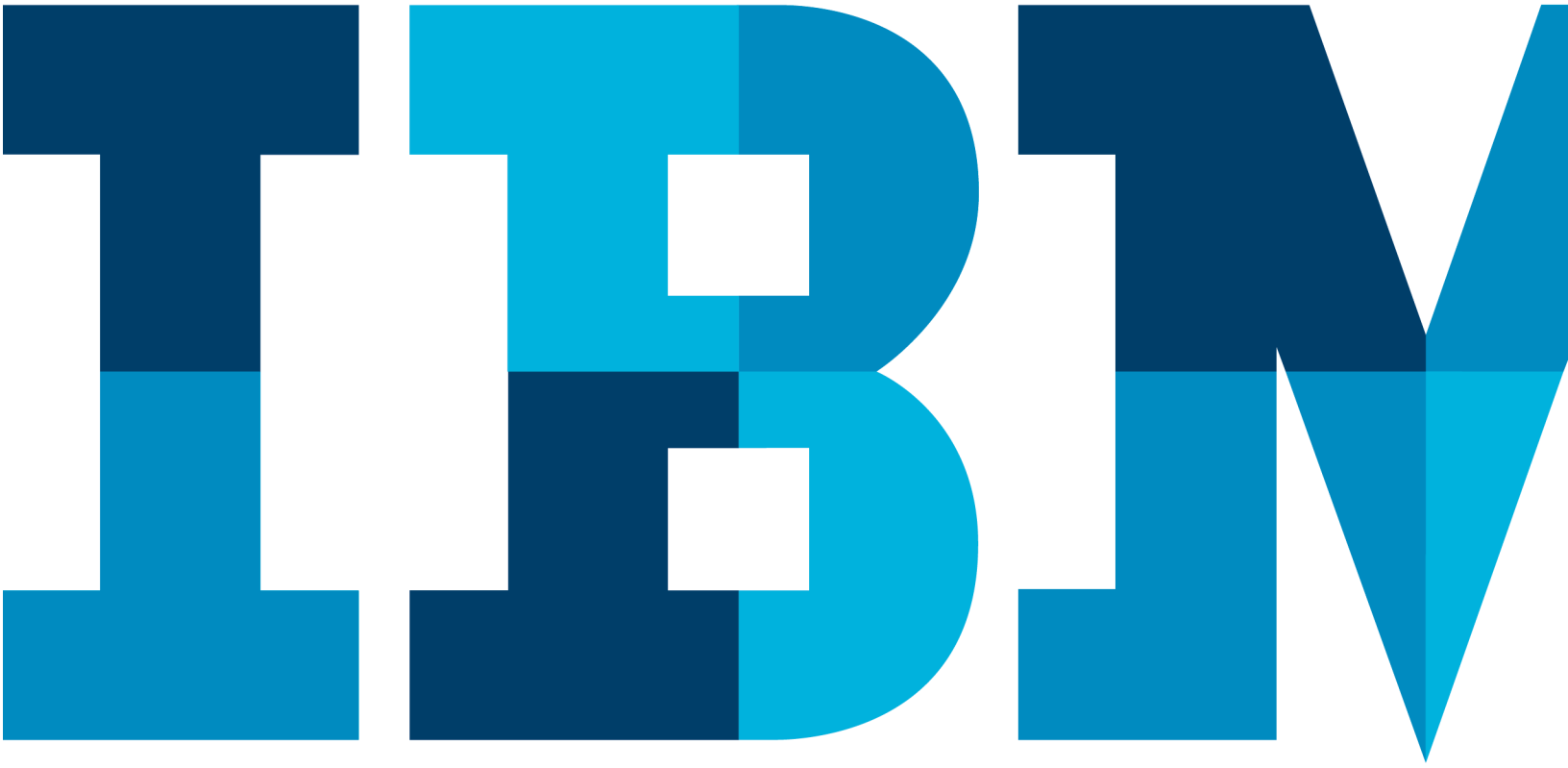
IBM Blockchain Hands-On

IBM Blockchain Platform Visual Studio Code Extension:

Import Commercial Paper Sample

Lab Five



IBM

Table of Contents

Disclaimer

IBM’s statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM’s sole discretion. Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.

The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract.

The development, release, and timing of any future features or functionality described for our products remains at our sole discretion I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results like those stated here.

Information in these presentations (including information relating to products that have not yet been announced by IBM) has been reviewed for accuracy as of the date of initial publication and could include unintentional technical or typographical errors. IBM shall have no responsibility to update this information. **This document is distributed “as is” without any warranty, either express or implied. In no event, shall IBM be liable for any damage arising from the use of this information, including but not limited to, loss of data, business interruption, loss of profit or loss of opportunity.** IBM products and services are warranted per the terms and conditions of the agreements under which they are provided.

IBM products are manufactured from new parts or new and used parts.
In some cases, a product may not be new and may have been previously installed. Regardless, our warranty terms apply.”

Any statements regarding IBM's future direction, intent or product plans are subject to change or withdrawal without notice.

Performance data contained herein was generally obtained in controlled, isolated environments. Customer examples are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual performance, cost, savings or other results in other operating environments may vary.

References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business.

Workshops, sessions and associated materials may have been prepared by independent session speakers, and do not necessarily reflect the views of IBM. All materials and

1 Overview of the lab 5 environment and scenario

This lab is a technical introduction to blockchain, specifically smart contract development using the latest developer enhancements in the Linux Foundation's Hyperledger Fabric v1.4 and shows you how IBM's Blockchain Platform's developer experience can accelerate your pace of development.

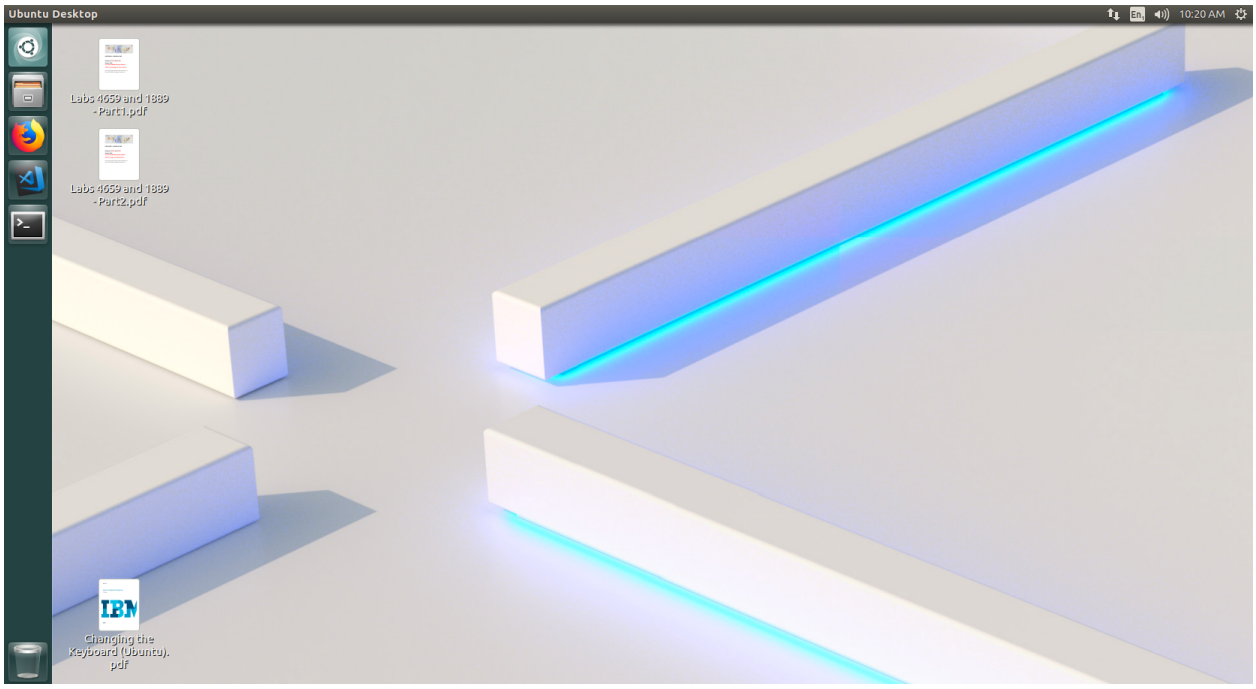
***Note**: The screenshots in this lab guide were taken using version **1.31.1** of **VSCode**, and version **0.3.0** of the **IBM Blockchain Platform** plugin. If you use different versions, you may see differences those shown in this guide.*

Start here. Instructions are always shown on numbered lines like this one:

__ 1. If it is not already running, start the virtual machine for the lab. The instructor will tell you how to do this if you are unsure.

