SmartContractGen

Blockchain SmartContract generator  
Lionel MARTINEZ, [lionel.martinez@oracle.com](mailto:lionel.martinez@oracle.com)

March 29th, 2019

History

|  |  |  |
| --- | --- | --- |
| 20190429 | Lionel.martinez@oracle.com | Added the “Is it the right tool for me” introduction section, so the audience does not waste time if the tool is not made for them. |
| 20190429 | Lionel.martinez@oracle.com | Added the *history* section |
| 20190502 | [Jens.lusebrink@oracle.com](mailto:Jens.lusebrink@oracle.com) | Added addtl. info |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Table of Contents

[1. Introduction 3](#_Toc8055886)

[Objective 3](#_Toc8055887)

[Is it the right tool for me? 3](#_Toc8055888)

[2. Prerequisites 3](#_Toc8055889)

[Skills 3](#_Toc8055890)

[Environment 3](#_Toc8055891)

[2.1 GoLang installation – Linux 4](#_Toc8055892)

[2.2 Shim Installation 4](#_Toc8055893)

[3. Chaincode generation 5](#_Toc8055894)

[3.1 Overall principle 5](#_Toc8055895)

[3.2 Architecture 5](#_Toc8055896)

[3.3 Generate the Spec-file 5](#_Toc8055897)

[3.4 Generate the SmartContract 8](#_Toc8055898)

[3.5 Deploying the SmartContract 11](#_Toc8055899)

[3.6 Testing deployed SmartContract 15](#_Toc8055900)

[4. Annex: Description of various examples: 16](#_Toc8055901)

[Release 5.7.2.2 16](#_Toc8055902)

[The Spec-File syntax 17](#_Toc8055903)

[Test statements 18](#_Toc8055904)

[Example spec-file.txt 18](#_Toc8055905)

[Some generated code explained 23](#_Toc8055906)

# Introduction

## Objective

SmartContracGen is a tool designed to help SEs in the generation of ChainCode for BlockChain environments, by making it easier, faster, and less error prone. This will speed-up delivery of proof of concepts and can be a significant competitive advantage by cutting down the creation of ChainCode from days to minutes.

The goal of Chaincode is just to write some info into the ledgers, and retrieve them according to criteria, with or without some historical versioning.

This document is designed to:

* Provide step-by-step instructions to install and use the SmartContractGen generator,
* Help running the tool to create SmartContracts for the HyperLedger Blockchain contained in Oracle BlockChain Platform.

## Is it the right tool for me?

* This generator will generate **CRUD-like implementations** of ChainCode, to be deployed on an OBCS channel. That includes Create, Get, GetHistory, and FindByIndexed criterion.
* It also creates APIs following the SQL specification (not documented so far because not handled by the VBCS tool)
* We define a number of entities that contain a key field, some index fields to generate fast searches, and other (flat) fields, then the create/get/getHistory/findByIndexField functions are generated in milliseconds
* A **local test program** is also generated to make sure the ChainCode behaves correctly BEFORE irreversibly deploying it. It also generates a remote test script that will do the very same test, but on the real blockchain after ChainCode deployment.
* A **deploy script** is also generated to help deploying the generated ChainCode to your Channel (right now the deploy script is **not working**, it soon will be, but manual deployment in a console is documented here too)
* Once the ChainCode deployed, the remote test script can be used to validate the behavior and generate a test report that testifies of the right behavior.
* The generated code is in Go Language. The generated scripts are in Bash, and use the CURL client and the Python JSON prettyprinter (python -m json.tool)
* It is very easy to modify the generated code, but in the current release, those modifications will be lost if you re-generate the code: be warned to save your code!!!

# Prerequisites

## Skills

* Basic knowledge of working from the LINUX shell, Blockchain, ChainCode, and GoLang is required
* To have already deployed the ChainCode examples to OBCS.

## Environment

* You need a 64bits Linux platform. The SmartContractGen is written in Go language and the executable is compiled to run on a 64Bits.
* It can be a compute cloud instance, or a virtual machine. The instance used by the author is 4GB RAM and 50 GB Disk, 2 cores
* The GoLang environment will be needed. This document explains how to add the relevant packages. (see <https://golang.org/doc/install>)
* To test locally the generated code, the GoLang Mock Shim environment has to be available, please see <https://docs.oracle.com/en/cloud/paas/blockchain-cloud/user/use-mock-shim-test-chaincode.html> to install it on your 64 Bits Intel Linux Instance

The screenshots in this document assume the use of the Microsoft Visual Code editor, available on Linux. Feel free to use whichever editor suits you.

## 2.1 GoLang installation – Linux

1. Go to [GoLang Website](https://golang.org/doc/install#install) and choose suitable package:

<https://golang.org/doc/install?download=go1.12.1.linux-amd64.tar.gz>

1. Execute as root:

$> tar -C /usr/local -xzf go1.12.1.linux-amd64.tar.gz

1. Execute as normal user (e.g: oracle):

$> export PATH=$PATH:/usr/local/go/bin

$> cd ~

$> mkdir go

1. Follow extra steps described on [GoLang Documentation](https://golang.org/doc/install#testing) to test installation

## 2.2 Shim Installation

1. For installing Shim, you’ll have to clone the GitHub Hyperledger fabric project into the go workplace in the home folder. We assume git tool is available on the machine.

Execute as normal user:

$> cd ~/go/src

$> mkdir -p github.com/ hyperledger

$> cd github.com /hypeledger

$> git clone <https://github.com/hyperledger/fabric.git>

1. Troubleshooting:

If git clone raises the following

error: while accessing <https://github.com/hyperledger/fabric/info/refs>

a possible cause might be the curl package. Try to run this as root:

$> yum update -y nss curl libcurl

# Chaincode generation

## 3.1 Overall principle

Without SmartContractGen, the steps required to implement ChainCode are:

1. Coding the ChainCode and the test code
2. Testing the ChainCode locally and correct the code if needed.
3. Deploy to a ledger
4. Remote test the deployed ChainCode
5. Tell the other members of the team that the work is done

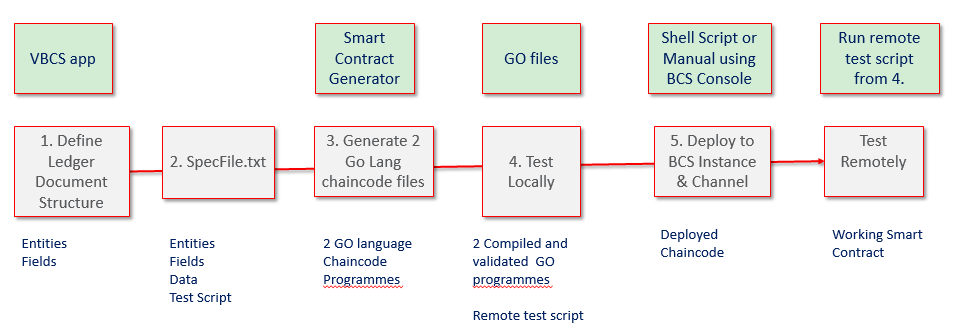
Characteristics of a ChainCode:

* It implements the SmartContract Interface
* It needs an *init* function
* It has an invoke function, with a flat list of string parameters.
* It returns a string (or byte array).

SmartContractGen will take care of most of that work by generating CRUD-like APIs on entities

## 3.2 Architecture

The overall smart contract generation process consists of a number of steps as below.



## 3.3 Generate the Spec-file

The main Smart Contract Generator requires details of the document structure to be created, in the form of a specifically formatted text file called the **spec-file**. This paragraph describes how to create the spec-file using a VBCS GUI app.

1. Access the VBCS Spec-file Builder

The VBCS Spec-file Builder can be accessed in either of 2 ways;

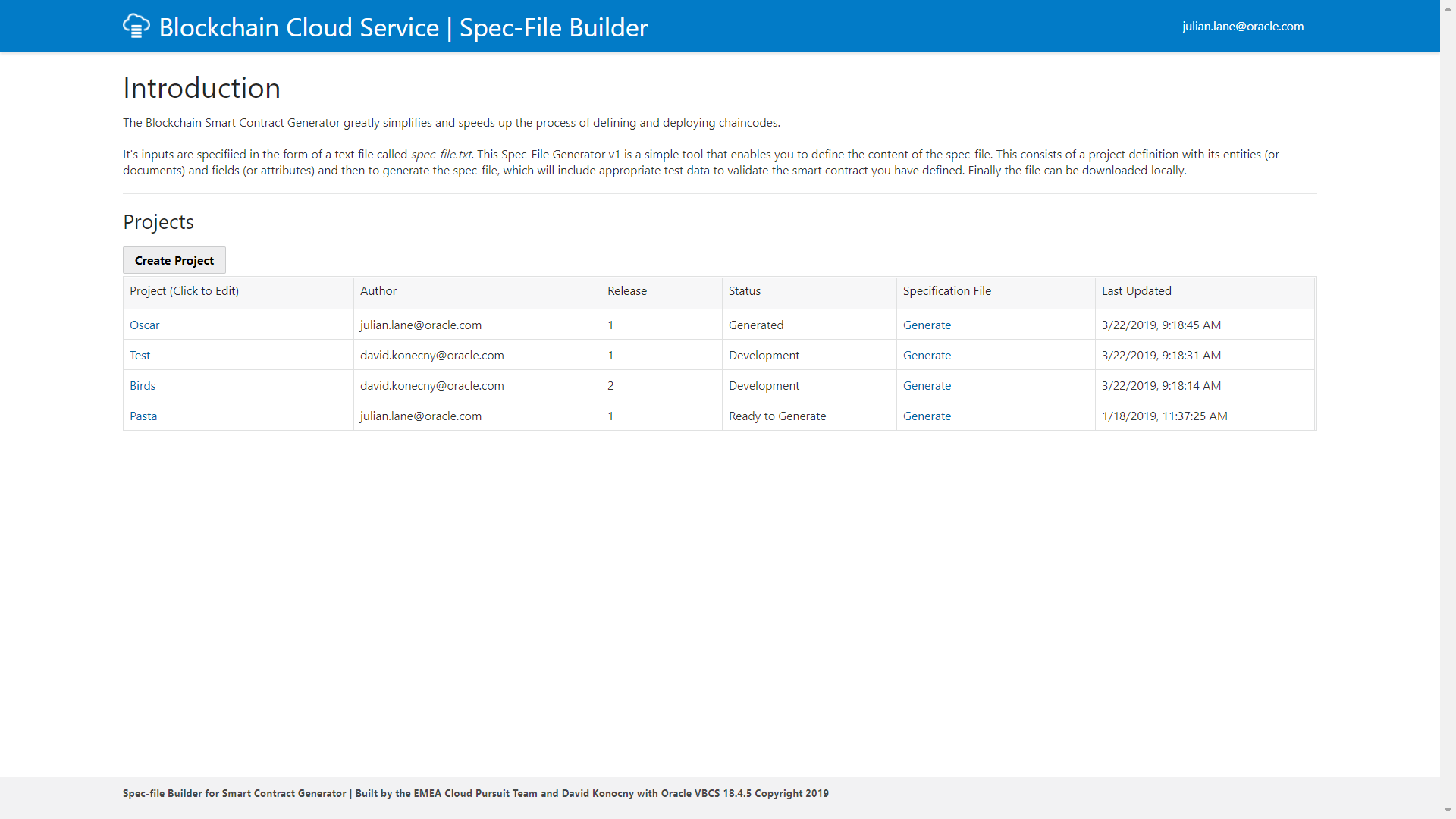
* 1. For anonymous access to the TEST version of the app.

