Recipes

> This page is still under development. It is not complete, and subject to change at any time.

> The recipe event is a server event.

Contents

- How Recipes & Callbacks Work
- Adding Recipes
  - Shaped
  - Shapeless
  - Smithing
  - Smelting & other Cooking
  - Stonecutting
  - Custom (JSON)
- Removing Recipes
- Modifying & Replacing Recipes
- Helpers, Efficiency and Advanced Ingredients (a.k.a. "how to avoid repeating yourself")

Recipes, Callbacks, and You ↑

The recipe event can be used to add, remove, or replace recipes.

Any script that modifies recipes should be placed in the `kubejs/server_scripts` folder, and can be reloaded at any time with `/reload`.

Any modifications to the recipes should be done within the context of a `recipes` event. This means that we need to register an "event listener" for the `ServerEvents.recipes` event, and give it some code to execute whenever the game is ready to modify recipes. Here's how we tell KubeJS to execute some code whenever it's recipe time:

```
/*
 * ServerEvents.recipes() is a function that accepts another function,
 * called the "callback", as a parameter. The callback gets run when the
 * server is working on recipes, and then we can make our own changes.
 * When the callback runs, it is also known as the event "firing".
*/

ServerEvents.recipes(event => { //listen for the "recipes" server event.
  // You can replace `event` with any name you like, as
  // long as you change it inside the callback too!

  // This part, inside the curly braces, is the callback.
  // You can modify as many recipes as you like in here,
  // without needing to use ServerEvents.recipes() again.

  console.log('Hello! The recipe event has fired!')
})
```

In the next sections, you can see what to put inside your callback.

Adding Recipes ↑

The following is all code that should be placed *inside* your recipe callback.

Shaped ↑

Shaped recipes are added with the `event.shaped()` method. Shaped recipes must have their ingredients in a specific order and shape in order to match the player's input. The arguments to `event.shaped()` are:

1. The output item, which can have a count of 1-64
2. An array (max length 3) of crafting table rows, represented as strings (max length 3). Spaces represent slots with no item, and letters represent items. The letters don't have to mean anything; you explain what they mean in the next argument.
3. An object mapping the letters to Items, like `{letter: item}`. Input items must have a count of 1.

If you want to force strict positions on the crafting grid or disable mirroring, see Methods of Custom Recipes.

```
event.shaped('3x minecraft:stone', [// arg 1: output
    'A B',
    ' C ', // arg 2: the shape (array of strings)
    'B A'
  ], {
    A: 'minecraft:andesite',
    B: 'minecraft:diorite',  //arg 3: the mapping object
    C: 'minecraft:granite'
  }
)
```

Shapeless↑

Shapeless recipes are added with the `event.shapeless()` method. Players can put ingredients of shapeless recipes anywhere on the grid and it will still craft. The arguments to `event.shapeless()` are:

1. The output item
2. An array of input items. The total input items' count must be 9 at most.

```
event.shapeless('3x minecraft:dandelion', [ // arg 1: output
  'minecraft:bone_meal',
  'minecraft:yellow_dye', //arg 2: the array of inputs
  '3x minecraft:ender_pearl'
])
```

Smithing↑

Smithing recipes have 2 inputs and one output and are added with the `event.smithing()` method. Smithing recipes are crafted in the smithing table.

```
event.smithing(
  'minecraft:netherite',  // arg 1: output
  'minecraft:iron_ingot', // arg 2: the item to be upgraded
  'minecraft:black_dye'   // arg 3: the upgrade item
)
```

Smelting & Cooking↑

Cooking recipes are all very similar, accepting one input (a single item) and giving one output (which can be up to 64 of the same item). The fuel cannot be changed in this recipe event and

should be done with tags instead.

- Smelting recipes are added with `event.smelting()`, and require the regular Furnace.
- Blasting recipes are added with `event.blasting()`, and require the Blast Furnace.
- Smoking recipes are added with `event.smoking()`, and require the Smoker.
- Campfire cooking recipes are added with `event.campfireCooking()`, and require the Campfire.

```
// Cook 1 stone into 3 gravel in a Furnace:
event.smelting('3x minecraft:gravel', 'minecraft:stone')

// Blast 1 iron ingot into 10 nuggets in a Blast Furnace:
event.blasting('10x minecraft:iron_nugget', 'minecraft:iron_ingot')

// Smoke glass into tinted glass in the Smoker:
event.smoking('minecraft:tinted_glass', 'minecraft:glass')

// Burn sticks into torches on the Campfire:
event.campfireCooking('minecraft:torch', 'minecraft:stick')
```

Stonecutting↑

Like the cooking recipes, stonecutting recipes are very simple, with one input (a single item) and one output (which can be up to 64 of the same item). They are added with the `event.stonecutting()` method.

```
//allow cutting 3 sticks from any plank on the stonecutter
event.stonecutting('3x minecraft:stick', '#minecraft:planks')
```

Custom/Modded JSON recipes↑

If a mod supports Datapack recipes, you can add recipes for it without any addon mod support! Unfortunately, we can't give specific advice because every mod's layout is different, but if a mod has a GitHub (most do!) or other source code, you can find the relevant JSON files in `/src/generated/resources/data/<modname>/recipes/`. Otherwise, you may be able to find it by unzipping the mod's `.jar` file.

