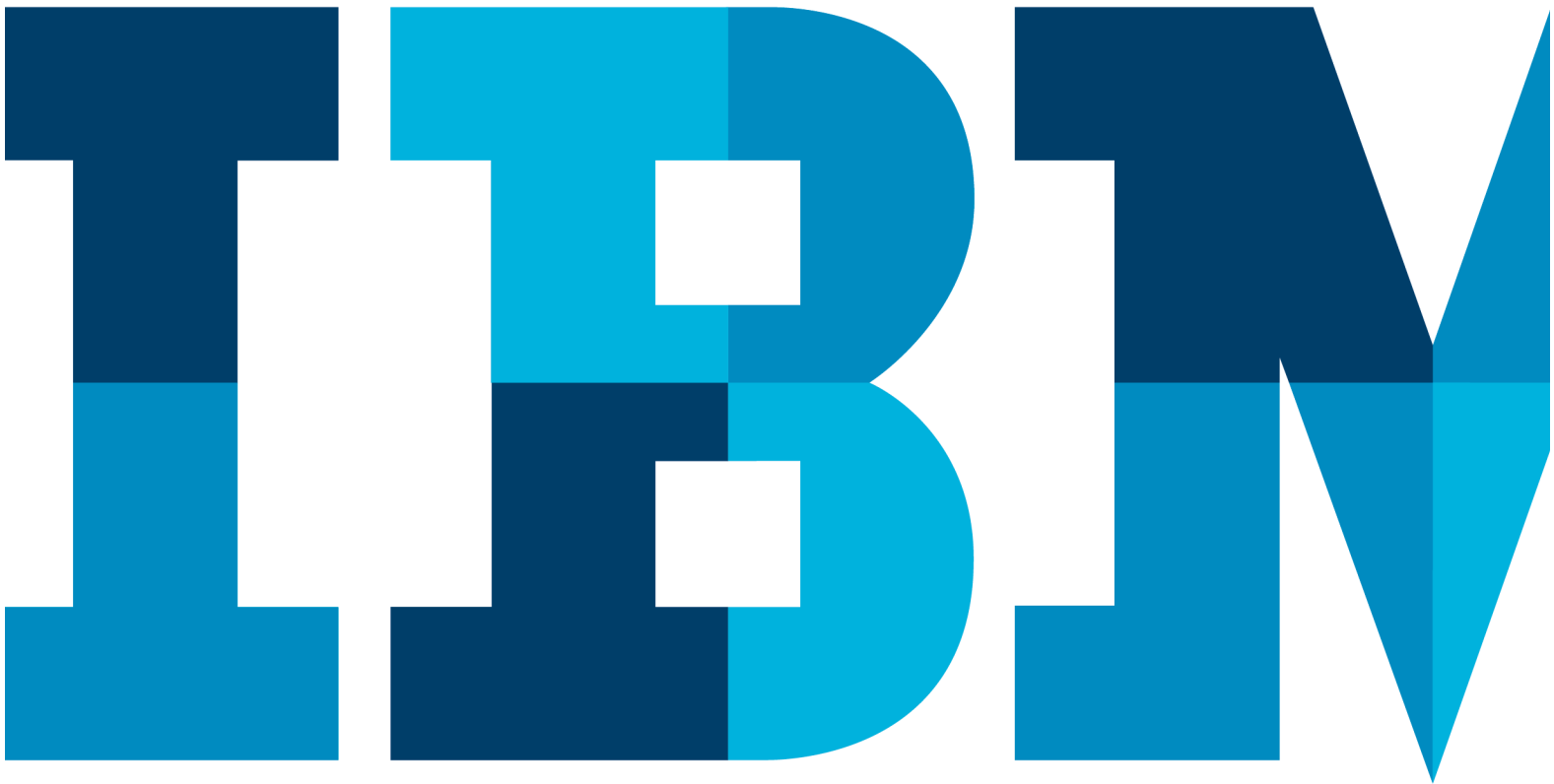


IBM Blockchain Hands-On Introduction to Hyperledger Composer

Lab One



Contents

SECTION 1.	STANDING UP THE NETWORK	4
1.1.	RUNNING THE SCRIPT	4
SECTION 2.	COMPOSER'S MODELLING LANGUAGE	5
2.1.	SYNTAX.....	5
2.2.	WRITING A TRANSACTION.....	10
2.3.	DEPLOY AND CREATE SOME TEST ASSETS.....	10
SECTION 3.	TRANSACTION LOGIC	15
3.1.	CREATE THE LOGIC FILE.....	15
3.2.	ADD THE CHANGEOWNER FUNCTION	15
3.3.	ADD CHANGOWNER'S BODY	16
3.4.	TEST CHANGEOWNER.....	17
SECTION 4.	ACCESS CONTROL & IDENTITY	20
3.1.	ACCESS CONTROL LISTS	20
NOTICES	24	
APPENDIX A.	TRADEMARKS AND COPYRIGHTS.....	26

Overview

The aim of this lab is to get you familiar with developing Hyperledger Composer business networks. We will do this by exploring the Composer modelling language, how to write transaction processor functions in JavaScript and lastly examine how Access Control is managed in Composer.

The lab will also familiarise you with the Composer Playground, a web based tool that allows for rapid development and testing of Composer business networks.

It should be noted that while the contents of this lab will predominantly occur within Composer Playground (for the sake of accelerating the learning and development process), the Lab can easily be completed offline and using a text editor such as Visual Studio Code or Atom. In this case please refer to the next Lab for instructions on how to use the command line tools available in Composer.

Introduction

Pre-requisites:

- 4 cores
- 4GB RAM
- VMWare V10+
- The lab virtual machine

The virtual machine is based on Linux Ubuntu 16.04 and contains Hyperledger Fabric V1.0, Golang, Git, Visual Studio Code and Firefox.

A network needs to be visible to the virtual machine (even if the network is just to the host environment). If you do not see the up/down arrows in the status bar at the top of the screen, or if you receive errors about no network being available, please tell the lab leader. The virtual machine might need to be reconfigured in NAT mode.

There are no additional files or software that is proprietary to the lab in the virtual machine. This means that the lab may be run on a machine without the without a lab virtual machine if Hyperledger Fabric and the other pre-requisites have been installed.

