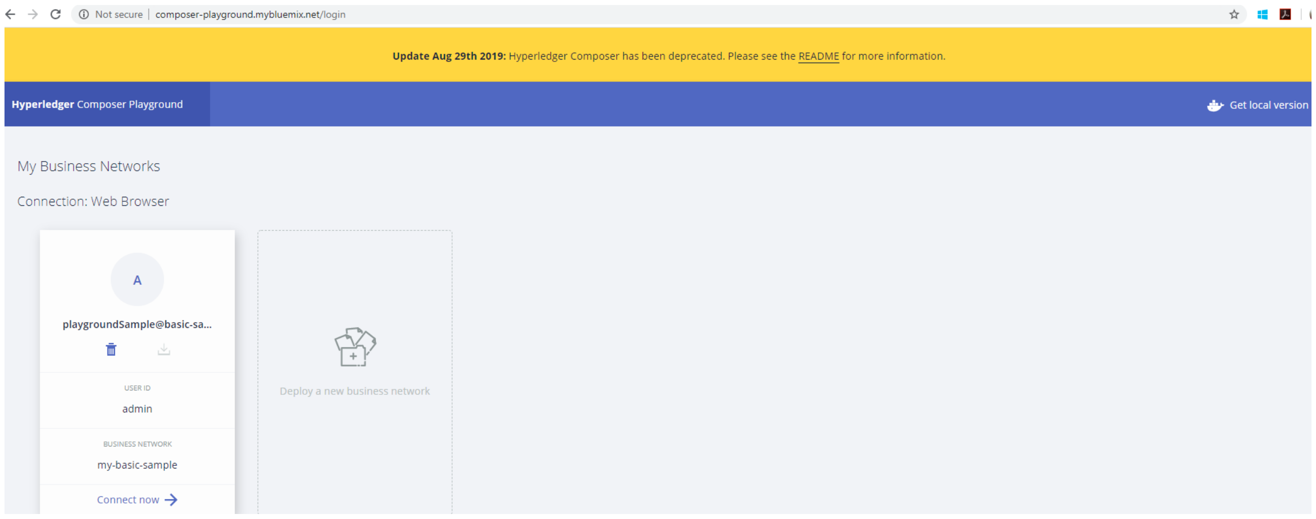
Building a practical Blockchain App

Thanks for participating in this Hands-On Lab during TechTogether Boston 2020. After following this hands-on lab, you will have experience with Hyperledger Composer.

➔ Open the browser (Firefox) by clicking on the third icon from the top in the taskbar.

➔ Click on the bookmark called **Hyperledger Composer** or go to the URL: http://composer-playground.mybluemix.net/login



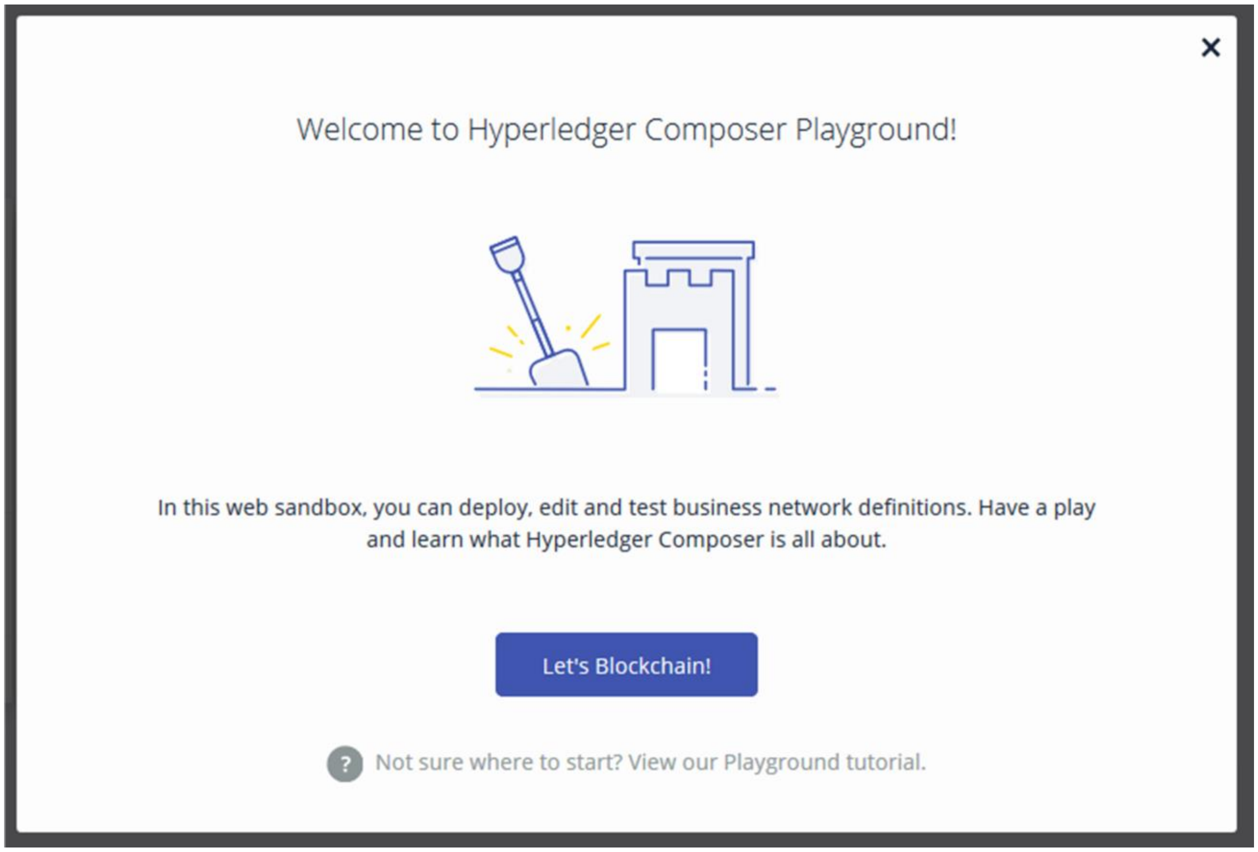
Not secure | composer-playground.mybluemix.net/login

Update Aug 29th 2019: Hyperledger Composer has been deprecated. Please see the README for more information.

Step 1: Create your first blockchain

This will bring you to the welcome page of the Hyperledger Composer Playground. In this web UI, you can deploy, edit and test business network definitions. You can test and play with Hyperledger Composer and learn how easy it is to create your own business network.

➔ `Click the` **`Let's Blockchain`** `button to start`

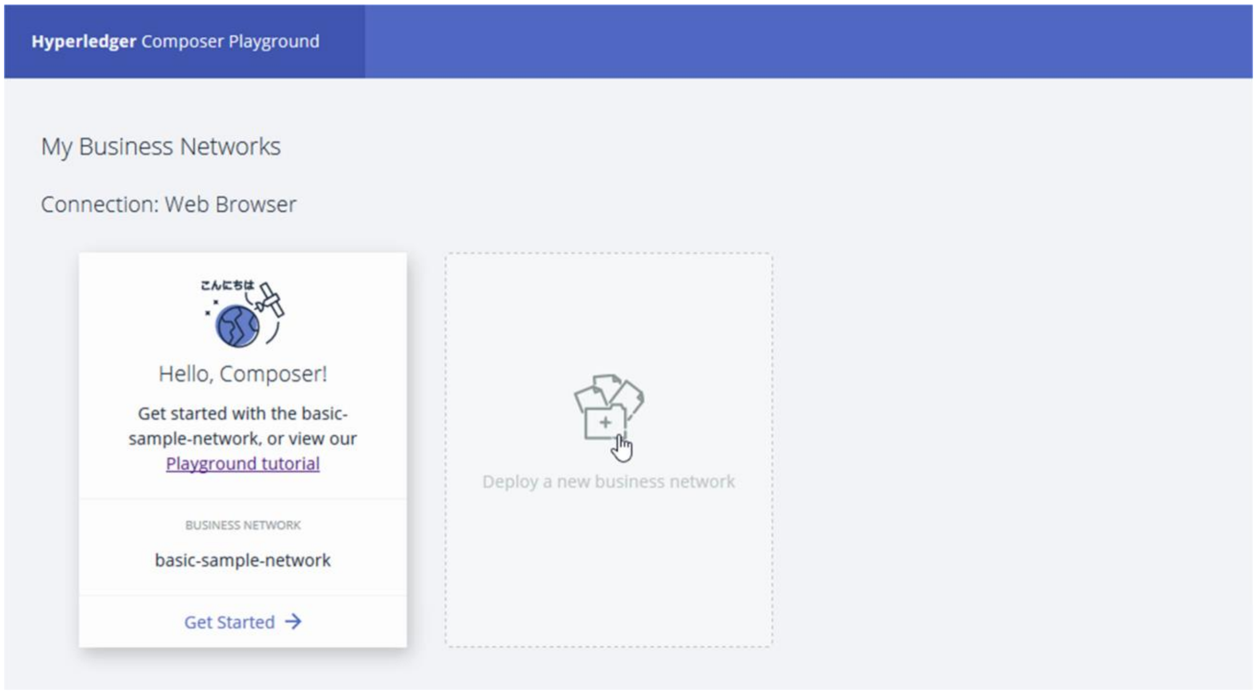


Welcome to Hyperledger Composer Playground!

In this web sandbox, you can deploy, edit and test business network definitions. Have a play and learn what Hyperledger Composer is all about.

