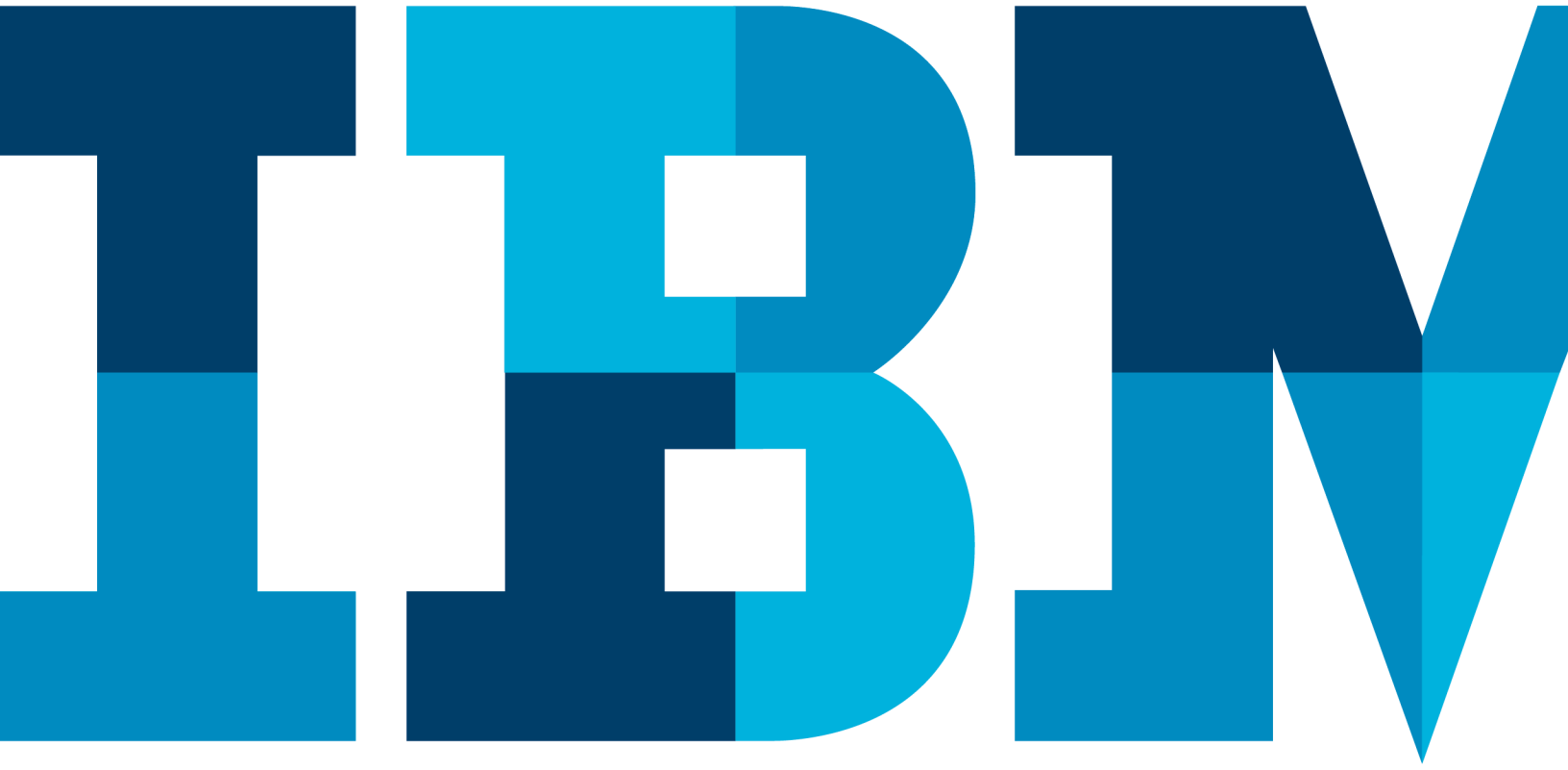
IBM Blockchain

# IBM Blockchain Hands-On
# Hyperledger Composer Playground

*Lab – Exercises*



IBM

# Contents



## Introduction to this section of the lab

Skill requirements:

- There are no skill prerequisites to completing the first section called ‘Car Auction Sample’. It is desirable but not essential to have some background knowledge of JavaScript for the later section called ‘Explore the Editor Views’.

Technical pre-requisites:

- Internet Connection
- Web browser: Chrome or Firefox recommended

This section of the lab takes place entirely in the web browser using Hyperledger *Composer Playground*.

Playground simulates the entire blockchain network within the browser by providing a sandpit environment to define, test and explore business networks defined using Composer. It is possible to connect to a live blockchain Hyperledger Fabric instance, or install the Composer Playground on a local machine for more developer friendly tools.

Hyperledger Composer Playground is one method to use Hyperledger Composer, other methods are also available at https://hyperledger.github.io/composer/installing/installing-index.html.

# Section 1. Using Hyperledger Composer

Hyperledger Composer (https://hyperledger.github.io/composer) is an open-source set of tools designed to make building blockchain applications easier.

It allows users to model the business networks, assets and transactions that are required for blockchain applications, and to implement those transactions using simple JavaScript functions. The blockchain applications run on instances of Linux Foundation Hyperledger Fabric (www.hyperledger.org).

The purpose of this lab is to introduce you to the concepts of a blockchain by showing you how a blockchain transfers assets between participants in a business network. We will use the implementation of a simple blind car auction as the scenario for the demo.

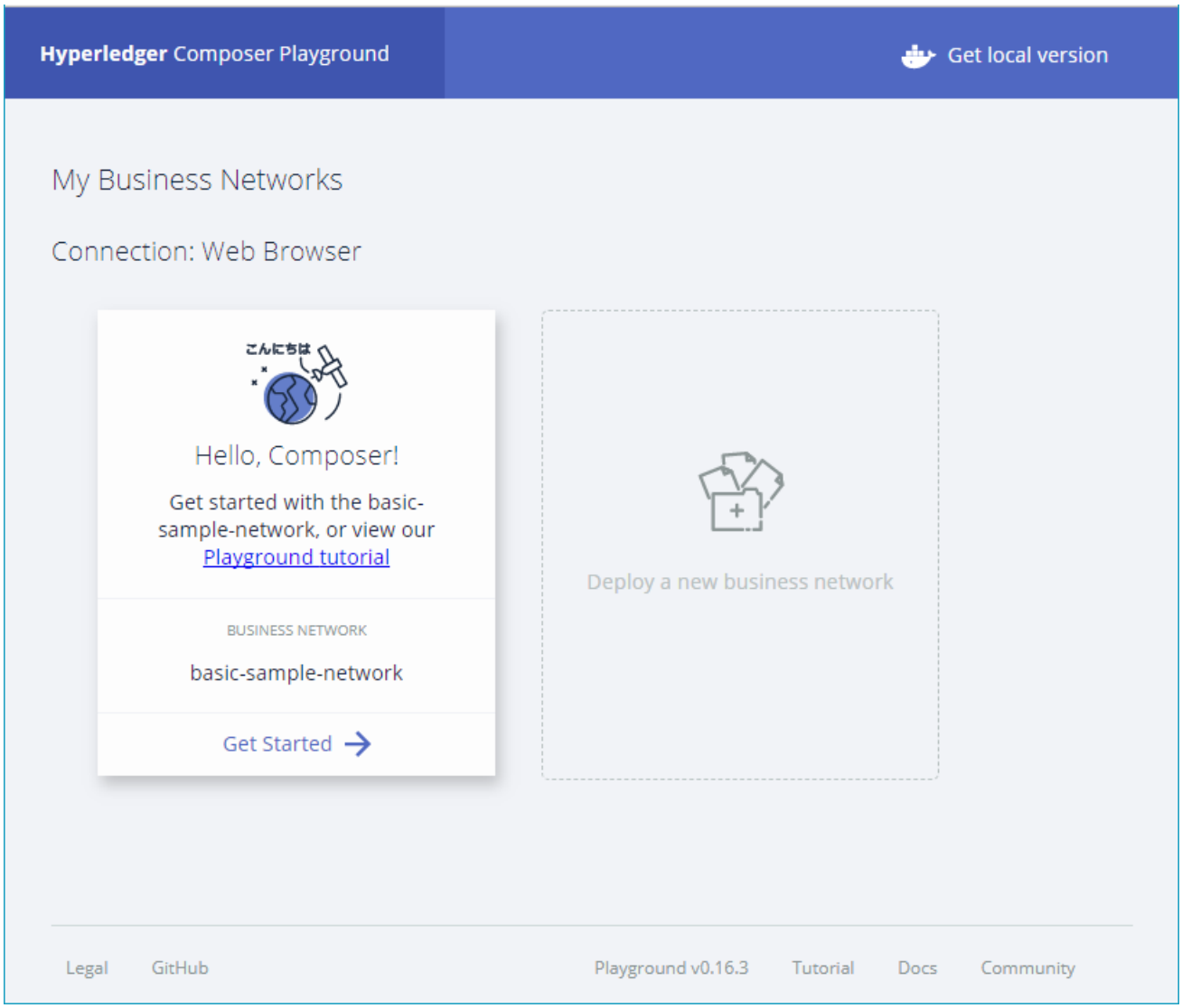
The car auction business network has a set of known participants (buyers and sellers), assets (cars and car listings) and transactions (placing bids and closing auctions). We will model these using Hyperledger Composer and test the business logic that makes the auction work.

