KubeJS

This wiki is for KubeJS 6+, for Minecraft versions 1.19.2+. This wiki still is incomplete, so you may fallback to 1.18.2 one, see KubeJS Legacy page.

Intro

FAQ

What does this mod do?

This mod lets you create scripts in JavaScript language to manage your server, add new blocks and items, change recipes, add custom handlers for quest mods and more!

How to use it?

Run the game with mod installed once. It should generate `kubejs` folder in your minecraft directory with example scripts and README.txt. Read that!

Here's a video tutorial for 1.19.2:

https://www.youtube.com/embed/xhJJbNJjics

I don't know JavaScript

There's examples and pre-made scripts here. And you can always ask in discord support channel for help with scripts, but be specific.

Can I reload scripts?

Yes, use `/reload` to reload `server_scripts/`, `F3 + T` to reload `client_scripts/` and `/kubejs reload startup_scripts` to reload `startup_scripts/`. If you don't care about reloading recipes but are testing some world interaction event, you can run `/kubejs reload server_scripts`. Note: Not everything is reloadable. Some things require you to restart game, some only world, some work on fly. Reloading startup scripts is not recommended, but if you only have event listeners, it shouldn't be a problem.

What mod recipes does it support / is mod X supported?

If the mod uses datapack recipes, then it's supported by default. Some more complicated mods require addon mods, but in theory, still would work with datapack recipes. See Recipes and Addons sections for more info.

What features does this mod have?

See list of all Features.

How does this mod work?

It uses a fork of Rhino, a JavaScript engine by Mozilla to convert JS code to Java classes at runtime. KubeJS wraps minecraft classes and adds utilities to simplify that a lot and remove need for mappings. Architectury lets nearly the same source code be compiled for both Forge and Fabric making porting extremely easy.

Ok, but what if it.. doesn't work?

You can report issues here.

I have more questions/suggestions!

If wiki didn't have the answer for what you were looking for, you can join the Discord server and ask for help on #support channel!

Website: https://kubejs.com/

Source and issue tracker: https://github.com/KubeJS-Mods/KubeJS

Download: https://www.curseforge.com/minecraft/mc-mods/kubejs

> Anything below 1.18 is no longer supported!

Features

Here's a list of all documented (and sometimes undocumented) features that I can remember:

- [Full list of events]
- Editing Recipes
- Editing Tags
- Adding New Items
- Adding New Blocks
- Default Options
- Changing Window Title and Icon
- Changing Mod Display Names
- Loading Assets and Data
- Modify Items
- Modify Blocks
- Adding Fluids
- Worldgen
- Chat event
- Block placement event
- Item right-click event
- And a bunch more that I forgot...

Global

Primitive prototype additions

- `String#namespace` : String - namespace part of namespaced string, aka of "minecraft:oak_planks" it's "minecraft". Defaults to "minecraft" if there's no `:` .
- `String#path` : String - path part of namespaced string, aka of "minecraft:oak_planks" it's "oak_planks"

Constants

- `SECOND` : Number = 1000
- `MINUTE` : Number = 60000 (60 * SECOND)
- `HOUR` : Number = 3600000 (60 * MINUTE)

Objects

- `global` : Map<String, Object>
- `console` : Console

Classes

- `Platform`
- `ResourceLocation`
- `Utils`
- `Java`
- `Text`
- `UUID`
- `JsonIO`
- `Block`
- `Item`
- `Ingredient`
- `IngredientHelper`
- `NBT`
- `NBTIO`
- `Direction`
- `Facing`
- `AABB`
- `Fluid`
- `Color`

- BlockStatePredicate

Wrapped Classes

Name	Class
JavaMath	java.lang.Math
Blocks	net.minecraft.world.level.block.Blocks
Items	net.minecraft.world.item.Items
Stats	net.minecraft.stats.Stats
DecorationGenerationStep	net.minecraft.world.level.levelgen.GenerationStep.Decoration
CarvingGenerationStep	net.minecraft.world.level.levelgen.GenerationStep.Carving
BlockPos	net.minecraft.core.BlockPos
DamageSource	net.minecraft.world.damagesource.DamageSource
BlockProperties	net.minecraft.world.level.block.state.properties.BlockState Properties
Vec3 , Vec3d	net.minecraft.world.phys.Vec3
Vec3i	net.minecraft.core.Vec3i

Events

List of Events

This is a list of all events. It's possible that not all events are listed here, but this list will be updated regularly.

Click on event ID to open its class and see information, fields, and methods.

Type descriptions:

- **Startup**: scripts go into the `/kubejs/startup_scripts/` folder. Startup scripts run once, at startup, on both the client and server. Most events that require registering or modifying something at game start (like custom blocks, items, and fluids) will be Startup events.
- **Server**: scripts go into the `/kubejs/server_scripts/` folder. It will be reloaded when you run `/reload` command. Server events are always accessible, even in single-player worlds. Most events that make changes to the world while the game is running (such as breaking blocks, teleporting players, or adding recipes) will go here.
- **Server Startup**: same as Server, and the event will be fired at least once when the server loads.
- **Client**: scripts go into the `/kubejs/client_scripts/` folder. Will be reloaded when you press `F3+T`. Most changes that are per-client (such as resource packs, Painter, and JEI) are client events.
- **Client Startup**: Same as Client, and the event will be fired at least once when the client loads.

Base KubeJS Events

Folder	Method	Cancellable
`/startup_scripts/`	`StartupEvents.init` (link)	⬜
`/startup_scripts/`	`StartupEvents.postInit` (link)	⬜

Folder	Method	Cancellable
/startup_scripts/	`StartupEvents.registry` (fluid) `StartupEvents.registry` ([block](#)) `StartupEvents.registry` ([item](#)) `StartupEvents.registry` (enchantment) `StartupEvents.registry` (mob effects) `StartupEvents.registry` (sound event) `StartupEvents.registry` (block entity type) `StartupEvents.registry` (potion) `StartupEvents.registry` (particle type) `StartupEvents.registry` (painting variant) `StartupEvents.registry` (custom stat) `StartupEvents.registry` (point of interest type) `StartupEvents.registry` (villager type) `StartupEvents.registry` (villager profession)	☐
/client_scripts/	`ClientEvents.highPriorityAssets` (link)	☐
/client_scripts/	`ClientEvents.init` (link)	☐
/client_scripts/	`ClientEvents.loggedIn` (link)	☐
/client_scripts/	`ClientEvents.loggedOut` (link)	☐
/client_scripts/	`ClientEvents.tick` (link)	☐
/client_scripts/	`ClientEvents.painterUpdated` (link)	☐
/client_scripts/	`ClientEvents.leftDebugInfo` (link)	☐
/client_scripts/	`ClientEvents.rightDebugInfo` (link)	☐
/client_scripts/	`ClientEvents.paintScreen` (link)	☐
/server_scripts/	`ServerEvents.lowPriorityData` (link)	☐
/server_scripts/	`ServerEvents.highPriorityData` (link)	☐
/server_scripts/	`ServerEvents.loaded` (link)	☐

Folder	Method	Cancellable
/server_scripts/	ServerEvents.unloaded (link)	☐
/server_scripts/	ServerEvents.tick (link)	☐
/server_scripts/	ServerEvents.tags (link)	☐
/server_scripts/	ServerEvents.commandRegistry (link)	☐
/server_scripts/	ServerEvents.command (link)	☐
/server_scripts/	ServerEvents.customCommand (link)	☐
/server_scripts/	ServerEvents.recipes (link)	☐
/server_scripts/	ServerEvents.afterRecipes (link)	☐
/server_scripts/	ServerEvents.specialRecipeSerializers (link)	☐
/server_scripts/	ServerEvents.compostableRecipes (link)	☐
/server_scripts/	ServerEvents.recipeTypeRegistry (link)	☐
/server_scripts/	ServerEvents.genericLootTables (link)	☐
/server_scripts/	ServerEvents.blockLootTables (link)	☐
/server_scripts/	ServerEvents.entityLootTables (link)	☐
/server_scripts/	ServerEvents.giftLootTables (link)	☐
/server_scripts/	ServerEvents.fishingLootTables (link)	☐
/server_scripts/	ServerEvents.chestLootTables (link)	☐
/server_scripts/	LevelEvents.loaded (link)	☐
/server_scripts/	LevelEvents.unloaded (link)	☐
/server_scripts/	LevelEvents.tick (link)	☐

Folder	Method	Cancellable
/server_scripts/	LevelEvents.beforeExplosion (link)	☐
/server_scripts/	LevelEvents.afterExplosion (link)	☐
/startup_scripts/	WorldgenEvents.add (link)	☐
/startup_scripts/	WorldgenEvents.remove (link)	☐
/client_scripts/	NetworkEvents.fromServer (link)	☐
/server_scripts/	NetworkEvents.fromClient (link)	☐
/startup_scripts/	ItemEvents.modification (link)	☐
/startup_scripts/	ItemEvents.toolTierRegistry (link)	☐
/startup_scripts/	ItemEvents.armorTierRegistry (link)	☐
/server_scripts/	ItemEvents.rightClicked (link)	☐
/server_scripts/	ItemEvents.canPickUp (link)	☐
/server_scripts/	ItemEvents.pickedUp (link)	☐
/server_scripts/	ItemEvents.dropped (link)	☐
/server_scripts/	ItemEvents.entityInteracted (link)	☐
/server_scripts/	ItemEvents.crafted (link)	☐
/server_scripts/	ItemEvents.smelted (link)	☐
/server_scripts/	ItemEvents.foodEaten (link)	☐
/client_scripts/	ItemEvents.tooltip (link)	☐
/startup_scripts/	ItemEvents.modelProperties (link)	☐
/client_scripts/	ItemEvents.clientRightClicked (link)	☐
/client_scripts/	ItemEvents.clientLeftClicked (link)	☐

Folder	Method	Cancellable
`/server_scripts/`	`ItemEvents.firstRightClicked` (link)	☐
`/server_scripts/`	`ItemEvents.firstLeftClicked` (link)	☐
`/startup_scripts/`	`BlockEvents.modification` (link)	☐
`/server_scripts/`	`BlockEvents.rightClicked` (link)	☐
`/server_scripts/`	`BlockEvents.leftClicked` (link)	☐
`/server_scripts/`	`BlockEvents.placed` (link)	☐
`/server_scripts/`	`BlockEvents.broken` (link)	☐
`/server_scripts/`	`BlockEvents.detectorChanged` (link)	☐
`/server_scripts/`	`BlockEvents.detectorPowered` (link)	☐
`/server_scripts/`	`BlockEvents.detectorUnpowered` (link)	☐
`/server_scripts/`	`BlockEvents.farmlandTrampled` (link)	☐
`/server_scripts/`	`EntityEvents.death` (link)	☐
`/server_scripts/`	`EntityEvents.hurt` (link)	☐
`/server_scripts/`	`EntityEvents.checkSpawn` (link)	☐
`/server_scripts/`	`EntityEvents.spawned` (link)	☐
`/server_scripts/`	`PlayerEvents.loggedIn` (link)	☐
`/server_scripts/`	`PlayerEvents.loggedOut` (link)	☐
`/server_scripts/`	`PlayerEvents.respawned` (link)	☐
`/server_scripts/`	`PlayerEvents.tick` (link)	☐
`/server_scripts/`	`PlayerEvents.chat` (link)	☐
`/server_scripts/`	`PlayerEvents.decorateChat` (link)	☐

Folder	Method	Cancellable
/server_scripts/	PlayerEvents.advancement (link)	☐
/server_scripts/	PlayerEvents.inventoryOpened (link)	☐
/server_scripts/	PlayerEvents.inventoryClosed (link)	☐
/server_scripts/	PlayerEvents.inventoryChanged (link)	☐
/server_scripts/	PlayerEvents.chestOpened (link)	☐
/server_scripts/	PlayerEvents.chestClosed (link)	☐

Mod Compatibility

These events are enabled when certain other mods are present.

