Software Studio
軟體設計與實驗

Cocos Creator : Script

Hung-Kuo Chu
Department of Computer Science
National Tsing Hua University

CS2410

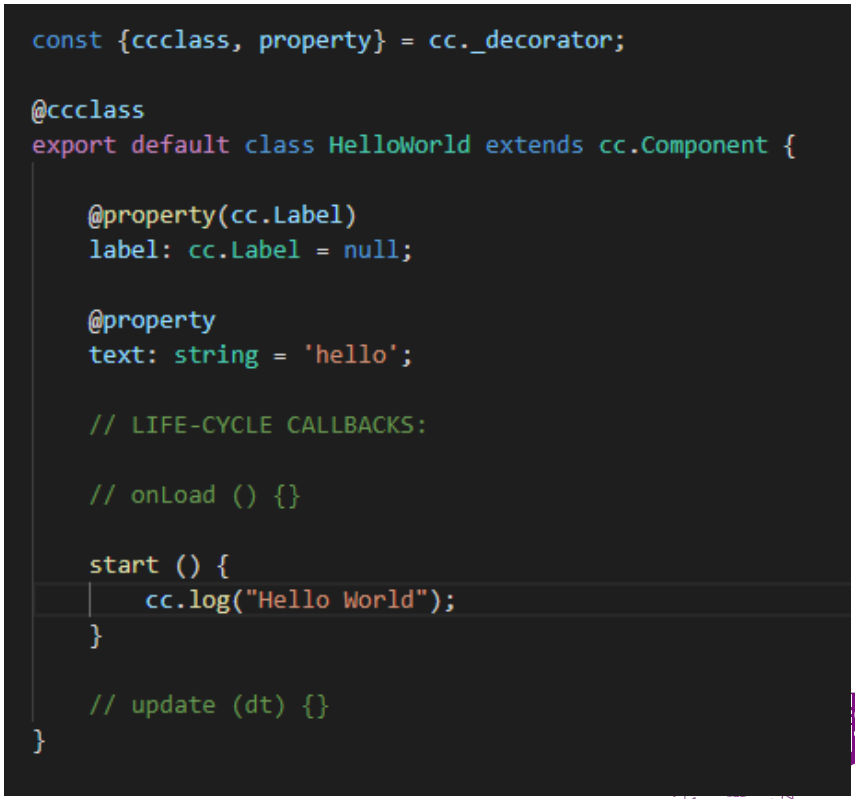
Codeblock Conventions

JavaScript / TypeScript
Program

Script: Basics

- Cocos Creator supports language:
  - **Typescript**, JavaScript, CoffeeScript

- Recommended IDE
  - Visual Studio Code



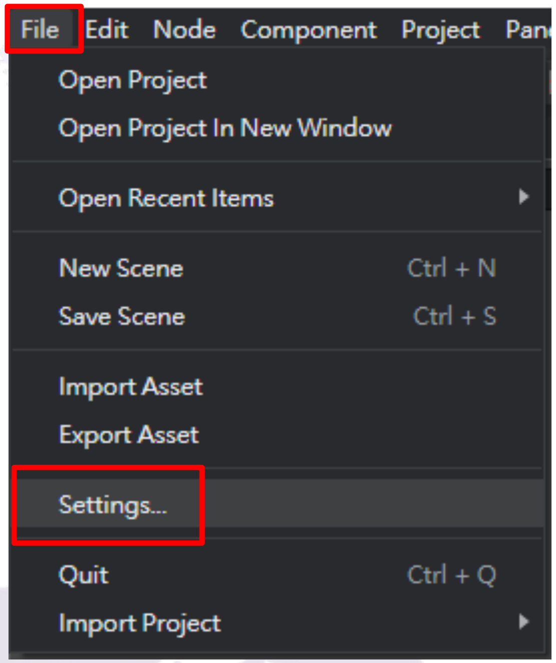
```
const {ccclass, property} = cc._decorator;

@ccclass
export default class HelloWorld extends cc.Component {

    @property(cc.Label)
    label: cc.Label = null;

    @property
    text: string = 'hello';

    // LIFE-CYCLE CALLBACKS:

    // onLoad () {}

    start () {
        cc.log("Hello World");
    }

    // update (dt) {}
}
```

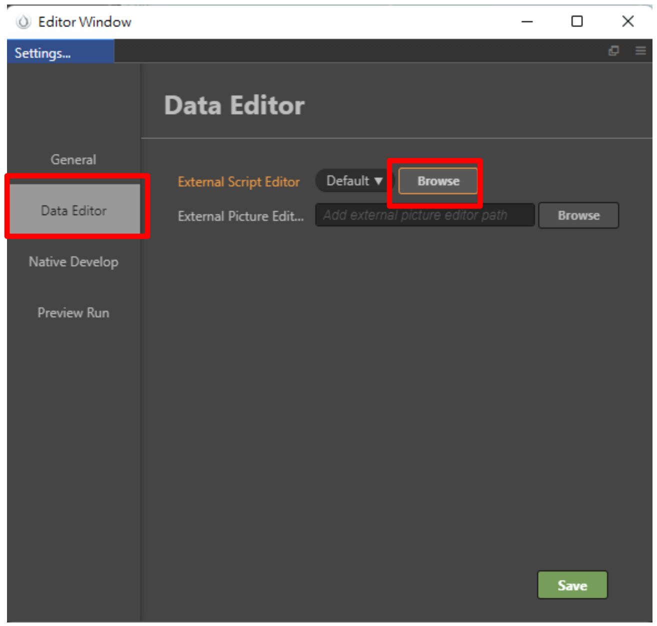
Useful References

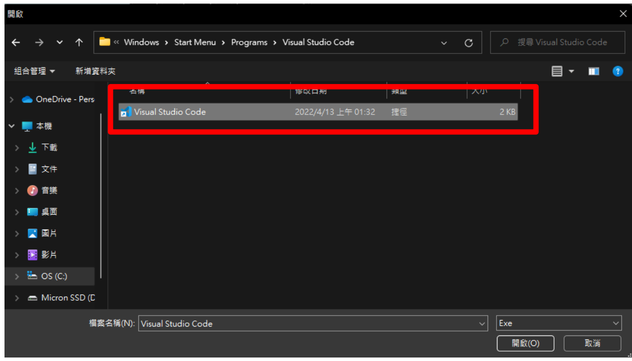
- Cocos (English)
  - https://docs.cocos.com/creator/2.4/manual/en/
- Cocos (Chinese)
  - https://docs.cocos.com/creator/2.4/manual/zh/
- Cocos Creator Forum
  - https://discuss.cocos2d-x.org/c/creator

Environment Setting (Windows)

- Choose default IDE editor
  - File → Settings → Data Editor →External Script Editor → Browse →Choose your IDE