__ 2. Wait for the image to boot and for the associated services to start. This happens automatically but might take several minutes. The image is ready to use when the desktop is visible as per the screenshot below.

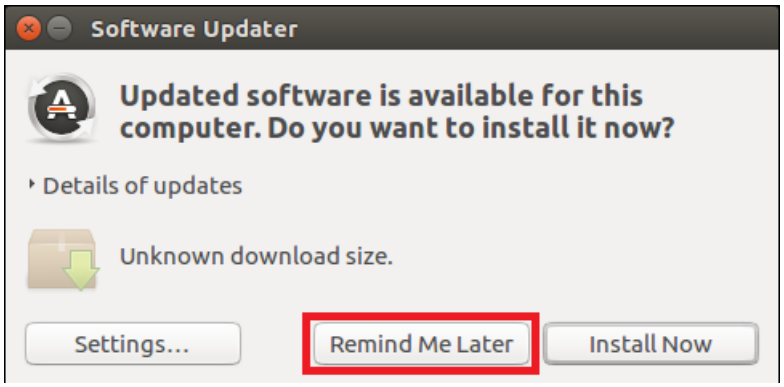
If it asks you to login, the userid and password are both “blockchain”.



1.1 Lab 5 Scenario

In this lab, we will import the Commercial Paper sample into VSCode and modify the Smart Contract to add a new transaction while also leveraging features of the IBM Blockchain Platform extension for VSCode to update an existing version of the Smart Contract running in the local_fabric runtime and generating tests for the Smart Contract transactions.

Note *that if you get an “Software Updater” pop-up at any point during the lab, please click “**Remind Me Later**”:*


Software Updater
Updated software is available for this computer. Do you want to install it now?
Details of updates
Unknown download size.
Settings...
Remind Me Later
Install Now

2 Lab 5: Import Commercial Paper Sample

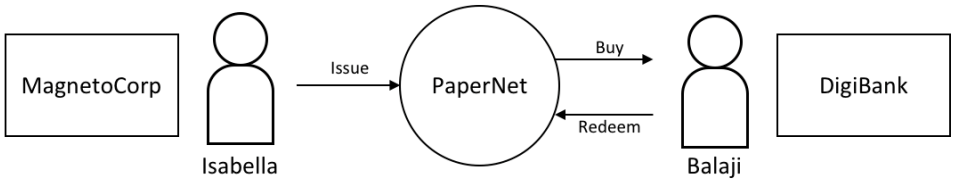
As mentioned above, this lab will be using the Hyperledger Fabric "Commercial Paper" tutorial. The full version of this tutorial is available online and we will be using a simplified version of it.

The scenario the tutorial follows is one of a commercial paper trading network called **PaperNet**. Commercial paper itself is a type of unsecured lending in the form of a "promissory note". The papers are normally issued by large corporations to raise funds to meet short-term financial obligations at a fixed rate of interest. Once issued at a fixed price, for a fixed term, another company or bank will purchase them at a discount to the face value and when the term is up, they will be redeemed for their face value.

As an example, if a paper was **issued** at a face value of 10M USD for a 6-month term at 2% interest then it could be **bought** for 9.8M USD (10M – 2%) by another company or bank who are happy to bear the risk that the issuer will not default. Once the term is up, then the paper could be **redeemed** or sold back to the issuer for their full face value of 10M USD. Between buying and redemption, the paper can be bought or sold between different parties on a commercial paper market.

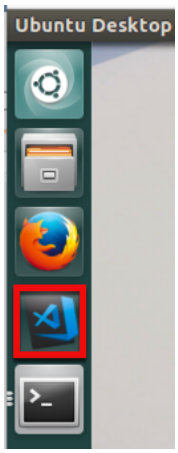
These three key steps of, **issue**, **buy** and **redeem** are the main transactions in a simplified commercial paper marketplace, which we will mirror in our lab. We will see a commercial paper **issued** by a company called MagnetoCorp and once issued on the PaperNet blockchain network, another company called DigiBank will first **buy** the paper and then **redeem** it.

In diagram form it looks like this:

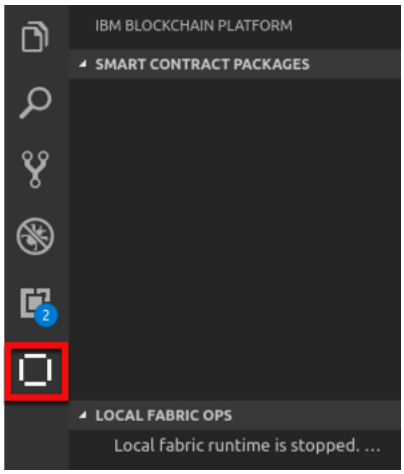


So, let's begin!

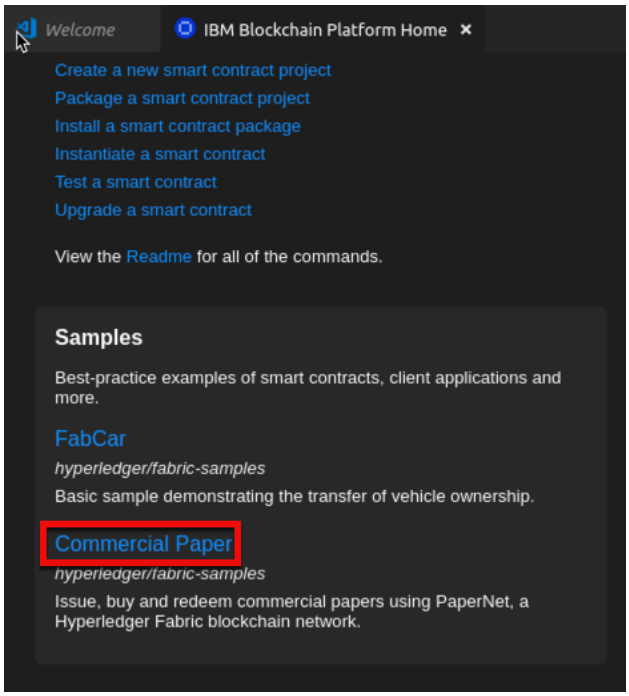
__ 3. Launch VSCode by clicking on the VSCode Icon in the toolbar.



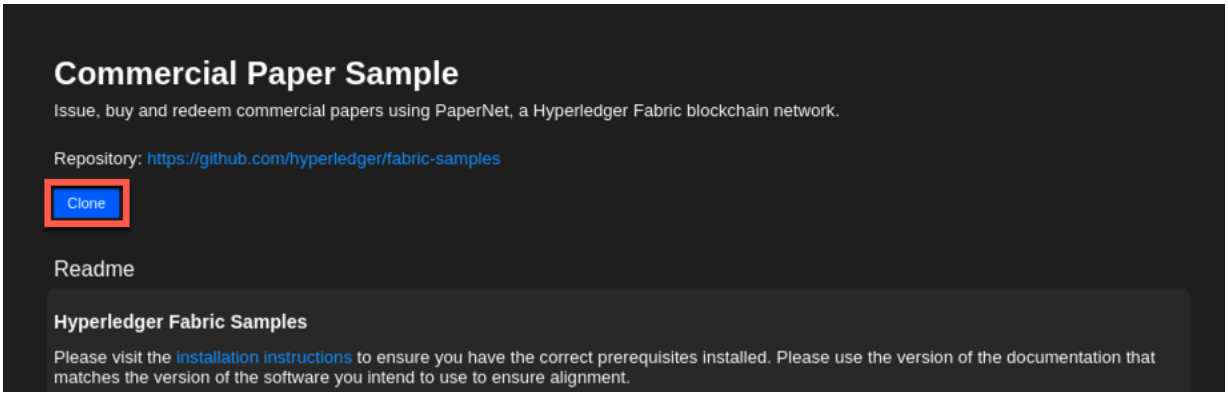
__ 4. When VSCode opens, click on the IBM Blockchain Platform (IBP) icon in the Activity Bar in VSCode as shown below.