From here you can create a basic-sample-network, create a new business network or import an existing business network. In our case we are going to create a new business network.

➔ `Click on the box "`**`Deploy a new business network`**`"`

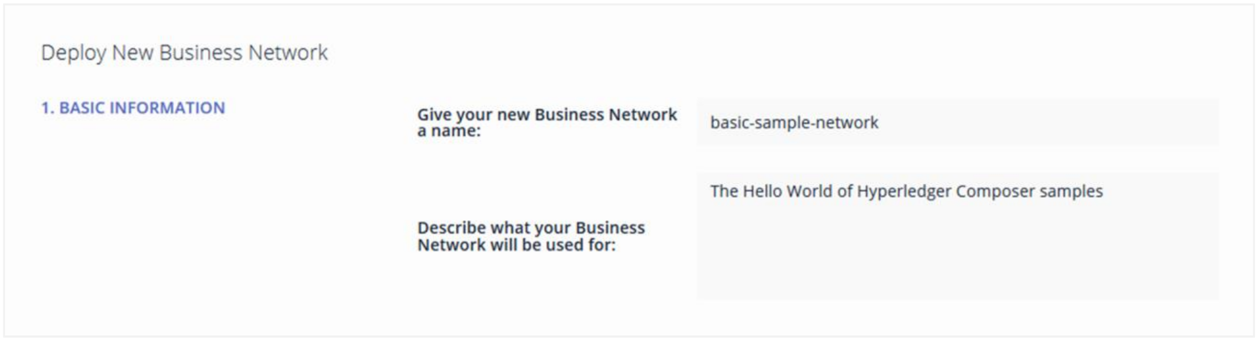

Hyperledger Composer Playground

My Business Networks

Connection: Web Browser

こんにちは
Hello, Composer!
Get started with the basic-sample-network, or view our Playground tutorial
BUSINESS NETWORK
basic-sample-network
Get Started
Deploy a new business network

We are navigated to a new page where we can define are new business network. Your network is identified by a name, which should be unique in the playground editor if you don't want to override an already existing one with the same name. Also, you can describe what your network will be used from.


Deploy New Business Network

1. BASIC INFORMATION

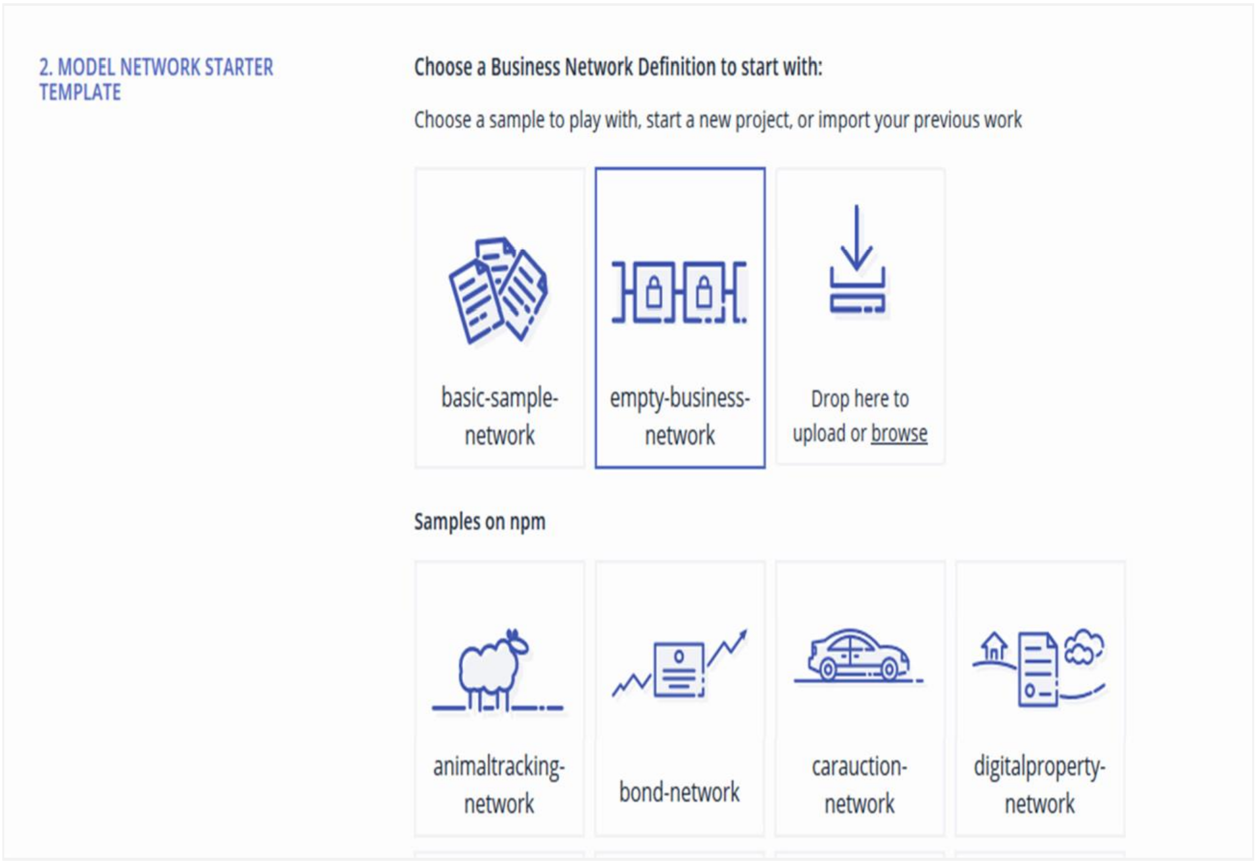
Give your new Business Network a name: basic-sample-network

Describe what your Business Network will be used for: The Hello World of Hyperledger Composer samples

➔ Complete the basic information with the following details:

Give your new Business Network a name	`vehicle-accident-registry`
Describe what your Business Network will be used for	`Business network to register accidents with vehicles`

If you scroll down, you can choose the starter template to create your network definition. You can base your network on the basic-sample-network template, on samples available on npm or on an empty business network. In our case we will use the latter one, but you can always play with the existing samples later.


2. MODEL NETWORK STARTER TEMPLATE
Choose a Business Network Definition to start with:
Choose a sample to play with, start a new project, or import your previous work
basic-sample-network
empty-business-network
Drop here to upload or browse
Samples on npm
animaltracking-network
bond-network
carauction-network
digitalproperty-network

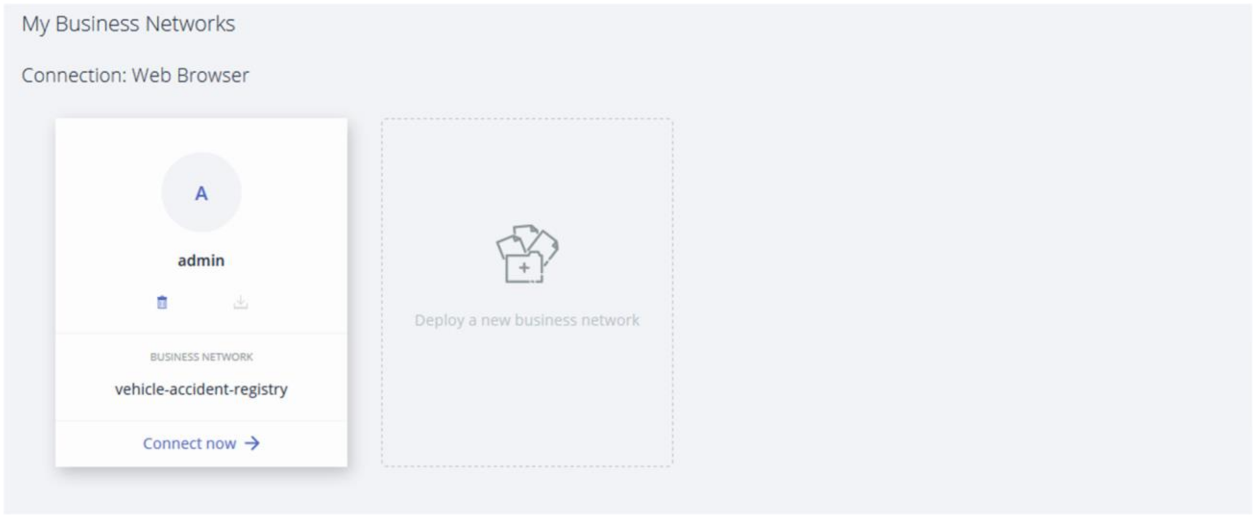
Notice when you click on the available models in the right corner of the page the details of the network are displayed including the amounts of participants, assets and transactions.

➔ Select the **empty-business-network** template and click on the **Deploy** button to create an empty network project.

Now that we've created and deployed the business network, you should see a new business network card called admin for our business network *vehicle-accident-registry* in your wallet. The wallet can contain business network cards to connect to multiple deployed business networks.

