IBM Blockchain

IBM Blockchain Hands-On Composer Development

Lab Three

Contents

Overview

The aim of this lab is to get you familiar with developing Hyperledger Composer business networks. We will do this by exploring the Composer modelling language, how to write transaction processor functions in JavaScript and lastly examine how Access Control is managed in Composer.

The lab will also familiarise you with the Composer Playground, a web based tool that allows for rapid development and testing of Composer business networks.

It should be noted that while the contents of this lab will predominantly occur within Composer Playground (for the sake of accelerating the learning and development process), the Lab can easily be completed offline and using a text editor such as Visual Studio Code or Atom. In this case please refer to the next Lab for instructions on how to use the command line tools available in Composer.

Introduction

Pre-requisites:

- 4 cores
- 6GB RAM
- VMWare V12+
- The lab virtual machine

The virtual machine is based on Linux Ubuntu 16.04 and contains Hyperledger Fabric V1.0.4, Golang, Git, Visual Studio Code and Firefox.

A network needs to be visible to the virtual machine (even if the network is just to the host environment). If you do not see the up/down arrows in the status bar at the top of the screen, or if you receive errors about no network being available, please tell the lab leader. The virtual machine might need to be reconfigured in NAT mode.

There are no additional files or software that is proprietary to the lab in the virtual machine. This means that the lab may be run on a machine without the without a lab virtual machine if Hyperledger Fabric and the other pre-requisites have been installed.

Section 1. Standing up the network

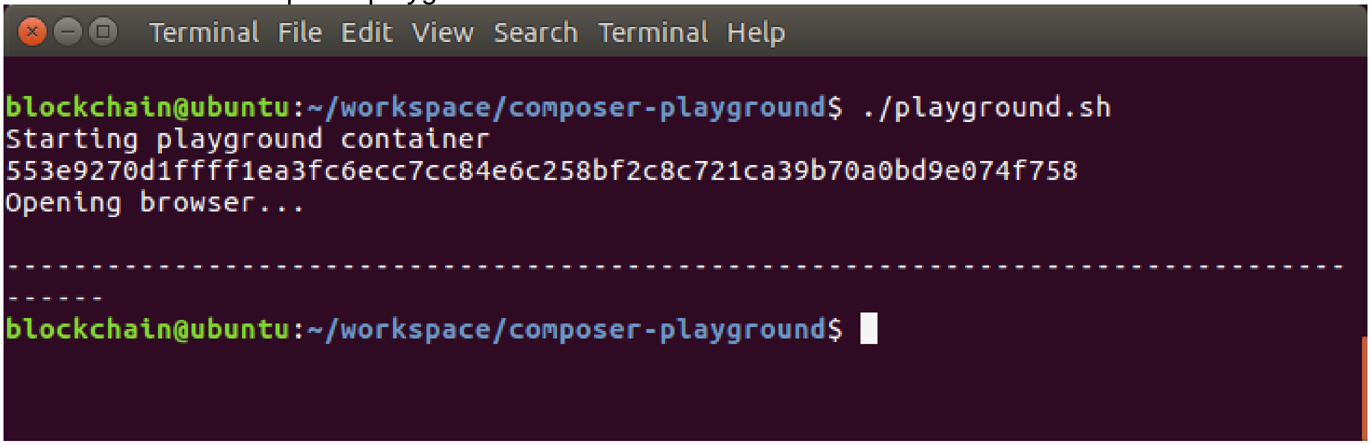
This section will cover standing up the network on the VM provided. As the focus of the Lab is on learning Composer development, scripts have been written to stand up the network for you. Please feel free to inspect these scripts to understand how they work.

1.1. Running the script

Navigate to `~/workspace/composer-playground` and issue the following command:

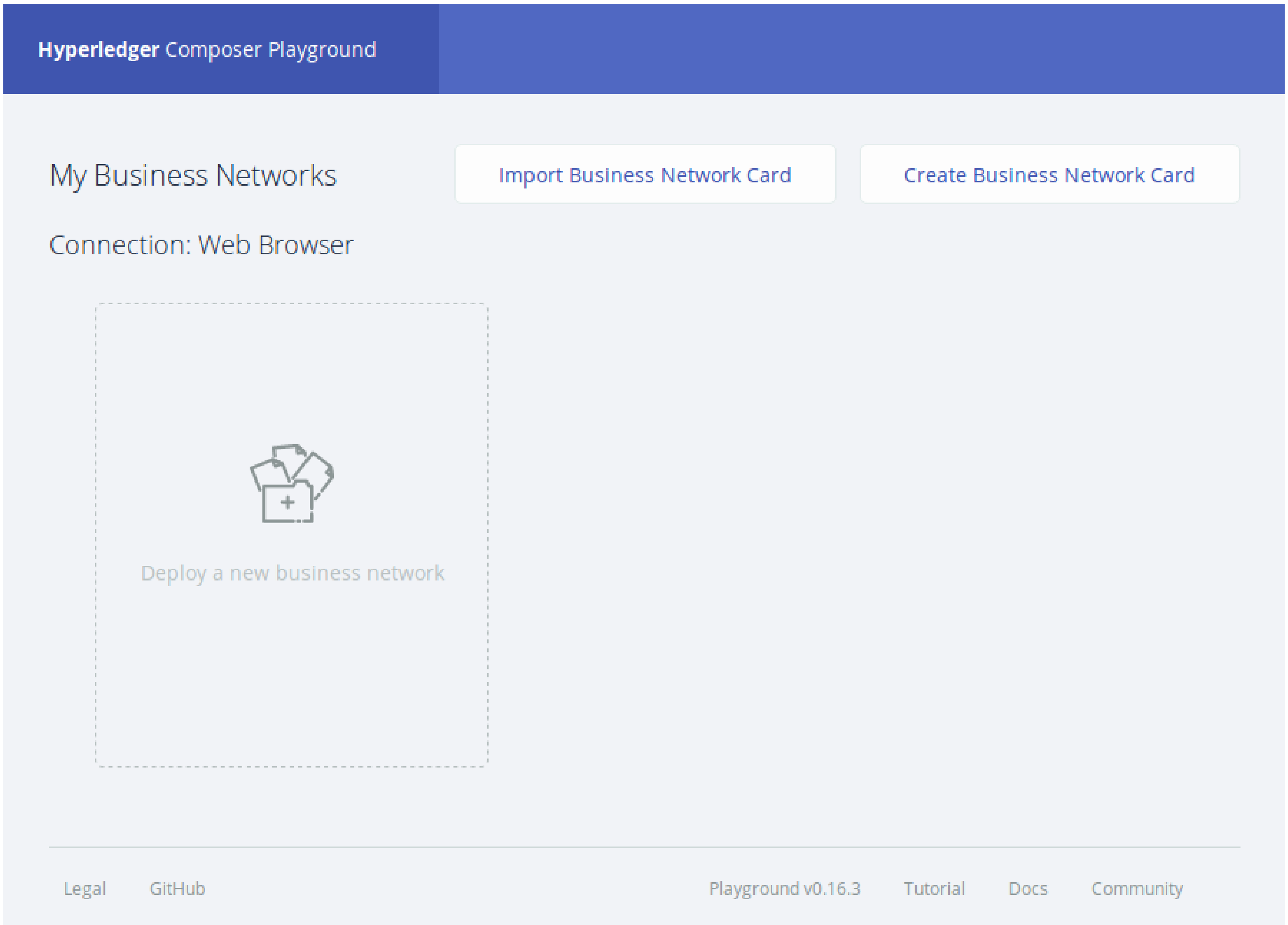
```
./playground.sh
```

This will start the composer playground container:



```
blockchain@ubuntu:~/workspace/composer-playground$ ./playground.sh
Starting playground container
553e9270d1ffff1ea3fc6ecc7cc84e6c258bf2c8c721ca39b70a0bd9e074f758
Opening browser...

------------------------------------------------------------------------------
------
blockchain@ubuntu:~/workspace/composer-playground$ 
```

It should also launch the default browser and open the playground web UI. If it doesn't open a browser and go to http://localhost:8080:



Hyperledger Composer Playground

My Business Networks

Connection: Web Browser

Import Business Network Card
Create Business Network Card
Deploy a new business network
Legal
GitHub
Playground v0.16.3
Tutorial
Docs
Community

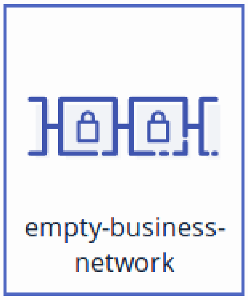
Section 2. Composer's Modelling Language

In this section you will learn about the modelling language Hyperledger Composer uses to define resources in it's business networks. You will define a basic business network that allows participants to exchange marbles with each other, using this as a base from which to explore the language's features.

