. In code, classes are collections of code that run for a particular object, since this is a "High," we want the code for `Hediff_High` to run for this object. There are other Hediff classes for Implants, Pregnancy, Diseases etc, each with slightly different code.

This next one is kind of interesting. `<defaultLabelColor>` is a color, it's defined in RGB (Red, Green, Blue) format. We've defined it as `(1,0,0.5)`, which means **FULL RED**, **NO GREEN**, AND **HALF BLUE.** If you were to visualize mixing those colours as if you were mixing paint, you'd get a kind of purpley-red color. The purpley red color is the color that the effect will use when displayed in game on the health tab.

`<maxSeverity>` is the maximum severity a drug can hit when you take a whole bunch at once. Having it be `1.0` is nice because it is consistent with other point values in the game.

`<isBad>`... Is really weird. Literally all of the drug effects in base Rimworld have this set to `false`. I know that weather events have this set to true so that forced weather events know which weather effects are bad enough to be worth a quest (a forced weather event for "gentle snow" isn't really that intimidating), but I have no idea what it does in the context of drugs. It's just… There for some reason.

NOTE FROM THE FUTURE: I've been told that `<isBad>` relates to the kinds of Hediffs that the various Mech Serums can heal.

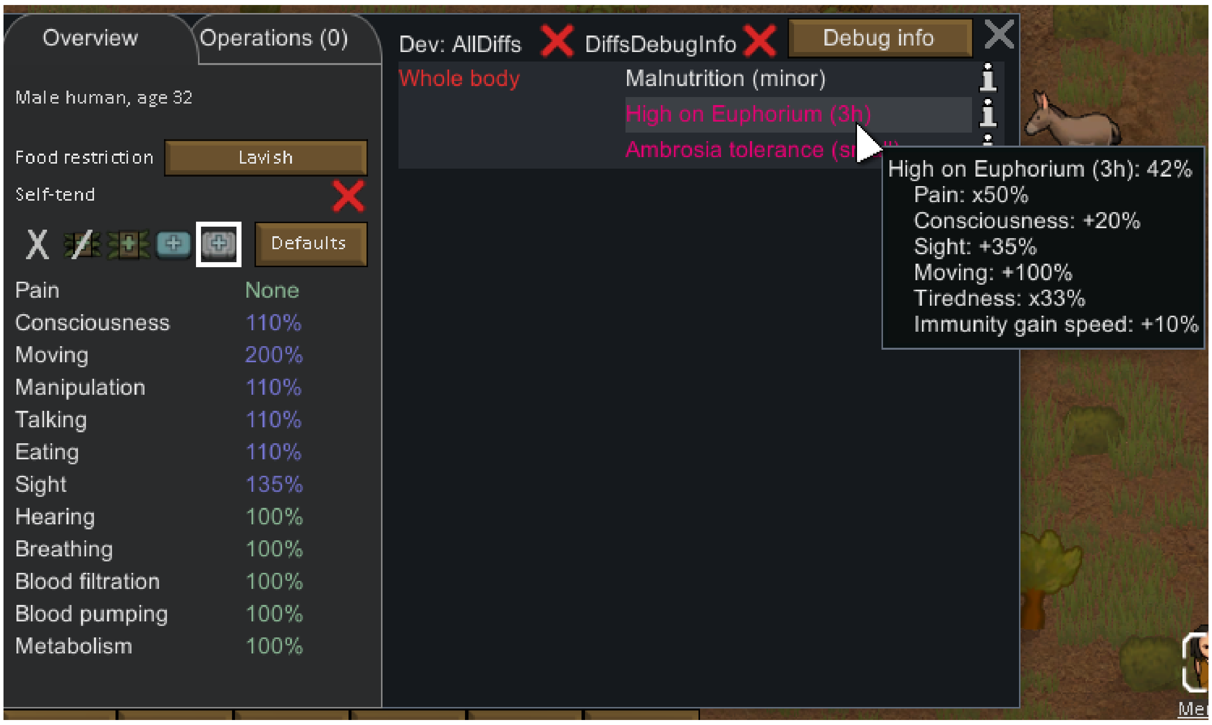
`<comps>` are a familiar friend from our previous Def, but here they're used in a different way. We have a list `<li>` of 1 comp called `"HediffCompProperties_SeverityPerDay"`. This causes the severity of the drug (how much of it is in a pawn's system) to slowly disappear. `<severityPerDay>` is how much severity will wear off each day, it is a negative value, because we want the severity to go down, not up (though you could create some cool drugs where the severity gets higher and higher over time, that would be fun).
`<showHoursToRecover>` lets us see how long the effect will last from inside the game. Sometimes you want to hide this away from the player so they don't know how long an effect will last, but for this drug we've set it to true.

Last, but certainly not least, is the `<stages>` tag. Drugs can have many stages depending on the severity (for instance, alcohol has a "warm" stage, "tipsy" stage and a "drunk" stage based on how much beer has been drunk), but here we have a list of 1 stage, because we want the effect to be the same regardless of severity.
Inside that `<li>` tag we have a BUNCH of effects. These are all the stats that our drug effect will change once consumed. I've got a whole lot in here that I pulled from a bunch of different drugs in the base game. There's an effect that modifies the pawn's movement speed by 100% (`<capacity>Moving</capacity>`), there's a `<painFactor>0.5</painFactor>` effect to make the pawn feel less pain, there's one for `<ImmunityGainSpeed>0.1</ImmunityGainSpeed>` so the pawn will gain immunity to disease faster and there's a couple of others too. This is designed to be a super-drug after all, so might as well give it some juice.

And that's it. Next time we run the game our drug will actually do something. Before we do that, [Here is the completed XML](#)for this section of the tutorial. If you've already downloaded the first one, make sure you delete it, you don't want both of them in your Defs folder.