Just Enough Items (JEI)

Folder	Method	Cancellable
/client_scripts/	JEIEvents.subtypes (source)	☐
/client_scripts/	JEIEvents.hideItems (source)	☐
/client_scripts/	JEIEvents.hideFluids (source)	☐
/client_scripts/	JEIEvents.hideCustom (source)	☐
/client_scripts/	JEIEvents.removeCategories (source)	☐
/client_scripts/	JEIEvents.removeRecipes (source)	☐
/client_scripts/	JEIEvents.addItems (source)	☐
/client_scripts/	JEIEvents.addFluids (source)	☐
/client_scripts/	JEIEvents.information (source)	☐

Roughly Enough Items (REI)

Folder	Method	Cancellable
/client_scripts/	REIEvents.hide (source)	☐
/client_scripts/	REIEvents.add (source)	☐
/client_scripts/	REIEvents.information (source)	☐
/client_scripts/	REIEvents.removeCategories (source)	☐
/client_scripts/	REIEvents.groupEntries (source)	☐

GameStages

Folder	Method	Cancellable
/server_scripts/	GameStageEvents.stageAdded (source)	☐
/server_scripts/	GameStageEvents.stageRemoved (source)	☐

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

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.

Tags

> The tag event is a server event.

> Tags are per item/block/fluid/entity_type and as such cannot be added based on things like NBT data!

The tags event takes an extra parameter that determines which registry it's adding tags to. You will generally only need item, block, and fluid tags. However, it does support adding tags to any registry, including other mods ones. For mod ones make sure to include the namespace!

```
// Listen to item tag event
ServerEvents.tags('item', event => {
  // Get the #forge:cobblestone tag collection and add Diamond Ore to it
  event.add('forge:cobblestone', 'minecraft:diamond_ore')

  // Get the #forge:cobblestone tag collection and remove Mossy Cobblestone from it
  event.remove('forge:cobblestone', 'minecraft:mossy_cobblestone')

  // Get #forge:ingots/copper tag and remove all entries from it
  event.removeAll('forge:ingots/copper')

  // Required for FTB Quests to check item NBT
  event.add('itemfilters:check_nbt', 'some_item:that_has_nbt_types')

  // You can create new tags the same way you add to existing, just give it a name
  event.add('forge:completely_new_tag', 'minecraft:clay_ball')

  // It supports adding tags to tags as well. Just prefix the second tag with #
  event.add('c:stones', '#forge:stone')

  // Removes all tags from this entry
  event.removeAllTagsFrom('minecraft:stick')

  // Add all items from the forge:stone tag to the c:stone tag, unless the id contains diorite
```

```
  const stones = event.get('forge:stone').getObjectIds()
  const blacklist = Ingredient.of(/.*diorite.*/)
  stones.forEach(stone => {
    if (!blacklist.test(stone)) event.add('c:stone', stone)
  })
})
```

> Recipes use item tags, not block or fluid tags. Even if items representing those are blocks, like `minecraft:cobblestone`, it still uses an item tag for recipes.

```
// Listen to the block tag event
ServerEvents.tags('block', event => {
  // Add tall grass to the climbable tag. This does make it climbable!
  event.add('minecraft:climbable', 'minecraft:tall_grass')
})
```

Custom Items

> The custom item event is a startup event.

Custom items are created in a startup script. They cannot be reloaded without restarting the game. The event is not cancellable.

```
// Listen to item registry event
StartupEvents.registry('item', e => {
  // The texture for this item has to be placed in kubejs/assets/kubejs/textures/item/test_item.png
  // If you want a custom item model, you can create one in Blockbench and put it in
kubejs/assets/kubejs/models/item/test_item.json
  e.create('test_item')

  // If you want to specify a different texture location you can do that too, like this:
  e.create('test_item_1').texture('mobbo:item/lava') // This texture would be located at
kubejs/assets/mobbo/textures/item/lava.png

  // You can chain builder methods as much as you like
  e.create('test_item_2').maxStackSize(16).glow(true)

  // You can specify item type as 2nd argument in create(), some types have different available methods
  e.create('custom_sword', 'sword').tier('diamond').attackDamageBaseline(10.0)
})
```

Valid item types:

- basic (this is the default)
- sword
- pickaxe
- axe
- shovel
- shears
- hoe

- helmet
- chestplate
- leggings
- boots

Other methods item builder supports: [you can chain these methods after create()]

- `maxStackSize(size)`
- `displayName(name)`
- `unstackable()`
- `maxDamage(damage)` This is the item's durability, not actual weapon damage.
- `burnTime(ticks)`
- `containerItem(item_id)`
- `rarity('rarity')`
- `tool(type, level)`
- `glow(true/false)`
- `tooltip(text...)`
- `group('group_id')`
- `color(index, colorHex)`
- `texture(customTextureLocation)`
- `parentModel(customParentModelLocation)`
- `food(foodBuilder => ...)` For full syntax see below

Methods available if you use a tool type (`'sword'`, `'pickaxe'`, `'axe'`, `'shovel'` or `'hoe'`):

- `tier('toolTier')`
- `modifyTier(tier => ...)` Same syntax as custom tool tier, see [Custom Tiers](#)
- `attackDamageBaseline(damage)` You only want to modify this if you are creating a custom weapon such as Spear, Battleaxe, etc.
- `attackDamageBonus(damage)`
- `speedBaseline(speed)` Same as attackDamageBaseline, only modify for custom weapon types
- `speed(speed)`

Default available tool tiers:

- wood
- stone
- iron
- gold
- diamond
- netherite

Methods available if you use an armour type ('helmet', 'chestplate', 'leggings' or 'boots'):
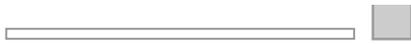
- `tier('armorTier')`
- `modifyTier(tier => ...)` Same syntax as custom armor tier, see [Custom Tiers](#)

Default available armor tiers:

- leather
- chainmail
- iron
- gold
- diamond
- turtle
- netherite

Vanilla group/creative tab IDs:

- search
- buildingBlocks
- decorations
- redstone
- transportation
- misc
- food
- tools
- combat
- brewing

Custom Foods

```
StartupEvents.registry('item', event => {
  event.create('magic_steak').food(food => {
    food
      .hunger(6)
      .saturation(6)//This value does not directly translate to saturation points gained
      //The real value can be assumed to be:
      //min(hunger * saturation * 2 + saturation, foodAmountAfterEating)
      .effect('speed', 600, 0, 1)
```

```
      .removeEffect('poison')
      .alwaysEdible()//Like golden apples
      .fastToEat()//Like dried kelp
      .meat()//Dogs are willing to eat it
      .eaten(ctx => {//runs code upon consumption
        ctx.player.tell(Text.gold('Yummy Yummy!'))
          //If you want to modify this code then you need to restart the game.
          //However, if you make this code call a global startup function
          //and place the function *outside* of an event handler
          //then you may use the command:
          //  /kubejs reload startup_scripts
          //to reload the function instantly.
          //See example below
      })
  })

  event.create('magicer_steak').unstackable().food(food => {
    food
      .hunger(7)
      .saturation(7)
      // This references the function below instead of having code directly, so it is reloadable!
      .eaten(ctx => global.myAwesomeReloadableFunction(ctx))
  })
})

global.myAwesomeReloadableFunction = ctx => {
  ctx.player.tell('Hello there!')
  ctx.player.tell(Text.of('Change me then reload with ').append(Text.red('/kubejs reload
startup_scripts')).append(' to see your changes!'))
}
```

Item modification

> Item modification is a startup event.

`ItemEvents.modification` is a startup script event used to modify various properties of existing items.

```
ItemEvents.modification(event => {
  event.modify('minecraft:ender_pearl', item => {
    item.maxStackSize = 64
    item.fireResistant = true
    item.rarity = "UNCOMMON"
  })
  event.modify('minecraft:ancient_debris', item => {
    item.rarity = "RARE"
    item.burnTime = 16000
  })
  event.modify('minecraft:turtle_helmet', item => {
    item.rarity = "EPIC"
    item.maxDamage = 481
    item.craftingRemainder = Item.of('minecraft:scute').item
  })
})
```

Available properties:

Property	Value Type	Description
maxStackSize	int	Sets the maximum stack size for items. Default is 64 for most items.
maxDamage	int	Sets the maximum damage an item can take before it is broken.
craftingRemainder	Item	Sets the item left behind in the crafting grid when this item is used as a crafting ingredient (like milk buckets in the cake recipe). Most items do not have one.

Property	Value Type	Description
fireResistant	boolean	If this item burns in fire and lava. Most items are false by default, but Ancient Debris and Netherite things are not.
rarity	Rarity	Sets the items rarity. This is mainly used for the name colour. COMMON by default. Nether Stars and Elytra are UNCOMMON, Golden Apples are RARE and Enchanted Golden Apples are EPIC.
burnTime	int	Sets the burn time (in ticks) in a regular furnace for this item. Note that Smokers and Blast Furnaces burn fuel twice as fast. Coal is 1600.
foodProperties	FoodProperties	Sets the items food properties to the provided properties. Can be `null` to remove food properties.
foodProperties	Consumer<FoodBuilder>	Sets the properties according to the consumer. See below for more info.
digSpeed	float	Sets the items digging speed to the number provided. See table below for defaults.
tier	Consumer<MutableToolTier>	Currently BROKEN! Sets the tools tier according to the consumer. See below for more info.
attackDamage	double	Sets the attack damage of this item.
attackSpeed	double	Sets the attack speed of this item
armorProtection	double	Sets the armor protection for this item. 20 is a full armour bar.
armorToughness	double	Adds an armor toughness bonus.
armorKnockbackResistance	double	Add an armor knockback resistance bonus. Can be negative. 1 is full knockback resistance.

