Electrical Circuit Controller

Reference Guide

Version 1.0

John Hnidec Software

johnhnidecsoftware.com

john.hnidec+unity@gmail.com

Electrical Circuit Controller

Sincere thanks for your purchase of Electrical Circuit Controller (ECC) for Unity. We are a small software development team here in Australia and it is only with your valued support and patronage that we can continue to enhance and expand our line of products. We genuinely appreciate your interest and of course are happy to assist in any way we can.

Introduction

So then, what is ECC? It is a set of simple yet extremely powerful classes, thoughtfully crafted to deliver some very deep capability with regard to creating, controlling and switching any kind of electrical network. By simply plugging in any of the supplied Scriptable Objects you can easily construct electrical networks complete with Power Supplies, Circuits and Power Consumers (more on these later). By its very nature ECC is highly configurable, so swapping out a AAA battery Power Supply and replacing it with a car battery is literally a line of code (or two clicks in the custom editor).

Some good news too, you absolutely don't need to know anything about how power circuits work, what a Watt is, what the difference between a Volt and an Amp is and so on. All of that complexity is hidden away (as it should be), but if you do know about such things and you know who George Ohm is, then that's great too!

It's also important to note what ECC is not. It is not a simulator of electronics and it does not contain any graphical assets. Some extremely complicated electrical concepts are simplified here, for example the Loss property of a circuit is expressed as a simple percentage but in the real world that value would depend on an enormous number of factors and conditions.

Possibly the above explanation of what ECC is all about still has you scratching your head. By way of examples then, here are some of the (unlimited) use-cases that ECC could be applied to…

FLASHLIGHT NETWORK

Technically speaking (not visually or graphically) what makes up a flashlight? It needs a power source, a switch, a bulb of some sort and maybe some small microprocessor. If your game includes a flashlight, then you can simply create an electrical network for it and instantly have access to how much power is left in the batteries, is it switched on or not, does the little microprocessor draw power even when the flashlight is off, and so on and so on.

HOUSEHOLD NETWORK

A house is very different to a flashlight, but ECC can create an electrical network for it just as easily as for a flashlight. A house also needs a power supply, some circuits and some consumers of that power. In this example we would use a 240 V power supply (at least here in Australia anyway), perhaps two or three circuits (maybe each circuit has different amperage), and a bunch of appliances like ovens, toasters, hair dryers, washing machines and so on. Believe it or not, setting all that up in ECC is quick and easy, and once done your game would have fully access to all the Amps, Watts, switches, circuit load and more for the entire household.

CITY NETWORK

What the heck, how can ECC manage something that large? Well, each module in ECC (more on modules later) can have multiple submodules each doing its own thing. A city network then could be thought of as lots of houses (or factories and buildings), all on different power supplies, all with differing power requirements, but still all contained within a single larger umbrella network. Your game could then intelligently manage power to each of these city blocks, houses and television screens.

Remember too, each of these examples are easily configured inside the custom ECC editor.

What is included with ECC?

A set of scripts located in the ECC Scripts folder that contain the core logic and functionality that enables ECC to do its thing.

A set of modules located in the ECC Modules folder that contain many pre-configured Scriptable Objects. These modules are what drives the logic and operation of ECC and include Networks, Power Supplies, Circuits and Power Consumer modules. It is very straightforward to create your own modules or to simply change the ones that are already there.

A custom editor for configuring your network inside the Unity editor. Easily click to add and delete any of the various modules and sub-modules.

A simple demo project showcasing some of the ECC fundamentals.

Full source code is included, use to understand how it all works or to modify and enhance to suit your own requirements.

This Reference Guide, lovingly written and hand-crafted just for you :)

What are Modules?

We've referred to modules several times already in this document, so now is the time to explain them fully.

ECC is hierarchical in nature, and each level of the hierarchy is a module. The top level of this structure is one or more Networks. A Network has one or more Power Sources. A Power Source has one or more Circuits. A Circuit has one or more Power Consumers.

When all modules are logically joined together a fully operational electrical network is created.

Each module is a Unity Scriptable Object and therefore swapping out a Power Source with a different one, or adding a new Power Consumer to your network can be simply configured in the editor, or via code if need be.

NETWORK MODULE

This is the overarching container for everything else. In and of itself it doesn't do too much but it does end up holding all of the other functionality. Examples of a Network would include a flashlight, a house, a digital watch, a small village, a factory, a computer and so on.

POWER SOURCE MODULE

This is the thing that supplies power to your network. Each Power Source can be configured with Volts and Amps (among other things). Examples of a Power Source would include AA battery, AAA battery, 2 x D Cell batteries in Series, 120 Volt AC, solar panel and so on.

CIRCUIT MODULE

A circuit is what distributes the power (the Power Source) to all of the devices on the circuit. A circuit has Loss and an Amperage Rating (how much load the circuit can provide). If the load on the Circuit is exceeded it will switch itself off, a bit like a circuit breaker in a house does. Examples of a Circuit would include a 20 Amp domestic circuit, 100 Amp industrial circuit, 500 milliamp usb charger circuit and so on.

POWER CONSUMER MODULE

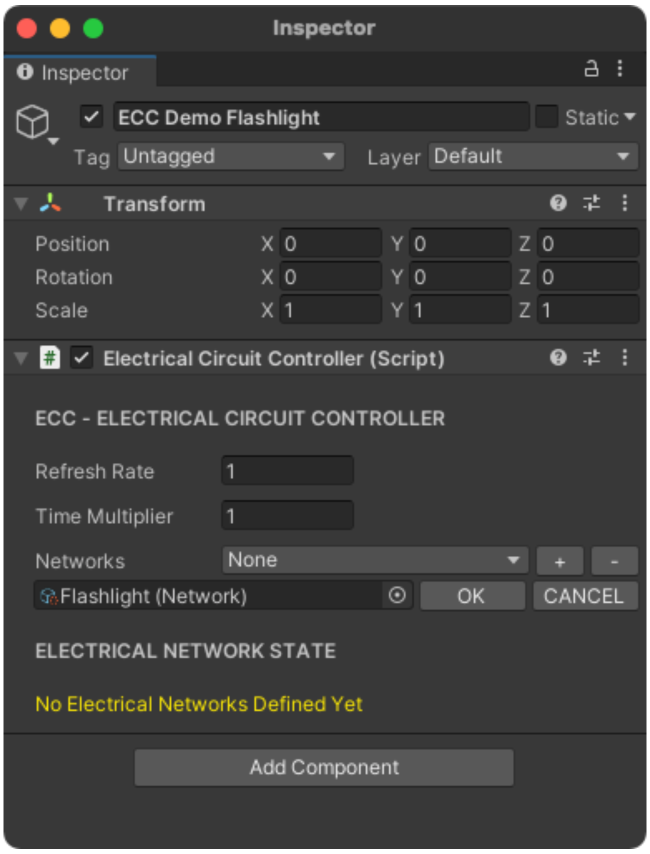
The end of the road is the device on the circuit, the Power Consumer. It is the entity that consumes the power and generally is the object of interest in your game. A Power Consumer has Watts and Parasitic Drain (how much power is consumed even when the device is off). Examples of a Power Consumer would include a lamp, a kettle, a microprocessor, an LED bulb, a huge factory machine, or anything that consumes power really.

