# Exercises: Hyper Ledger

This document describes the **exercise assignments** for the ["Blockchain Academy" course @ Software University](https://softuni.bg/courses/programming-fundamentals). In this lesson we learned about **Hyper Ledger**. The goal of this exercise is to get some practical skills about it.

## Built a Network

For this exercise we will be needing the **Docker Toolbox** so we start with installing it.[[1]](#footnote-1) Open the following link

<https://docs.docker.com/toolbox/toolbox_install_windows/>

You should be presented with the page from which the **Docker Toolbox** can be downloaded. Click on **[Get Docker Toolbox for Windows] button.**

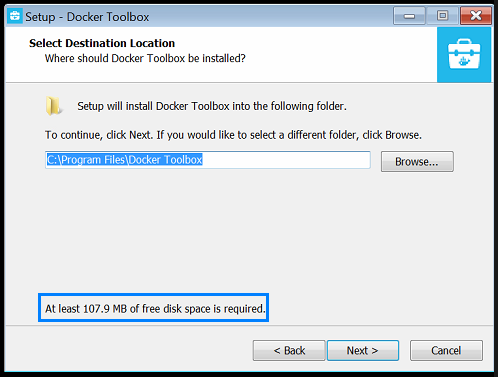
****

You will be prompted to save a file with the name **DockerToolbox.exe**. Save it and then **run** it. You will receive some notifications asking you for different permissions. Click **[Yes]**. Now you should see the following window:

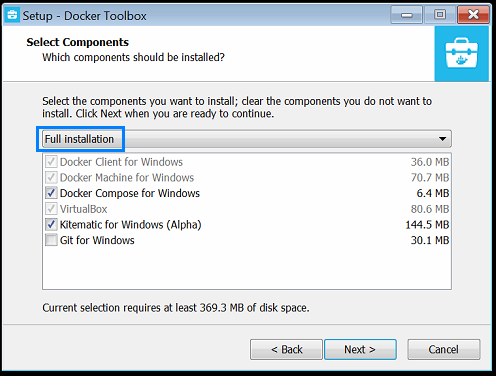


Notice that **Help Docker improve Toolbox** is selected by default. It is up to you to deselect this option or leave it as is. Click **[Next]**.

Now you are asked where the **Docker Toolbox** is to be installed:

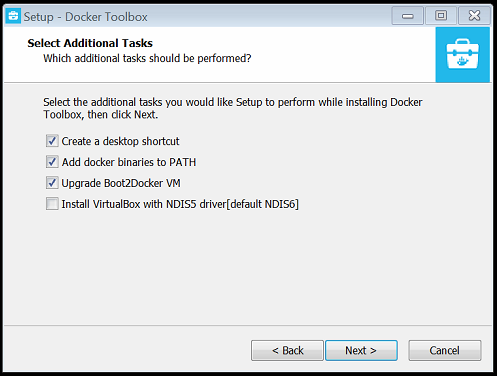


If you have enough available space leave the default path otherwise change it. Click **[Next]**.



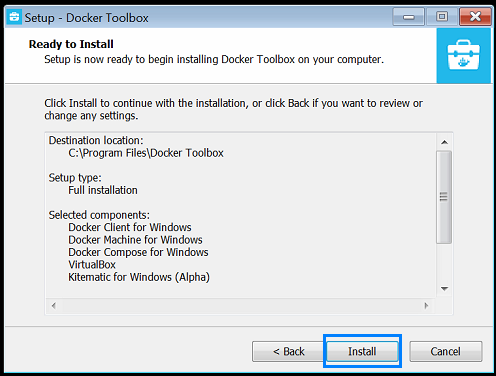
By default the installation option should be **Full installation**. If it is not, select it from the dropdown menu and click **[Next]**.

Now you should choose what type of shortcuts you will be using.

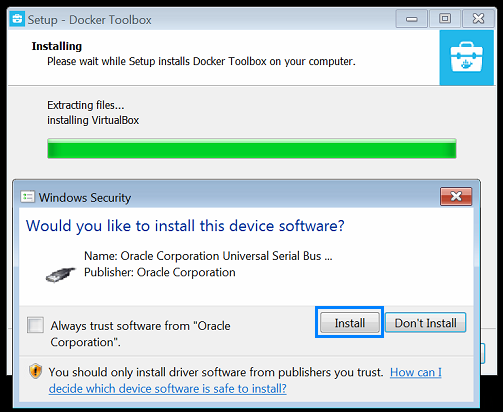


Select whichever option best suit you and click **[Next]**.

Now you are presented with the installation summary. Click **[Install]**.



During the installation process you will be asked a number of times whether to install a component. A sample screenshot follows:



Click **[Install]** every single time.

When the installation is completed the following window will appear:

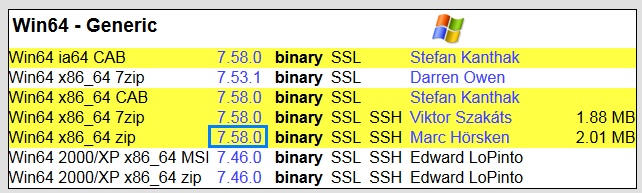


It is up to you whether you would like to see the **Shortcuts** in the file explorer. Click **[Finish]**.

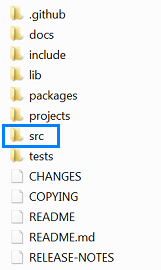
The next tool that we are going to need is **cURL**. Go to the following address:

<https://curl.haxx.se/download.html>

Then scroll down almost to the bottom of the page. You should find the section depicted in the next screenshot:



Click on the numbers in the blue rectangle[[2]](#footnote-2). You are going to be prompted to save a **.ZIP** file locally. After downloading the file open it with a software of your choice and extract the files. The folder should have the following structure:



In the folder **SRC** resides the **curl.exe** file that we are going to be using.

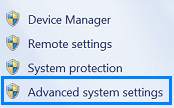
We would like to be able to **access** this file from anywhere and to achieve that we will add the path to it to the **Environmental Variable PATH**.

One way to do this is to open the **Windows Explorer** and to **right click** on the **Computer** **icon[[3]](#footnote-3)**.

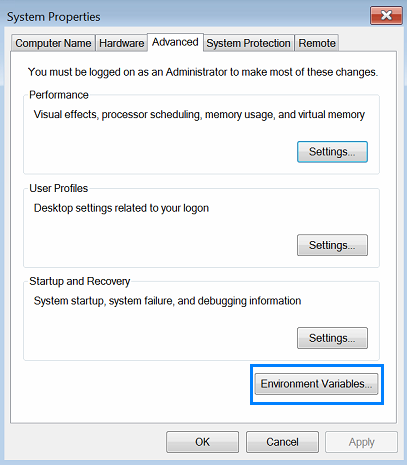


Next, we left click on **Properties**.

In the upper-left part of the next window you should be seeing this:

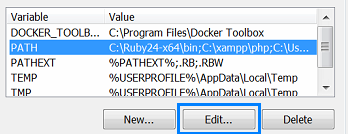


Left click on th**e** **advanced system settings**. Now you should be presented with the following window:



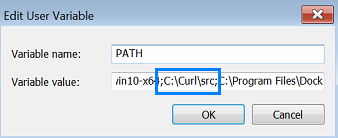
Left click on the **[Environment Variable] button**.

In the next window the Environment variable **PATH** is located:



Select PATH with a **single left click** and then click the **[Edit] button**.

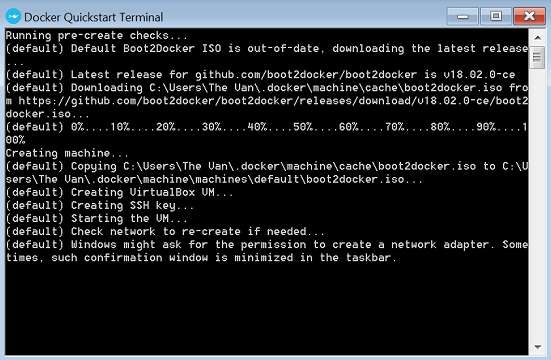
**Add** the **path to the curl.exe file** in the field by separating it with a semi-colon. Since I have installed the cURL folder on **C:\** what is added is simply **C:\Curl\src**



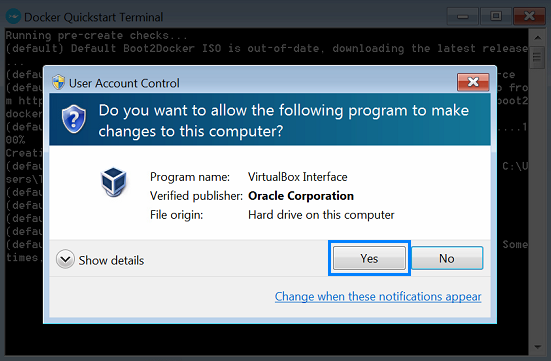
Notice the separating semi-colons.