Tool defaults

Tier	level	maxDamage	digSpeed	attackDamage (this is a bonus modified by the tool type value, not the final value)	enchantmentValue
Wood	0	59	2	0	15
Stone	1	131	4	1	5
Iron	2	250	6	2	14
Diamond	3	1561	8	3	10
Gold	0	32	12	0	22
Netherite	4	2031	9	4	15

Armor defaults

All boxes with multiple values are formatted [head, chest, legs, feet]. Boxes with single values are the same for every piece.

Tier	maxDamage	armourProtection	armorToughness	armorKnockbackResistance
Leather	[65, 75, 80, 55]	[1, 2, 3, 1]	0	0
Chain	[195, 225, 240, 165]	[1, 4, 5, 2]	0	0
Iron	[195, 225, 240, 165]	[2, 5, 6, 2]	0	0
Gold	[91 ,105, 112, 77]	[1, 3, 5, 2]	0	0
Diamond	[429, 495, 528, 363]	[3, 6, 8, 3]	2	0
Turtle (only has helmet)	[325, nil, nil. nil]	[2, nil, nil, nil]	0	0
Netherite	[481, 555, 592, 407]	[3, 6, 8, 3]	3	0.1
Elytra (not actually armor)	[nil, 432, nil, nil]	0	0	0

Tier

Broken at the moment! https://github.com/KubeJS-Mods/KubeJS/issues/662. Use the non tier methods instead.

Tools

```
ItemEvents.modification(event => {
  event.modify('golden_sword', item => {
    item.tier = tier => {
      tier.speed = 12
      tier.attackDamageBonus = 10
      tier.repairIngredient = '#forge:storage_blocks/gold'
      tier.level = 3
    }
  })
  event.modify('wooden_sword', item => {
    item.tier = tier => {
      tier.enchantmentValue = 30
    }
  })
})
```

Property	Value Type	Description
uses	int	The maximum damage before this tool breaks. Identical to maxDamage.
speed	float	The digging speed of this tool.
attackDamageBonus	float	The bonus attack damage of this tool.
level	int	The mining level of this tool.
enchantmentValue	int	The enchanting power of this tool. The higher this is, the better the enchantments at an Enchanting Table are.
repairIngredient	Ingredient	The material used to repair this tool in an anvil.

Armor

Doesnt actually exist/work at the moment. Sorry.

Food

```
ItemEvents.modification(event => {
  event.modify('minecraft:diamond', item => {
```

```
    item.foodProperties = food => {
        food.hunger(2)
        food.saturation(3)
        food.fastToEat(true)
        food.eaten(e => e.player.tell('you ate')) // this is broken, use ItemEvents.foodEaten instead.
    }
})

    event.modify('pumpkin_pie', item => {
        item.foodProperties = null // make pumpkin pies inedible
    })
})
```

Method	Parameters	Description
hunger	int h	Sets the hunger restored when this item is eaten
saturation	float s	Sets the saturation mulitplier when this food is eaten. This is not the final value, it goes through some complicated maths first
meat	boolean flag (optional, true by default)	Sets if this item is considered meat. Meat can be fed to wolves to heal them.
alwaysEdible	boolean flag (optional, true by default)	If this item can be eaten even if your food bar is full. Chorus Fruit has this true by default.
fastToEat	boolean flag (optional, true by default)	If this item is fast to eat, like Dried Kelp.
effect	ResourceLocation mobEffectId, int duration, int amplifier, float probability	Adds an effect to the entity who eats this, like a Golden Apple
removeEffect	MobEffect mobEffect	Removes the effect from the entity who eats this, like Honey Bottles (poison).
eaten	Consumer<FoodEatenEventJS> e	BROKEN! Use ItemEvents.foodEaten in server scripts instead.

Custom Blocks

> This is a [startup script](#), meaning that you will need to *restart your game* each time you want to make changes to it.

You can register many types of custom blocks in KubeJS. Here's the simplest way:

```
StartupEvents.registry("block", (event) => {
  event.create("example_block") // Create a new block with ID "kubejs:example_block"
})
```

That's it! Launch the game, and assuming you've left KubeJS's auto-generated resources alone, there should be a fully-textured block in the Creative menu under KubeJS (purple dye). KubeJS will also generate the name "Example Block" for you.

To make modifications to this block, we use the **block builder** returned by the `event.create()` call. The block builder allows us to chain together multiple modifications. Let's try making some of the more common modifications:

```
StartupEvents.registry("block", (event) => {
    event.create("example_block") // Create a new block
    .displayName("My Custom Block") // Set a custom name
    .material("wood") // Set a material (affects the sounds and some properties)
    .hardness(1.0) // Set hardness (affects mining time)
    .resistance(1.0) // Set resistance (to explosions, etc)
    .tagBlock("my_custom_tag") // Tag the block with `#minecraft:my_custom_tag` (can have multiple tags)
    .requiresTool(true) // Requires a tool or it won't drop (see tags below)
    .tagBlock("my_namespace:my_other_tag") // Tag the block with `#my_namespace:my_other_tag`
    .tagBlock("mineable/axe") //can be mined faster with an axe
    .tagBlock("mineable/pickaxe") // or a pickaxe
    .tagBlock('minecraft:needs_iron_tool') // the tool tier must be at least iron
})
```

All Block Builder Methods

In case it wasn't covered above, here's list of each method you can use when building a block.

- `displayName('name')`
  - Sets the item's display name.
- `material('material')` (No longer supported in 1.20+, see `mapColor` and `soundType` below!)
  - Set the item's material to an available material from the Materials List:

Materials List

air
amethyst
bamboo
bamboo_sapling
barrier
bubble_column
buildable_glass
cactus
cake
clay
cloth_decoration
decoration
dirt
egg
explosive
fire
froglight
frogspawn
glass
grass
heavy_metal
ice
ice_solid
lava
leaves
metal
moss
nether_wood
piston
plant
portal
powder_snow
replaceable_fireproof_plant
replaceable_plant
replaceable_water_plant

sand
sculk
shulker_shell
snow
sponge
stone
structural_air
top_snow
vegetable
water
water_plant
web
wood
wool

- `mapColor(MapColor)` (1.20.1+ only)
  - Set block map color, you can [find the entire list here](), use ID in lowercase, e.g. `'color_light_green'`.
- `soundType(SoundType)` (1.20.1+ only)
  - Set block sound type:

SoundType List

Instead of using `soundType(SoundType)` you can also use one of these shortcut methods:

- `noSoundType()`
- `woodSoundType()`
- `stoneSoundType()`
- `gravelSoundType()`
- `grassSoundType()`
- `sandSoundType()`
- `cropSoundType()`
- `glassSoundType()`

wood
gravel
grass
lily_pad
stone
metal
glass
wool
sand

snow
powder_snow
ladder
anvil
slime_block
honey_block
wet_grass
coral_block
bamboo
bamboo_sapling
scaffolding
sweet_berry_bush
crop
hard_crop
vine
nether_wart
lantern
stem
nylium
fungus
roots
shroomlight
weeping_vines
twisting_vines
soul_sand
soul_soil
basalt
wart_block
netherrack
nether_bricks
nether_sprouts
nether_ore
bone_block
netherite_block
ancient_debris
lodestone
chain
nether_gold_ore
gilded_blackstone
candle
amethyst
amethyst_cluster
small_amethyst_bud
medium_amethyst_bud

large_amethyst_bud
tuff
calcite
dripstone_block
pointed_dripstone
copper
cave_vines
spore_blossom
azalea
flowering_azalea
moss_carpet
pink_petals
moss
big_dripleaf
small_dripleaf
rooted_dirt
hanging_roots
azalea_leaves
sculk_sensor
sculk_catalyst
sculk
sculk_vein
sculk_shrieker
glow_lichen
deepslate
deepslate_bricks
deepslate_tiles
polished_deepslate
froglight
frogspawn
mangrove_roots
muddy_mangrove_roots
mud
mud_bricks
packed_mud
hanging_sign
nether_wood_hanging_sign
bamboo_wood_hanging_sign
bamboo_wood
nether_wood
cherry_wood
cherry_sapling
cherry_leaves
cherry_wood_hanging_sign

chiseled_bookshelf
suspicious_sand
suspicious_gravel
decorated_pot
decorated_pot_cracked

You can construct your own sound type with `new SoundType(volume, pitch, breakSound, stepSound,` `placeSound, hitSound, fallSound)` where volume and pitch are floats 0.0 - 1.0 (usually leave it as 1.0) and all sounds are SoundEvents.

- `property(BlockProperty)`
    - Adds more blockstates to the block, like being waterlogged or facing a certain direction. A full list of properties is available in the Properties List:

Properties List

Usage: `.property(BlockProperties.PICKLES)`

Boolean Properties (true/false):

attached,
berries,
bloom,
bottom,
can_summon,
conditional,
disarmed,
down,
drag,
east,
enabled,
extended,
eye,
falling,
hanging,
has_book,
has_bottle_0,
has_bottle_1,
has_bottle_2,
has_record,
inverted,
in_wall,
lit,

locked,
north,
occupied,
open,
persistent,
powered,
short,
shrieking,
signal_fire,
snowy,
south,
triggered,
unstable,
up,
vine_end,
waterlogged,
west

Integer properties:

age_1,
age_2,
age_3,
age_4,
age_5,
age_7,
age_15,
age_25,
bites,
candles,
delay,
distance,
eggs,
hatch,
layers,
level,
level_cauldron,
level_composter,
level_flowing,
level_honey,
moisture,
note,
pickles,
power,

respawn_anchor_charges,
rotation_16,
stability_distance,
stage

Directional Properties:

facing,
facing_hopper,
horizontal_facing,
vertical_direction

Other (enum) Properties:

attach_face,
axis,
bamboo_leaves,
bed_part,
bell_attachment,
chest_type,
door_hinge,
double_block_half,
dripstone_thickness,
east_redstone,
east_wall,
half,
horizontal_axis,
mode_comparator,
north_redstone,
north_wall,
noteblock_instrument,
orientation,
piston_type,
rail_shape,
rail_shape_straight,
sculk_sensor_phase,
slab_type,
south_redstone,
south_wall,
stairs_shape,
structureblock_mode,
tilt,
west_redstone,
west_wall