Getting Started

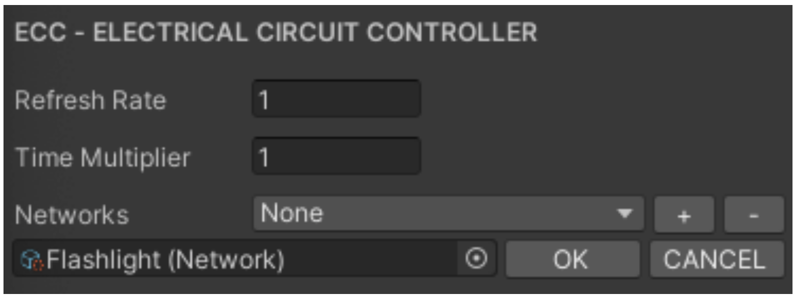
Let's work through each step of creating a Flashlight. We'll be using the custom Unity editor and we'll be using existing modules that come with ECC. At the end of this you'll see your Flashlight network inside the editor and you will be able to switch it on or off and observe its power usage.

Create an empty GameObject in a new or existing project.

Click the Add Component button and add Electrical Circuit Controller script. You should now see the custom editor ready to configure your network.

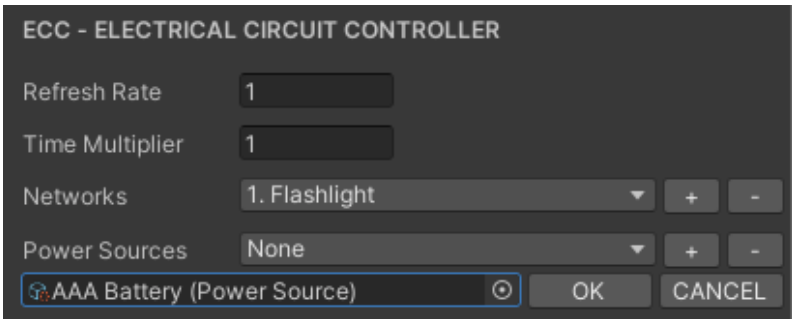


Click the little plus sign next to Networks and choose the Flashlight network from the Unity field.


ECC - ELECTRICAL CIRCUIT CONTROLLER
Refresh Rate 1
Time Multiplier 1
Networks None + -
Flashlight (Network) OK CANCEL

Click the OK button, then select the Flashlight you just added from the dropdown Networks field.

Now that a Network has been added, you will be able to see the Power Sources field. Click the little plus sign next to Power Sources and choose AAA Battery.



ECC - ELECTRICAL CIRCUIT CONTROLLER

Refresh Rate: 1

Time Multiplier: 1

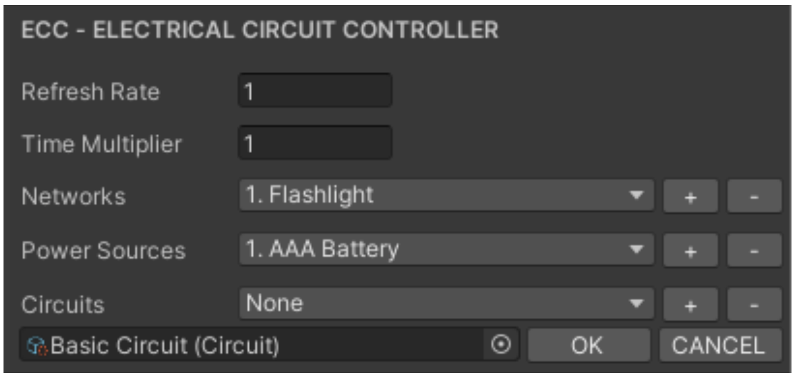
Networks: 1. Flashlight

Power Sources: None

AAA Battery (Power Source)

Click the OK button, then select the AAA Battery you just added form the dropdown Power Sources field.

Now that a Power Source has been added, you will be able to see the Circuits field. Click the little plus sign next to Circuits and choose Basic Circuit.



ECC - ELECTRICAL CIRCUIT CONTROLLER

Refresh Rate: 1

Time Multiplier: 1

Networks: 1. Flashlight

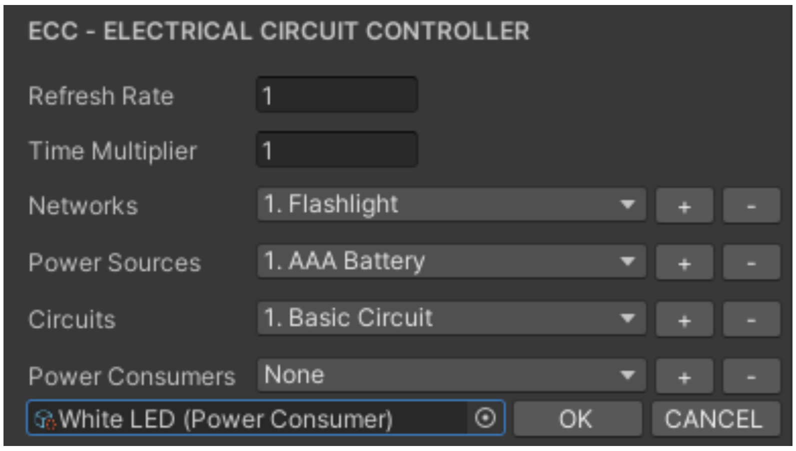
Power Sources: 1. AAA Battery

Circuits: None

Basic Circuit (Circuit)

Click the OK button, then select the Basic Circuit you just added form the dropdown Circuits field.

Now that a Circuit has been added you will be able to see the Power Consumers field. Click the little plus sign next to Power Consumers and choose White LED.