Now we are ready to begin!

Start the **Docker Quickstart Terminal**. Some information will be shown:

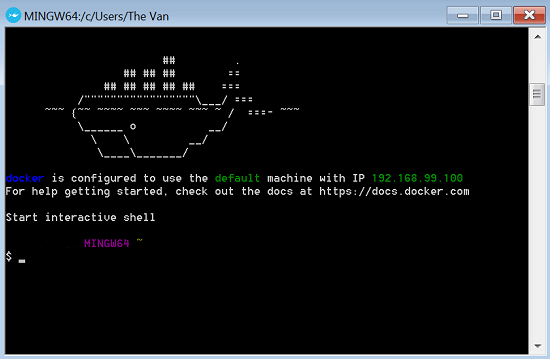


Some windows may appear that asks you for permission to make changes to the computer. A sample screenshot follows:

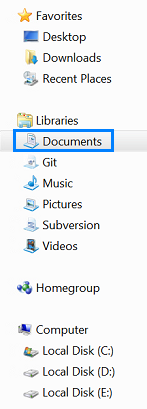


Click the **[Yes] button** every time.

Depending on your computer configuration some time will be needed in order for everything to start so be patient. When everything is ready you should see a window depicted by the next screenshot:



Now open the **Windows Explorer** and in the **Documents Library** create a folder **Hyperledger**.





Now go back to the **Docker** and navigate to this folder.

To achieve this we mu execute the following commands:



and



To be sure that you have navigated correctly type the



command. It should not return anything meaning the folder is empty.[[4]](#footnote-4)

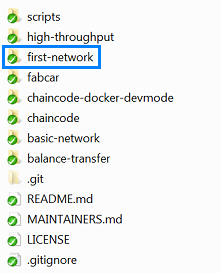
Now that we are in the correct folder we will execute the following commands:

|  |
| --- |
| git config –-global core.autocrlf false |
| git config –-global core.longpaths true |

Both of these commands are needed if we are to get the repository correctly. Next we write the following command:

|  |
| --- |
| git clone https://github.com/hyperledger/fabric-samples.git |

When the execution is completed go to the folder **Hyperledger** in the **Documents** library. In it you must be seeing a new directory with the name **fabric-samples**. Open it. Its structure should look like the following screenshot:



Go back to the **Docker**. In it go into the folder **fabric-samples**.



In it write the following command:

|  |
| --- |
| curl –sSL <https://goo.gl/byy2Qj> | bash -s 1.0.5 |

This command downloads and executes a bash script that gets all the needed binaries that are needed to set up the network.

The execution may take a while so be patient.

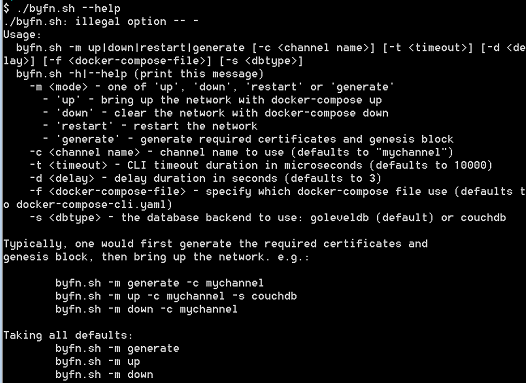
After everything is complete navigate to the **first-network** folder.



In it is the file that we need in order to start our network. The file is **byfn.sh** if you execute the command

|  |
| --- |
| /byfn.sh -help |

you will be represented with the file’s functionality:



Before continuing navigate with **Widows Explorer** to the folder **channel-artifacts** (it is a subfolder of **first-network**). In it you should find only one **.gitkeep** file. This will change in a minute.

Now, in the **Docker** write the following command:

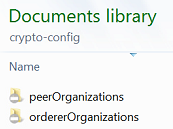
|  |
| --- |
| ./byfn.sh –m generate |

This command creates the information that is going to be needed to start the network. After entering the command you should see something similar to the following screenshot:

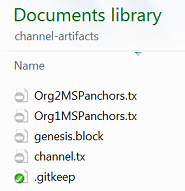


Since we have not provided any information about the channel, the default values for name and timeout will be used. Type **y** and press enter.

After the operation is concluded you will notice that there is a new folder called **crypto-config** (it is a subfolder of the **first-network** folder). In it are all the keys needed for the network to function properly.



In addition, new files have been added to the **channel-artifacts** folder. These contain information about the functionality of the network.



Since we now have the structure of our network and the necessary keys we are ready to bring it up. Write the following command in the **Docker** and press enter:

|  |
| --- |
| ./byfn.sh –m up |

You will be asked whether you are willing to create a channel with certain parameters:



Answer with **y** and press enter.

## Building an App

In this exercise we will examine some functionality of the **Hyperledger** **fabric**.

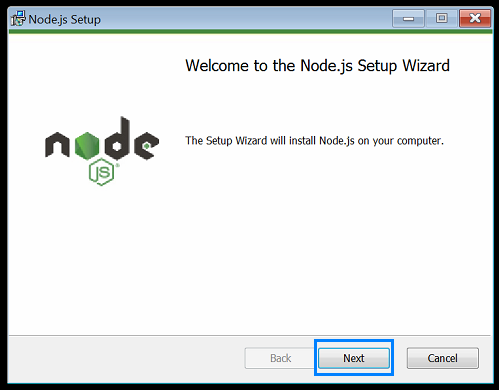
For this exercise we will be needing **Node.js** and **npm** package manager. To install them we need to access the following link:

<https://nodejs.org/en/download/>

And then click[[5]](#footnote-5)



After the installer is download **run** it. You should be represented with the following window:



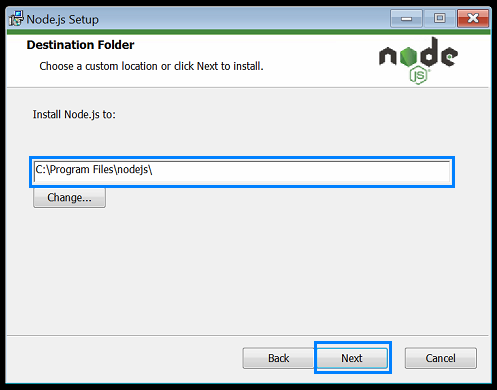
Click the **[Next] button**.

The next window displays the **Terms and Conditions**:



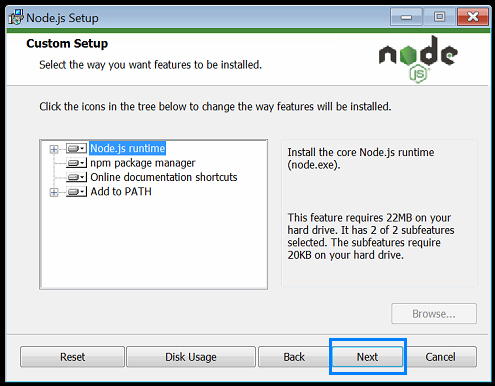
Select that you agree with them and click the **[Next] button**.

Next you should choose the path where you want **Node.js** installed:



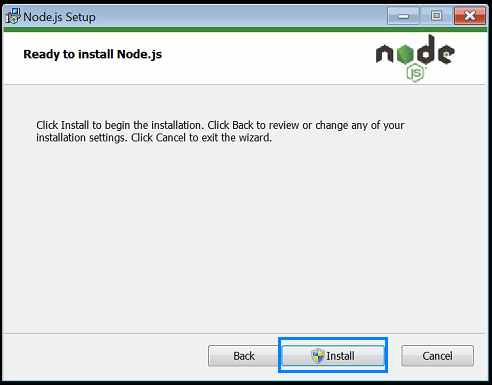
After you have made your choice click on the **[Next] button**.

Next you are presented with some options concerning what parts of **Node.js** to install.



