KubeJS Create

Download: CurseForge, Modrinth

> The example scripts are only here to demonstrate the recipes. They are not meant to be used with the items shown.

Compacting

Syntax: `compacting(output[], input[])`

Features:

- supports multiple inputs and outputs
- supports `.heated()` and `.superheated()`
- can have a fluid output as long as it has another item output
- supports chance-based output
- uses the **Mechanical Press**, **Basin**, and optionally a **Blaze Burner**

```
ServerEvents.recipes(e => {
  e.recipes.create.compacting('diamond', 'coal_block')
  e.recipes.create.compacting('diamond', 'coal_block').heated()
  e.recipes.create.compacting('diamond', 'coal_block').superheated()
  e.recipes.create.compacting([Fluid.water(10), 'dead_bush'], ['#minecraft:saplings',
'#minecraft:saplings'])
  e.recipes.create.compacting(['diamond', Item.of('diamond').withChance(0.3)], 'coal_block')
})
```

Crushing

Syntax: `crushing(output[], input)`

Features:

- supports multiple chance-based outputs
- supports `.processingTime()`
- uses the **Crushing Wheels**

```
ServerEvents.recipes(e => {
  e.recipes.create.crushing('diamond', 'coal_block')
  e.recipes.create.crushing('diamond', 'coal_block').processingTime(500)
  e.recipes.create.crushing(['diamond', Item.of('diamond').withChance(0.5)], 'coal_block')
})
```

Cutting

Syntax: `cutting(output[], input)`

Features:

- supports multiple chance-based outputs
- supports `.processingTime()`
- uses the **Mechanical Saw**

```
ServerEvents.recipes(e => {
  e.recipes.create.cutting('diamond', 'coal_block')
  e.recipes.create.cutting('diamond', 'coal_block').processingTime(500)
  e.recipes.create.cutting(['diamond', Item.of('diamond').withChance(0.5)], 'coal_block')
})
```

Deploying

Syntax: `deploying(output[], input[])`

Features:

- supports multiple chance-based outputs
- requires exactly two inputs, the second input is what the Deployer is holding
- supports `.keepHeldItem()`
- uses the **Deployer**

```
ServerEvents.recipes(e => {
  e.recipes.create.deploying('diamond', ['coal_block', 'sand'])
  e.recipes.create.deploying(['diamond', 'emerald'], ['coal_block', 'sand']).keepHeldItem()
  e.recipes.create.deploying(['diamond', Item.of('diamond').withChance(0.5)], ['coal_block',
'sand'])
})
```

Emptying

Syntax: `emptying(output[], input)`

Features:

- requires one input and two outputs, the outputs must be an item and a fluid
- uses the **Item Drain**

```
ServerEvents.recipes(e => {
  e.recipes.create.emptying([Fluid.water(), 'bucket'], 'water_bucket')
})
```

Filling

Syntax: `filling(output, input[])`

Features:

- requires two inputs and one output, the inputs must be an item and a fluid
- uses the **Spout**

```
ServerEvents.recipes(e => {
  e.recipes.create.filling('water_bucket', [Fluid.water(), 'bucket'])
})
```

Haunting

Syntax: `haunting(output[], input)`

Features:

- supports multiple chance-based outputs
- uses the **Encased Fan** and **Soul Fire**

```
ServerEvents.recipes(e => {
  e.recipes.create.haunting('soul_campfire', 'campfire')
  e.recipes.create.haunting(['wheat', 'oak_sapling'], 'potato')
  e.recipes.create.haunting(['wheat', Item.of('oak_sapling').withChance(0.2)], 'potato')
})
```

Mechanical Crafting

Syntax: `mechanical_crafting(output, pattern[], keys{})`

Features:

- mostly identical to the default Shaped Crafting
- supports up to 9x9 grid size
- uses the **Mechanical Crafter**

```
ServerEvents.recipes(e => {
  e.recipes.create.mechanical_crafting('emerald', [
    ' DDD ',
    'D   D',
    'D   D',
    'D   D',
    ' DDD '
  ], {
    D: 'dirt'
  })
})
```