ECC - ELECTRICAL CIRCUIT CONTROLLER

Refresh Rate: 1

Time Multiplier: 1

Networks: 1. Flashlight

Power Sources: 1. AAA Battery

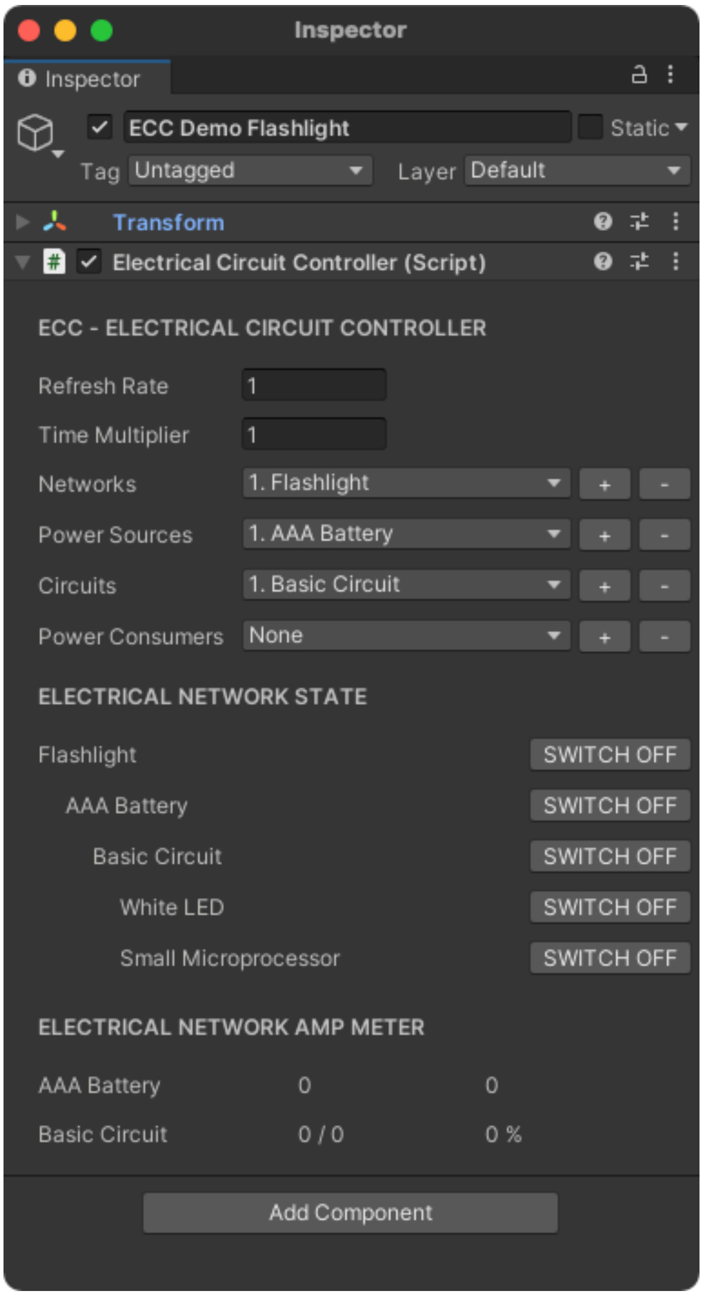
Circuits: 1. Basic Circuit

Power Consumers: None

White LED (Power Consumer)

Click the OK button and repeat the above step, this time adding a Small Microprocessor Power Consumer.

All going well, that's it, you have now created a fully working Flashlight Network. Let's check and make sure you're all good, and quickly summarise what we've just done. Make sure your Network looks like this:



Inspector

ECC Demo Flashlight

Tag Untagged   Layer Default

Transform

Electrical Circuit Controller (Script)

ECC - ELECTRICAL CIRCUIT CONTROLLER

Refresh Rate: 1
Time Multiplier: 1
Networks: 1. Flashlight
Power Sources: 1. AAA Battery
Circuits: 1. Basic Circuit
Power Consumers: None

ELECTRICAL NETWORK STATE

Flashlight — SWITCH OFF
AAA Battery — SWITCH OFF
Basic Circuit — SWITCH OFF
White LED — SWITCH OFF
Small Microprocessor — SWITCH OFF

ELECTRICAL NETWORK AMP METER

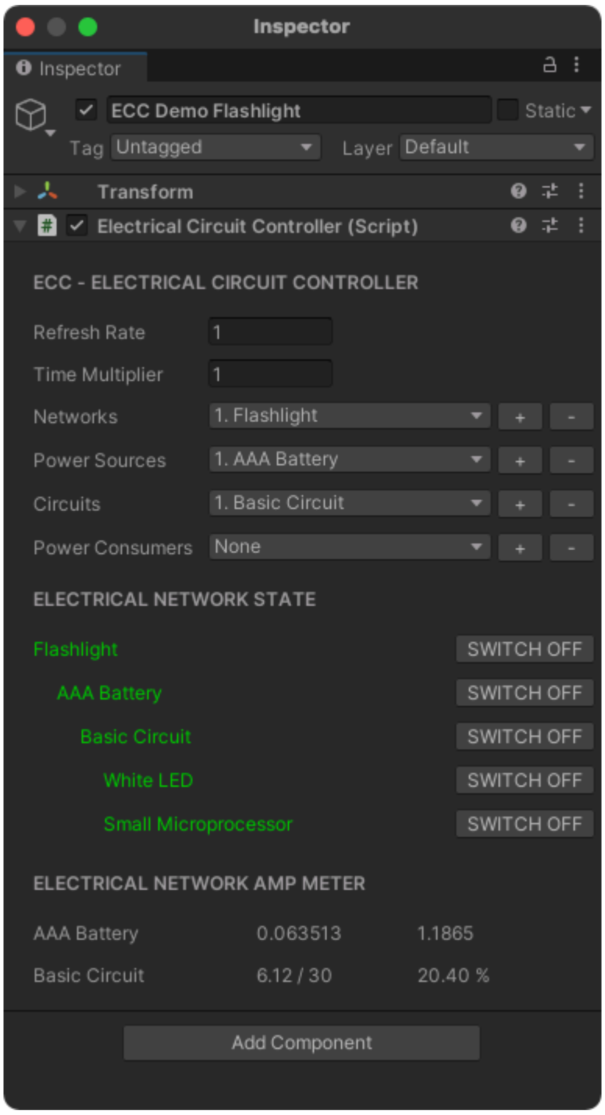
AAA Battery	0	0
Basic Circuit	0 / 0	0 %

Add Component

We we've got a Network called Flashlight, it contains a AAA Battery Power Source, with one Circuit, a White LED and a Small Microprocessor.

You can see a summary of your Network in the custom editor, and right now you can also switch the various modules OFF and ON. Run your game now and see what happens.

You should all modules switched on and running, as indicated by the green highlighting. At the very bottom of the display you can see the drain on the battery and the load on the circuit. Try switching some of the modules OFF and see what happens.