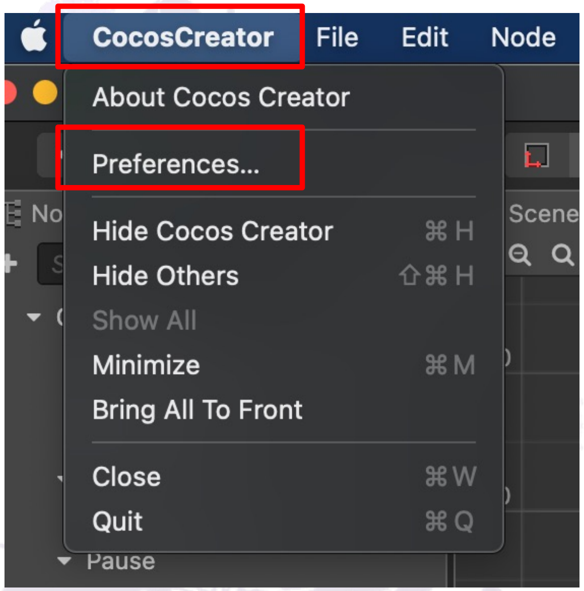


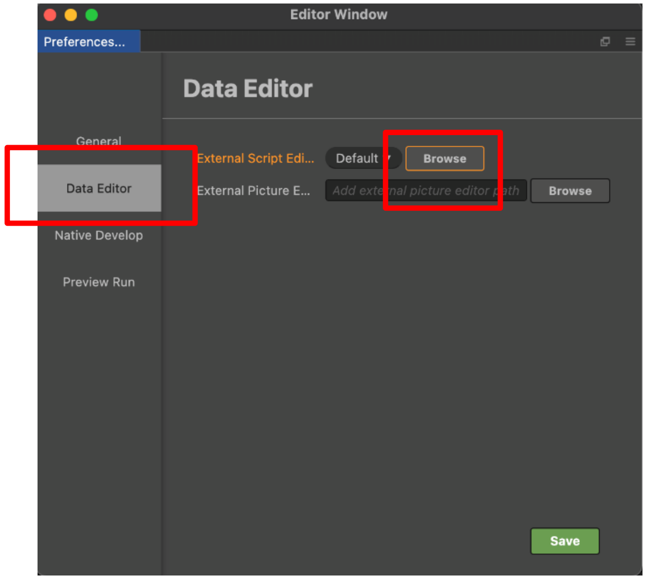


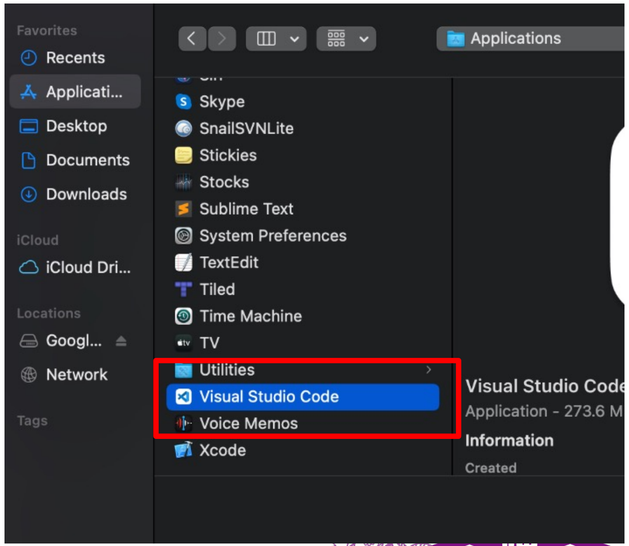


Environment Setting (MacOS)

- Choose default IDE editor
  - Cocoscreator → Preferences → Data Editor →External Script Editor → Browse → Choose your IDE

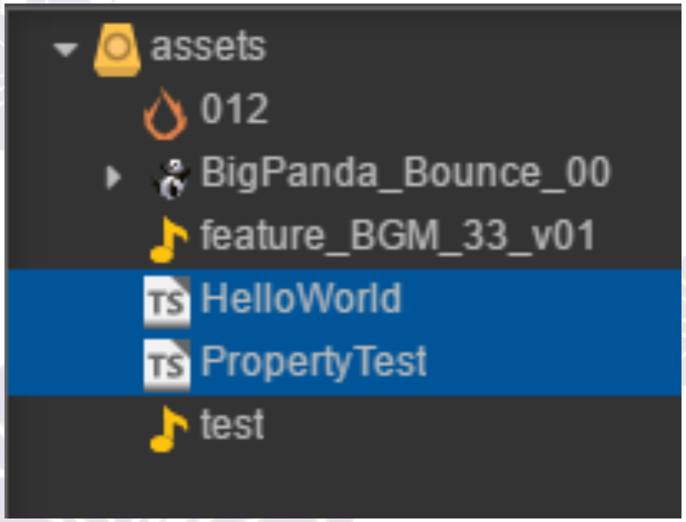


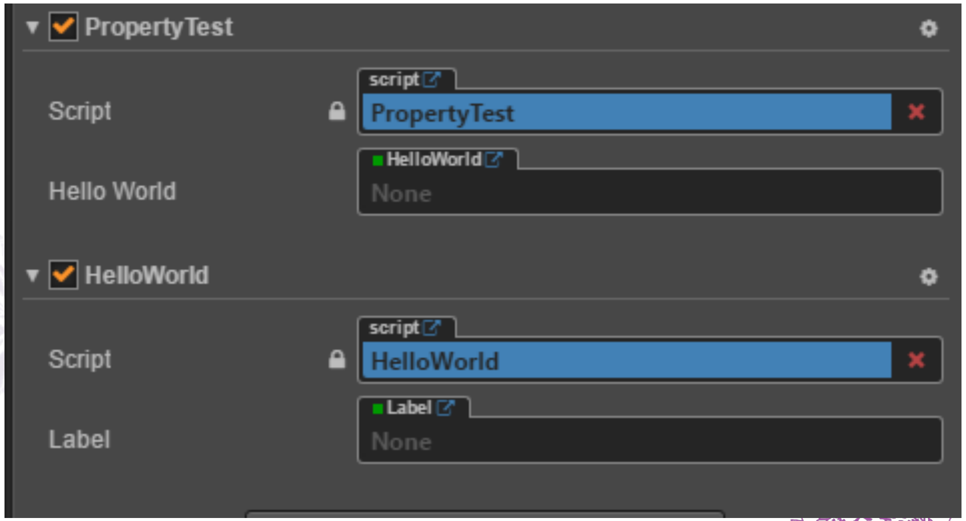




How does Script do?

- Control the behaviors of the Node
- Get information from the Node
- Run your own component