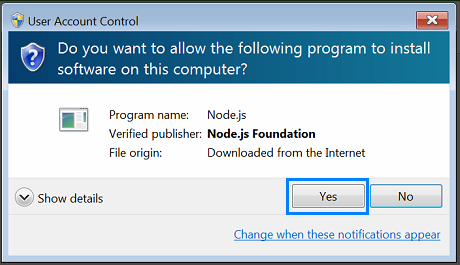
Simply click the **[Next] button**.

Then you are presented with the following window:



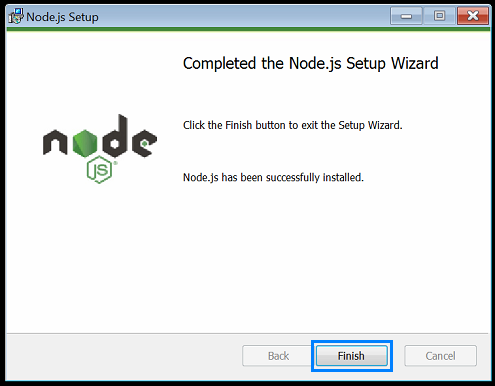
Click the **[Install] button**.

You may be shown some windows that resemble the one in the following screenshot:



Click the **[Yes] button** every time.

After the installation is completed, you should be presented with the following window:

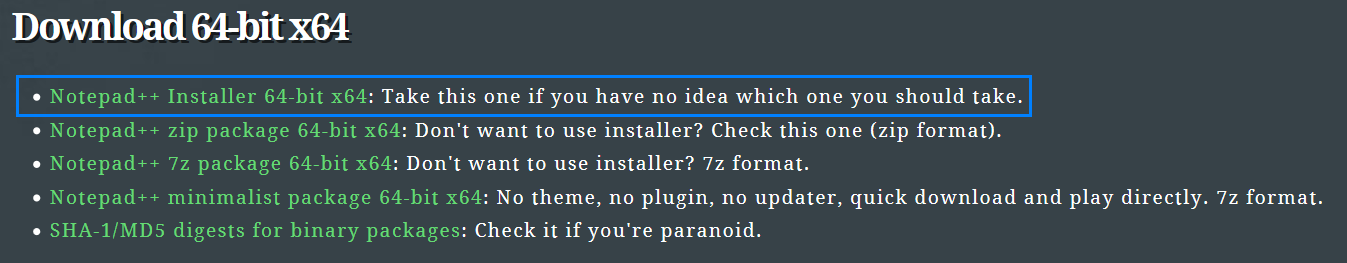


Click the **[Finish] button**.

To download **Notepad++** go to the following link:

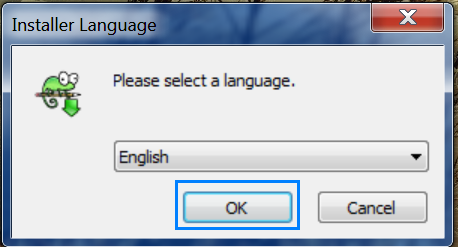
<https://notepad-plus-plus.org/download/v7.5.4.html>

Scroll down until you reach the bottom of the page. There you should see:



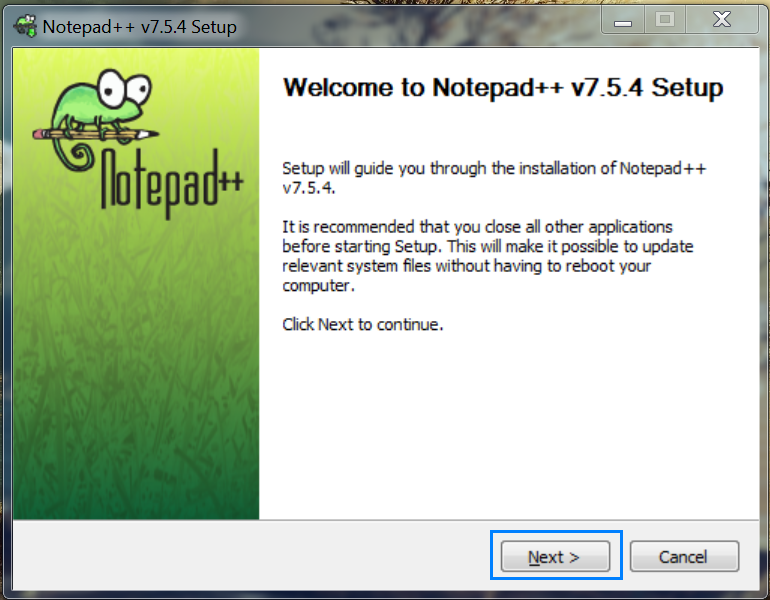
You may choose the format that you like best, but the safest bet is the one in the blue rectangle. Save the file and run it.

You will be prompted to choose a language:



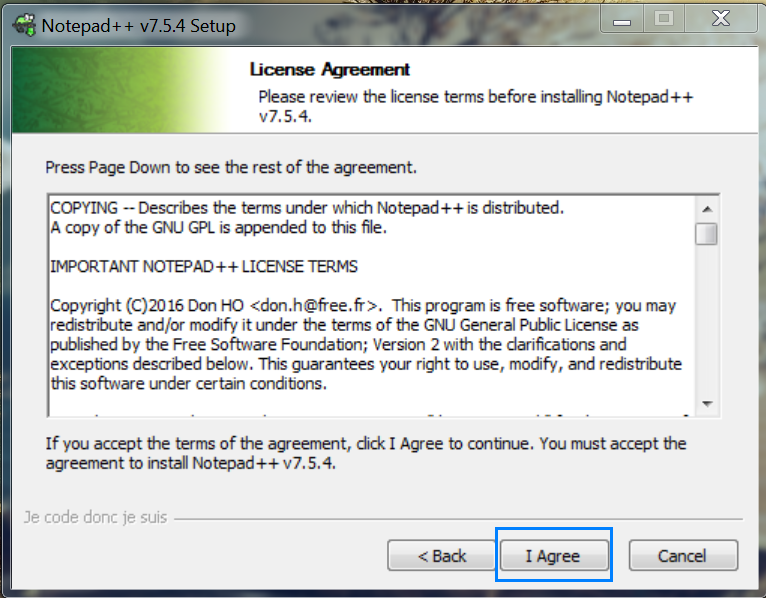
After you have made your choice click the **[OK] button**.

The next window presents you with some information about the installation:



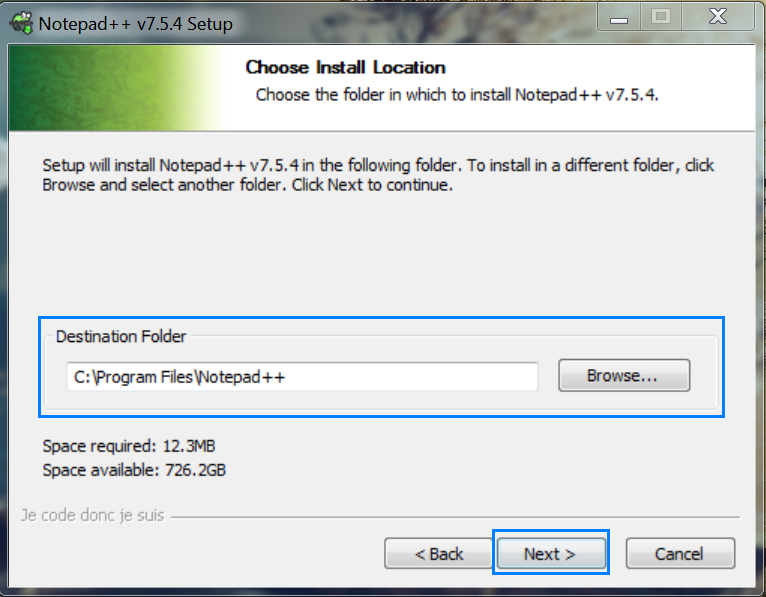
Click the **[Next] button**.