Crucially, a blockchain could be used to bring together the buyers and sellers of these assets without needing any trusted third party. However, an auctioneer could be used to provide visibility and governance of the network if required.

## Car Auction Sample

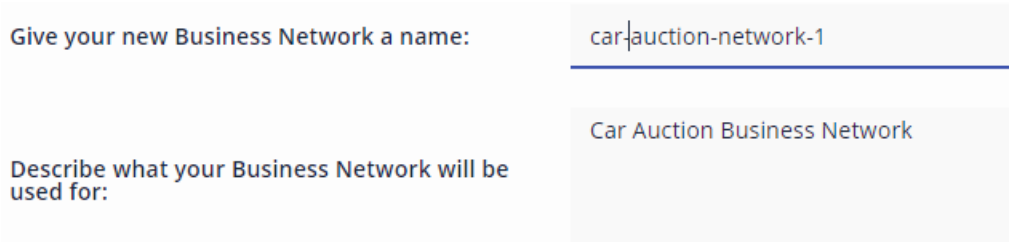
### 1.1.1. Open the Playground

__1. Open a web browser and go to http://composer-playground.mybluemix.net. Dismiss the welcome screen to show the playground wallet screen which is used to connect and deploy new business networks:

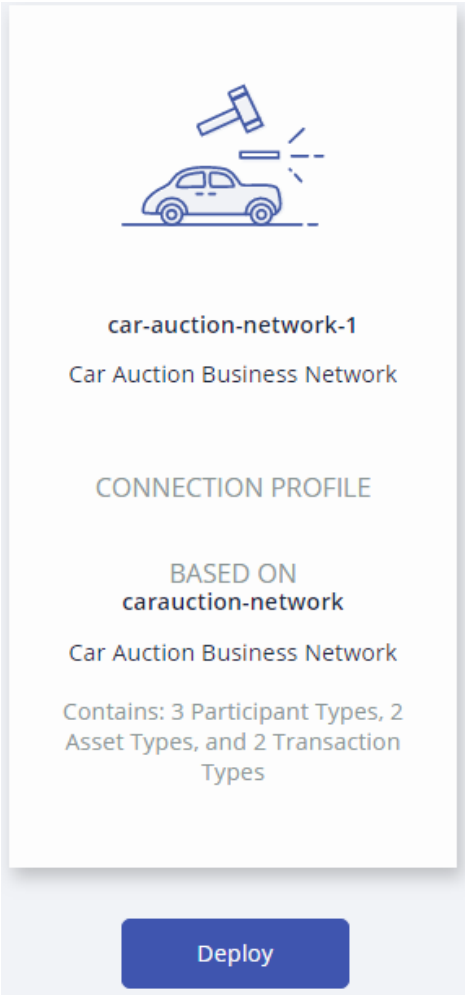



__2. Click the “Deploy a business network” box. Then scroll down and select the carauction-network:

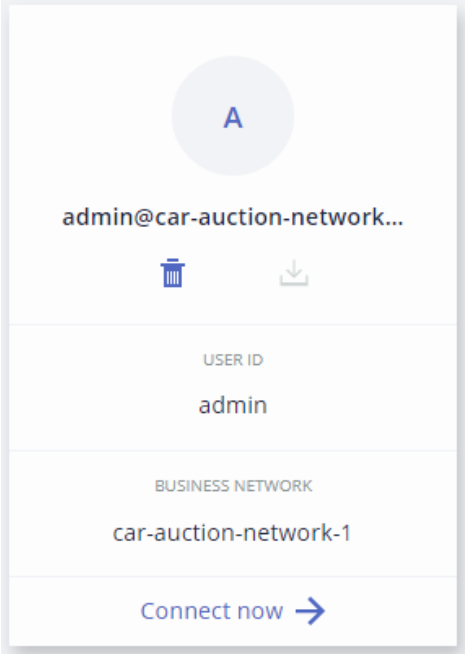

__3. Next give the business network a name and description:




__4. Click the Deploy button to deploy the new car auction business network:

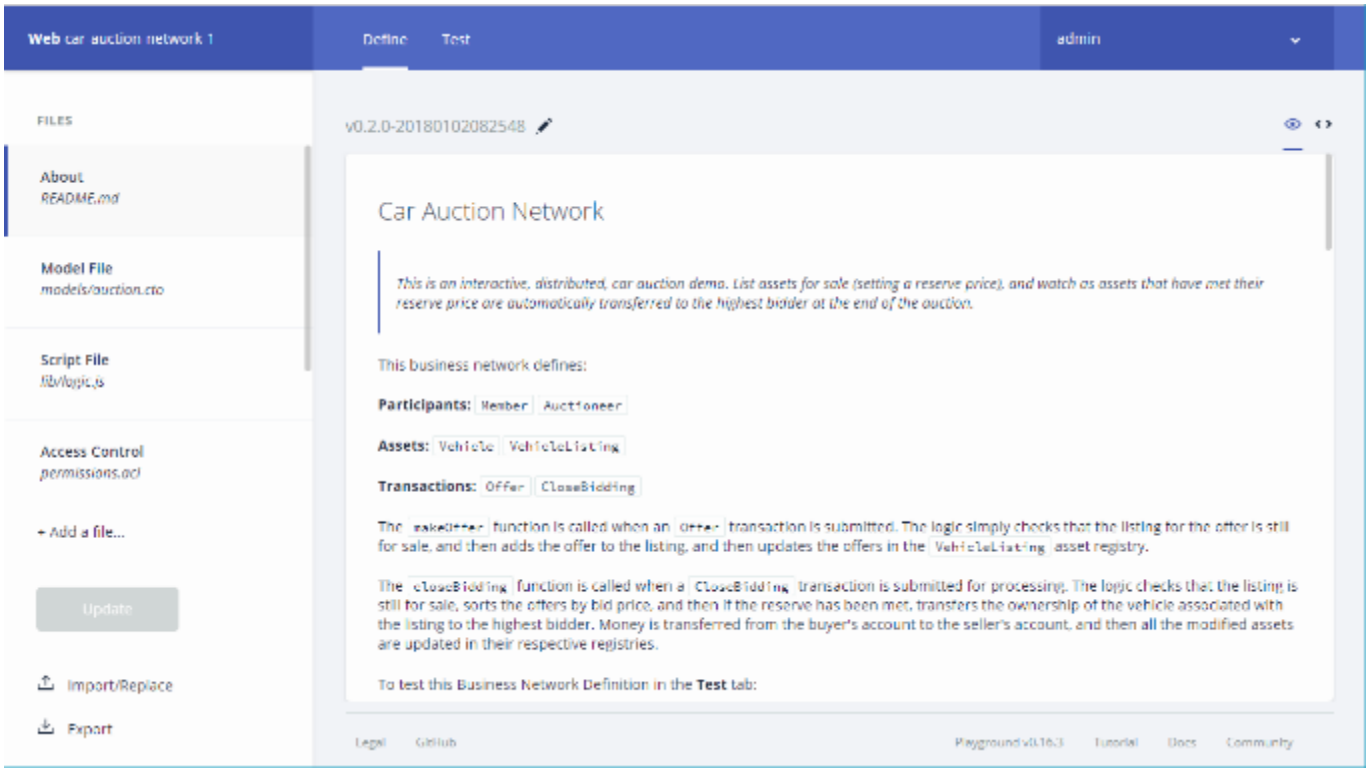



__5. Click “Connect now” in the new identity card for the car-auction-network:



__6. Take a few minutes to read through the description of the car auction sample, to help understand the participants, assets and transactions associated with this particular network.

__7.




### 1.1.2. Add Three Participants

In the next section we will now work with the deployed car auction blockchain network.

