Playground Tutorial

In this step by step tutorial we'll walk through setting up a business network, defining our assets, participants and transactions, and testing our network by creating some participants and an asset, and submitting transactions to change the ownership of the asset from one to another. This tutorial is intended to act as an introduction to Hyperledger Composer concepts using the online playground environment.

Step One: Open the Hyperledger Composer Playground

Open [Composer Playground](https://composer-playground.mybluemix.net/) (note, this link will take you to the web Composer Playground - you can also follow along in a local version if you've already installed the development environment).

You should see the **My Business Networks** screen. The **My Business Networks** page shows you a summary of the business networks you can connect to, and the identities you can use to connect to them. Don't worry about this too much for the time being, as we're going to create our own network.

Step Two: Creating a new business network

Next, we want to create a new business network from scratch. A business network has a couple of defining properties; a name, and an optional description. You can also choose to base a new business network on an existing template, or import your own template.

1. Click **Deploy a new business network** under the Web Browser heading to get started.
2. The new business network needs a name, let's call it tutorial-network.
3. Optionally, you can enter a description for your business network.
4. Next we must select a business network to base ours on, because we want to build the network from scratch, click **empty-business-network**.
5. Now that our network is defined, click **Deploy**.

**NOTE:** If you are using playground locally and connecting to a *real* Fabric please refer to the additional notes at the bottom of the tutorial.

Step Four: Adding a model file

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.

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

For more information on our modeling language, check our [documentation](https://hyperledger.github.io/composer/latest/reference/cto_language.html).

1. Click the **Model file** to view it.
2. Delete the lines of code in the model file and replace it with this:

Copy

/\*\*

\* My commodity trading network

\*/

namespace org.example.mynetwork

asset Commodity identified by tradingSymbol {

o String tradingSymbol

o String description

o String mainExchange

o Double quantity

--> Trader owner

}

participant Trader identified by tradeId {

o String tradeId

o String firstName

o String lastName

}

transaction Trade {

--> Commodity commodity

--> Trader newOwner

}

This domain model defines a single asset type Commodity and single participant type Trader and a single transaction type Trade that is used to modify the owner of a commodity.

Step Five: Adding a transaction processor script file

Now that the domain model has been defined, we can define the transaction logic for the business network. Composer expresses the logic for a business network using JavaScript functions. These functions are automatically executed when a transaction is submitted for processing.

For more information on writing transaction processor functions, check our [documentation](https://hyperledger.github.io/composer/latest/reference/js_scripts.html).

1. Click the **Add a file** button.
2. Click the **Script file** and click **Add**.
3. Delete the lines of code in the script file and replace it with the following code:

Copy

/\*\*

\* Track the trade of a commodity from one trader to another

\* @param {org.example.mynetwork.Trade} trade - the trade to be processed

\* @transaction

\*/

async function tradeCommodity(trade) {

trade.commodity.owner = trade.newOwner;

let assetRegistry = await getAssetRegistry('org.example.mynetwork.Commodity');

await assetRegistry.update(trade.commodity);

}

This function simply changes the owner property on a commodity based on the newOwner property on an incoming Trade transaction. It then persists the modified Commodity back into the asset registry, used to store Commodity instances.

Step Eight: Testing the business network definition

Next, we need to test our business network by creating some participants (in this case *Traders*), creating an asset (a *Commodity*), and then using our *Trade* transaction to change the ownership of the *Commodity*.

Click the **Test** tab to get started.

Step Ten: Creating an asset

Now that we have two *Trader* participants, we need something for them to trade. Creating an asset is very similar to creating a participant. The *Commodity* we're creating will have an *owner* property indicating that it belongs to the *Trader* with the *tradeId* of TRADER1.

* 1. Click the **Commodity** tab under **Assets** and click **Create New Asset**.
  2. Delete the asset data and replace it with the following:

Copy

{

"$class": "org.example.mynetwork.Commodity",

"tradingSymbol": "ABC",

"description": "Test commodity",

"mainExchange": "Euronext",

"quantity": 72.297,

"owner": "resource:org.example.mynetwork.Trader#TRADER1"

}

* 1. After creating this asset, you should be able to see it in the **Commodity** tab.

Logging out of the business network

Now that transactions have successfully run, we should log out of the business network, ending up at the **My Business Network** screen where we started.

* 1. In the upper-right of the screen is a button labelled **admin**. This lists your current identity, to log out, click **admin** to open the dropdown menu, and click **My Business Networks**.

Deploying a Business Network to a real Fabric.

Using Playground locally, you can use connections to "Web Browser" which works in the browser local storage, or you can use Connections to a *real* Fabric usually in a group called "hlfv1"

If you are connecting to a *real* Fabric, then you will likely have already created a Card for an identity with PeerAdmin and ChannelAdmin roles - this is often called PeerAdmin. This is the card that you use to Deploy and Update your network with Composer.

When you are deploying your network to a *real* Fabric there are additional fields to complete before you can click the **Deploy** button - you need to supply the details of the **Network Administrator**.

Scroll to the bottom of the Deploy Screen to find **CREDENTIALS FOR NETWORK ADMINISTRATOR**. For a simple Development Fabric and many Test networks you can supply an ID and Secret. Enrollment ID - admin Enrollment Secret - adminpw

When the ID and Secret are specified, you can click the **Deploy** button and resume the tutorial at Step Three.

If you are working with a Custom or Production Fabric - contact your Fabric Administrator for details of the Network Administrator.

Updating a Business Network when connected to a real Fabric

When you are using a *real* Fabric and click **Deploy Changes** you will see an addition popup dialog asking you to specify an Installation Card and an Upgrade card from dropdown lists. Typically you specify the same PeerAdmin card as used to deploy the initial network. If you are uncertain, contact your Fabric Administrator.

Select the cards, and click the **Upgrade** button. Note that on a real Fabric this can take a few minutes to complete.

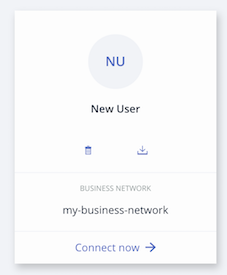
Resume the Tutorial at Step Eight.

# Business Network Cards

A Business Network Card provides all of the information needed to connect to a blockchain business network. It is only possible to access a blockchain Business Network through a valid Business Network Card. A Business Network Card contains and Identity for a single Participant within a deployed business network. Business Network Cards are used in the Hyperledger Composer Playground to connect to deployed Business Networks. You can have multiple Business Network Cards for a single deployed Business Network, where those Business Network Cards belong to multiple Participants.

Business Network Cards are grouped under a Connection Profile, and each card shows the business network that may be accessed using the displayed Identity via the Connect now option. A Business Network Card may be deleted or exported using the icons present on the card.

Here is a Business Network Card that may be used to connect to the business network named 'my-business-network' using the Identity 'New User'



A Business Network Card can be created when an Identity is issued within the Playground. This Business Network Card can then be exported and shared with others, allowing them to connect to the business network using the issued identity. If a user is provided with an Enrollment ID and Secret by an administrator, which corresponds to a valid Identity within a business network, then a Business Network Card may be directly created in the Playground. It is also possible to manually create a Business Network Card.

## Creating a peer admin card using Playground

Deploying a business network to a Hyperledger Fabric instance for the first time requires that the Hyperledger Composer chaincode be installed on the relevant Hyperledger Fabric peers, and that the business network then be instantiated on the channel. This process requires special Hyperledger Fabric privileges possessed by a peer or channel administrator.

A peer admin business network card must be created in order to deploy a Hyperledger Composer business network to a Hyperledger Fabric instance. To create the peer admin business network card:

1. From the **My Business Networks** screen, click **Create Business Network Card**.
2. Select **Hyperledger Fabric v1.1** and click **Next**.
3. Enter the details of your connection profile. For a better understanding of creating a connection profile, see [Deploying to Hyperledger Fabric](https://hyperledger.github.io/composer/latest/tutorials/deploy-to-fabric-single-org.html) and click **Next**.
4. Select **Certificates** and upload the certificate and private key information for the peer admin identity. The [Deploying to Hyperledger Fabric](https://hyperledger.github.io/composer/latest/tutorials/deploy-to-fabric-single-org.html) tutorial gives an overview of where to find the correct certificates.
5. Enter a name for the business network card.
6. Select **Admin Card**, and then **Peer Admin** and **Channel Admin**.
7. Click **Next**

The peer admin card should now be present in your **My Business Networks** screen. You can now deploy a business network to the Hyperledger Fabric instance using Playground.

## Providing Access To Your Business Network With a Business Network Card

Access to your Business Network is granted to another user through the provision of a Business Network Card. Once created, the Business Network Card may be exported and provided to the user for whom the card was created.

There must be a Participant existing in the business network in order to bind an Identity required to create the Business Network Card.

When an identity is issued within Playground, there is an option to save the identity for use in your Business Networks page. By selecting this option a Business Network Card is created for the issued identity that is valid for the current business network. The Business Network Card will appear on your Business Networks page once you log out of the current business network, and it is from this location where it may be exported to share with another user.