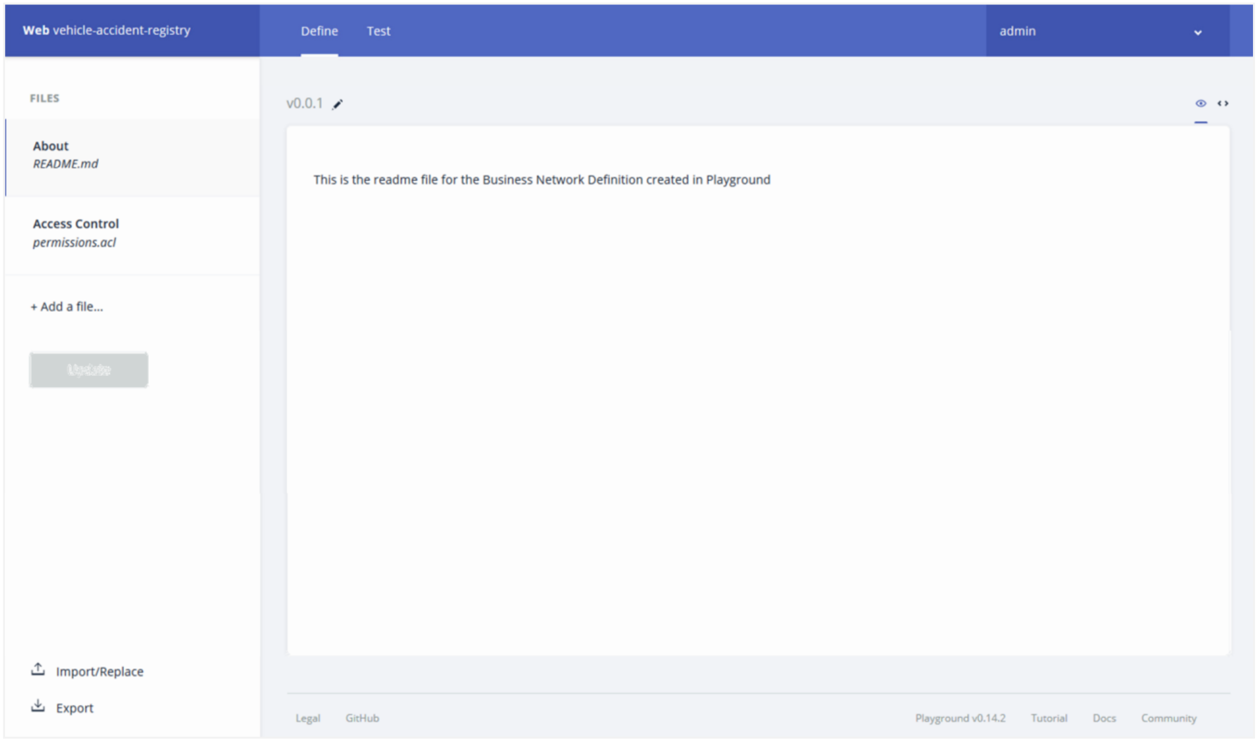
Step 2: Connect to your network

When connecting to an external blockchain, business network cards represent everything necessary to connect to a business network. They include connection details, authentication material, and metadata.



➔ `Click on` **`Connect now`** `link at the bottom of your vehicle-accident-registry network to start.`

You are now connected to your business network. We are now in the Composer editor. As you can see, we're in the Define tab right now, this tab is where you create and edit the files that make up a business network definition, before deploying them and testing them using the Test tab.



This is the readme file for the Business Network Definition created in Playground

As we selected an empty business network template, we need to define our business network files. The first step is to add our model files. Model files define the assets, participants, transactions, and events in our business network.

Step 3: Create your first model

For our network we are going to define four different model files for our assets, participants and transactions. One for each of the following entities; vehicles, persons, companies and accidents. Let's start with our *persons* model.

1. Click the Add a file button at the left side of the editor.
2. Select Model File (.cto) and click the Add button.
3. Click on the pencil icon next to the Model File name and change it to persons.cto.
4. Delete all the lines and replace with the lines below and click Update to save changes.

```
/**
* Contains the definitions for persons
*/
namespace nl.amis.registry.persons
participant Person identified by nationalNumber {
o String nationalNumber
o String firstName
o String lastName
}
participant Driver extends Person {
  o String driverLicence
  o String phoneNumber optional
}
participant Responder extends Person {
  o Integer batchNumber
}
participant CaseWorker extends Person {
o String phoneNumber optional
}
```

Create companies asset model

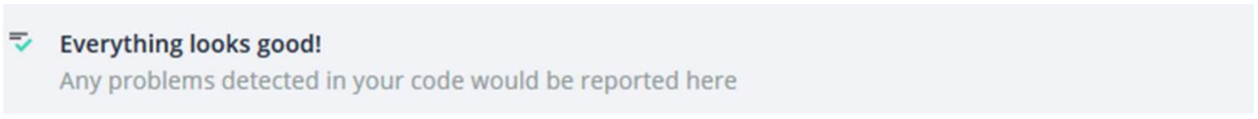
Next create a new model file for the companies involved. The companies we define are modelled as assets (i.e. objects or classes).

1. Create / Add a new model file and change the name to companies.cto.
2. Delete all the lines and replace with the lines below and click Update to save changes.

```
/**
* Contains the definitions for companies
*/
namespace nl.amis.registry.companies
import nl.amis.registry.persons.Person
import nl.amis.registry.persons.Responder
import nl.amis.registry.accidents.Accident

abstract asset Company identified by name {
o String name
}

asset InsuranceCompany extends Company {
  --> Person contact optional
  --> Accident[] cases optional
}

asset EmergencyService extends Company {
  --> Responder[] responders
}
```

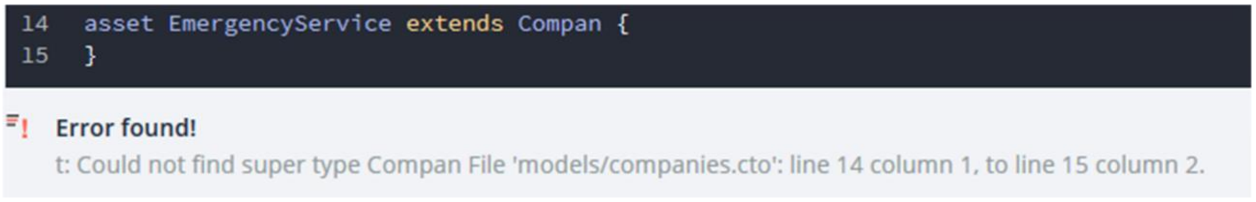
Validate your model file

In the background a validator runs constantly to check your model definition code. Check the validation message at the bottom of the code editor to know if you modelled it correctly.



Everything looks good!
Any problems detected in your code would be reported here

If you encounter an error check if all assets are correctly defined, e.g. extending wrong asset name.



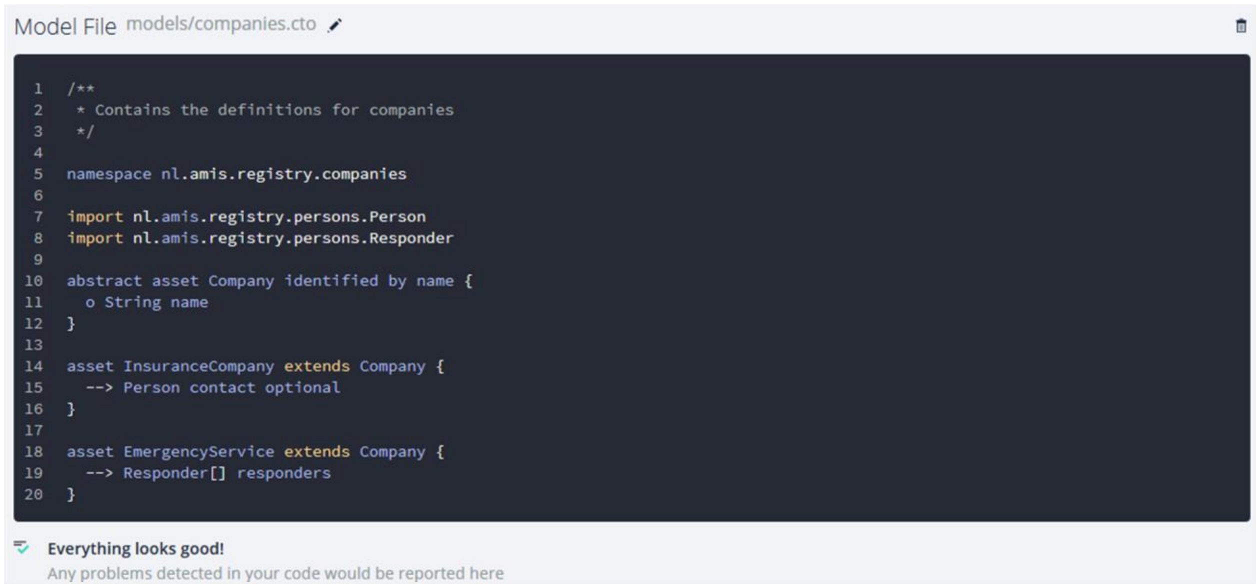
```
14  asset EmergencyService extends Compan {
15  }
```

Error found!
t: Could not find super type Compan File 'models/companies.cto': line 14 column 1, to line 15 column 2.

Extend the Company asset with relations

The company asset has relation with both persons and accidents (we will create that relation later on). Just like in other programming languages this is done using imports. In our case the companies model uses objects (i.e participants) from the persons model.