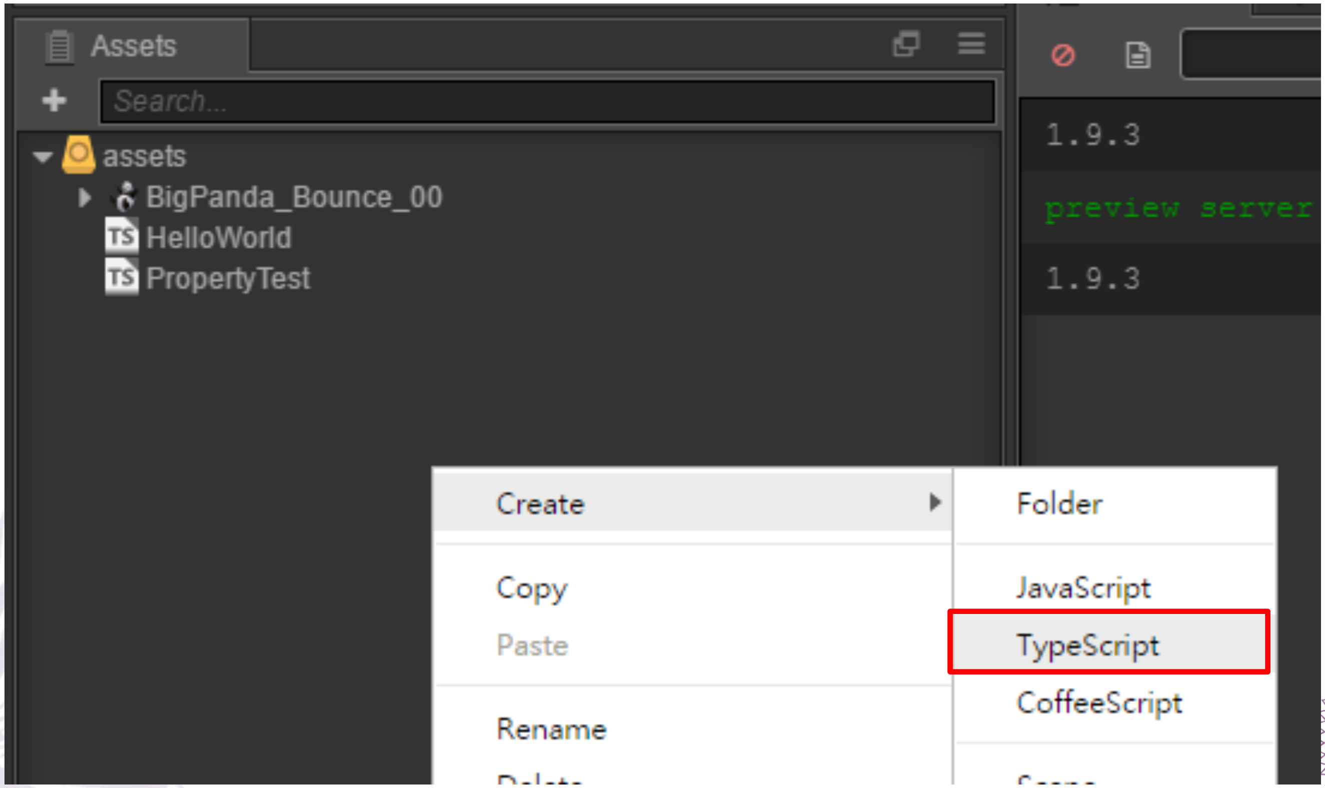




```
PropertyTest
  Script        PropertyTest
  Hello World   None
HelloWorld
  Script        HelloWorld
  Label         None
```

Create a Script

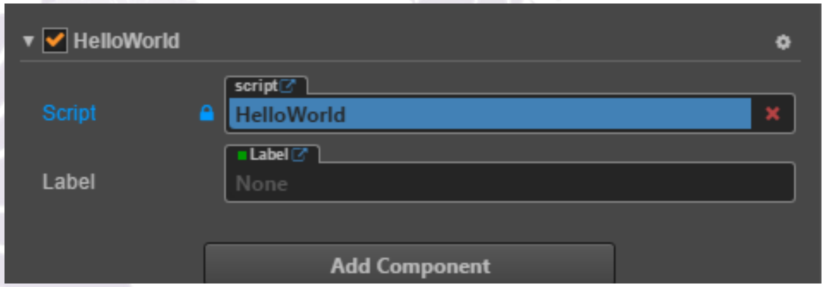
- Assets window → Right click → Create → TypeScript

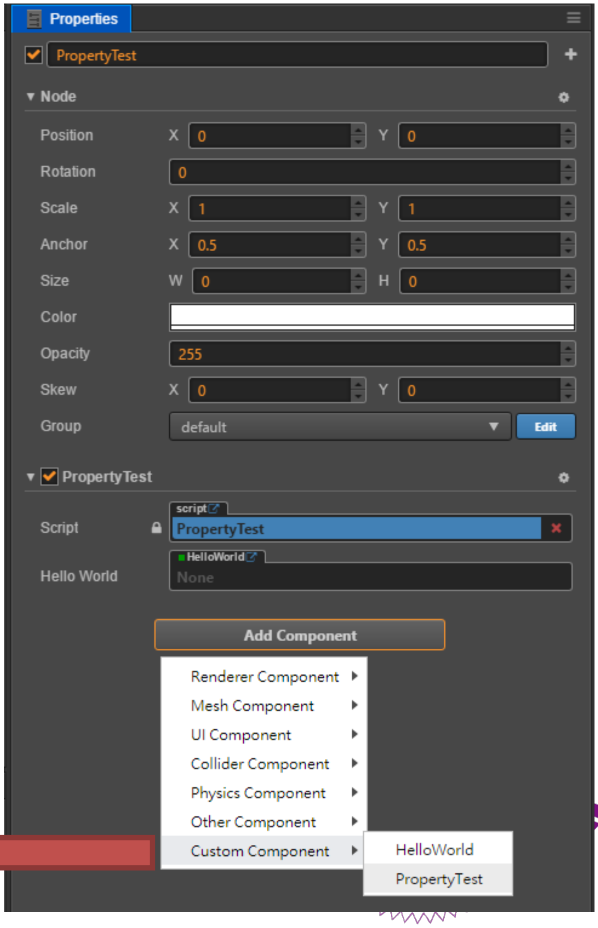


```
1.9.3
preview server
1.9.3
```

Attach a Script to a Node

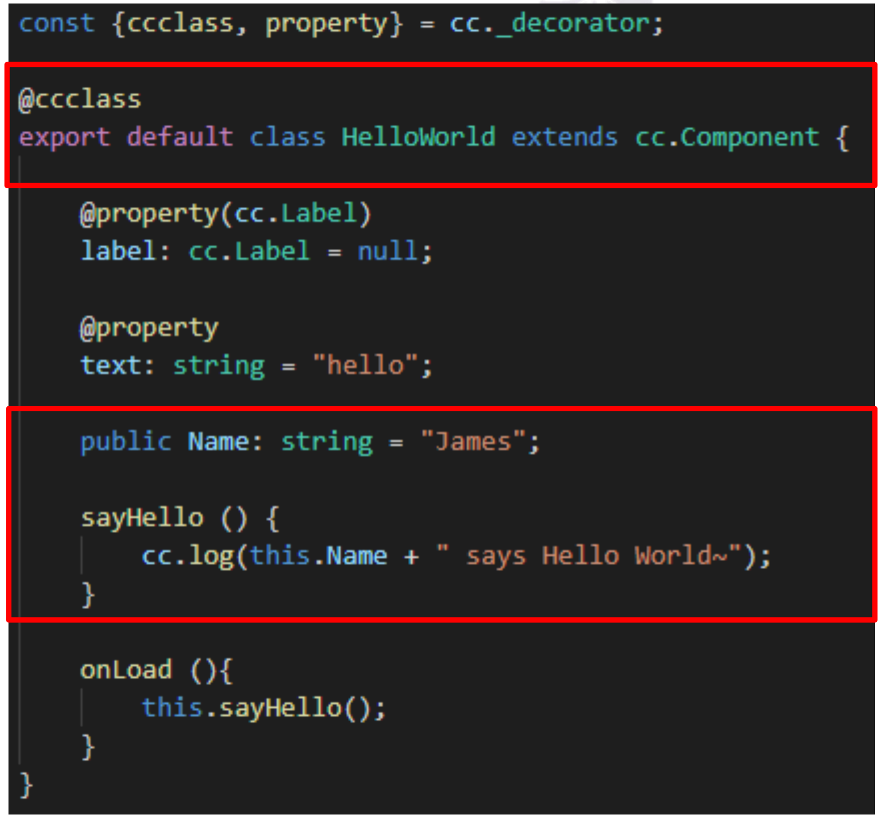
1. Click Node → Properties→ Add Component → Add Custom Component → Choose the script
2. Drag and drop the script to the property field of target node