It is recommended that students have previously completed the Blockchain Explained and Blockchain Explored labs.

Section 1. Standing up the network

This section will cover standing up the network on the VM provided. As the focus of the Lab is on learning Composer development, scripts have been written to stand up the network for you. Please feel free to inspect these scripts to understand how they work.

1.1. Running the script

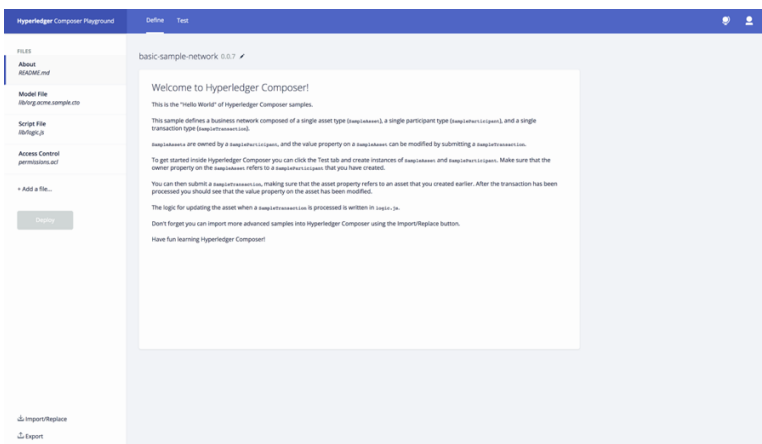
Navigate to `~/workspace/composer-playground` and issue the following command:

```
./start-playground.sh
```

This will stand up a basic network:

```
dGgQIBCCWvbV/\nTvvcBgGiaYmethH4qc/u3KK4U1H0Nwvt13epx3jAKBggqhkjOPQQDAAGNHADBEAIAe\|n1/wXZht2G6KVxfS  
LPdaOeoBWZG0/TQN1KxTH7/QwIgtMDJowxbq2EziHNJA/U0|\n3+aFesZjYUGvuvOA0ijYfGa=\n-----END CERTIFICATE-  
-----\|n|x1a|x06Adms|\\"1\\n|x06Adms|x12'|x12|x1d\b|x01|x19|x12\b|x12|x06\b|x01|x12|x02\b|x00|  
x1a|r|x12|\n|x1a|oarg2MSP|x10|x01|x1a|x06Adms|\\"0\\n|aReaders|x12%|x12|x1b\b|x01|x12|x17|x12\b|x12|x  
06\b|x01|x12|x02\b|x00|x1a|\v|x12|\t|\n|x1a|oarg2MSP|x1a|x06Adms|\\"0\\n|awriters|x12%|x12|x1b\b|x01|x12|  
x17|x12\b|x12|x06\b|x01|x12|x02\b|x00|x1a|\v|x12|\t|\n|x1a|oarg2MSP|x1a|x06Adms|\\"1\\n|awriters|x12|x17|  
|x12|\r\b|x03|x12|\t|\n|awriters|x1a|x06Adms|\\"||\"|\n|x06Adms|x12|x18|x12|x0e\b|x03|x12|\n|\n|x06Admi  
ns|x10|x02|x1a|x06Adms|\\"i\\\"|\n|aReaders|x12|x17|x12|\r\b|x03|x12|\t|\n|aReaders|x1a|x06Adms|x1a-\n|\n|x19BlockDataHashingStructure|x12|x10|x12|x06\b|xff\xff\xff|x0f|x1a|x06Adms|x1a-\n|\n|x10order  
erAddresses|x12|x19|x12|x0f|\n|rorderer0:7050|x1a|x06Adms|x1a&\n|\n|x10HashingAlgorithm|x12|x12|x12|  
|\b|\n|x06SHA256|x1a|x06Adms|\\"l\\\"|\n|aReaders|x12|x17|x12|\r\b|x03|x12|\t|\n|aReaders|x1a|x06Adms|\\"|"  
|\n|n|aWriters|x12|x17|x12|\r\b|x03|x12|\t|\n|aWriters|x1a|x06Adms|\\"|\"|\n|x06Adms|x12|x18|x12|x0e\b|  
|x03|x12|\n|\n|x06Adms|x10|x02|x1a|x06Adms|\\"%\\n|x0fAcceptAllPolicy|x12|x12|x12|\b|\b|x01|x12|x04|  
x12|x02|x12|x00|x1a|x06Adms|x12-\n|\n|x104cert|x12|x18|x03|x06\x0d\xf2\xf2Jn\xd9|id|1MDa|x9a|xf1!|  
x8c|xdc|xec|x9a|xfec|x12|signature|x12|signature\" 2:\"0D|x02 T|x89|x07|xe4|x09|xc2|xf1?S|xaf=Z|x  
a4|xb0|x9e|xfbJAK).\x02|x005\xa3|x11|x86|xc1|xc9|x02 \v|x0e|xa5|xac|x83nt|C|xa6|x8aiV|xb6w|xaOXPO  
|xf9|x91|u0600m?|xd2|xfaw|xdd|\t\" >metadata:<metadata:\" \" metadata:\" \" metadata:\" \" metadata:\" \">  
2017-05-19 12:43:47.644 UTC [main] main -> INFO 001 Exiting.....
```

It should also launch firefox and open the playground web ui, if it doesn't open firefox and go to <http://localhost:8080>:



Section 2. Composer's Modelling Language

In this section you will learn about the modelling language Hyperledger Composer uses to define resources in its business networks. You will define a basic business network that allows participants to exchange marbles with each other, using this as a base from which to explore the language's features.