<https://intoraic-cloud01.integration.ocp.oraclecloud.com/ic/builder/rt/BCS_SpecGen/1.0/webApps/specgenv1/>

* 1. OR, with a login to the DEV version of the app EMEA SC OIC / VB environment

<https://intoraic-cloud01.integration.ocp.oraclecloud.com/ic/builder/design/BCS_SpecGen/1.0/preview/webApps/specgenv1/>

This should take you to the home page.



The list of projects may be different from the screenshot.

1. Define high level Project information
   * 1. Click on **Create Project,** enter details. The mandatory fields are preceded with a \*
     2. Click **Save.** This returns you to the home page with project list.

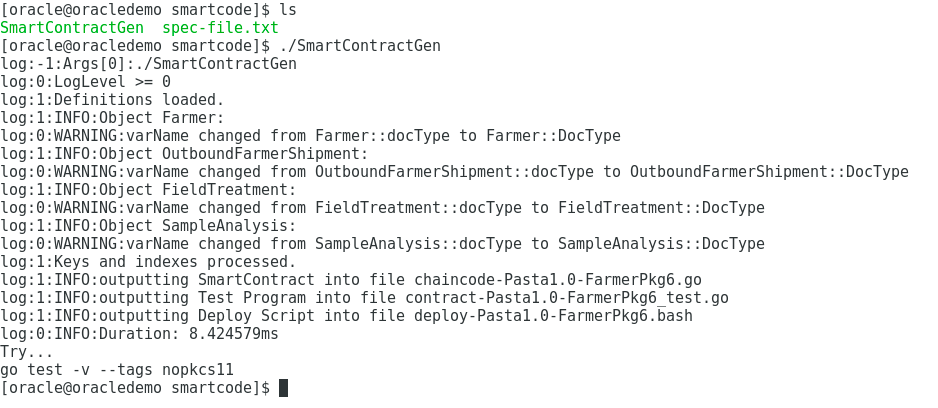
1. Define the Entities (documents) for your ledger
   * 1. From the home page, click on your new project name
     2. Optionally make changes to the project and click Save
     3. Click **Create Entity,** enter the name and any notes and click **Save.** This returns you to the project details page with the new entity listed.
     4. Repeat for each entity required
2. Define the Fields (attributes) for each entity
   * 1. From the project details page click on an Entity name
     2. Optionally make changes to the entity and click **Save,** or **Delete**
     3. Click on **Create Field,** enter the name and select a mode.
     + Note that each entity should have exactly 1 KEY field and any number of IDX and N/A fields
     1. Click **Save.** This returns you to the Entity details page.
     2. Repeat for each field required for this entity
     3. Once created, fields can be edited or deleted by clicking on the field name from the Entity details page.
3. Generate the spec-file
   * 1. From the home page click on **Generate**
     2. If the project has no entities or any entities do not have a key field, an error message will be shown.
     3. Otherwise the generated text is shown. This can be edited but changes made will be over-written next time you generate
     4. If necessary scroll down the page. Click **Download** to save locally.
     5. Repeat after any updates to the project, entities or fields

You now have a spec-file.txt that you need to copy to your working directory

## 3.4 Generate the SmartContract

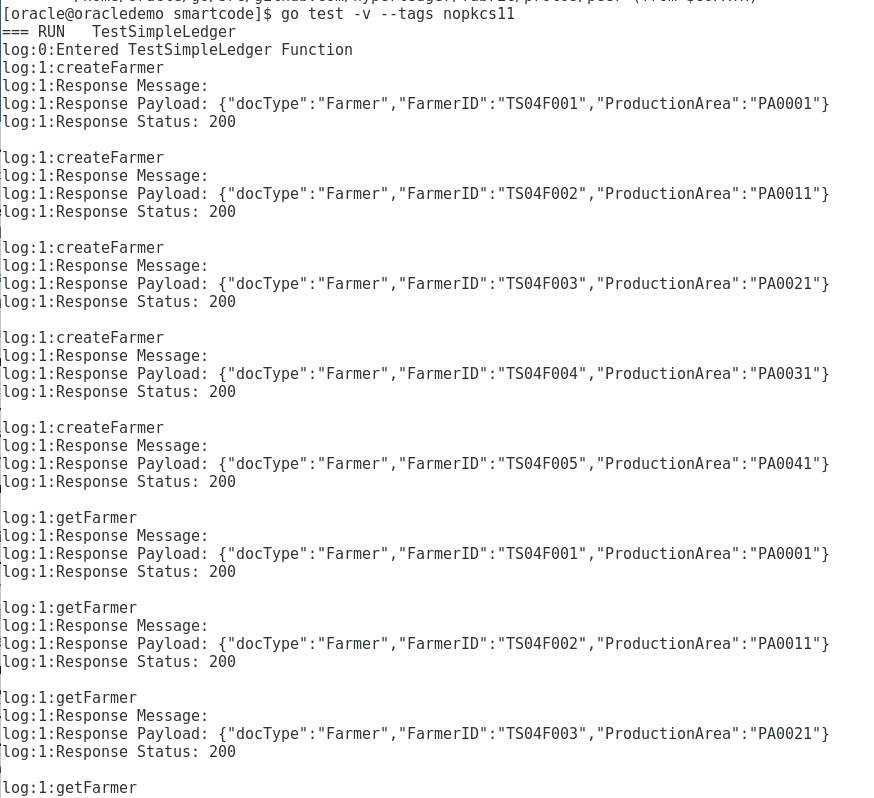
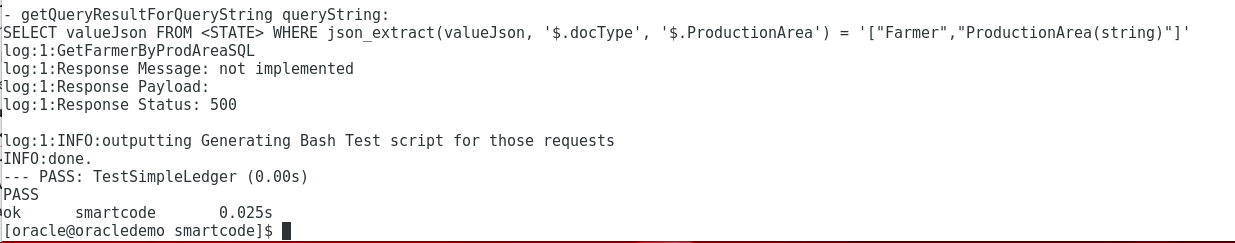
1. Make sure the specfile.txt is in your working directory
2. Download the SmartContractGen package from [here](https://oradocs-corp.documents.us2.oraclecloud.com/documents/link/LD79C850E0E8AF0914FC651EF6C3FF17C1177A968060/fileview/DD31EC3761F2E58B48ECD24AF6C3FF17C1177A968060/_SmartContractGen), in your working directory
3. In a terminal window, invoke the SmarContractGen by typing:

$>./SmartContractGen



1. You can now run the local test.
2. Again, in the terminal type:

$>go test –v –tags nopkcs11

  
.  
.  
.  


You should obtain a set of Response Status 200 for each of the entities created.

Notes:

* *For the getxxxHistory, it's normal to see a response 500 with a Response Message not implemented (the History APIs are not implemented on the Mock Shim)*
* *Same comment for the SQL REST APIS: All the SQL queries will return a Response status 500*

1. After the end of the local test script, you should see the cURL bash test script created. Type the ‘ll’ command to verify:

>$ ll

total 88

-rwxrwx---. 1 root vboxsf 51361 22 mars 14:32 chaincode-Pasta1.0-FarmerPkg6.go

-rwxrwx---. 1 root vboxsf 9252 22 mars 14:32 contract-Pasta1.0-FarmerPkg6\_test.go

-rwxrwx---. 1 root vboxsf 3334 22 mars 14:32 deploy-Pasta1.0-FarmerPkg6.bash

-rwxrwx---. 1 root vboxsf 5259 22 mars 14:30 spec-file.txt

-rwxrwx---. 1 root vboxsf 9853 22 mars 14:32 test-Pasta1.0-FarmerPkg6.bash

>$

This bash file will be used after deployment to do the remote testing.