We will first instantiate three *Member* participants of the car auction business network:

- Alice Smith (alice@email.com), who will make a bid on a car,
- Bob Jones (bob@email.com), who will also make a bid on a car, and
- Charlie Brown (charlie@email.com), who currently owns a car.

We will not instantiate an Auctioneer in this demo; this could be used in order to provide oversight of the network, although is not necessary.

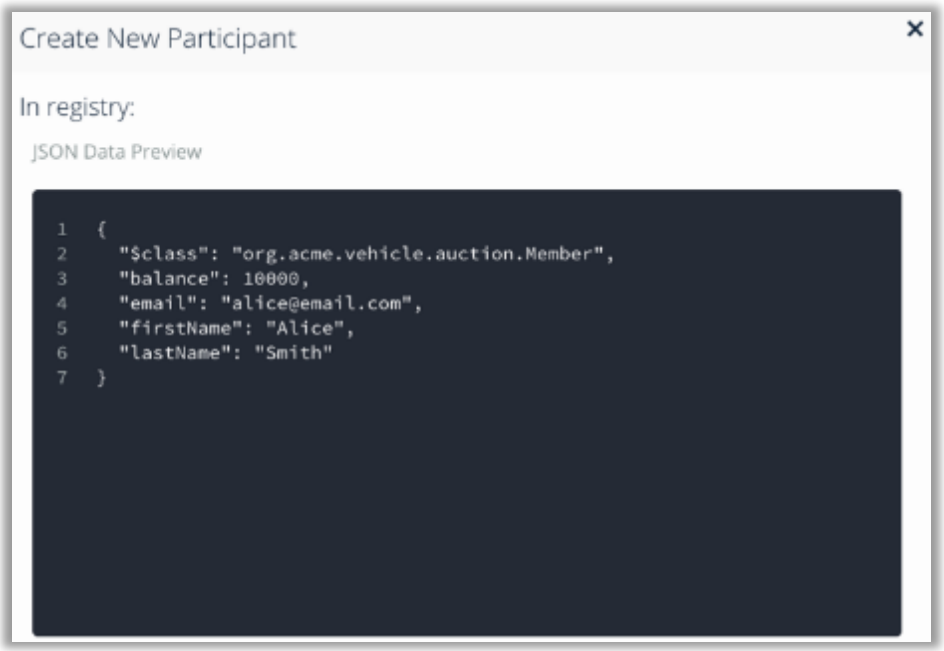
__8. Click the **Test** tab and then click on the *Member* participant registry:


The registry is empty as no members have currently been defined.

__9. Click on **Member** to view there are no current members in the environment
*Note: make sure you choose "**Member**" and not **Auctioneer**"*

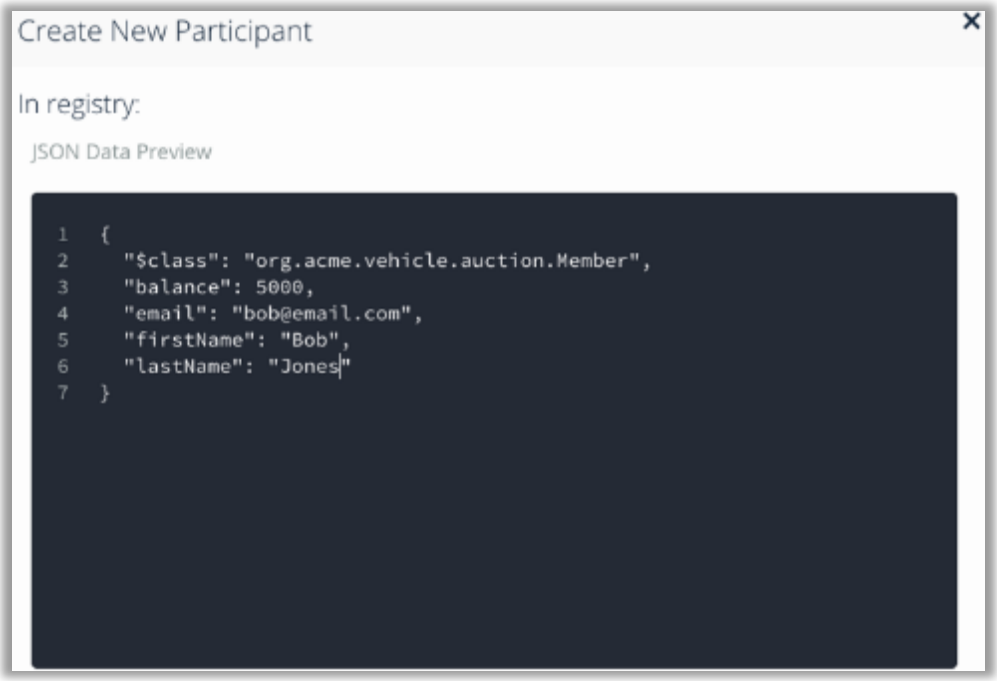

__10. Click **Create New Participant** to add a new Member.


__11. Type the correct values into the JSON data structure to add Alice to the business network. Let's give her a starting balance of 10000.