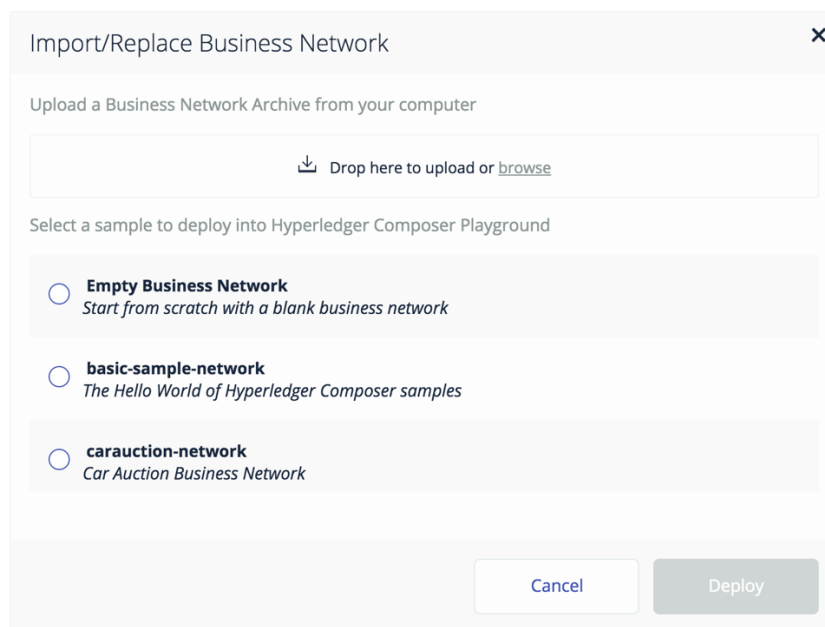
2.1. Syntax

a. Reset the playground

Select **Import/Replace** from the bottom-left corner of the **Define** tab:



Select '**Empty Business Network**' from the list:



b. Name the business network

Click the text saying **unnamed-network** at the top:

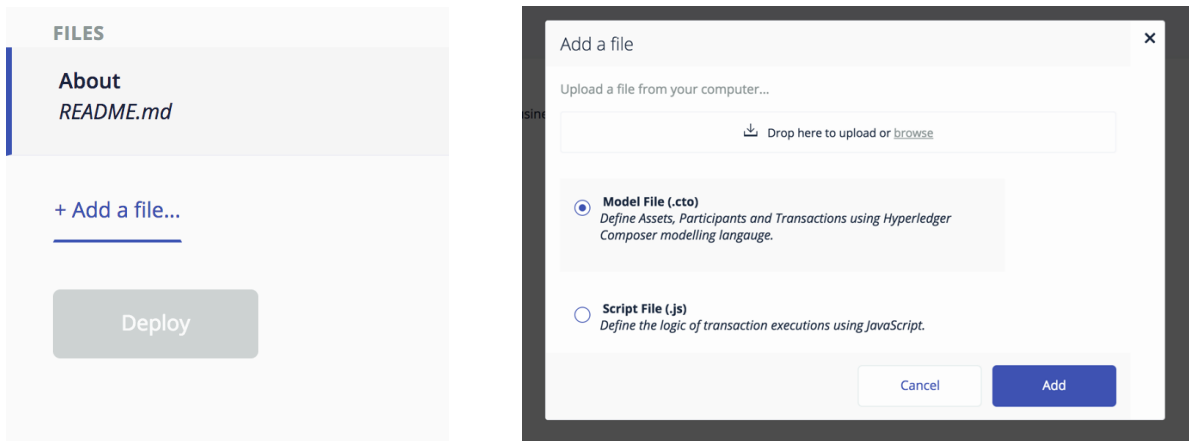
unnamed-network 0.0.1

Rename the network to something of your choosing.

Name	basic-sample-network	Version	0.0.7	View/edit full metadata in package.json
-------------	----------------------	----------------	-------	---

c. Add a model file

Select the **+Add a file...** button on the top left and from the list select



d. Resources

Composer's modelling language is first and foremost oriented around high level business concepts. As such, the **three top-level resources** that can be defined are as follows:

Assets

Digital representations of assets that are recorded on the ledger.

Participants

Individuals and Organisations that will contribute to and make use of the ledger.

Transactions

Business logic governing the manipulation of assets.

Additionally, each resource belongs to a namespace (a default namespace is at the top of the newly created file) which acts in a similar manner to how namespaces and packages work in other languages. In much the same way, resources can be imported from other namespaces. Namespace names can be any combination of letters and periods.

The modelling language describes these resources in a similar manner that you would describe a class in another language, this being an entity with attributes.

e. Writing an asset

Underneath namespace `org.acme.model` write the following:

```
asset Marble identified by Id {
  o String Id
}
```

This defines a Marble asset and gives it an identifier to be referred to by (similar to the keys used in Fabric). Let's flesh this out to include the same attributes as in our Fabric Marble:

```
asset Marble identified by Id {
  o String Id
  --> Collector owner
  o Integer diameter
  o String colour
}
```

You'll have noticed that the attributes in this do not all have the same prefix. The owner attribute is preceded by a --> instead of a o.

The o attributes are 'named fields' – they belong to the resource, for example the Marble will have a size and colour property.

The --> attributes are 'relationships' – while they make up the information that describes the resource they are not part of it, for example a Marble will have an owner but the owner is not part of the Marble.

The currently supported attribute types are as follows:

String, Boolean, DateTime, Integer, Double, Long

f. Writing a participant

With this in mind, we shall define an owner. Add the following below the Marble definition:

```
participant Collector identified by email {
  o String email
}
```

Attributes for participants work in an identical manner to those of Marbles. As such, expand the Collector class:

```
participant Collector identified by email {
  o String email
  o String firstname
  o String surname
  o Address address
  o Sex sex
}
```

You will notice that there are two fields that are not included in the base types listed in **d**. Address and Sex are, respectively, a **Concept** and an **Enumeration**.

g. Concepts

When modelling data, the base types will only go so far. Some pieces of data require a more complex definition yet cannot be considered resources in their own right.

An address is a perfect example of this, it requires a more complex structure than a single field yet it is not a resource in it's own right – it is simply part of one. Add the following to your model file:

```
concept Address {
  o String house
  o String street
  o String county
  o String postcode
  o String country
}
```

h. Enumerations

Enumerations in Composer function identically to that in other languages, add the following to your model file:

```
enum Sex {
  o MALE
  o FEMALE
}
```

i. Abstract Resources and Inheritance

n.b. For this section, do not actually modify your code.