Script: Structure

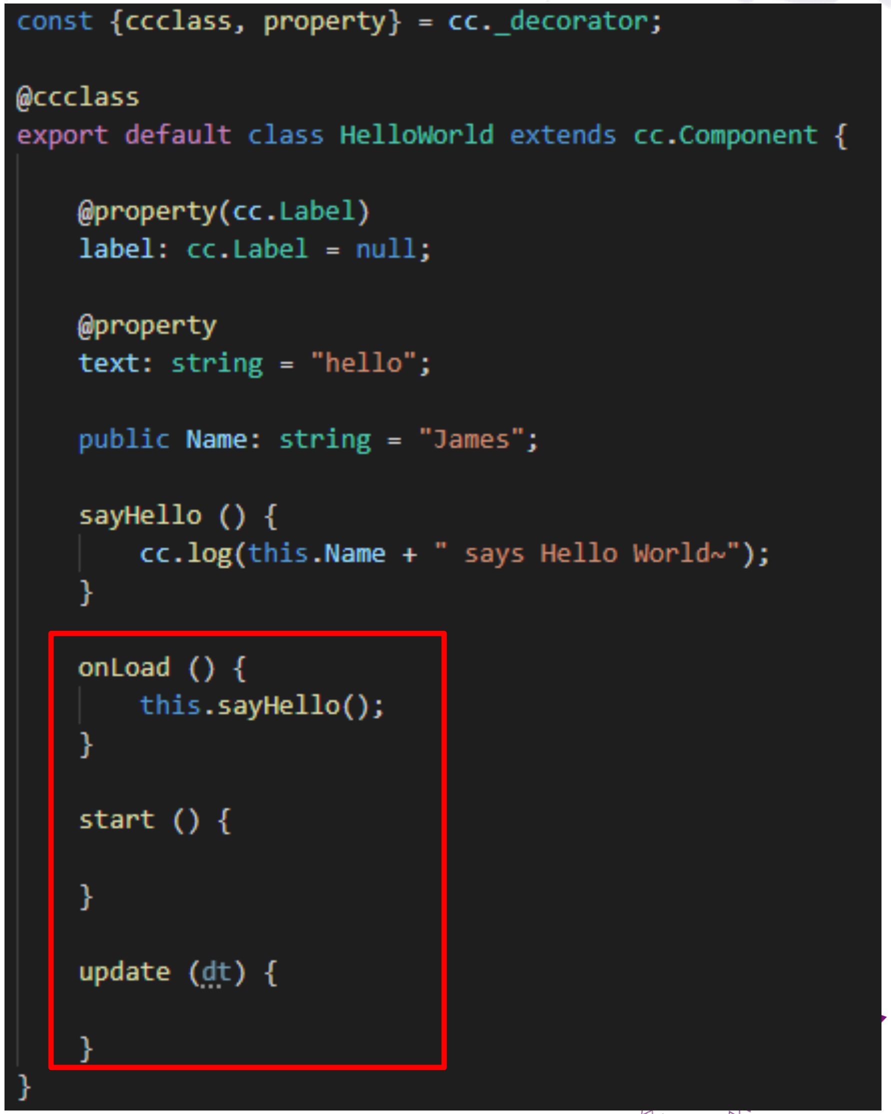
- Class
  - Define your own component
  - Class name is the name of the component
- Member variables and functions
  - Access using "**this**" keyword



```
const {ccclass, property} = cc._decorator;

@ccclass
export default class HelloWorld extends cc.Component {

    @property(cc.Label)
    label: cc.Label = null;

    @property
    text: string = "hello";

    public Name: string = "James";

    sayHello () {
        cc.log(this.Name + " says Hello World~");
    }

    onLoad (){
        this.sayHello();
    }
}
```

Script: Life-Cycle Callbacks

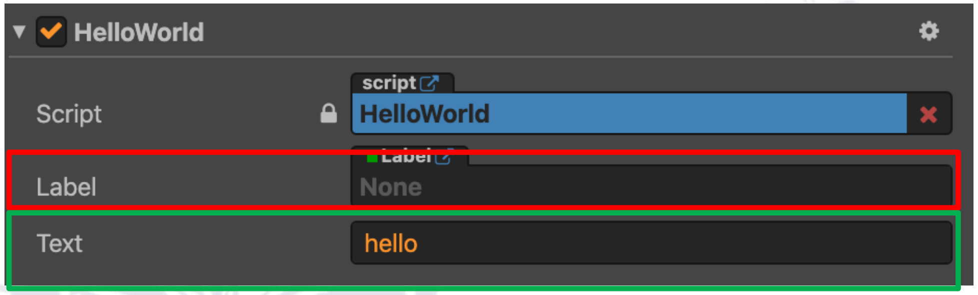
- **onLoad():**
  - Run the code when the game start
- **start():**
  - Run the code after all Component finish onLoad()
- **update():**
  - Like a loop, keep running in the game



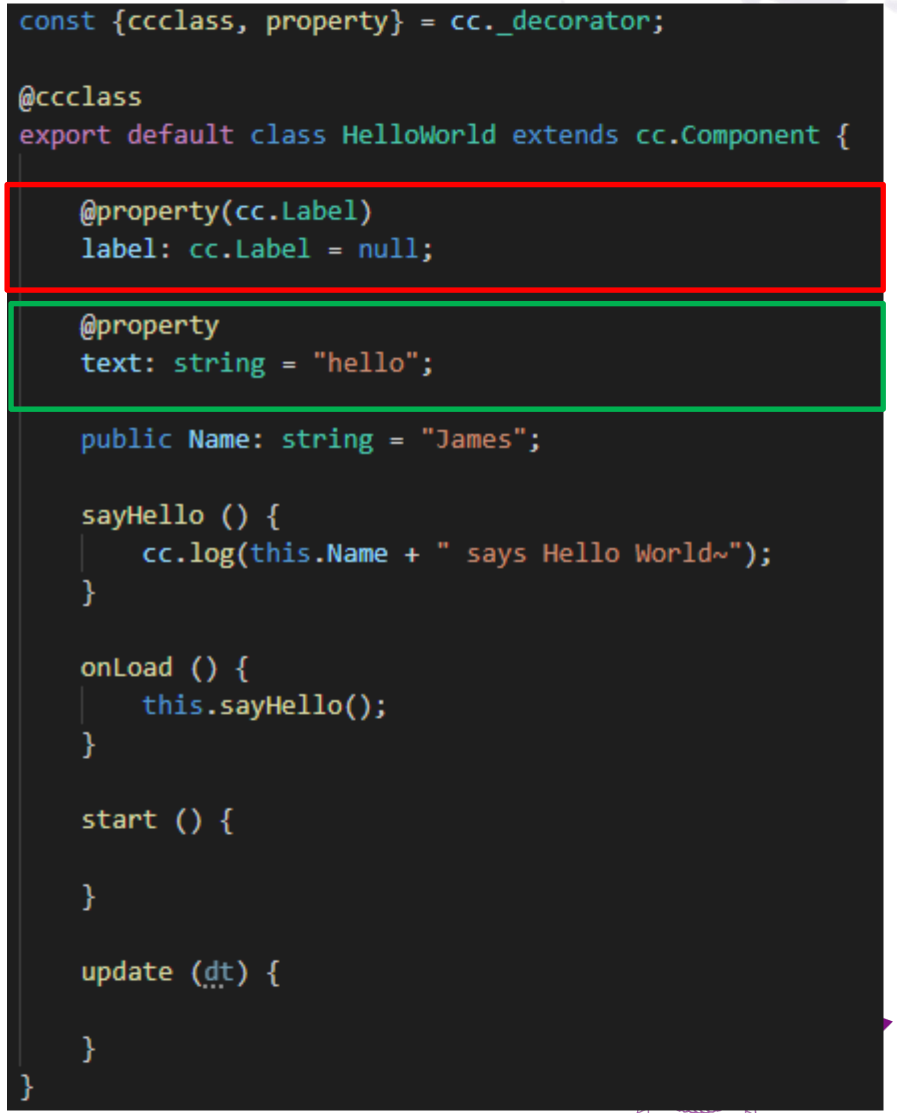
```
const {ccclass, property} = cc._decorator;

@ccclass
export default class HelloWorld extends cc.Component {

    @property(cc.Label)
    label: cc.Label = null;

    @property
    text: string = "hello";

    public Name: string = "James";

    sayHello () {
        cc.log(this.Name + " says Hello World~");
    }

    onLoad () {
        this.sayHello();
    }

    start () {

    }

    update (dt) {

    }
}
```

