Exploring the Chaincode

Hyperledger Fabric Smart Contract
Lecture Goals

● Smart Contracts in Hyperledger Fabric
● Components of a Chaincode
   ■ Structure of Chaincode
   ■ Chaincode Interface
   ■ Asset Definition
   ■ Init & Invoke Functions
   ■ Query Method
   ■ Command Method
● Exercise
   ■ Start Parking Session
   ■ End Parking Session
Stakeholder

- Execute transactions
- Receive ledger updates

Smart contract

Chaincode

Digital asset

Track changes

Ledger
Chaincode

Its purpose is twofold:

1. Definition of Assets
   • Digital representation of assets
   • Stored in a “state database”
   • Queried/updated via chaincode functions

2. Implementation of business logic
   • Rules for reading assets from/altering assets into state database
   • Asset updates result in write sets, which are stored in the ledger
Chaincode

Hyperledger fabric promotes the use of general purpose programming languages instead of domain specific ones
• This contrast with the choice of other platforms, e.g. Ethereum supports several domain specific programming languages, including solidity

By the time of this writing, Hyperledger fabric provides bindings for Golang, Javascript (via NodeJS) and Java
• Golang is thoroughly document and, hence, it is the language used in this material
Components of Chaincode
package main

import {
    "github.com/hyperledger/fabric/core/chaincode/shim"
    sc "github.com/hyperledger/fabric/protos/peer"
}

type SmartContract struct {
}

func (s *SmartContract) Init(APIstub shim.ChaincodeStubInterface) sc.Response {
    return shim.Success(nil)
}

func (s *SmartContract) Invoke(APIstub shim.ChaincodeStubInterface) sc.Response {
    return shim.Success(nil)
}

func main() {
    shim.Start(new(SmartContract))
}
Object Orientation in Golang: Structs + Functions

```go
package main

import {
    "github.com/hyperledger/fabric/core/chaincode/shim"
    sc "github.com/hyperledger/fabric/protos/peer"
}

type SmartContract struct {
}

func (s *SmartContract) Init(APIstub shim.ChaincodeStubInterface) sc.Response {
    return shim.Success(nil)
}

func (s *SmartContract) Invoke(APIstub shim.ChaincodeStubInterface) sc.Response {
    return shim.Success(nil)
}

func main() {
    shim.Start(new(SmartContract))
}
```
package main

import {
    "github.com/hyperledger/fabric/core/chaincode/shim"
    sc "github.com/hyperledger/fabric/protos/peer"
}

type SmartContract struct {
}

func (s *SmartContract) Init(APIstub shim.ChaincodeStubInterface) sc.Response {
    return shim.Success(nil)
}

func (s *SmartContract) Invoke(APIstub shim.ChaincodeStubInterface) sc.Response {
    return shim.Success(nil)
}

func main() {
    shim.Start(new(SmartContract))
}
package main

import (  "github.com/hyperledger/fabric/core/chaincode/shim"  sc "github.com/hyperledger/fabric/protos/peer"
)

type SmartContract struct {
}

func (s *SmartContract) Init(APIstub shim.ChaincodeStubInterface) sc.Response {
  return shim.Success(nil)
}

func (s *SmartContract) Invoke(APIstub shim.ChaincodeStubInterface) sc.Response {
  return shim.Success(nil)
}

func main() {
  shim.Start(new(SmartContract))
}
/*
 * The following asset definition is taken from the sample application "fabcar"
 */

type Car struct {
    Make string `json:"make"
    Model string `json:"model"
    Colour string `json:"colour"
    Owner string `json:"owner"
}
Invoke Function as a Dispatcher

```go
func (s *SmartContract) Invoke(APIstub shim.ChaincodeStubInterface) sc.Response {
    // Retrieve function and arguments
    function, args := APIstub.GetFunctionAndParameters()

    if function == "queryCar" {
        return s.queryCar(APIstub, args)
    } else if function == "initLedger" {
        return s.initLedger(APIstub)
    } else if function == "createCar" {
        return s.createCar(APIstub, args)
    } else if function == "queryAllCars" {
        return s.queryAllCars(APIstub)
    } else if function == "changeCarOwner" {
        return s.changeCarOwner(APIstub, args)
    }

    return shim.Error("Invalid Smart Contract function name.")
}
```