2.1. Syntax

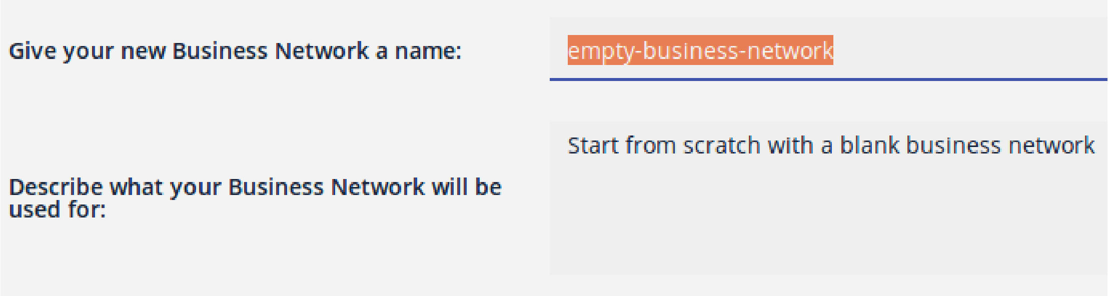
a. Reset the playground

Click the "Deploy a new business network" box.
Then scroll down and select '**Empty Business Network**':



empty-business-network

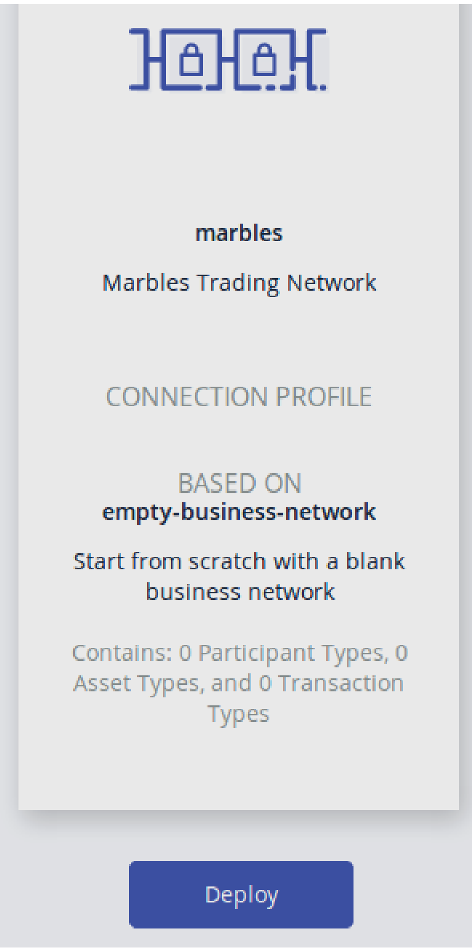
b. Give the network the name "marbles" and a description of "marbles trading network"



Give your new Business Network a name: empty-business-network

Describe what your Business Network will be used for: Start from scratch with a blank business network

c. Click the "Deploy" button to deploy the new marbles network



marbles

Marbles Trading Network

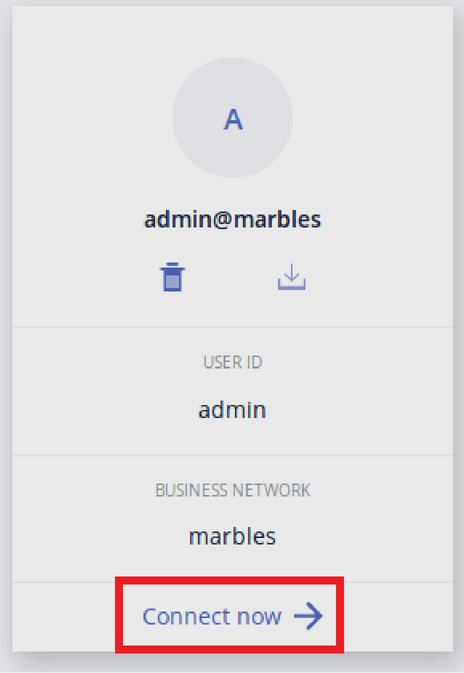
CONNECTION PROFILE

BASED ON
empty-business-network

Start from scratch with a blank business network

Contains: 0 Participant Types, 0 Asset Types, and 0 Transaction Types

d. **Click “Connect Now” in the new card for the marbles network.**



admin@marbles

USER ID

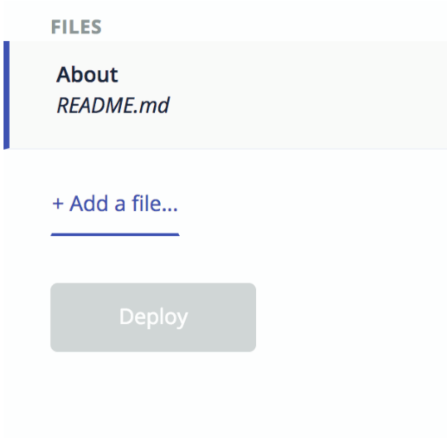
admin

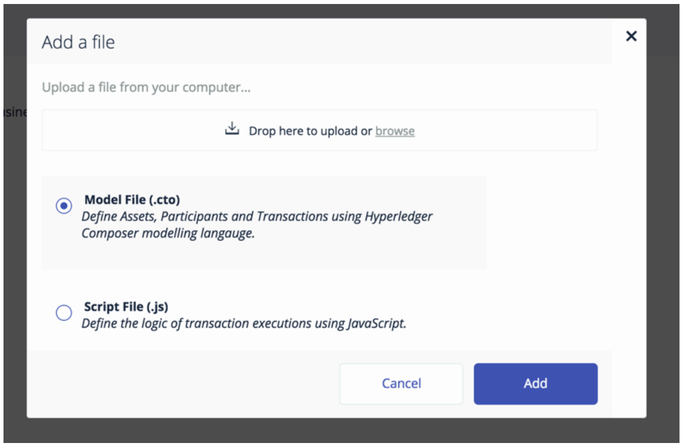
BUSINESS NETWORK

marbles

e. Add a model file

Select the **+Add a file…** button on the top left and from the list select





Upload a file from your computer...

● **Model File (.cto)**
Define Assets, Participants and Transactions using Hyperledger Composer modelling langauge.

○ **Script File (.js)**
Define the logic of transaction executions using JavaScript.

f. Resources

Composer's modelling language is first and foremost oriented around high level business concepts. As such, the **three top-level resources** that can be defined are as follows:

Assets	Participants	Transactions
Digital representations of assets that are recorded on the ledger.	*Individuals and Organisations that will contribute to and make use of the ledger.*	*Business logic governing the manipulation of assets.*

Additionally, each resource belongs to a namespace (a default namespace is at the top of the newly created file) which acts in a similar manner to how namspaces and packages work in other languages. In much the same way, resources can be imported from other namespaces. Namspace names can be any combination of letters and periods.

The modelling language describes these resources in a similar manner that you would describe a class in another language, this being an entity with attributes.

g. Writing an asset

Underneath namespace `org.acme.model` write the following:

```
asset Marble identified by Id {
      o String Id
}
```

This defines a Marble asset and gives it an identifier to be referred to by (similar to the keys used in Fabric). Let's flesh this out to include some additional attributes:

```
asset Marble identified by Id {
      o String Id
      --> Collector owner
      o Integer diameter
      o String colour
}
```

You'll have noticed that the attributes in this do not all have the same prefix. The `owner` attribute is preceded by a `-->` instead of a `o`.

The o attributes are 'named fields' - they belong to the resource, for example the Marble will have a size and colour property.

The `-->` attributes are 'relationships' - while they make up the information that describes the resource they are not part of it, for example a Marble will have an owner but the owner is not part of the Marble.

The currently supported attribute types are as follows:

```
String, Boolean, DateTime, Integer, Double, Long
```

h. Writing a participant

With this in mind, we shall define an owner. Add the following below the `Marble` definition:

```
participant Collector identified by email {
  o String email
}
```

Attributes for participants work in an identical manner to those of Marbles. As such, expand the `Collector` class:

```
participant Collector identified by email {
  o String email
  o String firstname
  o String surname
  o Address address
  o Sex sex
}
```

You will notice that there are two fields that are not included in the base types listed in **d**. `Address` and `Sex` are, respectively, a **Concept** and an **Enumeration**.

i. Concepts

When modelling data, the base types will only go so far. Some pieces of data require a more complex definition yet cannot be considered resources in their own right.