- `tagBlock('namespace:tag_name')`
  - adds a tag to the block
- `tagItem('namespace:tag_name')`
  - adds a tag to the block's item, if it has one
- `tagBoth('namespace:tag_name')`
  - adds both block and item tag if possible
- `hardness(float)`
  - Sets the block's Hardness value. Used for calculating the time it takes for the block to be destroyed.
- `resistance(float)`
  - Set's the block's resistance to things like explosions
- `unbreakable()`
  - Shortcut to set the resistance to MAX_VALUE and hardness to -1 (like bedrock)
- `lightLevel(number)`
  - Sets the block's light level.
  - Passing an integer (0-15) will set the block's light level to that value.
  - Passing a float (0.0-1.0) will multiply that number by 15, then set the block's light level to the nearest integer
- `opaque(boolean)`
  - Sets whether the block is opaque. Full, opaque blocks will not let light through.
- `fullBlock(boolean)`
  - Sets whether the block renders as a full block. Full blocks have certain optimizations applied to them, such as not rendering terrain behind them. If you're using `.box()` to make a custom hitbox, please set this to `false`.
- `requiresTool(boolean)`
  - If `true`, the block will use certain block tags to determine whether it should drop an item when mined. For example, a block tagged with `#minecraft:mineable/axe`, `#minecraft:mineable/pickaxe`, and `#minecraft:needs_iron_tool` would drop nothing unless it was mined with an axe or pickaxe that was at least iron level.
- `renderType('solid'|'cutout'|'translucent')`
  - Sets the render type.
    - `cutout` is required for blocks with texture like glass, where pixels are either transparent or not
    - `translucent` is required for blocks like stained glass, where pixels can be semitransparent
    - otherwise, use `solid` if all pixels in your block are opaque.
- `color(tintindex, color)`
  - Recolors a block to a certain color
- `textureAll('texturepath')`
  - Textures all 6 sides of the block to the same texture.
  - The path must look like `kubejs:block/texture_name` (which would be included under `kubejs/assets/kubejs/textures/block/texture_name.png`).
  - Defaults to `kubejs:block/<block_name>`
- `texture('side', 'texturepath')`
  - Texture one side by itself. Valid sides are `up`, `down`, `north`, `south`, `east`, and `west`.
- `model('modelpath')`

- ○ Specify a custom model.
  - ○ The path must look like `kubejs:block/texture_name` (which would be included under `kubejs/assets/kubejs/models/block/texture_name.png`).
  - ○ Defaults to `kubejs:block/<block_name>` .
- `noItem()`
  - ○ Removes the associated item. Minecraft does this by default for a few blocks, like nether portal blocks. Use this if the player should never be able to hold or place the block.
- `box(x0, y0, z0, x1, y1, z1, boolean)`
- `box(x0, y0, z0, x1, y1, z1)` // defaults to true
  - ○ Sets a custom hitbox for the block, affecting collision. You can use this multiple times to define a complex shape composed of multiple boxes.
  - ○ Each box is a rectangular prism with corners at (x0,y0,z0) and (x1,y1,z1)
  - ○ You will probably want to set up a custom block model that matches the shape you define here.
  - ○ The final boolean determines the coordinate scale of the box. Passing in `true` will use the numbers 0-16, while passing in `false` will use coordinates ranging from 0.0 to 1.0
- `noCollision()`
  - ○ Removes the default full-block hitbox, allowing you to fall through the block.
- `notSolid()`
  - ○ Tells the renderer that the block isn't solid.
- `waterlogged()`
  - ○ Allows the block to be waterloggable.
- `noDrops()`
  - ○ The block will not drop itself, even if mined with silk touch.
- `slipperiness(float)`
  - ○ Sets the slipperiness of the block. Affects how much entities slide while moving on it. Almost every block in Vanilla has a slipperiness value of 0.6, except slime (0.8) and ice (0.98).
- `speedFactor(float)`
  - ○ A modifier affecting how quickly players walk on the block.
- `jumpFactor(float)`
  - ○ A modifier affecting how high players can jump off the block.
- `randomTick(consumer<randomTickEvent>)`
  - ○ A function to run when the block recieves a random tick.
- `item(consumer<`[itemBuilder](#)`>)`
  - ○ Modify certain properties of the block's item (see link)
- `setLootTableJson(json)`
  - ○ Pass in a custom loot table JSON directly
- `setBlockstateJson(json)`
  - ○ Pass in a custom blockstate JSON directly
- `setModelJson(json)`
  - ○ Pass in a custom model JSON directly
- `noValidSpawns(boolean)`
  - ○ If `true` , the block is not counted as a valid spawnpoint for entities

- `suffocating(boolean)`
  - Whether the block will suffocate entities that have their head inside it
- `viewBlocking(boolean)`
  - Whether the block counts as blocking a player's view.
- `redstoneConductor(boolean)`
  - Sets whether the block will conduct redstone. True by default.
- `transparent(boolean)`
  - Sets whether the block is transparent or not
- `defaultCutout()`
  - batches a bunch of methods to make blocks such as glass
- `defaultTranslucent()`
  - similar to defaultCutout() but using translucent layer instead

- `suffocating(boolean)`
  - Whether the block will suffocate entities that have their head inside it
- `viewBlocking(boolean)`
  - Whether the block counts as blocking a player's view.

Block Modification

> The block modification event is a startup event.

`BlockEvents.modification` event is a startup script event that allows you to change properties of existing blocks.

```
BlockEvents.modification(e => {
  e.modify('minecraft:stone', block => {
    block.destroySpeed = 0.1
    block.hasCollision = false
  })
})
```

All available properties:

- `String material`
- `boolean hasCollision`
- `float destroySpeed`
- `float explosionResistance`
- `boolean randomlyTicking`
- `String soundType`
- `float friction`
- `float speedFactor`
- `float jumpFactor`
- `int lightEmission`
- `boolean requiresTool`

Custom Tiers

The custom tier event is a startup event.

You can make custom tiers for armor and tools in a startup script. They are not reloadable without restarting the game. The events are not cancellable.

Tool tiers

```
ItemEvents.toolTierRegistry(event => {
  event.add('tier_id', tier => {
    tier.uses = 250
    tier.speed = 6.0
    tier.attackDamageBonus = 2.0
    tier.level = 2
    tier.enchantmentValue = 14
    tier.repairIngredient = '#forge:ingots/iron'
  })
})
```

Armor tiers

```
ItemEvents.armorTierRegistry(event => {
  event.add('tier_id', tier => {
    tier.durabilityMultiplier = 15 // Each slot will be multiplied with [13, 15, 16, 11]
    tier.slotProtections = [2, 5, 6, 2] // Slot indicies are [FEET, LEGS, BODY, HEAD]
    tier.enchantmentValue = 9
    tier.equipSound = 'minecraft:item.armor.equip_iron'
    tier.repairIngredient = '#forge:ingots/iron'
    tier.toughness = 0.0 // diamond has 2.0, netherite 3.0
    tier.knockbackResistance = 0.0
  })
})
```

Worldgen

General Notes

Biome Filters:

Biome filters work similarly to *recipe filters* and can be used to create complex and exact filters to fine-tune where your features may and may not spawn in the world. They are used for the `biomes` field of a feature and may look something like this:

```
WorldgenEvents.add(event => {
  event.addOre(ore => {
    // let's look at all of the 'simple' filters first
    ore.biomes = 'minecraft:plains'    // only spawn in exactly this biome
    ore.biomes = /^minecraft:.*/    // spawn in all biomes that match the given pattern
    ore.biomes = '#minecraft:is_forest'    // spawn in all biomes tagged as 'minecraft:is_forest'

    // filters can be arbitrarily combined using AND, OR and NOT logic
    ore.biomes = {}    // empty AND filter, always true
    ore.biomes = []    // empty OR filter, always true
    ore.biomes = { not: 'minecraft:ocean' }    // spawn in all biomes that are NOT 'minecraft:ocean'

    // since AND filters are expressed as maps and expect string keys,
    // all of the 'primitive' filters can also be expressed as such
    ore.biomes = {    // see above for an explanation of these filters
      id: 'minecraft:plains',
      id: /^minecraft:.*/,    // regex (also technically an ID filter)
      tag: 'minecraft:is_forest',
    }
    // note all of the above syntax may be mixed and matched individually
  })
})
```

Rule Tests and Targets:

In 1.18, Minecraft WorldGen has changed to a "target-based" replacement system, meaning you can specify specific blocks to be replaced with specific other blocks within the same feature configuration. (For example, this is used to replace Stone with the normal ore and Deepslate with the Deepslate ore variant).

Each target gets a "rule test" as input (something that checks if a given block state should be replaced or not) and produces a specific output block state. While scripting, both of these concepts are expressed as the same class: `BlockStatePredicate`.

Syntax-wise, `BlockStatePredicate` is pretty similar to biome filters as they too can be combined using `AND` or `OR` filters (which is why we will not be repeating that step here), and can be used to match one of three different things fundamentally:

1. **Blocks:** these are simply parsed as strings, so for example `'minecraft:stone'` to match Stone
2. **Block States:** these are parsed as the block ID followed by an array of properties. You would use something like `'minecraft:furnace[lit=true]'` to match only Furnace blocks that are lit. You can use `F3` to figure out a block's properties, as well as possible values through using the debug stick.
3. **Block Tags**: these are parsed in the "familiar" tag syntax, so you could use `'#minecraft:base_stone_overworld'` to match all types of stone that can be found generating in the ground in the Overworld.

> Note that these are **block** tags, not **item** tags. They may (and probably will) be different! (F3 is your friend!)

> You can also use regular expressions with block filters, so `/^mekanism:.+_ore$/` would match any block from Mekanism whose ID ends with `_ore`. Keep in mind this will *not* match block state properties!

> When a `RuleTest` is required instead of a `BlockStatePredicate`, you can also supply that rule test directly in the form of a JavaScript object (it will then be parsed the same as vanilla would parse JSON or NBT objects). This can be useful if you want rule tests that have a random chance to match.

More examples of how targets work can be found in the example script down below.

Height Providers:

Another system that may appear a bit confusing at first is the system of "height providers", which are used to determine at what Y level a given ore should spawn and with what frequency. Used in tandem with this feature are the so-called "vertical anchors", which may be used to get the height of something relative to a specific anchor point (for example the top or bottom of the world).

In KubeJS, this system has been simplified a bit to make it easier to use for script developers. There are two common types of ore placement:

1. **Uniform**: has the same chance to spawn anywhere in between the two anchors
2. **Triangle**: is more likely to spawn in the center of the two anchors than it is to spawn further outwards

To use these two, you can use the methods `uniformHeight` and `traingleHeight` in `AddOreProperties`, respectively. Vertical anchors have also been simplified, as you can use the `aboveBottom` / `belowTop` helper methods in `AddOreProperties`.

Once again, see the example script for more information!