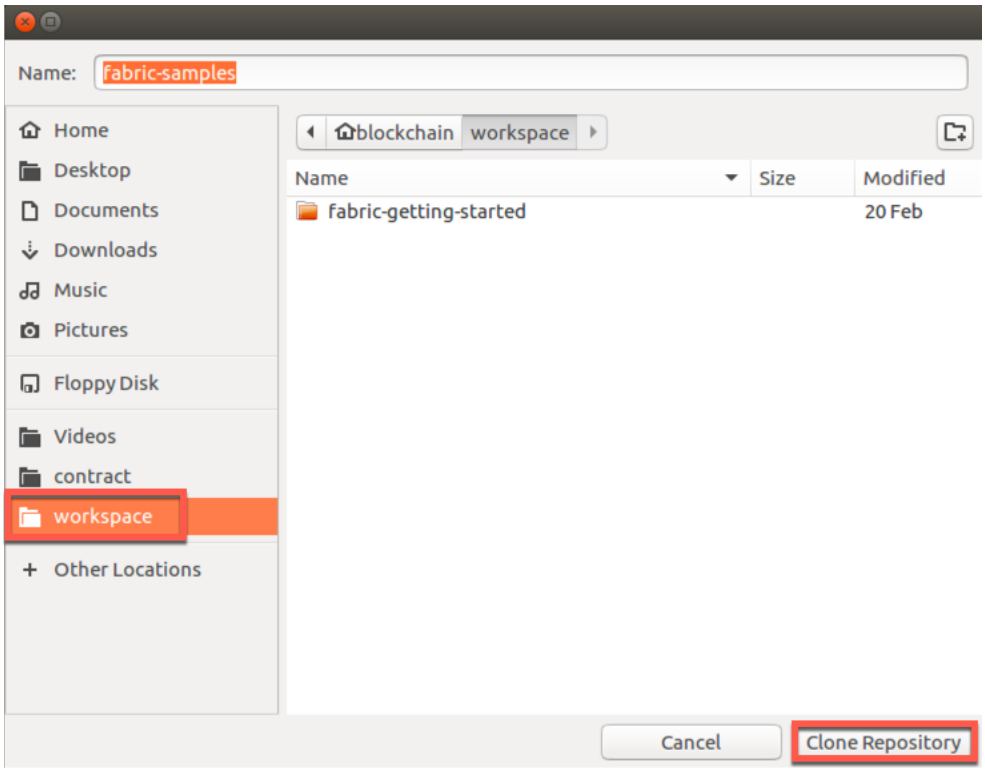
__ 5. Navigate to the IBM Blockchain Platform Home page and click the **Commercial Paper** link under Samples.



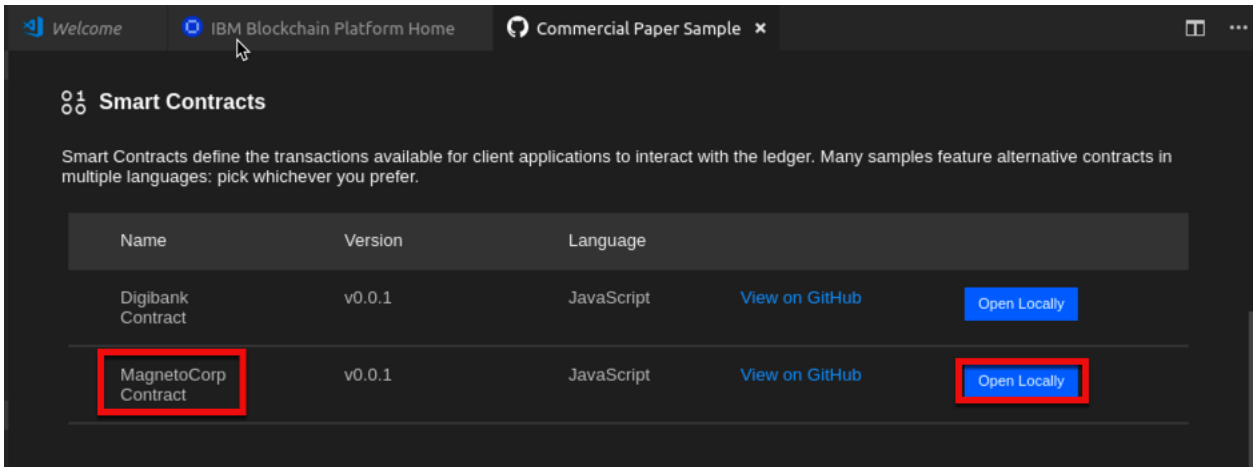
__ 6. The Commercial Paper Sample tab opens. Clone the samples repository by clicking on the **Clone** button as shown below.



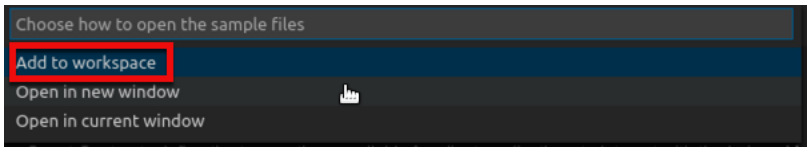
__ 7. At the next panel, select the workspace folder and click the **Clone Repository** button.



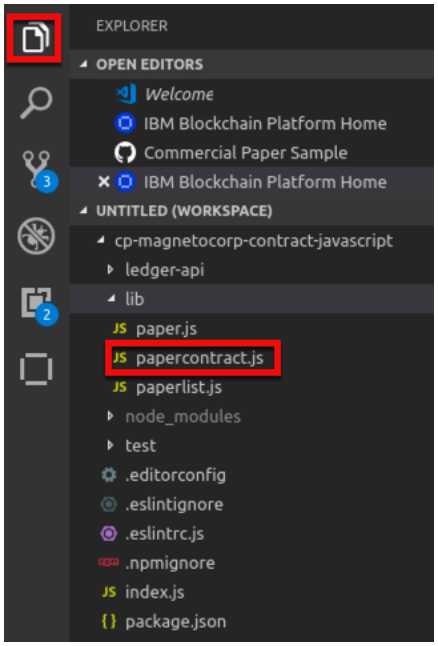
__ 8. Scroll down in the Commercial Paper Sample page and under Smart Contracts, click Open Locally next to **MagnetoCorp Contract.**



__ 9. Select **Add to workspace** at the Choose how to open the sample files prompt.



__ 10. Select the Explorer icon in the Activity Bar in VSCode as shown below, expand the lib folder and click the **papercontract.js** file.



__ 11. The Commercial Paper contract is more sophisticated than the contracts we saw in earlier labs, but it mostly works the same way. The main **CommercialPaperContract** class starts on **line 31**, and if we use the **"-"** buttons in the VSCode editor to fold methods of this class we can see that the main transactions are **instantiate**, **issue**, **buy** and **redeem**.

```
31   class CommercialPaperContract extends Contract {
32
33     constructor() {…
36     }
37
38     /**…
41     createContext() {…
43     }
44
45     /**…
49     async instantiate(ctx) {…
53     }
54
55     /**…
65     async issue(ctx, issuer, paperNumber, issueDateTime, matur
81     }
82
83     /**…
94     async buy(ctx, issuer, paperNumber, currentOwner, newOwner
120    }
121
122    /**…
131    async redeem(ctx, issuer, paperNumber, redeemingOwner, red
152    }
153
154  }
```

__ 12. Let's expand the **issue** transaction and perform a quick review on what it will do in case you did not complete the previous lab.

Line 68 creates a new **CommercialPaper** object from the parameters passed in using the static **createInstance** method on the **CommercialPaper** class. This class is defined in the separate "**paper.js**" file which is also if the **lib** folder alongside **papercontract.js** if you want to take a look at this method.

Line 72 Then moved the newly created paper into the **ISSUED** state and on **line 74** it has its owner set from the parameters passed in.

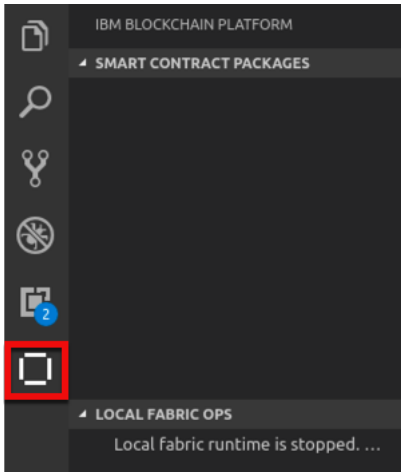
Line 77 adds the paper to a "**paperList**" which is responsible for storing the state of the paper in the world state. This is defined in the **paperlist.js** file if you want to take a deeper look.

Line 80 then returns a serialized form of the paper to the client who called this transaction.

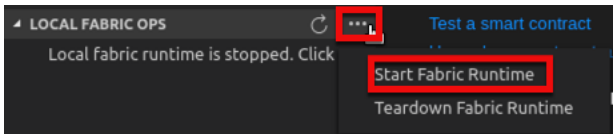
Feel free to have a look at the other files that make up the commercial paper smart contract. If you want to delve even deeper into the design of the CommercialPaper contract, there is much more information online if you have time to take a look.

