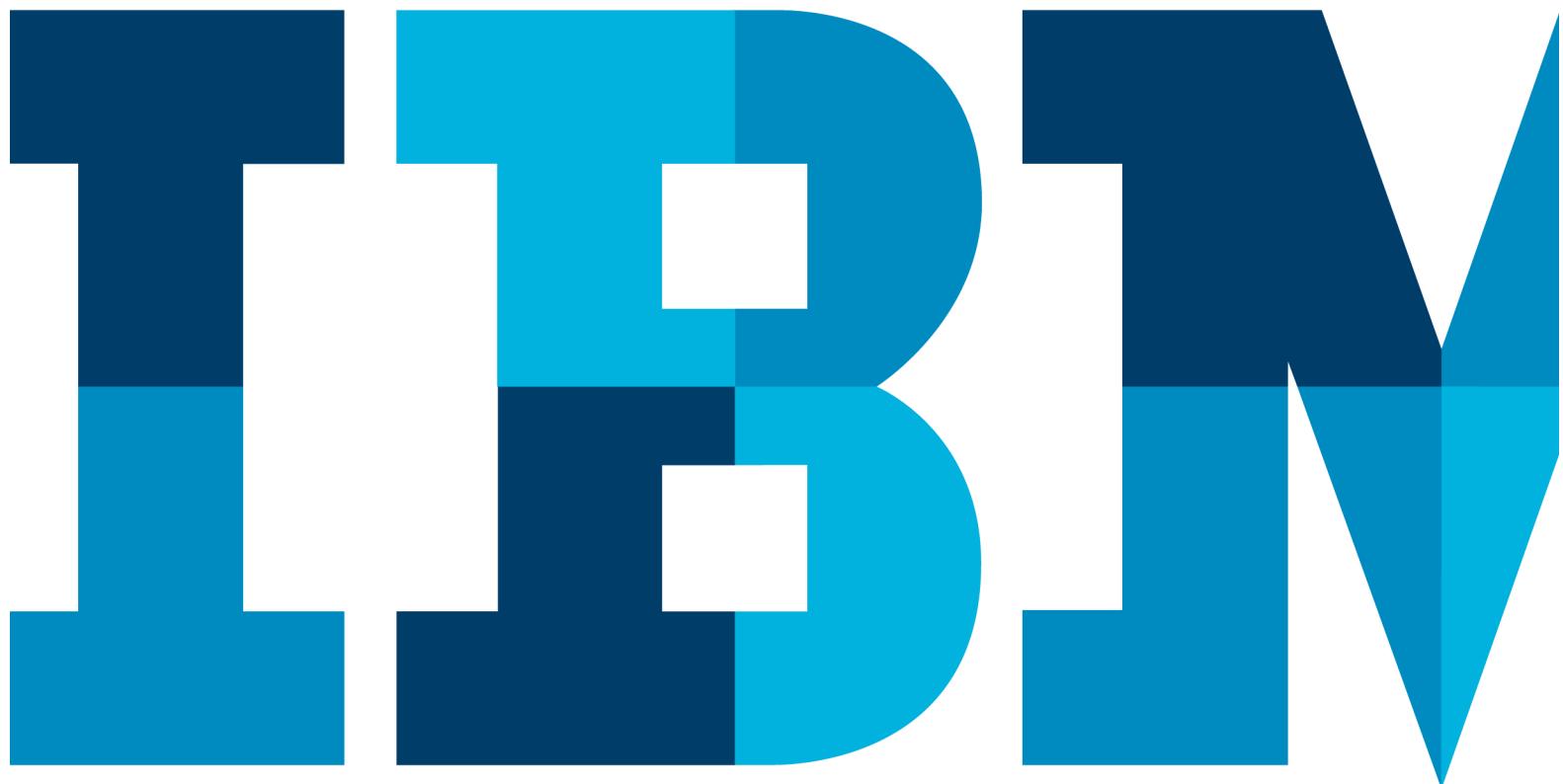


IBM Blockchain Hands-On Hyperledger Composer Playground

Lab – Exercises



Contents

SECTION 1. USING HYPERLEDGER COMPOSER.....	3
1.1. OPEN THE PLAYGROUND	4
1.2. IMPORT THE CAR AUCTION SAMPLE	5
1.3. EXPLORE THE EDITOR VIEWS	6
1.3.1. MODEL FILE	6
1.3.2. TRANSACTION PROCESSORS	8
1.3.3. ACCESS CONTROL LIST.....	10
1.4. ADD THREE PARTICIPANTS	11
1.5. ADD AN ASSET.....	14
1.6. ADD A VEHICLE LISTING.....	15
1.7. SUBMIT OFFERS ON THE VEHICLE	16
1.8. CLOSING THE BIDDING	18
1.9. EXPORT THE BUSINESS NETWORK ARCHIVE	19
APPENDIX B. TRADEMARKS AND COPYRIGHTS	23

Introduction to this section of the lab

Skill requirements:

- There are no skill prerequisites to completing this section although some background knowledge of object-oriented concepts and/or JavaScript is desirable.