### **Creating a Business Network Card**

1. From the **Business Networks** page, select an identity to use to connect to your business network. Click **Connect Now**. Please note: You must use an identity that has permission to create new identities.
2. If required, create the participant to which you intend to assign the identity:
   * Click the **Test** tab, and click **Create New Participant**.
   * Complete the Participant definition and click **Create New**.

### **Exporting a Business Network Card**

Business Network Cards are exported from the **Business Networks** page in Playground, through interaction with the Business Network Card that you wish to export and is a one-step process.

* + On the **Business Networks** page, click the **Export** icon on the Business Network Card you wish to export. The Business Network Card should download as a .card file.

Please note: If you export a Business Network Card that has never been used, for example to send to a new participant, it will contain the enrollment ID and enrollment secret required to obtain the certificate and private key which are then used to identify Participants. Alternatively, if you export a Business Network Card that has been used before, it will already contain the certificate and private key. For more information on Participants and Identities, please see our [main documentation](https://hyperledger.github.io/composer/latest/managing/participantsandidentities.html).

**Important**: Exported identity cards should be handled with care since they contain unprotected credentials. We recommend that you only send identity cards that have been encrypted.

## Gaining Access To A Business Network

### **Importing a Business Network Card**

Importing a Business Network Card allows you to connect to a deployed business network.

* + On the **Business Networks** page, click **Import Business Network Card** in the upper right.
  + Drag and drop, or browse, to select a Business Network Card (.card) file to import. Click **Import**.

The Business Network Card should now be visible in your Business Networks page; you can now connect to the deployed Business Network.

### **Connecting using Credentials**

There are two ways a Business Network Card can be created from the **Business Networks** page.

A Business Network Card can be created using certificates, which requires the following: a certificate and private key, provided by an admin of a business network; a name for the Business Network Card; the business network name for which the credentials are valid; the Connection Profile details for the runtime on which the target business network is deployed.

To create a Business Network Card from the **Business Networks** page using certificates:

* + After receiving a certificate and private key, click the **Connect using credentials** button in the upper right of the **Business Networks** page.
  + If you have previously connected to the deployed runtime, select it from the displayed list, click **Next** and proceed to step 4; otherwise select the radio option to connect to a new Blockchain and click **Next**.
  + Specify the Connection Profile details that were provided to you and click **Save**.
  + Enter the certificate, private key, an Business Network Card name and business network name, and click **Create**.

The Business Network Card should now be displayed in the **Business Networks** page.

### **Manually Creating a Business Network Card**

Business Network Cards are archive (.zip) files containing up to three elements:

* + A Connection Profile. (.json)
  + A metadata file containing the data for the Identity to use to connect to the Business Network. (metadata.json)
  + An optional credentials directory containing a certificate and private key.

Please note: If there is no credentials directory, the metadata file must contain the Enrollment Secret property with the property name enrollmentSecret. If an enrollmentSecret is specified, a credentials directory with certificates will be created and populated if the Business Network Card is exported.

The metadata file should take the following format:

Copy

{

"name": "PeerAdmin",

"description": "A valid Business Network Card",

"businessNetwork": "basic-sample-network",

"enrollmentId": "UserID",

"enrollmentSecret": "UserSecret",

"roles": [

]

}

The businessNetworkName, image, enrollmentSecret, and roles properties are optional. The available roles are PeerAdmin and ChannelAdmin.

To create the Business Network Card file, compress the Connection Profile, metadata file, and optionally a credentials directory, then modify the file type to .card.

This Business Network Card can now be imported using the Hyperledger Composer Playground.

# Developing Business Networks

Developers use Hyperledger Composer to digitize business networks. The business network is accessed by multiple participants in the network, some of which may be responsible for the maintenance (hosting) of the network itself, referred to as maintainers of the network.

Typically each maintainer of the network will run several peer nodes (for crash fault tolerance) and Hyperledger Fabric replicates the distributed ledger across the set of peer nodes.

## Model

Developers work with business analysts to define the domain data model for the business network. The data model is expressed using the Composer Modeling Language and defines the structure of the resources that will be stored on the ledger, or processed as transactions.

Once the domain model is in place, developers can capture smart contracts as executable transaction processor functions, written in JavaScript.

## Access Control

In parallel developers or technical analysts can define the access control rules for the business network, to enforce which participants have access to the data on the ledger and under which conditions.

## Deploy

Developers package the models, scripts and access control rules into a deployable Business Network Archive and use command line tools to deploy the archive to a runtime for testing.

## Test

Like all business logic, it is important to create unit and system tests for business networks. Developers can use popular JavaScript testing frameworks such as Mocha and Chai to run unit tests (against the Node.js embedded runtime) or run system tests against a Hyperledger Fabric.

## Integrate

Once the business network is tested and in place, front-end applications need to be created. Use the REST Server to automatically generate a REST API for a business network, and then a skeleton generate Angular application using the Yeoman code generator.

The REST Server can be configured to authenticate the participants in the business network, ensuring that credentials and permissions are enforced.

Business Network Definition

The Business Network Definition is a key concept of the Hyperledger Composer programming model. They are represented by the BusinessNetworkDefinition class, defined in the composer-common module and exported by both composer-admin and composer-client.

Business Network Definitions are composed of:

* a set of model files
* a set of JavaScript files
* an Access Control file

The model files defined the business domain for a business network, while the JavaScript files contain transaction processor functions. The transaction processor functions run on a Hyperledger Fabric and have access to the asset registries that are stored in the world state of the Hyperledger Fabric blockchain.

The model files are typically created by business analysts, as they define the structure and relationships between model elements: assets, participants and transactions.

The JavaScript files are typically created by developers who are implementing business requirements provided by business analysts.

The Access Control file contains a set of access control rules that define the rights of the different participants in the business network.

Once defined, a Business Network Definition can be packaged into an archive using the composer command line interface. These archives can then be deployed or updated on a Fabric, using the AdminConnection class from the composer-admin module.

Create a Business Network Definition

A business network definition has the following layout:

Copy

models/ (optional)

lib/

permissions.acl (optional)

package.json

README.md (optional)

The easiest way to create a new business network definition is to either git clone an example, or to use the Hyperledger Composer Yeoman generator to create a skeleton business network.

README.md

A description of the purpose of the business network using the Markdown mark-up language.

Package.json

A Business Network Definition has a name (limited to basic ASCII alphanumeric characters and -), a human-readable description and a version number. The version number for the network should take the form Major.Minor.Micro and [Semantic Versioning](http://semver.org/) principles should be used when incrementing the version number.

The identifier of the network is formed from its name, the - character and its version number. A valid identifier (and example) is therefore mybusinessnetwork-1.0.3.

The metadata for a business network definition is read from package.json, meaning that business network definitions may also be valid npm packages.

Models

The set of domain models for a business network definition define the types that are valid within the network and outside the network when it is integrated with external systems (for example systems that submit transactions to the network).

A domain model may either be packaged within the business network definition (typically stored under the models directory), or may be declared in package.json as an external dependency. You refer to models via an npm dependency if you wanted to share them across business network definitions.

Scripts

The scripts for a business network definition are typically stored under the lib directory and are packaged within the business network definition. The scripts are written in ES 5 JavaScript and refer to the types that are defined in the domain models for the business network.

Permissions.acl

The permissions for the business network expressed are expressed in an optional permissions.acl file.

Cloning an Example Business Network Definition

The sample business network definitions are stored on GitHub at https://github.com/hyperledger/composer-sample-networks. You can git clone this repository to pull down all the sample networks. Each sample network is stored under the packages directory.

Generating a Business Network Definition

Generation

1. yo hyperledger-composer

Copy

? Please select the type of project: (Use arrow keys)

❯ Angular

Business Network

Model

And select Business Netork

1. Answer all of the questions

Copy

Welcome to the Hyperledger Composer project generator

? Please select the type of project: Business Network

You can run this generator using: 'yo hyperledger-composer:businessnetwork'

Welcome to the business network generator

? Business network name: mynetwork

? Description: This is my test network

? Author name: Mr Conga

? Author email: conga@congazone.org

? License: Apache-2

? Namespace: org.conga

create package.json

create permissions.acl

create README.md

create models/org.conga.cto

create .eslintrc.yml

create features/sample.feature

create features/support/index.js

create test/logic.js

create lib/logic.js

This generates a skeleton business network with an asset, participant and transaction defined, as well as a mocha unit test.

Also included, is a 'best practice' eslint config file which defines sample linting rules for JS code.

# Deploying Business Networks

Before a business network definition can be deployed it must be packaged into a Business Network Archive (.bna) file. The composer archive create command is used to create a business network archive file from a business network definition folder on disk.

Once the business network archive file has been created it can be deployed to Hyperledger Fabric using the [composer network install](https://hyperledger.github.io/composer/latest/reference/composer.network.install.html) command followed by a [composer network start](https://hyperledger.github.io/composer/latest/reference/composer.network.start.html) command.

For example:

Copy

composer network install --archiveFile tutorial-network@1.0.0.bna --card PeerAdmin@fabric-network

composer network start --networkName tutorial-network --networkVersion 1.0.0 --card PeerAdmin@fabric-network --networkAdmin admin --networkAdminEnrollSecret adminpw

To upgrade the business network definition for an already deployed business network use the [composer network upgrade](https://hyperledger.github.io/composer/latest/reference/composer.network.upgrade.html) CLI command.