The current companies model should look like the following definition



```
/**
 * Contains the definitions for companies
 */

namespace nl.amis.registry.companies

import nl.amis.registry.persons.Person
import nl.amis.registry.persons.Responder

abstract asset Company identified by name {
  o String name
}

asset InsuranceCompany extends Company {
  --> Person contact optional
}

asset EmergencyService extends Company {
  --> Responder[] responders
}
```

Everything looks good!
Any problems detected in your code would be reported here

Create vehicles asset model

Next create a new model file for the vehicles involved. The vehicles we define are modelled as assets. This time we want you to model the file completely.

```
namespace nl.amis.registry.vehicles

import nl.amis.registry.persons.Driver
import nl.amis.registry.companies.InsuranceCompany

asset Vehicle identified by vin {
  o String vin
  o String brand
  o String licensePlate
  --> InsuranceCompany insurer
  --> Driver owner
}
```

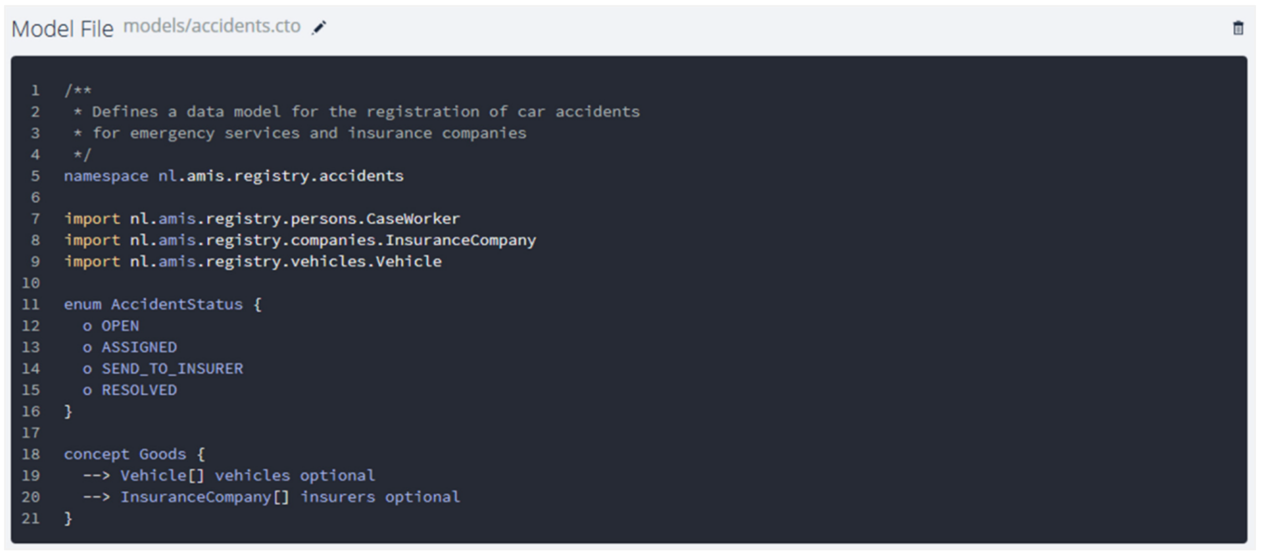
Create Accident asset model

The last model file we need to create is the model for the actual accident registrations. This file will also contain our transactions later in this lab.

1. Create / Add a new model file and change the name to accidents.cto.
2. Delete all the lines and replace with the lines below and click Update to save changes.

```
/**
* Defines a data model for the registration of car
accidents
* for emergency services and insurance companies
*/
namespace nl.amis.registry.accidents
import nl.amis.registry.persons.CaseWorker
import nl.amis.registry.companies.InsuranceCompany
import nl.amis.registry.vehicles.Vehicle

enum AccidentStatus {
  o OPEN
  o ASSIGNED
  o SEND_TO_INSURER
  o RESOLVED
}

concept Goods {
  --> Vehicle[] vehicles optional
  --> InsuranceCompany[] insurers optional

}

concept Location {
  o Long longitude
  o Long latitude
  o String description
}
```

If you modelled everything correctly the model should look like the following image:



```
/**
 * Defines a data model for the registration of car accidents
 * for emergency services and insurance companies
 */
namespace nl.amis.registry.accidents

import nl.amis.registry.persons.CaseWorker
import nl.amis.registry.companies.InsuranceCompany
import nl.amis.registry.vehicles.Vehicle

enum AccidentStatus {
  o OPEN
  o ASSIGNED
  o SEND_TO_INSURER
  o RESOLVED
}

concept Goods {
  --> Vehicle[] vehicles optional
  --> InsuranceCompany[] insurers optional
}
```

Now that we have defined these types we can model the Accident *asset*.

3. Add the following definition at the bottom of your model file and click Update to save changes.

```
asset Accident identified by accidentId {
  o String accidentId
  o String description
  o AccidentStatus status
  o Location location
  o Goods goods
  --> CaseWorker assignee optional
}
```

4. Finally go back to your companies.cto model file
   a. Add the import for the *{nl.amis.registry.accidents}* Accident asset.
   b. Add an *optional relation* to an *Array* of Accident called *cases* for InsuranceCompany.
   c. Click Update to save your progress.

Create Accident transactions

Congratulation the modelling of our participants and assets are done! The final thing we have to do to this current model is to define our transactions. Transactions let us interact with and transact assets.

For our application we will define four transactions:

- RegisterAccident – which lets a responder register new accident reports
- AssignToCaseWorker – which lets an CaseWorker to be assigned to the report
- SendToInsurance – which lets an CaseWorker give insurance companies access to the report
- ResolveAccident – which lets an contact person from an insurance company close the report

We are going to add these transactions to the model file that holds the main asset we are exchanging or interacting with. In our case this is the accidents.cto model file.

1. Open the accident.cto model file and add the following two transactions at end of the file:

```
transaction RegisterAccident  {
  o Accident accident
}

transaction AssignToCaseWorker {
  --> Accident accident
  --> CaseWorker assignee
}

transaction SendToInsurance {
o String accidentId
-->InsuranceCompany[] insurers
}

transaction ResolveAccident {
o String accidentId
}
```

Step 4: Adding a script file for processing transactions

We are now done with defining the domain model, and we can define the transition logic for the business network. The logic for a business network in the Composer is expressed using JavaScript functions. These functions are automatically executed when a transaction is submitted for processing.

1. Click the Add a file button at the left side of the editor.
2. Select Script File (.js) and click the Add button.
3. Click on the pencil icon next to the Model File name and change it to logic.js.
4. Delete all the lines and replace with the lines below and click Update to save changes.

```
/**
 * Create a new accident
 * @param {nl.amis.registry.accidents.RegisterAccident} newAccident - the new
accident to be registered
 * @transaction
 */
function RegisterAccident(newAccident) {

  // make a new resource containing the information given to the function
  // save this new resource in the registry
  return getAssetRegistry('nl.amis.registry.accidents.Accident')
      .then(function (registry) {
          var factory = getFactory();
          var newAccidentAsset =
              factory.newResource('nl.amis.registry.accidents',
              'Accident', newAccident.accident.accidentId);
          newAccidentAsset = newAccident.accident;
          return registry.add(newAccidentAsset);
      });
}
```

The above function simply creates a new Accident in the asset registry based on the given RegisterAccident transaction.

Add the second function AssignToCaseWorker by adding the following lines to the script file.

```
/**
* Assign an Administrator to the accident report
* @param {nl.amis.registry.accidents.AssignToCaseWorker} assignToCaseWorker -
the particular accident that you want to assign to a responsible Case worker
* @transaction
*/
function AssignToCaseWorker(assignToCaseWorker) {
if (assignToCaseWorker.accident.status !== 'OPEN')
throw new Error('Case already assigned to an Case worker.');
else {
assignToCaseWorker.accident.status = 'ASSIGNED';
assignToCaseWorker.accident.assignee = assignToCaseWorker.assignee;
}
return getAssetRegistry('nl.amis.registry.accidents.Accident')
.then(function (assetRegistry) {
return assetRegistry.update(assignToCaseWorker.accident);
});
}
```

5. Add the third function SendToInsurance by adding the following lines to the script file.

```
function SendToInsurance(sendToInsurance) {
 var assetRegistry;
 var accident;
 return getAssetRegistry('nl.amis.registry.accidents.Accident')
        .then(function(ar) {
               assetRegistry = ar;
               return assetRegistry.get(sendToInsurance.accidentId);
        })
        .then(function(acc){
               accident = acc;
               if (accident.status == 'OPEN')
                      throw new Error('No case worker has been assigned to the report
yet.');
               else if (accident.status == 'SEND_TO_INSURER')
                      throw new Error('This case has already been sent to one or more
insurers.');
               else {
                 // update ACCIDENT
                 accident.status = 'SEND_TO_INSURER';
                 if (!accident.goods.insurers) accident.goods.insurers = [];
                 accident.goods.insurers = sendToInsurance.insurers;
                              for (var i = 0, len = accident.goods.insurers.length;
i < len; i++)
                 {
                 if (!accident.goods.insurers[i].cases)
                 accident.goods.insurers[i].cases - [];
                 accident.goods.insurers[i].cases.push(accident);
                 }
                 return assetRegistry.update(accident);
               }
        })
        .then(function() {
               return getAssetRegistry('nl.amis.registry.companies.InsuranceCompany')
        })
        .then(function(assetRegistry) {
               return assetRegistry.updateAll(accident.goods.insurers);
 });
 }
```

6. Add the last function ResolveAccident by adding the following lines to the script file and click Update to save your progress.

```
/**
* Update status when accident is resolved
* @param {nl.amis.registry.accidents.ResolveAccident} resolveAccident - the
particular accident that should be resolved
* @transaction
*/
function ResolveAccident(resolveAccident) {
var assetRegistry;
return getAssetRegistry('nl.amis.registry.accidents.Accident')
.then(function(ar) {
assetRegistry = ar;
return assetRegistry.get(resolveAccident.accidentId);
})
.then(function(accident){
if (accident.status == 'RESOLVED')
throw new Error('Case is already resolved.');
if (accident.status !== 'SEND_TO_INSURER')
throw new Error('Case report not yet sent to insurer(s).');
else
accident.status = 'RESOLVED';
return assetRegistry.update(accident);
});
}
```

Step 5: Access control / Permissions

Hyperledger Composer includes an access control language (ACL) that provides declarative access control over the elements of the domain model.