Technical pre-requisites:

- Internet Connection
- Any Modern Browser (IE9+)

This section of the lab takes place entirely in the web browser using *Composer Playground*. A GitHub logic is required to import the sample into the 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 www.fabric-composer.org.

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.



Open the Playground

1. Open a web browser and go to <http://composer-playground.mybluemix.net>

The screenshot shows the Hyperledger Composer Playground interface. At the top, there's a navigation bar with tabs for 'Define' (which is selected) and 'Test'. To the right of the tabs are links for 'Get local version' and a user profile icon. Below the navigation bar is a sidebar titled 'FILES' containing the following files:

- About** (selected)
- README.md*
- Model File** (*lib/org.acme.sample.cto*)
- Script File** (*lib/logic.js*)
- Access Control** (*permissions.acl*)
- + Add a file...

Below the sidebar is a large central panel. At the top of the panel, it says 'basic-sample-network 0.0.7' with a edit icon. The main content area has a heading 'Welcome to Hyperledger Composer!'. It includes several paragraphs of descriptive text and code snippets:

- This is the "Hello World" of Hyperledger Composer samples.
- This sample defines a business network composed of a single asset type (`SampleAsset`), a single participant type (`SampleParticipant`), and a single transaction type (`SampleTransaction`).
- `SampleAssets` are owned by a `SampleParticipant`, and the value property on a `SampleAsset` can be modified by submitting a `SampleTransaction`.
- To get started inside Hyperledger Composer you can click the Test tab and create instances of `SampleAsset` and `SampleParticipant`. Make sure that the owner property on the `SampleAsset` refers to a `SampleParticipant` that you have created.
- You can then submit a `SampleTransaction`, making sure that the asset property refers to an asset that you created earlier. After the transaction has been processed you should see that the value property on the asset has been modified.
- The logic for updating the asset when a `SampleTransaction` is processed is written in `logic.js`.
- Don't forget you can import more advanced samples into Hyperledger Composer using the Import/Replace

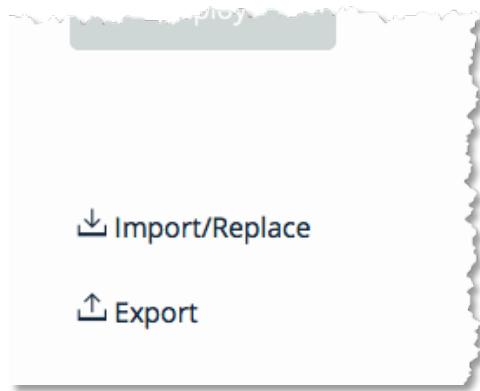
At the bottom left of the central panel, there are two buttons: 'Import/Replace' with a downward arrow icon and 'Export' with an upward arrow icon. There is also a 'Deploy' button in a grey box.

2. By default, files are saved to local browser storage. If you have previously run this lab or edited files within this web page, then in order to run through this lab you might need to delete any web browser cookies from the `mybluemix.net` domain.