## Deploying business networks to Hyperledger Fabric v1.1

In Hyperledger Fabric v1.1, peers enforce the concepts of administrators and members. Administrators have permission to install Hyperledger Fabric chaincode for a new business network onto peers. Members do not have permission to install chaincode. In order to deploy a business network to a set of peers, you must provide an identity that has administrative rights to all of those peers.

To make that identity and its certificates available, you must create a Peer Admin business network card using the certificate and private key associated with the peer admin identity. Hyperledger Composer provides a sample Hyperledger Fabric v1.1 network. The peer administrator for this network is called PeerAdmin, and the identity is automatically imported for you when you use the sample scripts for starting the network. Please note that the peer administrator may be given a different name for other Hyperledger Fabric networks.

**Important**: When deploying a business network to Hyperledger Fabric v1.1 a bootstrap registrar is defined in the Hyperledger Fabric Certificate Authority (CA) configuration. The Hyperledger Composer development environment contains a preconfigured instance of Hyperledger Fabric with a specific enrollment ID and enrollment secret for the bootstrap registrar.

## Business network administrators

When you deploy a business network, access controls are enforced as per the access control rules specified in the business network definition. Each business network must have at least one participant, and that participant must have a valid identity for accessing the business network. Otherwise, client applications cannot interact with the business network.

A business network administrator is a participant who is responsible for configuring the business network for their organisation after the business network is deployed, and is responsible for on-boarding other participants from their organisation. Because business networks include multiple organisations, there should be multiple business network administrators for any given business network.

A built-in participant type, org.hyperledger.composer.system.NetworkAdmin, representing a business network administrator is provided by Hyperledger Composer. This built-in participant type does not have any special permissions; they are still subject to the access control rules specified in the business network definition. For this reason, it is recommended that you start with the following sample access control rules that grant business network administrators full access to a business network:

Copy

rule NetworkAdminUser {

description: "Grant business network administrators full access to user resources"

participant: "org.hyperledger.composer.system.NetworkAdmin"

operation: ALL

resource: "\*\*"

action: ALLOW

}

rule NetworkAdminSystem {

description: "Grant business network administrators full access to system resources"

participant: "org.hyperledger.composer.system.NetworkAdmin"

operation: ALL

resource: "org.hyperledger.composer.system.\*\*"

action: ALLOW

}

By default, Hyperledger Composer will automatically create a single business network administrator participant during deployment. The identity that is used for deploying the business network will also be bound to that business network administrator participant, so that identity can be used to interact with the business network after deployment.

Hyperledger Fabric peer administrators may not have permission to issue new identities using the Hyperledger Fabric Certificate Authority (CA). This may restrict the ability of the business network administrator to on-board other participants from their organisation. For this reason, it may be preferable to create a business network administrator that does have permission to issue new identities using the Hyperledger Fabric Certificate Authority (CA).

You can use additional options to the [composer network start](https://hyperledger.github.io/composer/latest/reference/composer.network.start.html) command to specify the business network administrators that should be created during the deployment of the business network.

If the business network administrator has an enrollment ID and enrollment secret, you can use the -A(business network administrator) and -S (business network administrator uses enrollment secret) flags. For example, the following command will create a business network administrator for the existing adminenrollment ID:

Copy

composer network start --networkName tutorial-network --networkVersion 1.0.0 --c PeerAdmin@fabric-network -A admin -S adminpw

## Deploying business networks using Playground locally

**Please note**: When using a local Playground instance to deploy a business network to Hyperledger Fabric v1.1, as part of the deployment process you must choose how to provide credentials for the initial business network participant. The initial participant will be a **[NetworkAdmin](https://github.com/hyperledger/composer/blob/master/packages/composer-common/lib/system/org.hyperledger.composer.system.cto)**.

When deploying a business network using playground, you will be prompted to enter the credentials for the initial participant. Credentials can be provided either as a certificate or as a pre-defined enrollment ID and enrollment secret. If you are using the instance of Hyperledger Fabric set up in the Hyperledger Composer development environment, the bootstrap registrar enrollment ID is admin and the bootstrap registrar enrollment secret is adminpw. This initial participant uses the credentials set for the bootstrap registrar in the Hyperledger Fabric Certificate Authority (CA), and will be a **[NetworkAdmin](https://github.com/hyperledger/composer/blob/master/packages/composer-common/lib/system/org.hyperledger.composer.system.cto)**.

When deploying a business network using Playground locally, you must have at least one business network card with the PeerAdmin role and at least one business network card with the ChannelAdmin role. Each of these business network cards must contain the correct admin certificates.

Emitting Events

Events can be emitted by Hyperledger Composer and subscribed to by external applications. Events are defined in the model file of a business network definition, and are emitted by transaction JavaScript in the transaction processor functions file.

Before you begin

Before you begin adding events to your business network, you should have a good understanding of the modeling language for business networks, and what makes up a full business network definition.

Procedure

1. Events are defined in the model file (.cto) of your business network definition, in the same way as assets and participants. Events use the following format:

Copy

event BasicEvent {

}

1. In order for the event to be published the transaction which creates the event must call three functions, the first is the getFactory function. The getFactory allows events to be created as part of a transaction. Next, an event must be created by using factory.newEvent('org.namespace', 'BasicEvent'). This creates a BasicEvent defined in a specified namespace. Then the required properties on the event must be set. Lastly, the event must be emitted by using emit(BasicEvent). A simple transaction which calls this event would look like this:

Copy

/\*\*

\* @param {org.namespace.BasicEventTransaction} basicEventTransaction

\* @transaction

\*/

async function basicEventTransaction(basicEventTransaction) {

let factory = getFactory();

let basicEvent = factory.newEvent('org.namespace', 'BasicEvent');

emit(basicEvent);

}

This transaction creates and emits an event of the BasicEvent type as defined in the business network's model file. For more information on the getFactory function, see the [Composer API documentation](https://hyperledger.github.io/composer/latest/jsdoc/module-composer-runtime.html#getFactory).

# Testing Business Networks

Hyperledger Composer supports three types of testing: interactive testing, automated unit testing and automated system testing. All three serve different purposes and are vital to ensuring the success of your blockchain projects.

After you have deployed a business network definition it is often useful to run an interative "smoke test" to ensure that the deployment was successful. The composer CLI exposes several commands for running such smoke tests.

At the other end of the spectrum you can write full-blown system tests using Docker Compose and Mocha/Chai, that start a runtime, deploy your business network definition and then programmatically creates assets, submits transactions and inspect the state of asset registries.

Unit tests focus on ensuring that the correct changes to the world-state take place when a transaction is processed.

The execution of both unit tests and system tests may be automated using a CI/CD build pipeline, such as Jenkins, Travis CI, or Circle CI or alternatives.

## Interactive Testing

You can use the Playground to interactively test creating participants, assets and submitting transactions.

## Testing from the Command Line

The command line can be used to inspect the state of the runtime and to submit transactions. Use the composer network list command to see the state of asset and participant registries. Use the composer transaction submit command to submit transactions.

## Creating Unit Tests

The business logic in transaction processor functions should have unit tests, ideally with 100% code coverage. This will ensure that you do not have typos or logic errors in the business logic.

You can use standard JavaScript testing libraries, such as Mocha, Chai, Sinon and Istanbul to unit test the logic in your transaction processor functions.

The embedded runtime is very useful for unit testing, as it allows you to quickly test business logic in a simulated Node.js blockchain environment, without having to stand-up a Hyperledger Fabric.

Please refer to the sample networks for examples of unit tests. For example:<https://github.com/hyperledger/composer-sample-networks/blob/master/packages/bond-network/test/Bond.js>

# Publish Models or Business Network Definitions for use by applications

Hyperledger Composer can optionally use the npm package manager to publish both business networks, and models. By publishing business networks to npm applications that need to reference the business networks (for example to introspect them, or deploy them) can declare a binary package dependency on the published npm package. Using semantic versioning of the npm package for the business network also allows applications to specify their tolerance for accepting incompatible changes to the business network.

The npm package manager is a powerful (Internet scale) mechanism to distribute any binaries, along with metadata expressed in a package.json file.

Similarly, a set of Composer domain models (CTO files) may be packaged into an npm package for publication. The ability to publish models allows the models to be reused across multiple business networks (by declaring a package.json dependency), as well as ensures that semantic versioning can be used to control the evolution of the models themselves.

However, publication to npm is not required to use Composer. You may bundle a business network inside an application, and simply manage its source files using version control software, such as git.

The easiest way to publish a model or business network definition for use by applications it to push the business network definition to the npm package manager using the npm publish command. This will allow the applications that would like to use the business network definition (for example to deploy it via API) to reference the business network definition as a dependency in their package.json file.

Upgrading a deployed business network

After a business network has been successfully deployed to a blockchain it may be necessary to upgrade the business network definition. To upgrade a business network definition, first make the updates you wish to deploy to your local copy of the business network component files (model, script, query, access control, and transaction processor files), then update the version of your local business network files. After updating the version install the new version of the .bna to your blockchain, and use the composer network upgradecommand to switch to using your new version.

Before you begin

Before upgrading a deployed business network definition:

* Ensure your business network has successfully deployed.
* Make any required updates to your business network you wish to deploy.