Example script

```
WorldgenEvents.add(event => {
  // use the anchors helper from the event
  const { anchors } = event

  event.addOre(ore => {
    ore.id = 'kubejs:glowstone_test_lmao' // (optional, but recommended) custom id for the feature
    ore.biomes = {
      not: 'minecraft:savanna' // biome filter, see above (technically also optional)
    }

    // examples on how to use targets
    ore.addTarget('#minecraft:stone_ore_replaceables', 'minecraft:glowstone') // replace anything in the vanilla stone_ore_replaceables tag with Glowstone
    ore.addTarget('minecraft:deepslate', 'minecraft:nether_wart_block')     // replace Deepslate with Nether Wart Blocks
    ore.addTarget([
      'minecraft:gravel',     // replace gravel...
      /minecraft:(.*)_dirt/   // or any kind of dirt (including coarse, rooted, etc.)...
    ], 'minecraft:tnt')       // with TNT (trust me, it'll be funny)

    ore.count([15, 50])         // generate between 15 and 50 veins (chosen at random), you could use a single number here for a fixed amount of blocks
      .squared()              // randomly spreads the ores out across the chunk, instead of generating them in a column
      .triangleHeight(□□□      // generate the ore with a triangular distribution, this means it will be more likely to be
```

```
placed closer to the center of the anchors
      anchors.aboveBottom(32),   // the lower bound should be 32 blocks above the bottom of the world, so in
this case, Y = -32 since minY = -64
      anchors.absolute(96)     // the upper bound, meanwhile is set to be just exactly at Y = 96
    )                 // in total, the ore can be found between Y levels -32 and 96, and will be most likely at Y = (9
32) / 2 = 32


  // more, optional parameters (default values are shown here)
  ore.size = 9                    // max. vein size
  ore.noSurface = 0.5                // chance to discard if the ore would be exposed to air
  ore.worldgenLayer = 'underground_ores'  // what generation step the ores should be generated in (see below)
  ore.chance = 0            // if != 0 and count is unset, the ore has a 1/n chance to generate per chunk
  })


  // oh yeah, and did I mention lakes exist, too?
  // (for now at least, Vanilla is likely to remove them in the future)
  event.addLake(lake => {
    lake.id = 'kubejs:funny_lake' // BlockStatePredicate
    lake.chance = 4
    lake.fluid = 'minecraft:lava'
    lake.barrier = 'minecraft:diamond_block'
  })
})


WorldgenEvents.remove(event => {
  // print all features for a given biome filter
  event.printFeatures('', 'minecraft:plains')


  event.removeOres(props => {
    // much like ADDING ores, this supports filtering by worldgen layer...
    props.worldgenLayer = 'underground_ores'
    // ...and by biome
    props.biomes = [
      { category: 'icy' },
      { category: 'savanna' },
      { category: 'mesa' }
    ]


    // BlockStatePredicate to remove iron and copper ores from the given biomes
    // Note tags may NOT be used here, unfortunately...
```

```
    props.blocks = ['minecraft:iron_ore', 'minecraft:copper_ore']
  })


  // remove features by their id (first argument is a generation step)
  event.removeFeatureById('underground_ores', ['minecraft:ore_coal_upper', 'minecraft:ore_coal_lower'])
})
```

Generation Steps

1. raw_generation
2. lakes
3. local_modifications
4. underground_structures
5. surface_structures
6. strongholds
7. underground_ores
8. underground_decoration
9. fluid_springs
10. vegetal_decoration
11. top_layer_modification

> It's possible you may not be able to generate some things in their layer, like ores in Dirt, because Dirt hasn't spawned yet. You may have to change the layer to one of the above generation steps by calling ore.worldgenLayer = 'top_layer_modification' . However, this is not recommended.

> Nether ores are generated in underground_decoration step!

Examples

Other

Default Options

You can ship default options from options.txt with KubeJS. This includes keybindings, video settings, enabled resource packs, controls like autojump and toggle sprint and wierd things like advanced tooltips.

Why use this instead of just shipping options.txt? If you ship options.txt then the users options will get overridden every time they update your modpack, where-as KubeJS only sets the options once, on the first time the modpack boots.

To use it simply make a file called `defaultoptions.txt` in the `kubejs/config` folder. Then copy any lines you want to set by default over from the normal options.txt file. You can also just copy the entire file if you want to include everything.

A full list of what options the options.txt file can contain is available on the Minecraft Wiki:

https://minecraft.fandom.com/wiki/Options.txt

Other

Changing Window Title and Icon

Yes, you can do that with KubeJS too.

To change title, all you have to do is change `title` in `kubejs/config/client.properties`.

To change icon, you create a `kubejs/config/packicon.png` image in standard Minecraft texture size preferably (64x64, 128x128, 256x256, that kind of size).

The image has to be saved as 32-bit PNG, not Auto-detect/24-bit, otherwise you will get a JVM crash!

Here's how to do that in PaintNET:

Image not found or type unknown

Example result:

Image not found or type unknown

Image not found or type unknown
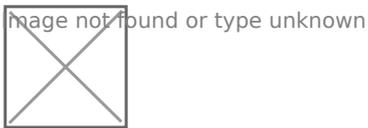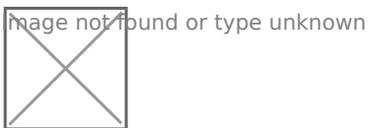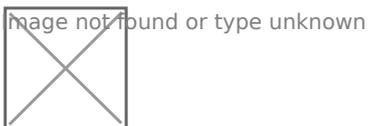
Currently incompatible with Fancy Menu!

Loading Assets and Data

You can also use KubeJS to load assets from resource packs and data from datapacks! While this isn't the only method, its one of the easiest. Other options are <TODO: make and link server datapack load page and client generate assets event page>

The data folder is loaded identically to the data folder in a datapack. If you already have a datapack just copy the folder(s) from inside the datapacks data folder to KubeJS' data folder.

The assets folder is loaded identically to the assets folder in a resourcepack. If you already have a resourcepack just copy the folder(s) from inside the resourcepacks assets folder to KubeJS' assets folder.

Changing Mod Display Names

Yes, it's cursed, but possible!

In a startup script, add this line:

```
Platform.mods.kubejs.name = 'My Modpack Name'
```

This is useful when you add a bunch of items with KubeJS but want them to show your modpack name instead of "KubeJS"

And yes, you can change name of other mods as well:

```
Platform.mods.botania.name = 'Plant Tech Mod'
```

KubeJS 6.1 Update

For script and pack developers

- **Scheduled events now take in durations** (especially strings such as `200 t` for tick durations as well) for their delays!
- `NetworkEvents.fromServer` and `NetworkEvents.fromClient` **have been merged into** `NetworkEvents.dataReceived`, which will handle incoming data from the corresponding side based on the script type.
- **Registry:** `event.custom(T)` **is now** `event.createCustom(() => T)`, which takes in a *supplier* rather than an object directly in order to avoid possible early loading of other registry elements it might depend on. Note that custom still exists, but is HEAVILY discouraged for this very reason!
- **Event** `.cancel()` **now exits event block** - This may be a small change but it may affect some scripts. Previously it would only mark event as cancelled and didn't do anything, but now it will act as return; call as well.
- **Event results have been added!** You now have more granular control over how events work, closer to how they are handled on the Architectury / Minecraft side as well! For example:

```
ItemEvents.rightClicked('minecraft:stick', event => {
  // (note that only one of these will work at a time since they all immediately return!)
  event.cancel() // cancels the event and prevents the click from going through
  event.success() // cancels the event and forces the click to go through
  event.exit() // cancels the event without setting a result
  // in events that support custom results like item stacks, you can also do the following:
  event.success('minecraft:apple') // success + the result is an apple 🍎
})
```

Right now, this new system is only actively used for item right click events, but will be expanded to more events as time goes on (obviously without breaking scripts, and just using `event.cancel()` will still work just fine)!

- **Massive backend rewrites, improved performance a lot** - Lat did another pass over the recipe event and has reworked the way recipes are parsed, as well as fixed async recipe operations, so you should generally notice a decrease in reload times if all works as intended! In some cases, recipes should now load even faster with KJS than they do with just vanilla!

- **No more tag workarounds!** (hopefully) - We have fixed resolving tag ingredients during the recipe event on first world load and generally improved the way recipe filters work, so you shouldn't have to use hacky double-reload workarounds anymore (please just... stop using them already :ioa:)
- **Registries have been fixed on both Forge and Fabric** - We have ironed out some issues with the registry events, so you should now again be able to properly register Fluids, modded registries, etc.
- **Renamed kubejs/logs files from .txt to .log** - So you can now have formatting in VSCode, etc.
- **Fixed resource and data pack order** - User added resource packs and datapacks will now be above KJS generated packs, so you should be able to change textures and other things with them.
- **Added .zip loading from kubjes/data and kubejs/assets** - You simply drop a .zip file in that folder and it will force-load it (above KJS, under user packs)
- **Moved** `debugInfo` **config** from `kubejs/config/common.properties` to `local/kubejsdev.properties`. No idea why it was in common properties in first place, its a *debug* config for devs.
- **Improved** `Platform.mods.modid.name = 'Custom Name'` It should work with custom mod IDs on REI and ModNameTooltip now. You should use `Platform.getInfo('custom_mod_id').name = 'Custom Name'` for non-existent mods.
- **Better recipe integration with ProbeJS** - Because of new schema system in KJS, probe is able to much better display what ingredients go where, with less hacks!
- **.stage(string)** recipe function no longer requires Recipe Stages to work.
- **Fixed flowing fluid textures** on Fabric and other fluid related issues.
- **Fixed errors being way too long in logs** - Believe or not, KJS was not supposed to spit out 150 lines of errors for each recipe.
- **Added a new wrapper** `FluidAmounts` **for... fluid amounts!** For those of you who can't remember just how many blocks, ingots and nuggets are needed to make a bucket, or who just want to have cross-platform script compatibility with their scripts (since Fabric uses "81000 droplets" rather than "1000 mB" for more precise fluid measurements)
- **Added custom toast notifications** - You can use `player.notify(title)`, `(title, subtitle)` or `(Notification.make(...))`.
- **Added /kubejs reload config command** - No longer you have to restart the game to update configs!
- **Added /kubejs packmode [mode] command** - Same as above, but you don't have to mess with files at all.
- **Added /kubejs help command** - Useful links directly in-game.
- **Removed /kjs_hand command** - Instead added /kjs hand (with space) redirect. Some might hate this change, but _ is much harder to reach than space, and I'm sure you'll get used to it quickly and like it better.
- **Fluid registry .tag() fixed** - Now tags flowing fluids too.
- **You can now replace and match fluids** - You <u>must</u> use `Fluid.of('minecraft:water')` instead of plain string, but you can use it in both `{input: Fluid.of('minecraft:water')}` recipe filter and `event.replaceInput('*', Fluid.of('minecraft:water'), Fluid.of('minecraft:lava'))` replace functions for supported recipe types.