Milling

Syntax: `milling(output[], input)`

Features:

- supports multiple chance-based outputs
- uses the **Millstone**

```
ServerEvents.recipes(e => {
  e.recipes.create.milling('diamond', 'coal_block')
  e.recipes.create.milling(['diamond', 'emerald'], 'coal_block')
  e.recipes.create.milling(['diamond', Item.of('diamond').withChance(0.5)], 'coal_block')
})
```

Mixing

Syntax: `mixing(output[], input)`

Features:

- supports multiple chance-based outputs
- supports fluid inputs and outputs
- supports `.heated()` and `.superheated()`

- uses the **Mechanical Mixer, Basin**, and optionally a **Blaze Burner**

```
ServerEvents.recipes(e => {
  e.recipes.create.mixing('diamond', 'coal_block')
  e.recipes.create.mixing('diamond', 'coal_block').heated()
  e.recipes.create.mixing('diamond', 'coal_block').superheated()
  e.recipes.create.mixing([Fluid.water(10), 'dead_bush'], ['#minecraft:saplings',
'#minecraft:saplings'])
  e.recipes.create.mixing(['diamond', Item.of('diamond').withChance(0.3)], 'coal_block')
})
```

Pressing

Syntax: `pressing(output[], input)`

Features:

- supports multiple chance-based outputs
- uses the **Mechanical Press**

```
ServerEvents.recipes(e => {
  e.recipes.create.pressing('diamond', 'coal_block')
  e.recipes.create.pressing(['diamond', 'emerald'], 'coal_block')
  e.recipes.create.pressing(['diamond', Item.of('diamond').withChance(0.5)], 'coal_block')
})
```

Sandpaper Polishing

Syntax: `sandpaper_polishing(output, input)`

Features:

- supports chance-based output
- uses any item tagged with `create:sandpaper`

```
ServerEvents.recipes(e => {
  e.recipes.create.sandpaper_polishing('diamond', 'coal_block')
  e.recipes.create.sandpaper_polishing(Item.of('diamond').withChance(0.5), 'coal_block')
})
```

Sequenced Assembly

Syntax: `sequenced_assembly(output[], input, sequence[]).transitionalItem(item).loops(int)`

`Output` is an item or an array of items. If it is an array:

- The first item is the real output, the remainder is scrap.
- Only one item is chosen, with an equal chance of each.
- You can use `Item.of('create:shaft').withChance(2)` to double the chance of that item being chosen.

`Transitional Item` is any item used during the intermediate stages of the assembly.

`Sequence` is an array of recipes of the following types:

- `create:cutting`
- `create:pressing`
- `create:deploying`
- `create:filling`

The transitional item needs to be the input ***and*** output of each of these recipes.