## 3.5 Deploying the SmartContract

Deployment Steps

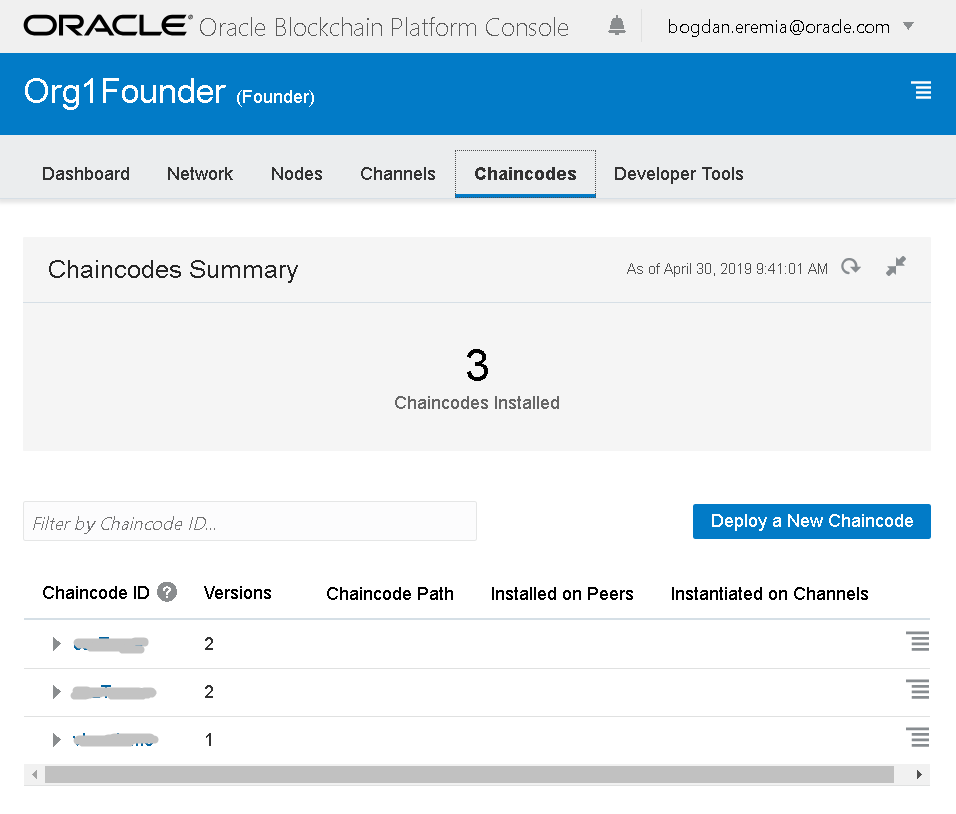
* Create a zip file containing the generated go file:

$ zip chaincode-<ProjectName><ReleaseVersion>-<TestChaincodeName>.zip chaincode-<ProjectName><ReleaseVersion>-<TestChaincodeName>.go

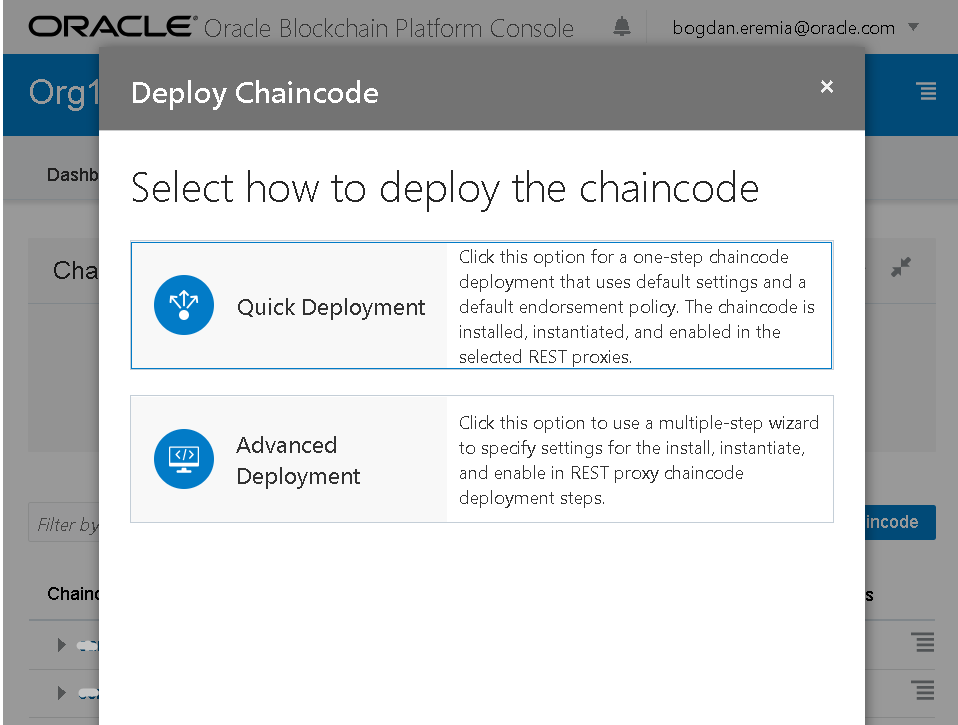
For example:

$ zip chaincode-Pasta1.0-BogdanFarmerPkg.zip chaincode-Pasta1.0-BogdanFarmerPkg.go

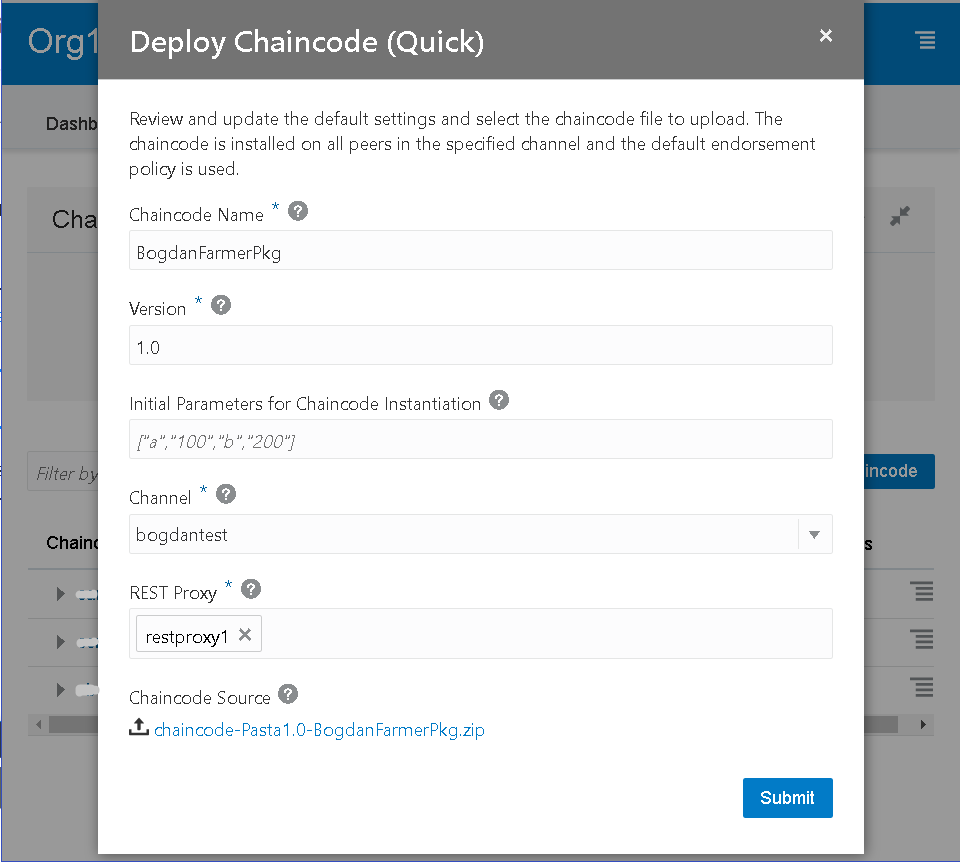
* Go to BC Console, *Chaincodes* tab and *Deploy a New Chaincode*