TESTING IT AGAIN

Back to the game again, and we'll need to **restart Rimworld** to load the changes to our mod. If everything has gone well, you can get a volunteer to consume some of the Euphorium on the ground, and they'll get this **MASSIVE boost** to all of their stats.



Male human, age 32

Food restriction Lavish

Self-tend

Pain	None
Consciousness	110%
Moving	200%
Manipulation	110%
Talking	110%
Eating	110%
Sight	135%
Hearing	100%
Breathing	100%
Blood filtration	100%
Blood pumping	100%
Metabolism	100%

Whole body — Malnutrition (minor); High on Euphorium (3h); Ambrosia tolerance (s

High on Euphorium (3h): 42%
Pain: x50%
Consciousness: +20%
Sight: +35%
Moving: +100%
Tiredness: x33%
Immunity gain speed: +10%

Now obviously there are more stats here that we could mess with, we could change metabolism or blood filtration or whatever. I'll leave you to figure out how to mod those if you want to. For now, we're going to move onto...

ADDING RESEARCH/MANUFACTURING STUFF

We're nearly done (I promise), but spawning our drug through the debug menu doesn't feel right. A **real** Rimworld item can be made in game, and it usually has some kind of research to unlock the recipe. To do that, we need to add two more sections of XML. The first is a `<recipeMaker>` section added to the ThingDef that we created, the second is a new type of definition, called a "`<ResearchProjectDef>`," which will define the piece of research that will unlock Euphorium.

Starting with the `<recipeMaker>` section, we want to add this underneath the `</comps>` tag in our ThingDef:

```
        <!--This gives the game information about the research we need-->
        <!--to unlock this item, and the buildings that we can make-->
        <!--our item from.-->
        <recipeMaker>
            <researchPrerequisite>EuphoriumProduction</researchPrerequisite>
            <recipeUsers>
                <li>DrugLab</li>
            </recipeUsers>
            <soundWorking>Recipe_Drug</soundWorking>
        </recipeMaker>
```

This is a nice *little* piece of XML for once.
`<researchPrerequisite>` is the `<defName>` of the research we need to unlock before we can make our drug. We're going to be creating that in a second.
`<recipeUsers>` is a list `<li>` of the production facilities that can make this item. Sometimes you can make the same item from multiple different production buildings (a fueled stove and an electric stove can both make "meals" for example), but for Euphorium it makes sense to make it from a drug lab, so there's just 1 entry in this list.
`<soundWorking>` is the sound that plays when a pawn is creating our item.

Last piece of XML, and then we're done! This is the ResearchProjectDef and it needs to go between our `</HediffDef>` tag and the `</Defs>` tag:

```
<!--This is the research project that will unlock our drug.-->
    <ResearchProjectDef>
        <defName>EuphoriumProduction</defName>
        <label>Euphorium Production</label>
        <description>Refine Ambrosia into a powerful drug that vastly increases a wide
range of stats.</description>
        <!--Number of research points it costs to unlock our drug-->
        <baseCost>400</baseCost>
        <!--How "advanced" this research is-->
        <!--Tribal factions will have a harder time researching it-->
        <techLevel>Industrial</techLevel>
        <!--Research needed before this node becomes available-->
        <prerequisites>
            <li>DrugProduction</li>
        </prerequisites>
        <!--Position on the research screen-->
        <researchViewX>4.00</researchViewX>
        <researchViewY>3.20</researchViewY>
    </ResearchProjectDef>
```