Inspector

Inspector

ECC Demo Flashlight — Static
Tag: Untagged — Layer: Default

Transform

Electrical Circuit Controller (Script)

ECC - ELECTRICAL CIRCUIT CONTROLLER

Refresh Rate: 1
Time Multiplier: 1
Networks: 1. Flashlight
Power Sources: 1. AAA Battery
Circuits: 1. Basic Circuit
Power Consumers: None

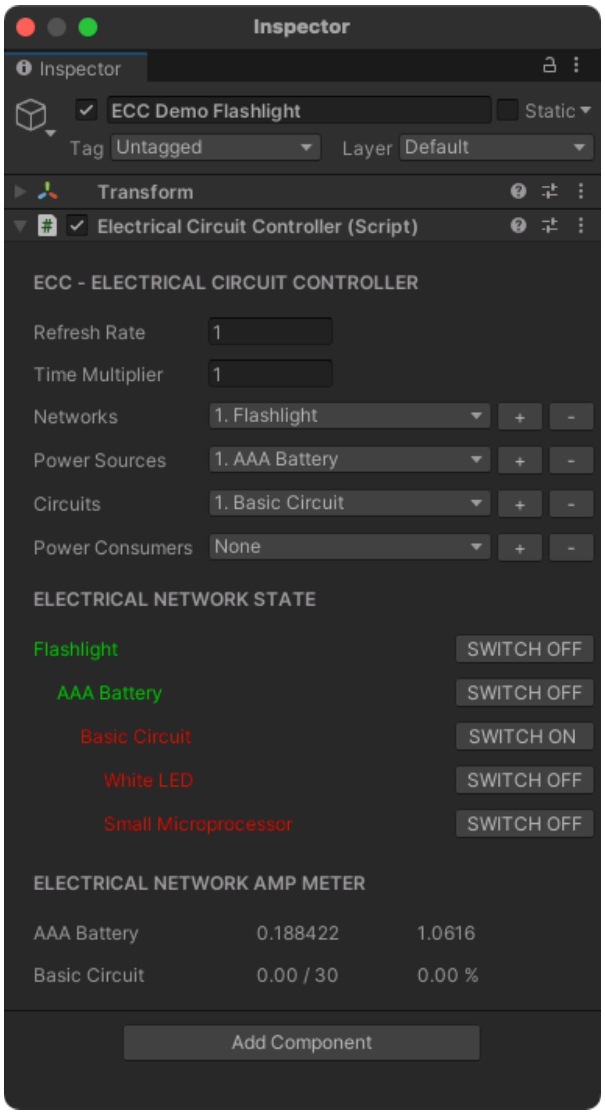
ELECTRICAL NETWORK STATE

Flashlight — SWITCH OFF
AAA Battery — SWITCH OFF
Basic Circuit — SWITCH OFF
White LED — SWITCH OFF
Small Microprocessor — SWITCH OFF

ELECTRICAL NETWORK AMP METER

```
AAA Battery      0.063513     1.1865
Basic Circuit    6.12 / 30    20.40 %
```

Add Component


Inspector

Inspector

ECC Demo Flashlight — Static
Tag: Untagged — Layer: Default

Transform

Electrical Circuit Controller (Script)

ECC - ELECTRICAL CIRCUIT CONTROLLER

Refresh Rate: 1
Time Multiplier: 1
Networks: 1. Flashlight
Power Sources: 1. AAA Battery
Circuits: 1. Basic Circuit
Power Consumers: None

ELECTRICAL NETWORK STATE

Flashlight — SWITCH OFF
AAA Battery — SWITCH OFF
Basic Circuit — SWITCH ON
White LED — SWITCH OFF
Small Microprocessor — SWITCH OFF

ELECTRICAL NETWORK AMP METER

```
AAA Battery      0.188422     1.0616
Basic Circuit    0.00 / 30    0.00 %
```

Add Component

Notice that turning the Basic Circuit OFF also turned off the White LED and Microprocessor even though they are both switched ON. Well, that’s because they’re not ON because there’s no power on the circuit. Think of a lamp on your desk, it might be switched on but if it’s not plugged in it won’t work.

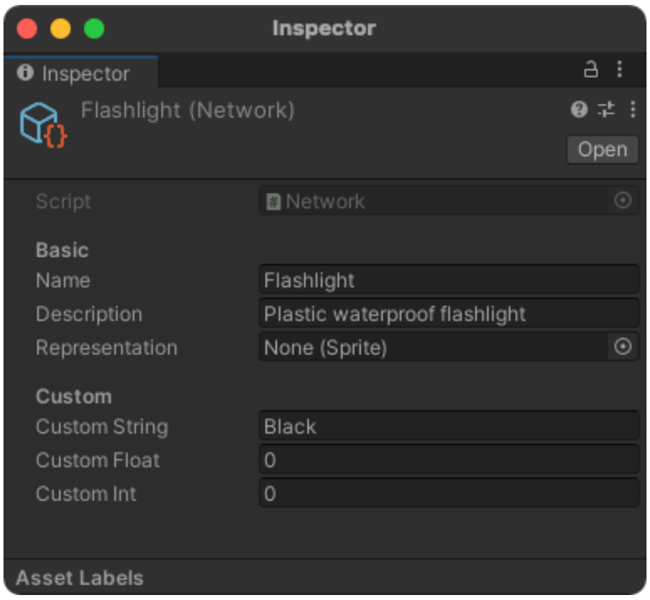
Interestingly, if you turn everything on and then turn off the White LED and the Small Microprocessor you will still see some usage on the Network. Why is this? Well, our Small Microprocessor was configured with Parasitic Drainage which means it will still use power even when it’s switched off. Think of your television, even when you switch it off it is still consuming power.

Scriptable Objects - The Heart of Everything

If you're wondering what on earth a Scriptable Object is then it's probably worthwhile having a snoop around the Unity help and forums for more information. You don't really need to understand them to use ECC though, other than to say that all the modules you used in the previous example (the Network, the AAA Battery, the Circuit, the White LED and the Small Microprocessor) are all Scriptable Objects, located in the **ECC/Modules** folder.

You can create your own brand new Scriptable Objects (Modules) based on your own requirements. For example you could copy the AAA Battery module, rename it to D Cell, change the Voltage or Amps value, and as simple as that you've now got a brand new Power Source.

NETWORK SCRIPTABLE OBJECT



Flashlight (Network)

Script	Network
Basic	
Name	Flashlight
Description	Plastic waterproof flashlight
Representation	None (Sprite)
Custom	
Custom String	Black
Custom Float	0
Custom Int	0

Name: Meaningful name for the object

Description: Meaningful description for the object

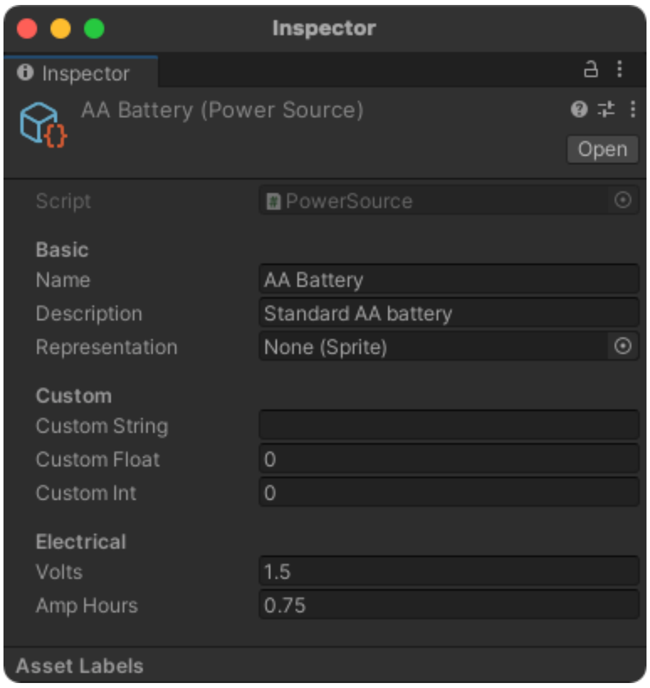
Representation: Sprite image for the object

Custom String: Metadata string field used to represent whatever you want.