➔ `In the composer UI click on the` **`Access Control`** `file.`

Add the following access control rules. The first rule will give the Reponder participants access to read data available in the network and the second rule will add all access rights to the RegisterAccident transaction to the Responder.

```
rule ResponderReadALL {
    description: "Allow the responder read access"
    participant: "nl.amis.registry.persons.Responder"
    operation: READ
    resource: "nl.amis.registry.accidents.*"
    action: ALLOW
}

rule ResponderRegisterAccident {
    description: "Allow the responder access to create accident reports"
    participant: "nl.amis.registry.persons.Responder"
    operation: ALL
    resource: "nl.amis.registry.accidents.Accident"
    transaction: "nl.amis.registry.accidents.RegisterAccident"
    action: ALLOW
}
```

Add the following access control rules. The first rule will give the CaseWorker all rights on the available accident reports. And the second rule will add read access on the available accident reports for the contact persons of the insurance

```
rule CaseWorker {
    description: "Allow the admin full access"
    participant: "nl.amis.registry.persons.CaseWorker"
    operation: ALL
    resource: "nl.amis.registry.accidents.*"
    transaction: "nl.amis.registry.accidents.*"
    action: ALLOW
}

rule Person {
    description: "Allow contacts from insurance companies, read access"
    participant: "nl.amis.registry.persons.Person"
    operation: READ
    resource: "nl.amis.registry.accidents.*"
    action: ALLOW
}
```

You can play with these access rules later your own, but for sake of the hands-on lab at a SystemACL rule to give all participants all rights on the network.

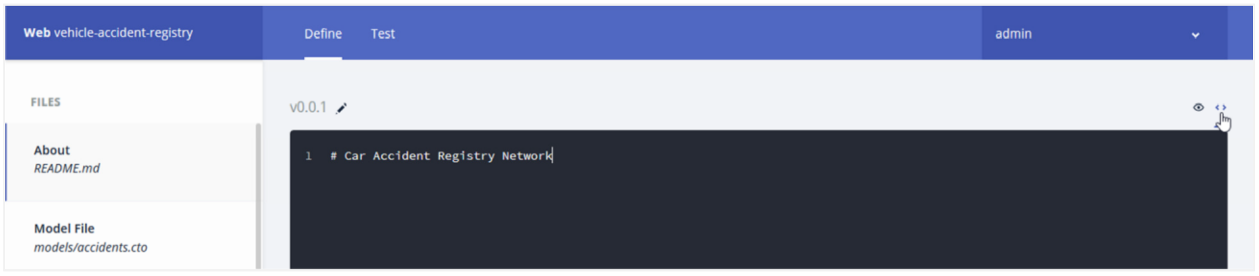
```
rule SystemACL {
  description:  "System ACL to permit all access"
  participant: "org.hyperledger.composer.system.Participant"
  operation: ALL
  resource: "org.hyperledger.composer.system.**"
  action: ALLOW
}
```

➔ In the composer UI click the **Update** button the save your progress.

Step 6: Changing the README.md

To help your network admins and developers to understand your business network you can edit the About (README.MD) file.

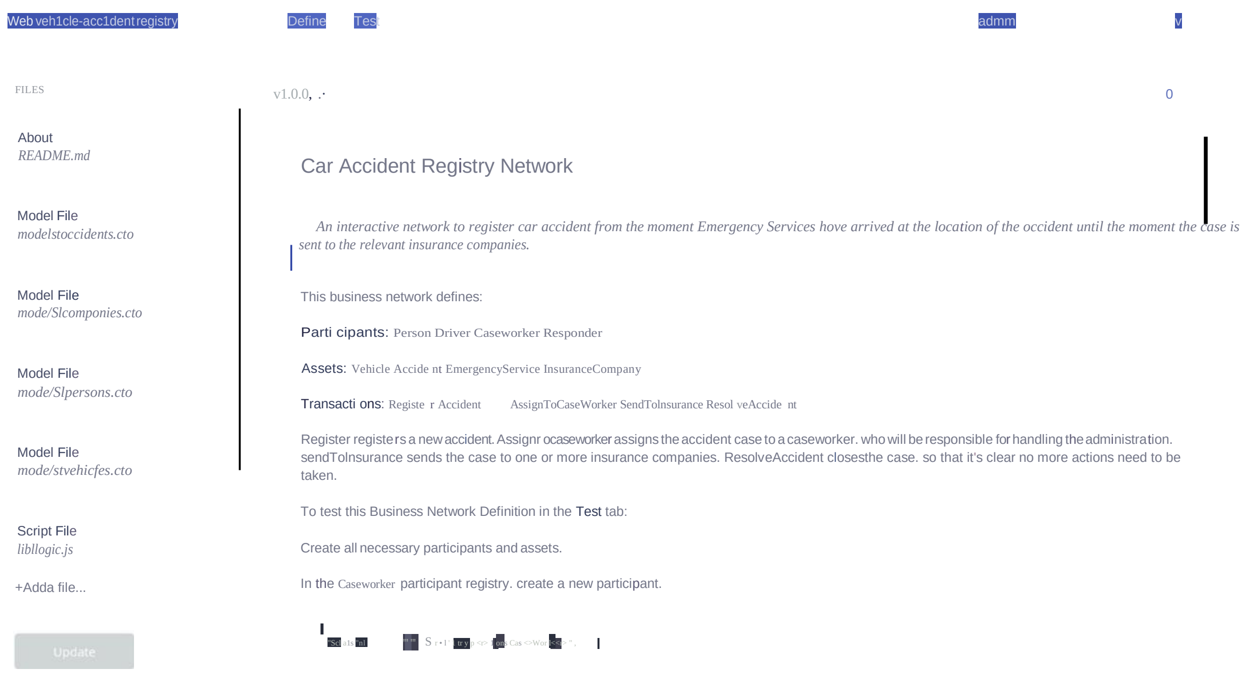
➔ In the composer UI click on the About file and then the code icon on the right side of the code editor.



```
# Car Accident Registry Network
```

Open the README.md file, which you can find on the desktop of the VM, and copy the contents of it into the code editor.

➔ To see the result, click on the eye icon on the right side of the code editor to switch back.



Car Accident Registry Network

An interactive network to register car accident from the moment Emergency Services hove arrived at the location of the occident until the moment the case is sent to the relevant insurance companies.

This business network defines:

Parti cipants: Person Driver Caseworker Responder

Assets: Vehicle Accide nt EmergencyService InsuranceCompany

Transacti ons: Registe r Accident AssignToCaseWorker SendToInsurance Resol veAccide nt

Register registers a new accident. Assignr ocaseworker assigns the accident case to a caseworker. who will be responsible for handling the administration. sendToInsurance sends the case to one or more insurance companies. ResolveAccident closesthe case. so that it's clear no more actions need to be taken.

To test this Business Network Definition in the Test tab:

Create all necessary participants and assets.

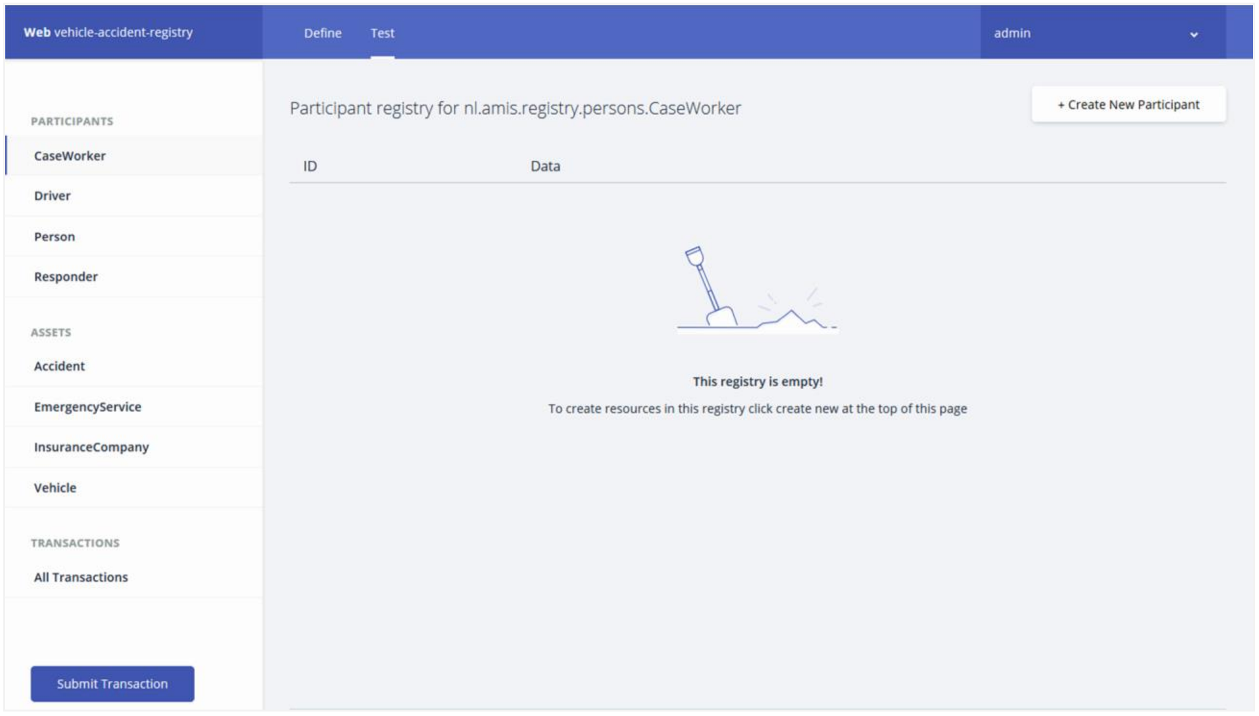
In the Caseworker participant registry. create a new participant.