`Loops` is the number of times that the recipe repeats. Calling `.loops()` is optional and defaults to **4**.

```
ServerEvents.recipes(e => {
□e.recipes.create.sequenced_assembly([
□□Item.of('create:precision_mechanism').withChance(130.0), // this is the item that will appear
in JEI as the result
□□Item.of('create:golden_sheet').withChance(8.0), // the rest of these items will be part of th
scrap
□□Item.of('create:andesite_alloy').withChance(8.0),
□□Item.of('create:cogwheel').withChance(5.0),
□□Item.of('create:shaft').withChance(2.0),
□□Item.of('create:crushed_gold_ore').withChance(2.0),
□□Item.of('2x gold_nugget').withChance(2.0),
□□'iron_ingot',
□□'clock'
□], 'create:golden_sheet', [ // 'create:golden_sheet' is the input
□□// the transitional item set by `transitionalItem('create:incomplete_large_cogwheel')` is the
item used during the intermediate stages of the assembly
□□e.recipes.createDeploying('create:incomplete_precision_mechanism',
['create:incomplete_precision_mechanism', 'create:cogwheel']),
□□// like a normal recipe function, is used as a sequence step in this array. Input and output
have the transitional item
```

```
  e.recipes.createDeploying('create:incomplete_precision_mechanism',
  ['create:incomplete_precision_mechanism', 'create:large_cogwheel']),
  e.recipes.createDeploying('create:incomplete_precision_mechanism',
  ['create:incomplete_precision_mechanism', 'create:iron_nugget'])
 ]).transitionalItem('create:incomplete_precision_mechanism').loops(5) // set the transitional
item and the number of loops

 // for this code to work, kubejs:incomplete_spore_blossom needs to be added to the game
 let inter = 'kubejs:incomplete_spore_blossom' // making a variable to store the transitional
item makes the code more readable
 e.recipes.create.sequenced_assembly([
  Item.of('spore_blossom').withChance(16.0), // this is the item that will appear in JEI as the
result
  Item.of('flowering_azalea_leaves').withChance(16.0), // the rest of these items will be part
of the scrap
  Item.of('azalea_leaves').withChance(2.0),
  'oak_leaves',
  'spruce_leaves',
  'birch_leaves',
  'jungle_leaves',
  'acacia_leaves',
  'dark_oak_leaves'
 ], 'flowering_azalea_leaves', [ // 'flowering_azalea_leaves' is the input
  // the transitional item is a variable, that is 'kubejs:incomplete_spore_blossom' and is used
during the intermediate stages of the assembly
  e.recipes.createPressing(inter, inter),
  // like a normal recipe function, is used as a sequence step in this array. Input and output
have the transitional item
  e.recipes.createDeploying(inter, [inter, 'minecraft:hanging_roots']),
  e.recipes.createFilling(inter, [inter, Fluid.water(420)]),
  e.recipes.createDeploying(inter, [inter, 'minecraft:moss_carpet']),
  e.recipes.createCutting(inter, inter)
 ]).transitionalItem(inter).loops(2) // set the transitional item and the number of loops
})
```

Transitional Items

As mentioned earlier, any item can be a transition item. However, this is not completely recommended.

If you wish to make your own transitional item, it's best if you make the type `create:sequenced_assembly`.

```
StartupEvents.registry('item', e => {
  e.create('incomplete_spore_blossom', 'create:sequenced_assembly')
})
```

Splashing/Washing

Syntax: `splashing(output[], input)`

Features:

- supports multiple chance-based outputs
- uses the **Encased Fan** and **Water**

```
ServerEvents.recipes(e => {
  e.recipes.create.splashing('soul_campfire', 'campfire')
  e.recipes.create.splashing(['wheat', 'oak_sapling'], 'potato')
  e.recipes.create.splashing(['wheat', Item.of('oak_sapling').withChance(0.2)], 'potato')
})
```

Mysterious Conversion

> Mysterious Conversion recipes are client-side only, so the only way to add them currently is using reflection.

> Goes inside `client_scripts` and ***not*** in an event.

```
//reference the classes used for the recipe
let MysteriousItemConversionCategory =
Java.loadClass('com.simibubi.create.compat.jei.category.MysteriousItemConversionCategory')
let ConversionRecipe = Java.loadClass('com.simibubi.create.compat.jei.ConversionRecipe')

//add the recipes manually
MysteriousItemConversionCategory.RECIPES.add(ConversionRecipe.create('apple', 'carrot'))
MysteriousItemConversionCategory.RECIPES.add(ConversionRecipe.create('golden_apple',
'golden_carrot'))
```

Preventing Recipe Auto-Generation

If you don't want smelting, blasting, smoking, crafting, or stonecutting to get an auto-generated counterpart, then include `manual_only` at the end of the recipe id:

```
ServerEvents.recipes(e => {
e.shapeless('wet_sponge', ['water_bucket',
'sponge']).id('kubejs:moisting_the_sponge_manual_only')
})
```

Other types of prevention, can be done in the create config (the goggles button leads you there).

If it is not in the config, then you can not change it.

Revision #10
Created 2023-01-27 21:22:54 UTC by Lat
Updated 2023-08-31 21:13:33 UTC by Nat