An address is a perfect example of this, it requires a more complex structure than a single field yet it is not a resource in it's own right - it is simply part of one. Add the following to your model file:

```
concept Address {
  o String house
  o String street
  o String county
  o String postcode
  o String country
}
```

j. Enumerations

Enumerations in Composer function identically to that in other languages, add the following to your model file:

```
enum Sex {
  o MALE
  o FEMALE
}
```

k. Abstract Resources and Inheritance

NOTE: This section is for reference only; do not actually modify your code. Start editing again at section 2.2

Like other OOP languages, Composer supports abstract types and inheritance. Say we wanted to add a new shape of Marble - a PolyhedronMarble:

```
asset PolyhedronMarble identified by Id {
  o String Id
  --> Collector owner
  o String colour
  o Integer faces
}
```

This new type of Marble shares many characteristics with Marble, as such it would make sense to not repeat these common attributes (particularly if we add more types). As such we can define an **Abstract** Marble class and have our two types of Marble extend it:

```
abstract asset Marble identified by Id {
  o String Id
  --> Collector owner
  o String colour
}

asset SphericalMarble extends Marble {
  o Integer diameter
}

asset PolyherdalMarble extends Marble {
  o Integer faces
  o Integer arcLength
}
```

Now code is not pointlessly repeated. You will also notice that because the identifying attribute is defined in the abstract class, it is not necessary to do so in the child classes (although doing so would simply overwrite it). Identifiers are optional in abstract classes.

Validators

Some attributes will come in a specific format and indeed it is often the case that code will be written expecting them in this format for example emails are always of the form `<name>@<domain>.<suffix>`. Likewise limits may be placed on data or default values may be enforced. The following validators are supported (add them to your resources):

Regex

```
participant Collector identified by id {
  o String email regex=/[a-z0-9.-]+@[a-z0-9.]+/
  ...
}
```

Regex or Regular Expressions are a method of specifying a query for searching strings. In this situation the regex pattern can be used to enforce rules about string form, rejecting anything that does not match the pattern.

Optional

```
concept Address {
..
  o String county optional
..
}
```

As the name suggests, this validator denotes that the field is not required by the model.

Along with default, this is a way of extending models after initial deployment. If new fields are added to a resource, errors that would otherwise be thrown from existing resources becoming non-compliant with the spec can be resolved by making the new fields optional so that the existing resources can be updated as needed.

Default

```
asset Marble identified by Id {
..
  o String colour default="red"
}
```

Default sets an initial value for an attribute that can be overridden later.

Along with optional, this is a way of extending models after initial deployment. If new fields are added to a resource, default values can be set so that they remain compliant with their required fields. This can also be used to set placeholder values that mark fields out for needing to be filled in with real data

Range

```
asset Marble {
...
  o Integer diameter range=[1,20]
...
}
```

Range enforces a inclusive limit on numerical values, in the above this requires he diameter to be between 1 and 20.

2.2. Writing a transaction

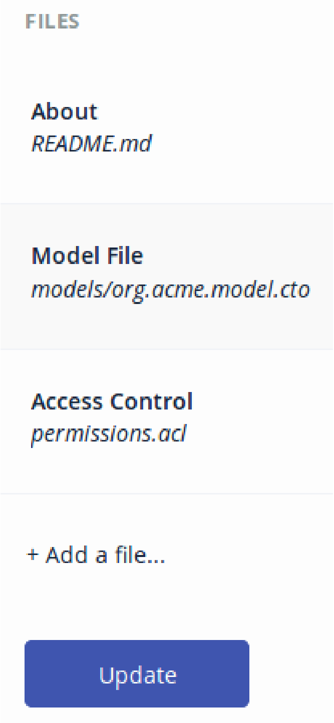
Transactions are also declared in the modelling file using the same syntax as with Assets and Participants, add the following to your modelling file:

```
transaction ChangeOwner {
  --> Marble marble
  --> Collector newOwner
}
```

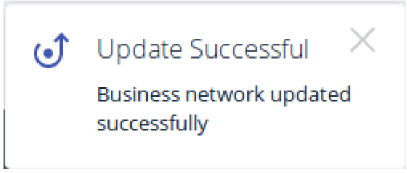
Instead of denoting attributes, the variables within the body of a transaction denote the arguments that the transaction logic function will take (this will be covered in more detail in the next section).

2.3. Update and create some test assets

a. Update the network



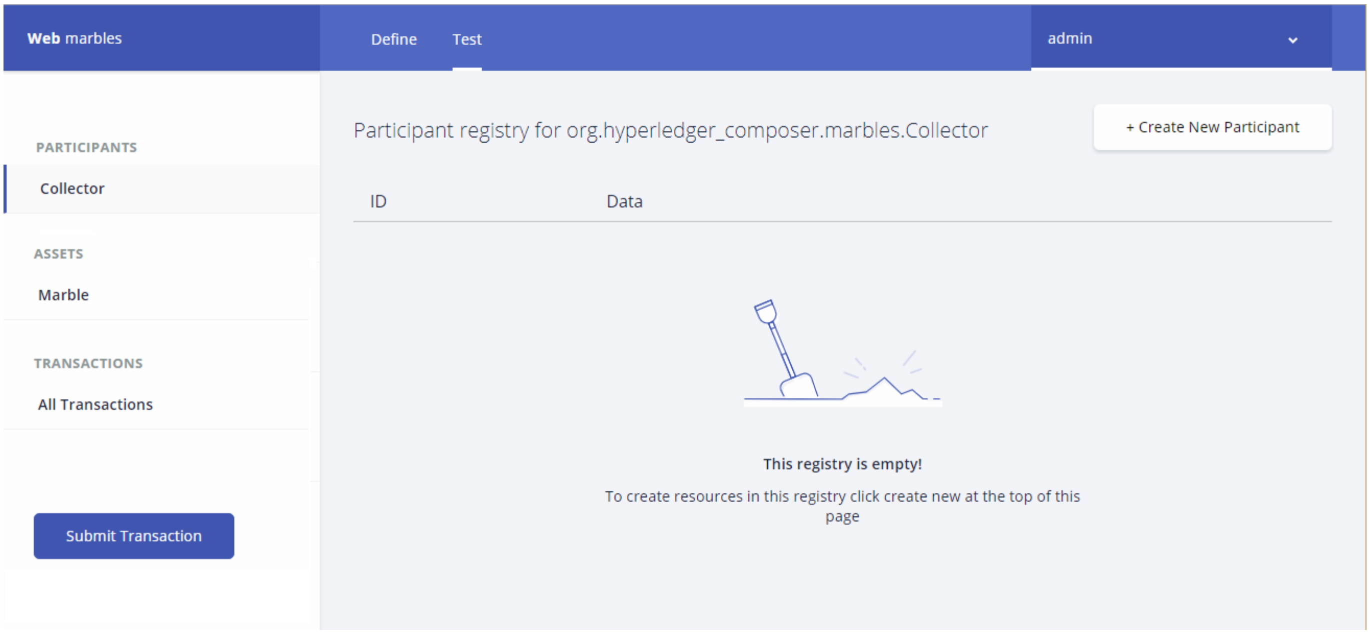
Now we have some asset definitions, hit the “Update” button on the left side of the screen. On success a small pop-up should appear in the top right and the “Update” button will grey out.