Now we are going to install the **papercontract** onto a peer in the local_fabric network.

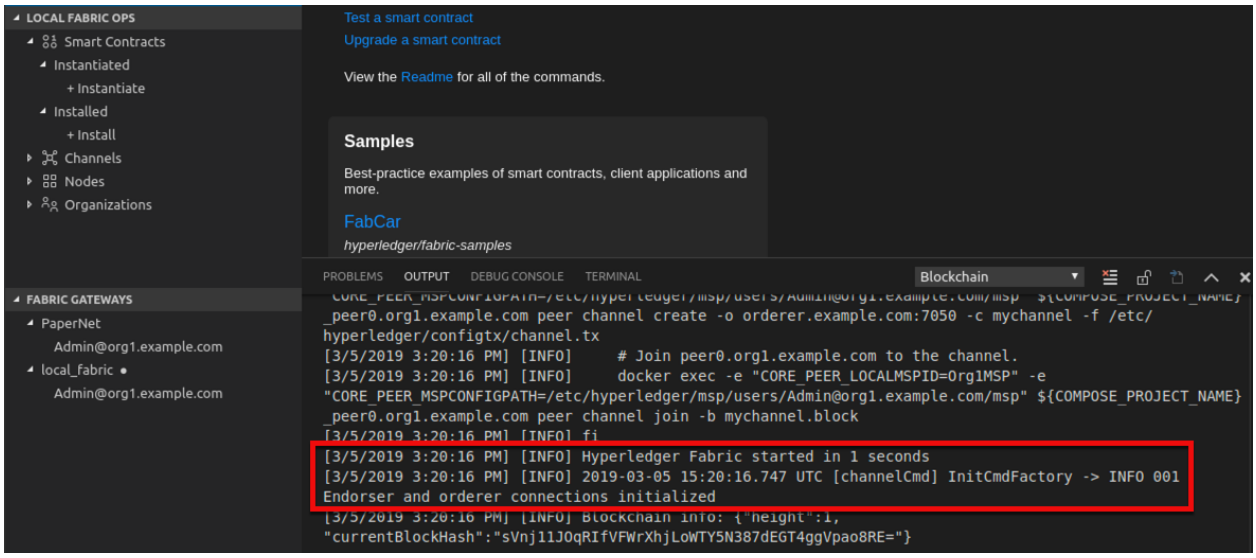
__ 13. Click on the IBM Blockchain Platform (IBP) icon in the Activity Bar in VSCode as shown below.



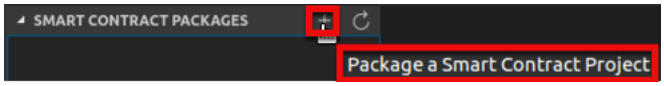
__ 14. Start the local_fabric by hovering over the ... next to LOCAL FABRIC OPS and selecting Start Fabric Runtime.



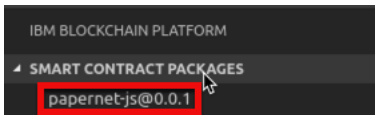
__ 15. The local_fabric runtime is successfully started when you see the following messages in the console pane. You are automatically connected to the local_fabric runtime.



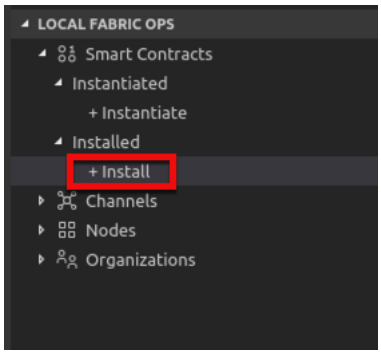
__ 16. Now we will package the Commercial Paper Smart Contract. Click the **+** next to Smart Contract packages to Package a Smart Contract Project.



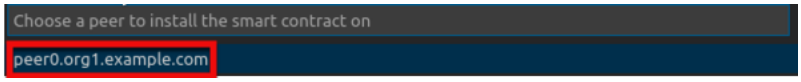
__ 17. The papernet-js@0.0.1 Smart Contract appears under the list of smart contract packages as shown below.



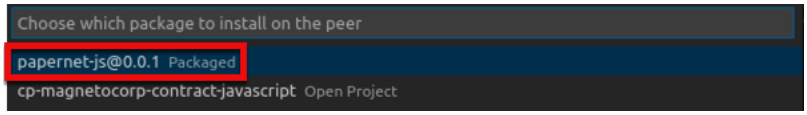
__ 18. Now we will install the Smart Contract. Under LOCAL FABRIC OPS, click **+ Install**.



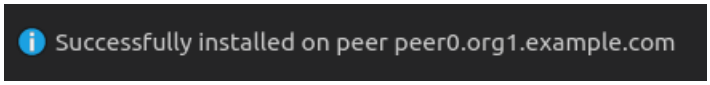
__ 19. Select **peer0.org1.example.com** at the Choose a peer to install the smart contract on prompt.



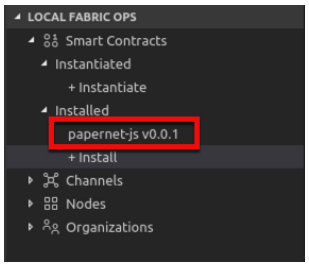
__ 20. Select **papernet-js@0.0.1** at the Choose which package to install on the peer prompt.



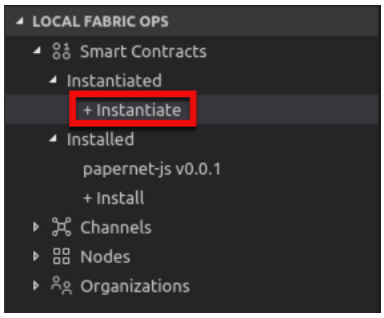
When the package is installed, an information message will be shown confirming the install:



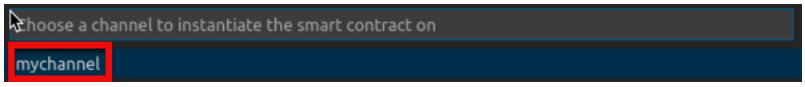
Now under Installed for Smart Contracts you can see the installed contract:



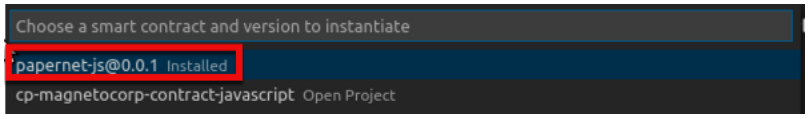
__ 21. Next, we have to instantiate the contract. Click **+ Instantiate** under LOCAL FABRIC OPS.



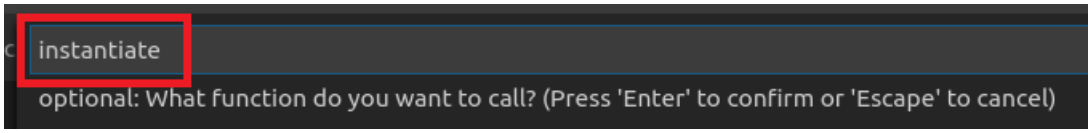
__ 22. Select **myChannel** at the Choose a channel to instantiate the smart contract on prompt.



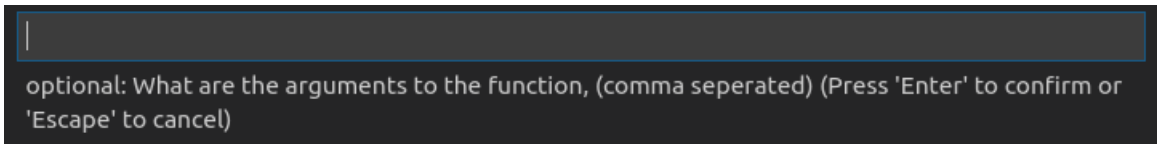
__ 23. Select **papernet-js@0.0.1** at the Choose a smart contract and version to instantiate prompt.



__ 24. In the pop-up dialogue box at the top of the screen asking "**optional: What function do you want to call?** ..." make sure you enter the word **instantiate** into the entry field as shown below. Before you press enter, check your spelling and make sure it is correct and is all lowercase without any quotes or spaces around it. This name has to exactly match the name of the transaction in your contract that will be called at instantiate time and in our default contract as we saw above this is called **instantiate**.

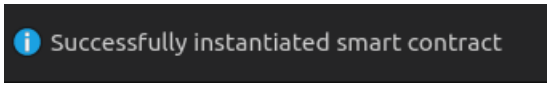


In the next dialogue that asks for parameters to the function, just press "**Enter**" as our **instantiate** function does not require any apart from the context "**ctx**" which is automatically provided by the framework.