Assets are defined as Golang structs.
The following function is taken from the sample application "fabcar"

```go
func (s *SmartContract) queryCar(APIstub shim.ChaincodeStubInterface, args []string) sc.Response {
    if len(args) != 1 {
        return shim.Error("Incorrect number of arguments. Expecting 1")
    }
    carAsBytes, _ := APIstub.GetState(args[0])
    return shim.Success(carAsBytes)
}
```

“arg[0]” is expected to be the key associated with a digital asset (e.g. car) stored in the state database.

Function “GetState/1” takes a key and fetches the value associated with such key from the state database.
Command Methods

STEP 1
Retrieve digital asset from state database

```go
func (s *SmartContract) changeCarOwner(APIstub shim.ChaincodeStubInterface, args []string) sc.Response {
    if len(args) != 2 {
        return shim.Error("Incorrect number of arguments. Expecting 2")
    }
    carAsBytes, _ := APIstub.GetState(args[0])
    car := Car{}
    json.Unmarshal(carAsBytes, &car)
    car.Owner = args[1]
    carAsBytes, _ := json.Marshal(car)
    APIstub.PutState(args[0], carAsBytes)
    return shim.Success(nil)
}
```

STEP 2
Translate JSON into a struct value to allow one to update the asset

STEP 3
Translate to JSON and store the new asset value into the state database
Exercise: Parking Session Start & End Scenario
Example

Consider the simplified version of the parking application, as captured in the following model.

We will assume that the customer starts and ends explicitly the parking session (e.g. via an SMS or parking dApp)
Asset and Basic Interface Functions

Let us start by considering that the application manipulates only “parking sessions” (No definition for parking space nor customer yet).

We will cover only the two initial steps in the process model (the payment handling will come later).
func (s *SmartContract) startParkingSession(APIstub shim.ChaincodeStubInterface, args []string) sc.Response {

    if len(args) != 2 {
        return shim.Error("Incorrect number of arguments. Expecting 2")
    }

    txTimestamp, _ := APIstub.GetTxTimestamp();
    timestamp := time.Unix(txTimestamp.GetSeconds(), int64(txTimestamp.GetNanos()))
    session := ParkingSession{
        ParkingSpaceID: args[0],
        Customer: args[1],
        StartDateTime: timestamp.String(),
        EndDateTime: "",
        Cost: 0
    }

    sessionAsBytes, _ := json.Marshal(session)
    APIstub.PutState("ID123", sessionAsBytes)

    fmt.Printf("WROTE: %s", sessionAsBytes)
    return shim.Success(nil)
}

Start Parking Session
func (s *SmartContract) stopParkingSession(APIstub shim.ChaincodeStubInterface, args []string) sc.Response {
    txTimestamp, _ := APIstub.GetTxTimestamp();
    timestamp := time.Unix(txTimestamp.GetSeconds(), int64(txTimestamp.GetNanos()));

    sessionAsBytes, _ := APIstub.GetState("ID123")
    session := ParkingSession{}

    json.Unmarshal(sessionAsBytes, &session)
    session.EndDateTime = timestamp.String()

    startDateTime, _ := time.Parse(time.RFC3339, session.StartDateTime)
    endDateTime, _ := time.Parse(time.RFC3339, session.EndDateTime)

    sessionDuration := endDateTime.Sub(startDateTime)

    session.Cost = sessionDuration.Minutes() * 0.05

    sessionAsBytes, _ = json.Marshal(session)
    APIstub.PutState("ID123", sessionAsBytes)

    fmt.Printf("WROTE: %s", sessionAsBytes)
    return shim.Success(nil)
}
Lecture Goals

● Smart Contracts in Hyperledger Fabric
● Components of a Chaincode
  ■ Structure of Chaincode
  ■ Chaincode Interface
  ■ Asset Definition
  ■ Invoke Function
  ■ Query Method
  ■ Command Method
● Exercise
  ■ Start Parking Session
  ■ End Parking Session