"Scl a1s "nl [illegible] S r • 1' 1 tr y p <r> 1 ons Cas <>Wor l<<r> " ,

Testing your business network

Now that the business network is fully modelled and configured, we can test it using the Composer

➔ `In the composer UI click on the Test tab at the top of the page next to the Define tab.`



Participant registry for nl.amis.registry.persons.CaseWorker

This registry is empty!

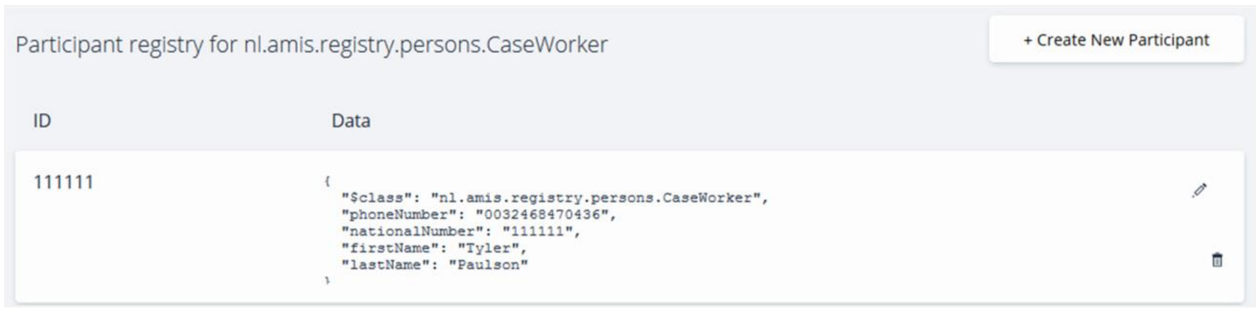
To create resources in this registry click create new at the top of this page

Step 1: Creating the necessary participants and assets

Follow the README.md file for more details to create all the required participants and assets.

1. In the CaseWorker participant registry, *create* a new participant.

```
{
  "$class": "nl.amis.registry.persons.CaseWorker",
  "nationalNumber": "111111",
  "firstName": "Tyler",
  "lastName": "Paulson",
  "phoneNumber": "0032468470436"
}
```


Participant registry for nl.amis.registry.persons.CaseWorker
+ Create New Participant
ID
Data

111111

```
{
  "$class": "nl.amis.registry.persons.CaseWorker",
  "phoneNumber": "0032468470436",
  "nationalNumber": "111111",
  "firstName": "Tyler",
  "lastName": "Paulson"
}
```

Do the same for the other participants that are going to be used in our tests.

2. In the Driver participant registry, *create* a new participant.

```
{
  "$class": "nl.amis.registry.persons.Driver",
  "nationalNumber": "222222",
  "firstName": "Robert",
  "lastName": "Durden",
  "driverLicense": "4268721476",
  "phoneNumber": "0032472830240"
}
```

3. In the Person participant registry, *create* a new participant.

```
{
  "$class": "nl.amis.registry.persons.Person",
  "nationalNumber": "333333",
  "firstName": "Marcus",
  "lastName": "Aurelius"
}
```

4. Finally, in the Responder participant registry, create a new participant.

```
{
  "$class": "nl.amis.registry.persons.Responder",
  "nationalNumber": "444444",
  "firstName": "John",
  "lastName": "Doe",
  "batchNumber": "4896524"
}
```

Now you have the minimal amount of participants needed to register a new accident properly: a driver who has had an accident, a caseworker to handle the registration of the accident and a contact from an insurance company to deal with the financial issues. Time to register the vehicle and companies.

5. In the EmergencyService asset registry, *create* a new asset.

```
{
  "$class": "nl.amis.registry.companies.EmergencyService",
  "name": "ResQ",
  "responders": [
    "resource:nl.amis.registry.persons.Responder#444444"
  ]
}
```

Notice how the company ResQ has *John Doe* as a responder. A EmergencyService can have multiple responders assigned to it, but the need to exists as assets in the registry.

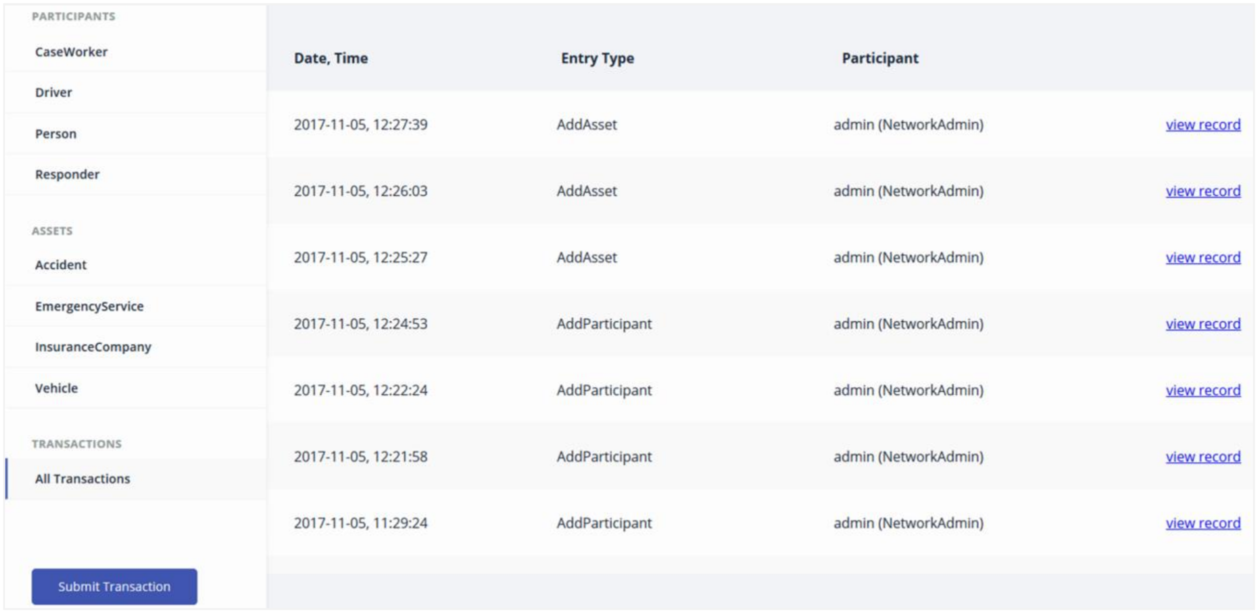
6. In the InsuranceCompany asset registry, *create* a new asset. The contact of insurance company BeSure is Marcus Aurelius, and this company has not yet had any cases to solve.

```
{
  "$class": "nl.amis.registry.companies.InsuranceCompany",
  "name": "BeSure",
  "contact":
    "resource:nl.amis.registry.persons.Person#333333",
  "cases": []
}
```

7. In the Vehicle asset registry, *create* a new asset of a vehicle owned by *Robert Durden* and insured by *BeSure*.

```
{
  "$class": "nl.amis.registry.vehicles.Vehicle",
  "vin": "1FUYDZYB4XPA21495",
  "brand": "Tesla P100D",
  "licensePlate": "0-GXS-248",
  "insurer":
    "resource:nl.amis.registry.companies.InsuranceCompany#BeSure",
  "owner":
    "resource:nl.amis.registry.persons.Driver#222222"
}
```

