Addons

- KubeJS UI
- KubeJS Create
- KubeJS Thermal
- KubeJS Mekanism
- KubeJS Immersive Engineering
- KubeJS Blood Magic
- KubeJS Tinkers Construct
- PonderJS
- LootJS
- ProbeJS
- KubeJS Additions
- MoreJS
- PowerfulJS
- beJS
- ScreenJS
- KubeJS REI Runtime
- KubeJS Botany Pots
- KubeJS Ars Nouveau
- KubeJS ProjectE
- KubeJS Powah
- KJSPKG
- KubeJS Offline Documentation
- KubeJS Farmers Delight
- KubeJS Industrial Foregoing

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 th
scrap
□□Item.of('create:andesite_alloy').withChance(8.0),
□□Item.of('create:cogwheel').withChance(5.0),
□□Item.of('create:shaft').withChance(2.0),
□□Item.of('create:crushed_gold_ore').withChance(2.0),
□□Item.of('2x gold_nugget').withChance(2.0),
□□'iron_ingot',
□□'clock'
□], 'create:golden_sheet', [ // 'create:golden_sheet' is the input
□□// the transitional item set by `transitionalItem('create:incomplete_large_cogwheel')` is the
item used during the intermediate stages of the assembly
□□e.recipes.createDeploying('create:incomplete_precision_mechanism',
['create:incomplete_precision_mechanism', 'create:cogwheel']),
□□// like a normal recipe function, is used as a sequence step in this array. Input and output
have the transitional item
```

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

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

Transitional Items

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

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

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

Splashing/Washing

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

Features:

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

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

Mysterious Conversion

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

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

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

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

Preventing Recipe Auto-Generation

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

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

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

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

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`

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
    )
})
```