Following will be the **License Agreement**:



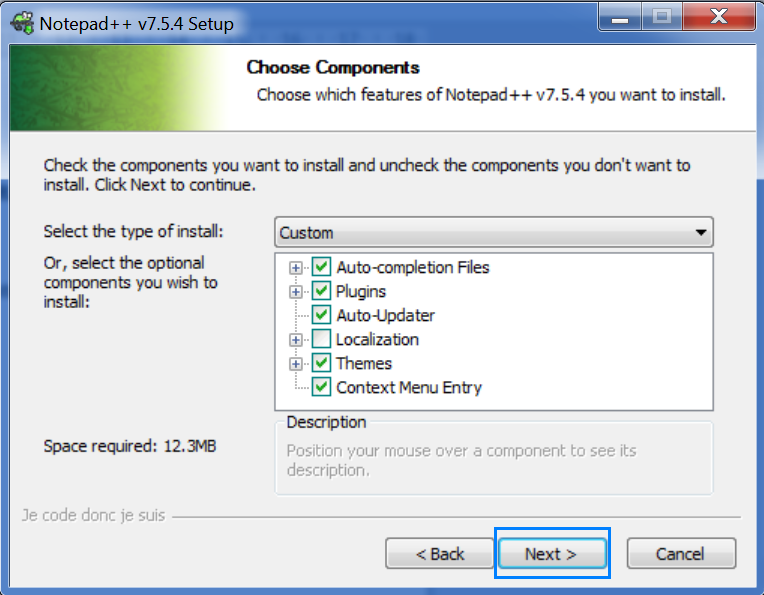
Click the **[I Agree] button**.

Next you must choose where **Notepad++** is to be installed:



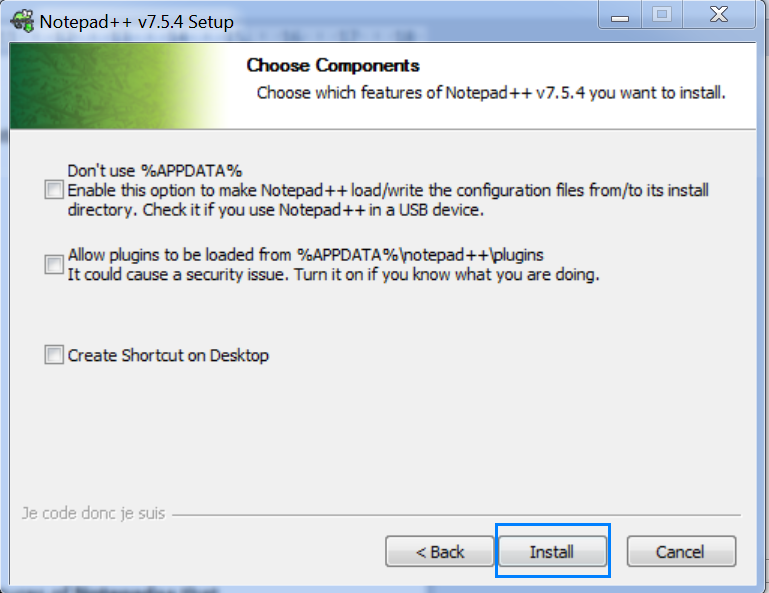
Click the **[Next] button**.

After you have made your choice you will be presented with the features of **Notepad++** that are going to be installed:



Click the **[Next] button**.

Now you are going to be presented with some additional options:



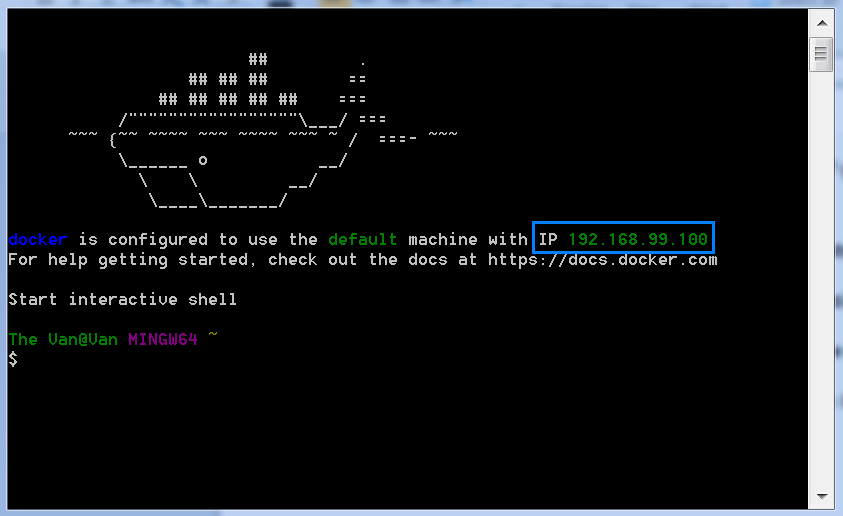
Simply click the **[Install] button**.

When the installation is completed you will see the window, depicted in the next screenshot:



Notice that you gen run **Notepad++** tight away if you decide. Otherwise, uncheck the box. Click the **[Finish] button**.

Now run the **Docker Quickstart Terminal**. When it starts, you will notice that a certain **IP address** is displayed:



Copy it, because we will be needing it soon.

After the installation is completed we need to do some changes to the files that we have cloned from the **Hyperledger fabric repo**. Navigate to the folder **fabric-samples**. In it you will find a folder with the name **fabcar**. In it there are some files that we need to edit.

The first file that we will open with **Notepad++** (or the editor that you are going to be using) is **registerUser.js**. In it you should search for **"**[**http://localhost**:7054](http://localhost:7054/)". When you find this, you should replace **ONLY** **"localhost"** with **the IP address that was displayed by the Docker**. Here is an example:



Do this for every **"localhost"** in the file.

Next, repeat this for the files:

* **query.js;**
* **enrollAdmin.js;**
* **invoke.js.**

Now you should open **PowerShell as an administrator[[6]](#footnote-6)**. In it write the following command:

|  |
| --- |
| npm install –global windows-build-tools |

and press **[Enter]**. The installation will take some time so be patient.

When you are done go to the **Docker** and navigate to the **fabcar** (it is a subfolder of the **fabric-samples** folder). In it write the following commands:

|  |
| --- |
| npm install |
| npm install grpc |

Now, before continuing let’s clean our working environment. To achieve this, first we write in the **Docker** the following command:

|  |
| --- |
| docker rm –f $(docker ps –aq) |

It removes all currently active containers. Next, write:

|  |
| --- |
| docker network prune |

and click **y** when prompted.

Having installed all prerequisites and having cleaned our workspace we write the following in the **Docker**:

|  |
| --- |
| ./startFabric.sh |

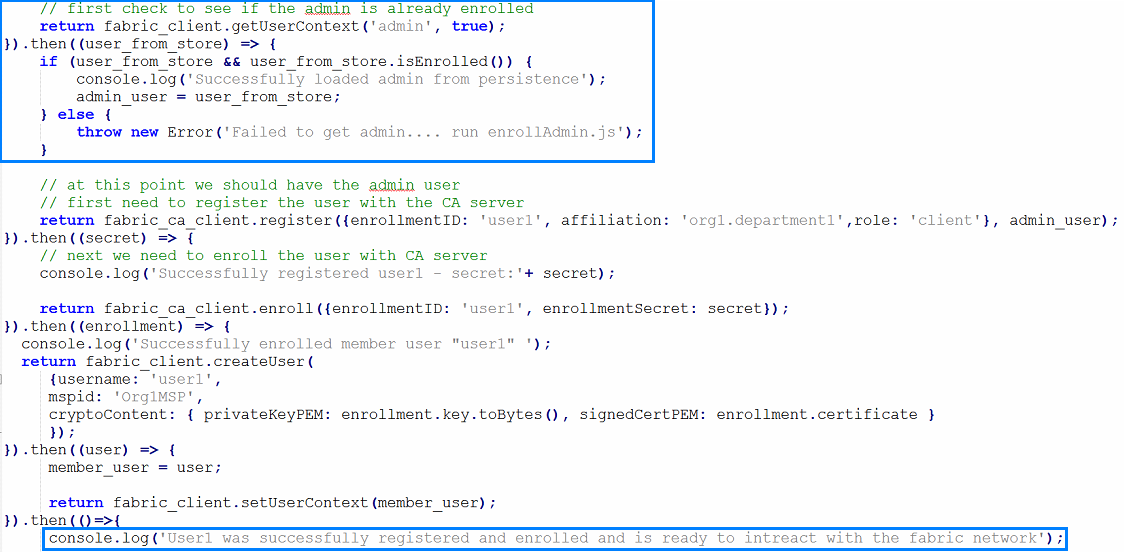
This command creates the environment in which we will take a closer look at how **Hyperledger fabric** can be used.