For addon mod developers

- **Complete rewrite of recipe system** - Recipes now use recipe schemas, a new system that (almost) fully replaces the old RecipeJS objects. More on that in the [Discord announcement](#)
- **Events now have results** for more precise control over return values and we've added a `hasListeners()` check for performance reasons. The most noticeable change for you is going to be that your own events will need to return a `EventResult`, as well.
- **Fixed datagen issue** - KJS should no longer keep datagens from closing game forever in dev environment. [We truly live in an age of wonders!](#)

Update Primer (sorted by topics, still incomplete):

Recipe Schemas

From the announcement:

> This is <u>the big one</u>. Recipe schemas *completely* change the way custom recipe handlers are registered in KubeJS, and should hopefully also mean a lot less boilerplate code down the line for you. Each recipe is now defined by a *schema* containing a set of *recipe components*, with those components acting as "codecs" for the underlying values. For you, this means the following:
> - Instead of primarily using RecipeJS subclasses, you will now have to define a **RecipeSchema**
>   - Each schema uses a set of **RecipeKeys**, which are named **RecipeComponents** with some additional properties like optional default values and settings for automatic constructor and method generation
>   - A **RecipeComponent** is a reusable definition of a recipe element (such as an in/output item, a fluid, or even just a number value) that has a role (input, output, other), a description (for use in addon mods like ProbeJS) and contains logic for (de)serialisation and bulk recipe operations (i.e. recipe filtering and replacement). There are lots of standard components provided in the `dev.latvian.mods.kubejs.recipe.component` package, including blocks, items and fluids, generic group and logic components (array, map, and, or), and all kinds of primitives (including specialised ones such as number ranges and characters)
>   - While the recipe schema *will* generate constructors by default, you can override this behaviour by defining one yourself using `constructor(factory, keys)`. Note that this will stop the default constructor from being generated, so if you want to keep that, you will have to define it yourself again.
>   (A good example of complex custom recipe constructors is `ShapedRecipeSchema`)

> - While schemas replace **RecipeJS** on the java side, on the JS side, the user is still passed a **RecipeJS** object after creation, with additional autogenerated "builder" methods for each component to allow for the user to set e.g. optional values after recipe creation (e.g. `event.smelting(...).xp(20).cookingTime(100)`). and you can add even more properties or do additional after-load validation by overriding the recipe factory entirely!

Download

You can download KubeJS 6.1 at https://kubejs.com/downloads!

Addons

KubeJS UI

Download: CurseForge

No info yet!

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 the 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.

KubeJS Thermal

Download: [CurseForge](), [Modrinth]()

> This info is currently incomplete!

Supported recipe types:

- furnace
- sawmill
- pulverizer
- smelter
- centrifuge
- press
- crucible
- chiller
- refinery
- brewer
- bottler

event.recipes.thermal.press('minecraft:bone', '#forge:dyes/black')
event.recipes.thermal.crucible(Fluid.of('minecraft:water', 300), '#minecraft:saplings')

- insolator

event.recipes.thermal.insolator('minecraft:bone', '#forge:dyes/black').water(400)

- pulverizer_catalyst
- smelter_catalyst
- insolator_catalyst

event.recipes.thermal.pulverizer_catalyst('minecraft:coal').primaryMod(1.0).secondaryMod(1.0).energyMod(1.0).minChance(0.0).useChance(1.0)

- stirling_fuel
- compression_fuel
- magmatic_fuel
- numismatic_fuel
- lapidary_fuel

```
event.recipes.thermal.lapidary_fuel('minecraft:coal').energy(100000)
```

KubeJS Mekanism

Download: [CurseForge](), [Modrinth]()

No info yet!

KubeJS Immersive Engineering

Download: [CurseForge](#), [Modrinth](#)

No info yet!

KubeJS Blood Magic

Download: CurseForge

No info yet!

KubeJS Tinkers Construct

Download: CurseForge

No info yet!

PonderJS

Download: CurseForge

No info yet!

LootJS

Download: [CurseForge](), [Modrinth]()

No info yet!

ProbeJS

Download: [CurseForge](#)

No info yet!

KubeJS Additions

Download: [CurseForge](#), [Modrinth](#)

No info yet!

For more information please see the project's [Github Page](#), which has usage examples and documentation.

MoreJS

Download: CurseForge

No info yet!

PowerfulJS

Download: CurseForge

No info yet!

beJS

Download: [CurseForge](#)

> The custom BlockEntity event is a startup event.

Block Entities

Custom BlockEntities are created in a startup script. They cannot be reloaded without restarting the game. The event is not cancellable.

```
StartupEvents.registry('block', event => {
⬚event.create('example_block', 'entity' /*has to be here for the BE builder to work*/).displayName('Example Block')
⬚.entity(builder => { // adds a BlockEntity onto this block
⬚    builder.ticker((level, pos, state, be) => { // a tick method, called on block entity tick
        if(!level.clientSide) { // ALWAYS check side, the tick method is called on both CLIENT and SERVER
            if(level.levelData.gameTime % 20 == 0) { // only .levelData.gameTime works for some reason??
                if(level.getBlockState(pos.above()) === Blocks.AIR.defaultBlockState()) {
                    level.setBlock(pos.above(), Blocks.GLASS.defaultBlockState(), 3)
                  ⬚be.persistentData.putBoolean("placed", true)
                } else {
                    level.setBlock(pos.above(), Blocks.AIR.defaultBlockState(), 3)
                    be.persistentData.putBoolean("placed", false)
                }
              ⬚console.info("placed: " + be.persistentData.getBoolean("placed"))
            }
        }
    ⬚}).defaultValues(tag => tag = { progress: 0, example_value_for_extra_saved_data: '0mG this iz Crazyyy'}) //
adds a 'default' saved value, added on block entity creation (place etc)
                                                                // [1st param: CompoundTag consumer]
    .addValidBlock('example_block') // adds a valid block this can attach to, useless in normal circumstances
(except if you want to attach to multiple blocks or are building the BE separately)
    .itemHandler(27) // adds a basic item handler to this block entity, use something like PowerfulJS for more
```

```
advanced functionality
                  // [1st param: slot count]
      .energyHandler(10000, 1000, 1000) // adds a basic FE handler, same as above
                        // [1st param: max energy, 2nd param: max input, 3rd param: max output]
      .fluidHandler(1000, stack => true) // adds a basic fluid handler
          ⬚                  // [1st param: max amount, 2nd param: fluid filter]
   })
})
```

alternatively, you can create the BlockEntity separately and attach it with `EntityBlockJS.Builder#entity('kubejs:be_id')`

```
StartupEvents.registry('block_entity_type', event => {
⬚event.create('example_block')
⬚.ticker((level, pos, state, be) => { // a tick method, called on block entity tick
      if(!level.clientSide) { // ALWAYS check side, the tick method is called on both CLIENT and SERVER
         if(level.levelData.gameTime % 20 == 0) { // only .levelData.gameTime works for some reason??
            if(level.getBlockState(pos.above()) === Blocks.AIR.defaultBlockState()) {
               level.setBlock(pos.above(), Blocks.GLASS.defaultBlockState(), 3)
            } else {
               level.setBlock(pos.above(), Blocks.AIR.defaultBlockState(), 3)
            }
         }
      }
   }).saveCallback((level, pos, be, tag) => { // called on BlockEntity save, don't see why you would ever need
these tbf, but they're here
      tag.putInt("tagValueAa", be.getPersistentData().getInt('progress'))
   }).loadCallback((level, pos, be, tag) => { // called on BlockEntity load, same as above
       be.getPersistentData().putInt("progress", tag.getInt("tagValueAa"))
   }).defaultValues(tag => tag = { progress: 0, example_value_for_extra_saved_data: '0mG this iz Crazyyy'}) //
adds a 'default' saved value, added on block entity creation (place etc)
                                                   // [1st param: CompoundTag consumer]
   .addValidBlock('example_block') // adds a valid block this can attach to, useless in normal circumstances
(except if you want to attach to multible blocks)
   .hasGui() // if ScreenJS is installed, marks this blockentity as having a GUI, doesn't do anything otherwise
   .itemHandler(27) // adds a basic item handler to this block entity, use something like PowerfulJS for more
advanced functionality
                  // [1st param: slot count]
   .energyHandler(10000, 1000, 1000) // adds a basic FE handler, same as above
```

```
                        // [1st param: max energy, 2nd param: max input, 3rd param: max output]
    .fluidHandler(1000, stack => true) // adds a basic fluid handler
                        // [1st param: max amount, 2nd param: fluid filter]
})
```

all valid methods available on all builders:

- addValidBlock('block_id')
- ticker((level, pos, state, blockEntity) => ...)
- defaultValues(tag => ...)
- itemHandler(capacity)
- energyHandler(capacity, maxReceive, maxExtract)
- fluidHandler(capacity, fluidStack => isValid)

Multiblocks

multiblock builder example:

```
StartupEvents.registry('block', event => {
  let CAP_PREDICATE = be => { // has *any* forge capability (item, energy, fluid)
    return be != null && (be.getCapability(ForgeCapabilities.ITEM_HANDLER).present ||
be.getCapability(ForgeCapabilities.FLUID_HANDLER).present ||
be.getCapability(ForgeCapabilities.ENERGY).present)
  }

event.create('multi_block', 'multiblock').material('metal').hardness(5.0).displayName('Multiblock')
    .entity(builder => {
      builder.ticker((level, pos, state, be) => { // tick me here, but ONLY WHEN MULTIBLOCK IS FORMED!!

    })
    .pattern(() => { // ordering is: [aisle: z, aisle contents[]: y, single string: x]
      return BlockPatternBuilder.start()
        .aisle( 'BBB',
                'ACA',
                'AAA')
          .aisle( 'BBB',
                'AAA',
                'AAA')
```

```
        .aisle( 'BBB',
               'AAA',
               'AAA')
        .where('A', BlockInWorld.or(BlockInWorld.hasState(BlockPredicate.forBlock('minecraft:iron_block')),
  BlockInWorld.hasBlockEntity(CAP_PREDICATE)))
        ▢▢▢▢// ^ iron block OR any capability on a BE
            .where('C', BlockInWorld.hasState(BlockPredicate.forBlock('kubejs:multi_block')))
        ▢▢▢▢// ^ controller block
            .where('B', BlockInWorld.hasState(BlockPredicate.forBlock('minecraft:copper_block')))
        ▢▢▢▢// ^ self explanatory
     ▢})
     })
     .property(BlockProperties.HORIZONTAL_FACING) // block builder stuff, facing direction
     .defaultState(state => {
        state.setValue(BlockProperties.HORIZONTAL_FACING, Direction.NORTH)
     })
     .placementState(state => {
        state.setValue(BlockProperties.HORIZONTAL_FACING, state.horizontalDirection.opposite)
     })
  })
```

> currently only 1 input & 1 output per type are set as the multiblock's IO, and it's the last one found in the scan.

extra valid methods on `multiblock` builder:

- `pattern(builder => ...)`

available static methods in `BlockInWorld`:

- `hasState(predicate => ... return boolean)`
- `hasBlockEntity(predicate => ... return boolean)`
- `or(predicate1, predicate2)`
- `and(predicate1, predicate2)`

more advanced example: link

multiblock (and recipe type) example: link

Recipe Types

beJS can create custom recipe types for your block entities to use!

```
StartupEvents.registry('recipe_type', event => {
    event.create('name_here')
        .assembler((recipe, container) => { // optional, but very much suggested
            let results = recipe.results
            for (let i = 0; i < results.size() && i < container.containerSize; ++i) {
                container.setItem(i, results.get(i))
            }
        })
        .maxInputs(2) // required
        .maxOutputs(4) // required
        .toastSymbol('kubejs:block_id_here') // optional
})
```

valid methods on all RecipeType builders:

- assembler((recipe, container) => ...)
- maxInputs(count)
- maxOutputs(count)
- toastSymbol(stack)

Item/Fluid Handlers

beJS has multiple custom handlers that have extra functionality:

IMultipleItemHandler

IMultipleItemHandler is an item handler with multiple slots. valid methods listed below:

- getAllContainers() : List<IItemHandlerModifiable>
- getContainer(index) : IItemHandlerModifiable
- getStackInSlot(container, slot) : ItemStack
- insertItem(container, slot, stack, simulate) : ItemStack
- extractItem(container, slot, amount, simulate) : ItemStack
- getSlotLimit(container, slot) : int
- isItemValid(container, slot, stack) : boolean
- setStackInSlot(container, slot, stack)

IMultipleFluidHandler

IMultipleItemHandler is a fluid handler with multiple slots. valid methods listed below:

- default forge IFluidHandler methods (not listed here)
- `fill(tank, fluidStack, action) : int`
- `drain(tank, fluidStack, action) : FluidStack`
- `drain(tank, maxDrain, action) : FluidStack`

Addons

ScreenJS

Download: [CurseForge](#)

> The custom ContainerMenu event is a startup event.