Update Successful

Business network updated successfully

Go to the **Test** tab at the top:



Participant registry for org.hyperledger_composer.marbles.Collector

ID	Data

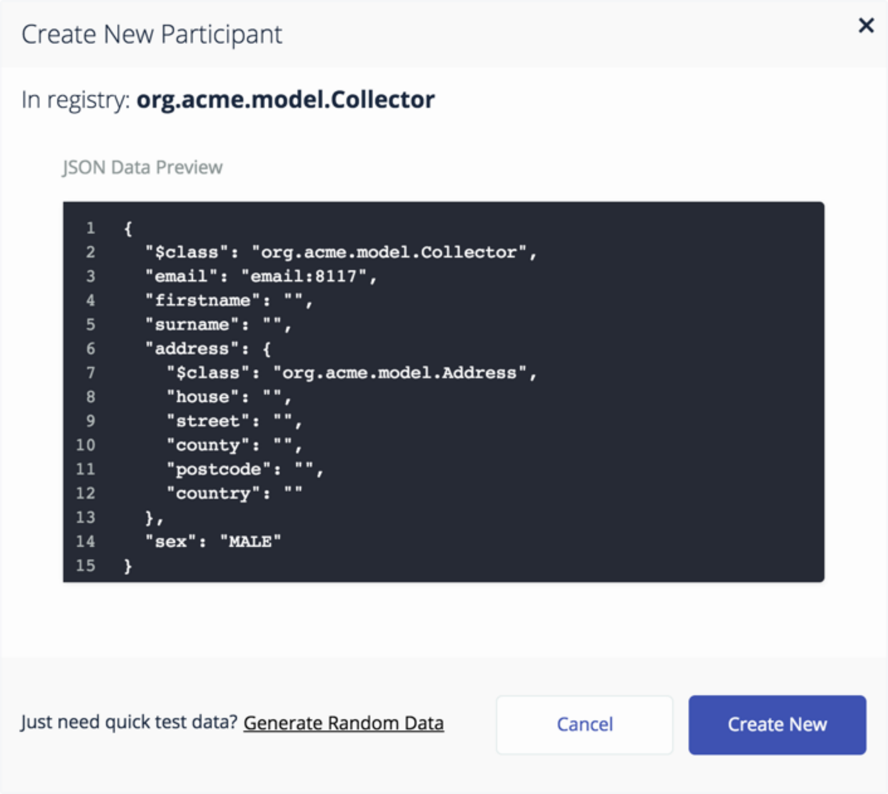
This registry is empty!

To create resources in this registry click create new at the top of this page

Here you can see the assets and participants we’ve made.

b. Create the assets and participants

Hit **+ Create New Participant** on the top right. A dialogue box will appear prompting you to enter details of the new participant:


Create New Participant
×
In registry: **org.acme.model.Collector**
JSON Data Preview

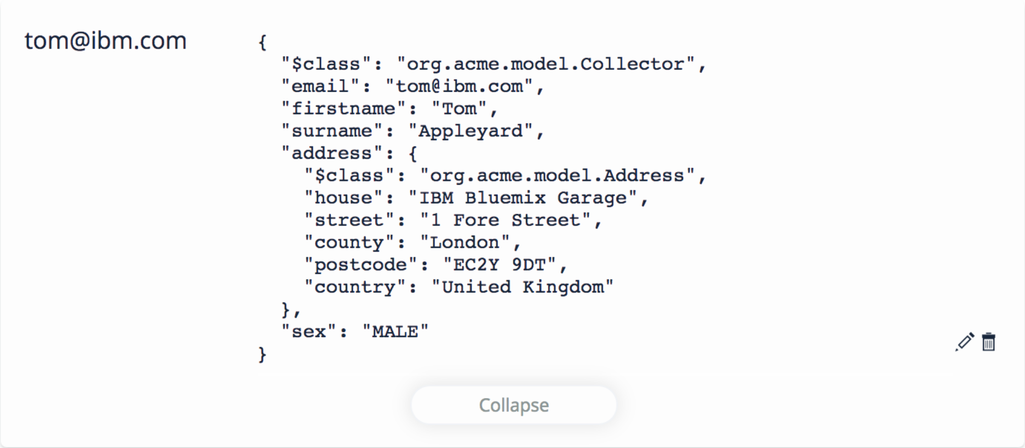
```
{
  "$class": "org.acme.model.Collector",
  "email": "email:8117",
  "firstname": "",
  "surname": "",
  "address": {
    "$class": "org.acme.model.Address",
    "house": "",
    "street": "",
    "county": "",
    "postcode": "",
    "country": ""
  },
  "sex": "MALE"
}
```

Just need quick test data? Generate Random Data
Cancel
Create New

Enter the following:

```
{
  "$class": "org.acme.model.Collector",
  "email": "tom@ibm.com",
  "firstname": "Tom",
  "surname": "Appleyard",
  "address": {
    "$class": "org.acme.model.Address",
    "house": "IBM Bluemix Garage",
    "street": "1 Fore Street",
    "county": "London",
    "postcode": "EC2Y 9DT",
    "country": "United Kingdom"
  },
  "sex": "MALE"
}
```

Fill this in and select **Create New**, you will see the new participant appear:



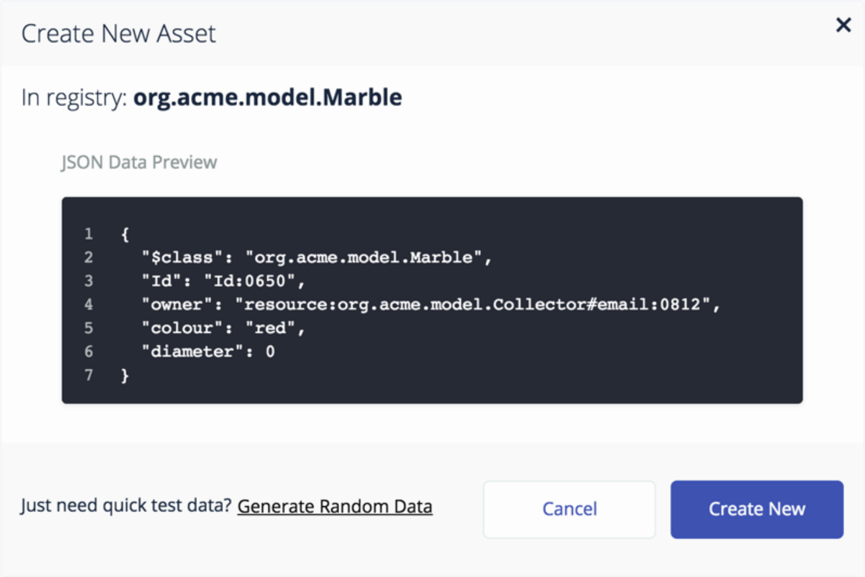
tom@ibm.com

```
{
  "$class": "org.acme.model.Collector",
  "email": "tom@ibm.com",
  "firstname": "Tom",
  "surname": "Appleyard",
  "address": {
    "$class": "org.acme.model.Address",
    "house": "IBM Bluemix Garage",
    "street": "1 Fore Street",
    "county": "London",
    "postcode": "EC2Y 9DT",
    "country": "United Kingdom"
  },
  "sex": "MALE"
}
```

Create a second Collector with the following:

```
{
  "$class": "org.acme.model.Collector",
  "email": "mgk@ibm.com",
  "firstname": "Matthew",
  "surname": "Golby-Kirk",
  "address": {
    "$class": "org.acme.model.Address",
    "house": "IBM Hursley",
    "street": "Hursley Park",
    "county": "Hampshire",
    "postcode": "SO21 2JN",
    "country": "United Kingdom"
  },
  "sex": "MALE"
}
```

If you swap to the Marble asset and select **+ Create New Asset** you will see a similar dialogue box:



Create New Asset

In registry: **org.acme.model.Marble**

JSON Data Preview

```
{
  "$class": "org.acme.model.Marble",
  "Id": "Id:0650",
  "owner": "resource:org.acme.model.Collector#email:0812",
  "colour": "red",
  "diameter": 0
}
```

Just need quick test data? Generate Random Data

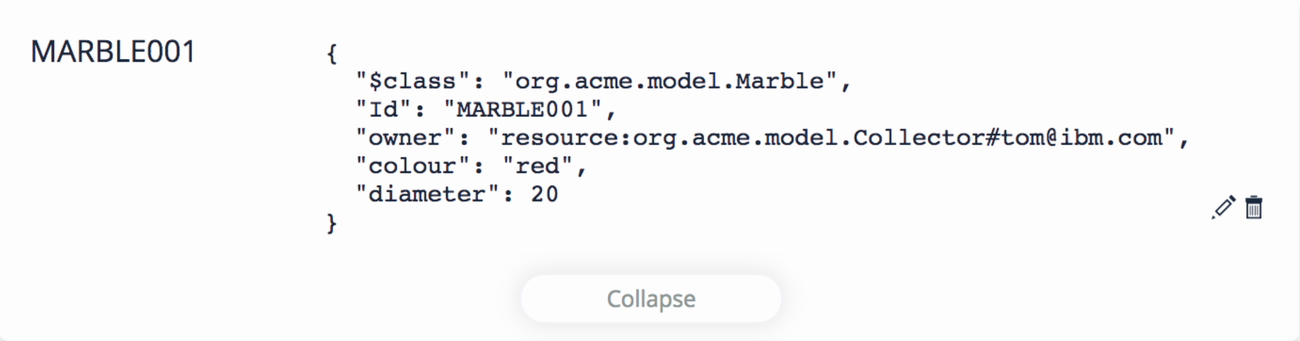
Enter the following. Note that when **filling out relationships**, **you can either supply a fully qualified identifier** - this being as follows:

```
 resource:<namepace>.<resource name>#identifier
"resource:org.acme.model.Collector#tom@ibm.com",
```

or supply just the identifier part and let it be qualified automatically:

```
{
  "$class": "org.acme.model.Marble",
  "Id": "MARBLE001",
  "owner": "tom@ibm.com",
  "colour": "red",
  "diameter": 20
}
```