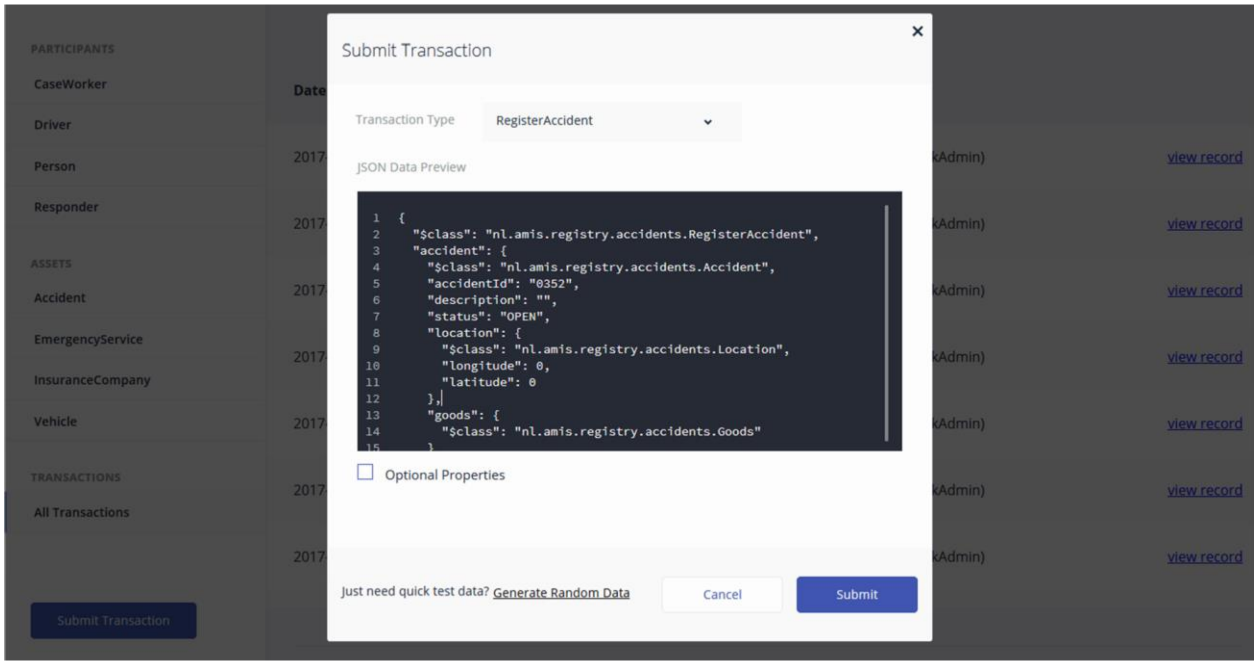
Creating, editing and deleting participants and assets result in transactions on the blockchain. You can view these transactions under the *section* All Transaction.



Date, Time	Entry Type	Participant	
2017-11-05, 12:27:39	AddAsset	admin (NetworkAdmin)	view record
2017-11-05, 12:26:03	AddAsset	admin (NetworkAdmin)	view record
2017-11-05, 12:25:27	AddAsset	admin (NetworkAdmin)	view record
2017-11-05, 12:24:53	AddParticipant	admin (NetworkAdmin)	view record
2017-11-05, 12:22:24	AddParticipant	admin (NetworkAdmin)	view record
2017-11-05, 12:21:58	AddParticipant	admin (NetworkAdmin)	view record
2017-11-05, 11:29:24	AddParticipant	admin (NetworkAdmin)	view record

Step 2: Submit and Accident transactions

Now you can register an accident that has happened to Robert Durden on the E19 near Scholten, Belgium. There are no other cars involved then its own. *Click* the Submit Transaction button and make sure RegisterAccident is selected. You can also choose for the other three transactions we can execute on the network.

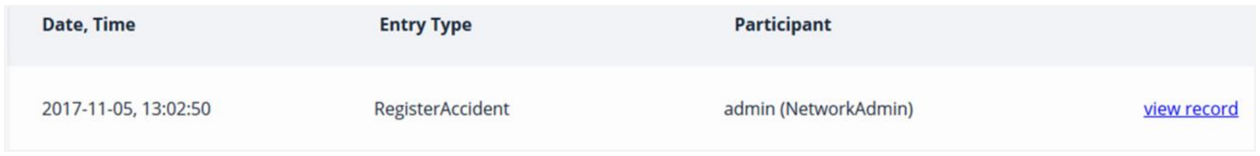


```
{
  "$class": "nl.amis.registry.accidents.RegisterAccident",
  "accident": {
    "$class": "nl.amis.registry.accidents.Accident",
    "accidentId": "0352",
    "description": "",
    "status": "OPEN",
    "location": {
      "$class": "nl.amis.registry.accidents.Location",
      "longitude": 0,
      "latitude": 0
    },
    "goods": {
      "$class": "nl.amis.registry.accidents.Goods"
```

1. For now, register a new accident in the registry:

```
{
  "$class": "nl.amis.registry.accidents.RegisterAccident",
  "accident": {
    "$class": "nl.amis.registry.accidents.Accident",
    "accidentId": "20171106JD1",
    "description": "Driver got a flat tire on the highway and hit the
crash barrier. Driver seems OK but vehicle has sustained damage.",
    "status": "OPEN",
    "location": {
      "$class": "nl.amis.registry.accidents.Location",
      "longitude": 4.4533999,
      "latitude": 51.2675439,
      "description": "E19 near Schoten."
    },
    "goods": {
      "$class": "nl.amis.registry.accidents.Goods",
      "vehicles": [
        "resource:nl.amis.registry.vehicles.Vehicle#1FUYDZYB4XPA21495"
      ]
    }
  }
}
```

TIP: You can add more Driver and Vehicle assets if you want and reference them in message above.

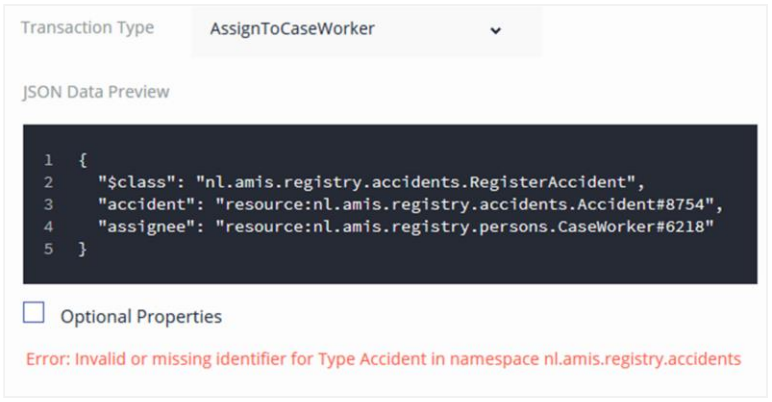


Date, Time	Entry Type	Participant	
2017-11-05, 13:02:50	RegisterAccident	admin (NetworkAdmin)	view record

The accident has been registered and to view the transaction details click on the link view record.

Step 3: Assign a Case worker

Time to assign a case worker to this case. *Click* the Submit Transaction button and select AssignToCaseWorker. Notice that the UI gives us an error. This is because the namespace it generated for the message is wrong.



Transaction Type: AssignToCaseWorker

JSON Data Preview

```
{
  "$class": "nl.amis.registry.accidents.RegisterAccident",
  "accident": "resource:nl.amis.registry.accidents.Accident#8754",
  "assignee": "resource:nl.amis.registry.persons.CaseWorker#6218"
}
```

Optional Properties

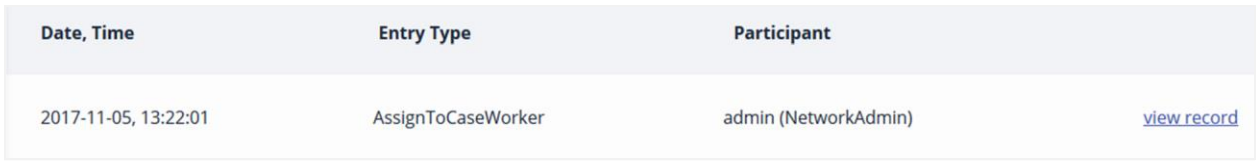
Error: Invalid or missing identifier for Type Accident in namespace nl.amis.registry.accidents

It should be nl.amis.registry.accidents.AssignToCaseWorker . The message itself has a relation to the accident you want to assign to the CaseWorker, which is also a relation.

2. Assign a case worker to the accident you just registered by submitting the following message:

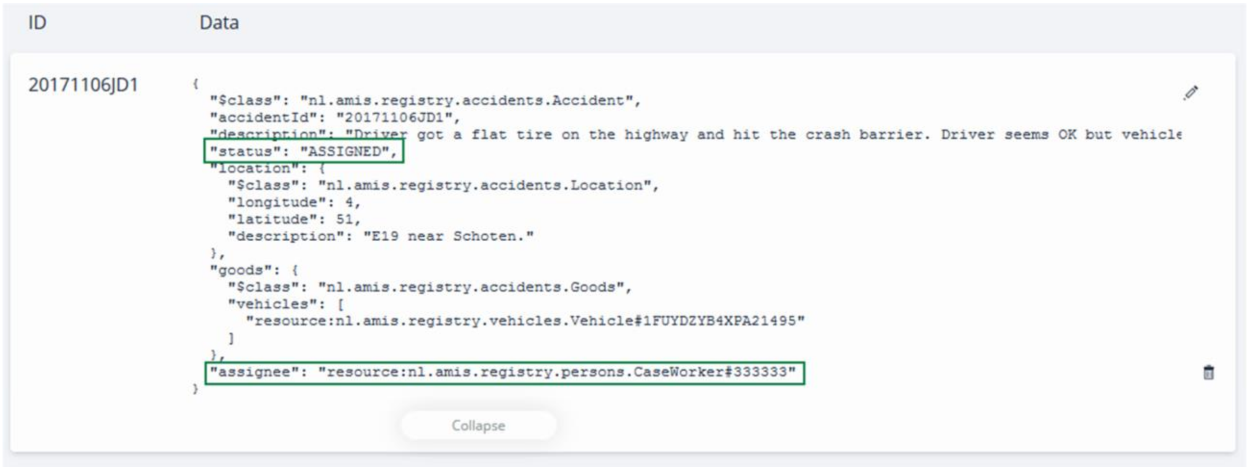
```
{
  "$class": "nl.amis.registry.accidents.AssignToCaseWorker",
  "accident":
"resource:nl.amis.registry.accidents.Accident#20171106JD1",
  "assignee": "resource:nl.amis.registry.persons.CaseWorker#333333"
}
```

This will register that Marcus Aurelius is on the case and enable the option to send the case to the driver's insurance company.