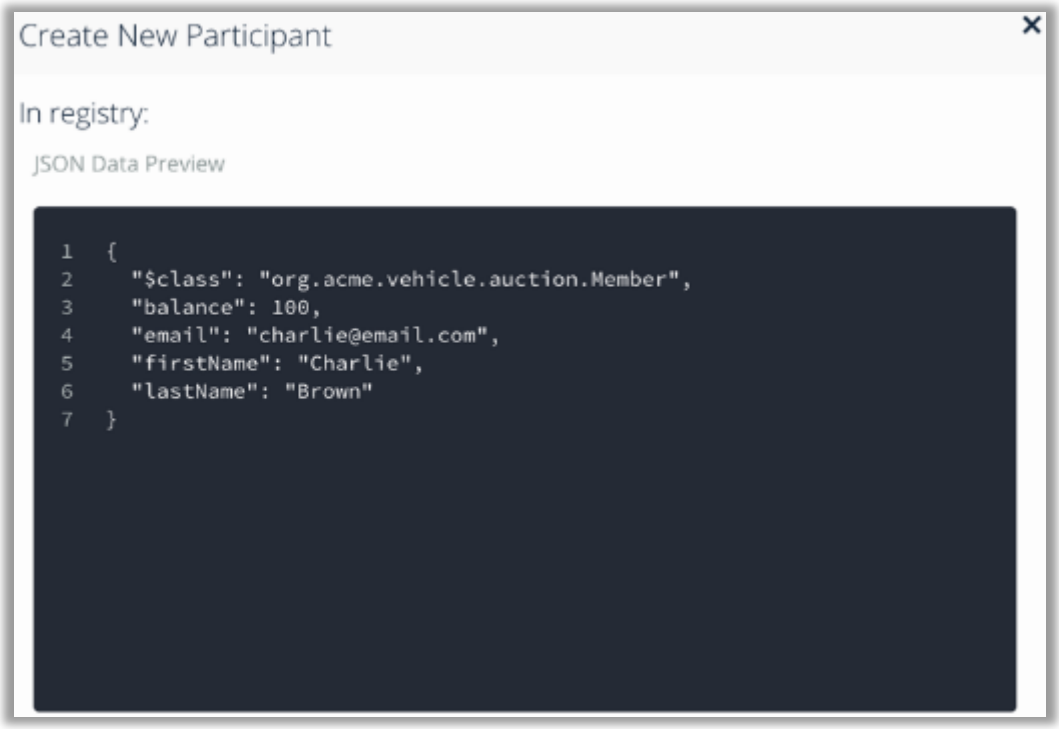



__12. Click **Create New** to add Alice to the registry.

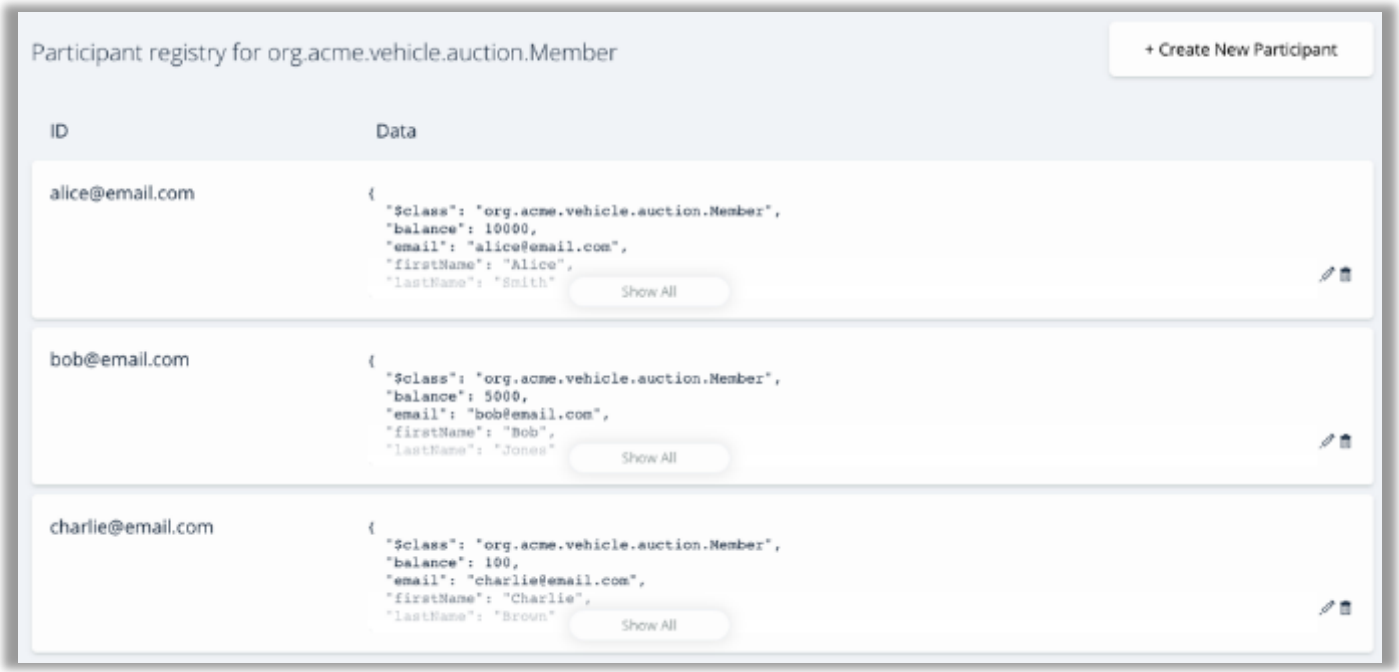
__13. Do the same for Bob. Let's give him a starting balance of 5000.



__14. Finally do the same for Charlie. He hasn't got so much money (he's selling his car, after all) so let's give him a starting balance of 100.




__15. Verify that all participants in the business network have been correctly defined. Use the appropriate Edit button ( ) to make any changes.



### 1.1.3. Add an Asset

We will now add Charlie's car to the Vehicle Asset registry.

__16. Click the ***Vehicle*** asset registry.


__17. This registry contains no assets currently. Click the **Create New Asset** button to add a new asset.

__18. Instantiate the car by adding a vehicle identification number (VIN) of 1234 and assign it to Charlie by filling in the JSON object as follows. (We use his email address to identify him; this was specified as the key field in the User definition using the 'identified by' statement.)