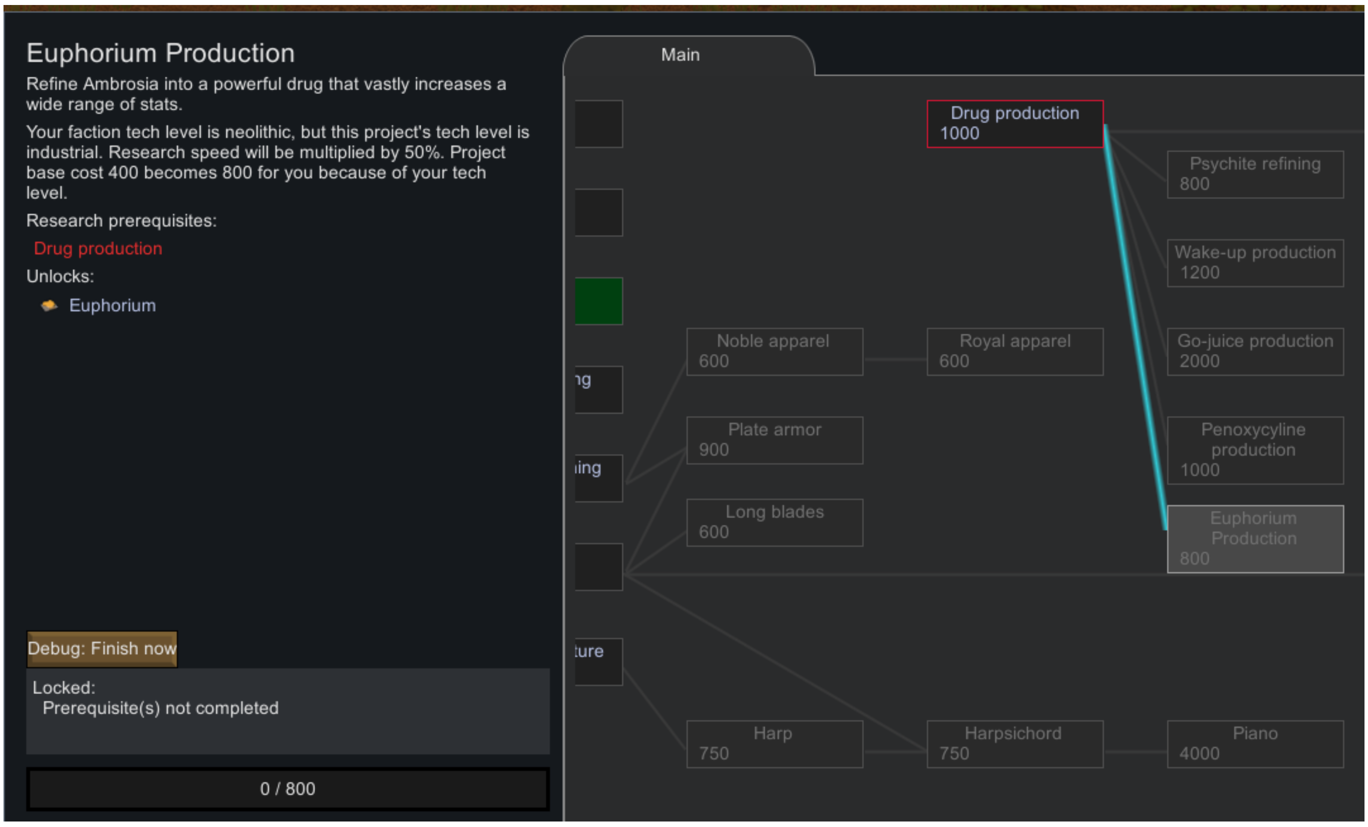
Looking through this snippet, our old friends `<defName>`, `<label>` and `<description>` are still here, and they still do the same thing that they do in other Defs.
`<baseCost>` is new, that's the number of research points we need to unlock this research in game.
`<techLevel>` is an important part of the points calculation, did you know that tribals have a harder time researching advanced technology than spacemen? It's true, and this defines how "advanced" our technology is.
`<prerequisites>` is a list of technologies that need to be researched before this one can be researched. In this case we want "`DrugProduction`" as a prerequisite for this research.
Finally, `<researchViewX>` and `<researchViewY>` control the location that our research node will appear on the research screen. They're (x,y) coordinates, just like a graph has an x axis and a y axis. The x coordinate is the horizontal position and the y coordinate is the vertical position.

And that's it, our XML is complete. Here's my copy of the final .XML file, in case you want that. All that's left to do is to test it in game, one last time...

A FINAL TEST

Once more into the breach. Restart Rimworld and let's see if everything works.

Once we're in-game we can check the research tree to see if Euphorium production is a node we can research, and there it is!:



Euphorium Production

Refine Ambrosia into a powerful drug that vastly increases a wide range of stats.

Your faction tech level is neolithic, but this project's tech level is industrial. Research speed will be multiplied by 50%. Project base cost 400 becomes 800 for you because of your tech level.

Research prerequisites:

Drug production

Unlocks:

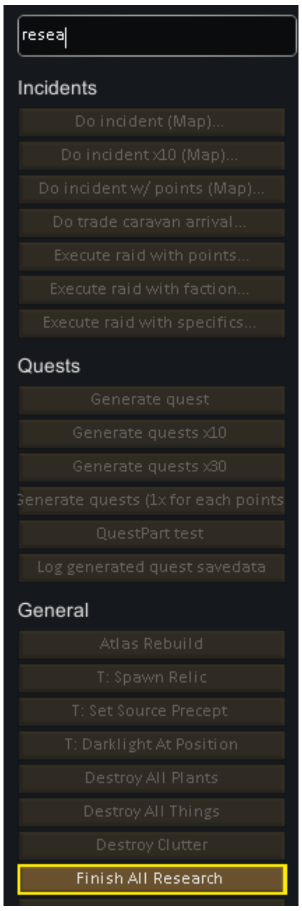
Euphorium

Locked:
Prerequisite(s) not completed

0 / 800

Not exactly the most elegant place to put it but it'll do.

There's also an option in the debug menu to finish all research. Let's do that now so we have access to the Drug Lab and Euphorium production:


resea

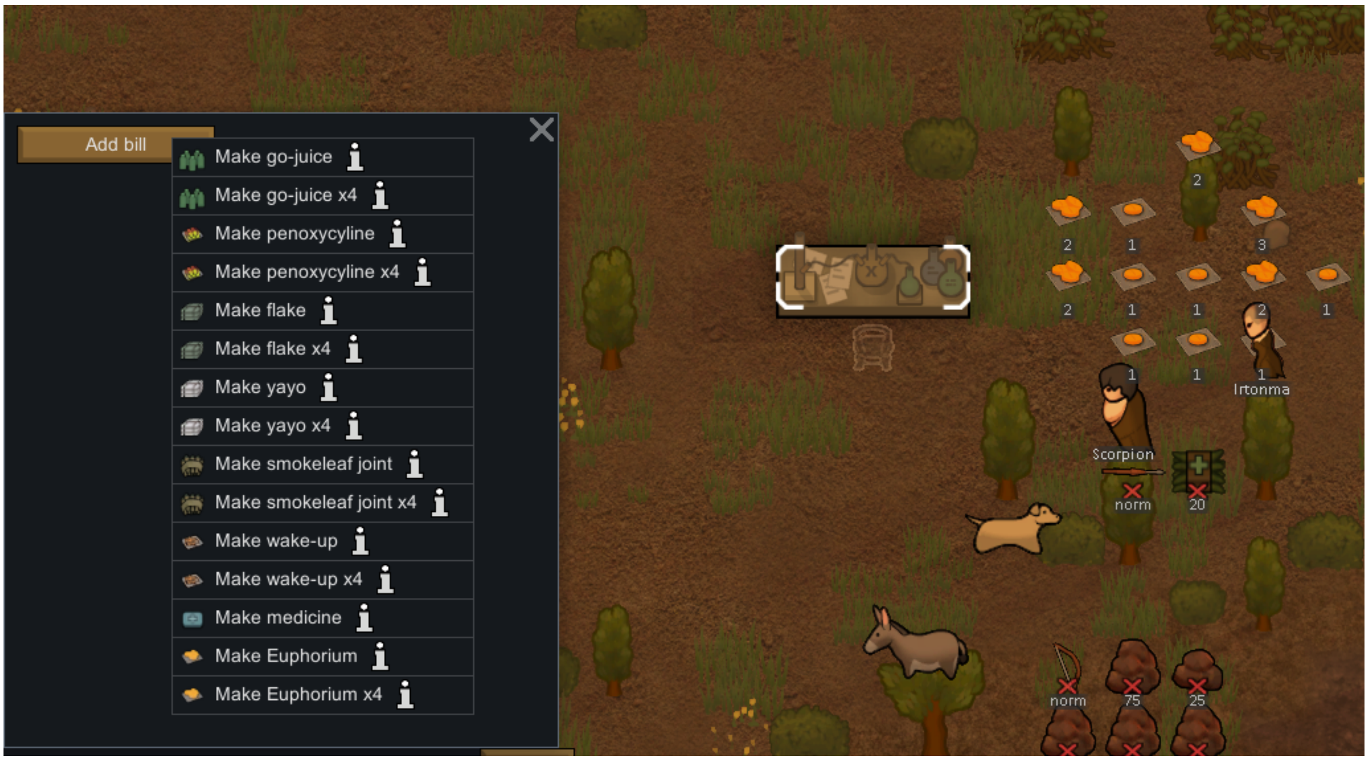
Incidents

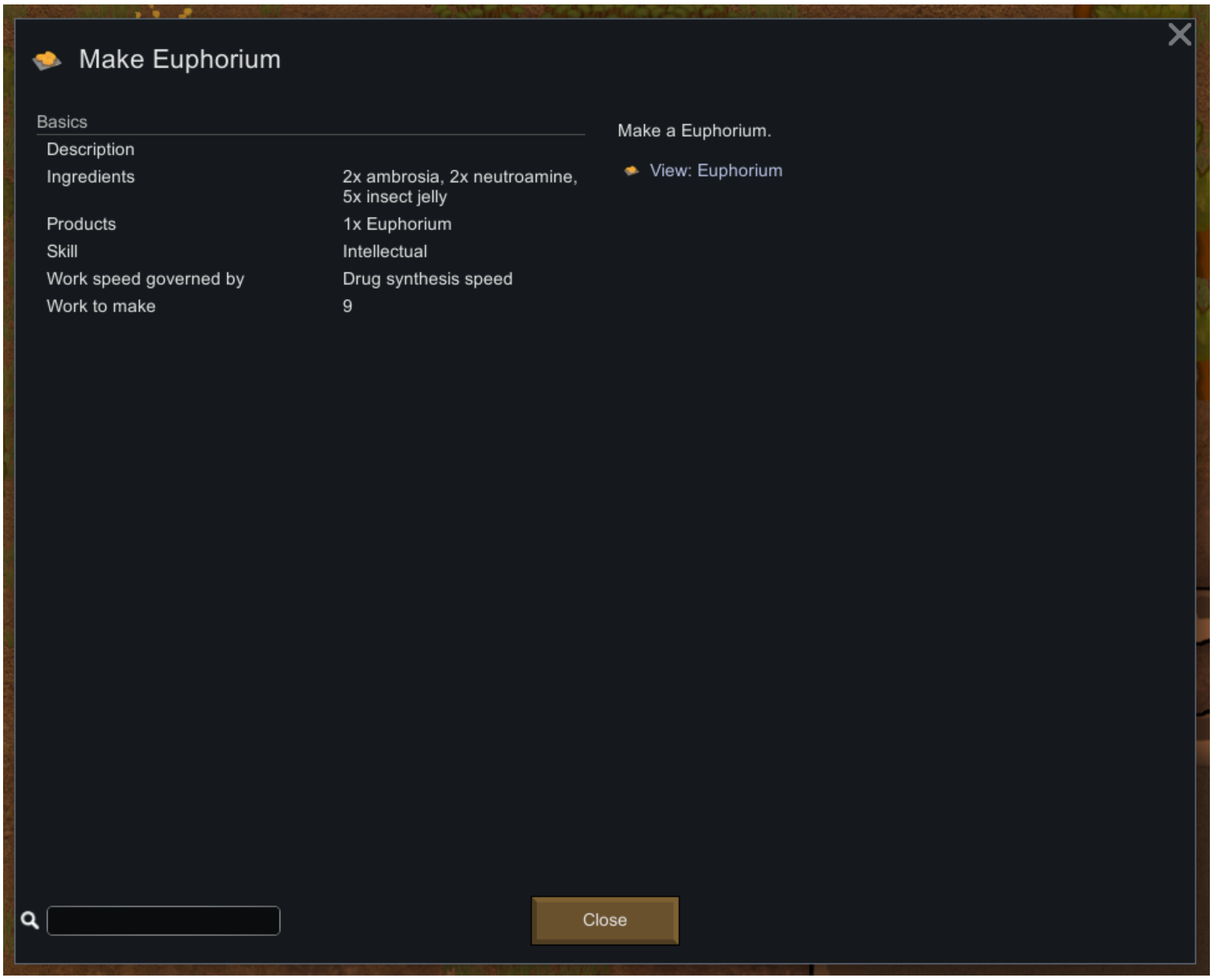
Do incident (Map)...
Do incident x10 (Map)...
Do incident w/ points (Map)...
Do trade caravan arrival...
Execute raid with points...
Execute raid with faction...
Execute raid with specifics...