Like other OOP languages, Composer supports abstract types and inheritance. Say we wanted to add a new shape of Marble – a PolyhedronMarble:

```
asset PolyhedronMarble identified by Id {
  o String Id
  --> Collector owner
  o String colour
  o Integer faces
}
```

This new type of Marble shares many characteristics with Marble, as such it would make sense to not repeat these common attributes (particularly if we add more types). As such we can define an **Abstract** Marble class and have our two types of Marble extend it:

```
abstract asset Marble identified by Id {
  o String Id
  --> Collector owner
  o String colour
}
```

```
asset SphericalMarble extends Marble {
  o Integer diameter
}
```

```
asset PolyherdalMarble extends Marble {
  o Integer faces
  o Integer arcLength
}
```


Now code is not pointlessly repeated. You will also notice that because the identifying attribute is defined in the abstract class, it is not necessary to do so in the child classes (although doing so would simply overwrite it). Identifiers are optional in abstract classes.

j. Validators

Some attributes will come in a specific format and indeed it is often the case that code will be written expecting them in this format for example emails are always of the form <name>@<domain>.<suffix>. Likewise limits may be placed on data or default values may be enforced. The following validators are supported (add them to your resources):

Regex

```
participant Collector identified by id {
  o String email regex=/[a-z0-9.-]+@[a-z0-9.]+/
  ...
}
```

Regex or Regular Expressions are a method of specifying a query for searching strings. In this situation the regex pattern can be used to enforce rules about string form, rejecting anything that does not match the pattern. It should be noted that at the time of writing regex expressions should not be applied to the identifiers of resources as it will throw an error when an initial ID is auto generated:

```
Error: Invalid validator for field org.acme.sample.SampleParticipant#email:3747
org.acme.sample.SampleParticipant.email: Value + 'email:3747' failed to match validation
regex: /[a-z0-9.-]+@[a-z0-9.]+/
```

Optional

```
concept Address {
  ...
  o String county optional
  ...
}
```

As the name suggests, this validator denotes that the field is not required by the model.

Along with default, this is a way of extending models after initial deployment. If new fields are added to a resource, errors that would otherwise be thrown from existing resources becoming non-compliant with the spec can be resolved by making the new fields optional so that the existing resources can be updated as needed.

Default

```
asset Marble identified by Id {
  ...
  o String colour default="red"
}
```

Default sets an initial value for an attribute that can be overridden later.

Along with optional, this is a way of extending models after initial deployment. If new fields are added to a resource, default values can be set so that they remain compliant with their required fields. This can also be used to set placeholder values that mark fields out for needing to be filled in with real data

Range

```
asset Marble {
    ...
    o Integer diameter range=[1,20]
    ...
}
```

Range enforces a inclusive limit on numerical values, in the above this requires the diameter to be between 1 and 20.

2.2. Writing a transaction

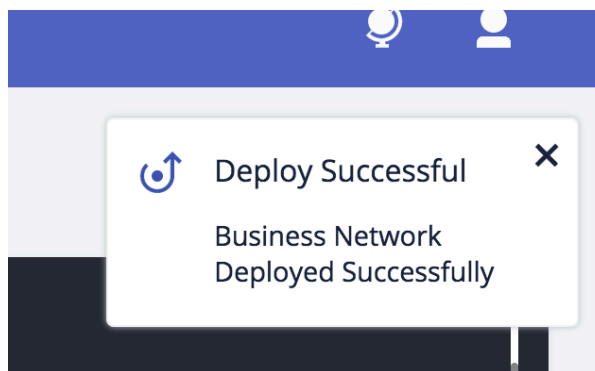
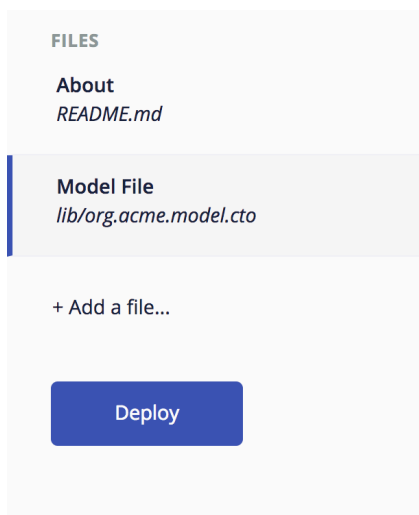
Transactions are also declared in the modelling file using the same syntax as with Assets and Participants, add the following to your modelling file:

```
transaction ChangeOwner identified by txId {
    o String txId
    --> Marble marble
    --> Collector newOwner
}
```

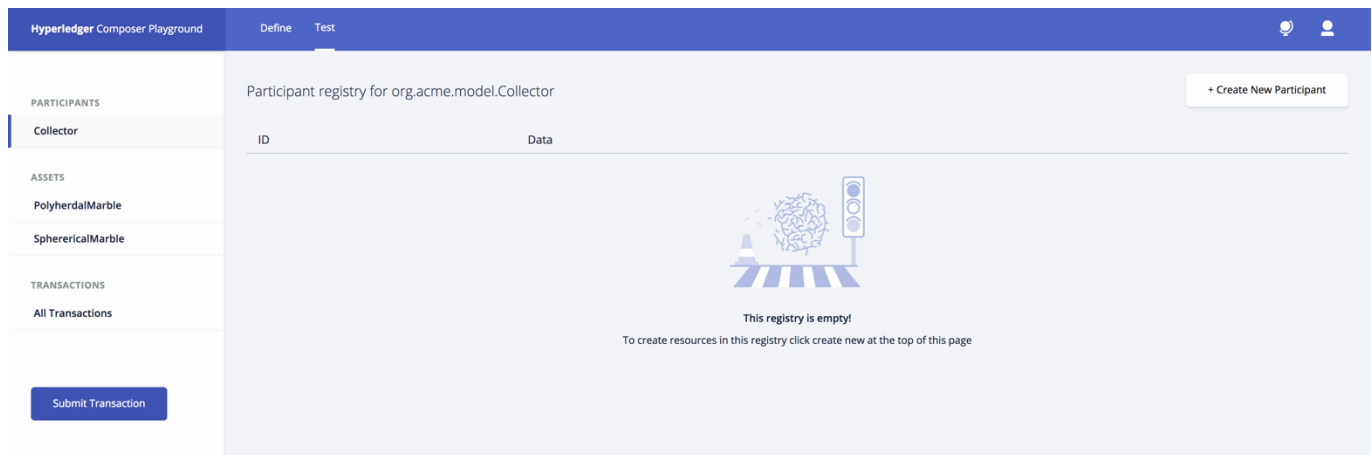
Instead of denoting attributes, the variables within the body of a transaction denote the arguments that the transaction logic function will take (this will be covered in more detail in the next section).