Step One: Updating the business network version

It is important that the package.json file is updated before installing a new version of your business network to your blockchain.

1. Open the package.json file in your business network directory.
2. Update the **version** property. The version property must be a . separated number, for example, 0.0.2or 1.16.4. Make sure to take note of the version number you are setting as it is required for the following steps.

Step Two: Package your business network

After updating the version number, the business network must be packaged into a business network archive (.bna). The .bna can then be installed on the blockchain and started. The composer archive createcommand can package either a directory or an npm module, for this example we'll use the directory command.

1. From your business network directory run the composer archive create command:

Copy

composer archive create -t dir -n .

Step Three: Installing the new business network

When the business network has been packaged, it must be installed to the blockchain. It is installed using the same process as when the original business network was installed.

1. Install the business network to your blockchain using the following command:

Copy

composer network install -a NETWORK-FILENAME.bna -c peeradmin@hlfv1

The business network card used in the command must be a peer admin card in order to install the business network to the blockchain peers.

Step Four: Upgrading to the new business network

Now that the business network has been installed to the peers, it must be started. The composer network upgrade command will instruct the peers to stop using the older version of the business network and begin using the version specified in the command.

1. Upgrade to the business network that was installed using the following command:

Copy

composer network upgrade -c peeradmin@hlfv1 -n NETWORK-NAME -V NETWORK-VERSION

The network name and network version must match the contents of the package.json in the installed business network.

Your business network should now be successfully upgraded.

# Querying and filtering business network data

Queries are used to return data about the blockchain world-state; for example, you could write a query to return all drivers over a specified age, or all drivers with a specific name. The composer-rest-servercomponent exposes named queries via the generated REST API.

Queries are an optional component of a business network definition, written in a single query file (queries.qry).

Note: When using the Hyperledger Fabric v1.1 runtime Hyperledger Fabric must be configured to use CouchDB persistence.

Filters are similar to queries, but use the LoopBack filter syntax, and can only be sent using the Hyperledger Composer REST API. Currently, only the WHERE LoopBack filter is supported. The supported operators within WHERE are: **=**, **and**, **or**, **gt**, **gte**, **lt**, **lte**, **neq**. Filters are submitted using a GET call against an asset type, participant type, or transaction type; the filter is then supplied as a parameter. Filters return the results from the specified class, and will not return results from classes extending the specified class.

## Types of Queries

Hyperledger Composer supports two types of queries: named queries and dynamic queries. Named queries are specified in the business network definition and are exposed as GET methods by the composer-rest-server component. Dynamic queries may be constructed dynamically at runtime within a Transaction Processor function, or from client code.

### **Writing Named Queries**

Queries must contain a description and a statement. Query descriptions are a string that describe the function of the query. Query statements contain the operators and functions that control the query behavior.

Query descriptions can be any descriptive string. A query statement must include the SELECT operator and can optionally include FROM, WHERE, AND, ORDER BY, SKIP, and LIMIT.

Queries should take the following format:

Copy

query Q1{

description: "Select all drivers older than 65."

statement:

SELECT org.example.Driver

WHERE (age>65)

}

#### Query Parameters

Queries may embed parameters using the \_$ syntax. Note that query parameters must be primitive types (String, Integer, Double, Long, Boolean, DateTime), a Relationship or an Enumeration.

The named query below is defined in terms of 3 parameters:

Copy

query Q18 {

description: "Select all drivers aged older than PARAM"

statement:

SELECT org.example.Driver

WHERE (\_$ageParam < age)

ORDER BY [lastName DESC, firstName DESC]

LIMIT \_$limitParam

SKIP \_$skipParam

}

Query parameters are automatically exposed via the GET method created for named queries by the composer-rest-server.

For more information on the specifics of the Hyperledger Composer query language, see the [query language reference documentation](https://hyperledger.github.io/composer/latest/reference/query-language.html).

### **Queries using the API**

Queries can be invoked by calling the buildQuery or query APIs. The buildQuery API requires the entire query string to be specified as part of the API input. The query API requires you to specify the name of the query you wish to run.

For more information on the query APIs, see the [API documentation](https://hyperledger.github.io/composer/latest/api/client-businessnetworkconnection#buildquery).

### **Access Control for Queries**

When returning the results of a query, your access control rules are applied to the results. Any content which the current user does not have authority to view is stripped from the results.

For example, if the current user sends a query that would return all assets, if they only have authority to view a limited selection of assets, the query would return only that limited set of assets.

## Using filters

Filters can only be submitted using the Hyperledger Composer REST API, and must use the [LoopBack syntax](https://loopback.io/doc/en/lb2/Where-filter.html). To submit a query, a **GET** REST call must be submitted against an asset type, participant type, or transaction type with the filter supplied as a parameter. The supported data types for parameters to be filtered are numbers, Boolean, DateTime, and strings. A basic filter takes the following format, where op indicates an operator:

Copy

{"where": {"field1": {"op":"value1"}}}

Please note: Only the top level WHERE operator can have more than two operands.

Currently, only the WHERE LoopBack filter is supported. The supported operators within WHERE are: **=**, **and**, **or**, **gt**, **gte**, **lt**, **lte**, **neq**. Filters can combine multiple operators, in the following example, an **and** operator is nested within an **or** operator.

Copy

{"where":{"or":[{"and":[{"field1":"foo"},{"field2":"bar"}]},{"field3":"foobar"}]}}

The **between** operator returns values between the given range. It accepts numbers, datetime values, and strings. If supplied with strings, the **between** operator returns results between the supplied strings alphabetically. In the example below, the filter will return all resources where the driver property is alphabetically between a and c, inclusively.

Copy

{"where":{"driver":{"between": ["a","c"]}}}

Programmatic access control

It is recommended that you use declarative access control to implement access control rules in your business network definition. However, you can implement programmatic access control in your transaction processors by retrieving and testing either the current participant or the current identity. You can run tests against the properties of the current participant or the current identity to permit or reject the execution of a transaction processor function.

A transaction processor function can call the getCurrentParticipant function to get the current participant:

Copy

let currentParticipant = getCurrentParticipant();

The current participant is an instance of a modelled participant from the business network definition, or an instance of the system type org.hyperledger.composer.system.NetworkAdmin.

A transaction processor function can call the getCurrentIdentity function to get the current identity:

Copy

let currentIdentity = getCurrentIdentity();

The current identity is an instance of the system type org.hyperledger.composer.system.Identity, which represents an identity within a deployed business network.

Before you start

Before you follow these steps, you must have modeled a participant in a business network definition and deployed it as a business network. You must have created some instances of those participants, and issued those participants with identities.

The procedure below shows an example using the following participant models:

Copy

namespace net.biz.digitalPropertyNetwork

participant Person identified by personId {

o String personId

o String firstName

o String lastName

}

participant PrivilegedPerson extends Person {

}

Procedure

1. In your transaction processor function, verify the type of the current participant meets the requirements by using the getCurrentParticipant function:

Copy

async function onPrivilegedTransaction(privilegedTransaction) {

let currentParticipant = getCurrentParticipant();

if (currentParticipant.getFullyQualifiedType() !== 'net.biz.digitalPropertyNetwork.PrivilegedPerson') {

throw new Error('Transaction can only be submitted by a privileged person');

}

// Current participant must be a privileged person to get here.

}

1. In your transaction processor function, verify the participant ID of the current participant by using the getCurrentParticipant function:

Copy

async function onPrivilegedTransaction(privilegedTransaction) {

let currentParticipant = getCurrentParticipant();

if (currentParticipant.getFullyQualifiedIdentifier() !== 'net.biz.digitalPropertyNetwork.Person#PERSON\_1') {

throw new Error('Transaction can only be submitted by person 1');

}

// Current participant must be person 1 to get here.

}

The participant ID of the current participant can be compared to a participant that is linked to an asset (by a relationship) to verify that the current participant has the authority to access or modify an asset:

Copy

async function onPrivilegedTransaction(privilegedTransaction) {

// Get the owner of the asset in the transaction.

let assetOwner = privilegedTransaction.asset.owner;

let currentParticipant = getCurrentParticipant();

if (currentParticipant.getFullyQualifiedIdentifier() !== asset.owner.getFullyQualifiedIdentifier()) {

throw new Error('Transaction can only be submitted by the owner of the asset');

}

// Current participant must be the owner of the asset to get here.

}

1. In your transaction processor function, verify the certificate of the current identity meets the requirements by using the getCurrentIdentity function:

Copy

async function onPrivilegedTransaction(privilegedTransaction) {

let currentIdentity = getCurrentIdentity();

// Get the PEM encoded certificate from the current identity.

let certificate = currentIdentity.certificate;

// Perform testing on the PEM encoded certificate.

if (!certificate.match(/^----BEGIN CERTIFICATE----/)) {

throw new Error('Transaction can only be submitted by a person with a valid certificate');

}

// Current identity must have a valid certificate to get here.

}

# Hyperledger Composer Historian

The Hyperledger Composer Historian is a specialised registry which records successful transactions, including the participants and identities that submitted them. The historian stores transactions as HistorianRecordassets, which are defined in the Hyperledger Composer system namespace.

