Using ProbeJS

ProbeJS is an add-on that is built exclusively to help you program.

What it does:

It generates documentation files from digging around in the game code itself. So, you get all the methods, not only from KubeJS, but also from base Minecraft, no matter they're added by modloader, or from the other mods you install. Not only can you view these docs, but they are also formatted in a way that a sufficiently advanced code editor, like VSCode, can understand. So, you will now get more relevant code suggestions too.

Installation:

Find ProbeJS on the 3rd Party addons list and download the relevant version for you.
Once you've installed it and relaunched your game, run the command `/probejs dump`.
Now you will need to wait a little while, but after some time, you should see a message alerting you that the dump is complete.

What just happened?

You can now look and see that there is a new folder located at `instance/kubejs/probe/` and inside of here there are a more folders and files. These are your docs.

Setting up VS Code

1. In VS Code select `file > open folder`
2. This opens up a file explore window, select the KubeJS folder (`instance/`) and choose select folder.

You're done!

Troubleshooting

For many people, autocompletions won't be popped up as they type. You need to configure your VSCode to setup a valid JavaScript IDE so you can get 100% power of ProbeJS!

No Intellisense at All

For some reason, VSCode downloaded by some people are not having builtin JavaScript/TypeScript support. To check if you have such support enabled, search `@builtin JavaScript` in the extension tab in your VSCode, you should see a plugin named `TypeScript and JavaScript Language Features`, that's the builtin extension for VSCode to support JS/TS.

If not, then you'll have to install the `JavaScript and TypeScript Nightly` to get JS/TS support.

Downloading Intellisense Models

If your ISP is weird, downloading Intellisense models for enabling support can take a long time. You can consider switch to proxy or some other methods to change your Internet environment, maybe even changing a WiFi can work. If not, then sorry, it's an Internet problem, there's no way to solve it on VSCode's end.

Too Many Mods

Completion takes a significant amount of performance. You can't expect VSCode to run super-fast on some ATM8-like modpacks, that's not possible.

For less than 150 mods, VSCode should run at a decent speed, for more than 300 mods, completions are taking >10s since now VSCode need to examine over 100k item/block/entity entries before telling you what to type next.

Usage

Properties and Methods of a Class

To know the methods of a class just type in the class name, like `Item` or `BlockProperties`, then type a `.` now you will see a list of the public methods and properties.

ProbeJS will display the **bean**ed accessors and mutators. However, due to the limitation of JavaScript syntax, if there's a method having same name with a field/bean, then the name will always be resolved to the method.

Type Checking and JSDoc

To add type checking for extra safety when coding JavaScript, add `//@ts-check` to the first line of a JS file, then you will have VSCode guarding your types for the rest of the file. It's extremely useful when you're working with some dangerous code which is likely to crash the game if you have a mistake in type.

Sometimes, due to limitations of TypeScript, you might need to persuade VSCode to skip checking for some part of your code. Adding `//@ts-ignore` would help you to do that.

Or maybe you want to tell VSCode: "This should be a list of item names!", or "This method should have ... as params, and ... as return types!". Then you can add `JSDoc` to tell VSCode to do that:

```
/**
 * @type {Special.Item[]}
 */
let consumableItems = []


ServerEvents.recipes(event => {
/**
 *
 * @param {Internal.Ingredient_} input
 * @param {Internal.ItemStack_} output
 * @returns {Internal.ShapedRecipeJS}
 */
let make3x3Recipe = (input, output) => {
return event.recipes.minecraft.crafting_shaped(output, ["SSS", "SSS", "SSS"], { S: input })
}
})
```

Sometimes, if with `//@ts-check` enabled, you will need to add `//@ts-ignore` to calm VSCode to accept your docs.

Searching by Keyword

If you are in VSCode press the explorer button in the top-ish left to open up the explorer pane.