2.3. Deploy and create some test assets

a. Deploy the network



Now we have some asset definitions, hit the deploy button on the left side of the screen. On success a small pop-up should appear. Go to the **Test** tab at the top:



Here you can see the assets and participants we've made.

b. Create the assets and participants

Hit **+ Create New Participant** on the top right. A dialogue box will appear prompting you to enter details of the new participant:

Create New Participant

In registry: **org.acme.model.Collector**

JSON Data Preview

```

1  {
2    "$class": "org.acme.model.Collector",
3    "email": "email:8117",
4    "firstname": "",
5    "surname": "",
6    "address": {
7      "$class": "org.acme.model.Address",
8      "house": "",
9      "street": "",
10     "county": "",
11     "postcode": "",
12     "country": ""
13   },
14   "sex": "MALE"
15 }

```

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

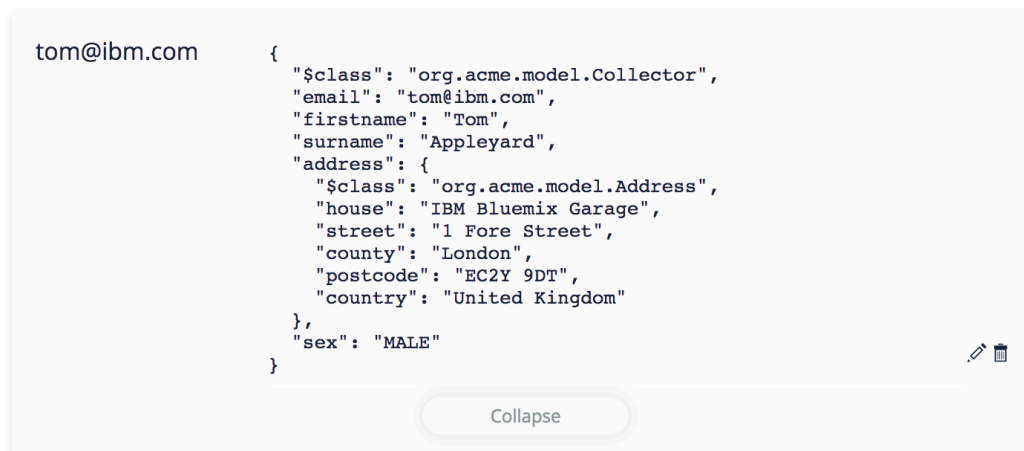
Cancel

Create New

Enter the following:

```
{
  "$class": "org.acme.model.Collector",
  "email": "tom@ibm.com",
  "firstname": "Tom",
  "surname": "Appleyard",
  "address": {
    "$class": "org.acme.model.Address",
    "house": "IBM Bluemix Garage",
    "street": "1 Fore Street",
    "county": "London",
    "postcode": "EC2Y 9DT",
    "country": "United Kingdom"
  },
  "sex": "MALE"
}
```

Fill this in and select **Create New**, you will see the new participant appear:



Create a second Collector with the following:

```
{
  "$class": "org.acme.model.Collector",
  "email": "mgk@ibm.com",
  "firstname": "Matthew",
  "surname": "Golby-Kirk",
  "address": {
    "$class": "org.acme.model.Address",
    "house": "IBM Hursley",
    "street": "Hursley Park",
    "county": "Hampshire",
    "postcode": "SO21 2JN",
    "country": "United Kingdom"
  },
  "sex": "MALE"
}
```

If you swap to the Marble asset and select **+ Create New Asset** you will see a similar dialogue box:

Create New Asset

In registry: **org.acme.model.Marble**

JSON Data Preview

```

1  {
2    "$class": "org.acme.model.Marble",
3    "Id": "Id:0650",
4    "owner": "resource:org.acme.model.Collector#email:0812",
5    "colour": "red",
6    "diameter": 0
7  }

```

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

Cancel

Create New

Enter the following. Note that when **filling out relationships**, you must supply a fully qualified **identifier** – this being as follows:

resource:<namespace>.<resource name>#identifier

```

{
  "$class": "org.acme.model.Marble",
  "Id": "MARBLE001",
  "owner": "resource:org.acme.model.Collector#tom@ibm.com",
  "colour": "red",
  "diameter": 20
}

```

You will see the new participant appear.

MARBLE001

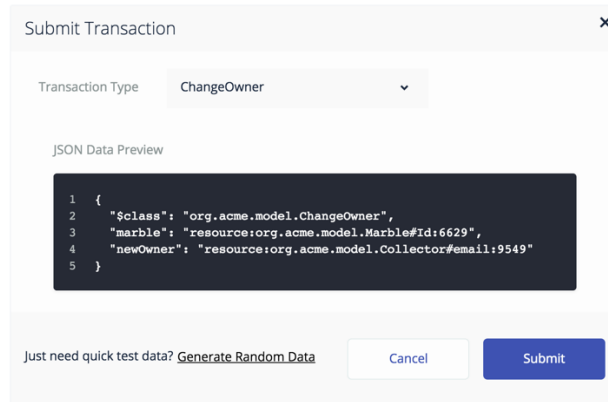
```

{
  "$class": "org.acme.model.Marble",
  "Id": "MARBLE001",
  "owner": "resource:org.acme.model.Collector#tom@ibm.com",
  "colour": "red",
  "diameter": 20
}

```

Collapse

If you select **Submit Transaction** at the bottom left you will see a similar dialogue box:



Submit Transaction

Transaction Type: **ChangeOwner**

JSON Data Preview

```
1 {  
2   "$class": "org.acme.model.ChangeOwner",  
3   "marble": "resource:org.acme.model.Marble#id:6629",  
4   "newOwner": "resource:org.acme.model.Collector#email:9549"  
5 }
```