The historian registry is a Hyperledger Composer system-level entity. To refer to the historian registry as a resource for access control the historian must be referenced as: org.hyperledger.composer.system.HistorianRecord.

**Please note:** All participants must have the permission to create HistorianRecord assets. If a transaction is submitted by a participant who does not have the permission to create HistorianRecord assets, the transaction will fail.

## HistorianRecord assets

The historian registry stores successful transactions as HistorianRecord assets. Whenever a transaction successfully completes, a HistorianRecord asset is created and added to the historian registry. Record assets are defined in the system namespace, and have the following definition:

Copy

asset HistorianRecord identified by transactionId {

o String transactionId

o String transactionType

--> Transaction transactionInvoked

--> Participant participantInvoking optional

--> Identity identityUsed optional

o Event[] eventsEmitted optional

o DateTime transactionTimestamp

}

* String transactionId The transactionId of the transaction that caused the HistorianRecord asset to be created.
* String transactionType The class of transaction that caused the HistorianRecord asset to be created.
* Transaction transactionInvoked A relationship to the transaction which caused the HistorianRecordasset to be created.
* Participant participantInvoking A relationship to the participant who submitted the transaction.
* Identity identityUsed A relationship to the identity used to submit the transaction.
* Event[] eventsEmitted An optional property containing any events which were emitted by the transaction.
* DateTime transactionTimestamp The timestamp of the transaction which caused the HistorianRecordasset to be created.

All HistorianRecord assets have relationships to the transaction that created them, the invoking participant of that transaction, and the identity used when the transaction was submitted. Applications that wish to obtain these attributes must resolve this relationship.

## System transactions

Several operations that the Hyperledger Composer runtime makes are classed as transactions. These 'system transactions' are defined in the Hyperledger Composer system model. The following will add HistorianRecordassets:

* Adding, removing and updating assets
* Adding, removing and updating participants
* Issuing, binding, activating and revoking identities
* Updating the business network definition

## Securing historian data

As a registry, access to the historian data can be controlled with access control rules. However, as a system-level entity the resource name for the historian registry is always org.hyperledger.composer.system.HistorianRecord.

The following access control rule allows members to only see historian data if it references transactions they submitted.

Copy

rule historianAccess{

description: "Only allow members to read historian records referencing transactions they submitted."

participant(p): "org.example.member"

operation: READ

resource(r): "org.hyperledger.composer.system.HistorianRecord"

condition: (r.participantInvoking.getIdentifier() == p.getIdentifier())

action: ALLOW

}

## Retrieving historian data

Data from the historian registry can be retrieved using either an API call, or queries. All examples that follow make use of the async/await feature and assume that the code is encapsulated in a function with the asyncattribute.

### **Using the client and REST APIs with historian**

HistorianRecord assets can be returned using the system/historian and system/historian/{id} calls using the REST API.

When using the REST API, a GET call of system/historian will return ALL historian data. This call should be used with care, the return is not limited and may result in large volumes of data being returned.

A GET call of system/historian/{id} using the REST API will return the HistorianRecord asset specified.

### **Querying the Historian**

Historian can be queried in the same manner as other registries. For example, a typical query to return all HistorianRecord assets would be as follows:

Copy

let historian = await businessNetworkConnection.getHistorian();

let historianRecords = await historian.getAll();

console.log(prettyoutput(historianRecords));

As this is a 'getAll' call it will potentially return high volume of data. Therefore the query capability is vital in being able to select a subset of records. A typical example would be to select records based on a time. This uses the query capability to select records where the transaction timestamp is past a certain point. The returned records can be processed in the same way.

Copy

let now = new Date();

now.setMinutes(10); // set the date to be time you want to query from

let q1 = businessNetworkConnection.buildQuery('SELECT org.hyperledger.composer.system.HistorianRecord ' +

'WHERE (transactionTimestamp > \_$justnow)');

await businessNetworkConnection.query(q1,{justnow:now});

More advanced queries can be used; for example, the following query selects and returns the Add, Update, and Remove asset system transactions.

Copy

// build the special query for historian records

let q1 = businessNetworkConnection.buildQuery(

`SELECT org.hyperledger.composer.system.HistorianRecord

WHERE (transactionType == 'AddAsset' OR transactionType == 'UpdateAsset' OR transactionType == 'RemoveAsset')`

);

await businessNetworkConnection.query(q1);

Customising the card store

The default card store is the /home/username/.composer directory on the host machine. Local wallets can be problematic for applications running in cloud environments, and it may be desired to have the card store at different directory location. By using custom wallets, users can control where business network cards and the certificates and private keys used for Hyperledger Fabric authentication are stored.

Architecture

Whenever a BusinessNetworkConnection or AdminConnection is made, it has an associated CardStore. Each connection can be configured to use a specific CardStore. In the Hyperledger Composer repository, there are two pre-configured options for stores:

* composer-wallet-filesystem
* composer-wallet-inmemory

Custom implementations can be written for any given backend database or object store, enabling the specification of a CardStore that is in a non-default file location, a separate docker container, or hosted in a cloud based data store. The store configuration can be completed using either a configuration file, or by using environment variables.

* [composer-tools/composer-wallet-redis](https://github.com/hyperledger/composer-tools/tree/master/packages/composer-wallet-redis) - provides a backing store using a Redis server
* [@ampretia/composer-wallet-ibmcos](https://github.com/ampretia/composer-wallet-ibmcos) - provides a backing store using the IBM Cloud Object Store. This has an S3 compatible API

Multiple cloud wallet implementations can be installed using global npm installs.

For more details of the writing a new cloud wallet implementation, see the following [README](https://github.com/hyperledger/composer-tools/tree/master/packages/composer-wallet-redis).

Configuring a custom wallet

There are two ways to define the configuration for a custom wallet: by using a .json configuration file, or by defining environment variables.

Please note: any custom wallet implementation **must** include the composer-wallet prefix in the module name.

Using a configuration file

For production deployments, it is more useful to be able to configure the card store outside of the application, Hyperledger Composer uses the standard configuration module config. The configuration file is loaded from a sub-directory of the current working directory called config. The default configuration file is called default.json, the configuration file name can be changed using the NODE\_ENV environment variable.

The following configuration file uses the Redis format as an example:

Copy

{

"composer": {

"wallet": {

"type": "composer-wallet-redis",

"desc": "Uses a local redis instance,

"options": {

}

}

}

}

* type is the name of this module
* desc is some text for the humans

Please note: Each connection will have a new instance of the card store specified. If these resolve to the same backend store, cards can be shared.

Using an environment variable

Specifying the details of a custom wallet on the command line via environment variables may be achieved by setting an environment variable containing the same information as the configuration file.

The following environment variable example uses the same format and data as the preceding configuration file.

Copy

export NODE\_CONFIG={"composer":{"wallet":{"type":"composer-wallet-redis","desc":"Uses a local redis instance,"options":{}}}}

Any application that is in this shell will use the cloud wallets.

Configuring file system custom card stores

The location of the file system card store can be changed using a configuration file, through specification of a storePath as one of the wallet options.

Copy

{

"composer": {

"wallet" : {

"type": "composer-wallet-filesystem",

"options" : {

"storePath" : "/my/network/location"

}

}

}

The same .json snippet may be exported as an environment variable.

Configuring cloud based custom card stores

The following GitHub repositories contain implementations of cloud custom wallets using Redis and the IBM Cloud Object Store, respectively.

* [composer-tools/composer-wallet-redis](https://github.com/ampretia/composer-wallet-redis) - provides a backing store using a Redis server
* [@ampretia/composer-wallet-ibmcos](https://github.com/ampretia/composer-wallet-ibmcos) - provides a backing store using the IBM Cloud Object Store. This has an S3 compatible API.

Multiple cloud custom wallet implementations can be installed using global npm installs.

For more details of the writing a new cloud based custom wallet implementation, see the following [README](https://github.com/hyperledger/composer-tools/tree/master/packages/composer-wallet-redis).

To migrate to either the Redis or IBM Cloud Object Store cloud custom wallet solutions, refer to the README files of the relevant GitHub repository.

In a general sense, migrating to a cloud wallet implementation has three steps.

1. Export the business network cards you wish to use in the cloud custom wallet.
2. Change configuration to specify the cloud custom wallet.
3. Import the business network cards into the cloud custom wallet.

The composer-wallet-filesystem is the default card store and follows the same layout on disc, and by default is in the same location.

Some samples and test cases show the card stores being created programmatically. This is still possible and but is slightly different in terms of initial creation of the card store.

Using custom wallets with APIs

API CardStore configuration

Using the default location file system card store remains the default option within API calls. For instance:

Copy

adminConnection = new AdminConnection();

clientConnection = new BusinessNetworkConnection();

will use the file system card store at the location /home/username/.composer, or pick up on the exported custom wallet specified within NODE\_CONFIG if and only if executing within the same shell instance.

To specify a custom wallet within the API, without the use of a globally exported value, it must be included as an option passed to the connection:

Copy

const connectionOptions = {

wallet : {

type: 'composer-wallet-filesystem',

options : {

storePath :'/my/network/location'

}

}

};

adminConnection = new AdminConnection(connectionOptions);

clientConnection = new BusinessNetworkConnection(connectionOptions);

In the above, the wallet type may be that of a new file location, or a cloud based location.