Custom Float: Metadata float field used to represent whatever you want.

Custom Int: Metadata integer field used to represent whatever you want.

POWER SOURCE SCRIPTABLE OBJECT



AA Battery (Power Source)

Script	PowerSource
Basic	
Name	AA Battery
Description	Standard AA battery
Representation	None (Sprite)
Custom	
Custom String	
Custom Float	0
Custom Int	0
Electrical	
Volts	1.5
Amp Hours	0.75

Name: Meaningful name for the object

Description: Meaningful description for the object

Representation: Sprite image for the object

Custom String: Metadata string field used to represent whatever you want.

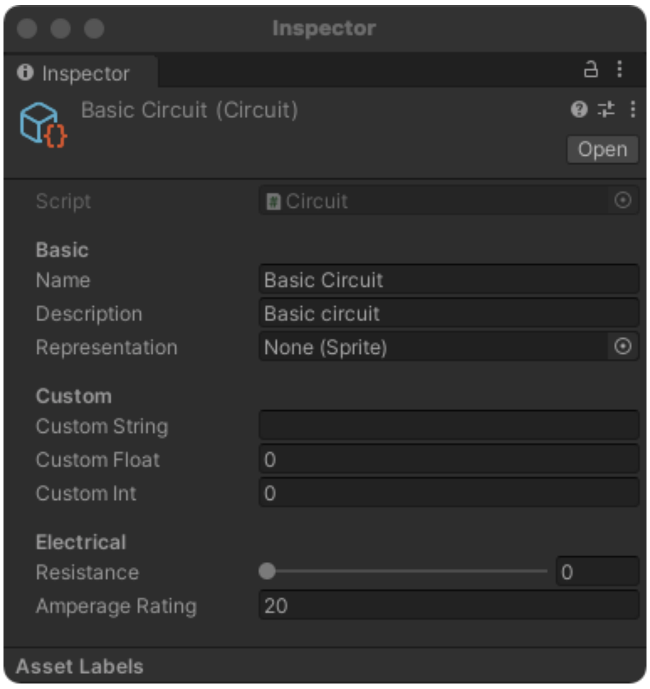
Custom Float: Metadata float field used to represent whatever you want.

Custom Int: Metadata integer field used to represent whatever you want.

Volts: Voltage of the Power Source

Amp Hours: Capacity of the Power Source

CIRCUIT SCRIPTABLE OBJECT



Basic Circuit (Circuit)

Script	Circuit
Basic	
Name	Basic Circuit
Description	Basic circuit
Representation	None (Sprite)
Custom	
Custom String	
Custom Float	0
Custom Int	0
Electrical	
Resistance	0
Amperage Rating	20

Asset Labels

Name: Meaningful name for the object

Description: Meaningful description for the object

Representation: Sprite image for the object

Custom String: Metadata string field used to represent whatever you want.

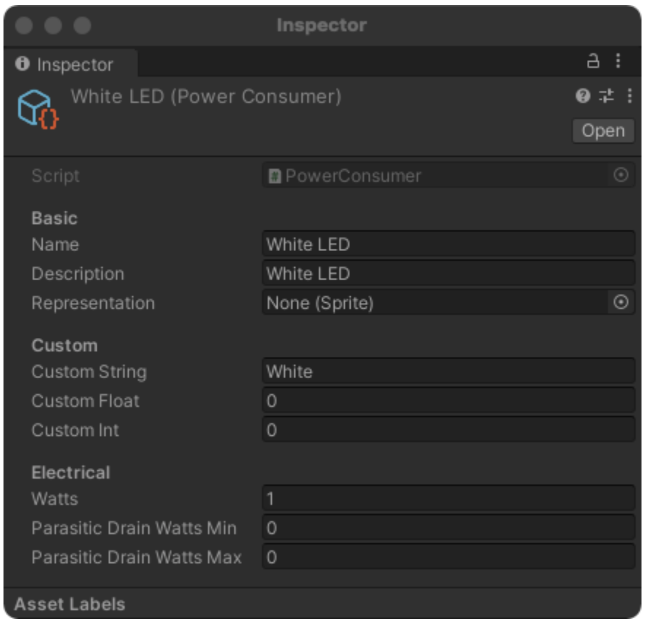
Custom Float: Metadata float field used to represent whatever you want.

Custom Int: Metadata integer field used to represent whatever you want.

Loss: Loss on the Circuit

Amperage Rating: Load capacity of the Circuit

POWER CONSUMER SCRIPTABLE OBJECT



White LED (Power Consumer)

Script	PowerConsumer
Basic	
Name	White LED
Description	White LED
Representation	None (Sprite)
Custom	
Custom String	White
Custom Float	0
Custom Int	0
Electrical	
Watts	1
Parasitic Drain Watts Min	0
Parasitic Drain Watts Max	0

Asset Labels

Name: Meaningful name for the object

Description: Meaningful description for the object

Representation: Sprite image for the object

Custom String: Metadata string field used to represent whatever you want.

Custom Float: Metadata float field used to represent whatever you want.

Custom Int: Metadata integer field used to represent whatever you want.

Watts: How much power this Power Consumer consumes

Parasitic Drain: Random value between Min and Max representing the amount of power consumed even when the Power Consumer is switched off.

Setting Up Your Network

As mentioned earlier, ECC is super flexible and can be configured in multiple different ways depending on your requirements.

You need at least one ElectricalCircuitController component on a GameObject (or in code) somewhere in your Project. From there though it's up to you. That single instance could be used to add multiple Networks and to control everything from a single, central location. Alternatively, you could have multiple instances of ElectricalCircuitController, with each instance having one (or more) Networks.

There are no real rules, and performance wise it won't make any difference (other than theoretical) how you slice and dice your setup.

- A single ElectricalCircuitController with a single Network, single Power Source and single Circuit makes sense for separate self-contained objects like a flashlight or a billboard sign.
- A single ElectronicCircuitController with multiple Networks, single Power Source and single Circuit makes sense for similar but unrelated objects like a player's flashlight, his digital watch, his mobile phone and his portable charger.
- A single ElectronicCircuitController with single or multiple Networks, multiple Power Sources and multiple Circuits makes sense for larger groupings of related objects like a city office block, an apocalypse survivor settlement or a fleet of electric vehicles.