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

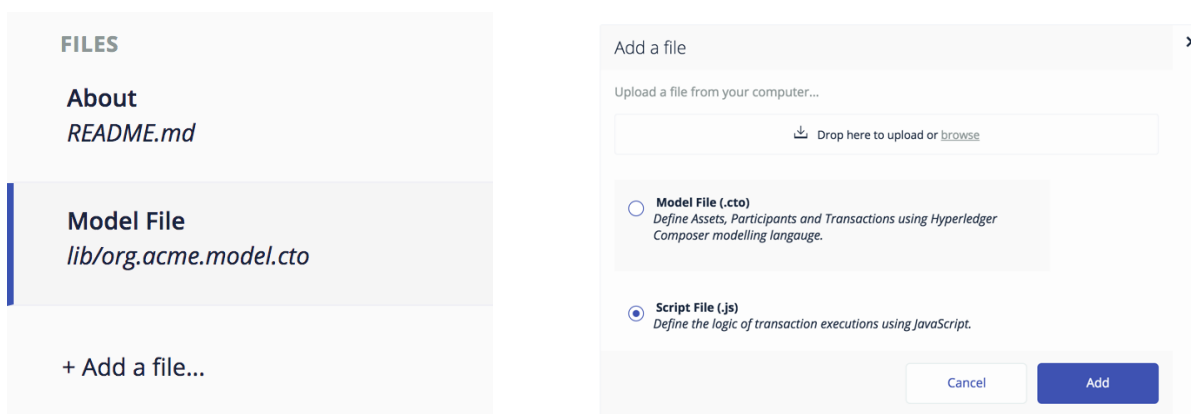
However if you submit this transaction, nothing will happen. We need to define some logic to associate with it. Before moving on to the next section, try creating another participant and an asset.

Section 3. Transaction Logic

In this section we will explore how to write transaction processor functions, these being the business logic that is executed when a transaction is invoked in Hyperledger Composer. Please note, while transaction processor functions are analogous to chaincode in their purpose we are not writing chaincode in this lab. Composer transaction logic, while achieving the same results, is not handled in the same way as chaincode is.

3.1. Create the logic file

Select **+ Add a file...** from the left hand side and select **Script File (.js)** from the dialogue:



3.2. Add the changeOwner function

Within the new file add the following:

```
/**
 * @param {org.acme.model.ChangeOwner} args - the changeOwner transaction arguments
 * @transaction
 */
function changeOwner(args) {

}
```

Transaction processor functions are defined by writing a function with a JS Doc decorator that maps the first argument to the transaction's model definition. The args argument represents the incoming transaction, in particular the data packaged in it.

Recall the transaction's definition:

```
transaction ChangeOwner identified by txId {
  o String txId
  --> Marble marble
  --> Collector newOwner
}
```

Args is an object where the keys are each of the attributes and the values are what has been attached to them during the transaction invocation.

args will have a marble and a newOwner attribute that are accessed the same way attributes are accessed in JS objects: args.marble or args.newOwner.

Transaction processor functions do not return anything, much like Invoke functions in Fabric, they simply execute and finish.

3.3. Add changOwner's body

To change the owner of a marble, simply assign the newOwner argument that was passed in to the marble argument's owner attribute. Add the following to the body of changeOwner:

```
args.marble.owner = args.newOwner;
```

Now the update has been made to the asset, we need to update the assets record in the world state. Add the following:

```
return getAssetRegistry('org.acme.model.Marble').then(function(marbleRegistry) {  
    return marbleRegistry.update(args.marble);  
});
```

Registries are indexes used by composer to store resources, they store a reference to every instance of that particular resources. To update an resources we get the registry for it's type and call the update function with the new version of the resources we want to update (composer will find it within the registry and update it for us).

Participants also have registries and are updated in the same way (although with getParticipantRegistry).

It should be noted that while the language composer uses for it's transaction process functions is JavaScript it only supports up to ES5, as such features like `() => {}` functions are not permitted. This is due to the Otto JavaScript engine that is currently used by composer. Otto is set to be replaced by an embedded version of node.js in a future release.

3.4. Test changeOwner

a. Create the assets and participants

Deploy the code and go to the Test tab. We are going to transfer a Marble between two Collectors. If you don't have 2 Collectors or a Marble follow the steps in 2.3 to create them:

Asset

```
MARBLE001      {
                  "$class": "org.acme.model.Marble",
                  "Id": "MARBLE001",
                  "owner": "resource:org.acme.model.Collector#tom@ibm.com",
                  "colour": "red",
                  "diameter": 20
                }
```



Collapse

Participants

```
tom@ibm.com     {
                  "$class": "org.acme.model.Collector",
                  "email": "tom@ibm.com",
                  "firstname": "Tom",
                  "surname": "Appleyard",
                  "address": {
                    "$class": "org.acme.model.Address",
                    "house": "IBM Bluemix Garage",
                    "street": "1 Fore Street",
                    "county": "London",
                    "postcode": "EC2Y 9DT",
                    "country": "United Kingdom"
                  },
                  "sex": "MALE"
                }
```



Collapse

```
matt@ibm.com    {
                  "$class": "org.acme.model.Collector",
                  "email": "matt@ibm.com",
                  "firstname": "Matt",
                  "surname": "Golby-Kirk",
                  "address": {
                    "$class": "org.acme.model.Address",
                    "house": "IBM Hursley",
                    "street": "Hursley Park",
                    "county": "Hampshire",
                    "postcode": "SO21 2JN",
                    "country": "United Kingdom"
                  },
                  "sex": "MALE"
                }
```



Collapse

b. Submit the transaction

Select **Submit Transaction** from the sidebar and fill in the fields accordingly to select your marble and the Collector who is not the owner:

```
{
  "$class": "org.acme.model.ChangeOwner",
  "marble": "resource:org.acme.model.Marble#MARBLE001",
  "newOwner": "resource:org.acme.model.Collector#mgk@ibm.com"
}
```

Select **Submit** to issue the transaction.