Custom Container menus are created in a startup script. They cannot be reloaded without restarting the game. The event is not cancellable.

for block entities:

```
StartupEvents.registry('menu', event => {
  event.create('example_block' /*name can be anything*/, 'block_entity')
    .addSlot(-10, -10) // adds a slot into this x,y position on the texture
    .addSlot(10, 200)
    .loop(builder /*this builder*/=> {
      for(let x = 0; x < 9; x++) {
        for (let y = 0; y < 4; y++) {
          builder.addSlot(x * 18 /*<- the width of a slot, remember to add this*/, y * 18, x + y * 4, 0)
        }
      }
    })
    .addOutputSlot(118, 118, 0, 0, 1, 'minecraft:smelting') // adds a slot you can't put an item into, but can pull
an item from
    // LAST PARAMETER CAN BE NULL FOR NO OUTPUT HANDLING
    .inputSlotIndices(0) // sets a list of ITEM HANDLER indexes to handle as slotChanged callback input
    .playerInventoryY(100) // marks the start of the player's inventory on the texture
    .tintColor(0xFF00FF00) // a color to tint the whole inventory texture, in hexadecimal [a, r, g, b]
    .progressDrawable(50, 50, new Rectangle(0, 0, 10, 30), 'forge:textures/white.png', 'up', 'energy') // displays
an energy bar from the blockentity's FE capability
    .slotChanged((menu, level, player, itemHandlers) => {
      console.info('' + player)
    })

    .setBlockEntity('kubejs:example_block') // the block entity type that should open this GUI on right-click
})
```

for any block:

```
StartupEvents.registry('menu', event => {
    event.create('grass_block' /*name can be anything*/, 'block')
        /*default parameter set*/
  ⬚.addItemHandler(9) // adds an item handler.
  ⬚.addItemHandler(1)
  ⬚.inputSlotIndices(0)
        .setBlock('minecraft:grass_block') // the block that should open this GUI on right-click
})
```

for entities:

```
StartupEvents.registry('menu', event => {
    event.create('snow_golem' /*name can be anything*/, 'entity')
        /*default parameter set*/
        .setEntity('minecraft:snow_golem') // the enity type that should open this GUI on right-click
})
```

and lastly, for completely separate 'basic' GUIs:

```
StartupEvents.registry('menu', event => {
    event.create('name_here' /*name can be anything*/)
        /*default parameter set*/
})
```

valid menu types:

- basic (this is the default)
- block_entity
- block
- entity

methods the menu builder supports:

- addSlot(x, y, slotIndex, containerIndex)
- addOutputSlot(x, y, slotIndex, inputContainerIndex, outputContainerIndex, recipeType)
- loop(builder => ...)
- inputSlotIndices(int[] indexes)
- tintColor(color)
- drawable(screenX, screenY, rectangle, textureLocation)

- `progressDrawable(x, y, rectangle, textureLocation, direction, type)`
- `fluidDrawable(x, y, rectangle, textureLocation, direction, tankIndex)`
- `customDrawable(x, y, rectangle, textureLocation, direction, (menu, screen, drawable, direction) => ...)`
- `backroundTexture(texture, rectangle)`
- `quickMoveFunc((player, slotIndex, menu) => ... return item)`
- `slotChanged((menu, level, player, itemHandler) => ...)`
- `validityFunc((player, pos) => ... return boolean)`
- `disablePlayerInventory()`
- `playerInventoryY(yPos)`
- `button(rectangle, textComponent, button => ...)`

default available types:

- PROGRESS
- FUEL
- ENERGY

default available move directions:

- UP
- DOWN
- LEFT
- RIGHT

available types:

- `Rectangle(x, y, u ,v)`
- `MenuUtils` (contains `progress(max, current, length)` for custom bars)
- `RecipeWrapper` (forge `IItemHandlerModifiable` wrapper for recipes)
- `CraftingWrapper` (ScreenJS wrapper class used for `crafting` recipes)

Custom Key Binds

ScreenJS can do custom key bindings! examples & available methods below:

> The custom KeyBind event is a Client event.

```
// client_scripts


KeybindEvents.register(event => {
    event.register(new KeyBind("open_menu" /* name */, InputConstants.KEY_G /* key index, opengl spec */,
"screenjs" /* category name */), (action, modifiers /* modifiers as per OpenGL spec */) => {
```

```
    if (action == 1) { // action == 1 is PRESS
        Minecraft.instance.gui.setOverlayMessage(Text.string('AAA').yellow(), false) // vanilla method
        MenuScreens.create('kubejs:separate', Minecraft.instance, 1000, Text.string('AAA').yellow()) // opens a
GUI container, preferably of type 'basic'
    } else if (action == 0) { // action == 0 is RELEASE
        Minecraft.instance.gui.setOverlayMessage(Text.string('BBB').yellow(), true)
    } else { // action == 2 is REPEAT (after a second of PRESS)
        Minecraft.instance.gui.setOverlayMessage(Text.string('REPEAT').red(), false)
    }
  })
})
```

available methods:

- register

available types:

- KeyBind(name, keyIndex, category)
- KeyAction(action, modifiers)
- InputConstants
- Minecraft (client main class)
-

KubeJS REI Runtime

Download: Curseforge Modrinth

KubeJS REI Runtime lets you show/hide items in REI dynamically, it provides these methods by default:

```
// in client_scripts

REIRuntime.showItem(item); // shows an item in REI
REIRuntime.showItems([item, item, ...]); // shows items in REI
REIRuntime.hideItem(item); // hides an item in REI
REIRuntime.hideItems([item, item, ...]); // hides items in REI
```

KubeJS Botany Pots

Download: Curseforge Modrinth

This mod allows you to create crops, soils, and fertilizers for the Botany Pots mod.

```
ServerEvents.recipes(event => {
  event.recipes.botanypots.crop(
    "minecraft:candle", // seed item
    ["oak_leaves"], // categories that this crop can be planted on
    { block: "minecraft:candle" }, // display block
    [
        Item.of ("minecraft:candle") // item
            .withChance(100) // weight of this entry compared to the others
            .withRolls(1, 2) // the times this loot will be chosen (min, max)
        // for example, when chosen this will give 1 to 2 candles
    ],
    10, // growthTicks
    1, // optional, growthModifier - this can be set to 1 in most cases
  )

  event.recipes.botanypots.soil(
    "minecraft:oak_leaves", // the item that this soil is attached to
    { block: "minecraft:oak_leaves" }, // display block
    ["oak_leaves"], // categories that this soil provides
    100, // growth ticks that this soil will provide, set to -1 for no modifier
    0.5 // optional, growth modifier, example: 0.5 means all crops will take half the time
  )

  event.recipes.botanypots.fertilizer(
    "minecraft:iron_ingot", // fertilizer item
    10, // min growth ticks applied
    20 // max growth ticks applied
    // ex: 10 to 20 ticks will be randomly given to the crop
  )
})
```

```
// fired everytime a crop grows
BotanyPotsEvents.onCropGrow(event => {
    // event.random : the random object associated with the event
    // event.crop : a crop object describing the crop grown
    // event.originalDrops : an array of items this crop drops
    // event.drops : a writable array that changes the drops of the crop
    console.log([event.random, event.crop, event.originalDrops, event.drops].join(","))
})
```

KubeJS Ars Nouveau

Download: [Curseforge](#), [Modrinth](#)

This addon allows you to create recipes for the mod [Ars Nouveau](#)

```
ServerEvents.recipes(event => {
 event.recipes.ars_nouveau.enchanting_apparatus(
    [
        "minecraft:sand",
        "minecraft:sand",
        "minecraft:sand",
        "minecraft:sand",
    ], // input items
    "minecraft:gunpowder", // reagent
    "minecraft:tnt", // output
    1000, // source cost
    // true // keep nbt of reagent, think like a smithing recipe
 );

 event.recipes.ars_nouveau.enchantment(
    [
        "minecraft:sand",
        "minecraft:sand",
        "minecraft:sand",
        "minecraft:sand",
    ], // input items
    "minecraft:vanishing_curse", // applied enchantment
    1, // enchantment level
    1000, // source cost
  );

 event.recipes.ars_nouveau.crush(
    "minecraft:tnt", // input block
    [
        Item.of("minecraft:sand").withChance(1.0),
```

```
//            { item: Item.of("minecraft:sand").withChance(1.0), maxRolls: 4 }
    ] // loot table
    // true // drop the item in world?
);


/*
// this *does* work, but the recipe must be a valid glyph
// in the tome, so this really can only be used to
// replace a glyph's recipe
event.recipes.ars_nouveau.glyph(
    "minecraft:tnt", // output item (glyph)
    [
        "minecraft:sand",
        "minecraft:gunpowder",
    ], // input items
    3 // exp cost
);
*/


// accessible via `/ars-tome id` in this case `/ars-tome kubejs:not_glow`
event.recipes.ars_nouveau.caster_tome(
    "Not-Glow Trap", // name,
    [
        "ars_nouveau:glyph_touch",
        "ars_nouveau:glyph_rune",
        "ars_nouveau:glyph_snare",
        "ars_nouveau:glyph_extend_time",
        "ars_nouveau:glyph_light"
    ], //spell
    "Doesn't snare the target and grant other targets Glowing.", // description
    16718260, // color
    {
        "family": "ars_nouveau:default",
        "pitch": 1.0,
        "volume": 1.0
    },
).id("kubejs:not_glow")

event.recipes.ars_nouveau.imbuement(
    "minecraft:sand", // input item
```

```
        "minecraft:tnt", // output
        1000, // source cost
        []
    )


    event.recipes.ars_nouveau.imbuement(
        "minecraft:red_sand", // input item
        "minecraft:tnt", // output
        1000, // source cost
        []
    )
})
```

KubeJS ProjectE

Download: [Curseforge](#), [Modrinth](#)

Lets you set the EMC values of items and the Philosopher's Stone transformations blocks with the [ProjectE](#) mod. Examples are shown below.