Now that we have the context, we need to have an **admin** that is going to add a user that is going to use the network. To do this we write the following:

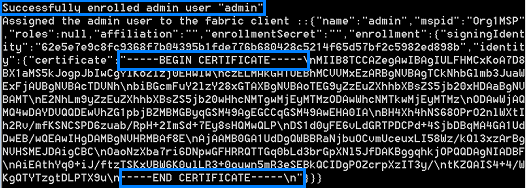
|  |
| --- |
| node enrollAdmin.js |

After the execution of this command a new folder in **fabcar** with the name **hfc-key-store** is created. In it resides all the necessary information needed for the users to access our app’s functionality.

Even though it is logical firstly to register the **admin** you may wish to examine the **registerUser.js** file and ascertain that the code necessitates it with the following section:



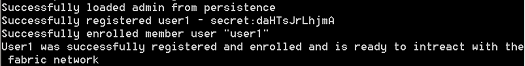
When the operation is successfully terminated you should be represented with an output, similar to the one in the following screenshot:



Now the folder **hfc-key-store** should have three files in it concerning only the **admin**. There you may see the output from stored in a file with name **admin**. Also, in this directory you can find the **admin’s** keys, both public and private. Now to move on. Write the following in the **Docker**:

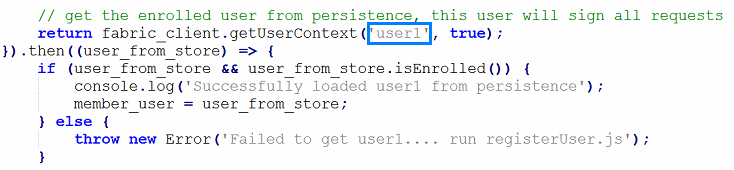
|  |
| --- |
| node registerUser.js |

The output from this command should look similar to the next screenshot:



If you examine the folder **hfc-key-store** you will notice that **three** new files have been created concerning **User1**. Feel free to examine them.

Now we are able to do some queries about the information held in the ledger. Now, the first question that pops to mind is “Who will do the querying – the admin or the user1?”. To answer this question we need to examine the file **query.js**. In it you will find the following code:

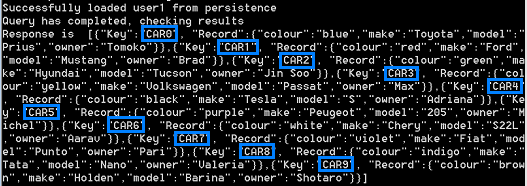


So now we know…

Let’s execute the fail through the following command:

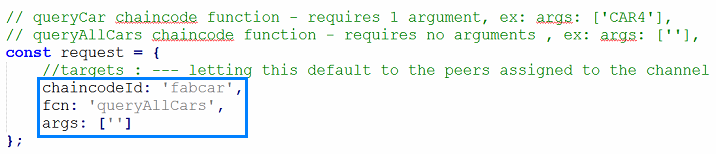
|  |
| --- |
| node query.js |

The output should resemble the following screenshot:



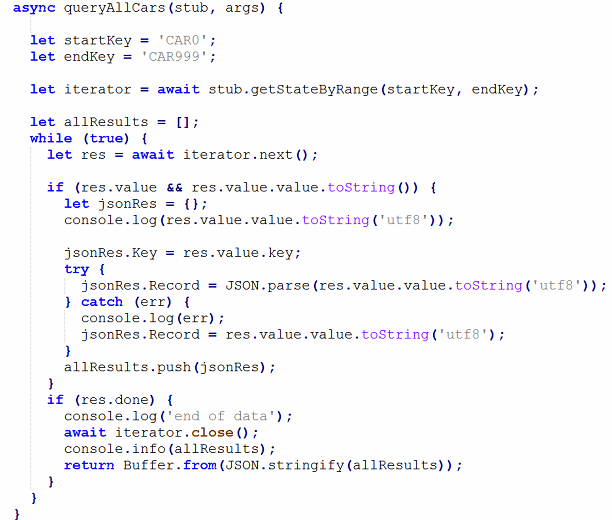
What we received is information concerning 10 cars – their make, model, owner, color.

To see how exactly this happened we need to examine once more the **query.js** file. In it you will find these lines of code:



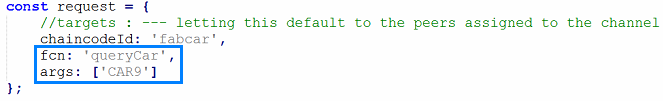
When we executed the file the **fabcar chaincode** was called on the peer and the function **queryAllCars** from it was executed without any arguments.

If you navigate to the **chaincode** subfolder of the **fabric-samples** folder, you will notice in it a folder with the name **fabcar**. In it are residing two folders, one concerning **JavaScript** code, the other one – **Go** code. We will examine the **fabcar.js** file that is inside the **node** folder. In it reside the following lines of code:



From it you can see that the total amount of cars that can be queried is set to 1 000.

Now, let’s try to query a single result. To achieve this, we need to open the **query.js** file and find the line of code that invokes the **queryAllCars** function and make some changes:



After changing the name of the function and passing to it a parameter (CAR9) we must save the file. Then write the following command:

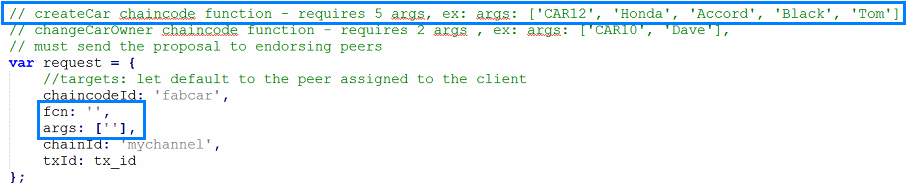
|  |
| --- |
| node query.js |

Now the output should look like:



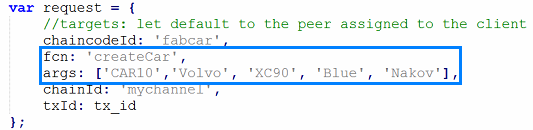
If you compare the result from this command with the one that printed all cars, you will undoubtedly notice that the information concerning CAR9 is the same, as is expected. This way you can query all cars one by one if need be.

Let’s now add a car to the ledger. To achieve this we will need another **JS** file with the name **invoke.js**. In it we must find the following piece of code:



The first thing we need to notice is that if we are about to create a car we must pass to the function **5 arguments**. The first one is the key through which the car will be identified. Next comes the **make, model,** color and owner.

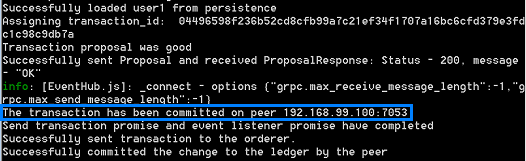
The next thing is that gets our attention are the empty fields for the function and the said arguments. There we should write, for example, the following:



Here we have added the name of the function and the necessary information as parameters. We save the file and what is left is run the file:

|  |
| --- |
| node invoke.js |

The output should look like this:

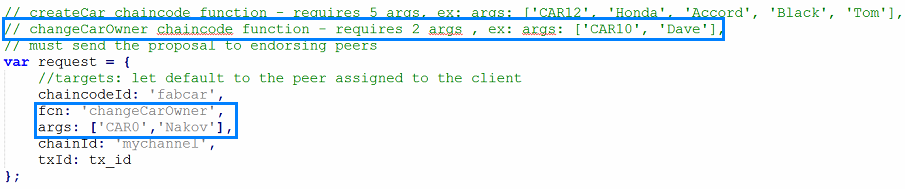


Of all the presented information we are concerned with the one that is in the blue rectangle.

Now we should check whether the car has been indeed added. To do this we open the **query.js** file and look for the lines of code that invoke the function returning the information for **CAR9**. We simply change this parameter with **CAR10**, then save the file and run it. The output should be:

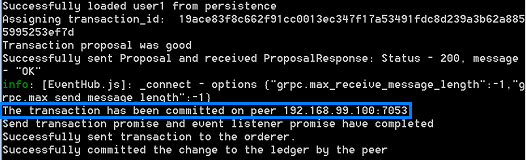


Great! We have successfully added a new car. Now, let’s assume that Nakov buys another car from the ones existing. Hence, we need to change its owner. To achieve this we need to open the file **invoke.js** and look for the lines of code invoke the function **createCar**. We need to change the name and pass the parameters associated with the new function:



This time we only need two parameters – which car has changed the owner and the name of the new owner.