Script: property

- **@property**
  - Declare variables that are visible in the cocos creator IDE
  - Primitive variable
  - Class component



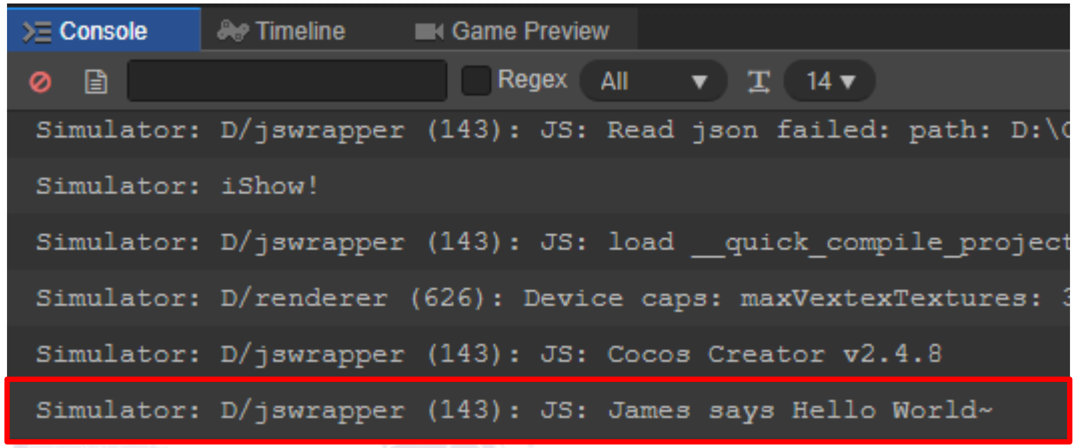
Script	HelloWorld
Label	None
Text	hello



```
const {ccclass, property} = cc._decorator;

@ccclass
export default class HelloWorld extends cc.Component {

    @property(cc.Label)
    label: cc.Label = null;

    @property
    text: string = "hello";

    public Name: string = "James";

    sayHello () {
        cc.log(this.Name + " says Hello World~");
    }

    onLoad () {
        this.sayHello();
    }

    start () {

    }

    update (dt) {

    }
}
```

Run the Script

- Print “name says Hello Word” in web console



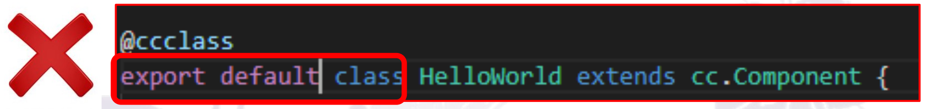
```
Simulator: D/jswrapper (143): JS: Read json failed: path: D:\
Simulator: iShow!
Simulator: D/jswrapper (143): JS: load __quick_compile_project
Simulator: D/renderer (626): Device caps: maxVextexTextures:
Simulator: D/jswrapper (143): JS: Cocos Creator v2.4.8
Simulator: D/jswrapper (143): JS: James says Hello World~
```

Script as a Component

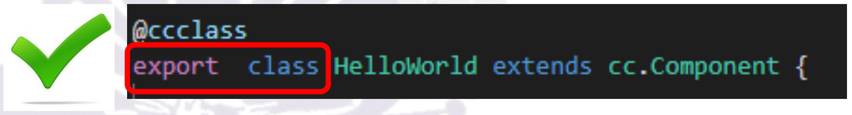
- Create another script named “PropertyTest”
- Import the “HelloWorld” script
- Declare a HelloWorld component property
- Using “this” to access the component
- Drag and drop a Node with HelloWorld component to the property field
- Run the program

Script: import

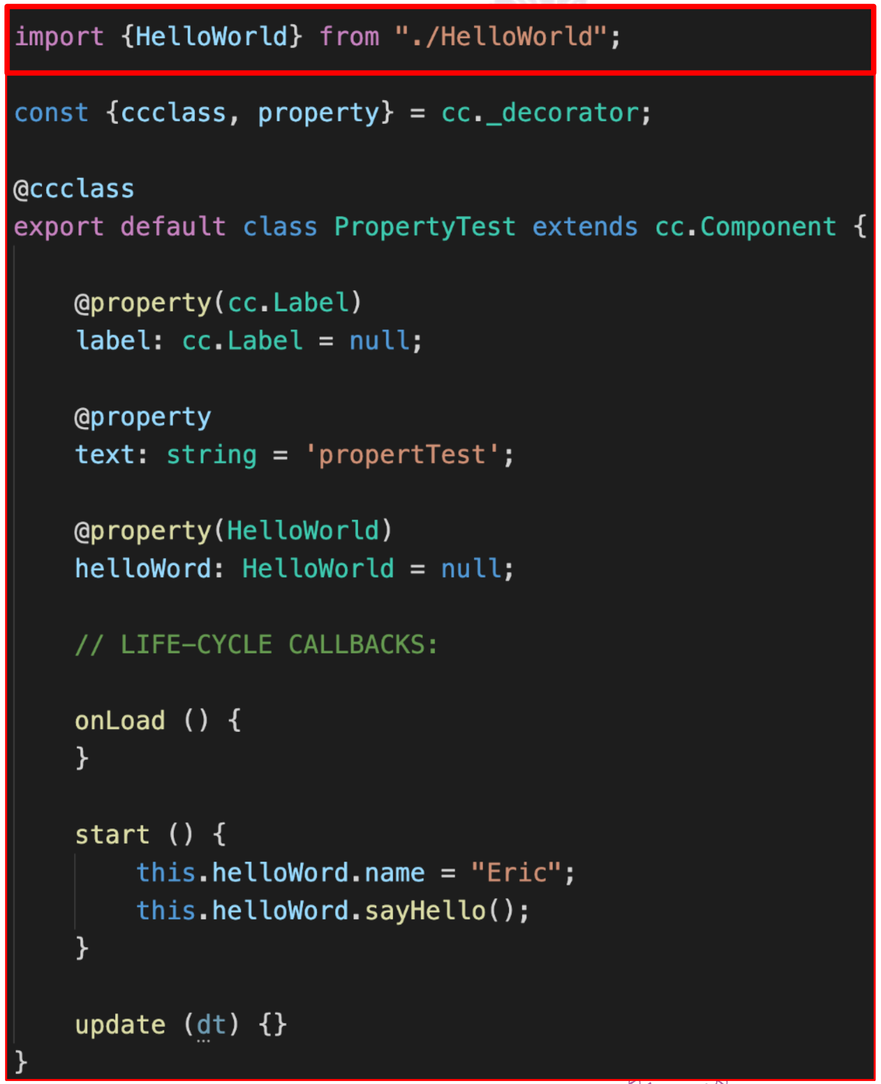
- Like “#include ” in C/C++
- Import different class from another file
- The imported class CANNOT be “export default”
- Import{<ClassName>} from “<PathToTS>”



```
@ccclass
export default class HelloWorld extends cc.Component {
```



```
@ccclass
export  class HelloWorld extends cc.Component {
```



```
import {HelloWorld} from "./HelloWorld";

const {ccclass, property} = cc._decorator;

@ccclass
export default class PropertyTest extends cc.Component {

    @property(cc.Label)
    label: cc.Label = null;

    @property
    text: string = 'propertTest';

    @property(HelloWorld)
    helloWord: HelloWorld = null;

    // LIFE-CYCLE CALLBACKS:

    onLoad () {
    }

    start () {
        this.helloWord.name = "Eric";
        this.helloWord.sayHello();
    }

    update (dt) {}
}
```

Access Node and Component

- Get the node where a component belongs to.
- Get the other component in the same node.
- Setup node and component in **Properties** panel.
- Find child node.
- Find node in global.