Multiple instances of ElectricalCircuitController could work equally well for each of the above scenarios, the design choice really depends on the objective.

Importantly, the pattern and complexity of the Network design doesn't necessarily equate to the physical size of the object you are representing. Imagine you wanted to control a digital watch at a deeper level than just battery life. It may look something like this:

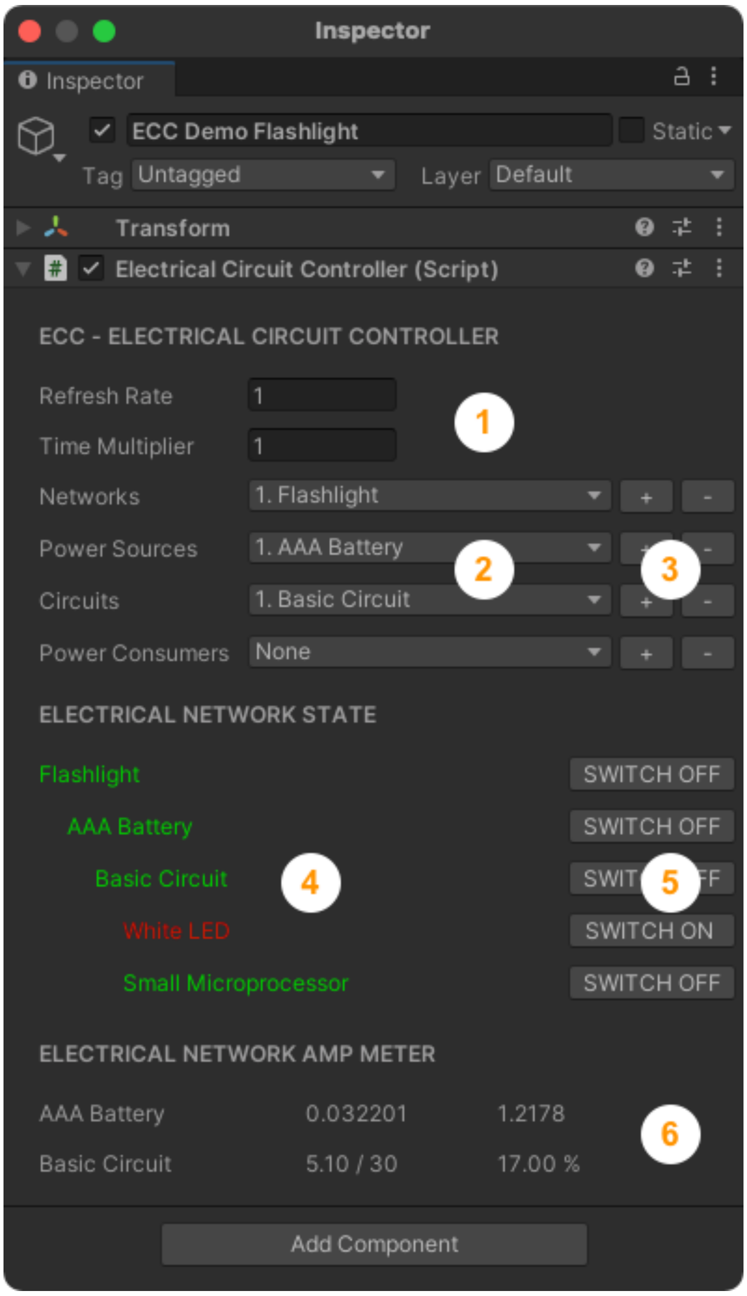
Digital Watch Network

Module Type	**Module Name**	**Purpose**
Network	Digital Watch	The top level container
Power Source	CR2032 Battery	Main power supply for the watch
Circuit	Main Circuit	Controls most of the watch electronics
Power Consumer	LED Screen	Digital display for the watch
Power Consumer	GPS Sensor	Tracks GPS location, might be high power usage
Power Consumer	Health Sensor	Tracks heart rate and step count
Power Source	Solar Panel	Charges the battery
And so on and so on…		

Unity Custom Editor

A fundamental design principle of ECC was to make sure it's dead easy to setup and use. We are not all gun developers, and in fact many of us are more adept at design and graphical creativity than with C# classes, coding and software best practice. The actual code and logic driving ECC is not overly complicated, rather, the way it all plugs together quickly and easily is what makes the asset most valuable.

An enormous step towards this simplicity is the custom editor provided with ECC. As soon as you add the ElectronicCircuitController component to a GameObject in the editor you will see it.



ECC Demo Flashlight

Tag Untagged   Layer Default

Electrical Circuit Controller (Script)

ECC - ELECTRICAL CIRCUIT CONTROLLER

Refresh Rate	1
Time Multiplier	1
Networks	1. Flashlight
Power Sources	1. AAA Battery
Circuits	1. Basic Circuit
Power Consumers	None

ELECTRICAL NETWORK STATE

Flashlight	SWITCH OFF
AAA Battery	SWITCH OFF
Basic Circuit	SWITCH OFF
White LED	SWITCH ON
Small Microprocessor	SWITCH OFF

ELECTRICAL NETWORK AMP METER

AAA Battery	0.032201	1.2178
Basic Circuit	5.10 / 30	17.00 %

1. Time related values controlling how often ECC should refresh itself, and at what timescale it should run at.

2. Module drop-downs to select Networks, Power Sources, Circuits and Power Consumers.

3. Module add/remove buttons to add new, or remove existing, Networks, Power Sources, Circuits and Power Consumers.

4. The status of the entire ECC network, not just the selected Network and not just what is selected in the drop-downs above. Green means operating, red means switched off and/or not operating.

5. Switches to turn ON or OFF each of the Modules separately. A module can be switched ON but still may not be operating because there is no power or the level above it is not switched on. Think of a desk lamp, it may be switched on but if it's not plugged in it won't work.

6. Electrical usage for the entire ECC network, not just the selected Network and not just what is selected in the drop-downs above. For Power Sources it displays the used and remaining Watts. For Circuits it displays Load amount and percentage. If a Circuit is overloaded it will switch itself off.

Properties and Methods

The main (and only) script you need to add to your project is the ElectricalCircuitController script located in the ECC/Scripts folder.