The output from the updated **invoke.js** file is:



Everything seems to have gone smoothly. The only thing left is to verify that the change has indeed taken place. You should be able to do this on your own.

Congratulations! You have successfully completed this part of the exercises!

## Chaincode with Go Programming Language Exercise

Firstly, for this exercise we will need to install **Go**. Go to the following link:

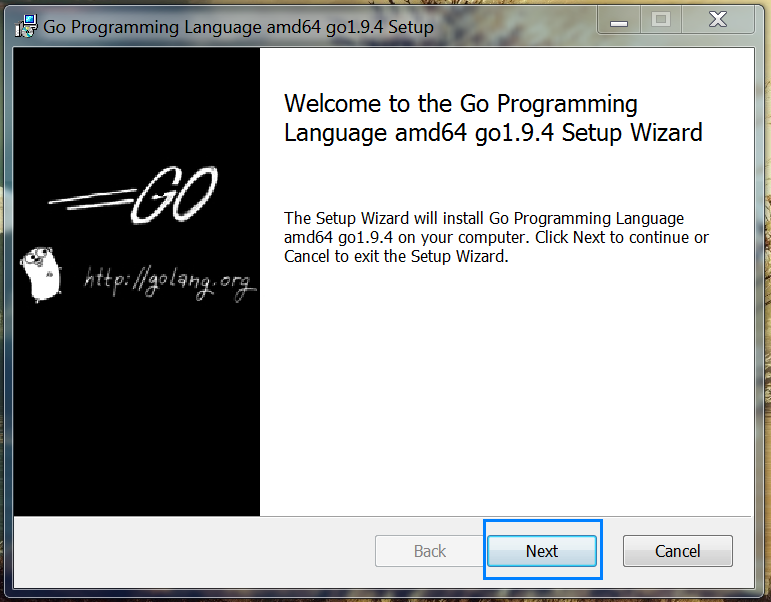
<https://golang.org/dl/>

In it, look for the section entitled **Featured downloads** and click on **Microsoft Windows[[7]](#footnote-7)**:



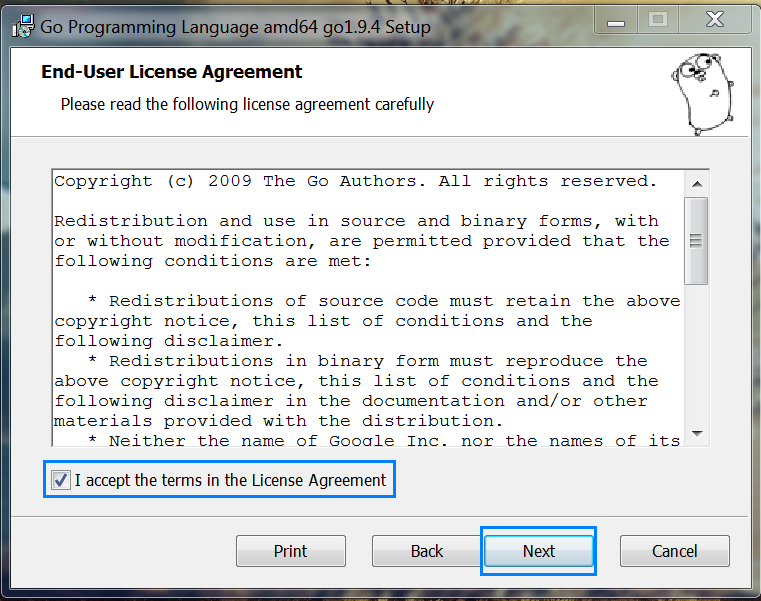
Download and run the installer.

You will be represented with the following window:



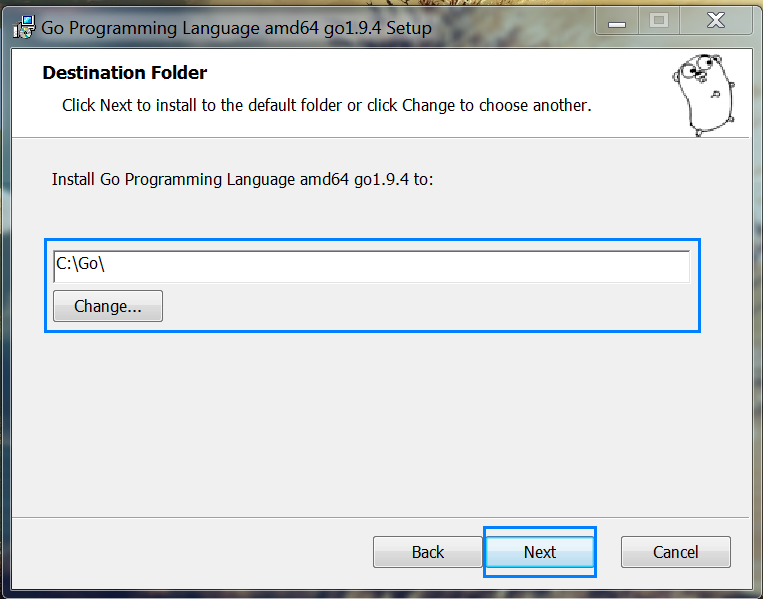
Click the **[Next] button**.

Afterwards, you will be shown the **Terms and Conditions**:



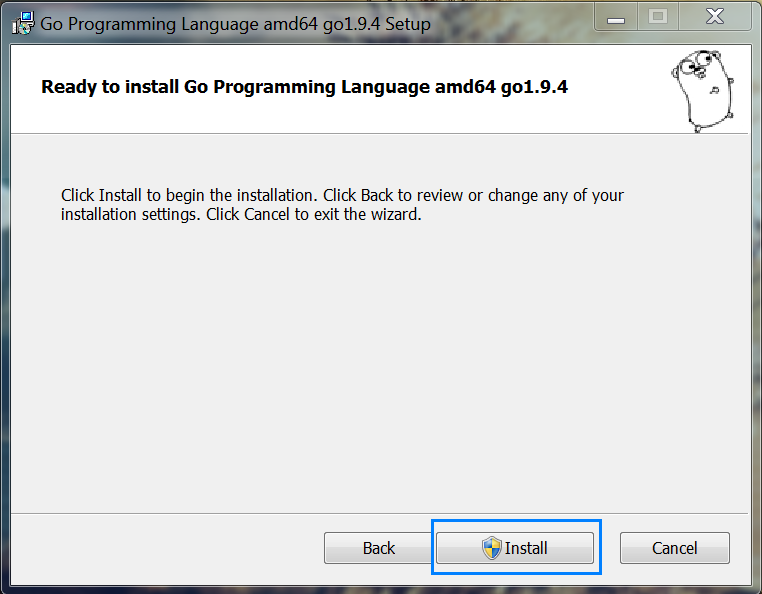
Notice that by default the checkbox that you agree with them is selected. Click the **[Next] button**.

Now you should select where **Go** is to be installed:



Choose where you wish **Go** to reside and click the **[Next] button**.

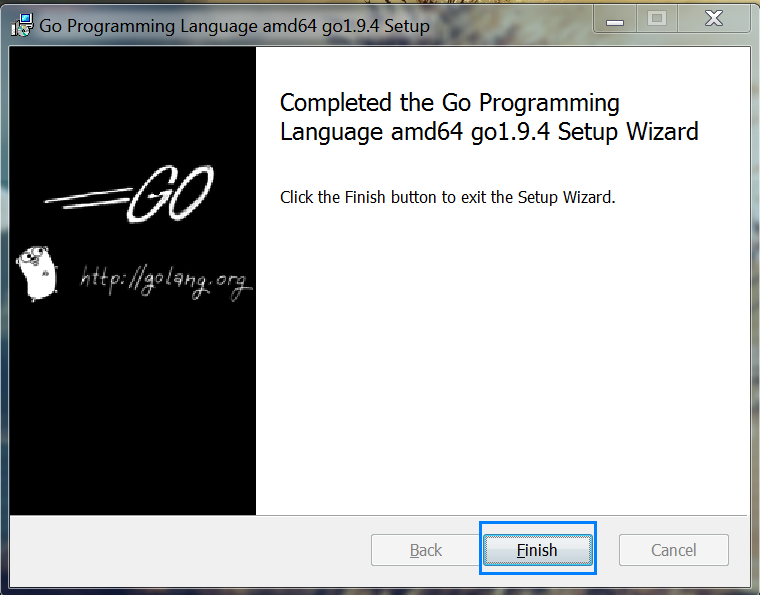
In the following window you are informed that the installation is about to be launched:



Click the **[Install] button**.