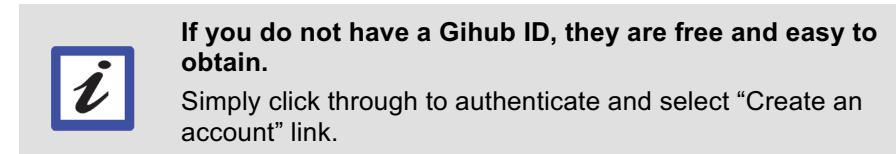
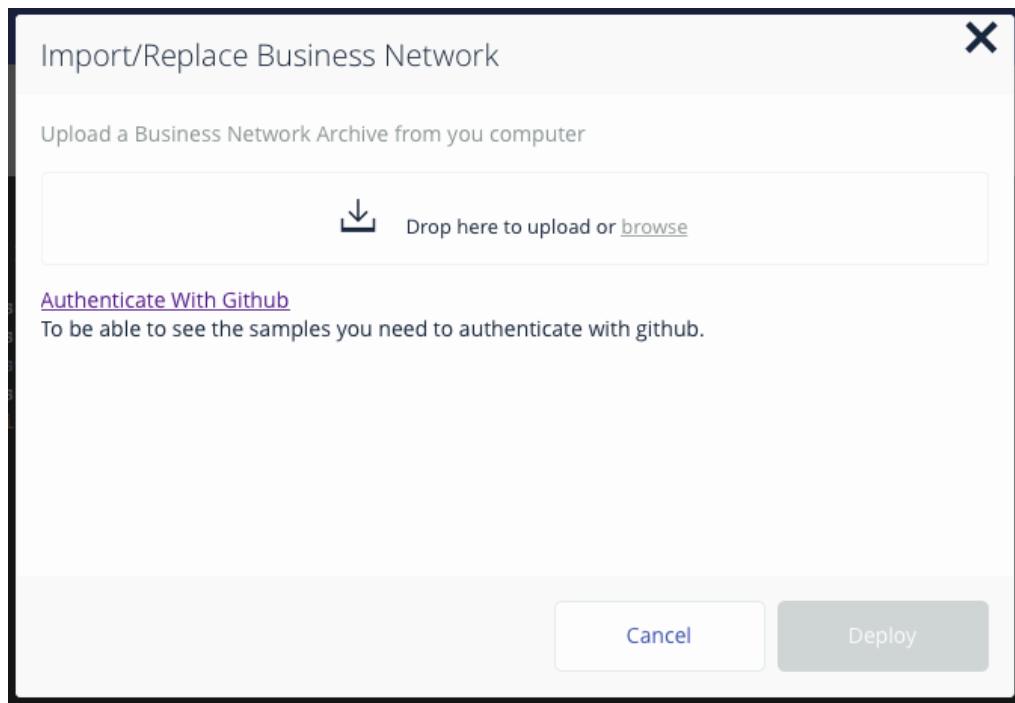
This will ensure that the asset, participant and transaction registries are empty, and that any changes made to the files are discarded.

Import the Car Auction Sample

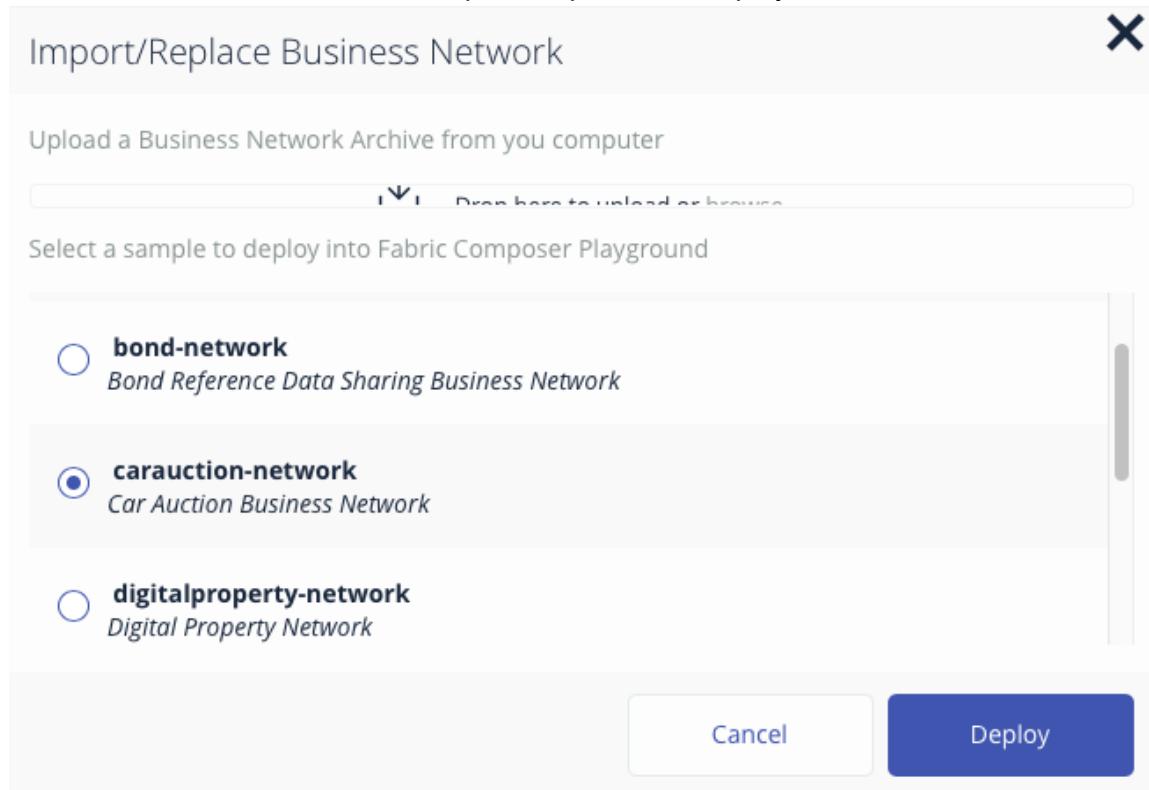
3. Click the Import/Replace button.