Name	**Description**
ElectricalCircuitController	
RefreshRate	How often should internal calculations be applied 0 = every frame, 0.5 = half second, 1 = second etc
TimeMultiplier	How much faster than “real time” should the controller run. 1 = real time, 5 = 5x faster, 50 = 50 times faster etc
FirstNetwork()	Returns the first NetworkObject
NetworkCount()	Returns the number of Networks
NetworkByName(string name)	Returns the first NetworkObject with the given name
NetworkByIndex(int index)	Returns the NetworkObject at the given index
FirstPowerSource(NetworkObject network)	Returns the first PowerSourceObject for the given network
PowerSourceCount(NetworkObject network)	Returns the number of PowerSources for the given network
PowerSourceByName(NetworkObject network, string name)	Returns the first PowerSourceObject with the given name for the given network
PowerSourceByIndex(NetworkObject network, int index)	Returns the PowerSourceObject at the given index for the given network
FirstCircuit(PowerSourceObject powerSource)	Returns the first CircuitObject for the given powerSource
CircuitCount(PowerSourceObject powerSource)	Returns the number of Circuits for the given powerSource
CircuitByName(PowerSourceObject powerSource, string name)	Returns the first CircuitObject with the given name for the given powerSource
CircuitByIndex(PowerSourceObject powerSource, int index)	Returns the CircuitObject at the given index for the given powerSource
FirstPowerConsumer(CircuitObject circuit)	Returns the first PowerConsumerObject for the given circuit
PowerConsumerCount(CircuitObject circuit)	Returns the number of PowerConsumers for the given circuit
PowerConsumerByName(CircuitObject circuit, string name)	Returns the first PowerConsumerObject with the given name for the given circuit
PowerConsumerByIndex(CircuitObject circuit, int index)	Returns the PowerConsumerObject at the given index for the given circuit
Base Module Properties (all modules inherit these properties)	
string Name	Name of the module

string Description	Description of the module
Sprite Representation	Sprite of the module
string CustomString	General purpose string field
float CustomFloat	General purpose float field
int CustomInt	General purpose int field
NetworkObject	
bool SwitchedOn	Switch off and on
Network	Network Module
PowerSources	List of PowerSourceObjects
Operating()	Returns true if powered and switched on
PowerSourceObject	
bool SwitchedOn	Switch off and on
PowerSource	PowerSource Module
PowerSource.Volts	Volts value
PowerSource.AmpHours	AmpHours value
Circuits	List of CircuitObjects
HasPower()	Returns true if powered
Operating()	Returns true if powered and switched on
AmpHoursUsed()	Returns the number of Amp Hours used
AmpHoursRemaining()	Returns the number of Amp Hours remaining
CircuitObject	
bool SwitchedOn	Switch off and on
Circuit	Circuit Module
Circuit.Loss	Loss value
Circuit.AmperageRating	AmperageRating value
PowerConsumers	List of PowerConsumerObjects
HasPower()	Returns true if powered
Operating()	Returns true if powered and switched on
CapacityWatts()	Returns the circuit capacity in Watts
LoadWatts()	Returns the circuit load in Watts
PowerConsumerObject	
bool SwitchedOn	Switch off and on

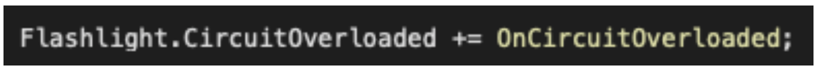
PowerConsumer	PowerConsumer Module
PowerConsumer.Watts	Number of Watts consumed value
PowerConsumer.ParasiticDrainMIn	Minimum parasitic drain value
PowerConsumer.ParasiticDrainMax	Maximum parasitic drain value
HasPower()	Returns true if powered
Operating()	Returns true if powered and switched on
ParasiticDrain()	Returns the ParasiticDrain value

Events

1. CircuitOverloaded

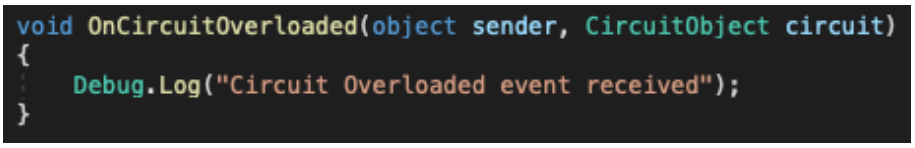
Triggers when a circuit is turned off (tripped) due to capacity overload

Example how to subscribe



```
Flashlight.CircuitOverloaded += OnCircuitOverloaded;
```

Example how to use



```
void OnCircuitOverloaded(object sender, CircuitObject circuit)
{
    Debug.Log("Circuit Overloaded event received");
}
```