During the installation you may be asked to affirm the installation of a certain modules. Click the **[Yes]** **button**.

When the installation is completed you should see the following window:



The next step is to add the **Go\bin** to the **PATH Environment variables[[8]](#footnote-8)**.

When we have done this, we need to open the **Docker** and write the following command:

|  |
| --- |
| go env GOPATH |

This will show us where our **workspace should** be located. If the displayed path does not exist, create it - we will be placing our files there.

Now we are ready to begin!

In it create a subfolder with the name **src**, i.e.:

C:\Users\%USERNAME%\go\src

In it create another folder with the name **sacc[[9]](#footnote-9)**.

We will be using the **Notepad++** (any other environment that you are comfortable with will suffice). Create a file with name **sacc** and extension **go** and place it inside the **sacc** folder, i.e.

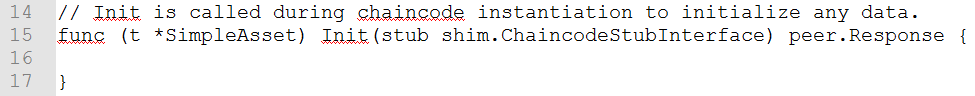
C:\Users\%USERNAME%\go\src\sacc.go

This is the file in which we will write our chain code.

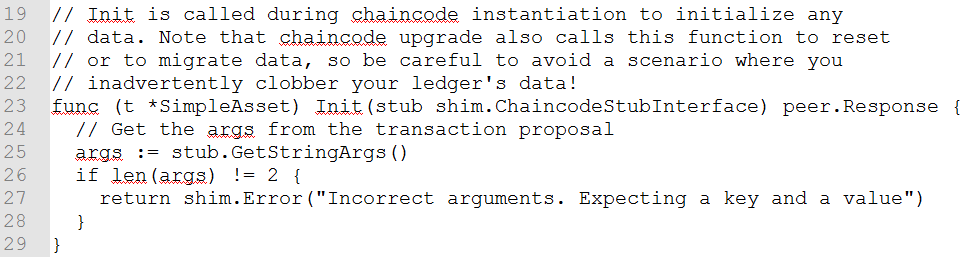
First, let’s add the go import statements for the **necessary dependencies** for our chaincode. We’ll import the chaincode shim package and the peer protobuf package. Next, let’s add a **struct** SimpleAsset as a receiver for Chaincode shim functions.



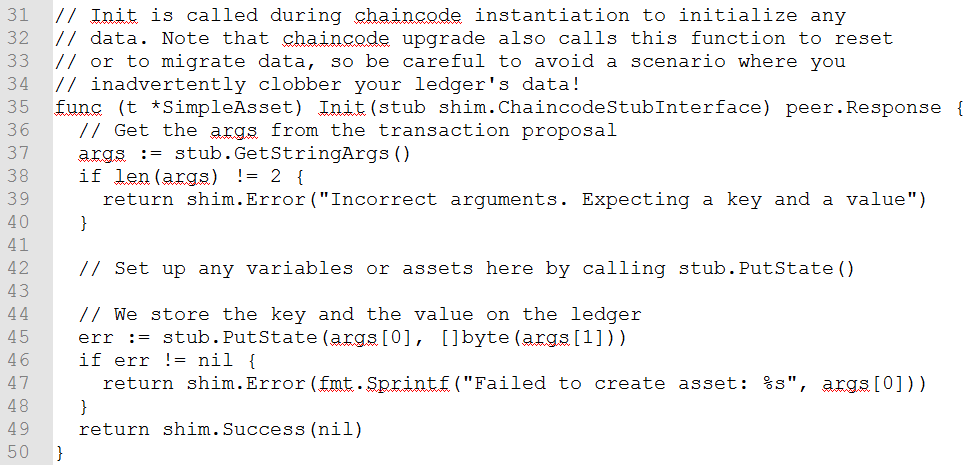
Next, we’ll implement the **Init** function.



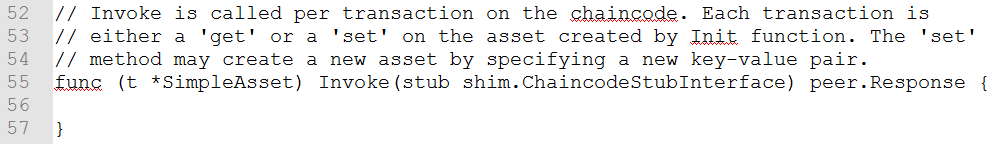
Next, we’ll retrieve the arguments to the **Init call** using the **ChaincodeStubInterface.GetStringArgs function** and check for validity. In our case, we are **expecting a key-value pair**.



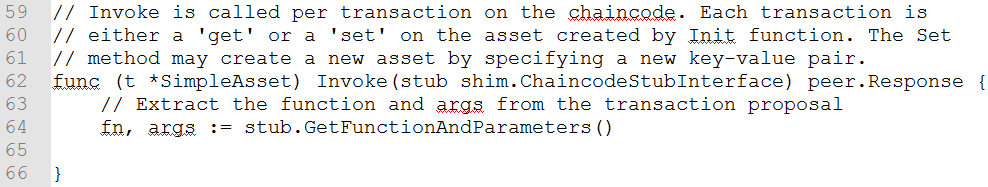
Next, now that we have established that **the call is valid**, we’ll store the initial state in the ledger. To do this, we will call **ChaincodeStubInterface.PutState with the key and value passed in as the arguments**. Assuming all went well, **return a peer.Response** object that indicates the initialization was a success.



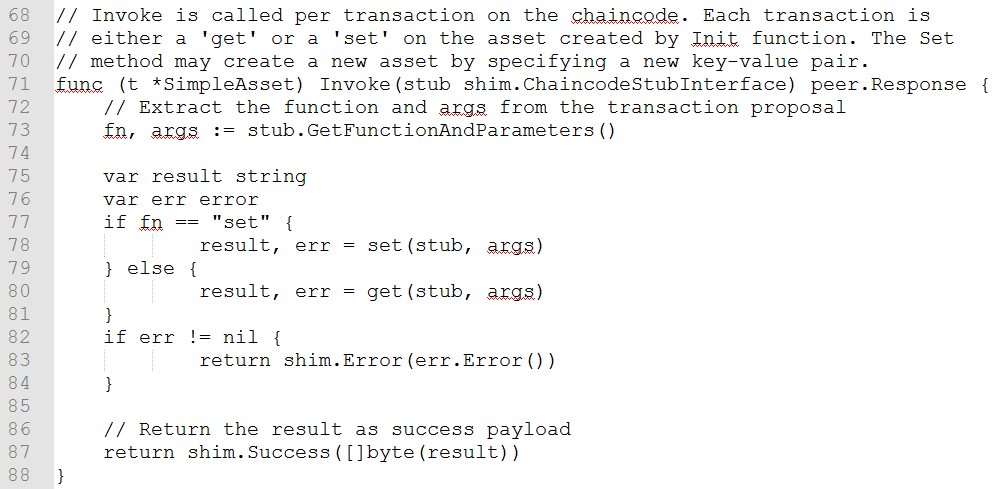
Now, let’s add the **Invoke function’s signature**.