Press the “Authenticate with GitHub” link if present:



4. Select the carauction-network sample and press the Deploy button:



Explore the Editor Views

1.3.1. Model File

5. Click the Model File (lib/org.acme.vehicle.auction.cto) to open it, if it is not already open.



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

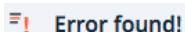
- __6. Look at the Vehicle asset:

```
5
6   asset Vehicle identified by vin {
7     o String vin
8     --> Member owner
9   }
10
```

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*'.

- __7. Type and add some characters in an appropriate point to show the live validation of the model.

```
16
17   asset VehicleListing identifiedxxx by listingId {
18     o String listingId
19     o Double reservePrice
20     o String description
21     o ListingState state
22     o Offer[] offers optional
23     --> Vehicle vehicle
24   }
25
```



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

- __8. 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.



```
26 abstract participant User identified by email {
27   o String email
28   o String firstName
29   o String lastName
30 }
31
32 participant Member extends User {
33   o Double balance
34 }
35
36 participant Auctioneer extends User {
37 }
38
```

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

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

```
39 transaction Offer identified by transactionId {
40   o String transactionId
41   o Double bidPrice
42   --> VehicleListing listing
43   --> Member member
44 }
45
46 transaction CloseBidding identified by transactionId {
47   o String transactionId
48   --> VehicleListing listing
49 }
50
```

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.

1.3.2. Transaction Processors

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



Script File
lib/logic.js

- __11. Scroll to **the bottom of the file** to review the logic used to make an offer on a car being auctioned:

```
84
85  /**
86   * Make an Offer for a VehicleListing
87   * @param {org.acme.vehicle.auction.Offer} offer - the offer
88   * @transaction
89   */
90  function makeOffer(offer) {
91      var listing = offer.listing;
92      if (listing.state !== 'FOR_SALE') {
93          throw new Error('Listing is not FOR SALE');
94      }
95      if (listing.offers == null) {
96          listing.offers = [];
97      }
98      listing.offers.push(offer);
99      return getAssetRegistry('org.acme.vehicle.auction.VehicleListing')
100         .then(function(vehicleListingRegistry) {
101             // save the vehicle listing
102             return vehicleListingRegistry.update(listing);
103         });
104     }
105 }
```

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:

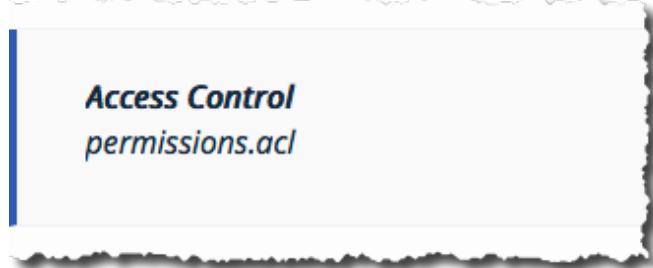
- The logic that the vehicle must be for sale to submit an offer on it
- The retrieval and update of the asset registry a few lines later
- Saving the updated asset back to the registry



1.3.3. 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.