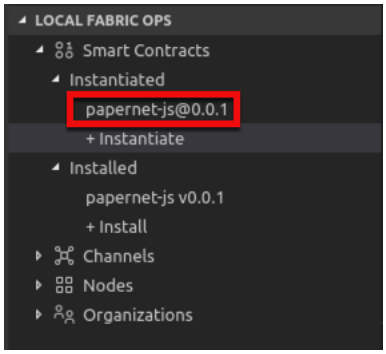


Instantiating a contract can take several minutes as a new docker container is built to contain the contract. Whilst it is happening you should see this information message

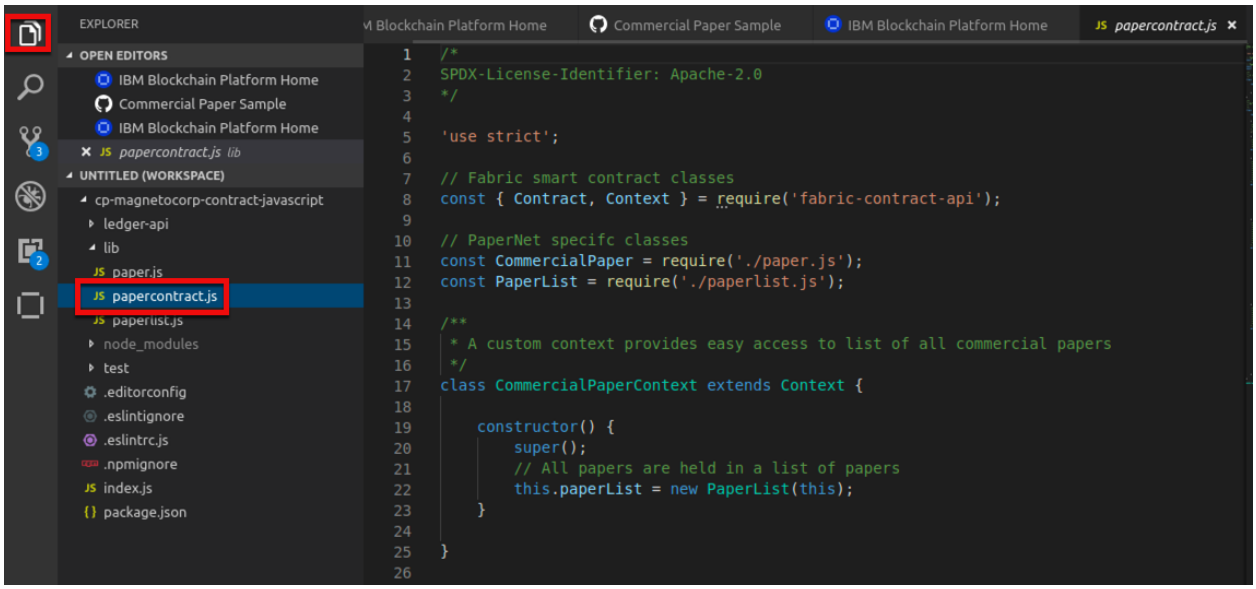
When it is complete you will see this information message



Once complete, the "**LOCAL FABRIC OPS**" view under **Instantiated** will change to show the Smart Contract, **papernet-js@0.0.1** to be instantiated.



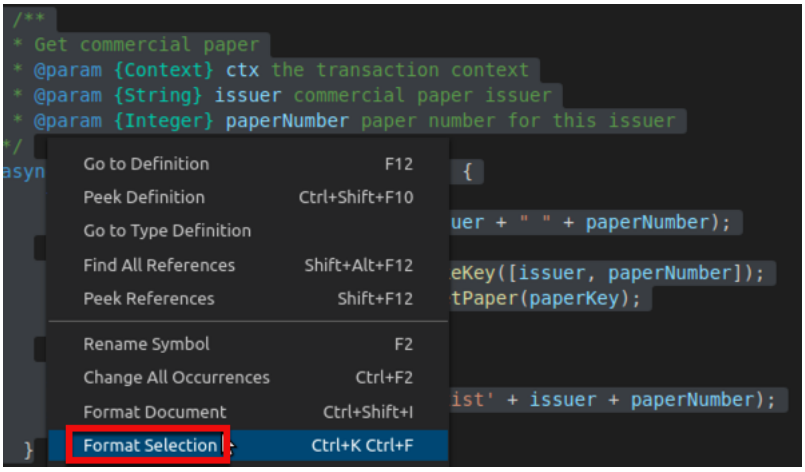
__ 25. Now we will modify the Smart Contract to add a **getPaper** transaction. Return to the explorer activity, expand the lib folder and double click on the **papercontract.js** smart contract to open it (if not already open) in the main editing view:



__ 26. We looked at the **issue** transaction in the **papercontract** earlier in this lab, but now we are going to create a new transaction called **getPaper**. It is going to be a simple transaction that just returns the paper that was requested as a parameter. We are going to insert it between the existing **instantiate** and **issue** transactions.

```
/**
  * Get commercial paper
  * @param {Context} ctx the transaction context
  * @param {String} issuer commercial paper issuer
  * @param {Integer} paperNumber paper number for this issuer
 */
 async getPaper(ctx, issuer, paperNumber) {
   try {
     console.log("getPaper for: " + issuer + " " + paperNumber);

     let paperKey = CommercialPaper.makeKey([issuer, paperNumber]);
     let paper = await ctx.paperList.getPaper(paperKey);
     return paper.toBuffer();

   } catch(e) {
     throw new Error('Paper does not exist' + issuer + paperNumber);
   }
 }
```

If you copy the code above, a handy capability in VSCode is to reformat the code so it has the appropriate indentation and tabs. To reformat the code you pasted in, select the code, right click and select **Format Selection**.



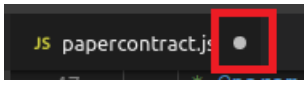
You can either copy the code above or type it in yourself, but make sure it is correct and in the right place as shown in the screenshot below:

```
49      async instantiate(ctx) {
50          // No implementation required with this example
51          // It could be where data migration is performed, if necessary
52          console.log('Instantiate the contract');
53      }
54
55      /**
56       * Get commercial paper
57       *
58       * @param {Context} ctx the transaction context
59       * @param {String} issuer commercial paper issuer
60       * @param {Integer} paperNumber paper number for this issuer
61       */
62      async getPaper(ctx, issuer, paperNumber) {
63          try {
64              console.log("getPaper for: " + issuer + " " + paperNumber);
65
66              let paperKey = CommercialPaper.makeKey([issuer, paperNumber]);
67              let paper = await ctx.paperList.getPaper(paperKey);
68              return paper.toBuffer();
69
70          } catch (e) {
71              throw new Error('Paper does not exist' + issuer + paperNumber);
72          }
73      }
74
75      /**
76       * Issue commercial paper
77       *
```

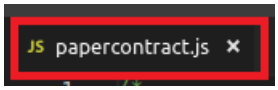
Notice how the new **getPaper** code is placed after the **instantiate** transaction finishes, and before the **issue** transaction begins.

__ 27. Make sure you save the changes, using the **File / Save** option or press **ctrl + s**

Note that when a file has changes pending it will have a filled in circle in its tab:

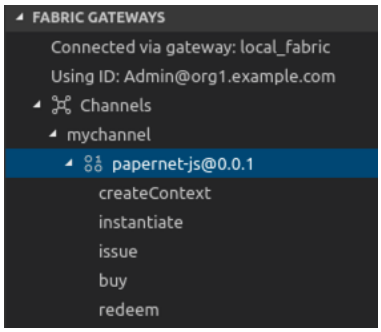


when it is saved this will change to an X:



__ 28. When the **papercontract.js** file is saved, click on the IBM Blockchain Platform icon in the sidebar to switch to the IBM Blockchain Platform view so we can use the IBM Blockchain Platform to simplify the smart contract upgrade experience.