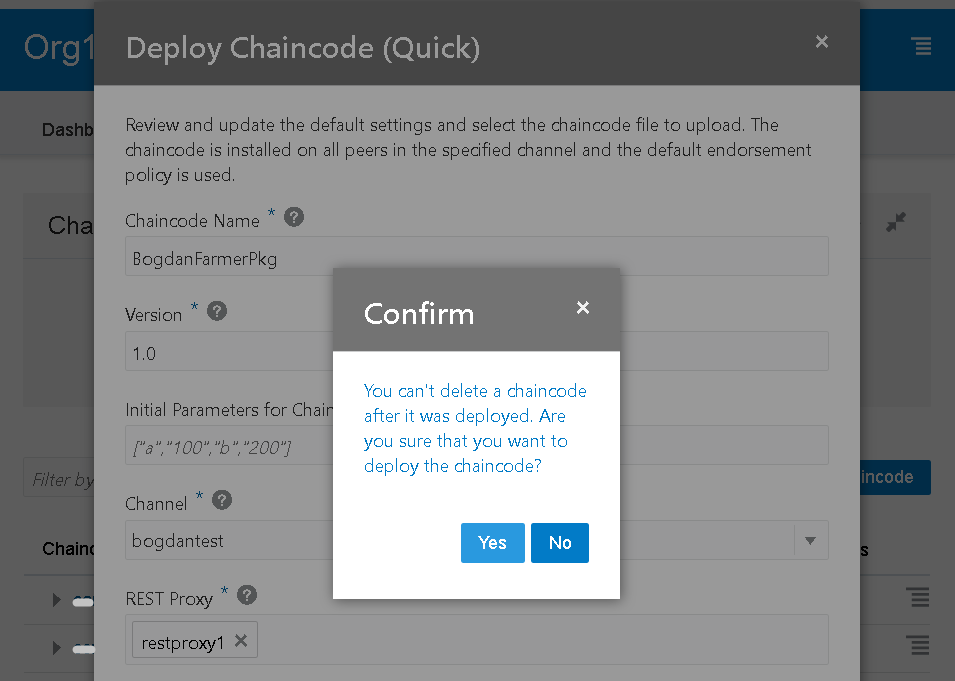
* Select Quick Deployment



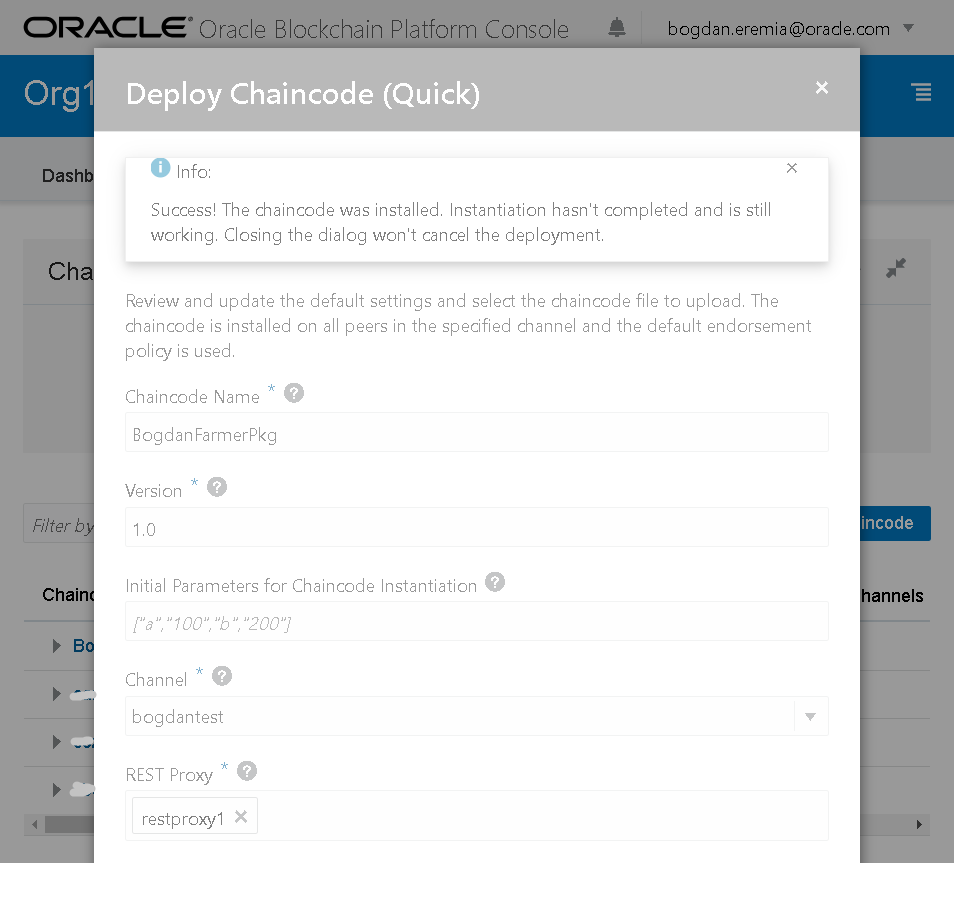
* Fill required information for ChainCode name, version; assign a Channel and a REST Proxy; upload the ChainCode zip file



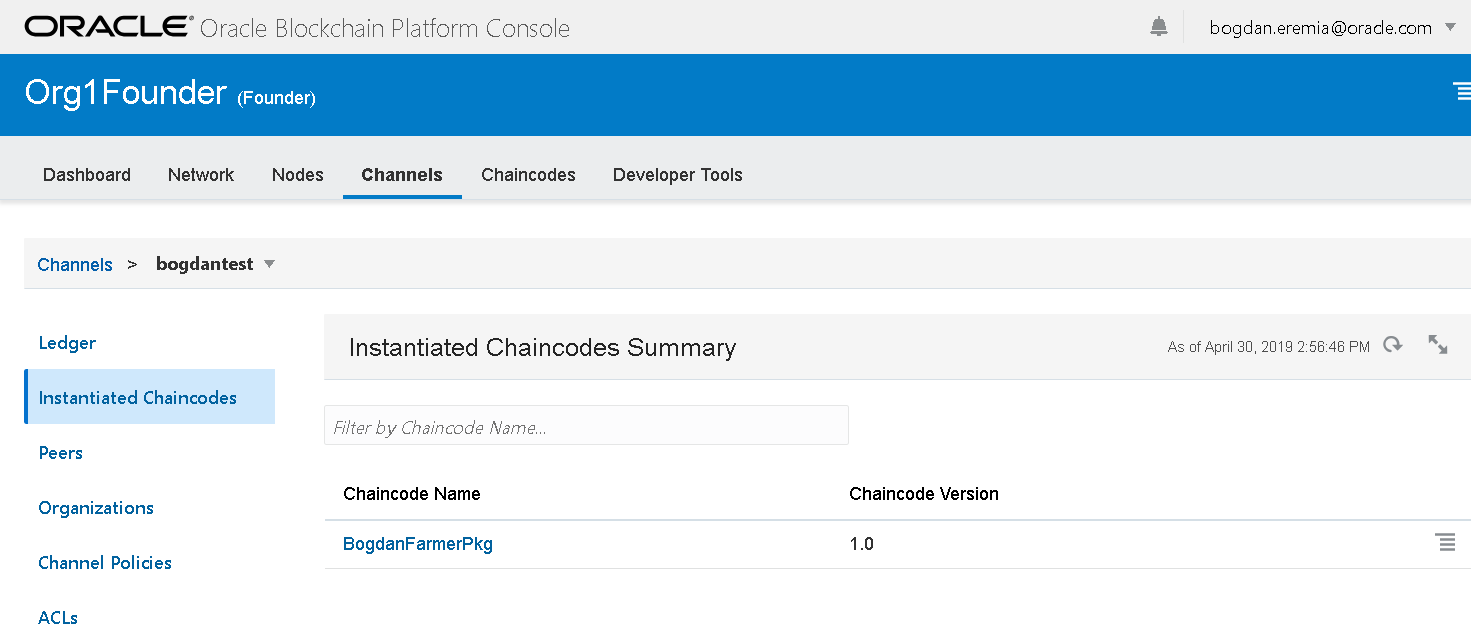
* Confirm deployment on dialog box



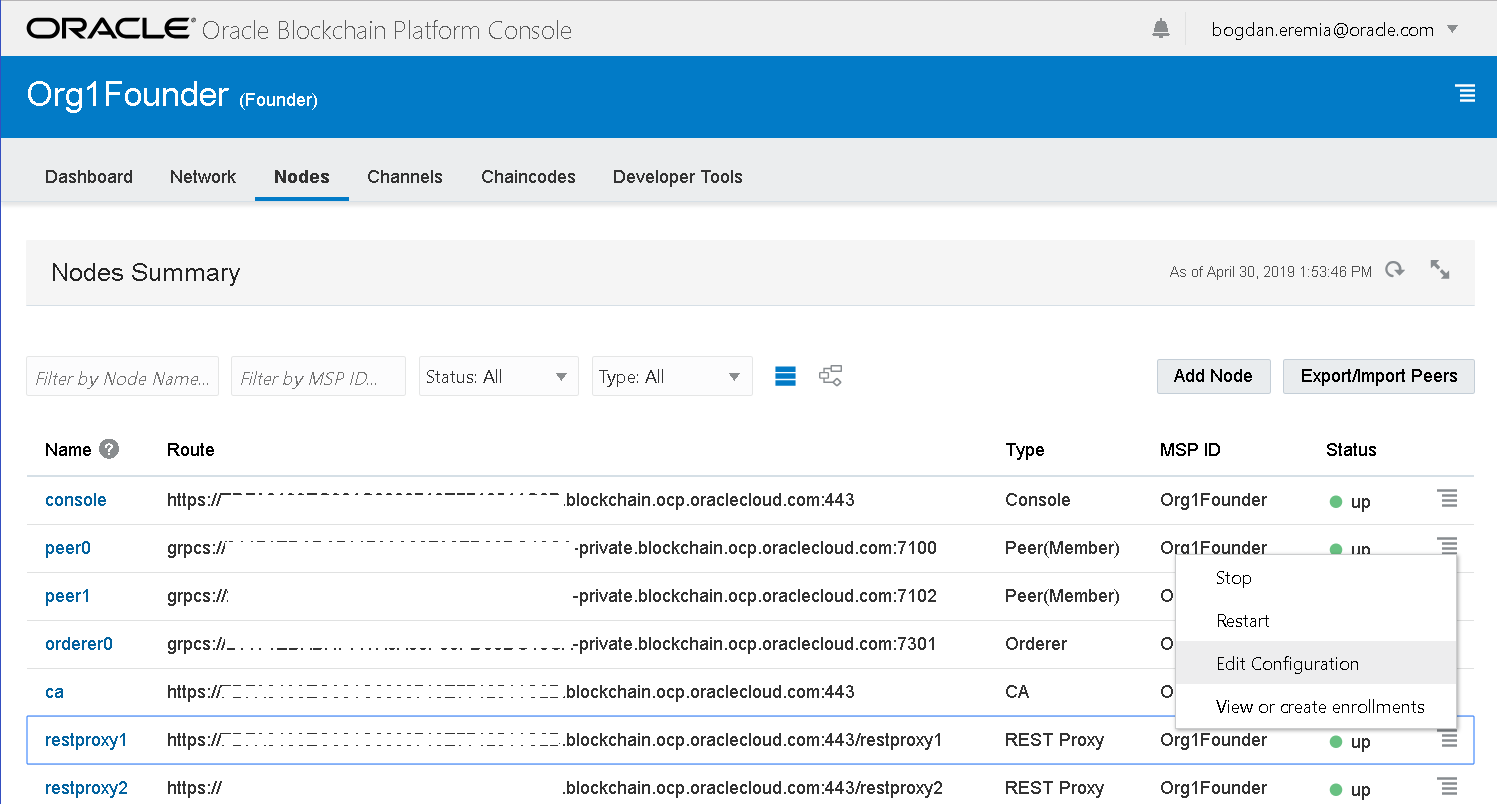
* Instantiation may take some time:



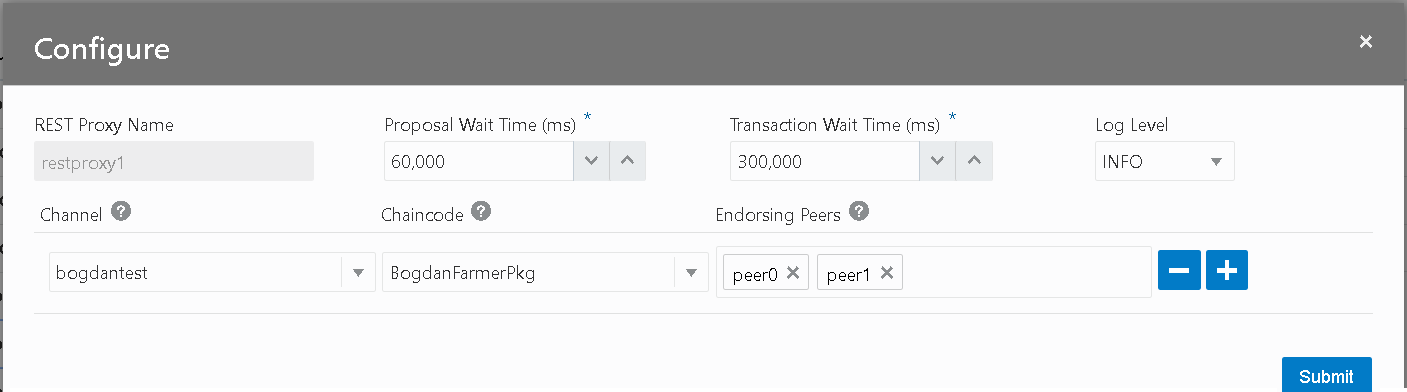
* After a while, you can check that the ChainCode has been instantiated:



* You should check also that the Chaincode has been assigned with selected REST proxy; Click *Edit Configuration* for the REST proxy:



* The new ChainCode should be in the list; if not, add a new row and map the Channel, ChainCode and Endorsing Peers information:



Additional link:

<https://docs.oracle.com/en/cloud/paas/blockchain-cloud/user/typical-workflow-deploy-chaincodes.html> - Typical Workflow to Deploy ChainCodes:

* Using the Quick Deployment or Advanced Deployment
* ChainCode instantiation
* Enabeling ChainCode in a REST proxy
* Upgrading a ChainCode

## 3.6 Testing deployed SmartContract

This will test remotely the insertion, retrieval, history, and search of entities in a Ledger.

Run the test-Pasta1.0-FarmerPkg6.bash

Please refer to the Annex for more details on the execution of this script

# Annex: Description of various examples:

## Release 5.7.2.2

Parsing, more functions generates:

Generate Find<ObjectType>By<Index> function

Force uppercases class and variable names

Generate Get<ObjectType>History function

Consolidate generated code into a unique output source file

Clean logging and printouts

Generate a contract test source

Generate a different contract test that, when it is run, even if manually modified, will generate a <Project><Release>\_cURL.bash program that sends the very same request through REST/JSON

This is v5

Sort the object names list before generating (otherwise order is undetermined)

generate the right output nama as <@TestChainscodeName>.go

5.5.2

support backslash in spec-file.txt for line continuation

5.5.3

generates SQL functions for @SQL:Object:FunctionName:Query: {type:hostVar,in:}\* : NONE where query looks line SQL, hostvariables are denoted in query as :hostVarName only string and int types are supported only 'in' is supported

NONE is the options field, not used so far.

Generating test call for SQL function.

Generating a Deploy script

invoke function in \_cURL\_ bash file corrected

ContentType HTTP header in \_cURL\_ bash file corrected

changed generated filenames

debugging sql

adding the $TEST:Call: and $TEST:Comment: statements to generate tests

use s as \*SmartContract pointer everywhere

## The Spec-File syntax

The tool SmartContractGen uses as input a file named spec-file.txt it parses it, creates an in-memory representation, performs some tests and computations, then generates a number of outputs.

#### Sample SpecFile.txt structure

#### Configuration Statements

Project related information  
Will be used to name the output files, and deploy the chaincode

@Project:Pasta

@Release:1.0

@Author:lionel.martinez@oracle.com

@VerbosityLevel:0

Credential, it is the same for testing as for deploying for the moment

@TestUser:scott

@TestPassword:tiger

Deployment information

@TestChannelName:farmertest001

@TestChaincodeName:farmerpkg1

@TestChaincodeVersion:v1

@TestURLQuery:https://D37FC4BDDE0B4497B97C0BCBE50FD9E9.blockchain.ocp.oraclecloud.com:443/restproxy1/bcsgw/rest/v1/transaction/query

@TestURLInvoke:https://D37FC4BDDE0B4497B97C0BCBE50FD9E9.blockchain.ocp.oraclecloud.com:443/restproxy1/bcsgw/rest/v1/transaction/invocation

#### Entity Statements

An Entity is an object made of fields, and it looks very much like a table in an RDBMS, all elements are flat (the decision to do so was based on the ChainCode interface, where the functions have a flat list of arguments).  
Each field can be a string, int, float, bool.

We are using a JSON format.

The field mode is either:

* KEY, one and only one field must have this /mode/ it will then be the key field, and the get<Entity> and get<Entity>History functions will use it as single argument
* IDX, an index will be created on this field, and a find<Entity>By<Field> function will be created
* N/A, this is just a field.

Dates are currently not defined.

Each entity definition will cause a number of functions to be generated.

create<Entity>(Field0, Field1, …, Fieldn)

get<Entity>(KeyField)

get<Entity>History(KeyField)

For each IDX field:

find<Entity>By<IndexField>(IndexField)

#### SQL Requests

When a search request is needed, but without the need to maintain an up-to-date index, a SQL function can be used.

## Test statements

The generator will always generate in the test script, one call per generated function.  
But you can add your own tests here. The syntax is **very simple** and the generator does not perform any arguments count verification.

## Example spec-file.txt

[bcsdemo@oabcs-vm DEMO]$ ll

total 8

-rwxrwx---. 1 root vboxsf 5259 22 mars 14:30 spec-file.txt

[bcsdemo@oabcs-vm DEMO]$ cat spec-file.txt

spec-file.txt

# \_\_/\\\\\\\\\\\\\_\_\_\_\_/\\\\\\\\\\\\\\\\_\_/\\\\\_\_\_\_\_\_\_\_\_\_\_\_/\\\\\_\_\_\_\_\_\_/\\\\\\_\_\_\_\_\_

# \_\/\\\////////\\\\_\_\/\\\///////////\_\_\/\\\\\\\_\_\_\_\_\_\_\_/\\\\\\\_\_\_\_\_/\\\///\\\\_\_\_\_

# \_\/\\\\_\_\_\_\_\_\//\\\\_\/\\\\_\_\_\_\_\_\_\_\_\_\_\_\_\/\\\//\\\\_\_\_\_/\\\//\\\\_\_\_/\\\/\_\_\///\\\\_\_

# \_\/\\\\_\_\_\_\_\_\_\/\\\\_\/\\\\\\\\\\\\_\_\_\_\_\/\\\\///\\\/\\\/\_\/\\\\_\_/\\\\_\_\_\_\_\_\//\\\\_

# \_\/\\\\_\_\_\_\_\_\_\/\\\\_\/\\\///////\_\_\_\_\_\_\/\\\\_\_\///\\\/\_\_\_\/\\\\_\/\\\\_\_\_\_\_\_\_\/\\\\_

# \_\/\\\\_\_\_\_\_\_\_\/\\\\_\/\\\\_\_\_\_\_\_\_\_\_\_\_\_\_\/\\\\_\_\_\_\///\_\_\_\_\_\/\\\\_\//\\\\_\_\_\_\_\_/\\\\_\_

# \_\/\\\\_\_\_\_\_\_\_/\\\\_\_\/\\\\_\_\_\_\_\_\_\_\_\_\_\_\_\/\\\\_\_\_\_\_\_\_\_\_\_\_\_\_\/\\\\_\_\///\\\\_\_/\\\\_\_\_\_

# \_\/\\\\\\\\\\\\/\_\_\_\/\\\\\\\\\\\\\\\\_\/\\\\_\_\_\_\_\_\_\_\_\_\_\_\_\/\\\\_\_\_\_\///\\\\\/\_\_\_\_\_

# \_\////////////\_\_\_\_\_\///////////////\_\_\///\_\_\_\_\_\_\_\_\_\_\_\_\_\_\///\_\_\_\_\_\_\_\/////\_\_\_\_\_\_\_

# SPEC-FILE syntax

# <objectTypeName> { :variableType: variableName: {N/A|KEY|IDX}}+

# ^ this is the mode ^

# Notes:

# First letter of Object name should be Uppercase

# First letter of variable name should be Uppercase

# Variable Type is one of: string, int, bool, // TODO make a real list

@Project:Pasta

@Release:1.0

@TestChaincodeName:FarmerPkg6

@Author:lionel.martinez@oracle.com

@VerbosityLevel:1

@TestUser:bcuser.vtc

@TestPassword:***<NotInADocumentation>***

@TestChannelName:test03

@DeployURLRoot:https://25B08FDB595B4220813663ECC440389F.blockchain.ocp.oraclecloud.com:443/