Date, Time	Entry Type	Participant	
2017-11-05, 13:22:01	AssignToCaseWorker	admin (NetworkAdmin)	view record

In the Accident registry you can verify that the status has changed from "OPEN" to "ASSIGNED".


ID Data

20171106JD1

```
{
  "$class": "nl.amis.registry.accidents.Accident",
  "accidentId": "20171106JD1",
  "description": "Driver got a flat tire on the highway and hit the crash barrier. Driver seems OK but vehicle
  "status": "ASSIGNED",
  "location": {
    "$class": "nl.amis.registry.accidents.Location",
    "longitude": 4,
    "latitude": 51,
    "description": "E19 near Schoten."
  },
  "goods": {
    "$class": "nl.amis.registry.accidents.Goods",
    "vehicles": [
      "resource:nl.amis.registry.vehicles.Vehicle#1FUYDZYB4XPA21495"
    ]
  },
  "assignee": "resource:nl.amis.registry.persons.CaseWorker#333333"
}
```

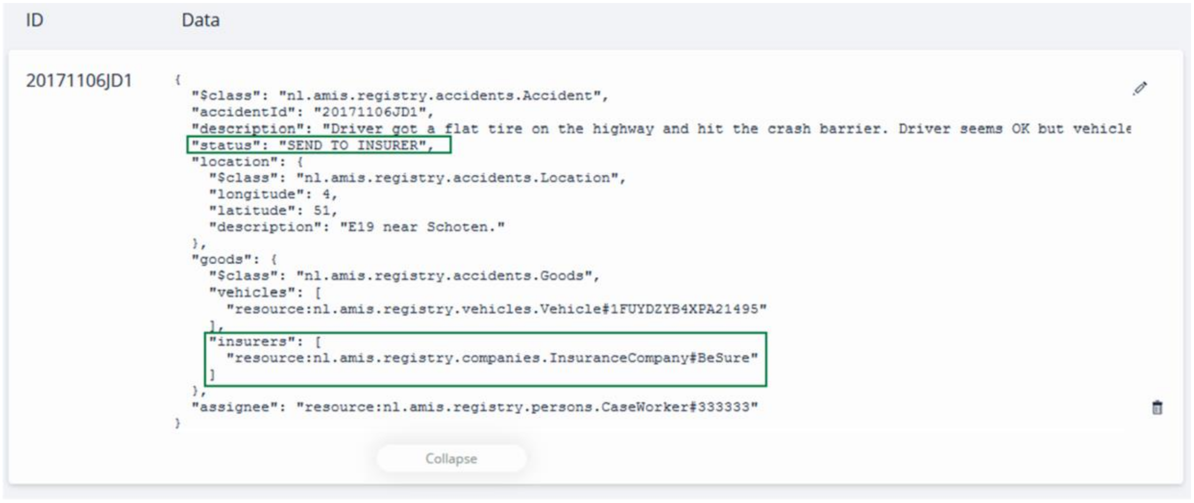
Step 4: Send case to insurance company

You can send the case to the insurance company by clicking the Submit Transaction button and selecting SendToInsurance. The accident ID is 20171106JD1 and be sure to send this accident to the BeSure company.

3. Send the case to the insurance company by submitting the following message:

```
{
  "$class": "nl.amis.registry.accidents.SendToInsurance",
  "accidentId": "20171106JD1",
  "insurers": [
    "resource:nl.amis.registry.companies.InsuranceCompany#BeSure"
  ]
}
```

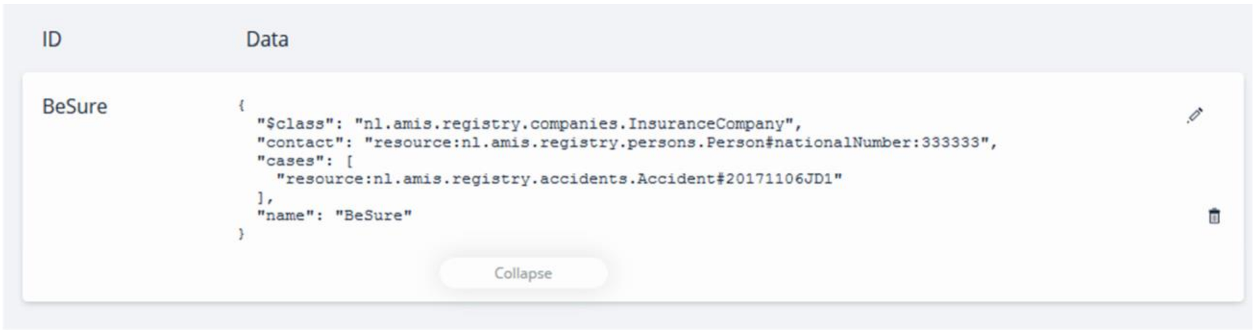
You can now check the *Accident* registry to see that the status has changed to SEND_TO_INSURANCE and the assigned *insurer* is BeSure.


ID Data

20171106JD1

```
{
  "$class": "nl.amis.registry.accidents.Accident",
  "accidentId": "20171106JD1",
  "description": "Driver got a flat tire on the highway and hit the crash barrier. Driver seems OK but vehicle
  "status": "SEND_TO_INSURER",
  "location": {
    "$class": "nl.amis.registry.accidents.Location",
    "longitude": 4,
    "latitude": 51,
    "description": "E19 near Schoten."
  },
  "goods": {
    "$class": "nl.amis.registry.accidents.Goods",
    "vehicles": [
      "resource:nl.amis.registry.vehicles.Vehicle#1FUYDZYB4XPA21495"
    ],
    "insurers": [
      "resource:nl.amis.registry.companies.InsuranceCompany#BeSure"
    ]
  },
  "assignee": "resource:nl.amis.registry.persons.CaseWorker#333333"
}
```

Additionally, in the InsuranceCompany registry you can verify that the BeSure company has had one case added to its cases: Accident#20171106JD1.



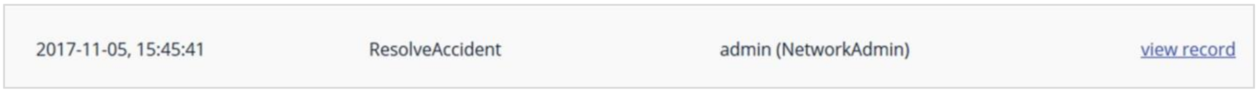
```
{
  "$class": "nl.amis.registry.companies.InsuranceCompany",
  "contact": "resource:nl.amis.registry.persons.Person#nationalNumber:333333",
  "cases": [
    "resource:nl.amis.registry.accidents.Accident#20171106JD1"
  ],
  "name": "BeSure"
}
```

Step 5: Resolve the case

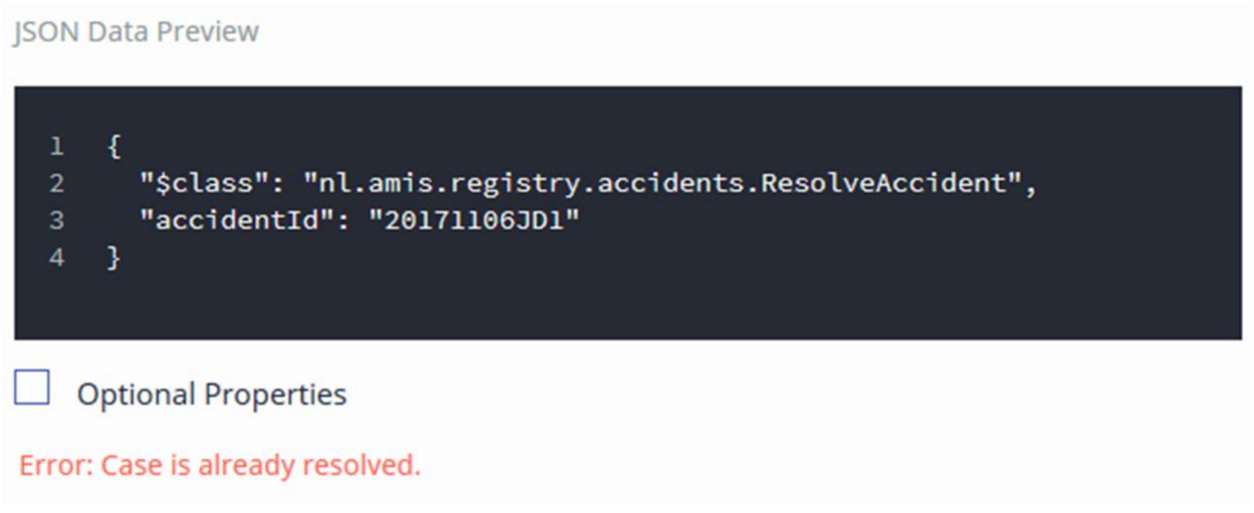
The final action to be done is to register the case as Resolved: click the Submit Transaction button and select ResolveAccident.

4. Send the case to the insurance company by submitting the following message:

```
{
  "$class": "nl.amis.registry.accidents.ResolveAccident",
  "accidentId": "20171106JD1"
}
```



2017-11-05, 15:45:41	ResolveAccident	admin (NetworkAdmin)	view record



```
{
  "$class": "nl.amis.registry.accidents.ResolveAccident",
  "accidentId": "20171106JD1"
}
```

Error: Case is already resolved.

➔ In the bottom left corner, click on Export to save your business network to local disk. This need to be transported to the VM.

You have now successfully resolved this case and part one of the hands-on lab. Congratulations!