Now navigate to `probe > generated > globals.d.ts`.
Press `Ctrl + F` and a little search window should pop up in your editor.
Now type in you key word and look through all the matches.

Tips

If you append `class` to the front and ` ` to the end then you will look for classes so like Item has 8635 results for me, but if I type `class Item` then the one I want!

In `events.d.ts` you will find events but only basic information about them.

In `constants.d.ts` you can see different pieces that you can use whereever.

If you want to find the methods of an event, say `item.pickup` find it in one of the files (In this case `events.documented.d.ts`) and here is the line describing it:

```
declare function onEvent(name: 'item.pickup', handler: (event: Internal.ItemPickupEventJS) =>
void)
```

Look closely and find `Internal.ItemPickupEventJS`. Since it says `Internal`, we will look in the the `globals.d.ts` file, but if it says `Registry` then we use `registries.d.ts`.

Now we will go to the generated file and search `ItemPickupEventJS`.
Then we find:

```
/**
* Fired when an item is about to be picked up by the player.
* @javaClass dev.latvian.mods.kubejs.item.ItemPickupEventJS
*/
class ItemPickupEventJS extends Internal.PlayerEventJS {
    getItem(): Internal.ItemStackJS;
    getEntity(): Internal.EntityJS;
    getItemEntity(): Internal.EntityJS;
    canCancel(): boolean;
    get item(): Internal.ItemStackJS;
    get itemEntity(): Internal.EntityJS;
    get entity(): Internal.EntityJS;
    /**
    * Internal constructor, this means that it's not valid unless you use `java()`.
    */
    constructor(player: Internal.Player, entity: Internal.ItemEntity, stack:
Internal.ItemStack);
}
```

This means that we can use the methods `.getItem()` `.getEntity()` `.getItemEntity()` `.canCancel()` `.item` `.itemEntity` and `.entity`.

But if we did `potion.registry` then we get `Registry.Potion` which brings us to:

```
class Potion extends Internal.RegistryObjectBuilderTypes$RegistryEventJS<any> {
create(id: string, type: "basic"): Internal.PotionBuilder;
create(id: string): Internal.PotionBuilder;
}
```

So we can use `event.create('cactus_juice')` but that does not do much so we need to follow one step further and go to the potion builder, which you see is `Internal.PotionBuilder`. Now we search

`PotionBuilder` in `globals.d.ts` then we see:

```
/**
* @javaClass dev.latvian.mods.kubejs.misc.PotionBuilder
*/
class PotionBuilder extends Internal.BuilderBase<Internal.Potion> {
    getRegistryType(): Internal.RegistryObjectBuilderTypes<Internal.Potion>;
    effect(effect: Internal.MobEffect_, duration: number, amplifier: number, ambient: boolean,
visible: boolean): this;
    effect(effect: Internal.MobEffect_, duration: number, amplifier: number, ambient: boolean,
visible: boolean, showIcon: boolean): this;
    effect(effect: Internal.MobEffect_, duration: number, amplifier: number, ambient: boolean,
visible: boolean, showIcon: boolean, hiddenEffect: Internal.MobEffectInstance_): this;
    effect(effect: Internal.MobEffect_, duration: number): this;
    effect(effect: Internal.MobEffect_, duration: number, amplifier: number): this;
    effect(effect: Internal.MobEffect_): this;
    addEffect(effect: Internal.MobEffectInstance_): this;
    createObject(): Internal.Potion;
    get registryType(): Internal.RegistryObjectBuilderTypes<Internal.Potion>;
    /**
    * Internal constructor, this means that it's not valid unless you use `java()`.
    */
    constructor(i: ResourceLocation);
}
```

Now we see the methods that we can call after this.

So in our code we could write:

```
onEvent('potion.registry', event => {
  event.create('cactus_juice').effect('speed', 10, 5)
})
```

Revision #6
Created 2022-10-17 15:07:58 UTC by Q6
Updated 2023-10-14 05:16:27 UTC by Prunoideae