@TestURLInvoke:https://25B08FDB595B4220813663ECC440389F.blockchain.ocp.oraclecloud.com:443/restproxy1/bcsgw/rest/v1/transaction/invocation

@TestURLQuery:https://25B08FDB595B4220813663ECC440389F.blockchain.ocp.oraclecloud.com:443/restproxy1/bcsgw/rest/v1/transaction/query

#@DeployPeersURLs : grpcs://41A0C23637DA45A2ADC6F53F5B80225D-private.blockchain.ocp.oraclecloud.com:7100;grpcs://41A0C23637DA45A2ADC6F53F5B80225D-private.blockchain.ocp.oraclecloud.com:7102

@DeployPeersURLs : grpcs://7D1D251254A64F7199C64B6AE9573540-private.blockchain.ocp.oraclecloud.com:7100;\

grpcs://7D1D251254A64F7199C64B6AE9573540-private.blockchain.ocp.oraclecloud.com:7102

Farmer: string:docType:IDX: string:FarmerID:KEY: string:ProductionArea:N/A

OutboundFarmerShipment: string:docType:IDX: string:FarmerShipmentId:KEY: string:FarmerID:IDX: \  
string:HandlerID:IDX: string:ShipmentDate:N/A: string:WheatType:N/A: int:WheatAmount:N/A: string:HarvestDate:N/A: \  
string:HarvestSeason:N/A

FieldTreatment: string:docType:IDX: string:FieldTreatmentID:KEY: string:FarmerID:IDX: string:HarvestSeason:N/A: \  
string:SeedsType:IDX: string:FieldTreatmentType:IDX: int:FieldTreatmentQty:N/A: string:FieldTreatmentDate:N/A

SampleAnalysis: string:docType:IDX: string:SampleAnalysisID:KEY: string:FarmerID:IDX: string:ProductionArea:N/A: \  
string:SeedsType:IDX: string:PhyBioChemAnalysis:N/A: string:AnalysisDate:N/A

## All data below are for tests... ##

#/!\ CAUTION, URLs contains colons(:) so they are separated by semi-colons (;)

#@DeployPeersURLs: \

# aaa ; \

# bbb;\

# ccc;\

# ddd

### GetAllFarmersSQL does bot workn but you can use GetFarmersByDocType, using DocType="Farmer"

$SQL :\

Farmer :\

GetAllFarmersSQL :\

SELECT valueJson \

FROM <STATE> WHERE \

json\_extract(valueJson, '$.docType') = '["Farmer"]' \

:\

NONE

$SQL :\

Farmer :\

GetFarmerByIdSQL :\

SELECT valueJson \

FROM <STATE> WHERE \

json\_extract(valueJson, '$.docType', '$.FarmerID') = '["Farmer","%FarmerID"]' \

:\

string:FarmerID:in :\

NONE

$SQL :\

Farmer :\

GetFarmerByProdAreaSQL :\

SELECT valueJson \

FROM <STATE> WHERE \

json\_extract(valueJson, '$.docType', '$.ProductionArea') = '["Farmer","%ProductionArea"]' \

:\

string:ProductionArea:in :\

NONE

$TEST:Comment:Generated Object Methods -----------------

$TEST:Call:createFarmer:Farmer:TS04F001:PA0001

$TEST:Call:createFarmer:Farmer:TS04F002:PA0011

$TEST:Call:createFarmer:Farmer:TS04F003:PA0021

$TEST:Call:createFarmer:Farmer:TS04F004:PA0031

$TEST:Call:createFarmer:Farmer:TS04F005:PA0041

$TEST:Call:getFarmer:TS04F001

$TEST:Call:getFarmer:TS04F002

$TEST:Call:getFarmer:TS04F003

$TEST:Call:getFarmer:TS04F004

$TEST:Call:getFarmer:TS04F005

$TEST:Call:getFarmerHistory:TS04F001

$TEST:Call:findFarmerBydocType:Farmer

$TEST:Comment:------------------------------------------

$TEST:Comment:SQL Functions ----------------------------

$TEST:Call:GetAllFarmersSQL

$TEST:Call:GetFarmerByIdSQL:TS04F001

$TEST:Call:GetFarmerByIdSQL:TS04F002

$TEST:Call:GetFarmerByIdSQL:TS04F003

$TEST:Call:GetFarmerByIdSQL:TS04F004

$TEST:Call:GetFarmerByIdSQL:TS04F005

$TEST:Call:GetFarmerByProdAreaSQL:PA0001

$TEST:Call:GetFarmerByProdAreaSQL:PA0011

$TEST:Call:GetFarmerByProdAreaSQL:PA0021

$TEST:Call:GetFarmerByProdAreaSQL:PA0031

$TEST:Call:GetFarmerByProdAreaSQL:PA0041

$TEST:Comment:------------------------------------------

Using SmartContractGen

[bcsdemo@oabcs-vm DEMO]$ **../SmartContractGen**

log:-1:Args[0]:../SmartContractGen

log:0:LogLevel >= 0

log:1:Definitions loaded.

log:1:INFO:Object Farmer:

log:0:WARNING:varName changed from Farmer::docType to Farmer::DocType

log:1:INFO:Object OutboundFarmerShipment:

log:0:WARNING:varName changed from OutboundFarmerShipment::docType to OutboundFarmerShipment::DocType

log:1:INFO:Object FieldTreatment:

This is next command-line to type…

log:0:WARNING:varName changed from FieldTreatment::docType to FieldTreatment::DocType

log:1:INFO:Object SampleAnalysis:

log:0:WARNING:varName changed from SampleAnalysis::docType to SampleAnalysis::DocType

log:1:Keys and indexes processed.

log:1:INFO:outputting SmartContract into file chaincode-Pasta1.0-FarmerPkg6.go

log:1:INFO:outputting Test Program into file contract-Pasta1.0-FarmerPkg6\_test.go

log:1:INFO:outputting Deploy Script into file deploy-Pasta1.0-FarmerPkg6.bash

log:0:INFO:Duration: 21.589004ms

Try...

go test -v --tags nopkcs11

[bcsdemo@oabcs-vm DEMO]$ ll

total 76

-rwxrwx---. 1 root vboxsf 51361 22 mars 14:32 chaincode-Pasta1.0-FarmerPkg6.go

-rwxrwx---. 1 root vboxsf 9252 22 mars 14:32 contract-Pasta1.0-FarmerPkg6\_test.go

-rwxrwx---. 1 root vboxsf 3334 22 mars 14:32 deploy-Pasta1.0-FarmerPkg6.bash

-rwxrwx---. 1 root vboxsf 5259 22 mars 14:30 spec-file.txt

[bcsdemo@oabcs-vm DEMO]$ head chaincode-Pasta1.0-FarmerPkg6.go

////////////////////////////////////

// SmartContractGen 5.7.2.2 generated SmartContractHeader file

//

// Generation time: 2019-03-22 14:32:26.733832584 +0100 CET m=+0.008308458

// Project: Pasta

// Release: 1.0

// Author: lionel.martinez@oracle.com

////////////////////////////////////

Local test session (needs the Mock Shim end GoLang installed)

[bcsdemo@oabcs-vm DEMO]$ **go test -v --tags nopkcs11**

=== RUN TestSimpleLedger

log:0:Entered TestSimpleLedger Function

log:1:createFarmer

log:1:Response Message:

log:1:Response Payload: {"docType":"Farmer","FarmerID":"TS04F001","ProductionArea":"PA0001"}

log:1:Response Status: 200

Means OK

log:1:createFarmer

log:1:Response Message:

log:1:Response Payload: {"docType":"Farmer","FarmerID":"TS04F002","ProductionArea":"PA0011"}

log:1:Response Status: 200

log:1:createFarmer

log:1:Response Message:

log:1:Response Payload: {"docType":"Farmer","FarmerID":"TS04F003","ProductionArea":"PA0021"}

log:1:Response Status: 200

log:1:createFarmer

log:1:Response Message:

log:1:Response Payload: {"docType":"Farmer","FarmerID":"TS04F004","ProductionArea":"PA0031"}

log:1:Response Status: 200

log:1:createFarmer

log:1:Response Message:

log:1:Response Payload: {"docType":"Farmer","FarmerID":"TS04F005","ProductionArea":"PA0041"}

log:1:Response Status: 200

log:1:getFarmer

log:1:Response Message:

log:1:Response Payload: {"docType":"Farmer","FarmerID":"TS04F001","ProductionArea":"PA0001"}

log:1:Response Status: 200

log:1:getFarmer

log:1:Response Message:

log:1:Response Payload: {"docType":"Farmer","FarmerID":"TS04F002","ProductionArea":"PA0011"}

log:1:Response Status: 200

log:1:getFarmer

log:1:Response Message:

log:1:Response Payload: {"docType":"Farmer","FarmerID":"TS04F003","ProductionArea":"PA0021"}

log:1:Response Status: 200

log:1:getFarmer

log:1:Response Message:

log:1:Response Payload: {"docType":"Farmer","FarmerID":"TS04F004","ProductionArea":"PA0031"}

log:1:Response Status: 200

log:1:getFarmer

log:1:Response Message:

log:1:Response Payload: {"docType":"Farmer","FarmerID":"TS04F005","ProductionArea":"PA0041"}

log:1:Response Status: 200

The History APIs are not implemented on the Mock Shim But at least the the ChainCode compiles OK

log:1:getFarmerHistory

log:1:Response Message: not implemented

log:1:Response Payload:

log:1:Response Status: 500

log:1:findFarmerBydocType

log:1:Response Message:

log:1:Response Payload: [{"docType":"Farmer","FarmerID":"TS04F001","ProductionArea":"PA0001"},{"docType":"Farmer","FarmerID":"TS04F002","ProductionArea":"PA0011"},{"docType":"Farmer","FarmerID":"TS04F003","ProductionArea":"PA0021"},{"docType":"Farmer","FarmerID":"TS04F004","ProductionArea":"PA0031"},{"docType":"Farmer","FarmerID":"TS04F005","ProductionArea":"PA0041"}]

log:1:Response Status: 200

- getQueryResultForQueryString queryString:

SELECT valueJson FROM <STATE> WHERE json\_extract(valueJson, '$.docType') = '["Farmer"]'

log:1:GetAllFarmersSQL

log:1:Response Message: not implemented

log:1:Response Payload:

log:1:Response Status: 500

The SQL Requests APIs are not implemented on the Mock Shim

- getQueryResultForQueryString queryString:

SELECT valueJson FROM <STATE> WHERE json\_extract(valueJson, '$.docType', '$.FarmerID') = '["Farmer","TS04F001"]'

log:1:GetFarmerByIdSQL

log:1:Response Message: not implemented

log:1:Response Payload:

log:1:Response Status: 500

- getQueryResultForQueryString queryString:

SELECT valueJson FROM <STATE> WHERE json\_extract(valueJson, '$.docType', '$.FarmerID') = '["Farmer","TS04F002"]'

log:1:GetFarmerByIdSQL

log:1:Response Message: not implemented

log:1:Response Payload:

log:1:Response Status: 500

- getQueryResultForQueryString queryString:

SELECT valueJson FROM <STATE> WHERE json\_extract(valueJson, '$.docType', '$.FarmerID') = '["Farmer","TS04F003"]'

log:1:GetFarmerByIdSQL

log:1:Response Message: not implemented

log:1:Response Payload:

log:1:Response Status: 500

- getQueryResultForQueryString queryString:

SELECT valueJson FROM <STATE> WHERE json\_extract(valueJson, '$.docType', '$.FarmerID') = '["Farmer","TS04F004"]'

log:1:GetFarmerByIdSQL

log:1:Response Message: not implemented

log:1:Response Payload:

log:1:Response Status: 500

- getQueryResultForQueryString queryString:

SELECT valueJson FROM <STATE> WHERE json\_extract(valueJson, '$.docType', '$.FarmerID') = '["Farmer","TS04F005"]'

log:1:GetFarmerByIdSQL

log:1:Response Message: not implemented

log:1:Response Payload:

log:1:Response Status: 500

- getQueryResultForQueryString queryString:

SELECT valueJson FROM <STATE> WHERE json\_extract(valueJson, '$.docType', '$.ProductionArea') = '["Farmer","PA0001"]'

log:1:GetFarmerByProdAreaSQL

log:1:Response Message: not implemented

log:1:Response Payload:

log:1:Response Status: 500

- getQueryResultForQueryString queryString:

SELECT valueJson FROM <STATE> WHERE json\_extract(valueJson, '$.docType', '$.ProductionArea') = '["Farmer","PA0011"]'

log:1:GetFarmerByProdAreaSQL

log:1:Response Message: not implemented

log:1:Response Payload:

log:1:Response Status: 500

- getQueryResultForQueryString queryString:

SELECT valueJson FROM <STATE> WHERE json\_extract(valueJson, '$.docType', '$.ProductionArea') = '["Farmer","PA0021"]'

log:1:GetFarmerByProdAreaSQL

log:1:Response Message: not implemented

log:1:Response Payload:

log:1:Response Status: 500

- getQueryResultForQueryString queryString:

SELECT valueJson FROM <STATE> WHERE json\_extract(valueJson, '$.docType', '$.ProductionArea') = '["Farmer","PA0031"]'

log:1:GetFarmerByProdAreaSQL

log:1:Response Message: not implemented

log:1:Response Payload:

log:1:Response Status: 500

- getQueryResultForQueryString queryString:

SELECT valueJson FROM <STATE> WHERE json\_extract(valueJson, '$.docType', '$.ProductionArea') = '["Farmer","PA0041"]'

log:1:GetFarmerByProdAreaSQL

log:1:Response Message: not implemented

log:1:Response Payload:

log:1:Response Status: 500

From here, the test values are automatically generated, so event if you don’t spend time putting relevant @TEST statements, you’ll always have one test generated per generated function

log:1:createFarmer

log:1:Response Message:

log:1:Response Payload: {"docType":"DocType(string)","FarmerID":"Farmer:FarmerID(string)","ProductionArea":"ProductionArea(string)"}

log:1:Response Status: 200

log:1:getFarmer

log:1:Response Message:

log:1:Response Payload: {"docType":"DocType(string)","FarmerID":"Farmer:FarmerID(string)","ProductionArea":"ProductionArea(string)"}

log:1:Response Status: 200

log:1:getFarmerHistory

log:1:Response Message: not implemented

log:1:Response Payload:

log:1:Response Status: 500

log:1:findFarmerBydocType

log:1:Response Message:

log:1:Response Payload: [{"docType":"DocType(string)","FarmerID":"Farmer:FarmerID(string)","ProductionArea":"ProductionArea(string)"}]

log:1:Response Status: 200

log:1:createFieldTreatment

log:1:Response Message:

log:1:Response Payload: {"docType":"DocType(string)","FieldTreatmentID":"FieldTreatment:FieldTreatmentID(string)","FarmerID":"FarmerID(string)","HarvestSeason":"HarvestSeason(string)","SeedsType":"SeedsType(string)","FieldTreatmentType":"FieldTreatmentType(string)","FieldTreatmentQty":1,"FieldTreatmentDate":"FieldTreatmentDate(string)"}

log:1:Response Status: 200

log:1:getFieldTreatment

log:1:Response Message:

log:1:Response Payload: {"docType":"DocType(string)","FieldTreatmentID":"FieldTreatment:FieldTreatmentID(string)","FarmerID":"FarmerID(string)","HarvestSeason":"HarvestSeason(string)","SeedsType":"SeedsType(string)","FieldTreatmentType":"FieldTreatmentType(string)","FieldTreatmentQty":1,"FieldTreatmentDate":"FieldTreatmentDate(string)"}

log:1:Response Status: 200

log:1:getFieldTreatmentHistory

log:1:Response Message: not implemented

log:1:Response Payload:

log:1:Response Status: 500

log:1:findFieldTreatmentBydocType

log:1:Response Message:

log:1:Response Payload: [{"docType":"DocType(string)","FieldTreatmentID":"FieldTreatment:FieldTreatmentID(string)","FarmerID":"FarmerID(string)","HarvestSeason":"HarvestSeason(string)","SeedsType":"SeedsType(string)","FieldTreatmentType":"FieldTreatmentType(string)","FieldTreatmentQty":1,"FieldTreatmentDate":"FieldTreatmentDate(string)"}]

log:1:Response Status: 200

log:1:findFieldTreatmentByFarmerID

log:1:Response Message:

log:1:Response Payload: [{"docType":"DocType(string)","FieldTreatmentID":"FieldTreatment:FieldTreatmentID(string)","FarmerID":"FarmerID(string)","HarvestSeason":"HarvestSeason(string)","SeedsType":"SeedsType(string)","FieldTreatmentType":"FieldTreatmentType(string)","FieldTreatmentQty":1,"FieldTreatmentDate":"FieldTreatmentDate(string)"}]

log:1:Response Status: 200

log:1:findFieldTreatmentBySeedsType

log:1:Response Message:

log:1:Response Payload: [{"docType":"DocType(string)","FieldTreatmentID":"FieldTreatment:FieldTreatmentID(string)","FarmerID":"FarmerID(string)","HarvestSeason":"HarvestSeason(string)","SeedsType":"SeedsType(string)","FieldTreatmentType":"FieldTreatmentType(string)","FieldTreatmentQty":1,"FieldTreatmentDate":"FieldTreatmentDate(string)"}]

log:1:Response Status: 200

log:1:findFieldTreatmentByFieldTreatmentType

log:1:Response Message:

log:1:Response Payload: [{"docType":"DocType(string)","FieldTreatmentID":"FieldTreatment:FieldTreatmentID(string)","FarmerID":"FarmerID(string)","HarvestSeason":"HarvestSeason(string)","SeedsType":"SeedsType(string)","FieldTreatmentType":"FieldTreatmentType(string)","FieldTreatmentQty":1,"FieldTreatmentDate":"FieldTreatmentDate(string)"}]

log:1:Response Status: 200

log:1:createOutboundFarmerShipment

log:1:Response Message:

log:1:Response Payload: {"docType":"DocType(string)","FarmerShipmentId":"OutboundFarmerShipment:FarmerShipmentId(string)","FarmerID":"FarmerID(string)","HandlerID":"HandlerID(string)","ShipmentDate":"ShipmentDate(string)","WheatType":"WheatType(string)","WheatAmount":1,"HarvestDate":"HarvestDate(string)","HarvestSeason":"HarvestSeason(string)"}

log:1:Response Status: 200

log:1:getOutboundFarmerShipment

log:1:Response Message:

log:1:Response Payload: {"docType":"DocType(string)","FarmerShipmentId":"OutboundFarmerShipment:FarmerShipmentId(string)","FarmerID":"FarmerID(string)","HandlerID":"HandlerID(string)","ShipmentDate":"ShipmentDate(string)","WheatType":"WheatType(string)","WheatAmount":1,"HarvestDate":"HarvestDate(string)","HarvestSeason":"HarvestSeason(string)"}