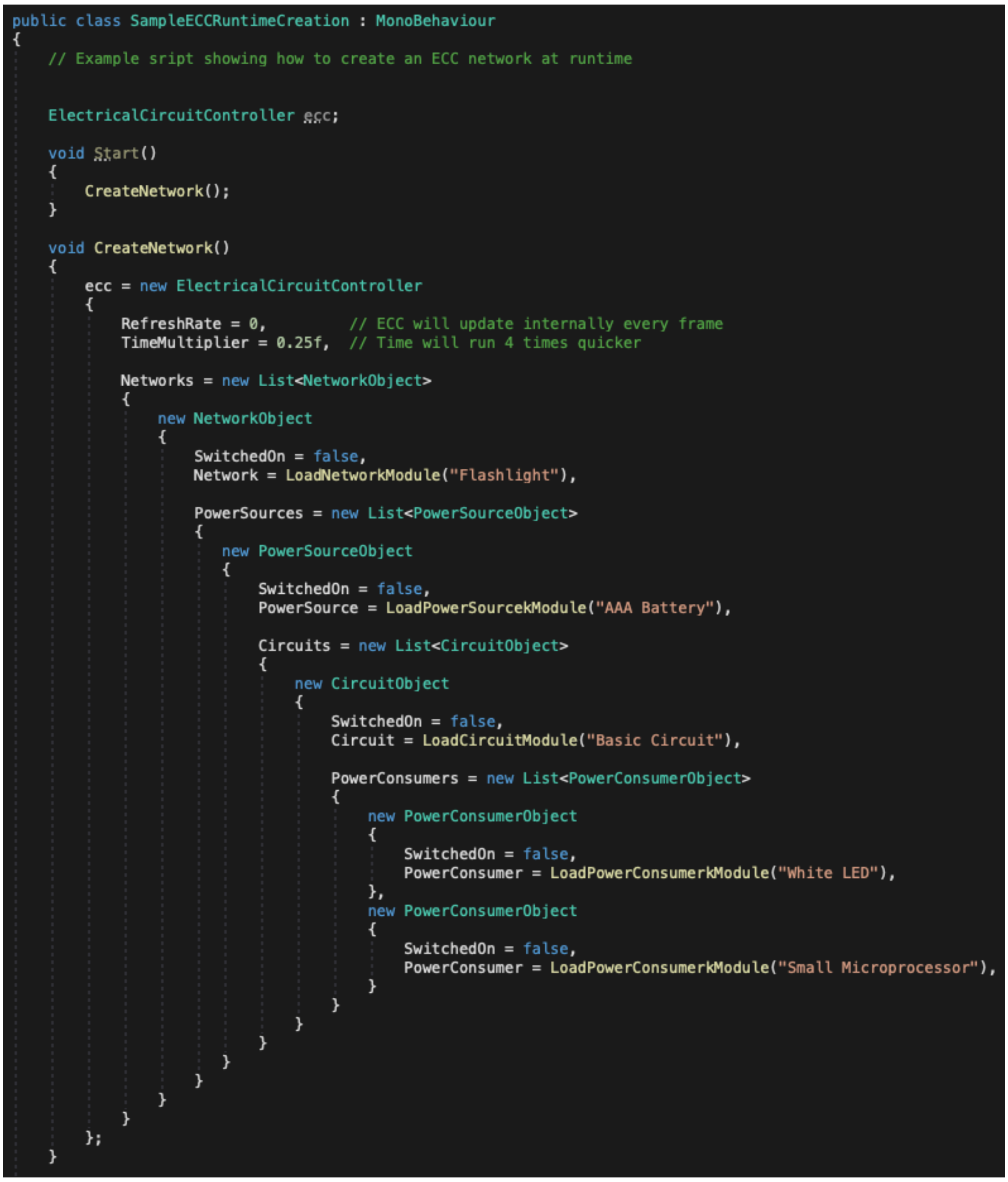
Runtime Creation

Everything that can be done in the custom editor at design time can be equally performed via code at runtime. Although it is arguably more complicated to do so, if you need to you can. Remember too if might be easier to create some ECC Prefabs and just instantiate them at runtime rather than rolling up your sleeves and digging into native C#.

The only real challenge with scripting a fully runtime enabled solution is the process of finding and loading the correct Scriptable Objects. That is beyond the scope of this reference guide but I'm comfortable in assuming that if you're going down the C# route you will be able to come up with a way to identify and load all the goodies you need.

One last time, it is strongly recommended to use the custom editor or prefabs wherever possible, the C# scripting solution should only be when completely necessary for some reason.

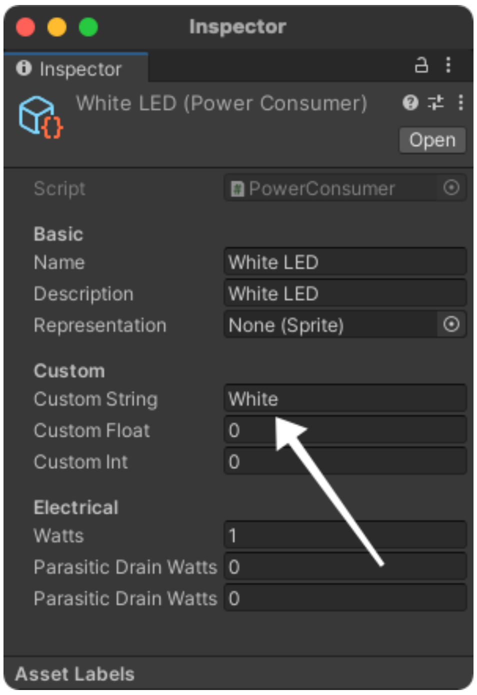
Having said all that, here is an extract of a simple example to create the Flashlight network mentioned earlier. This script is also available in the ECC/Demo/Scripts folder called SampleECCRuntimeCreation.cs. This script doesn't actually do anything and it doesn't actually load the Scriptable Objects, that's left as an exercise for the reader :)



```
public class SampleECCRuntimeCreation : MonoBehaviour
{
    // Example sript showing how to create an ECC network at runtime

    ElectricalCircuitController ecc;

    void Start()
    {
        CreateNetwork();
    }

    void CreateNetwork()
    {
        ecc = new ElectricalCircuitController
        {
            RefreshRate = 0,          // ECC will update internally every frame
            TimeMultiplier = 0.25f,   // Time will run 4 times quicker

            Networks = new List<NetworkObject>
            {
                new NetworkObject
                {
                    SwitchedOn = false,
                    Network = LoadNetworkModule("Flashlight"),

                    PowerSources = new List<PowerSourceObject>
                    {
                        new PowerSourceObject
                        {
                            SwitchedOn = false,
                            PowerSource = LoadPowerSourcekModule("AAA Battery"),

                            Circuits = new List<CircuitObject>
                            {
                                new CircuitObject
                                {
                                    SwitchedOn = false,
                                    Circuit = LoadCircuitModule("Basic Circuit"),

                                    PowerConsumers = new List<PowerConsumerObject>
                                    {
                                        new PowerConsumerObject
                                        {
                                            SwitchedOn = false,
                                            PowerConsumer = LoadPowerConsumerkModule("White LED"),
                                        },
                                        new PowerConsumerObject
                                        {
                                            SwitchedOn = false,
                                            PowerConsumer = LoadPowerConsumerkModule("Small Microprocessor"),
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        };
    }
```

Custom Metadata Fields

Back in the Scriptable Objects section of this reference guide there was mention of the custom metadata fields CustomString, CustomFloat and CustomInt. So what exactly are these for? They are general purpose fields to be used by you (or your developers) to add meaningful attributes to the Module.



White LED (Power Consumer)

Script	PowerConsumer
Basic	
Name	White LED
Description	White LED
Representation	None (Sprite)
Custom	
Custom String	White
Custom Float	0
Custom Int	0
Electrical	
Watts	1
Parasitic Drain Watts	0
Parasitic Drain Watts	0

For example, one of the Power Consumer Modules included with ECC is a White LED. What actually makes it white though, it's not really an LED and it doesn't really have a color. Looking at the Scriptable Object you will see the string "White" in the CustomString field. That alone doesn't do anything, but, in your game code and logic it is a field that can be interpreted and used.

Your game may have white, red, green and blue LED's and in ECC they will identical except for the CustomString values (or maybe a 0, 1, 2, 3 in the CustomInt field instead). The colour of the real LED object in your game could be driven by the value in one of the custom fields.

Of course these fields aren't just used for color, anything extra that needs to be referenced can be setup in the custom field. For example you could store the brand model "Luxeon DS25 LXHL-BW" in the CustomString field and some unique ID in the CustomInt field.

Creating New Modules

We've been saying how great it is that you can just create your own new modules to use with ECC, and it is, but how exactly do you do that?

An ECC module (Network, Power Source, Circuit, Power Consumer) is really just a standard Unity Scriptable Object. The easiest way to create a new module is to duplicate one of the existing ones and then change it.

For example, let's say you want to create a new power source, a D cell battery. Navigate to the **Power Sources** folder in your project's **ECC/Modules** folder. Click on say the AAA Battery and duplicate it (Ctrl+D on Windows, Cmd+D on Mac). Rename the newly created copy to "D Cell Battery", then in the Unity Inspector window change the Name, Description, Volts, Amp Hours etc to whatever you need. That's it, you now have a new D Cell Power Source.

Another way to create a new Module is to right click inside a folder in your Project Window, Select **Create**, then select **ECC**, then select the module type you need (Network, Power Source, Circuit or Power Consumer).

Finally, you could also click on the Unity **Assets** menu, then select **Create,** then select **ECC** and choose Network, Power Source, Circuit or Power Consumer from the menu.