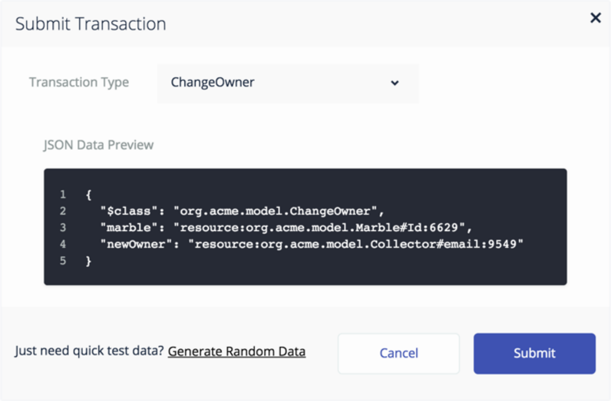
You will see the new asset appear.



MARBLE001

```
{
  "$class": "org.acme.model.Marble",
  "Id": "MARBLE001",
  "owner": "resource:org.acme.model.Collector#tom@ibm.com",
  "colour": "red",
  "diameter": 20
}
```

If you select **Submit Transaction** at the bottom left you will see a similar dialogue box:



Transaction Type: ChangeOwner

JSON Data Preview

```
{
  "$class": "org.acme.model.ChangeOwner",
  "marble": "resource:org.acme.model.Marble#Id:6629",
  "newOwner": "resource:org.acme.model.Collector#email:9549"
}
```

Just need quick test data? Generate Random Data

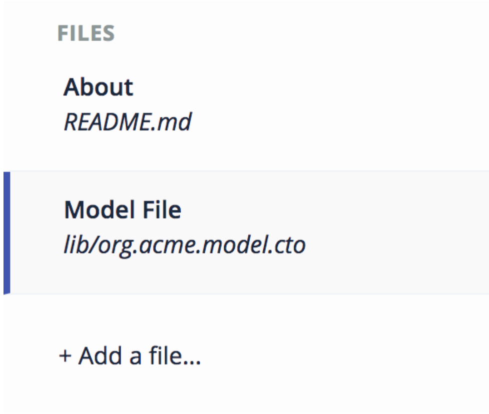
However if you submit this transaction, nothing will happen. We need to define some logic to associate with it. Before moving on to the next section, try creating another participant and an asset.

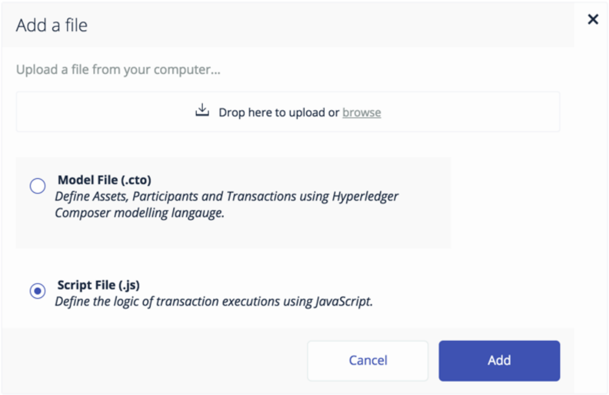
Section 3. Transaction Logic

In this section we will explore how to write transaction processor functions, these being the business logic that is executed when a transaction is invoked in Hyperledger Composer. Please note, while transaction processor functions are analogous to chaincode in their purpose we are not writing chaincode. Composer transaction logic, while achieving the same results, is not handled in the same way as chaincode is.

3.1. Create the logic file

Go back to the “Define” tab and select **+ Add a file…** from the left hand side and select **Script File (.js)** from the dialogue:




Add a file

Upload a file from your computer...

Drop here to upload or browse

○ **Model File (.cto)**
Define Assets, Participants and Transactions using Hyperledger Composer modelling langauge.

● **Script File (.js)**
Define the logic of transaction executions using JavaScript.

3.2. Add the changeOwner function

Within the new file add the following:

```
/**
 * @param {org.acme.model.ChangeOwner} args - the changeOwner transaction arguments
 * @transaction
 */
function changeOwner(args) {

}
```

Transaction processor functions are defined by writing a function with a JS Doc decorator that maps the first argument to the transaction’s model definition. The `args` argument represents the incoming transaction, in particular the data packaged in it.

Recall the transaction’s definition:

```
transaction ChangeOwner {
  --> Marble marble
  --> Collector newOwner
}
```

Args is an object where the keys are each of the attributes and the values are what has been attached to them during the transaction invocation.
`args` will have a `marble` and a `newOwner` attribute that are accessed the same way attributes are accessed in JS objects: `args.marble` or `args.newOwner`.

Transaction processor functions do not return anything, much like `Invoke` functions in Fabric, they simply execute and finish.

3.3. Add changeOwner's body

To change the owner of a marble, simply assign the `newOwner` argument that was passed in to the `marble` argument's `owner` attribute. Add the following to the body of `changeOwner`:

```
args.marble.owner = args.newOwner;
```

Now the update has been made to the asset, we need to update the assets record in the world state. Add the following:

```
return getAssetRegistry('org.acme.model.Marble').then(function(marbleRegistry) {
       return marbleRegistry.update(args.marble);
});
```

Registries are indexes used by composer to store resources, they store a reference to every instance of that particular resources. To update an resources we get the registry for it's type and call the update function with the new version of the resources we want to update (composer will find it within the registry and update it for us).

Participants also have registries and are updated in the same way (although with `getParticipantRegistry`).

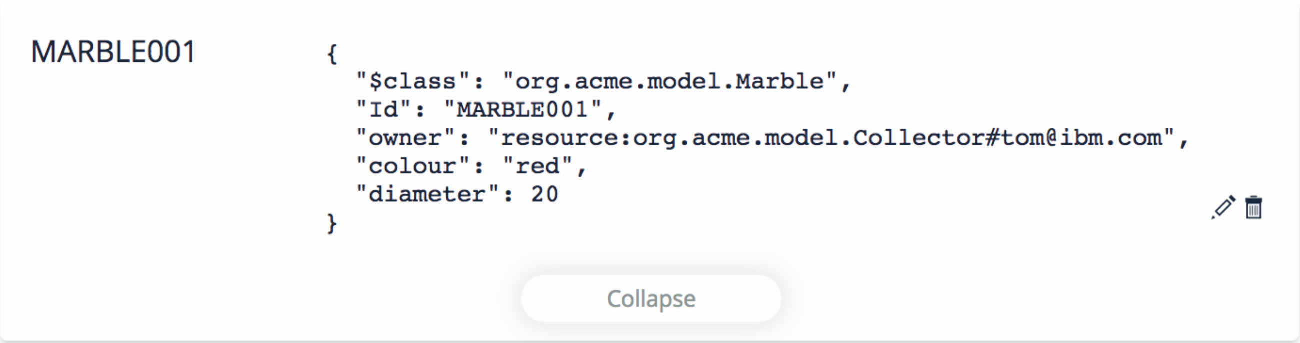
It should be noted that while the language composer uses for it's transaction process functions is JavaScript it currently only supports up to ES5, as such features like `() => {}` functions are not permitted. This is due to the 'duktape' JavaScript engine that is currently used by composer. Duktape is scheduled to be replaced by Node.js in a future release.