Ref: Access node and other component

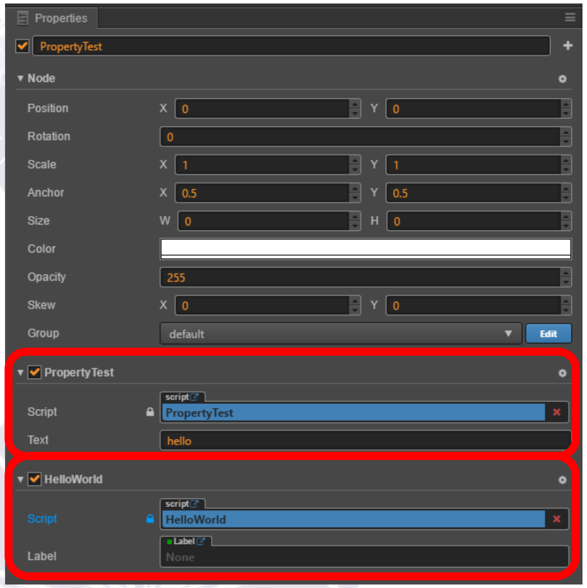
this.node

- Getting the node where the component belongs to using **this.node** variable.

```
const {ccclass, property} = cc._decorator;

@ccclass
export default class ComponentTest extends cc.Component {
  start () {
      this.node.x = 100;
      this.node.y = 100;
  }
}
```

Get the other Component

- Get and access a component in the same Node using **getComponent()**

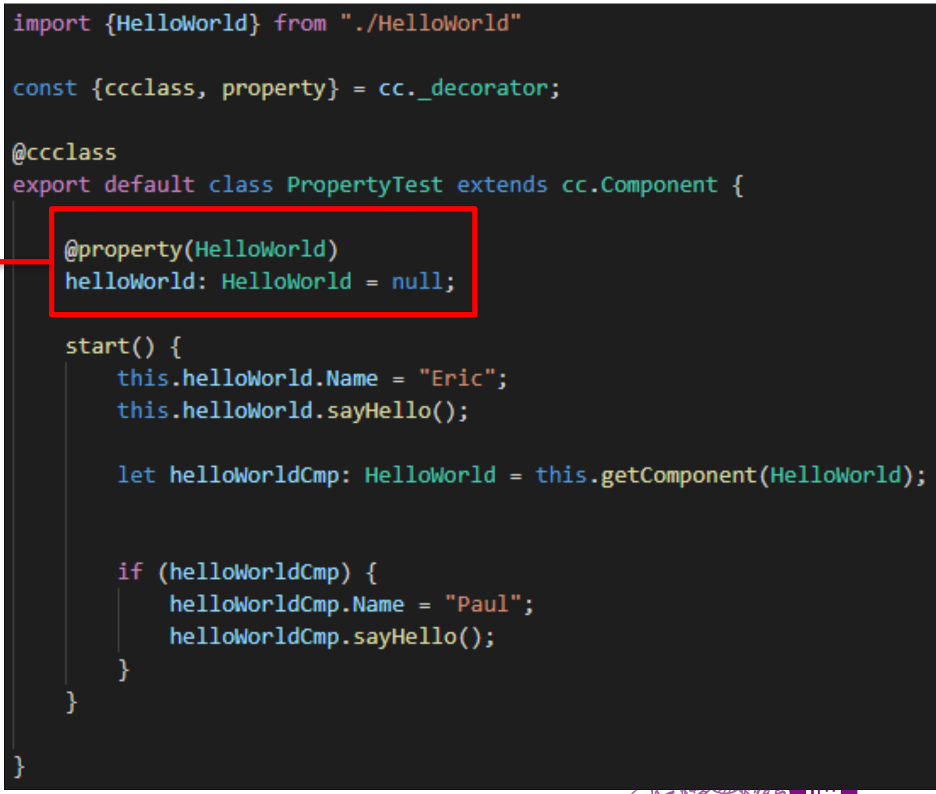


```
Properties
PropertyTest
▼ Node
Position   X 0     Y 0
Rotation   0
Scale      X 1     Y 1
Anchor     X 0.5   Y 0.5
Size       W 0     H 0
Color
Opacity    255
Skew       X 0     Y 0
Group      default   Edit
▼ PropertyTest
Script     script  PropertyTest
Text       hello
▼ HelloWorld
Script     script  HelloWorld
Label      Label   None
```

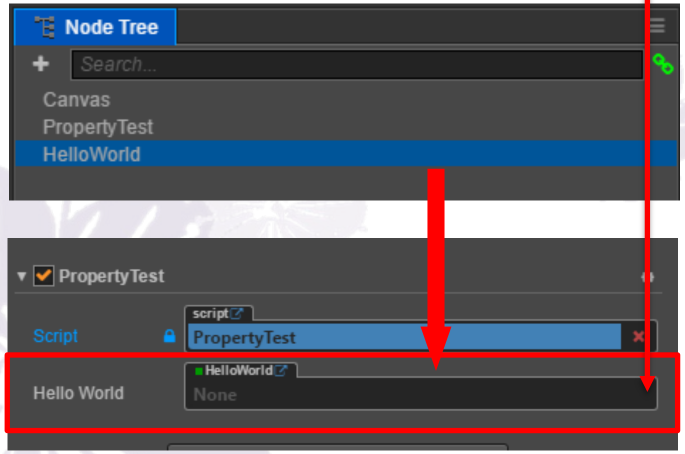
```
import {HelloWorld} from "./HelloWorld"
const {ccclass, property} = cc._decorator;

@ccclass
export default class PropertyTest extends cc.Component {
    @property(HelloWorld)
    helloWord: HelloWorld = null;
    start () {
        this.helloWord.Name = "Eric";
        this.helloWord.sayHello();

        let helloWordCmp: HelloWorld = this.getComponent(HelloWorld);
        if(helloWordCmp) {
                helloWordCmp.Name = "Paul";
                helloWordCmp.sayHello();
        }
    }
}
```

Use Properties Panel to Link Node/Component

- Declare a component using @property
- Access the component using “this”
- Drag and drop a node with specific component



```
import {HelloWorld} from "./HelloWorld"

const {ccclass, property} = cc._decorator;

@ccclass
export default class PropertyTest extends cc.Component {

    @property(HelloWorld)
    helloWorld: HelloWorld = null;

    start() {
        this.helloWorld.Name = "Eric";
        this.helloWorld.sayHello();

        let helloWorldCmp: HelloWorld = this.getComponent(HelloWorld);


        if (helloWorldCmp) {
            helloWorldCmp.Name = "Paul";
            helloWorldCmp.sayHello();
        }
    }

}
```