- __12. Click the permissions.acl file:



- __13. Look at the ACL rules defined:

```
1  /**
2   * Access Control List for the auction network.
3   */
4  rule Auctioneer {
5      description: "Allow the auctioneer full access"
6      participant: "org.acme.vehicle.auction.Auctioneer"
7      operation: ALL
8      resource: "org.acme.vehicle.auction"
9      action: ALLOW
10 }
11
12 rule Member {
13     description: "Allow the member read access"
14     participant: "org.acme.vehicle.auction.Member"
15     operation: READ
16     resource: "org.acme.vehicle.auction"
17     action: ALLOW
18 }
19
20 rule VehicleOwner {
21     description: "Allow the owner of a vehicle total access"
22     participant(m): "org.acme.vehicle.auction.Member"
23     operation: ALL
24     resource(v): "org.acme.vehicle.auction.Vehicle"
25     condition: (v.owner.getIdentifier() == m.getIdentifier())
26     action: ALLOW
27 }
28
29 rule VehicleListingOwner {
30     description: "Allow the owner of a vehicle total access to their vehicle listing"
31     participant(m): "org.acme.vehicle.auction.Member"
32     operation: ALL
33     resource(v): "org.acme.vehicle.auction.VehicleListing"
34     condition: (v.vehicle.owner.getIdentifier() == m.getIdentifier())
35     action: ALLOW
36 }
37
```

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

Add Three Participants

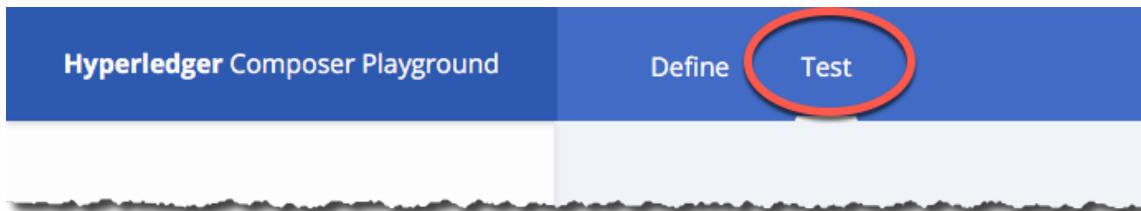
In the next section we will now work with this blockchain application as currently deployed. (It is also possible to make changes to these files and click the ‘Deploy’ button to make them live, although it might be first necessary to Reset the blockchain – see later.)

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

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

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



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

—15. Click on **Member** to view there are no current members in the environment



—16. Click **Create New Participant** to add a new Member.



- __17. 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.

Create New Participant X

In registry: org.acme.vehicle.auction.Member

JSON Data Preview

```
1  {
2    "$class": "org.acme.vehicle.auction.Member",
3    "balance": 10000,
4    "email": "alice@email.com",
5    "firstName": "Alice",
6    "lastName": "Smith"
7 }
```

Just need quick test data? [Generate Random Data](#)

[Cancel](#) [Create New](#)

- __18. Click **Create New** to add Alice to the registry.

- __19. Do the same for Bob. Let's give him a starting balance of 5000.

```
1  {
2      "$class": "org.acme.vehicle.auction.Member",
3      "balance": 5000,
4      "email": "bob@email.com",
5      "firstName": "Bob",
6      "lastName": "Jones"
7 }
```

- __20. 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.

```
1  {
2      "$class": "org.acme.vehicle.auction.Member",
3      "balance": 100,
4      "email": "charlie@email.com",
5      "firstName": "Charlie",
6      "lastName": "Brown"
7 }
```

- __21. Verify that all participants in the business network have been correctly defined. Use the appropriate Edit button () to make any changes.

Participant registry for org.acme.vehicle.auction.Member

+ Create New Participant

ID	DATA
alice@foo.com	{ "\$class": "org.acme.vehicle.auction.Member", "balance": 10000, "email": "alice@foo.com", "firstName": "Alice", "lastName": "Smith" }
bob@foo.com	{ "\$class": "org.acme.vehicle.auction.Member", "balance": 5000, "email": "bob@foo.com", "firstName": "Bob", "lastName": "Jones" }
charlie@foo.com	{ "\$class": "org.acme.vehicle.auction.Member", "balance": 100, "email": "charlie@foo.com", "firstName": "Charlie", "lastName": "Brown" }



Add an Asset

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

- 22. Click the **Vehicle** asset registry.



- 23. This registry contains no assets currently. Click the **Create New Asset** button to add a new asset.
- 24. 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.)

```
1  {
2    "$class": "org.acme.vehicle.auction.Vehicle",
3    "vin": "1234",
4    "owner": "resource:org.acme.vehicle.auction.Member#charlie@email.com"
5 }
```

- 25. View your newly added asset in the registry.