API MemoryCardStore configuration

Previously to use the in MemoryCardStore, the code would have been written

Copy

cardStore = new MemoryCardStore();

const adminConnectionOptions = {

cardStore : cardStore

};

adminConnection = new AdminConnection(adminConnectionOptions);

// or more concisely

clientConnection = new BusinessNetworkConnection({cardStore});

This has now changed and Card stores must now be specified differently:

Copy

const connectionOptions = {

wallet : {

type: 'composer-wallet-inmemory'

}

};

adminConnection = new AdminConnection(connectionOptions);

clientConnection = new BusinessNetworkConnection(connectionOptions);

# Managing your Hyperledger Composer Solution

### **Participants and identities**

Participants and identities are core concepts of Hyperledger Composer. A participant is a member of business networks and might represent individuals or organizations. Participants have identity documents which can be validated to prove their identity. For more information, see [**participants and identities**](https://hyperledger.github.io/composer/latest/managing/participantsandidentities.html).

### **Adding participants**

[**Participants must be added to a business network**](https://hyperledger.github.io/composer/latest/managing/participant-add.html) before they can make transactions. Participants can create assets, and also exchange assets with other participants. A participant works with assets by submitting transactions.

### **Creating, Exporting, and Importing Business Network Cards**

Business network cards combine a connection profile, identity, and certificates to allow a connection to a business network in Hyperledger Composer Playground. Business network cards can be [created, exported and imported](https://hyperledger.github.io/composer/latest/managing/id-cards-playground.html) from the **My Wallet** page in Hyperledger Composer Playground.

### **Issuing a new identity to a participant**

[**A new identity can be issued to a participant using either the API or the command line**](https://hyperledger.github.io/composer/latest/managing/identity-issue.html). Once a new identity has been issued, the identity can then be used by the participant to interact with the business network in the context of that participant.

### **Binding an existing identity to a participant**

[**An existing identity can be bound to a participant using either the API or the command line**](https://hyperledger.github.io/composer/latest/managing/identity-bind.html). Once an existing identity has been bound, the identity can then be used by the participant to interact with the business network in the context of that participant.

### **Listing all identities in a business network**

Identities issued or bound to a participant create a mapping. In order to perform identity management operations in a deployed business network, you will need to [list](https://hyperledger.github.io/composer/latest/managing/identity-list.html) and review the set of identities in the identity registry.

### **Revoking an identity from a participant**

[**An identity can be revoked from a participant using either the API or the command line**](https://hyperledger.github.io/composer/latest/managing/identity-revoke.html). Once an identity has been revoked, the identity can no longer be used by the participant to interact with the business network in the context of that participant.

### **Interacting with Hyperledger Fabric**

Hyperledger Composer is designed to be platform-agnostic. This section is about specifics in relation to interacting with Hyperledger Fabric.

Participants and identities

Concepts

A Participant is an actor in a business network. A participant might be an individual an organization. A participant can create assets, and also exchange assets with other participants. A participant works with assets by submitting transactions.

A participant has a set of Identity documents that can be validated to prove the identity of that participant. For example, an individual may have one or more of the following identity documents that prove who they are:

* Passport
* Driving license
* Fingerprints
* Retina scan
* SSL certificate

In Hyperledger Composer, participants are separated from the set of identity documents that they can use to interact with a business network.

In order for a new participant to join a business network, a new instance of that participant must be created in the business network. The participant instance stores all of the required information about that participant, but it does not give that participant access to interact with the business network.

In order to grant the participant access to interact with the business network, an identity document must then be Issued to that participant. The new participant can then use that identity document to interact with the business network.

A participant may have an existing identity document that they use to interact with other business networks or other external systems. These identity documents can be reused and Bound to that participant. The new participant can then use their existing identity document to interact with the business network.

Identity documents usually expire after a set period of time. Identity documents may also be lost or stolen. If the identity document expires, or if it needs to be replaced, then it must be Revoked so it can no longer be used to interact with the business network.

However, revoking an identity document does not remove the information about that participant and any assets that they own. Revoking the identity document simply removes the participants ability to interact with the business network using that identity document. Access to the business network can be restored by issuing the participant with a new identity document.

These participant and identity management actions are performed by an existing participant in the business network, for example a regulatory body, or a participant in the same organization who has been trusted to manage participants/identities in that organization.

Participants and identities in Hyperledger Composer

In Hyperledger Composer, the structure of a participant is modeled in a model file. This structure may include various information about the participant, for example the participants name, address, e-mail address, date of birth, etc. New instances of that modeled participant can then be created and added to a participant registry.

Hyperledger Composer requires the use Blockchain identities as the form of identity documents. For example, when deploying a business network to Hyperledger Fabric, enrollment certificates are used as the form of identity document. These enrollment certificates are used to cryptographically sign the transactions that are submitted to the deployed business network.

A deployed business network maintains a set of mappings of identities to participants in the Identity Registry. When an identity is Issued or Bound to a participant, a new mapping is added to the identity registry. When that participant uses that identity to submit transactions to the deployed business network, the Composer runtime looks for a valid mapping for that identity in the identity registry. This lookup is done using the public key signature or fingerprint, essentially a hash of the certificate contents that is unique to that certificate and identity.

Once a mapping is found in the identity registry, the participant for that identity is retrieved from that mapping. That participant becomes the Current Participant, the participant who submitted the transaction. All access control in Hyperledger Composer is based around the current participant. Access control rules that define which participants can perform which operations on which resources all operate on the current participant.

When a participant uses an identity to submit a transaction to the deployed business network for the first time, that identity is Activated. This means that the entry in the identity registry is updated to record the fact that the identity was used for the first time. Additional information about the identity, such as the certificate, may also be recorded in the identity registry during activation if it was not available when the identity was issued or bound to the participant.

If and when an identity is revoked, the entry in the identity registry for that identity is updated to change the status to Revoked. After an identity is revoked, if a participant tries to use that identity to submit a transaction to the deployed business network, that transaction will be rejected.

Identities and Business network cards in the Hyperledger Composer Playground

In the Hyperledger Composer Playground, there is a wallet containing locally stored Business network cards. A Business network card is an access card to a business network, comprising identity data, a connection profile, and the correct certificates for business network access. ID cards can be exported to allow the assignment of identities to others.

Performing identity management tasks in Hyperledger Composer

The Hyperledger Composer Node.js client APIs, REST APIs, and command line interfaces can all be used to perform identity management operations. For example, the following identity management operations are available through all Hyperledger Composer interfaces:

* Adding a new participant to a participant registry
* Issuing a new identity to a participant
* Binding an existing identity to a participant
* Revoking an identity from a participant
* Listing all identities in a deployed business network

Adding participants

A participant can be added to a participant registry using either the API or the command line.

Before you start

Before you follow these steps, you must have modeled a participant in a Business Network Definition and deployed it as a Business Network.

The procedure below shows an example using the following model of a participant from the Digital Property sample Business Network Definition: [digitalproperty-network](https://www.npmjs.com/package/digitalproperty-network)

*Please note*: If you are adding the participant using the composer participant add command, ensure that the JSON representation of the participant is wrapped in single quotes.

Copy

namespace net.biz.digitalPropertyNetwork

participant Person identified by personId {

o String personId

o String firstName

o String lastName

}

Procedure

1. Add the participant to a participant registry
   * JavaScript API

Copy

const BusinessNetworkConnection = require('composer-client').BusinessNetworkConnection;

async function addParticipant() {

let businessNetworkConnection = new BusinessNetworkConnection();

try {

await businessNetworkConnection.connect('admin@digitalPropertyNetwork');

let participantRegistry = await businessNetworkConnection.getParticipantRegistry('net.biz.digitalPropertyNetwork');

let factory = businessNetworkConnection.getFactory();

let participant = factory.newResource('net.biz.digitalPropertyNetwork', 'Person', 'mae@biznet.org');

participant.firstName = 'Mae';

participant.lastName = 'Smith';

await participantRegistry.add(participant);

await businessNetworkConnection.disconnect();

} catch(error) {

console.error(error);

process.exit(1);

}

}

addParticipant();

* Command line

Copy

composer participant add -c admin@network -d '{"$class":"net.biz.digitalPropertyNetwork.Person","personId":"mae@biznet.org","firstName":"Mae","lastName":"Smith"}'

# Creating, Exporting, and Importing Business Network Cards

Business network cards are represented by .card files containing a metadata.JSON file, a connection profile, and optional certificates.

Business network cards can be used in the Hyperledger Composer Playground to manage identities for different business networks and connection profiles.

## Creating Business Network Cards

Business network cards can be created in the wallet screen, created from the component files, or created within a business network.

Please note: If cards are created from the wallet screen, or created from the component files, there must be a corresponding identity already created in the business network.

### **Creating a business network card within a business network**

1. From the **My Wallet** screen, select an identity to use to connect to your business network. Click **Connect Now**. Please note: You must use an identity with the permission to create new identities.
2. Optional: To create a participant to assign to the identity, click the **Test** tab, and click **Create New Participant**.
3. Click the name of your identity in the upper right, and click **ID Registry**.
4. Click **Issue New ID**.
5. Choose an **ID Name**, and select a **Participant** to associate the new identity with.
6. Click **Create New**.
7. Click **Add to My Wallet**. Adding the business network card to your wallet allows you to use it to connect to the business network, or export it for someone else to use.