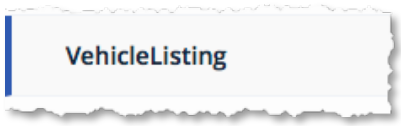

__19. Click **Create New** to add the new vehicle to the registry.

__20. View your newly added asset in the registry.

### 1.1.4. Add a Vehicle Listing

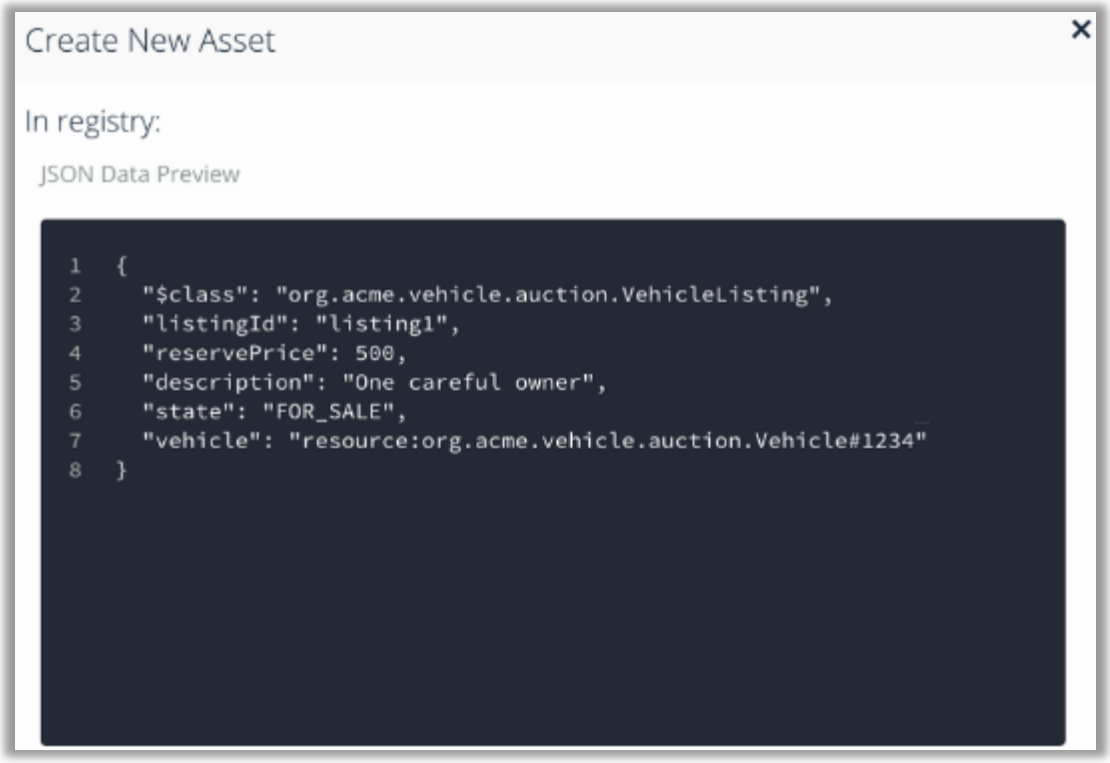
In this section we will put the car up for sale by creating a *VehicleListing* instance.

__21. Click the ***VehicleListing*** asset registry. Once more, the VehicleListing registry should be empty.



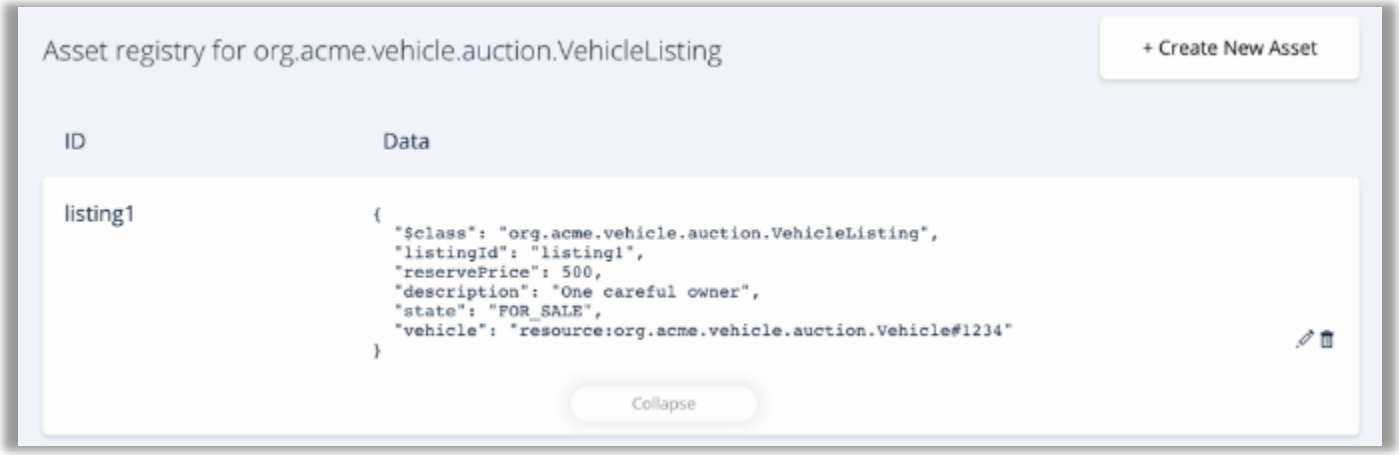

__22. Click the **Create New Asset** button to add the asset.

__23. Update the fields and remove the random offers to show the below. Syntactic validation of the object occurs at this point, so correct any errors if necessary.




__24. Click **Create New** to add the new vehicle listing to the registry.

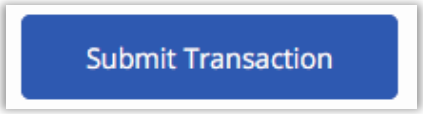
__25. View the listing in the registry.



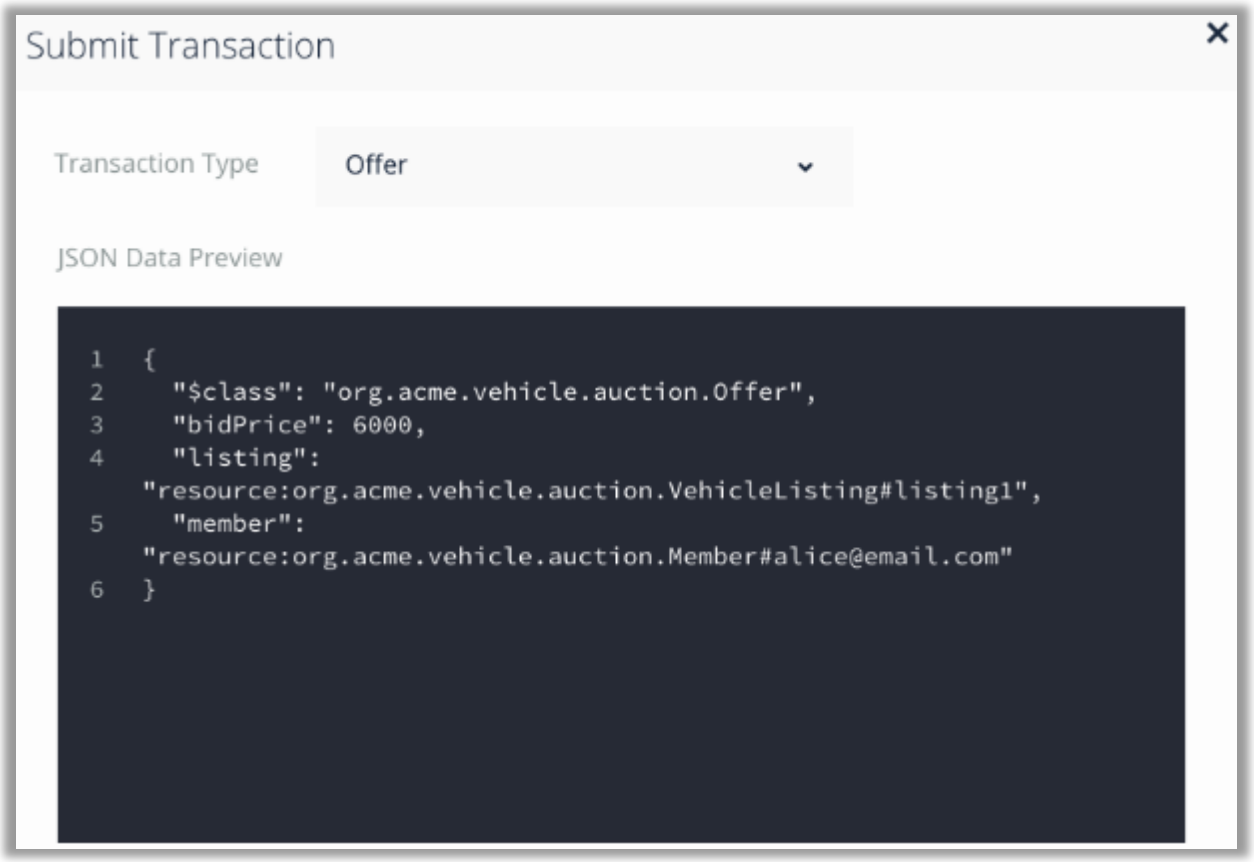
### 1.1.5. Submit offers on the vehicle

We will now let Alice and Bob bid on the vehicle.