Quests

General

We can also use the “place thing near” debug option to quickly get a drug lab instead of having to build one. Now we can check that we can produce Euphorium at the drug lab:





Make Euphorium

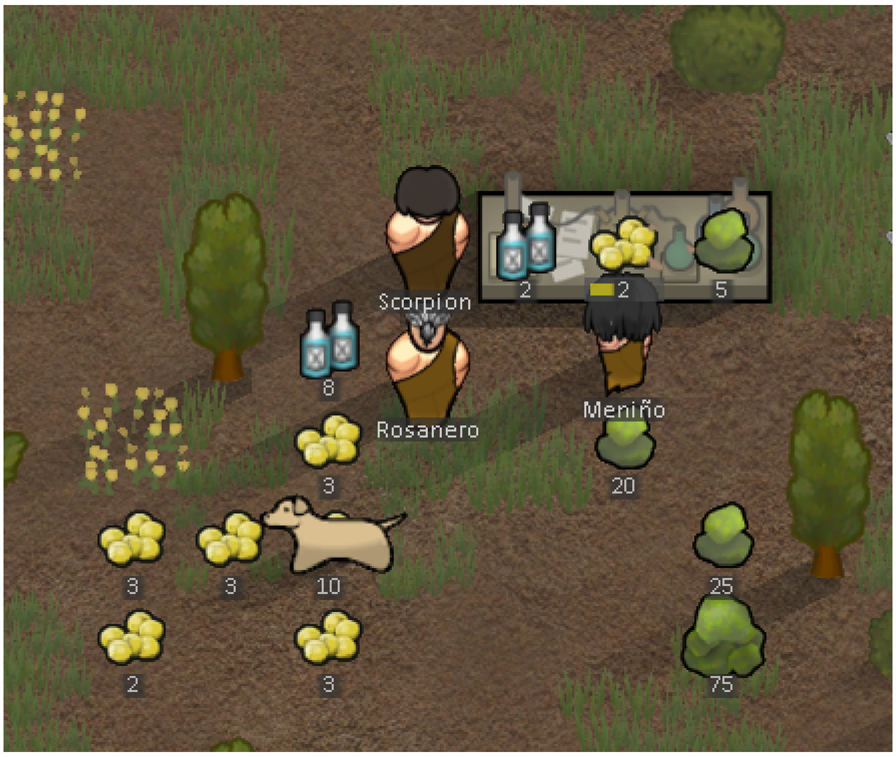
Basics

Description	
Ingredients	2x ambrosia, 2x neutroamine, 5x insect jelly
Products	1x Euphorium
Skill	Intellectual
Work speed governed by	Drug synthesis speed
Work to make	9

Make a Euphorium.

View: Euphorium

Looks like that’s working too! Now let’s try queuing up some Euphorium production:



There they go! (Once they stop eating the insect jelly)

What a feeling! There’s nothing quite like seeing a mod you’ve made come to life! Good stuff. And that’s it, we’ve now got a way to make our Euphorium, we can get our pawns to take it, it has a bunch of effects, all the things you’d expect from a real Rimworld mod!

THE COMPLETED MOD

If you managed to get this far, well done! I’m legitimately proud of you! This stuff is not easy to learn, and you made the effort to learn it!

Everything we’ve just created is available as a .zip file of the final mod here.

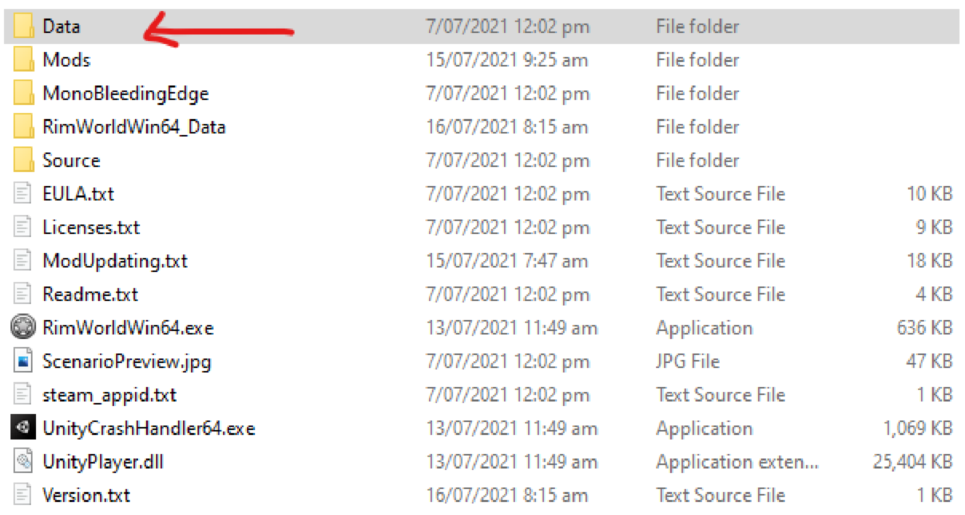
I’ve also uploaded a copy of it to the Steam workshop, where you can go and download it straight into your game.

That’s pretty much it, there’s no more tutorial here. But if you’re keen to keep going, I’ve added a couple of extra sections that will help you explore other def files.