A screenshot of the Asset registry for org.acme.vehicle.auction.Vehicle. The title bar says 'Asset registry for org.acme.vehicle.auction.Vehicle'. There is a '+ Create New Asset' button. A single entry is listed in a table:

ID	Data
1234	{ "\$class": "org.acme.vehicle.auction.Vehicle", "vin": "1234", "owner": "resource:org.acme.vehicle.auction.Member#charlie@email.com" }

The 'Data' column shows the JSON object for the asset with ID 1234. There are edit and delete icons next to the entry.

Add a Vehicle Listing

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

- __26. Click the **VehicleListing** asset registry. Once more, the VehicleListing registry should be empty.



- __27. Click the **Create New Asset** button to add the asset.

- __28. Press the “**Generate Random Data**” button to populate the asset with random data.



- __29. 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.

```
1  {
2    "$class": "org.acme.vehicle.auction.VehicleListing",
3    "listingId": "listing1",
4    "reservePrice": 500,
5    "description": "one careful owner",
6    "state": "FOR_SALE",
7    "offers": [],
8    "vehicle": "resource:org.acme.vehicle.auction.Vehicle#1234"
9 }
```

- __30. View the listing in the registry.

A screenshot of the "Asset registry for org.acme.vehicle.auction.VehicleListing" interface. At the top, there is a header with the registry name and a "+ Create New Asset" button. Below the header, a table lists the asset entries. The first entry is "listing1", which is expanded to show its JSON data. The data is identical to the code shown in the previous step. There are icons for edit and delete next to the entry. A "Collapse" button is located at the bottom of the expanded entry's row.

ID	Data
listing1	{ "\$class": "org.acme.vehicle.auction.VehicleListing", "listingId": "listing1", "reservePrice": 500, "description": "one careful owner", "state": "FOR_SALE", "offers": [], "vehicle": "resource:org.acme.vehicle.auction.Vehicle#1234" }



Submit offers on the vehicle

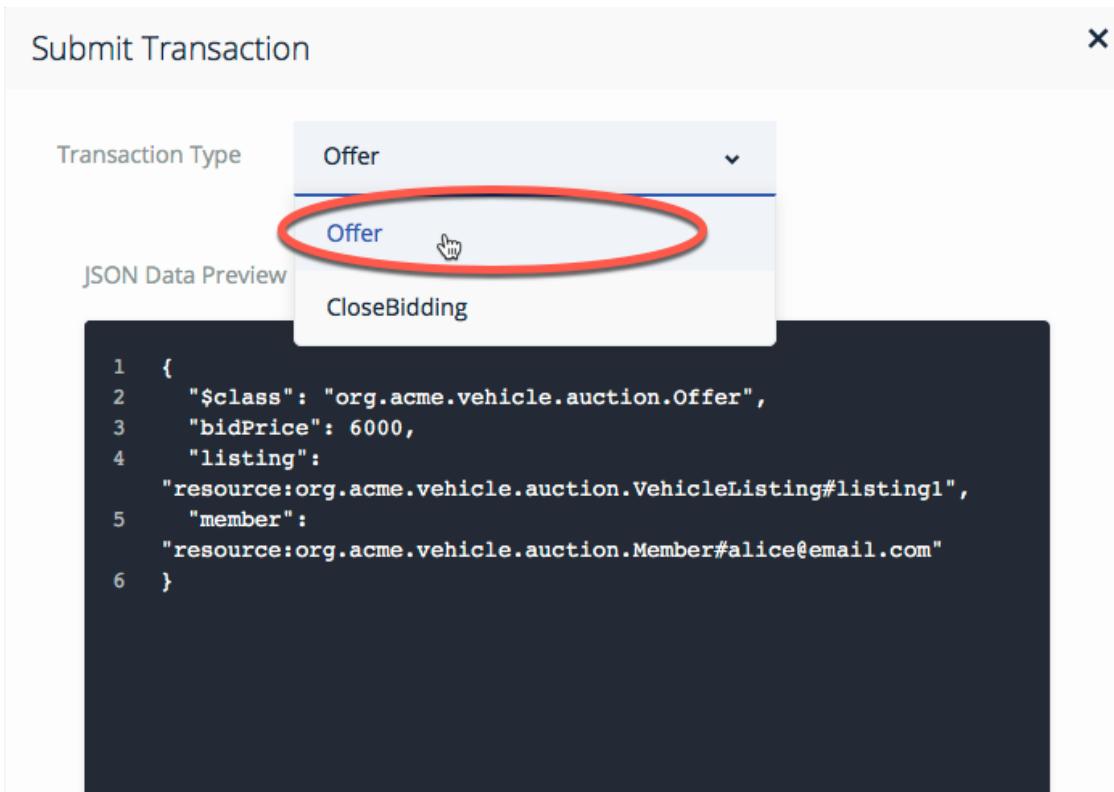
We will now let Alice and Bob bid on the vehicle.