Server side events (`server_scripts`):

```
ProjectEEvents.setEMC(event => {
    // sets the absolute emc value of an item
    event.setEMC("minecraft:cobblestone", 0) // alias. setEMCAfter

    // sets the emc of an item before anything else happens
    // this can sometimes result in this emc value not being
    // set, but also it allows for emc values to be generated
    // from this one; i.e crafting recipes
    event.setEMCBefore("minecraft:stick", 10000);
})

ItemEvents.rightClicked("minecraft:stick", event => {
    let player = event.player;

    // getPlayerEMC will always return a string
    // because emc values can get very large
    player.tell("Your emc is " + ProjectE.getPlayerEMC(player))

    ProjectE.addPlayerEMC(player, 1000);
    // the second argument can be a string because of the above
    // ProjectE.setPlayerEMC also exists

    player.tell("Your new emc is " + ProjectE.getPlayerEMC(player))
})
```

Startup events (`server_scripts`):

```
ProjectEEvents.registerWorldTransmutations(event => {

    event.transform("minecraft:tnt", "minecraft:oak_planks");

})
```

KubeJS Powah

Download: [Curseforge](#) [Modrinth](#)

Allows you to create Energizing Orb recipes from the [Powah](#) mod.

Example:

```
ServerEvents.recipes(event => {
    // .energizing([inputs, ...], output, energy)
    event.recipes.powah.energizing(["minecraft:cobblestone"], "minecraft:tnt", 1000)
})


PowahEvents.registerCoolants(event => {
    // .addFluid(fluid, coolness)
    event.addFluid("minecraft:lava", 10);


    // .addSolid(fluid, coolness)
    event.addSolid("minecraft:cobblestone", 10);
})


PowahEvents.registerHeatSource(event => {
    // .add(block, hotness)
    event.add("minecraft:cobblestone", 10);
})


PowahEvents.registerMagmaticFluid(event => {
    // .add(fluid, hotness)
    event.add("minecraft:water", 10);
})
```

KJSPKG

KJSPKG is a package manager for KubeJS that can allow you to download different example scripts and libraries into your instance. It will automatically manage minecraft version, modloader, dependency and incompatibility control. It works with KubeJS 6 (1.19), KubeJS Legacy (1.16/1.18) and should even work with some pre-legacy versions (1.12)!

Installation

1. Download either the CLI version of KJSPKG or the WIP GUI client.
2. Open a terminal in the `kubejs` directory inside of your instance.
3. Run `kjspkg init` and select your minecraft version/modloader.

Now you are able to install packages into your instance.

Usage

- To download a package, run `kjspkg install <package_name>`
- To remove a package, run `kjspkg remove <package_name>`
- To search for a package, run `kjspkg search <query>`
- To list all packages in your instance, run `kjspkg list`
- To list all of the commands available, run `kjspkg help`

Adding your own package

If you have an example script you would like to share on KJSPKG, check out [the "Adding your own package" section](#) of KJSPKG's README. We are always happy to add more scripts from different authors to our list!

Notable packages

[more-recipe-types](#) (Legacy, 1.16.5/1.18.2, Forge/Fabric)

This package simplifies the process of adding recipes to custom machines from different mods without downloading any addons. For example, this bit of code will add a recipe transforming a stick and an iron ingot to [Powah](#)'s Energizing Orb:

```
onEvent('recipes', event => {
  global.mrt.powah.energizing(event, "minecraft:gold_ingot", ["minecraft:stick", "minecraft:iron_ingot"], 1000);
})
```

For other types, check out the [README file on GitHub](#).

[create-depot-crafting](#) (Legacy, 1.18.2, Fabric)

This package allows you to add custom recipes that use manual combination on the create depot. Example from the README:

```
onEvent('block.right_click', event => {
    global.recipes.create.manual_depot_application(event,
        // Output
        Item.of('expandeddelight:cheese_sandwich'),
        // Inputs
        Ingredient.of('minecraft:bread'), // On depot
        Ingredient.of('expandeddelight:cheese_slice') // In hand
    )
});
```

Showcase:

If you're looking for a Forge port of this package, checkout **create-depot-crafting-forge**. A lot of the times KJSPKG's packages' names end in `-6` if they are a port of a different package for KubeJS 6 (1.19), and end in `-<modloader>` if they are a port of another package for a different modloader as per naming convention.

soljs (KubeJS 6, 1.19.2, Forge/Fabric)

This package ports the mechanics of the 1.12.2 version of The Spice of Life mod to 1.19 using only KubeJS. It works like a standalone mod and does not require any configuration. Depends on AppleSkin.

KubeJS Offline Documentation

Dynamic Documentation in a single html page.

Download: [Curseforge](), [Modrinth]()

KubeJS Offline is a mod that dumps all class data at runtime into a single html file using a single command. `/kubejs_offline`.

Preview Generated Documentation Pages:

[1.19.2 Forge 1.19.2 Fabric]()

[1.18.2 Forge 1.18.2 Fabric]()

[Enigmatica 9]()

How does it work?

When you execute the KubeJS Offline command, a scan of the Java runtime is performed to find what classes exist at that time. This is important as mods might provide new event classes and possibly new methods to existing Minecraft classes.

After the mod has searched what classes exist and are available at that time, it then proceeds to compress that data down into a json object.

It records everything from the full class name, package info, super classes, sub-classes, generic implementations, fields, methods, as well as their relationships to other classes.

This data is then used to create an html page which then runs dependency-less JavaScript to generate the webpage html elements.

You can then open the file in a modern web browser, no need to host it on a server or anything like that.

Additional Features:

You can right click inside the webpage to toggle certain tables, private fields, and other info that you may not need.

There is a search feature you can activate by adding a question mark to the end of the url.
An example of this search is:

https://hunter19823.github.io/kubejsoffline/1.19.2/forge#any--EventJS

KubeJS Farmers Delight

Download: [Curseforge](#)

Example:

Startup Scripts:

```
StartupEvents.registry("block", event => {
  event.create('example_pie', 'farmersdelight:pie')
    .sliceItem('kubejs:example_pie_slice')
    .displayName('Example Pie')
  event.create('example_feast', 'farmersdelight:feast')
    .servingsAmount(3)
    .servingItems(['kubejs:example_feast_serving', 'kubejs:example_feast_serving_2'])
    .displayName('Example Feast')
})


StartupEvents.registry("item", event => {
  event.create('example_knife', 'farmersdelight:knife')
    .displayName('Example Knife')
    .tier('diamond')
})
```

Server Scripts:

```
ServerEvents.recipes(event => {
 event.recipes.farmersdelight.cutting(
      "minecraft:cobblestone",
      "#forge:tools/pickaxes", // tool
      [ // results
        "minecraft:iron_ore",
        Item.of("minecraft:diamond")
          .withChance(0.1)
      ],
      // "" // sound
```

```
);


event.recipes.farmersdelight.cooking(
    ["minecraft:cobblestone"],
    "minecraft:iron_ore", // output
    30, // exp
    10, // cookTime
    "minecraft:bowl", // container
);
})
```

KubeJS Industrial Foregoing

Download: [Curseforge](#)

This lets you modify and create various recipes for [Industrial Foregoing](#)

```
ServerEvents.recipes(event => {
  event.recipes.industrialforegoing.dissolution_chamber(
    ["minecraft:tnt"], // input items
    "minecraft:water", // input fluid
    "minecraft:sand", // output item
    100 // time
  )
//  .outputFluid("minecraft:water"); // output fluid

  event.recipes.industrialforegoing.fluid_extractor(
    "minecraft:tnt", // input block
    "minecraft:sand", // output block
    0.5, // break chance
    "minecraft:lava" // output fluid
  )

  event.recipes.industrialforegoing.stonework_generate(
    "minecraft:tnt",
    100, // water needed
    100, // lava needed
    50, // water consumed
    50 // lava consumed
  )
  event.recipes.industrialforegoing.crusher( // the pickaxe action in the stonework factory
    "minecraft:tnt", // input item
    "minecraft:sand" // output item
  )

  event.recipes.industrialforegoing.laser_drill_ore(
    "minecraft:tnt", // output
```

```
    "minecraft:sand",  // catalyst
    [ //rarity, see below for more details
      {
        "blacklist": {
          "type": "minecraft:worldgen/biome",
          "values": [
            "minecraft:the_end",
            "minecraft:the_void",
            "minecraft:small_end_islands",
            "minecraft:end_barrens",
            "minecraft:end_highlands",
            "minecraft:end_midlands"
          ]
        },
        "depth_max": 16,
        "depth_min": 5,
        "weight": 4,
        "whitelist": {}
      },
      {
        "blacklist": {
          "type": "minecraft:worldgen/biome",
          "values": [
            "minecraft:the_end",
            "minecraft:the_void",
            "minecraft:small_end_islands",
            "minecraft:end_barrens",
            "minecraft:end_highlands",
            "minecraft:end_midlands"
          ]
        },
        "depth_max": 255,
        "depth_min": 0,
        "weight": 1,
        "whitelist": {}
      }
    ]
)

event.recipes.industrialforegoing.laser_drill_fluid(
```

```
            "minecraft:water", // output
            "minecraft:sand", // catalyst
            [ // rarity, see wiki for more details
                {
                    "blacklist": {
                        "type": "minecraft:worldgen/biome",
                        "values": [
                            "minecraft:the_end",
                            "minecraft:the_void",
                            "minecraft:small_end_islands",
                            "minecraft:end_barrens",
                            "minecraft:end_highlands",
                            "minecraft:end_midlands"
                        ]
                    },
                    "depth_max": 16,
                    "depth_min": 5,
                    "weight": 4,
                    "whitelist": {}
                },
                {
                    "blacklist": {
                        "type": "minecraft:worldgen/biome",
                        "values": [
                            "minecraft:the_end",
                            "minecraft:the_void",
                            "minecraft:small_end_islands",
                            "minecraft:end_barrens",
                            "minecraft:end_highlands",
                            "minecraft:end_midlands"
                        ]
                    },
                    "depth_max": 255,
                    "depth_min": 0,
                    "weight": 1,
                    "whitelist": {}
                }
            ],
            "minecraft:zombie" // entity required below
        )
```

```
})
```