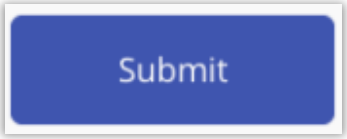
__26. Click on the Submit Transaction button




__27. Let Alice put in a bid of 6000.

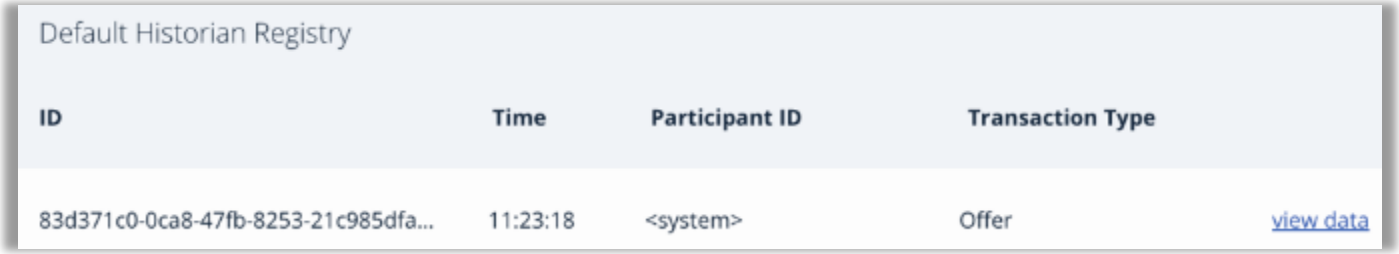



__28. Click **Submit** to submit the offer transaction.

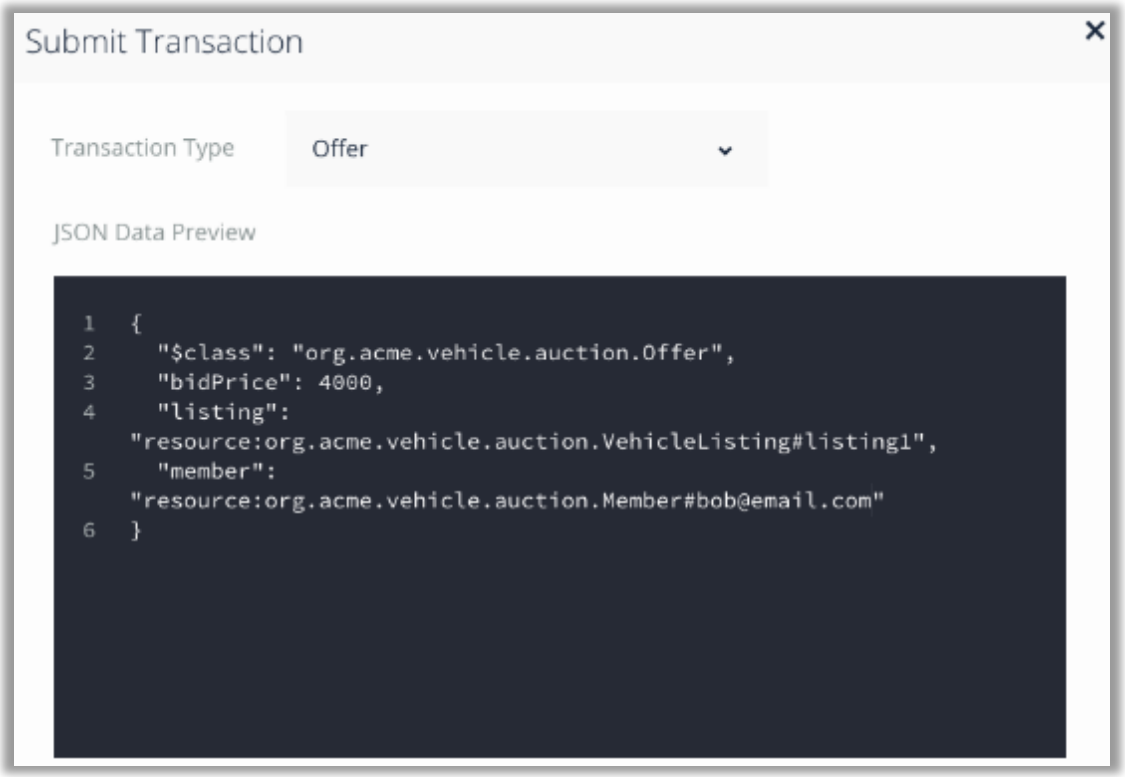


__29. See the transaction successful appear in the Historian registry. Swich to view all transactions by clicking ‘All Transactions’:


__30. You will also notice additional transactions for creating participants and assets. Click “view data” for more information.




__31. Let Bob put in a bid of 4000.




__32. Verify the transactions in the registry.

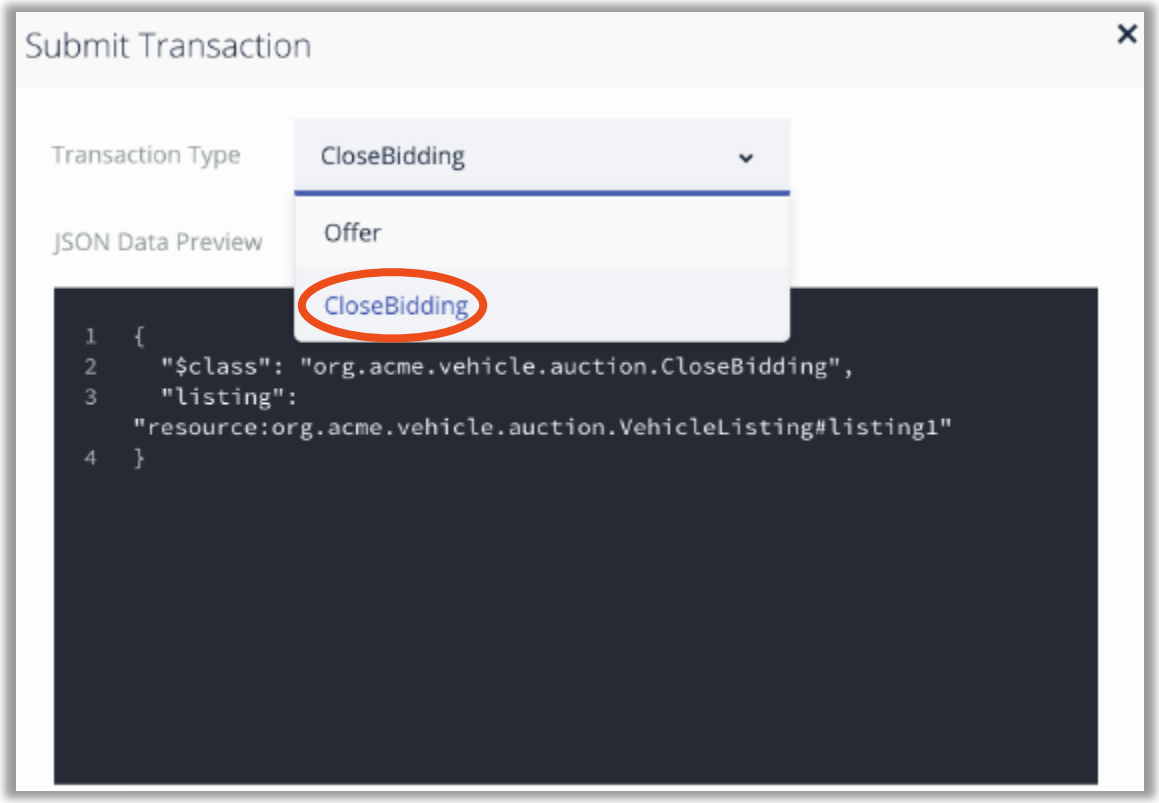
| ID | Time | Participant ID | Transaction Type | |
|---|---|---|---|---|
| 35d593ff-b222-4fd5-9a68-d41f61a891... | 13:13:42 | <system> | Offer | view data |
| 83d371c0-0ca8-47fb-8253-21c985dfa... | 11:23:18 | <system> | Offer | view data |