3.4. Test changeOwner

a. Create the assets and participants

Update the code and go to the Test tab. We are going to transfer a Marble between two Collectors. If you don't have 2 `Collectors` or a `Marble` follow the steps in 2.3 to create them:

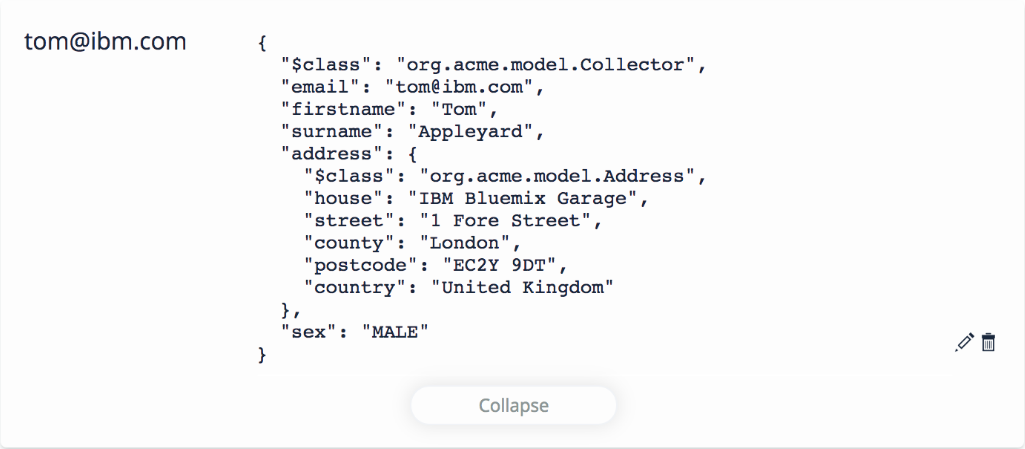
Asset



MARBLE001

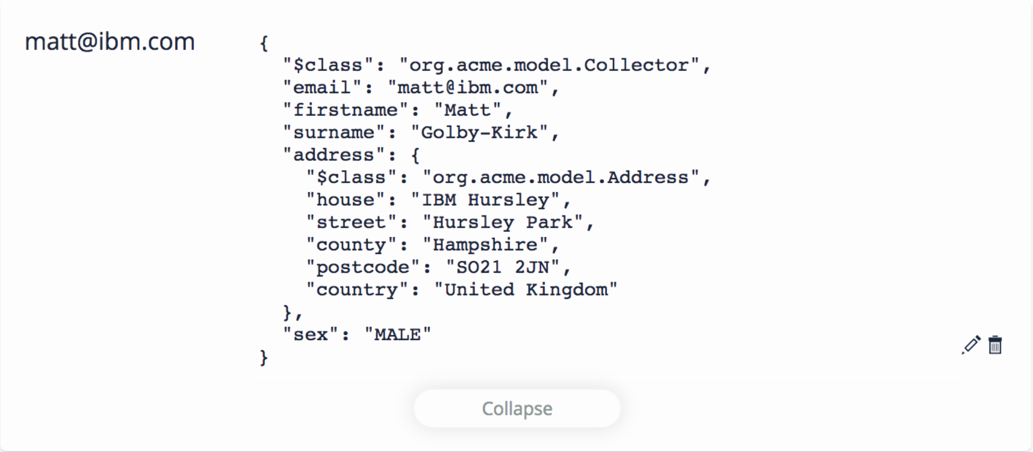
```
{
  "$class": "org.acme.model.Marble",
  "Id": "MARBLE001",
  "owner": "resource:org.acme.model.Collector#tom@ibm.com",
  "colour": "red",
  "diameter": 20
}
```

Participants



tom@ibm.com

```
{
  "$class": "org.acme.model.Collector",
  "email": "tom@ibm.com",
  "firstname": "Tom",
  "surname": "Appleyard",
  "address": {
    "$class": "org.acme.model.Address",
    "house": "IBM Bluemix Garage",
    "street": "1 Fore Street",
    "county": "London",
    "postcode": "EC2Y 9DT",
    "country": "United Kingdom"
  },
  "sex": "MALE"
}
```



matt@ibm.com

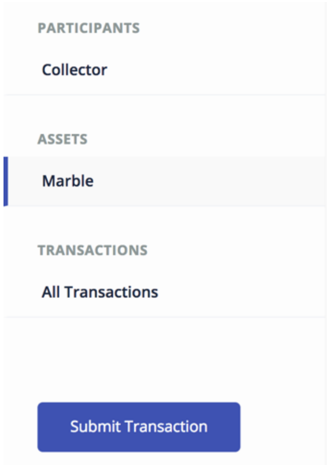
```
{
  "$class": "org.acme.model.Collector",
  "email": "matt@ibm.com",
  "firstname": "Matt",
  "surname": "Golby-Kirk",
  "address": {
    "$class": "org.acme.model.Address",
    "house": "IBM Hursley",
    "street": "Hursley Park",
    "county": "Hampshire",
    "postcode": "SO21 2JN",
    "country": "United Kingdom"
  },
  "sex": "MALE"
}
```

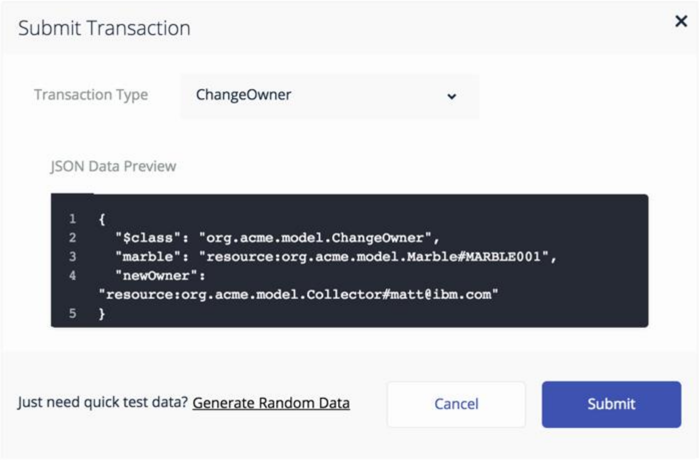
b. Submit the transaction

Select **Submit Transaction** from the sidebar and fill in the fields accordingly to select your marble and the Collector who is not the owner:

```
{
  "$class": "org.acme.model.ChangeOwner",
  "marble": "resource:org.acme.model.Marble#MARBLE001",
  "newOwner": "resource:org.acme.model.Collector#mgk@ibm.com"
}
```

Select **Submit** to issue the transaction.





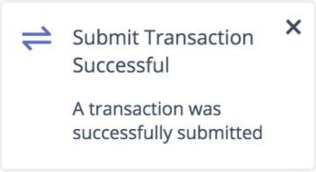
Transaction Type: ChangeOwner

JSON Data Preview

```
{
  "$class": "org.acme.model.ChangeOwner",
  "marble": "resource:org.acme.model.Marble#MARBLE001",
  "newOwner":
"resource:org.acme.model.Collector#matt@ibm.com"
}
```

Just need quick test data? Generate Random Data

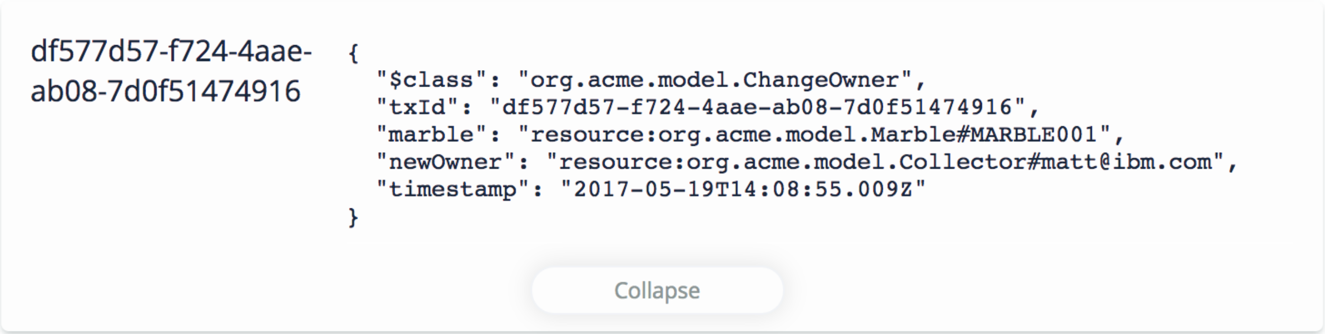
If successful the following dialogue will appear:



Submit Transaction Successful

A transaction was successfully submitted

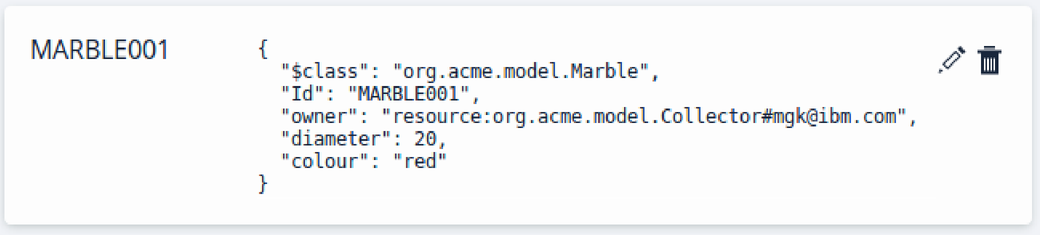
A transaction entry will also appear:



df577d57-f724-4aae-ab08-7d0f51474916

```
{
  "$class": "org.acme.model.ChangeOwner",
  "txId": "df577d57-f724-4aae-ab08-7d0f51474916",
  "marble": "resource:org.acme.model.Marble#MARBLE001",
  "newOwner": "resource:org.acme.model.Collector#matt@ibm.com",
  "timestamp": "2017-05-19T14:08:55.009Z"
}
```

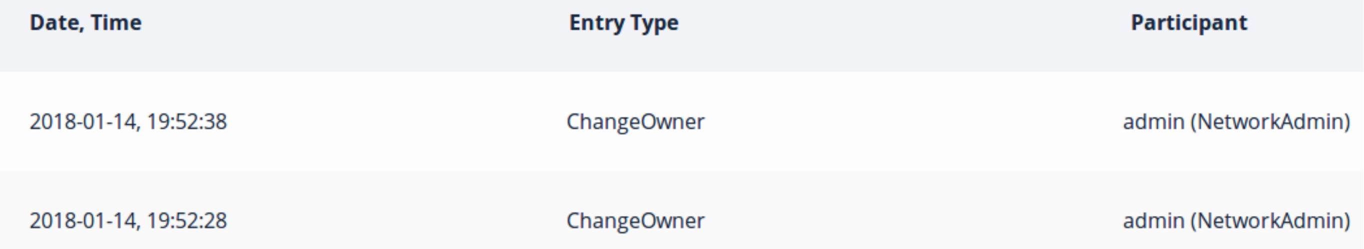
If you go back to the Marble, you will find its record has also been updated:



MARBLE001

```
{
  "$class": "org.acme.model.Marble",
  "Id": "MARBLE001",
  "owner": "resource:org.acme.model.Collector#mgk@ibm.com",
  "diameter": 20,
  "colour": "red"
}
```

Now select the “All Transactions” tab and look at the information the Composer “Historian” provides



Date, Time	Entry Type	Participant
2018-01-14, 19:52:38	ChangeOwner	admin (NetworkAdmin)
2018-01-14, 19:52:28	ChangeOwner	admin (NetworkAdmin)

.

The historian keeps a record of the details of all submitted composer transactions.

Section 4. Access Control

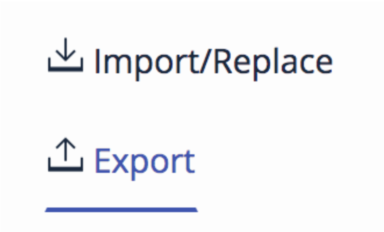
In this section we will explore how Hyperledger Composer restricts access to the resources on the network and the ability to modify them. Unlike previous sections, we will look at the creation of ACLs outside of Composer Playground and as such this section will cover how to export your business network to edit it locally in VSCode. This section will also not feature a demonstration of ACLs in practice - this will be covered in the next Lab when identites are covered (ACLs will be revisited then as well, looking at their effects).

3.1. Access Control Lists

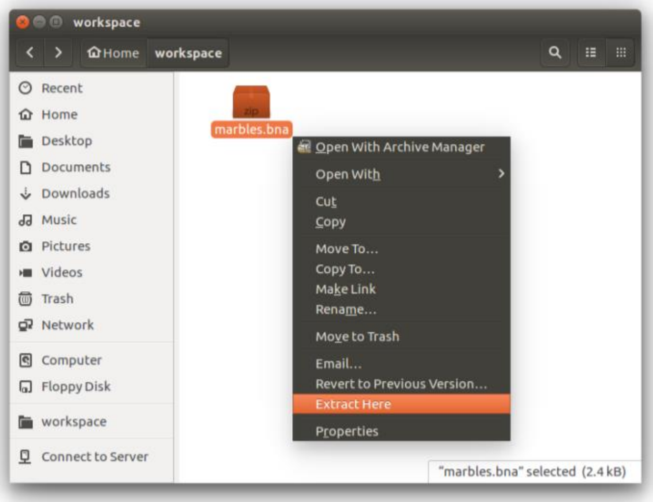
We now have some digital assets defined and the ability to move them between users. However, in a real system it would likely be the case that the `Marble` objects would not be available for all to see and if they were they would not be available for just anyone to change the ownership of.

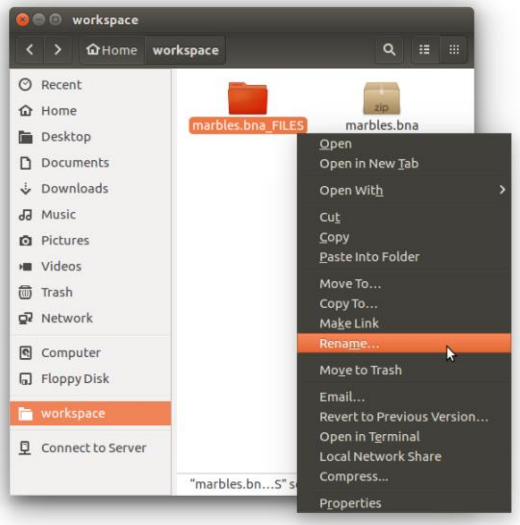
a. Export the Business Network

From the "Define" tab, go to the bottom-left of the screen and select **Export**:



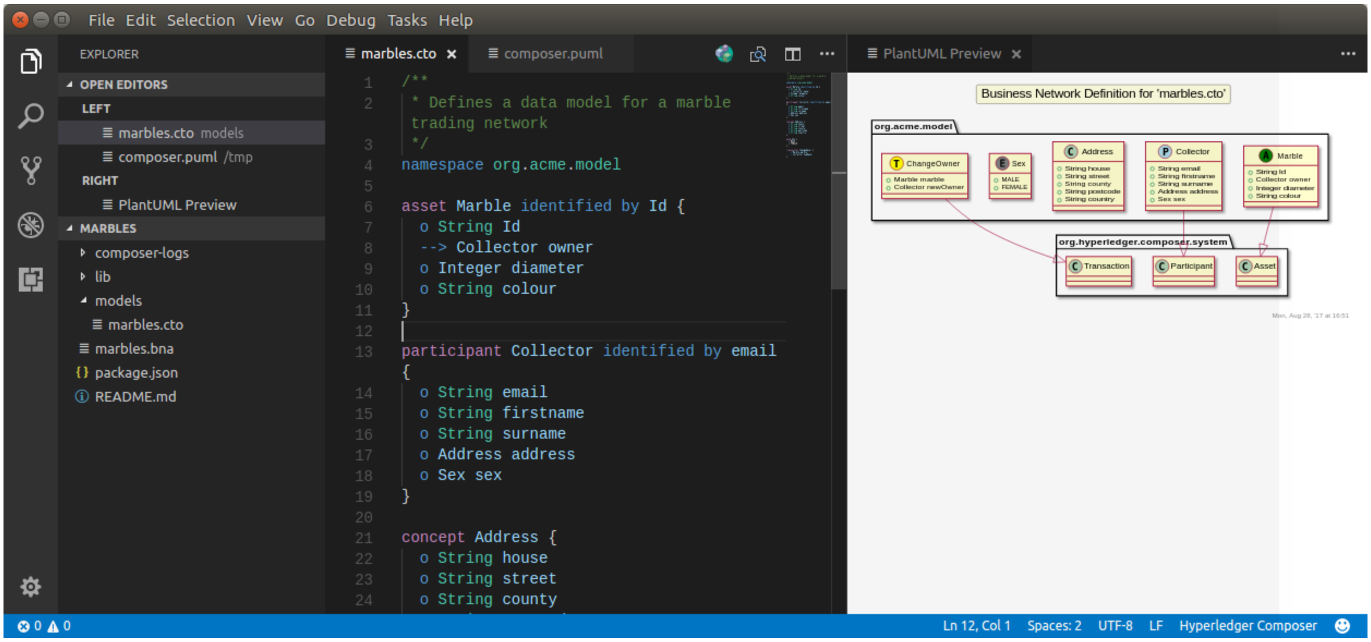
Rename the extension of the file to .zip and extract it in your workspace. Rename the extracted folder *marbles*:





Open the marbles folder in Visual Studio code. You can do this in two different ways:
1: right-click on the folder and choose "Open With -> Visual Studio Code"
2: at the cmd prompt type "code ~/workspace/marbles"

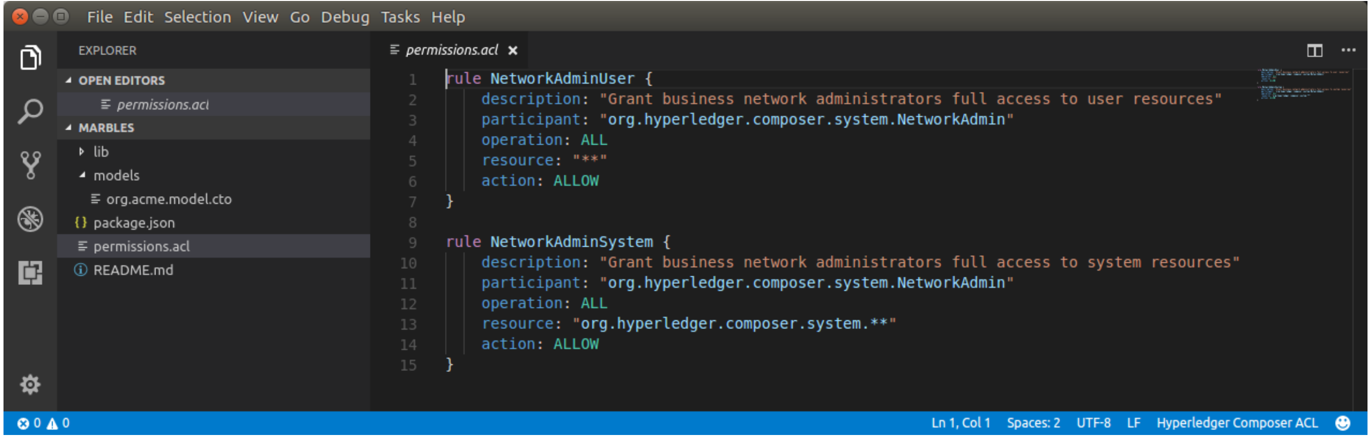
Next open up the `models/org.acme.model.cto` file to see the highlighting in VSCode. You can right click and choose "Generate UML" to see a visual representation of the model.



```
/**
 * Defines a data model for a marble
trading network
 */
namespace org.acme.model

asset Marble identified by Id {
  o String Id
  --> Collector owner
  o Integer diameter
  o String colour
}

participant Collector identified by email
{
  o String email
  o String firstname
  o String surname
  o Address address
  o Sex sex
}

concept Address {
  o String house
  o String street
  o String county
```

b. Open up the default ACL file

Open the default "`permissions.acl`" file from the root of the marbles folder in VSCode:



```
rule NetworkAdminUser {
    description: "Grant business network administrators full access to user resources"
    participant: "org.hyperledger.composer.system.NetworkAdmin"
    operation: ALL
    resource: "**"
    action: ALLOW
}

rule NetworkAdminSystem {
    description: "Grant business network administrators full access to system resources"
    participant: "org.hyperledger.composer.system.NetworkAdmin"
    operation: ALL
    resource: "org.hyperledger.composer.system.**"
    action: ALLOW
}
```

ACLs in Composer are files containing sets of rules dictating what actions participants are permitted to perform on certain resources and the conditions under which they can occur.

c. ACL Grammar

ACL Rules are of the following format:

```
rule <Rule Name> {
    description: <description of the rule>
    participant(p): <namespace and name of the participant performing the action>
    operation: <operation the participant wishes to perform>
    resource(r): <resources the operation is being performed on>
    condition: (<condition under which this rule applies>)
    action: <does this rule allow an operation or deny it>
}
```

In more detail:

Participant is the person or entity that has submitted the transaction.

Operation is what they wish to do to this resource, supported operations are `CREATE, READ, UPDATE, DELETE, ALL`

Resource is the asset that the transaction is being applied to. Resources (and indeed Participants) can simply be namespaces in which case they apply to all participants/resources in that namespace.

Condition is a JavaScript statement that can examine the participant and resource to check for certain conditions. Anything valid for use in an `if` statement is valid here.

Action is a simple ALLOW or DENY, as the name suggests this allows or denies the transaction.

d. Adding rules

Add the following rule at the top of the file, before the default NetworkAdmin rules. You can either copy this code directly or in VSCode press "`ctrl-space`" and choose the "Conditional ACL Rule" snippet and use the "`tab`" key to move through each field editing them the match the definition below:

```
rule OnlyOwnerCanUpdateAMarble {
  description: "Only an owner can edit a marble"
  participant(p): "org.acme.model.Collector"
  operation: UPDATE
  resource(r): "org.acme.model.Marble"
  condition: (r.owner.getIdentifier() == p.getIdentifier())
  action: ALLOW
}
```

This rule ensures that only the owners of `Marble` resources are able to edit them. It does this by `ALLOW`ing an `UPDATE` to `org.acme.model.Marble` resources only when the identifier of the `participant` and the `resource`'s owner are the same.

Apart from the Network default rules, all action is restricted unless explicitly permitted. As such while we do have a rule allowing updates of a `Marble` resource even the owner would be unable to read it. Add the following:

```
rule AnyoneCanReadMarbles {
  description: "Anyone can read a marble"
  participant(p): "org.acme.model.Collector"
  operation: READ
  resource(r): "org.acme.model.Marble"
  condition: (true)
  action: ALLOW
}
```

This rule allows all `Collector` participants to `READ` all `Marbles`. While we're at it we'll also need a rule to let `Collectors` read each other. The `ChangeOwner` transaction requires a submission of the identifier of a new `owner` which will not be possible if `Collectors` cannot read each other:

```
rule AnyoneCanReadCollectors {
    description: "Collectors can read"
    participant(p): "org.acme.model.Collector"
    operation: READ
    resource(r): "org.acme.model.Collector"
    condition: (true)
    action: ALLOW
}
```

Rules apply to transactions too. In addition to the first rule we wrote a second rule will need to be written to allow participants access to the tools needed to make the changes. In particular, we need to allow participants to create change owner transactions:

```
rule AnyoneCanIssueChangeOwner {
  description: "Participants can change owners"
  participant(p): "org.acme.model.Collector"
  operation: CREATE
  resource(r): "org.acme.model.ChangeOwner"
  condition: (true)
  action: ALLOW
}
```

Even with this rule, the transaction could be created by a non-owner but it would be rejected as they lack update access.

Finally, add a rule that allows Collectors access to the Network and update the Historian's record of transactions issued:

```
rule CollectorsCanInteractWithTheNetwork {
  description: "Collectors can interact with the network"
  participant: "org.acme.model.Collector"
  operation: READ,UPDATE,CREATE
  resource: "org.hyperledger.composer.system.**"
  action: ALLOW
}
```

This concludes ACL creation. Including the two default Network rules, there should now be seven rules in the ACL file. The next lab will show them in action.

Notices

This information was developed for products and services offered in the U.S.A.
IBM may not offer the products, services, or features discussed in this document in other countries.

Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.
IBM may have patents or pending patent applications covering subject MGKer described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental. All references to fictitious companies or individuals are used for illustration purposes only.

Appendix A. Trademarks and copyrights

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM AIX CICS ClearCase ClearQuest Cloudscape
Cube Views DB2 developerWorks DRDA IMS IMS/ESA
Informix Lotus Lotus Workflow MQSeries OmniFind
Rational Redbooks Red Brick RequisitePro System i
System z *Tivoli* *WebSphere* *Workplace* *System p*

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

ITIL is a registered trademark, and a registered community trademark of The Minister for the Cabinet Office, and is registered in the U.S. Patent and Trademark Office.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

Linear Tape-Open, LTO, the LTO Logo, Ultrium, and the Ultrium logo are trademarks of HP, IBM Corp. and Quantum in the U.S. and other countries.

NOTES

NOTES