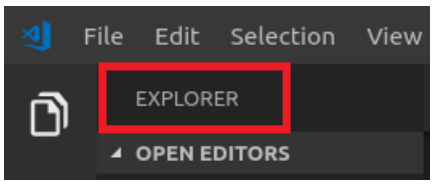
__ 29. Click on **Admin@org1.example.com** under local_fabric. Expand the channel **mychannel** and the previously instantiated **papernet-js@0.0.1** contract to see the transactions available:



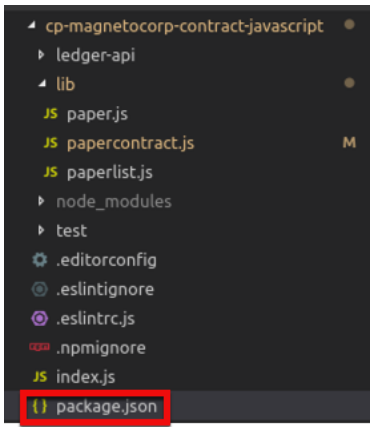
To change an instantiated contract, Hyperledger Fabric requires that we **upgrade** the existing contract to the new version. To do this we must increment the **version** number of the contract and make sure the **name** of the package will match the contract we are upgrading.

__ 30. Switch to the **Explorer** view in VSCode.

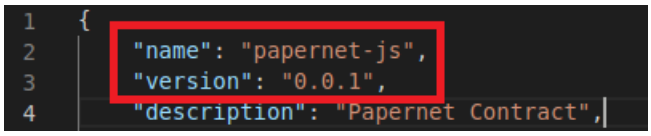
Note: If you are having problems and cannot see the **Explorer** view for any reason, click on its icon in the activity bar ( ) or press "**ctrl** + **shift + e**" to show it.



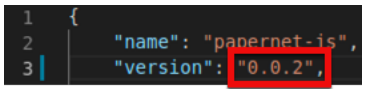
__ 31. Double click on the **package.json** file in the contract folder to open it for editing:



Lines two and three of the **package.json** file define the name and version of the contract package. Currently the name is **papernet-js** and the version is **0.0.1**:



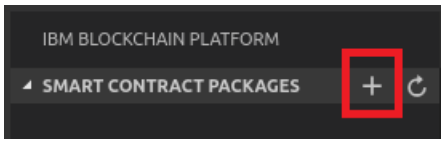
__ 32. Change the **version** to be **0.0.2**:



__ 33. Make sure you save the changes, using the **File / Save** option or press **ctrl + s**

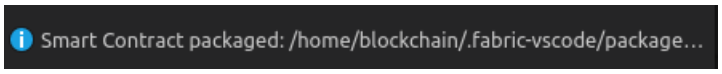
__ 34. Switch back to the IBM Blockchain Platform view:

__ 35. From the **Smart Contract Packages** pane, click the **+** to package the contract:

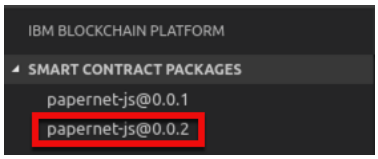


Note: The **+** only appears when you move your mouse over the **Smart Contract Packages** bar.

When the packaging is complete, you will see an informational message:

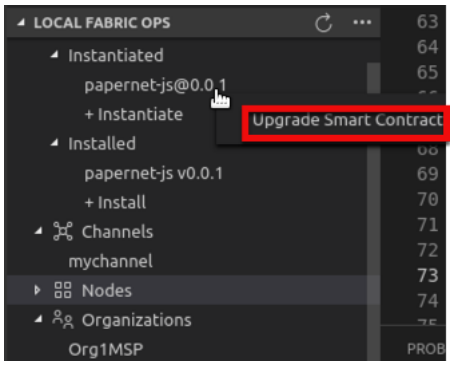


Also the package will appear in the Smart Contract Packages pane:

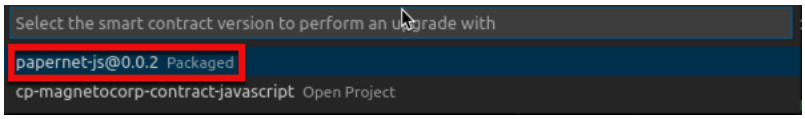


Note: Make sure that your package has got the correct name and version. If you don't see it, make sure you changed the version correctly as instructed above.

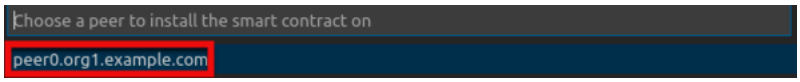
__ 36. To upgrade the contract to the new version that contains our **getPaper** transaction, right click on the existing **papercontract@0** instantiated contract and choose the **Upgrade Smart Contract** menu option:



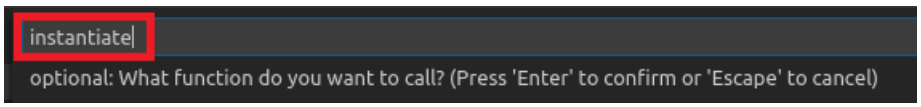
__ 37. From the "**Select the smart contract version to perform an upgrade with**" pop up at the top of the screen, choose the **papernet-js@0.0.2 Packaged** option:



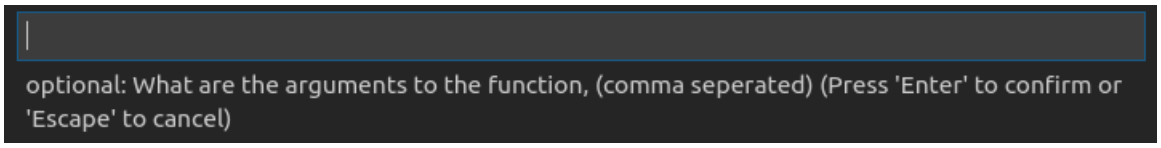
From the "**Choose a peer to install the smart contract on ...**" pop up menu, select **peer0.org1.example.com** as shown below.



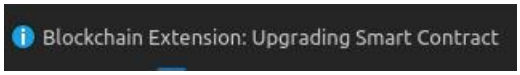
From the "**optional: What function do you want to call...**" pop up menu, enter the word **instantiate** as shown below. Remember this has to be entered exactly as shown:



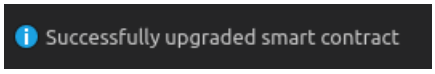
In the next dialogue that asks for parameters to the function, just press "**Enter**" as our **instantiate** function does not require any additional parameters:



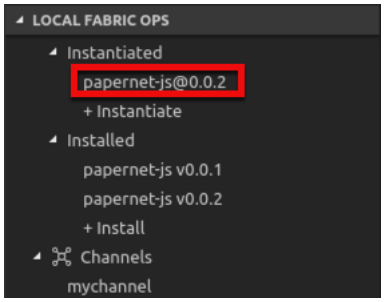
Upgrading a smart contract can take several minutes as a new docker container is built to contain the new contract. Whilst it is happening you should see this information message



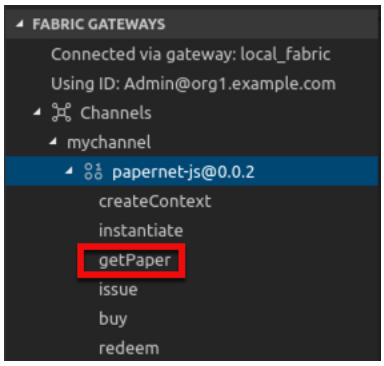
When it is complete you will see this information message



Once complete, the "**LOCAL FABRIC OPS**" view under **Instantiated** will change to show **papernet-js@0.0.2** at the same level as the peer **peer0.org1.example.com**.

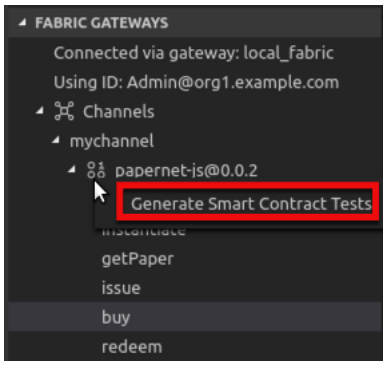


__ 38. Expand the newly upgraded contract **papernet-js@0.0.2**, and you will see the new **getPaper** transaction is now available:



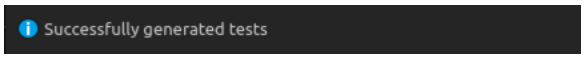
Note: If you don't see the **getPaper** transaction, make sure you edited the **package.json** and **papercontract.js** and saved the changes.

__ 39. Now we will generate a functional test module for the Smart Contract. Right click on papernet-js@0.0.2 and select **Generate Smart Contract Tests**.



__ 40. From the "**Choose preferred test language...**" pop up menu, select **JavaScript**.

When the npm install of dependencies is completed, you will see the Successfully generated tests message.



__ 41. Now let's review the newly generated functional test module, org.papernet.commercialpaper-papernet-js@0.0.2.test.js that opens in VSCode.