log:1:Response Status: 200

log:1:getOutboundFarmerShipmentHistory

log:1:Response Message: not implemented

log:1:Response Payload:

log:1:Response Status: 500

log:1:findOutboundFarmerShipmentBydocType

log:1:Response Message:

log:1:Response Payload: [{"docType":"DocType(string)","FarmerShipmentId":"OutboundFarmerShipment:FarmerShipmentId(string)","FarmerID":"FarmerID(string)","HandlerID":"HandlerID(string)","ShipmentDate":"ShipmentDate(string)","WheatType":"WheatType(string)","WheatAmount":1,"HarvestDate":"HarvestDate(string)","HarvestSeason":"HarvestSeason(string)"}]

log:1:Response Status: 200

log:1:findOutboundFarmerShipmentByFarmerID

log:1:Response Message:

log:1:Response Payload: [{"docType":"DocType(string)","FarmerShipmentId":"OutboundFarmerShipment:FarmerShipmentId(string)","FarmerID":"FarmerID(string)","HandlerID":"HandlerID(string)","ShipmentDate":"ShipmentDate(string)","WheatType":"WheatType(string)","WheatAmount":1,"HarvestDate":"HarvestDate(string)","HarvestSeason":"HarvestSeason(string)"}]

log:1:Response Status: 200

log:1:findOutboundFarmerShipmentByHandlerID

log:1:Response Message:

log:1:Response Payload: [{"docType":"DocType(string)","FarmerShipmentId":"OutboundFarmerShipment:FarmerShipmentId(string)","FarmerID":"FarmerID(string)","HandlerID":"HandlerID(string)","ShipmentDate":"ShipmentDate(string)","WheatType":"WheatType(string)","WheatAmount":1,"HarvestDate":"HarvestDate(string)","HarvestSeason":"HarvestSeason(string)"}]

log:1:Response Status: 200

log:1:createSampleAnalysis

log:1:Response Message:

log:1:Response Payload: {"docType":"DocType(string)","SampleAnalysisID":"SampleAnalysis:SampleAnalysisID(string)","FarmerID":"FarmerID(string)","ProductionArea":"ProductionArea(string)","SeedsType":"SeedsType(string)","PhyBioChemAnalysis":"PhyBioChemAnalysis(string)","AnalysisDate":"AnalysisDate(string)"}

log:1:Response Status: 200

log:1:getSampleAnalysis

log:1:Response Message:

log:1:Response Payload: {"docType":"DocType(string)","SampleAnalysisID":"SampleAnalysis:SampleAnalysisID(string)","FarmerID":"FarmerID(string)","ProductionArea":"ProductionArea(string)","SeedsType":"SeedsType(string)","PhyBioChemAnalysis":"PhyBioChemAnalysis(string)","AnalysisDate":"AnalysisDate(string)"}

log:1:Response Status: 200

log:1:getSampleAnalysisHistory

log:1:Response Message: not implemented

log:1:Response Payload:

log:1:Response Status: 500

log:1:findSampleAnalysisBydocType

log:1:Response Message:

log:1:Response Payload: [{"docType":"DocType(string)","SampleAnalysisID":"SampleAnalysis:SampleAnalysisID(string)","FarmerID":"FarmerID(string)","ProductionArea":"ProductionArea(string)","SeedsType":"SeedsType(string)","PhyBioChemAnalysis":"PhyBioChemAnalysis(string)","AnalysisDate":"AnalysisDate(string)"}]

log:1:Response Status: 200

log:1:findSampleAnalysisByFarmerID

log:1:Response Message:

log:1:Response Payload: [{"docType":"DocType(string)","SampleAnalysisID":"SampleAnalysis:SampleAnalysisID(string)","FarmerID":"FarmerID(string)","ProductionArea":"ProductionArea(string)","SeedsType":"SeedsType(string)","PhyBioChemAnalysis":"PhyBioChemAnalysis(string)","AnalysisDate":"AnalysisDate(string)"}]

log:1:Response Status: 200

log:1:findSampleAnalysisBySeedsType

log:1:Response Message:

log:1:Response Payload: [{"docType":"DocType(string)","SampleAnalysisID":"SampleAnalysis:SampleAnalysisID(string)","FarmerID":"FarmerID(string)","ProductionArea":"ProductionArea(string)","SeedsType":"SeedsType(string)","PhyBioChemAnalysis":"PhyBioChemAnalysis(string)","AnalysisDate":"AnalysisDate(string)"}]

log:1:Response Status: 200

- getQueryResultForQueryString queryString:

SELECT valueJson FROM <STATE> WHERE json\_extract(valueJson, '$.docType') = '["Farmer"]'

log:1:GetAllFarmersSQL

log:1:Response Message: not implemented

log:1:Response Payload:

log:1:Response Status: 500

- getQueryResultForQueryString queryString:

SELECT valueJson FROM <STATE> WHERE json\_extract(valueJson, '$.docType', '$.FarmerID') = '["Farmer","FarmerID(string)"]'

log:1:GetFarmerByIdSQL

log:1:Response Message: not implemented

log:1:Response Payload:

log:1:Response Status: 500

- getQueryResultForQueryString queryString:

SELECT valueJson FROM <STATE> WHERE json\_extract(valueJson, '$.docType', '$.ProductionArea') = '["Farmer","ProductionArea(string)"]'

log:1:GetFarmerByProdAreaSQL

log:1:Response Message: not implemented

log:1:Response Payload:

log:1:Response Status: 500

And at the end of the local test, the local tester generates the curl bash test script

log:1:INFO:outputting Generating Bash Test script for those requests

INFO:done.

--- PASS: TestSimpleLedger (0.06s)

PASS

ok \_/media/sf\_TMP/20190219demo/DEMO 0.100s

[bcsdemo@oabcs-vm DEMO]$ ll

total 88

-rwxrwx---. 1 root vboxsf 51361 22 mars 14:32 chaincode-Pasta1.0-FarmerPkg6.go

-rwxrwx---. 1 root vboxsf 9252 22 mars 14:32 contract-Pasta1.0-FarmerPkg6\_test.go

-rwxrwx---. 1 root vboxsf 3334 22 mars 14:32 deploy-Pasta1.0-FarmerPkg6.bash

-rwxrwx---. 1 root vboxsf 5259 22 mars 14:30 spec-file.txt

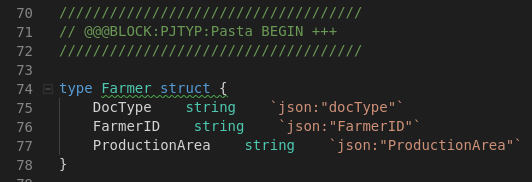
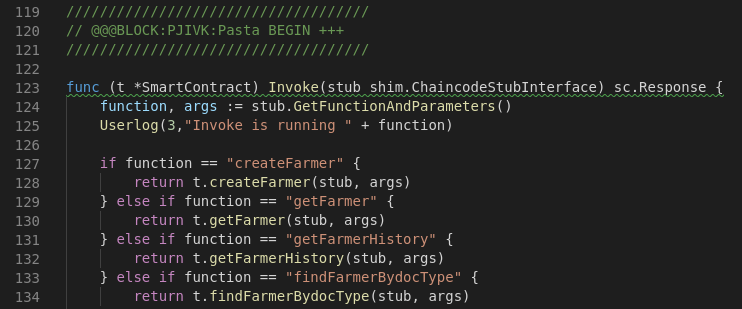
-rwxrwx---. 1 root vboxsf 9853 22 mars 14:32 test-Pasta1.0-FarmerPkg6.bash

[bcsdemo@oabcs-vm DEMO]$

## Some generated code explained

Here in the spec-file.txt file we have this entity:  
Farmer: string:docType:IDX: string:FarmerID:KEY: string:ProductionArea:N/A

It generates:

  
  
And the invoke function (truncated):  


And for instance, the createFarmer function is:

* First, we create a FarmerAssetAsBytes JSON stream of the Farmer object we will insert.
* PutState inserts it in the database
* Then we take care of the index fields (here doctype) with the CreateCompositeKey API.

This is VERY similar to the OBCS samples.

////////////////////////////////////

// @@@BLOCK:PJCRE:Pasta:createFarmer BEGIN +++

////////////////////////////////////

func (s \*SmartContract) createFarmer(stub shim.ChaincodeStubInterface, args []string) sc.Response {

    if len(args)!=3 {

       return shim.Error("createFarmer: Incorrect number of arguments. Expecting 3")

    }

    var FarmerAsset = Farmer {

        DocType: args[0],

        FarmerID: args[1],

        ProductionArea: args[2],

    }

    var err error

    FarmerAssetAsBytes, err := json.Marshal(FarmerAsset)

    if err != nil {

        return shim.Error(fmt.Sprintf("createFarmer: Failed to marshal Farmer: %s", args[0]))

    }

    err = stub.PutState(FarmerAsset.FarmerID, FarmerAssetAsBytes)

    if err != nil {

        return shim.Error(fmt.Sprintf("createFarmer: Failed to record Farmer: %s", args[0]))

    }

    // INDEX CREATION FOR DocType~FarmerID

    indexName00 := "DocType~FarmerID"

    FarmerIndexKey00, err00 := stub.CreateCompositeKey(indexName00, []string{ FarmerAsset.DocType, FarmerAsset.FarmerID})

    if err00 != nil {

        return shim.Error(err00.Error())

    }

    value00 := []byte{0x00}

    stub.PutState(FarmerIndexKey00, value00)

    if err00 != nil {

        return shim.Error(fmt.Sprintf("Failed to record indexName: %s", args[0]))

    }

    return shim.Success(FarmerAssetAsBytes)

}