PARTICIPANTS

Collector

ASSETS

Marble

TRANSACTIONS

All Transactions

Submit Transaction

Submit Transaction

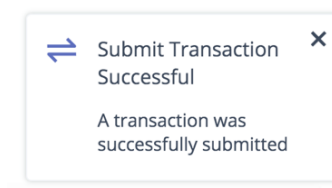
Transaction Type: ChangeOwner

JSON Data Preview

```
1 {
2   "$class": "org.acme.model.ChangeOwner",
3   "marble": "resource:org.acme.model.Marble#MARBLE001",
4   "newOwner":
5     "resource:org.acme.model.Collector#matt@ibm.com"
6 }
```

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

If successful the following dialogue will appear:



A transaction entry will also appear:

```
df577d57-f724-4aae-ab08-7d0f51474916 {
  "$class": "org.acme.model.ChangeOwner",
  "txId": "df577d57-f724-4aae-ab08-7d0f51474916",
  "marble": "resource:org.acme.model.Marble#MARBLE001",
  "newOwner": "resource:org.acme.model.Collector#matt@ibm.com",
  "timestamp": "2017-05-19T14:08:55.009Z"
}
```

Collapse

If you go back to the Marble, you will find its record has also been updated:

```
MARBLE001      {  
                  "$class": "org.acme.model.Marble",  
                  "Id": "MARBLE001",  
                  "owner": "resource:org.acme.model.Collector#matt@ibm.com",  
                  "colour": "red",  
                  "diameter": 20  
                }
```



Section 4. Access Control

In this section we will explore how Hyperledger Composer restricts access to the resources on the network and the ability to modify them. Unlike previous sections, ACLs are not currently supported in Composer Playground and as such this section will cover how to export your business network to edit it locally. This section will also not feature a demonstration of ACLs in practice – this will be covered in the next Lab when identities are covered (ACLs will be revisited then, looking at their effects).

3.1. Access Control Lists

We now have some digital assets defined and the ability to move them between users. However in a real system it would likely be the case that the Marb1e objects would not be available for all to see and if they were they would not be available for just anyone to change the ownership of.

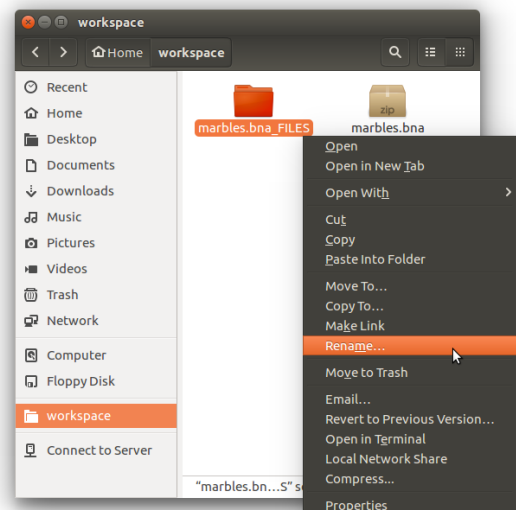
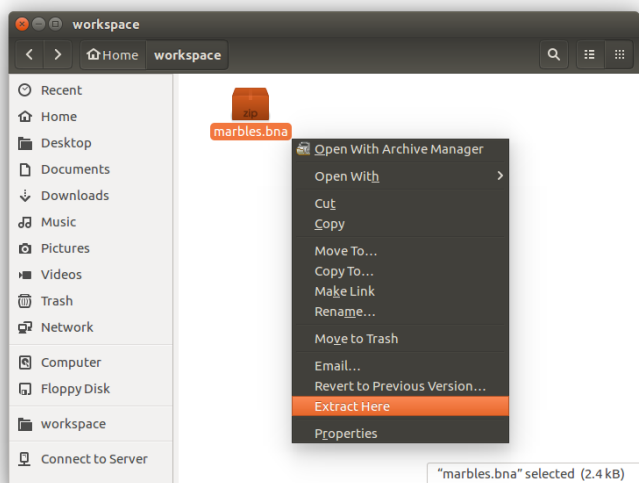
a. Export the Business Network

Go to the bottom-left of the screen and select **Export**:

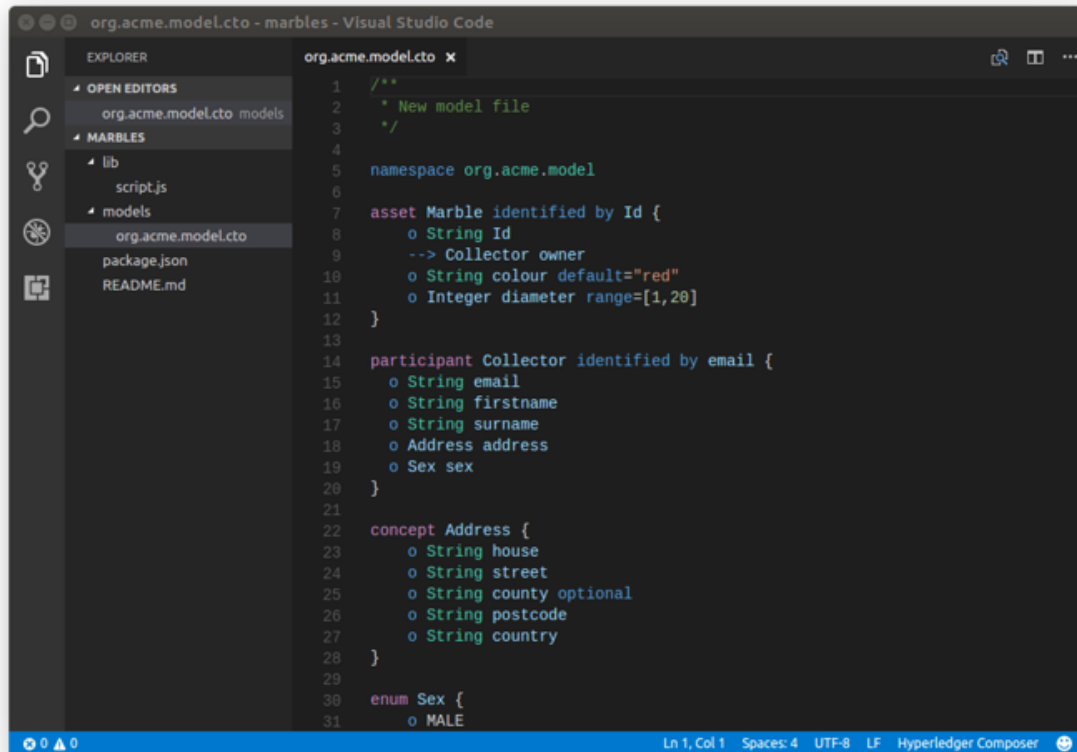
↓ Import/Replace

↑ **Export**

Rename the extension of the file to .zip and extract it in your workspace. Rename the extracted directory *marbles*:

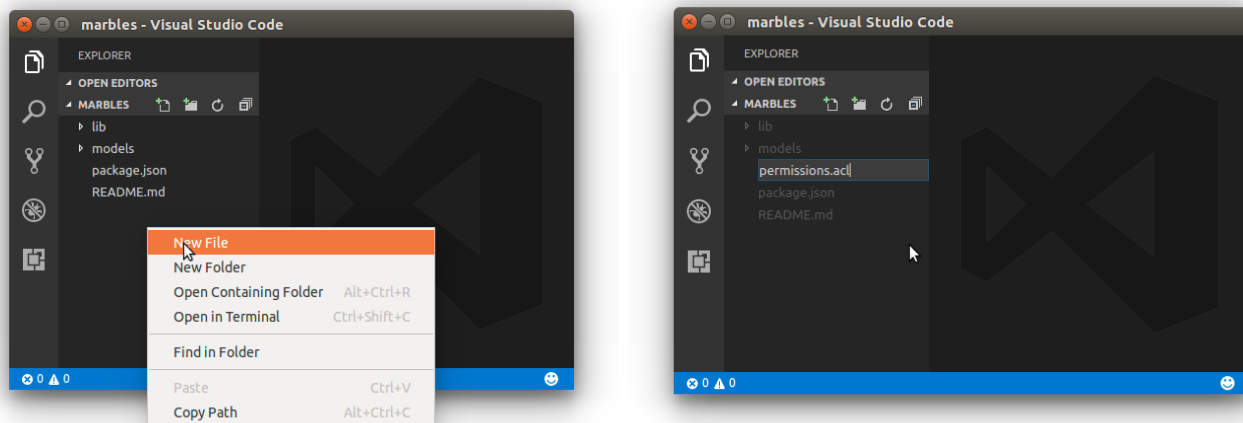


Open it in Visual Studio code:



b. Create the ACL file

Create a new file in the marbles directory called `permissions.acl`:



ACLs in Composer are files containing sets of rules dictating what actions participants are permitted to perform on certain resources and the conditions under which they can occur.

c. ACL Grammar

ACL Rules are of the following format:

```
rule <Rule Name> {
  description: <description of the rule>
  participant(p): <namespace and name of the participant performing the action>
  operation: <operation the participant wishes to perform>
  resource(r): <resources the operation is being performed on>
  condition: (<condition under which this rule applies>)
  action: <does this rule allow an operation or deny it>
}
```

In more detail:

Participant is the person or entity that has submitted the transaction.

Operation is what they wish to do to this resource, supported operations are CREATE, READ, UPDATE, DELETE, ALL

Resource is the asset that the transaction is being applied to. Resources (and indeed Participants) can simply be namespaces in which case they apply to all participants/resources in that namespace.

Condition is a JavaScript statement that can examine the participant and resource to check for certain conditions. Anything valid for use in an if statement is valid here.

Action is a simple ALLOW or DENY, as the name suggests this allows or denies the transaction.

d. Adding rules

Add the following:

```
rule OnlyOwnerCanUpdateAMarble {
  description: "Only an owner can edit a marble"
  participant(p): "org.acme.model.Collector"
  operation: UPDATE
  resource(r): "org.acme.model.Marble"
  condition: (r.owner.getIdentifier() == p.getIdentifier())
  action: ALLOW
}
```

This rule ensures that only the owners of Marble resources are able to edit them. It does this by ALLOWing and UPDATE to org.acme.model.Marble resources only when the identifier of the participant and the resource's owner are the same.

By default all action is restricted unless explicitly permitted. As such while we do have a rule allowing updates of a Marble resource even the owner would be unable to read it. Add the following:

```
rule AnyoneCanReadMarbles {
  description: "Only an owner can edit a marble"
  participant(p): "org.acme.model.Collector "
  operation: READ
  resource(r): "org.acme.model.Marble"
  condition: (true)
  action: ALLOW
}
```

This rule allows all Collector participants to READ all Marbles. While we're at it we'll also need a rule to let Collectors read each other. The ChangeOwner transaction requires a submission of the identifier of a new owner which will not be possible if Collectors cannot read each other:

```
rule AnyoneCanReadCollectors {
  description: "Only an owner can edit a marble"
  participant(p): "org.acme.model.Collector "
  operation: READ
  resource(r): "org.acme.model.Collector"
  condition: (true)
  action: ALLOW
}
```

Rules apply to transactions too.

In addition to the first rule we wrote a second rule will need to be written to allow participants access to the tools needed to make the changes. In particular we need to allow participants to create change owner transactions:

```
rule AnyoneCanIssueChangeOwner {
  description: "Only an owner can edit a marble"
  participant(p): "org.acme.model.Collector"
  operation: CREATE
  resource(r): "org.acme.model.ChangeOwner"
  condition: (true)
  action: ALLOW
}
```

Even with this rule, the transaction could be created by a non-owner but would still be rejected as they lack update access.

Notices

This information was developed for products and services offered in the U.S.A.
IBM may not offer the products, services, or features discussed in this document in other countries.

Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service. IBM may have patents or pending patent applications covering subject MGKer described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM

products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental. All references to fictitious companies or individuals are used for illustration purposes only.

COPYRIGHT LICENSE:

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

Appendix A. Trademarks and copyrights

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM	AIX	CICS	ClearCase	ClearQuest	Cloudscape
Cube Views	DB2	developerWorks	DRDA	IMS	IMS/ESA
Informix	Lotus	Lotus Workflow	MQSeries	OmniFind	
Rational	Redbooks	Red Brick	RequisitePro	System i	
<i>System z</i>	<i>Tivoli</i>	<i>WebSphere</i>	<i>Workplace</i>	<i>System p</i>	

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

ITIL is a registered trademark, and a registered community trademark of The Minister for the Cabinet Office, and is registered in the U.S. Patent and Trademark Office.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

Linear Tape-Open, LTO, the LTO Logo, Ultrium, and the Ultrium logo are trademarks of HP, IBM Corp. and Quantum in the U.S. and other countries.

NOTES

[illegible]

NOTES



© Copyright IBM Corporation 2016.

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

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



Please Recycle