MODDING FROM A BASE GAME ITEM

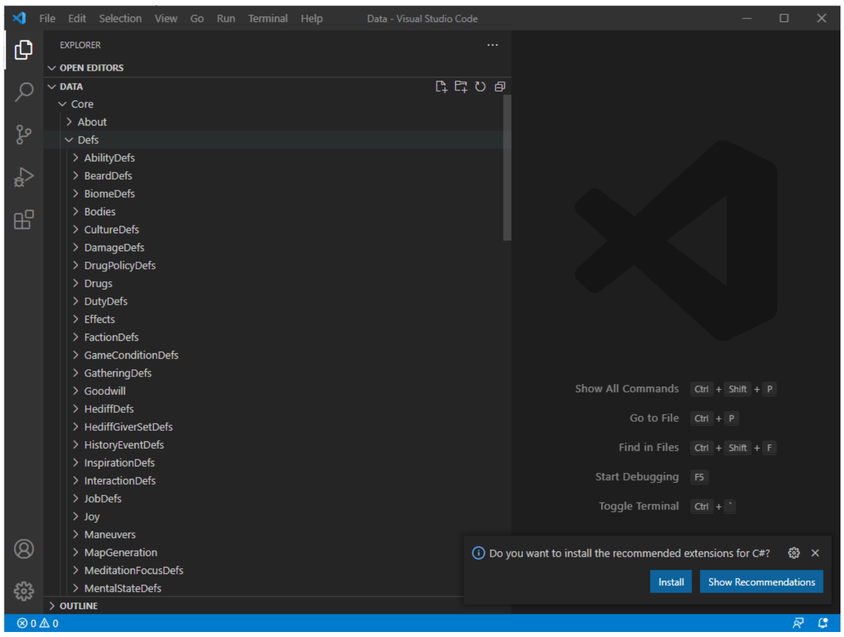
MODDING AN ITEM FROM THE BASE RIMWORLD ITEMS

Now is a really good time to mention that all of the Defs for Rimworlds items are available in XML, right in the base game. They're all located in the Data folder, in Rimworld's installation directory:



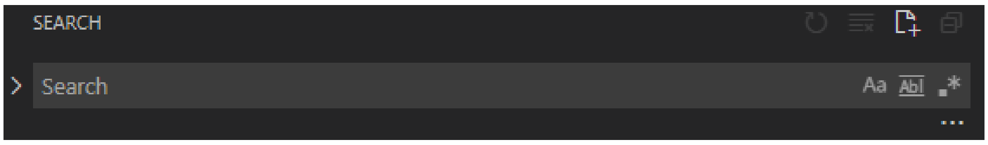
Data	7/07/2021 12:02 pm	File folder	
Mods	15/07/2021 9:25 am	File folder	
MonoBleedingEdge	7/07/2021 12:02 pm	File folder	
RimWorldWin64_Data	16/07/2021 8:15 am	File folder	
Source	7/07/2021 12:02 pm	File folder	
EULA.txt	7/07/2021 12:02 pm	Text Source File	10 KB
Licenses.txt	7/07/2021 12:02 pm	Text Source File	9 KB
ModUpdating.txt	15/07/2021 7:47 am	Text Source File	18 KB
Readme.txt	7/07/2021 12:02 pm	Text Source File	4 KB
RimWorldWin64.exe	13/07/2021 11:49 am	Application	636 KB
ScenarioPreview.jpg	7/07/2021 12:02 pm	JPG File	47 KB
steam_appid.txt	7/07/2021 12:02 pm	Text Source File	1 KB
UnityCrashHandler64.exe	13/07/2021 11:49 am	Application	1,069 KB
UnityPlayer.dll	13/07/2021 11:49 am	Application exten...	25,404 KB
Version.txt	16/07/2021 8:15 am	Text Source File	1 KB

Remember how I said that we were only going to use two features in VSCode, and one of them is "find in files," let's use that now. Right click on the data folder and select "Open in VSCode," this will open a blank VS code window, with a list of files on the left-hand side:

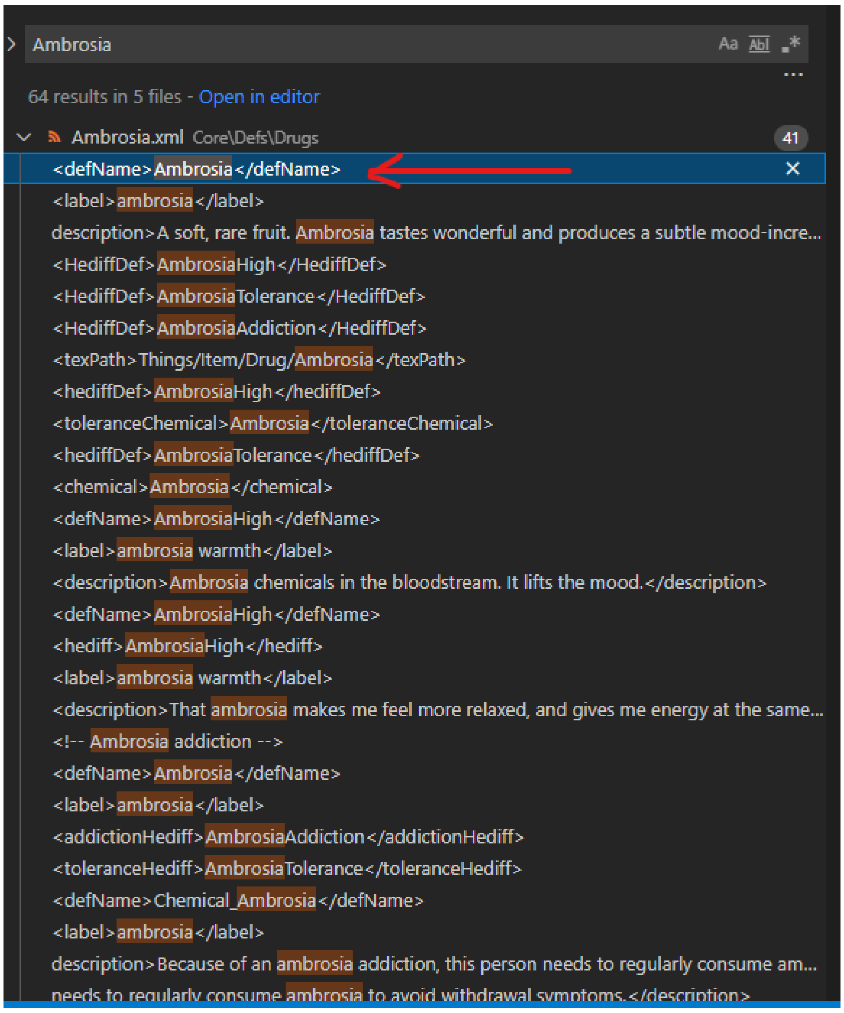


Do you want to install the recommended extensions for C#?

Now press CTRL->SHIFT->F, this is the shortcut key for find in files, and it'll open up a little search box:

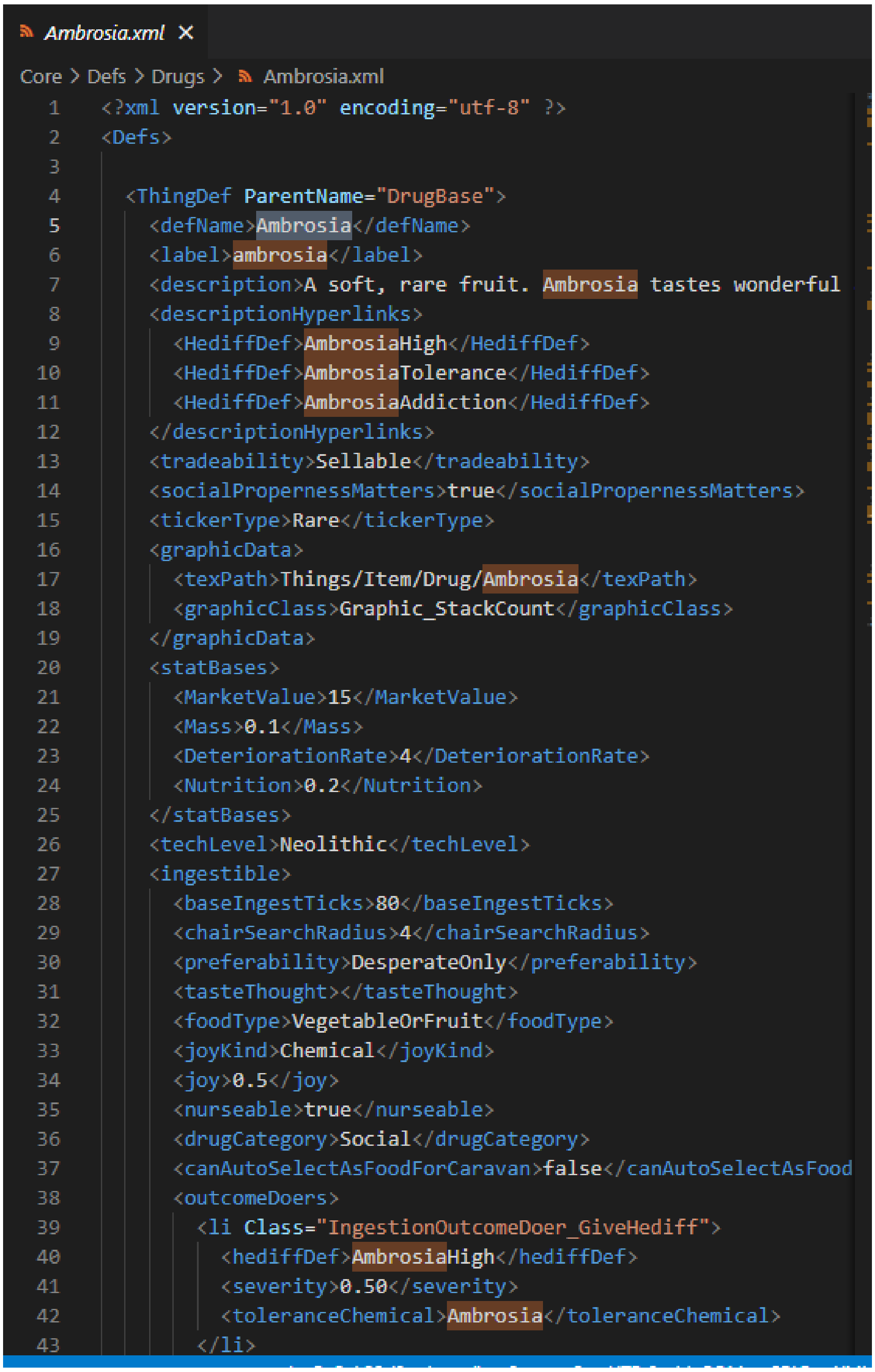


You can search for anything in this box and it will find every reference to that item in Rimworld. For example, we can search for "Ambrosia," and we'll see a bunch of different places that the word "Ambrosia" is used. Let's open up this one here:



```
64 results in 5 files - Open in editor
Ambrosia.xml Core\Defs\Drugs
<defName>Ambrosia</defName>
<label>ambrosia</label>
description>A soft, rare fruit. Ambrosia tastes wonderful and produces a subtle mood-incre...
<HediffDef>AmbrosiaHigh</HediffDef>
<HediffDef>AmbrosiaTolerance</HediffDef>
<HediffDef>AmbrosiaAddiction</HediffDef>
<texPath>Things/Item/Drug/Ambrosia</texPath>
<hediffDef>AmbrosiaHigh</hediffDef>
<toleranceChemical>Ambrosia</toleranceChemical>
<hediffDef>AmbrosiaTolerance</hediffDef>
<chemical>Ambrosia</chemical>
<defName>AmbrosiaHigh</defName>
<label>ambrosia warmth</label>
<description>Ambrosia chemicals in the bloodstream. It lifts the mood.</description>
<defName>AmbrosiaHigh</defName>
<hediff>AmbrosiaHigh</hediff>
<label>ambrosia warmth</label>
<description>That ambrosia makes me feel more relaxed, and gives me energy at the same...
<!-- Ambrosia addiction -->
<defName>Ambrosia</defName>
<label>ambrosia</label>
<addictionHediff>AmbrosiaAddiction</addictionHediff>
<toleranceHediff>AmbrosiaTolerance</toleranceHediff>
<defName>Chemical_Ambrosia</defName>
<label>ambrosia</label>
description>Because of an ambrosia addiction, this person needs to regularly consume am...
needs to regularly consume ambrosia to avoid withdrawal symptoms.</description>
```

Look familiar?



```
<?xml version="1.0" encoding="utf-8" ?>
<Defs>

  <ThingDef ParentName="DrugBase">
    <defName>Ambrosia</defName>
    <label>ambrosia</label>
    <description>A soft, rare fruit. Ambrosia tastes wonderful
    <descriptionHyperlinks>
      <HediffDef>AmbrosiaHigh</HediffDef>
      <HediffDef>AmbrosiaTolerance</HediffDef>
      <HediffDef>AmbrosiaAddiction</HediffDef>
    </descriptionHyperlinks>
    <tradeability>Sellable</tradeability>
    <socialPropernessMatters>true</socialPropernessMatters>
    <tickerType>Rare</tickerType>
    <graphicData>
      <texPath>Things/Item/Drug/Ambrosia</texPath>
      <graphicClass>Graphic_StackCount</graphicClass>
    </graphicData>
    <statBases>
      <MarketValue>15</MarketValue>
      <Mass>0.1</Mass>
      <DeteriorationRate>4</DeteriorationRate>
      <Nutrition>0.2</Nutrition>
    </statBases>
    <techLevel>Neolithic</techLevel>
    <ingestible>
      <baseIngestTicks>80</baseIngestTicks>
      <chairSearchRadius>4</chairSearchRadius>
      <preferability>DesperateOnly</preferability>
      <tasteThought></tasteThought>
      <foodType>VegetableOrFruit</foodType>
      <joyKind>Chemical</joyKind>
      <joy>0.5</joy>
      <nurseable>true</nurseable>
      <drugCategory>Social</drugCategory>
      <canAutoSelectAsFoodForCaravan>false</canAutoSelectAsFood
      <outcomeDoers>
        <li Class="IngestionOutcomeDoer_GiveHediff">
          <hediffDef>AmbrosiaHigh</hediffDef>
          <severity>0.50</severity>
          <toleranceChemical>Ambrosia</toleranceChemical>
        </li>
```

That's right. It's a def for Ambrosia, and it uses a lot of the same tags that our Euphorium drug used. I actually copied a lot of the Ambrosia def and modified it for this tutorial, the sourced tags from a bunch of different drugs and mangled them together. Now you can **do that too!**

So let's say you want to add a new type of plant to the game, one that grows Neutroamine or something. You can search for something like "devilstrand," copy the def for the devilstrand plant into your own mod, change the product it gives you to Neutroamine, and voila, you have infinite Neutroamine.

You can do this for pretty much anything in the game, you want a golden duck that projects a psychic soothe onto your colonists? You can do that. You want a one-punch man style bionic augmentation that can kill in a single blow, copy the power claw and go nuts. You can do **A LOT** this way, and if you can't find what you need, well...

MODDING ON TOP OF SOMEONE ELSE'S MOD

That's right, all of the Defs that other people create are available at your fingertips. Some people have GitHub repositories that they've willingly shared, and you can see exactly how they've made their mods. If you subscribe to a mod on the steam workshop, those mods get downloaded to your local machine too. You can use the same "find in files" trick to search through the Defs that other people have created and see how they've created their mods, they're located in:

[Steam Installation Directory]\steamapps\workshop\content

Don't steal them though. People hate that. **Seriously**.

There are also **frameworks** that people have created for you to work off of. Here's one for creating new races:
https://steamcommunity.com/workshop/filedetails/?id=839005762
Here's one for all of those "Vanilla Expanded" mods you've probably heard of:
https://steamcommunity.com/sharedfiles/filedetails/?id=2023507013

The point is, even if you exhaust everything available to you in the base game, you can still do a lot of cool stuff without touching any code.

ONLY THE BEGINNING

This is just the surface of modding in Rimworld, there's a whole other level of content that you can create when you add coding into the mix, or work off of the other great mods that people have created, but that is way beyond the scope of this tutorial.

If you want a "next step," I'd suggest picking something dirt easy and trying to create a mod from a copied Def. A decorative structure that doesn't have any added functionality would be a good place to start, maybe a new type of bed or chair if you're feeling adventurous. Otherwise, you have my permission to download and use any of my mods you like and start making changes from there. Go wild.

SHAMELESS SELF-PROMOTION / LEGAL WHATEVERS

If you like this guide but hate the form it's in, you have my blessing to turn it into a video, or add a new page to the Rimworld Wiki, or turn it into something else. Please provide attribution (basically don’t steal it) and don’t charge for it (no Udemy courses or anything). This guide is free, and **should always remain free**.

If you do want to support me, either because you got something out of this guide or you want to see more like it, you can get me a KoFi. Don’t feel obligated though. You can also support me by checking out me mods on Steam.

That’s it, that’s the end. Go make some mods.