Note that the transactions cannot be edited or individually deleted once submitted; this is one of the defining characteristics of a blockchain.

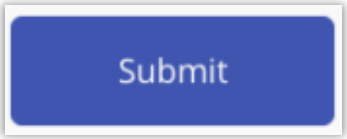
### 1.1.6. Closing the bidding

To close the bidding on the listing we need to submit a *CloseBidding* transaction.

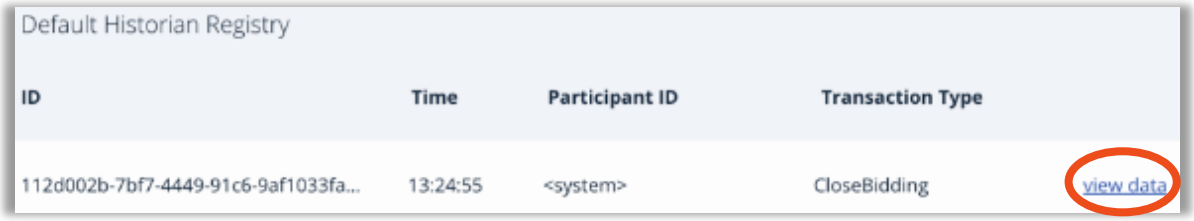
__33. Submit a new transaction, this time selecting **CloseBidding** from the drop-down 'Transaction Type' field.

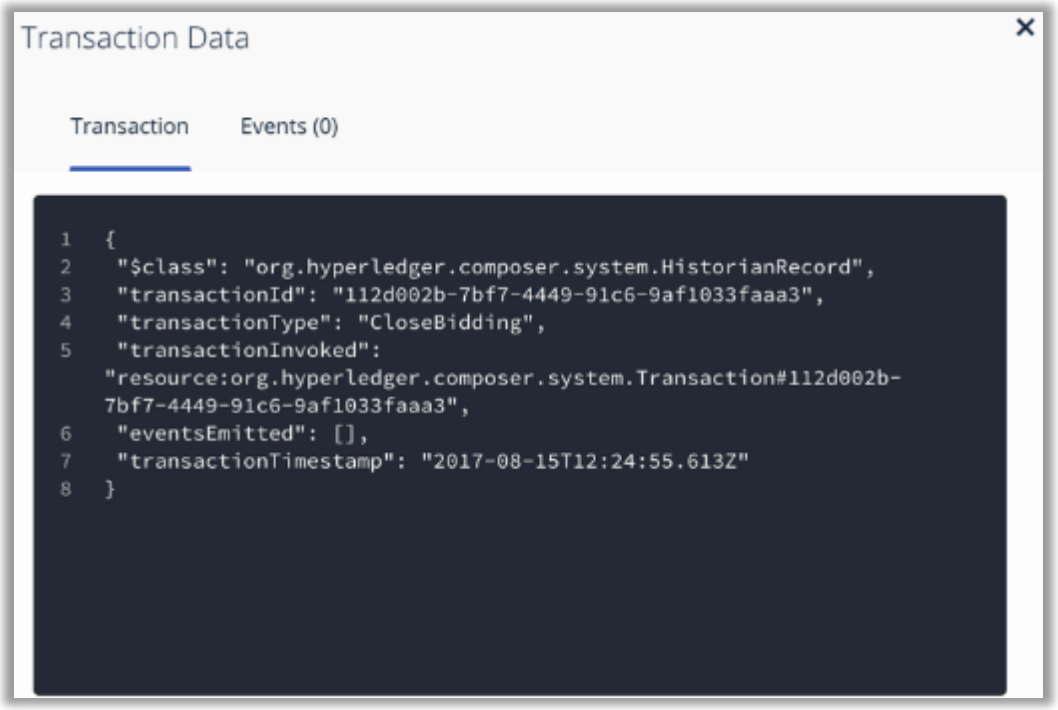



__34. Click **Submit** to submit the CloseBidding transaction.




__35. Verify that the transaction has been added to the blockchain transaction registry. Click on 'view data' to see the content of the transaction.



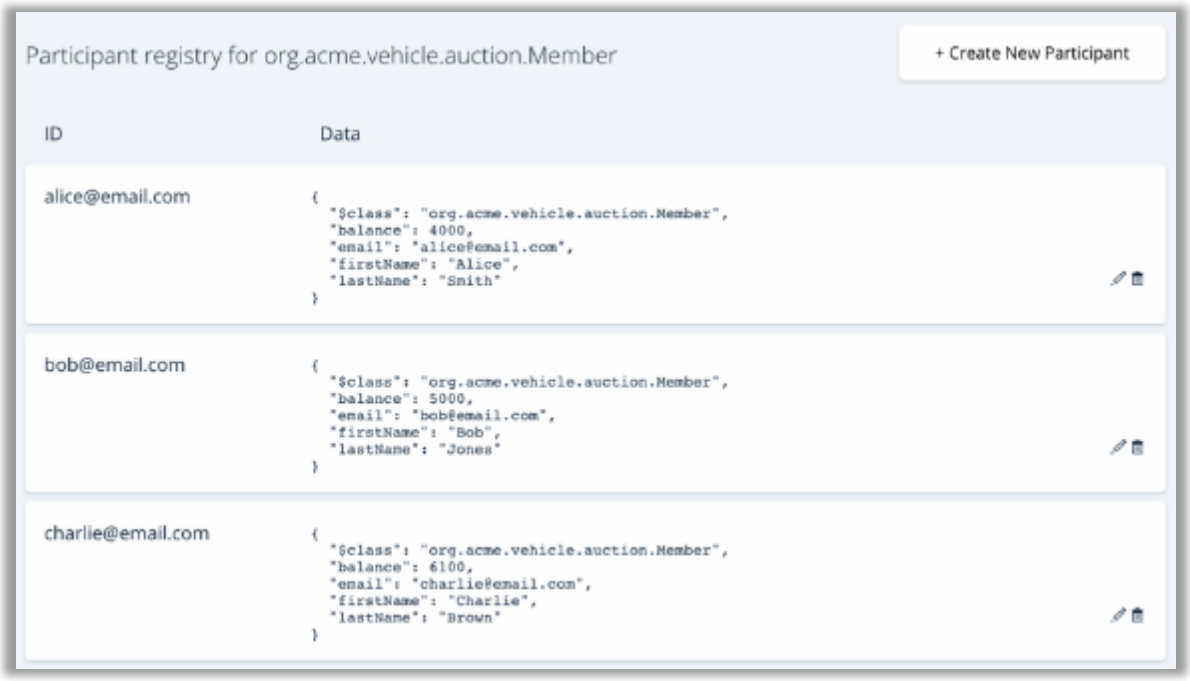



Based on the bids we submitted, Alice should now be the owner as she put in the highest bid. We should also be able to verify that the owner of the car has changed and appropriate balances increased or decreased accordingly.