Take a look at the code for the generated functional test module. The line numbers below may vary slightly depending on much whitepaper you added when inserting the getPaper transaction. The main points are:

Lines 21-24: import various dependencies

Line 28: Create a new Gateway

Line 40: Load the connection profile from file system

Line 44: Load the identity from the wallet on the file system

Line 49: Connect to the gateway before each request

Line 118: Get the **mychannel** channel from the gateway

Line 119: Get the **papercontract** contract from the gateway

Line 120: Use the contract to submit the **getPaper, issue, buy or redeem** transaction, passing in the transaction name as well as details associated with the commercial paper associated with that particular transaction.

Line 121: Return the response buffer

__ 42. Scroll down in the org.papernet.commercialpaper-papernet-js@0.0.2.test.js functional test module to the **getPaper** transaction test and replace the TODO sections of the code as shown below. The **getPaper** transaction test begins with **it('getPaper', async () => {**

BEFORE

```
// TODO: Update with parameters of transaction
    const args = [''];
```

AFTER

```
const args = ['MagnetoCorp', '00001'];
```

BEFORE

```
// TODO: Update with return value of transaction
// assert.equal(JSON.parse(response.toString()), undefined);
```

AFTER

```
console.log('Process getPaper transaction response.');
        const CommercialPaper = require('../lib/paper.js');
let paper = CommercialPaper.fromBuffer(response);
    console.log(`${paper.issuer} commercial paper :
${paper.paperNumber} successfully retrieved with owner ${paper.owner}`);
    console.log('Transaction complete.');
```

The code show look as follows:

```
    it('getPaper', async () => {
        // TODO: Update with parameters of transaction
        const args = ['MagnetoCorp', '00001'];

        const response = await submitTransaction('getPaper', args); // Returns buffer of transaction return value
        console.log('Process getPaper transaction response.');
        const CommercialPaper = require('../lib/paper.js');
        let paper = CommercialPaper.fromBuffer(response);
        console.log(`${paper.issuer} commercial paper : ${paper.paperNumber} successfully retrieved with owner ${paper.owner}`);
        console.log('Transaction complete.');

    }).timeout(10000);
```

__ 43. Scroll down in the org.papernet.commercialpaper-papernet-js@0.0.2.test.js functional test module to the **issue** transaction test and replace the TODO sections of the code as shown below. The **issue** transaction test begins with **it('issue', async () => {**

BEFORE

```
// TODO: Update with parameters of transaction
    const args = ['''];
```

AFTER

```
const args = ['MagnetoCorp', '00001', '2020-05-31', '2020-11-30',
'5000000'];
```

BEFORE

```
// TODO: Update with return value of transaction
// assert.equal(JSON.parse(response.toString()), undefined);
```

AFTER

```
console.log('Process issue transaction response.');
const CommercialPaper = require('../lib/paper.js');
     let paper = CommercialPaper.fromBuffer(response);
console.log(`${paper.issuer} commercial paper : ${paper.paperNumber}
successfully issued for value ${paper.faceValue}`);
console.log('Transaction complete.');
```

The code show look as follows:

```
93    it('issue', async () => {
94        const args = ['MagnetoCorp', '00001', '2020-05-31', '2020-11-30', '5000000'];
95
96        const response = await submitTransaction('issue', args); // Returns buffer of transaction return value
97        console.log('Process issue transaction response.');
98        const CommercialPaper = require('../lib/paper.js');
99        let paper = CommercialPaper.fromBuffer(response);
100       console.log(`${paper.issuer} commercial paper : ${paper.paperNumber} successfully issued for value ${paper.faceValue}`);
101       console.log('Transaction complete.');
102
103   }).timeout(10000);
```

__ 44. Scroll down in the org.papernet.commercialpaper-papernet-js@0.0.2.test.js functional test module to the **buy** transaction test and replace the TODO sections of the code as shown below. The **buy** transaction test begins with **it('buy', async () => {**

BEFORE

```
// TODO: Update with parameters of transaction
     const args = [''];
```

AFTER

```
const args = ['MagnetoCorp', '00001', 'MagnetoCorp', 'DigiBank',
'4900000', '2020-05-31'];
```

BEFORE

```
// TODO: Update with return value of transaction
// assert.equal(JSON.parse(response.toString()), undefined);
```

AFTER

```
console.log('Process buy transaction response.');
     const CommercialPaper = require('../lib/paper.js');
     let paper = CommercialPaper.fromBuffer(response);
     console.log(`${paper.issuer} commercial paper :
${paper.paperNumber} successfully purchased by ${paper.owner}`);
     console.log('Transaction complete.');
```

The code show look as follows:

```
106     it('buy', async () => {
107         const args = ['MagnetoCorp', '00001', 'MagnetoCorp', 'DigiBank', '4900000', '2020-05-31'];
108
109         const response = await submitTransaction('buy', args); // Returns buffer of transaction return value
110         console.log('Process buy transaction response.');
111         const CommercialPaper = require('../lib/paper.js');
112         let paper = CommercialPaper.fromBuffer(response);
113         console.log(`${paper.issuer} commercial paper : ${paper.paperNumber} successfully purchased by ${paper.owner}`);
114         console.log('Transaction complete.');
115
116     }).timeout(10000);
```

__ 45. Scroll down in the org.papernet.commercialpaper-papernet-js@0.0.2.test.js functional test module to the **redeem** transaction test and replace the TODO sections of the code as shown below. The **redeem** transaction test begins with **it('redeem', async () => {**

BEFORE

```
// TODO: Update with parameters of transaction
    const args = [''];
```

AFTER

```
const args = ['MagnetoCorp', '00001', 'DigiBank', '2020-11-30'];
```

BEFORE

```
// TODO: Update with return value of transaction
// assert.equal(JSON.parse(response.toString()), undefined);
```

AFTER

```
console.log('Process redeem transaction response.');
const CommercialPaper = require('../lib/paper.js');
    let paper = CommercialPaper.fromBuffer(response);
    console.log(`${paper.issuer} commercial paper :
${paper.paperNumber} successfully redeemed with ${paper.owner}`);
    console.log('Transaction complete.');
```

The code show look as follows:

```
117     it('redeem', async () => {
118         const args = ['MagnetoCorp', '00001', 'DigiBank', '2020-11-30'];
119
120         const response = await submitTransaction('redeem', args); // Returns buffer of transaction return value
121         console.log('Process redeem transaction response.');
122         const CommercialPaper = require('../lib/paper.js');
123         let paper = CommercialPaper.fromBuffer(response);
124         console.log(`${paper.issuer} commercial paper : ${paper.paperNumber} successfully redeemed with ${paper.owner}`);
125         console.log('Transaction complete.');
126
127     }).timeout(10000);
```

__ 46. Make sure you save the changes, using the **File / Save** option or press **ctrl + s**

__ 47. Now let's run our first test by issuing a new commercial paper. Scroll to the **issue** transaction and click the **Run Test** link.

```
Run Test | Debug Test
93    it('issue', async () => {
94        const args = ['MagnetoCorp', '00001', '2020-05-31', '2020-11-30', '5000000'];
95
96        const response = await submitTransaction('issue', args); // Returns buffer of transaction return value
97        console.log('Process issue transaction response.');
98        const CommercialPaper = require('../lib/paper.js');
99        let paper = CommercialPaper.fromBuffer(response);
100       console.log(`${paper.issuer} commercial paper : ${paper.paperNumber} successfully issued for value ${paper.faceValue}`);
101       console.log('Transaction complete.');
102
103   }).timeout(10000);
104
```

If successful, the **successfully issued** message and **1 passing** message will appear as below. The commercial paper was issued for MagnetoCorp.