- _31. Click on the Submit Transaction button



Submit Transaction

- _32. Let Alice put in a bid of 6000.



Submit Transaction

Transaction Type: Offer

Offer

CloseBidding

```
1  {
2      "$class": "org.acme.vehicle.auction.Offer",
3      "bidPrice": 6000,
4      "listing":
5          "resource:org.acme.vehicle.auction.VehicleListing#listing1",
6          "member":
7              "resource:org.acme.vehicle.auction.Member#alice@email.com"
8  }
```

Just need quick test data? [Generate Random Data](#)

Cancel

Submit

- _33. See the transaction successful

The screenshot shows a "Default Transaction Registry" interface. On the right, a success message box displays: "Submit Transaction Successful" and "A transaction was successfully submitted". The main table has columns "ID" and "Data". One row in the table is highlighted, showing an offer transaction with ID "34e04a3a-fe15-4bad-bdec-9517f09ca8af". The Data column for this row contains the JSON representation of the offer:

```
{
  "$class": "org.acme.vehicle.auction.Offer",
  "transactionId": "34e04a3a-fe15-4bad-bdec-9517f09ca8af",
  "bidPrice": 6000,
  "listing": "resource:org.acme.vehicle.auction.VehicleListing#listing1",
  "member": "resource:org.acme.vehicle.auction.Member#alice@email.com",
  "timestamp": "2023-01-17T10:35:78Z"
}
```

__34. Let Bob put in a bid of 4000.

```

1  {
2    "$class": "org.acme.vehicle.auction.Offer",
3    "bidPrice": 4000,
4    "listing":
5      "resource:org.acme.vehicle.auction.VehicleListing#listing1",
6      "member":
7        "resource:org.acme.vehicle.auction.Member#bob@email.com"
8  }

```

__35. Verify the transactions in the registry.

The screenshot shows a "Default Transaction Registry" interface. On the right, a success message box displays: "Submit Transaction Successful" and "A transaction was successfully submitted". The main table has columns "ID" and "Data". Two rows are visible in the table, each representing a transaction:

- The first row has ID "f685dc27-f0ad-42f5-9684-27587e4c3892" and contains the same JSON as the previous screenshot, representing an offer from Alice at 6000.
- The second row has ID "34e04a3a-fe15-4bad-bdec-9517f09ca8af" and contains the JSON from the previous code snippet, representing an offer from Bob at 4000.