Here's an example of adding a Farmer's Delight cutting board recipe, which defines an input, output, and tool taken straight from their GitHub. Obviously, you can substitute any of the items here to make your own recipe!

```
// Slice cake on a cutting board!
event.custom({
  type: 'farmersdelight:cutting',
  ingredients: [
    { item: 'minecraft:cake' }
  ],
  tool: { tag: 'forge:tools/knives' },
  result: [
    { item: 'farmersdelight:cake_slice', count: 7 }
  ]
})
```

Here's another example of `event.custom()` for making a Tinkers' Construct alloying recipe, which defines inputs, an output, and a temperature straight from their GitHub (conditions removed):

```
// Adding the Molten Electrum alloying recipe from Tinkers' Construct
event.custom({
  type: 'tconstruct:alloy',
  inputs: [
    { tag: 'forge:molten_gold', amount: 90 },
    { tag: 'forge:molten_silver', amount: 90 }
  ],
  result: { fluid: 'tconstruct:molten_electrum', amount: 180 },
  temperature: 760
})
```

Removing Recipes ↑

Removing recipes can be done with the `event.remove()` method. It accepts 1 argument: a Recipe Filter. A filter is a set of properties that determine which recipe(s) to select. There are many conditions with which you can select a recipe:

- by output item `{output: '<item_id>'}`
- by input item(s) `{input: '<item_id>'}`
- by mod `{mod: '<mod_id>'}`
- by the unique recipe ID `{id: '<recipe_id>'}`
- combinations of the above:
  - Require ALL conditions to be met: `{condition1: 'value', condition2: 'value2'}`

- Require ANY of the conditions to be met: `[{condition_a: 'true'}, {condition_b: 'true'}]`
- Require the condition NOT be met: `{not: {condition: 'requirement'}}`

All of the following can go in your recipe callback, as normal:

```
// A blank condition removes all recipes (obviously not recommended):
event.remove({})

// Remove all recipes where output is stone pickaxe:
event.remove({ output: 'minecraft:stone_pickaxe' })

// Remove all recipes where output has the Wool tag:
event.remove({ output: '#minecraft:wool' })

// Remove all recipes where any input has the Redstone Dust tag:
event.remove({ input: '#forge:dusts/redstone' })

// Remove all recipes from Farmer's Delight:
event.remove({ mod: 'farmersdelight' })

// Remove all campfire cooking recipes:
event.remove({ type: 'minecraft:campfire_cooking' })

// Remove all recipes that grant stone EXCEPT smelting recipes:
event.remove({ not: { type: 'minecraft:smelting' }, output: 'stone' })

// Remove recipes that output cooked chicken AND are cooked on a campfire:
event.remove({ output: 'minecraft:cooked_chicken', type: 'minecraft:campfire_cooking' })

// Remove any blasting OR smelting recipes that output minecraft:iron_ingot:
event.remove([{ type: 'minecraft:smelting', output: 'minecraft:iron_ingot' }, { type:
'minecraft:blasting', output: 'minecraft:iron_ingot' }])

// Remove a recipe by ID. in this case, data/minecraft/recipes/glowstone.json:
// Note: Recipe ID and output are different!
event.remove({ id: 'minecraft:glowstone' })
```

To find a recipe's unique ID, turn on advanced tooltips using the `F3`+`H` keys (you will see an announcement in chat), then hover over the [+] symbol (if using REI) or the output (if using JEI).

Modifying & Replacing Recipes ↑

You can bulk-modify supported recipes using `event.replaceInput()` and `event.replaceOutput()`. They each take 3 arguments:

1. A filter, similar to the ones used when removing recipes
2. The ingredient to replace
3. The ingredient to replace it with (can be a tag)

For example, let's say you were removing all sticks and wanted to make people craft things with saplings instead. Inside your callback you would put:

```
event.replaceInput(
  { input: 'minecraft:stick' }, // Arg 1: the filter
  'minecraft:stick',            // Arg 2: the item to replace
  '#minecraft:saplings'         // Arg 3: the item to replace it with
  // Note: tagged fluid ingredients do not work on Fabric, but tagged items do.
)
```

Advanced Techniques ↑

Helper functions ↑

Are you making a lot of similar recipes? Feel like you're typing the same text over and over? Try helper functions! Helper functions will perform repeated actions in much less space by taking in only the parts that are relevant, then doing the repetitive stuff for you!

Here's a helper function, which allows you to make items by crafting a flower pot around the item you specify.

```
ServerEvents.recipes(event => {
  let potting = (output, pottedInput) => {
    event.shaped(output, [
      'BIB',
      ' B '
    ], {
      B: 'minecraft:brick',
      I: pottedInput
```

```
    })
  }

  //Now we can make many 'potting' recipes without typing that huge block over and over!
  potting('kubejs:potted_snowball', 'minecraft:snowball')
  potting('kubejs:potted_lava', 'minecraft:lava_bucket')
  potting('minecraft:blast_furnace', 'minecraft:furnace')
})
```

Looping ↑

In addition to helper functions, you can also loop through an array to perform an action on every item in the array.

Revision #15
Created 2022-12-06 18:05:12 UTC by Nat
Updated 2023-11-08 19:45:42 UTC by Lexxie