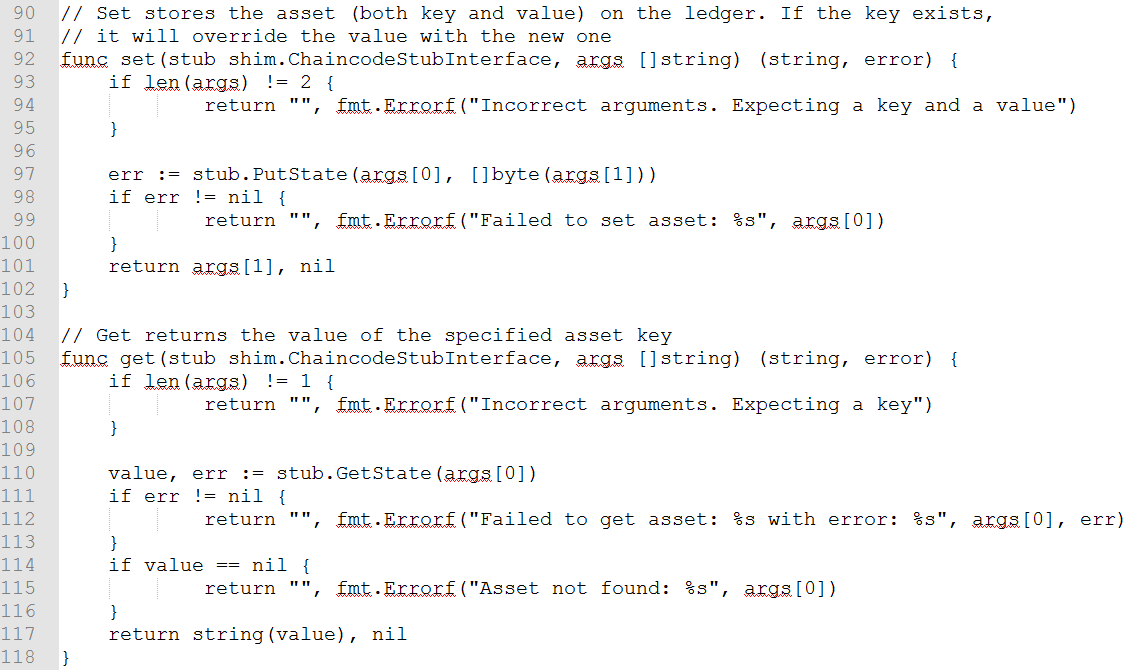
As with the **Init function** above, we need to extract the arguments from the ChaincodeStubInterface. The Invoke function’s arguments will be the **name** of the chaincode application function to invoke. In our case, our application will simply have **two functions: set and get**, that allow the value of an asset to be set or its current state to be retrieved. We first call ChaincodeStubInterface.GetFunctionAndParameters to extract the function name and the parameters to that chaincode application function.



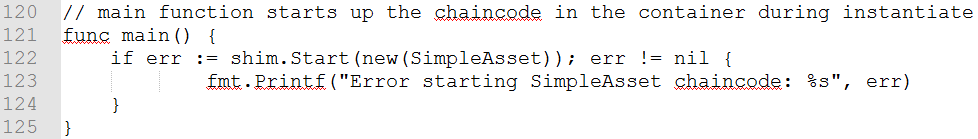
Next, we’ll **validate** **the function** name as being either set or get, and **invoke those chaincode application functions**, **returning** an appropriate response via the **shim.Success or shim.Error** functions that will serialize the response into a gRPC protobuf message.



As noted, our chaincode application implements two functions that can be invoked via the Invoke function. Let’s implement those functions now. Note that as we mentioned above, **to access the ledger’s state, we will leverage the ChaincodeStubInterface.PutState and ChaincodeStubInterface.GetState functions** of the chaincode shim API.



Finally, all that is left is writing the **main function**, which will call the shim.Start function.



Now let’s compile the chaincode!

In the **Docker** navigate where the **sacc.go** file resides and write the following command:

|  |
| --- |
| go get –u –-tags nopkcs11 github.com/hyperledger/fabric/core/chaincode/shim |

When it is completed, you will notice that a folder with the name **github.com** has been created next to the **sacc** folder. Going back to the **Docker** write:

|  |
| --- |
| go build –-tags nopkcs11 |

After it is completed, we need to open **three** new terminals. Each and every one of them has a function to perform.

With the first navigate into the **fabric-samples** folder. In it look for a folder with the name **chaincode-docker-devmode** and enter it. Then write the following command[[10]](#footnote-10):

|  |
| --- |
| docker-compose –f docker-compose-simple.yaml up |

The above starts the network with the SingleSampleMSPSolo orderer profile and launches the peer in **“dev mode”**. It also launches two additional containers - one for the chaincode environment and a CLI to interact with the chaincode. The commands for create and join channel are embedded in the CLI container, so we can jump immediately to the chaincode calls.

Now open a second **Docker** and again navigate to the **chaincode-docker-devmode** folder. There execute the following command:

|  |
| --- |
| docker exec –it chaincode bash |

After executing the command, you should see the following line:



Next, execute the command:

|  |
| --- |
| cd sacc |

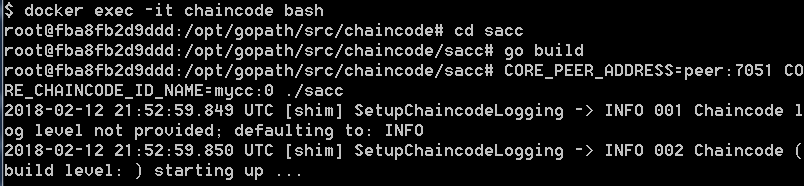
and right after it:

|  |
| --- |
| go build |

After the built is successful write the next command[[11]](#footnote-11):

|  |
| --- |
| CORE\_PEER\_ADDRESS=peer:7051 CORE\_CHAINCODE\_ID\_NAME=mycc:0 ./sacc |

You should see something similar to the next screenshot:



Now open the third **Docker** and one again navigate to the **chaincode-docker-devmode** folder. In it write the following command:

|  |
| --- |
| docker exec –it cli bash |

You should see the following:



Now write the command:

|  |
| --- |
| peer chaincode install –p chaincodedev/chaincode/sacc –n mycc –v 0 |

After the successful execution of the command, write the next one:

|  |
| --- |
| peer chaincode instantiate -n mycc -v 0 -c '{"Args":["a","10"]}' -C myc |

Now let’s write the command:

|  |
| --- |
| peer chaincode query -n mycc -c '{"Args":["query","a"]}' -C myc |

You should see that the result is 10:



Now to invoke a change to the value of “a”, that we just instantiated, to “20”. To achieve this, we write the command:

|  |
| --- |
| peer chaincode invoke -n mycc -c '{"Args":["set", "a", "20"]}' -C myc |

Finally, we should query **a** to check whether the value has indeed been changed to 20. We write:

|  |
| --- |
| peer chaincode query -n mycc -c '{"Args":["query","a"]}' -C myc |

Now the returned value should equal 20:



You have completed successfully this series of exercises!

1. The installation is for Windows. If you are on a different OS follow the steps outlined in <https://docs.docker.com/toolbox/> [↑](#footnote-ref-1)
2. If you are using a different OS download the file from the appropriate section. [↑](#footnote-ref-2)
3. Another way is to go to the **Control Panel** and there to find the **System** settings [↑](#footnote-ref-3)
4. If you have used different paths and are having troubles navigating to them remember:

   To go up one folder use **cd ..** (that is the command “**cd**” followed by a single space, followed by two dots);

   If the folder you are trying to navigate two contains a space, encapsulate the name within double quotation marks (i.e. were the name of our folder **Blockchain Hyperledger** we would write **cd “Blockchain Hyperledger”** in order to access the folder). [↑](#footnote-ref-4)
5. If you are using different OS, download the proper file. [↑](#footnote-ref-5)
6. Click on the Windows **[Start menu] button** and in the search bar simply write **PowerShell**. You will be presented with some options – click with the **right** mouse button on the **PowerShell** and choose the **Run as administrator** option. You probably are going to be asked if you will allow the **PowerShell** to make changes to the computer. Click the **[Yes] button**. [↑](#footnote-ref-6)
7. If you are using a different OS choose the appropriate file. [↑](#footnote-ref-7)
8. How this is done was explained in Problem 1. [↑](#footnote-ref-8)
9. It is rather important to use the same naming given in this example. Otherwise you are going to experience some unexpected program behavior. [↑](#footnote-ref-9)
10. If you get an error that says **port is already allocated** most probably you have forgotten to execute the commands from the end of the previous exercise. Now would be a good time to do it. [↑](#footnote-ref-10)
11. If you get an error it is most probably due to setting the wrong port number. If you have a free **Docker**  running switch to it, otherwise open one and write:

    docker ps

    This will give you information concerning on which port to use and write it at the place of the **7051.** [↑](#footnote-ref-11)