__36. Go to the ***Vehicle*** asset registry to see the vehicle owner has been updated to Alice.


__37. You will see the following vehicle owned by Alice in the vehicle registry.


__38. Go to the ***Member*** asset registry to see that Charlie's balance has increased by the winning bid amount, and that Alice's balance has decreased by the same.

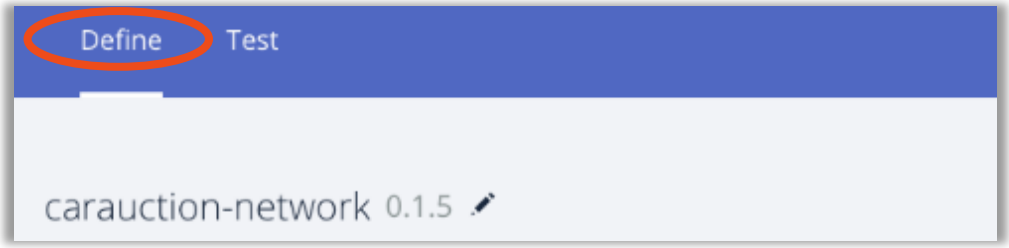



Congratulations! You have completed the first part of this lab.

## Explore the Editor Views

### 1.1.7. Model File

__39. Click on the define tab to go back to the main playground window.




__40. Click the Model File (models/auction.cto) to open it.


This .cto file models the assets, participants and transactions for this blockchain application.

__41. Look at the Vehicle asset:

```
asset Vehicle identified by vin {
  o String vin
  --> Member owner
}
```

This uses the Hyperledger Composer Modeling Language which will be looked at more later. An *asset* is anything of worth that will be transferred around the blockchain. Here we can see the asset class is called '*Vehicle*' and will have an associated *vin* and a reference (indicated by "-->") to a '*Member*' participant that we will call '*owner*'.

__42. Type and add some characters in an appropriate point to show the live validation of the model.

```
asset VehicleListing identified by listingId {
  o String listingId
  o Double reservePrice
  o String description
  o ListingState state
  o Offer[] offers optional
  --> Vehicle vehicle
}
```

**Error found!**

Error: Syntax error in file undefined. Expected "extends", "identified by", "{", comment, end of line or whitespace but "l" found. Line 17 column 22

__43. Scroll down and look at the abstract '*User*' participant.

The participant will be the people or companies within the business network. Each *User* participant will be defined as having a *email*, *firstName* and *lastName*. As the class is **abstract** instances of it cannot be created; instances are instead implemented by the *Member* and *Auctioneer* classes.

```
abstract participant User identified by email {
  o String email
  o String firstName
  o String lastName
}

participant Member extends User {
  o Double balance
}

participant Auctioneer extends User {
}
```

Here the user can become a *Member* requiring a *balance*, or an *Auctioneer* that does not.

__44. Look at the *Offer* and *CloseBidding* transaction definitions:

```
transaction Offer {
  o Double bidPrice
  --> VehicleListing listing
  --> Member member
}

transaction CloseBidding {
  --> VehicleListing listing
}
```

The *transaction* definitions give a description of the transactions that can be performed on the blockchain. They are implemented in a Transaction Processor file using the Javascript language.

### 1.1.8. Transaction Processors

__45. Click on the lib/logic.js file:

__46. Scroll to **the bottom of the file** to review the logic used to make an offer on a car being auctioned:

```
/**
 * Make an Offer for a VehicleListing
 * @param {org.acme.vehicle.auction.Offer} offer - the offer
 * @transaction
 */
function makeOffer(offer) {
    var listing = offer.listing;
    if (listing.state !== 'FOR_SALE') {
        throw new Error('Listing is not FOR SALE');
    }
    if (listing.offers == null) {
        listing.offers = [];
    }
    listing.offers.push(offer);
    return getAssetRegistry('org.acme.vehicle.auction.VehicleListing')
        .then(function(vehicleListingRegistry) {
            // save the vehicle listing
            return vehicleListingRegistry.update(listing);
        });
}
```

This implements the *makeOffer* function, which is executed when the *Offer* transaction is invoked on the blockchain. (It is the **@param** comment above the function that links the full transaction name as defined by the model to the Javascript method that implements it.)

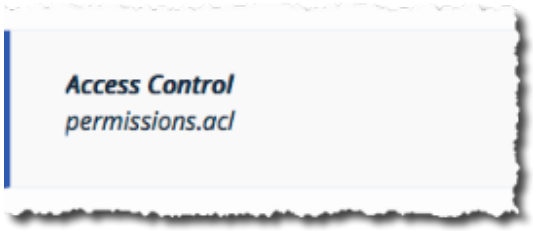
Other Interesting areas of the function implementation include:

a) The logic that the vehicle must be for sale to submit an offer on it
b) The retrieval and update of the asset registry a few lines later
c) Saving the updated asset back to the registry

### 1.1.9. Access Control List

The final file that defines the blockchain application is the Access Control List, which describes the rules which govern which participants in the business network can work with which parts of the blockchain.

__47. Click the permissions.acl file:




__48. Look at the ACL rules defined:

```
/**
 * Access Control List for the auction network.
 */
rule Auctioneer {
    description: "Allow the auctioneer full access"
    participant: "org.acme.vehicle.auction.Auctioneer"
    operation: ALL
    resource: "org.acme.vehicle.auction.*"
    action: ALLOW
}

rule Member {
    description: "Allow the member read access"
    participant: "org.acme.vehicle.auction.Member"
    operation: READ
    resource: "org.acme.vehicle.auction.*"
    action: ALLOW
}

rule VehicleOwner {
    description: "Allow the owner of a vehicle total access"
    participant(m): "org.acme.vehicle.auction.Member"
```

The rule allows or denies users to access aspects of the blockchain.

## Updating the Model (Advanced and Optional)

__49. Try updating the model (*auction.cto*) for the *Vehicle* asset definition to include manufacturer make and model fields. Add in new *String* fields and click 'Update' to make the changes live.

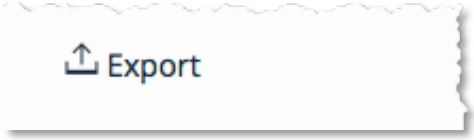
Note that when you update the model, the syntax of any existing assets in the registry must be compatible with the new model. Use either the **optional** or **default="…"** qualifiers next to the new fields. If you make incompatible changes, you must first reset the demo.

Once you've made the changes, try adding new *Vehicle* assets to the registry to test the changes.

For more information on the Hyperledger Composer modelling language please refer to: https://hyperledger.github.io/composer/reference/cto_language.html

## Export the Business Network Archive

__50. Exporting to a Business Network Archive will save the Read Me, Model File(s), Script File(s) and Access Control rules that can be easily imported to a local developer enviroment, handed to a network operator to deploy to a live network or saved asa backup. More details on local installation at https://hyperledger.github.io/composer/installing/installing-index.html.




Congratulations! You have completed this lab.

__51. Appendix A. 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 matter 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 B. 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







Please Recycle

IBM Software