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL                                   1: bash
er-papernet-js@0.0.2.test.js --grep="issue"


  org.papernet.commercialpaper-papernet-js@0.0.2
2019-03-05T22:12:27.487Z - info: [TransactionEventHandler]: _strategySuccess: strategy success for transaction "0cdbd88f61156b8356c17d01772fb4d05004189bf3c9b1997
dcd9de43f8e85c8"
Process issue transaction response.
MagnetoCorp commercial paper : 00001 successfully issued for value 5000000
Transaction complete.
    ✓ issue (2319ms)


  1 passing (3s)
```

__ 48. Now let's view what is stored in the ledger. Scroll to the **getPaper** transaction and click the **Run Test** link

```
Run Test | Debug Test
80    it('getPaper', async () => {
81        // TODO: Update with parameters of transaction
82        const args = ['MagnetoCorp', '00001'];
83
84        const response = await submitTransaction('getPaper', args); // Returns buffer of transaction return value
85        console.log('Process getPaper transaction response.');
86        const CommercialPaper = require('../lib/paper.js');
87        let paper = CommercialPaper.fromBuffer(response);
88        console.log(`${paper.issuer} commercial paper : ${paper.paperNumber} successfully retrieved with owner ${paper.owner}`);
89        console.log('Transaction complete.');
90
91    }).timeout(10000);
92
```

If successful, the **successfully retrieved** message and **1 passing** message will appear as below. The commercial paper was retrieved for MagnetoCorp and is currently owned by MagnetoCorp.

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL                                   1: bash
er-papernet-js@0.0.2.test.js --grep="getPaper"


  org.papernet.commercialpaper-papernet-js@0.0.2
2019-03-05T22:24:05.898Z - info: [TransactionEventHandler]: _strategySuccess: strategy success for transaction "9fb07ced48b55a22b357fab6272fe3f1356f361771d4a07c3
5f48005914d54cc"
Process getPaper transaction response.
MagnetoCorp commercial paper : 00001 successfully retrieved with owner MagnetoCorp
Transaction complete.
    ✓ getPaper (2191ms)


  1 passing (2s)
```

__ 49. Now let's assume the role of Digibank and purchase the commercial paper issued by MagnetoCorp. Scroll to the **buy** transaction and click the **Run Test** link.

```
Run Test | Debug Test
105   it('buy', async () => {
106       const args = ['MagnetoCorp', '00001', 'MagnetoCorp', 'DigiBank', '4900000', '2020-05-31'];
107
108       const response = await submitTransaction('buy', args); // Returns buffer of transaction return value
109       console.log('Process buy transaction response.');
110       const CommercialPaper = require('../lib/paper.js');
111       let paper = CommercialPaper.fromBuffer(response);
112       console.log(`${paper.issuer} commercial paper : ${paper.paperNumber} successfully purchased by ${paper.owner}`);
113       console.log('Transaction complete.');
114
115   }).timeout(10000);
```

If successful, the **successfully purchased** message and **1 passing** message will appear as below. The commercial paper was purchased by DigiBank.

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL                                   1: bash
er-papernet-js@0.0.2.test.js --grep="buy"


  org.papernet.commercialpaper-papernet-js@0.0.2
2019-03-05T22:36:47.210Z - info: [TransactionEventHandler]: _strategySuccess: strategy success for transaction "044a5f258615b8d7b6e1886245e0ac15011d3aa86ae6d9fd1
b4fc8567f7a8bdb"
Process buy transaction response.
MagnetoCorp commercial paper : 00001 successfully purchased by DigiBank
Transaction complete.
    ✓ buy (2218ms)


  1 passing (2s)
```

__ 50. Run the **getPaper** transaction again as you did in a previous step above to see the current state of the commercial paper. Note the owner of the commercial paper is now Digibank as a result of the buy transaction.

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL                                   1: bash
er-papernet-js@0.0.2.test.js --grep="getPaper"


  org.papernet.commercialpaper-papernet-js@0.0.2
2019-03-05T22:40:21.751Z - info: [TransactionEventHandler]: _strategySuccess: strategy success for transaction "ac8dce55181574c129467a777c43e6136a3ac57cf76ac8bfb
9f575cf231d00cf"
Process getPaper transaction response.
MagnetoCorp commercial paper : 00001 successfully retrieved with owner DigiBank
Transaction complete.
    ✓ getPaper (2177ms)


  1 passing (2s)
```

__ 51. Now let's assume the role of DigiBank and redeem the commercial paper previously purchased by DigiBank. Scroll to the **redeem** transaction and click the **Run Test** link.

```
117     it('redeem', async () => {
118         const args = ['MagnetoCorp', '00001', 'DigiBank', '2020-11-30'];
119
120         const response = await submitTransaction('redeem', args); // Returns buffer of transaction return value
121         console.log('Process redeem transaction response.');
122         const CommercialPaper = require('../lib/paper.js');
123         let paper = CommercialPaper.fromBuffer(response);
124         console.log(`${paper.issuer} commercial paper : ${paper.paperNumber} successfully redeemed with ${paper.owner}`);
125         console.log('Transaction complete.');
126
127     }).timeout(10000);
```

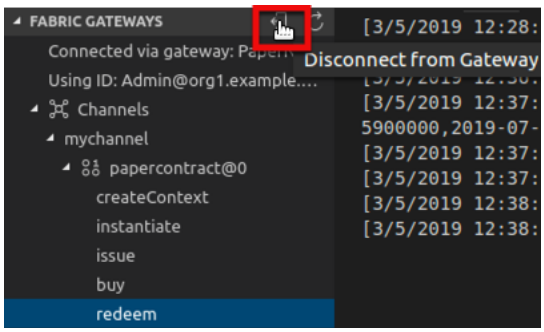
If successful, the **successfully redeemed** message and **1 passing** message will appear as below. The commercial paper was redeemed by MagetoCorp.

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL                                                    1: bash
er-papernet-js@0.0.2.test.js --grep="redeem"


  org.papernet.commercialpaper-papernet-js@0.0.2
2019-03-05T22:47:33.383Z - info: [TransactionEventHandler]: _strategySuccess: strategy success for transaction "f67138db27aaff74daa84a66d856a2d2c75fd325b4d96ad34
e707128b652d642"
Process redeem transaction response.
MagnetoCorp commercial paper : 00001 successfully redeemed with MagnetoCorp
Transaction complete.
    ✓ redeem (2168ms)


  1 passing (2s)
```

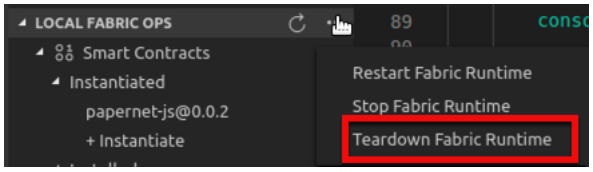
__ 52. Run the **getPaper** transaction again as you did in a previous step above to see the current state of the commercial paper. Note the owner of the commercial paper is now MagnetoCorp again as a result of the redeem transaction.

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL                                                    1: bash
er-papernet-js@0.0.2.test.js --grep="getPaper"


  org.papernet.commercialpaper-papernet-js@0.0.2
2019-03-05T22:49:34.859Z - info: [TransactionEventHandler]: _strategySuccess: strategy success for transaction "bb365ca7d7220304324ace432d99206334ad7e8151ba6b878
2b34257d6bd20eb"
Process getPaper transaction response.
MagnetoCorp commercial paper : 00001 successfully retrieved with owner MagnetoCorp
Transaction complete.
    ✓ getPaper (2136ms)


  1 passing (2s)
```

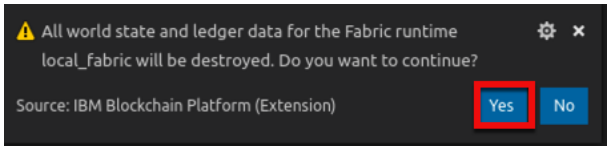
__ 53. From the IBP **FABRIC GATEWAY** view, select the **Disconnect from Gateway** icon as shown below:



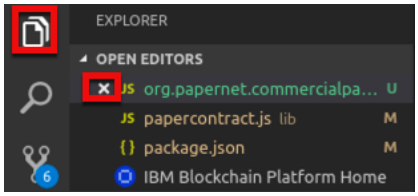
__ 54. From the IBP **FABRIC GATEWAY** view, click the ... and select **Teardown Fabric Runtime** as shown below:



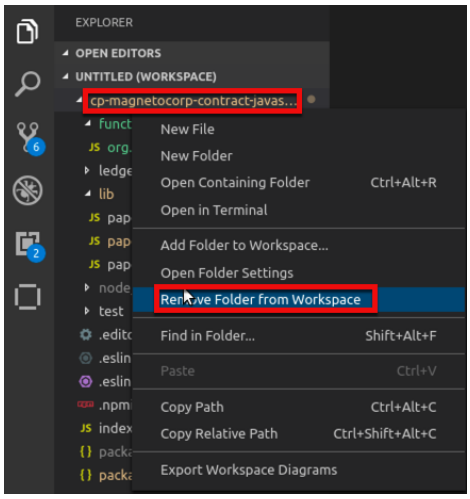
Click the Yes button to destroy all world state and ledger data.



__ 55. Switch back to the **Explorer** view and close all open editors in the **Open Editors** view, including the "**IBM Blockchain Platform Home**" by clicking on the "**x**" button on each one in turn:



__ 56. Right click on the **cp-magnetocorp-contract-javascript** top level folder and select **Remove Folder from Workspace**.



__ 57. We have now completed this lab **– Import Commercial Paper Sample** and we hope you enjoyed it.