The **My Wallet** screen should now show the new business network card.

### **Creating a business network card from the Wallet**

A business network card can be created from the **My Wallet** page, however, a corresponding identity must already have been created in the business network. Creating a business network card from the **My Wallet** page requires that you use the same User ID, User Secret and the correct Business network name credentials as when the identity was created within the business network.

To create a business network card from the **My Wallet** page:

1. After receiving a valid User ID and User Secret, click the **Create Business Network Card** button in the upper right of the **My Wallet** page.
2. Select a connection profile and click **Next**.
3. Enter the User ID and User Secret that were generated when the identity was created.
4. Enter the correct Business Network Name and click **Create**.

The business network card should now be displayed in the **My Wallet** page.

### **Creating a business network card from component files**

business network cards are composite files containing up to three elements:

* A connection profile. (connection.json)
* A metadata file containing the data for the identity to use to connect to the business network. (metadata.json)
* An optional credentials directory containing a certificate and private key for the identity in files named certificate and privateKey respectively.

Please note: If there is no credentials directory, the metadata file must contain the enrollment secret required to obtain the credentials with the property name enrollmentSecret. If an enrollmentSecret is specified and the business network card is used to connect to a business network, a credentials directory with certificates will be created and populated if the business network card is exported.

The metadata file should take the following format:

Copy

{

"version": 1,

"userName": "alice",

"description": "Alice's identity for basic-sample-network",

"businessNetwork": "basic-sample-network",

"enrollmentSecret": "UserSecret",

"roles": [

]

}

The businessNetworkName, description, enrollmentSecret, and roles properties are optional. The available roles are PeerAdmin and ChannelAdmin.

To create the business network card file, run the composer card create command.

This business network card can now be imported using the Hyperledger Composer Playground.

## Importing and Exporting business network cards

Importing and exporting business network cards is the simplest way to grant access to other users of the business network in Playground. Valid business network cards must be created using one of the methods above, but can then be exported and sent to other users.

### **Exporting Business Network Cards**

1. To export a business network card create an identity by [using a business network](https://hyperledger.github.io/composer/latest/managing/id-cards-playground#creating-an-id-card-within-a-business-network) and add the business network card to your wallet.
2. On the **My Wallet** page, click the **Export** icon on the business network card you wish to export. The business network card should download as a .card file.

Please note: If you export a business network card that has never been used, for example to send to a new participant, it will contain the enrollment ID and enrollment secret required to obtain the certificate and private key which are then used to identify participants. Alternatively, if you export a business network card that has been used before, it will already contain the certificate and private key.

**Important**: Exported identity cards should be handled with care since they contain unprotected credentials. For example, you should never send identity cards via email or other unencrypted means of communication.

### **Importing Business Network Cards**

Importing a business network card allows you to connect to a business network without creating a connection profile, identity, and certificates. Members of a business network can create business network cards and export them to give others access to a business network.

1. On the **My Wallet** screen, click **Import business network card** in the upper right.
2. Drag and drop, or browse, to select a business network card (.card) file to import. Click **Import**.

The business network card should now be visible in your wallet.

Issuing a new identity to a participant

A new identity can be issued to a participant using either the API, the command line, or by using ID cards in the Hyperledger Composer Playground. Once a new identity has been issued, the identity can then be used by the participant to interact with the business network in the context of that participant.

When using Hyperledger Fabric, Hyperledger Composer issues new identities by using the Hyperledger Fabric certificate authority (CA) to register new enrollment certificates. The Hyperledger Fabric certificate authority generates an enrollment secret that can be given to the participant, who can then use the enrollment secret to request their enrollment certificate and private keys from the Hyperledger Fabric certificate authority.

Before you start

Before you follow these steps, you must have added a participant to a participant registry. The **issuer** of a new identity (whether using command line or using the Javascript APIs below) must itself have 'issuer' authority and as appropriate, ACLs that permit them to issue the identity (to be associated with the participant) in Hyperledger Composer.

The procedure below shows an example using the following model of a participant from the Digital Property sample Business Network Definition: [digitalproperty-network](https://www.npmjs.com/package/digitalproperty-network)

Copy

namespace net.biz.digitalPropertyNetwork

participant Person identified by personId {

o String personId

o String firstName

o String lastName

}

The example assumes that an instance, net.biz.digitalPropertyNetwork#mae@biznet.org, of that participant has been created and placed into a participant registry.

Procedure

1. Connect to the business network and issue a new identity to a participant
   * JavaScript API

Copy

const BusinessNetworkConnection = require('composer-client').BusinessNetworkConnection;

async function identityIssue() {

let businessNetworkConnection = new BusinessNetworkConnection();

try {

await businessNetworkConnection.connect('admin@digitalPropertyNetwork');

let result = await businessNetworkConnection.issueIdentity('net.biz.digitalPropertyNetwork.Person#mae@biznet.org', 'maeid1')

console.log(`userID = ${result.userID}`);

console.log(`userSecret = ${result.userSecret}`);

await businessNetworkConnection.disconnect();

} catch(error) {

console.log(error);

process.exit(1);

}

}

identityIssue();

* Command line

Copy

composer identity issue -c admin@network -f maeid1.card -u maeid1 -a "resource:net.biz.digitalPropertyNetwork.Person#mae@biznet.org"

This will issue card for the user maeid1 and export a card file in your current directory.

1. As the participant, test the connection to the business network
   * JavaScript API

Copy

const BusinessNetworkConnection = require('composer-client').BusinessNetworkConnection;

async function testConnection() {

let businessNetworkConnection = new BusinessNetworkConnection();

try {

await businessNetworkConnection.connect('admin@digitalPropertyNetwork');

let result = await businessNetworkConnection.ping();

console.log(`participant = ${result.participant ? result.participant : '<no participant found>'}`);

await businessNetworkConnection.disconnect();

} catch((error) {

console.error(error);

process.exit(1);

}

}

testConnection();

* Command line

Copy

composer card import -f maeid1@network.card

composer network ping -c maeid1@network

You need to make sure to import the card into business network before pinging.

Binding an existing identity to a participant

An existing identity can be issued to a participant using either the API or the command line. Once the existing identity has been bound, the identity can then be used by the participant to interact with the business network in the context of that participant.

When using Hyperledger Fabric, you can bind existing certificates that have been created by using the Hyperledger Fabric certificate authority (CA) or by using other tooling such as cryptogen. The existing certificates must be valid for use for submitting transactions on the Hyperledger Fabric network.

Before you start

Before you follow these steps, you must have added a participant to a participant registry. You must have an existing certificate in the PEM format to bind to the participant. The **binder** of the existing identity (whether using command line or using the Javascript APIs below) must have ACLs that permit them to bind the identity (to be associated with the participant) in Hyperledger Composer.

The procedure below shows an example using the following model of a participant from the Digital Property sample Business Network Definition: [digitalproperty-network](https://www.npmjs.com/package/digitalproperty-network)

Copy

namespace net.biz.digitalPropertyNetwork

participant Person identified by personId {

o String personId

o String firstName

o String lastName

}

The example assumes that an instance, net.biz.digitalPropertyNetwork#mae@biznet.org, of that participant has been created and placed into a participant registry.

Procedure

1. Connect to the business network and bind an existing identity to a participant
   * JavaScript API

Copy

const BusinessNetworkConnection = require('composer-client').BusinessNetworkConnection;

async function bind() {

let businessNetworkConnection = new BusinessNetworkConnection();

let certificate = `-----BEGIN CERTIFICATE-----

MIIB8DCCAZegAwIBAgIURanHh55fqrUecvHNHtcMKiHJRkwwCgYIKoZIzj0EAwIw

czELMAkGA1UEBhMCVVMxEzARBgNVBAgTCkNhbGlmb3JuaWExFjAUBgNVBAcTDVNh

biBGcmFuY2lzY28xGTAXBgNVBAoTEG9yZzEuZXhhbXBsZS5jb20xHDAaBgNVBAMT

E2NhLm9yZzEuZXhhbXBsZS5jb20wHhcNMTcwNzI3MTc0MzAwWhcNMTgwNzI3MTc0

MzAwWjAQMQ4wDAYDVQQDEwVhZG1pbjBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IA

BAANIGFIrXXr5+h0NfUNJhx5YFQ4w6r182eZYRhc9KvYQhYo5D0ZbecfR9sGX2b6

0aW+C7bUaXc6DU3pJSD4fNijbDBqMA4GA1UdDwEB/wQEAwIHgDAMBgNVHRMBAf8E

AjAAMB0GA1UdDgQWBBRwuAyWrGlzVQFqRf0OqoTNuoq7QDArBgNVHSMEJDAigCAZ

q2WruwSAfa0S5MCpqqZknnCGjjq9AhejItieR+GmrjAKBggqhkjOPQQDAgNHADBE

AiBcj/JvxmKHel4zQ3EmjITEFhdYku5ijIZEDuR5v9HK3gIgTUbVEfq3MuasVZKx

rkM5DH3e5ECM7T+T1Ovr+1AK6bs=

-----END CERTIFICATE-----`

try {

await businessNetworkConnection.connect('admin@digitalPropertyNetwork');

await businessNetworkConnection.bindIdentity('net.biz.digitalPropertyNetwork.Person#mae@biznet.org', certificate);

await businessNetworkConnection.disconnect();

} catch(error) {

console.error(error);

process.exit(1);

}

}

bind();

* Command line

Copy

composer identity bind -c admin@digitalPropertyNetwork -a "resource:net.biz.digitalPropertyNetwork.Person#mae@biznet.org" -e mae-pub.pem

1. As the participant, test the connection to the business network
   * JavaScript API

Copy

const BusinessNetworkConnection = require('composer-client').BusinessNetworkConnection;

async function testConnection() {

let businessNetworkConnection = new BusinessNetworkConnection();

try {

await businessNetworkConnection.connect('admin@digitalPropertyNetwork');

let result = await businessNetworkConnection.ping();

console.log(`participant = ${result.participant ? result.participant : '<no participant found>'}`);

await businessNetworkConnection.disconnect();

} catch(error) {

console.error(error);

process.exit(1);

}

}

testConnection();

* Command line

Copy

composer network ping -c admin@digitalPropertyNetwork

The participant ID will be printed to the console, and should match the participant ID that was specified in the composer identity bind command.

