developerWorks COURSES

IBM

IBM Blockchain

Lab part 3: Explore the editor views and archive data

IBM

Prerequisites

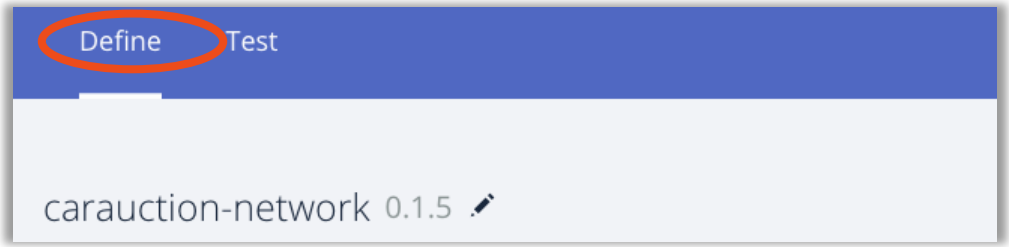
Be sure to complete part 1 and 2 of the lab and that the playground is running with the car auction demo.

Step 3. Explore the editor views and archive data

Let's look at the Model File, a JavaScript file, and the access control list (ACL) file.

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

__35. Click the **Define** tab to go back to the main playground window.



__36. Click the Model File (models/auction.cto) to open it.

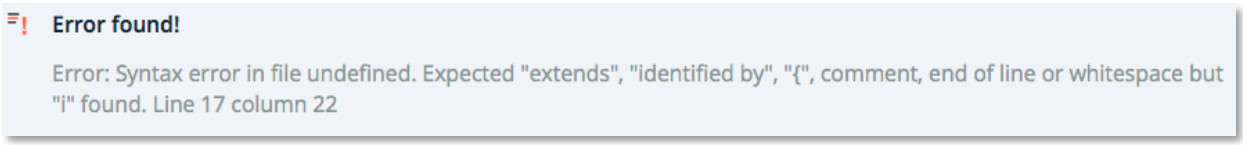
__37. Look at the Vehicle asset.

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

This uses the Hyperledger Composer Modeling Language. 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`.

__38. Type 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
}
```



__39. 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`. Because 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.

__40. 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 by using the JavaScript language.

__41. Click the **lib/logic.js** file.

__42. 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);
        });
}
```

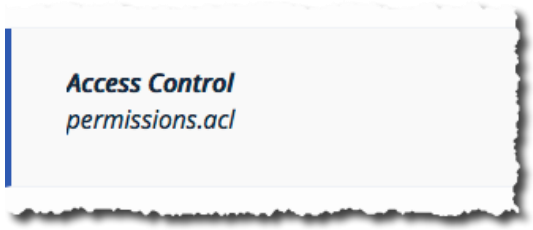
This implements the `makeOffer` function, which is executed when the `Offer` transaction is invoked on the blockchain. It is the **@param** comment above the function that links the full transaction name as defined by the model to the JavaScript method that implements it.

Other Interesting areas of the function implementation include:

- 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

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

__43. Click the **permissions.acl** file.



__44. Look at the ACL rules that are 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)

__45. 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 'Deploy' to make the changes live.

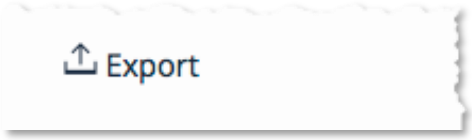
Note that when you update the model, the syntax of any existing assets in the registry must be compatible with the new model. Use either the **optional** or **default="…"** qualifiers next to the new fields. If you make incompatible changes, you must first reset the demo.

Once you've deployed 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

__46. 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.