Chaincode

1. **Choosing a Location for the Code**
2. Housekeeping
3. Initializing the Chaincode
4. Invoking the Chaincode
5. Implementing the Chaincode Application
6. Pulling it All Together
7. Building Chaincode
8. Testing Using dev mode

```
mkdir -p $GOPATH/src/sacc && cd $GOPATH/src/sacc
```

```
touch sacc.go
```

Chaincode

1. Choosing a Location for the Code
2. **Housekeeping**
3. Initializing the Chaincode
4. Invoking the Chaincode
5. Implementing the Chaincode Application
6. Pulling it All Together
7. Building Chaincode
8. Testing Using dev mode

```
package main

import (
    "fmt"

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

// SimpleAsset implements a simple chaincode to manage an asset
type SimpleAsset struct {
}
```

Chaincode

1. Choosing a Location for the Code
2. Housekeeping
3. **Initializing the Chaincode**
4. Invoking the Chaincode
5. Implementing the Chaincode Application
6. Pulling it All Together
7. Building Chaincode
8. Testing Using dev mode

```
// Init is called during chaincode instantiation to initialize any
// data. Note that chaincode upgrade also calls this function to reset
// or to migrate data, so be careful to avoid a scenario where you
// inadvertently clobber your ledger's data!
func (t *SimpleAsset) Init(stub shim.ChaincodeStubInterface) peer.Response {
  // Get the args from the transaction proposal
  args := stub.GetStringArgs()
  if len(args) != 2 {
    return shim.Error("Incorrect arguments. Expecting a key and a value")
  }

  // Set up any variables or assets here by calling stub.PutState()

  // We store the key and the value on the ledger
  err := stub.PutState(args[0], []byte(args[1]))
  if err != nil {
    return shim.Error(fmt.Sprintf("Failed to create asset: %s", args[0]))
  }
  return shim.Success(nil)
}
```

Chaincode

1. Choosing a Location for the Code
2. Housekeeping
3. Initializing the Chaincode
4. **Invoking the Chaincode**
5. Implementing the Chaincode Application
6. Pulling it All Together
7. Building Chaincode
8. Testing Using dev mode

```
// Invoke is called per transaction on the chaincode. Each transaction is
// either a 'get' or a 'set' on the asset created by Init function. The Set
// method may create a new asset by specifying a new key-value pair.
func (t *SimpleAsset) Invoke(stub shim.ChaincodeStubInterface) peer.Response {
    // Extract the function and args from the transaction proposal
    fn, args := stub.GetFunctionAndParameters()

    var result string
    var err error
    if fn == "set" {
            result, err = set(stub, args)
    } else {
            result, err = get(stub, args)
    }
    if err != nil {
            return shim.Error(err.Error())
    }

    // Return the result as success payload
    return shim.Success([]byte(result))
}
```

Chaincode

1. Choosing a Location for the Code
2. Housekeeping
3. Initializing the Chaincode
4. Invoking the Chaincode
5. **Implementing the Chaincode Application**
6. Pulling it All Together
7. Building Chaincode
8. Testing Using dev mode

```
// Set stores the asset (both key and value) on the ledger. If the key exists,
// it will override the value with the new one
func set(stub shim.ChaincodeStubInterface, args []string) (string, error) {
    if len(args) != 2 {
            return "", fmt.Errorf("Incorrect arguments. Expecting a key and a value")
    }

    err := stub.PutState(args[0], []byte(args[1]))
    if err != nil {
            return "", fmt.Errorf("Failed to set asset: %s", args[0])
    }
    return args[1], nil
}

// Get returns the value of the specified asset key
func get(stub shim.ChaincodeStubInterface, args []string) (string, error) {
    if len(args) != 1 {
            return "", fmt.Errorf("Incorrect arguments. Expecting a key")
    }

    value, err := stub.GetState(args[0])
    if err != nil {
            return "", fmt.Errorf("Failed to get asset: %s with error: %s", args[0], err)
    }
    if value == nil {
            return "", fmt.Errorf("Asset not found: %s", args[0])
    }
    return string(value), nil
}
```

Chaincode

1. Choosing a Location for the Code
2. Housekeeping
3. Initializing the Chaincode
4. Invoking the Chaincode
5. Implementing the Chaincode Application
6. **Pulling it All Together**
7. Building Chaincode
8. Testing Using dev mode

```
package main

import (
    "fmt"

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

// SimpleAsset implements a simple chaincode to manage an asset
type SimpleAsset struct {
}

// Init is called during chaincode instantiation to initialize any
// data. Note that chaincode upgrade also calls this function to reset
// or to migrate data.
func (t *SimpleAsset) Init(stub shim.ChaincodeStubInterface) peer.Response {

  ...

// main function starts up the chaincode in the container during instantiate
func main() {
    if err := shim.Start(new(SimpleAsset)); err != nil {
            fmt.Printf("Error starting SimpleAsset chaincode: %s", err)
    }
}
```

Chaincode

1. Choosing a Location for the Code
2. Housekeeping
3. Initializing the Chaincode
4. Invoking the Chaincode
5. Implementing the Chaincode Application
6. Pulling it All Together
7. **Building Chaincode**
8. Testing Using dev mode

```
go get -u github.com/hyperledger/fabric/core/chaincode/shim
go build
```