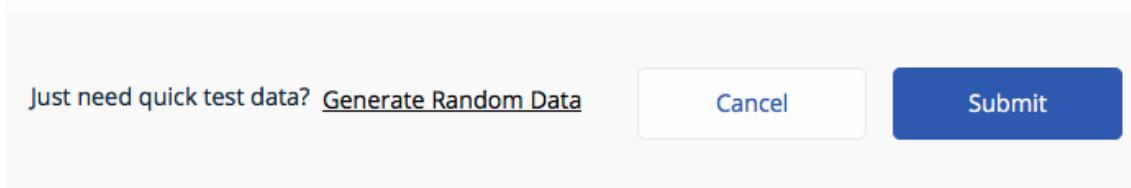
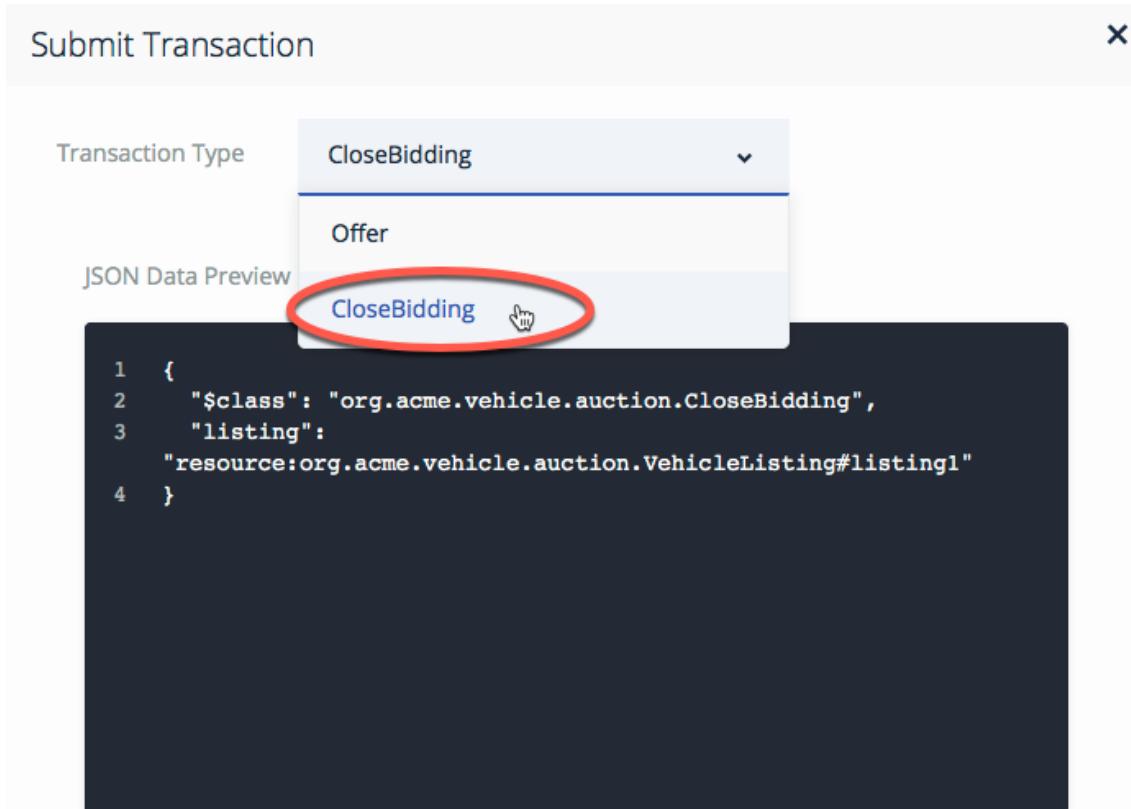
Note that the transactions cannot be edited or individually deleted once submitted; this is one of the defining characteristics of a blockchain.



Closing the bidding

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

- 36. Submit a new transaction, this time selecting **CloseBidding** from the drop-down ‘Transaction Type’ field.



- 37. Verify that the transaction has been added to the blockchain transaction registry.

ID	Data
6888fac8-72ac-47e3-976e-efe3987ad8ab	{ "\$class": "org.acme.vehicle.auction.CloseBidding", "transactionId": "6888fac8-72ac-47e3-976e-efe3987ad8ab", "listing": "resource:org.acme.vehicle.auction.VehicleListing#listing1", "timestamp": "2017-05-26T13:52:12.278Z" }

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.

- __38. Go to the *Vehicle* asset registry to see the vehicle owner has been updated to Alice.

ID	Data
1234	{ "\$class": "org.acme.vehicle.auction.Vehicle", "vin": "1234", "owner": "resource:org.acme.vehicle.auction.Member#alice@email.com" }

- __39. 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.

ID	Data
alice@email.com	{ "\$class": "org.acme.vehicle.auction.Member", "balance": 4000, "email": "alice@email.com", "firstName": "Alice", "lastName": "Smith" }
bob@email.com	{ "\$class": "org.acme.vehicle.auction.Member", "balance": 5000, "email": "bob@email.com", "firstName": "Bob", "lastName": "Jones" }
charlie@email.com	{ "\$class": "org.acme.vehicle.auction.Member", "balance": 6100, "email": "charlie@email.com", "firstName": "Charlie", "lastName": "Brown" }

■ Export the Business Network Archive

- __40. 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 environment, handed to a network operator to deploy to a live network or saved as a backup. More details on local installation at <http://fabric-composer.org>.



 Export

41. 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.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

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	
<i>System z</i>	<i>Tivoli</i>	<i>WebSphere</i>	<i>Workplace</i>	<i>System p</i>	

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



© Copyright IBM Corporation 2014.

The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, these materials. Nothing contained in these materials is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software. References in these materials to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. This information is based on current IBM product plans and strategy, which are subject to change by IBM without notice. Product release dates and/or capabilities referenced in these materials may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way.

IBM, the IBM logo and ibm.com are trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.



Please Recycle