Find Child Node

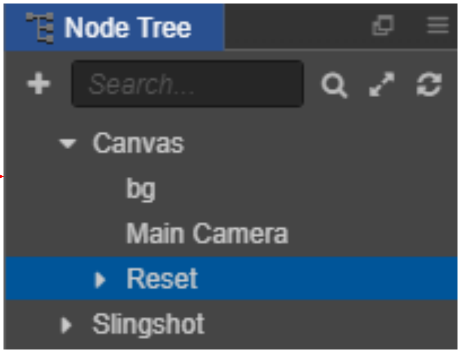
- Using the variable "**this.node.children**" to get the children nodes.
- Using **getChildByName(string)** to get a specific child node.

```
const {ccclass, property} = cc._decorator;

@ccclass
export default class PropertyTest extends cc.Component {
  start () {
      let childNode : any = this.node.children;
      cc.log (this.node.getChildByName("HelloWorld").text);
  }
}
```

Find Child Node (Cont'd)

- If the child's hierarchy is too deep, use **cc.find(path, referenceNode)** to find and get a node step by step based on the path passed into it.
- If the second parameter is not specified, they system will search from the scene root

```
findResetBtn () {
    …
    cc.find("Canvas/Reset");
}
```



Interact with Node

- In game playing, interaction is necessary
- Keyboard, mouse, joystick…

Register Mouse Events

- Register mouse events by “on”

```
start () {
  //the callback function when the mouse clicks down on the Node
  this.node.on(cc.Node.EventType.MOUSE_DOWN, function(event){
                        cc.log("Mouse down"); }, this);
  //the callback function when the mouse enters the Node
  this.node.on(cc.Node.EventType.MOUSE_ENTER, function(event){
                        cc.log("Mouse enter"); }, this);
  //the callback function when the mouse moves on the Node
  this.node.on(cc.Node.EventType.MOUSE_MOVE, function(event){
                        cc.log("Mouse move");}, this);
  //the callback function when the mouse finishes click on the Node
  this.node.on(cc.Node.EventType.MOUSE_UP, function(event){
                        cc.log("Mouse up");}, this);
}
```

Node.EventType

Register Keyboard Events

- Register keyboard events by “on”

```
start () {
   // add key down and key up events
   cc.systemEvent.on(cc.SystemEvent.EventType.KEY_DOWN, this.onKeyDown, this);
   cc.systemEvent.on(cc.SystemEvent.EventType.KEY_UP, this.onKeyUp, this);
}

onKeyDown(event){
   cc.log("Key Down: " + event.keyCode);
   if(event.keyCode==cc.macro.KEY.space){
         if(this.getComponent(cc.AudioSource).isPlaying){
                this.getComponent(cc.AudioSource).pause();
         }else{
                his.getComponent(cc.AudioSource).play();
          }
    }
}
```

SystemEvent.EventType
macro.KEY

Unregister Events

- Unregister events by “off”

```
start () {
   // add key down and key up events
   cc.systemEvent.on(cc.SystemEvent.EventType.KEY_DOWN, this.onKeyDown, this);
   cc.systemEvent.on(cc.SystemEvent.EventType.KEY_UP, this.onKeyUp, this);
}

onKeyDown(event){
    cc.log("Key Down: " + event.keyCode);
    cc.systemEvent.off(cc.SystemEvent.EventType.KEY_DOWN, this.onKeyDown,
this);
}

onKeyUp(event){
    cc.log("Key Up: " + event.keyCode);
    cc.systemEvent.off(cc.SystemEvent.EventType.KEY_UP, this.onKeyUp, this);
}
```

How to Move the Node?

Register the Touch Events

- Register touch events by “on”

```
onEnable () {
  this.node.on(cc.Node.EventType.TOUCH_START, this._onTouchBegan, this);
  this.node.on(cc.Node.EventType.TOUCH_MOVE, this._onTouchMove, this);
  this.node.on(cc.Node.EventType.TOUCH_END, this._onTouchEnded, this);
  this.node.on(cc.Node.EventType.TOUCH_CANCEL, this._onTouchEnded, this);
}
_onTouchBegan (event) { …
}
_onTouchMove (event) { …
}
_onTouchEnded (event) { …
}
_onTouchBegan (event) { …
}
```

[Node.EventType]()

Move the Node

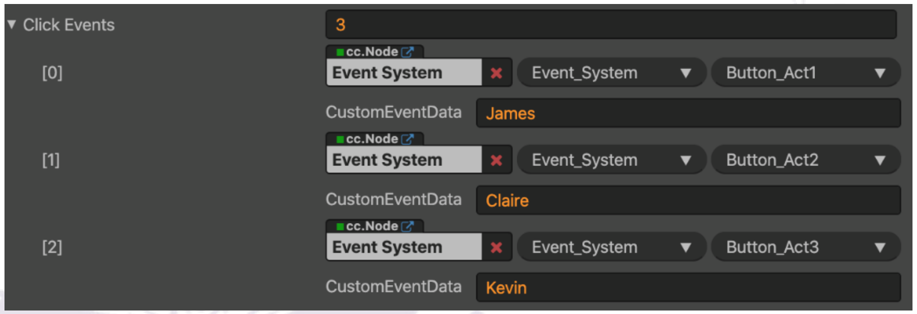
- Use .setPosition(cc.Vec2) to move a node

```
_onTouchMove (event) {
    …
    if(this.draggable){
        let start = event.getStartLocation();
        let cur = event.getLocation();

        cur.subSelf(start); // cur = cur – start, equals to cur = cur.sub(start)

        // Sets the position of the node in its parent's coordinates.
        this.node.setPosition(this.startPos.add(cur_v));
        …
    }
    event.stopPropagation();  // Stop propagating event to parent node.
}
```

Node System Events

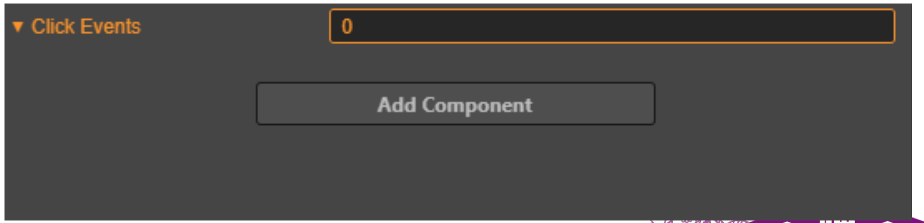
Dynamic Event Binding

- Using reference to listen to events could be tedious and lacks flexibility
- Using dynamic event binding will ease the above burden!
  - The specification of event handlers are created and bound at **run-time**!



```
▼ Click Events          3
  [0]   cc.Node  Event System  ✖  Event_System ▼  Button_Act1 ▼
        CustomEventData  James
  [1]   cc.Node  Event System  ✖  Event_System ▼  Button_Act2 ▼
        CustomEventData  Claire
  [2]   cc.Node  Event System  ✖  Event_System ▼  Button_Act3 ▼
        CustomEventData  Kevin
```