Listing all identities in a business network

When a new identity is issued to a participant, or an existing identity is bound to a participant, a mapping between the identity and the participant is created in the identity registry in the deployed business network. When that participant uses that identity to submit transactions to the deployed business network, the Composer runtime looks for a valid mapping for that identity in the identity registry. This lookup is done using the public key signature or fingerprint, essentially a hash of the certificate contents that is unique to that certificate and identity.

In order to perform identity management operations in a deployed business network, you will need to list and review the set of identities in the identity registry.

Before you start

Before you follow these steps, you should have added a participant to a participant registry, and issued a new identity or bound an existing identity to that participant. Otherwise the identity registry will be empty and you will not see any results.

Procedure

1. Connect to the business network and list the identities in the identity registry
   * JavaScript API

Copy

const BusinessNetworkConnection = require('composer-client').BusinessNetworkConnection;

async function identityList() {

let businessNetworkConnection = new BusinessNetworkConnection();

try {

await businessNetworkConnection.connect('admin@digitalPropertyNetwork');

let identityRegistry = await businessNetworkConnection.getIdentityRegistry();

let identities = await identityRegistry.getAll();

identities.forEach((identity) => {

console.log(`identityId = ${identity.identityId}, name = ${identity.name}, state = ${identity.state}`);

});

await businessNetworkConnection.disconnect();

} catch(error) {

console.log(error);

process.exit(1);

}

}

* Command line

Copy

composer identity list -c admin@digitalPropertyNetwork

Revoking an identity from a participant

An identity can be revoked from a participant using either the API or the command line. Once an identity has been revoked, the identity can no longer be used by the participant to interact with the business network in the context of that participant.

When using Hyperledger Fabric, Hyperledger Composer does not currently attempt to revoke the identity by using the Hyperledger Fabric certificate authority (CA) APIs. The identity can still be used to submit transactions to the underlying Blockchain network, but the transactions will be rejected by the deployed business network.

Before you start

Before you follow these steps, you must have added a participant to a participant registry, and issued or bound an identity to that participant. You must also find the unique identifier for that identity in the identity registry. For more information on finding the unique identifiers for identities, look at [Listing all identities in a business network](https://hyperledger.github.io/composer/latest/managing/identity-list.html).

The procedure below shows an example using the following model of a participant from the Digital Property sample Business Network Definition: [digitalproperty-network](https://www.npmjs.com/package/digitalproperty-network)

Copy

namespace net.biz.digitalPropertyNetwork

participant Person identified by personId {

o String personId

o String firstName

o String lastName

}

The example assumes that an instance, net.biz.digitalPropertyNetwork#mae@biznet.org, of that participant has been created and placed into a participant registry.

The example also assumes that an identity maeid1 has been issued to that participant, and the unique identifier for that identity is 'f1c5b9fe136d7f2d31b927e0dcb745499aa039b201f83fe34e243f36e1984862'.

Procedure

1. Connect to the business network and revoke an existing identity from a participant
   * JavaScript API

Copy

const BusinessNetworkConnection = require('composer-client').BusinessNetworkConnection;

async function revoke() {

let businessNetworkConnection = new BusinessNetworkConnection();

try {

await businessNetworkConnection.connect('admin@digitalPropertyNetwork');

await businessNetworkConnection.revokeIdentity('f1c5b9fe136d7f2d31b927e0dcb745499aa039b201f83fe34e243f36e1984862')

await businessNetworkConnection.disconnect();

} catch(error) {

console.log(error);

process.exit(1);

}

}

revoke();

* Command line

Copy

composer identity revoke -c admin@digitalPropertyNetwork -u f1c5b9fe136d7f2d31b927e0dcb745499aa039b201f83fe34e243f36e1984862

# Hyperledger Fabric

There are several cases where information specific to Hyperledger Fabric must be included in Hyperledger Composer commands, including composer network install, composer network start and composer identity issue. The --option, -o option and the --optionsFile, -O option allow connector specific information to be sent.

Multiple options can be specified using the --option, -o by repeating the tag, for example:

Copy

composer somecmd -o thisOpt=value2 -o thatOpt=value2

Alternatively you can create a single file to contain multiple options, for example a file called someCmdOpts.txtcould contain:

Copy

thisOpt=value1

thatOpt=value2

To reference an options file, use the following format:

Copy

composer somecmd --optionsFile=someCmdOpts.txt

Some API's will also include the option to pass a generic options object including AdminConnection.start()and AdminConnection.install()

## Providing npm config settings for install

### **CLI**

The npmrcFile option is available on the composer network install command.

The npmrcFile option allows you to specify npm configuration information when Hyperledger Fabric builds the chaincode image for the Hyperledger Composer runtime.

For example rather than using the default npm registry, you can specify an internal registry within your organization by including the registry option in an options file:

Copy

registry=http://mycompanynpmregistry.com:4873

Supply the fully qualified filename as part of an install command, for example if the file was called npmConfig in your /home/user1/config directory:

Copy

composer network install --c PeerAdmin@hlfv1 --a tutorial-network@0.0.1.bna -o npmrcFile=/home/user1/config/npmConfig

The file contents can be anything that permitted in the .npmrc configuration files of npm.

### **Admin API**

You can supply the name of the file as part of the AdminConnection api on the install method by specifying the npmrcFile property on the installOptions object. For example to pass the name of the npm configuration options file to be provided on install:

Copy

await AdminConnection.install(businessNetworkDefinition, {npmrcFile: '/tmp/npmrc'});

## Hyperledger Fabric Endorsement Policies

You can provide Hyperledger Fabric endorsement policies to both network start and network upgrade requests. The examples that follow show start but the approach is identical to upgrade as well.

### **composer network start/upgrade CLI**

Hyperledger Fabric endorsement policies can be sent using the -o and -O options in several ways.

* Using the -o option, the endorsement policy can be sent either as a single-line JSON string or as a fully qualified file path:

Copy

composer network start ... -o endorsementPolicy='{"identities": [.... }'

Copy

composer network start ... -o endorsementPolicyFile=/path/to/file/endorsementPolicy.json

When a file path is specified, the endorsement policy file should follow this format:

Copy

{"identities":[...],

"policy": {...}}

* Using the -O option, the endorsement policy must be sent as a file path as follows:

Copy

composer network start ... -O /path/to/file/options.json

In this case, the options file should follow this format:

Copy

{"endorsementPolicy": {"Identities": [...].

"policy: {...}"

},

"someOtherOption": "A Value"

}

For more information on writing Hyperledger Fabric endorsement policies, see the [Hyperledger Fabric Node.js SDK documentation](https://fabric-sdk-node.github.io/global.html" \l "ChaincodeInstantiateUpgradeRequest) which provides examples of endorsement policies.

### **Admin API**

To send an endorsement policy via the Admin API, the endorsement policy file must be included as part of the startOptions or deployOptions objects when calling start or deploy respectively. To pass an endorsement policy file it must be specified in the object property endorsementPolicyFile. To supply the policy as a JSON object, the endorsementPolicy object property must be specified.

Copy

await adminConnection.start('tutorial-network', '0.0.1', { networkAdmins: networkAdmins, endorsementPolicyFile: 'endorsement-policy.json'} );

## Identity Issue

When a new identity is issued, you may want to specify whether the issued identity has the authority to register new identities with a Hyperledger Fabric certificate authority server.

### **CLI**

To grant an identity the authority to register new identities with a certificate authority from the command line, the -x option is available (which is a shortcut replacement for -o issuer=true).

Copy

composer identity issue -c admin@digitalproperty-network -u MyUser -a net.biz.digitalPropertyNetwork.Person#P1 -x

## API

To specify the issuer property you set it in an object and pass this object as part of the issueOptions on issueIdentity. For example to issue an identity that has issuer authority

Copy

await businessNetworkConnection.issueIdentity(participantId, newUserId, {issuer: true});