**Adding a new food to Minecraft**

In survival or hardcore mode, users must manage their hunger to stay alive. In its out-of-the-box behavior, foods are usually provided as part of breaking something or through combining other elements. We’re going to add doughnuts as a new type of food (ItemFood**)** to our Minecraft instance, and provide a command that will let us add doughnuts to our inventory.

**Task 1: Basic Mod structure**

1. We’ll set this up as a separate mod. To do so, start with the DoughnutMod.java file, placed in a java package of com.nextcentury.wic.minecraft.doughnutmod. (Explanation: *packages* are Java’s means of letting you and the world have more than one software file named DoughnutMod. The full package name, when combined with the class name, indicates the specific class. There’s more to packages than that, but that’ll get you started for today.)
2. Our DoughnutMod.java file has a method named init, which takes an FMLInitializationEvent *parameter.* Calling a method on a class says which code to run; the parameters give the method useful information. See our first version of this method, which prints out a message to the console using System.out.println – that means you’ll get to see something in Eclipse when you run your mod. Adjust that message to be something useful to you.
3. Run the modification by clicking Run -> Run Configurations, and choosing the ‘Client’ configuration.

Result: Minecraft runs (start up a single player configuration), and your message shows up in the Eclipse console, looking something like the below.

[com.nextcentury.wic.minecraft.doughnutmod.DoughnutMod:init:16]: <YOUR MESSAGE HERE>

**Task 2: Basic Doughnut Food Item**

1. Things we interact with in Minecraft are either of type Entity, Block, or Item. Entities are generally things that move, including the player. Blocks are the things that lay out the terrain. Items are things that can be found or made in the world.
2. Open the file Doughnut.java and examine it. Key elements:

* Doughnut *extends* ItemFood. This tells Minecraft to treat it as something that can be eaten. ItemFood itself extends Item, which means it can be placed, as well as a number of other behaviors.
* Doughnut has a *constructor*. Java uses constructors to describe how an instance of a class is built. In this case, we use it to fill in some information about how much hunger this ItemFood will satisfy, and give it an ‘unlocalized’ name. (We’ll get to that in a bit…)
* We’ll also tell Forge/Minecraft to put our doughnut item into the tab for food (**CreativeTabs.tabFood**).

1. We have a Doughnut food item – now we need to tell Forge to register it with Minecraft. Within DoughnutMod.java, add the following method and variable declaration.

**public** **static** Doughnut *doughnut*;

@EventHandler

**public** **void** preLoad(FMLPreInitializationEvent event) {

*doughnut* = **new** Doughnut();

GameRegistry.*registerItem*(*doughnut*, "doughnut");

}

1. Now, to test to see whether we can now eat our doughnut… Start up Minecraft again, but this time create a single player world **with cheats enabled**. This is temporarily important: we want to be able to pull the food from the creative tab, but creative mode doesn’t let a user get hungry.
2. Use the following command to set game mode to creative:

**/gamemode creative**

1. Add your doughnut to your inventory by pressing ‘e’. You’ll see your doughnut on the food tab with a default icon and the name ‘item.doughnut.name’. You should only be able to add 5 to any one slot in your inventory, based on on the **MaxStackSize** we set in Doughnut.java.
2. Toggle to survival mode via

**/gamemode survival**

1. You should now be able to eat your doughnut and see your inventory and hunger status bar appropriately react.

**Task 3: The Better Doughnut**

1. First, remember that our doughnut showed up with the name ‘item.doughnut.name’. Let’s give it a better name. Minecraft, like many other systems, doesn’t assume that all of its users will work in English. So it provides something called ‘internationalization’, which basically means you can call it ‘doughnut’ in code, but then let users see it as ‘dona’ in Spanish or ‘โดนัท’ in Thai. We’ll just add the English variant today..
2. Use the src/main/resources/assets/doughnut/lang/en\_US.lang file we’ve provided, update the name to be something you’d like, and save the file (using the path just listed).
3. Next, let’s give the user a reaction when they eat a donut, particularly since our donut doesn’t yet look like a donut…. Within Doughnut.java, add the below, and customize the message to fit your preferences…

@Override

protected void onFoodEaten(ItemStack stack, World worldIn,

EntityPlayer player) {

super.onFoodEaten(stack, worldIn, player);

player.addChatComponentMessage(new ChatComponentText("You ate a doughnut"));

}

1. Eclipse will now show things with redlines under them, saying that it can’t compile the code because it doesn’t know about ItemStack, World, EntityPlayer, or ChatComponentText. Let’s take advantage of what our development environment can do for us, and have it figure out where those classes come from… Press Ctl-Shift-O, and then save the file. At the top of the file, Eclipse will have figured out in which packages those classes come from, and will have added the appropriate import statements.
2. Now run your Minecraft instance, again making use of your ‘cheat’-enabled world to toggle between game modes. You should now see your doughnut show up with *your* chosen name, and it will give the player playing your chosen message when they eat your food. (Maybe a ‘Great donut? How about a tip?’)

**Task 4: Bring me a Doughnut!**

1. Let’s get rid of the gamemode cheat, and create a command which lets us add doughnuts directly to our inventory. We’ll add a new class, BringMeDoughnuts, which we’ll put in the same food package as we did with Doughnut. BringMeDoughnuts implements the *ICommand* interface, which means it has to provide code for all of the methods ICommand declares. (ICommand you to implement these methods!). We’ll give you the guts, but you’re encouraged to experiment.
2. The meat of what it does is in the *execute* method. In our case, we want the user to give us a number of donuts to add to our inventory. If they don’t give us a value, we give them an error. If the value they give us isn’t a number, also a problem. Assuming they do give us a number, we update the player’s inventory to include that number of doughnuts.
3. Now, let’s make Minecraft aware of this new command. To do that, we need to register the command. To our DoughnutMod class, add the following:

**@EventHandler**

**public void registerCommands(FMLServerStartingEvent event) {**

**event.registerServerCommand(new BringMeDoughnuts());**

**}**

We’ve glossed over these @EventHandler methods before. Basically, what they’re doing is “listening” for Minecraft to say certain things have happened. In this case, Minecraft has a step in its startup where it indicates that it’s at a ‘serverstarting’ stage. It sends out notice to code listening (in our case, ours), which can then give additional instructions. In this case, we register our new server command.

1. Restart Minecraft, and test the behavior as you execute the command. If you haven’t changed the code, you can do either of the below to execute the command by its name or its alias… Try variants of the number, etc, to see how our code holds up.

**/giveMeDoughnuts 5**

**/yummy 5**

**Next Steps:**

1. Find the bug in the command’s behavior versus that of using our cheat mode. (Hint: try to eat your doughnut…)
2. Can we give our doughnut a real icon?
3. What about making it appear from a block or a chest in Minecraft?
4. How could we build it from a recipe?
5. How about bundling up our mod to use it in a real server? (Use ‘gradlew build’. Provide some better info in mcmod.info to bundle along.)