Conventional event handling

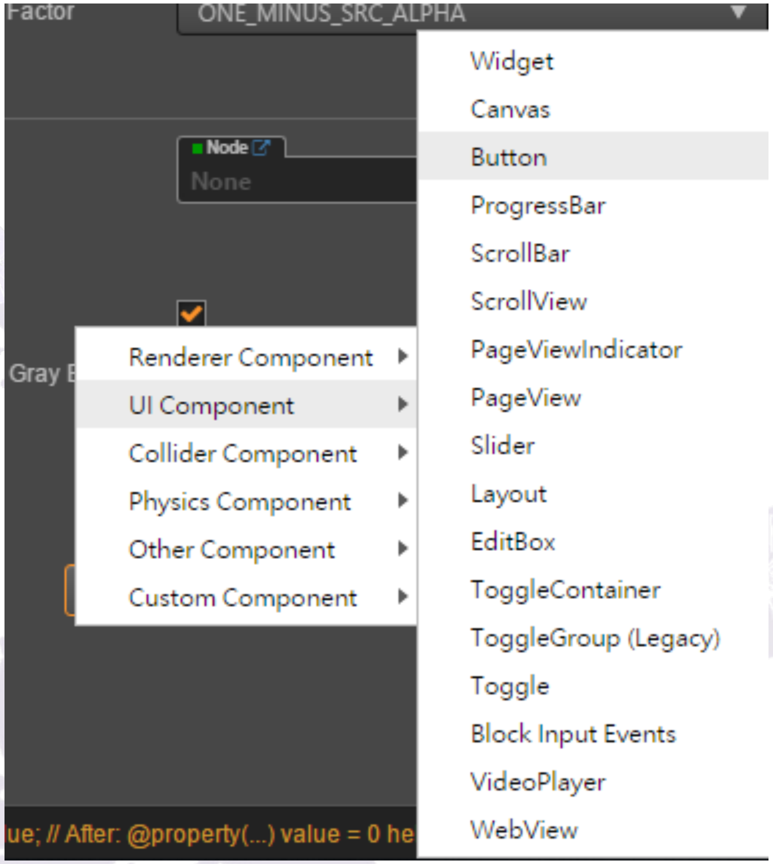


```
▼ Click Events          0
            Add Component
```

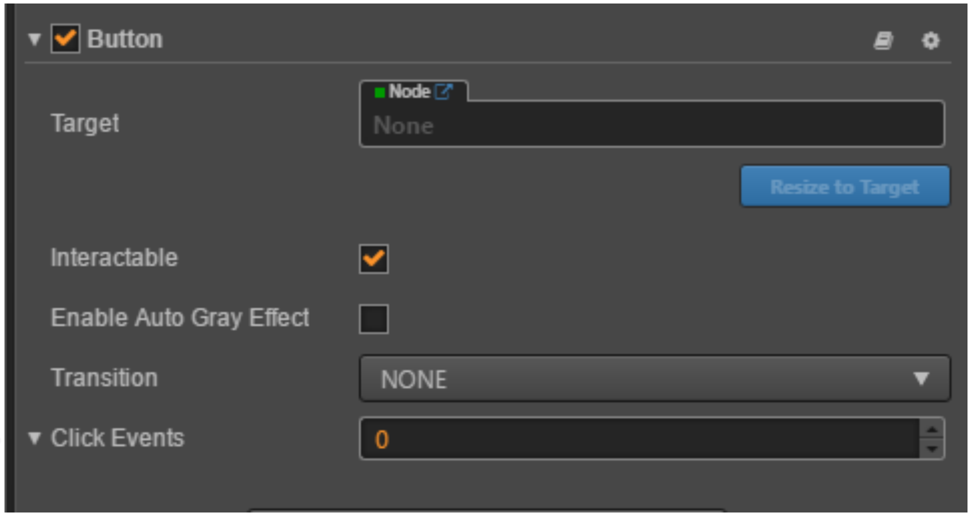
Using dynamic event binding

Example: Button Click Event

- Add a Button Component



```
ue; // After: @property(...) value = 0 he
```

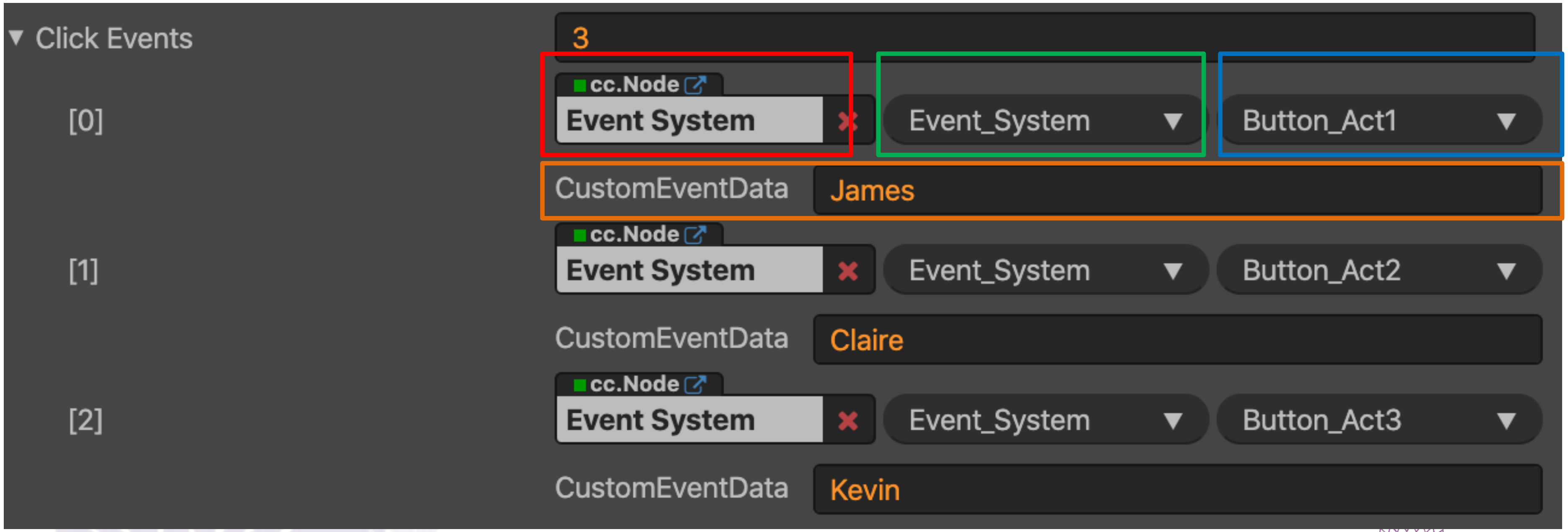


Button

Target	Node None
	Resize to Target
Interactable	✓
Enable Auto Gray Effect	
Transition	NONE
Click Events	0

Specify via Reference

- Specify reference **node** -> **component** -> **handler function** -> **custom event msg**



▼ Click Events 3

[0]	cc.Node	Event System	Event_System ▼	Button_Act1 ▼
	CustomEventData	James		
[1]	cc.Node	Event System	Event_System ▼	Button_Act2 ▼
	CustomEventData	Claire		
[2]	cc.Node	Event System	Event_System ▼	Button_Act3 ▼
	CustomEventData	Kevin		

cc.Component.EventHandler

properties	type	Description
target	Node	the node that contains target callback
component	String	name of the component (i.e., script) that contains target callback
handler	String	Event handler, such as function's name ‘Button_Act#' in example
customEventData	String	Custom Event Data

Ref: Component.EventHandler

Advanced: Dynamic Click Event

- Create a script

```
Button_Init() {
        let button_Act1 = new cc.Component.EventHandler();
        button_Act1.target = this.node;
        button_Act1.component = "Event_System";
        button_Act1.handler = "Button_Act1";
        button_Act1.customEventData = "James";

        cc.find("New Button").getComponent(cc.Button).clickEvents.push(button_Act1);
        ….
}
Button_Act1 (event, customEventData) {
    // do something here
}
```

thank you!

Question