Source file src/chaincode/chaincode_tokens /chaincode_tokens_test.go

Documentation: **chaincode/chaincode_tokens**

```
 1  package main
 2
 3  import (
 4          "fmt"
 5          "testing"
 6
 7          "github.com/hyperledger/fabric/core/chaincode/shim"
 8  )
 9
10  func checkInit(t *testing.T, stub *shim.MockStub, args [][]byte) {
11          res := stub.MockInit("1", args)
12          if res.Status != shim.OK {
13                  fmt.Println("Init failed", string(res.Message))
14                  t.Fail()
15          }
16  }
17
18  func checkInitFail(t *testing.T, stub *shim.MockStub, args [][]byte) {
19          res := stub.MockInit("1", args)
```

```
    20          if res.Status == shim.OK {
    21                  fmt.Println("Init should fail but it did not", string(res.Message))
    22                  t.Fail()
    23          }
    24  }
    25
    26  func checkState(t *testing.T, stub *shim.MockStub, name string, value string) {
    27          bytes := stub.State[name]
    28          if bytes == nil {
    29                  fmt.Println("State", name, "failed to get value")
    30                  t.Fail()
    31          }
    32          if string(bytes) != value {
    33                  fmt.Println("State value", name, "was", string(bytes), "instead required",
value)
    34                  t.Fail()
    35          }
    36  }
    37
    38  func checkInvoke(t *testing.T, stub *shim.MockStub, args [][]byte) {
    39          res := stub.MockInvoke("1", args)
    40          if res.Status != shim.OK {
    41                  fmt.Println("Invoke", args, "failed", string(res.Message))
    42                  t.Fail()
    43          }
    44  }
    45
    46  func checkInvokeFail(t *testing.T, stub *shim.MockStub, args [][]byte) {
    47          res := stub.MockInvoke("1", args)
    48          if res.Status == shim.OK {
    49                  fmt.Println("Invoke", args, "should fail but did not", string(res.Payload))
```

```
    50                  t.Fail()
    51          }
    52  }
    53
    54  func checkInvokeResponse(t *testing.T, stub *shim.MockStub, args [][]byte, expectedPayload
string) {
    55          res := stub.MockInvoke("1", args)
    56          if res.Status != shim.OK {
    57                  fmt.Println("Invoke", args, "failed", string(res.Message))
    58                  t.Fail()
    59          }
    60          if string(res.Payload) != expectedPayload {
    61                  fmt.Println("Expected payload:", expectedPayload)
    62                  fmt.Println("Instead got this:", string(res.Payload))
    63                  t.Fail()
    64          }
    65  }
    66
    67  func checkInvokeResponseFail(t *testing.T, stub *shim.MockStub, args [][]byte, expectedMessage
string) {
    68          res := stub.MockInvoke("1", args)
    69          if res.Status == shim.OK {
    70                  fmt.Println("Invoke", args, "should fail")
    71                  fmt.Println("Instead got payload:", string(res.Payload))
    72                  t.Fail()
    73          }
    74          if res.Message != expectedMessage {
    75                  fmt.Println("Expected message:", expectedMessage)
    76                  fmt.Println("Instead got this:", res.Message)
    77                  t.Fail()
    78          }
```

```
 79  }
 80
 81  func Test_Init(t *testing.T) {
 82          cc := new(Chaincode)
 83          stub := shim.NewMockStub("tokens_init_test", cc)
 84
 85          // It should Init 1 account with 10 000 tokens
 86          checkInit(t, stub, [][]byte{[]byte("10000")})
 87          checkState(t, stub, "1",
 88                  "{\"RecordType\":\"ACCOUNT\",\"AccountID\":\"1\",\"Name\":\"Init_Account
\",\"OwnerID\":\"\",\"Tokens\":10000}")
 89
 90          // It should Init 1 accounts with 0 tokens
 91          stub = shim.NewMockStub("tokens_init_test", cc)
 92          checkInit(t, stub, [][]byte{[]byte("0")})
 93          checkState(t, stub, "1",
 94                  "{\"RecordType\":\"ACCOUNT\",\"AccountID\":\"1\",\"Name\":\"Init_Account
\",\"OwnerID\":\"\",\"Tokens\":0}")
 95
 96          // It should not Init an account with negative number of tokens
 97          stub = shim.NewMockStub("tokens_init_test", cc)
 98          checkInitFail(t, stub, [][]byte{[]byte("-10")})
 99
100          // It should not Init with less args than 1
101          stub = shim.NewMockStub("tokens_init_test", cc)
102          checkInitFail(t, stub, [][]byte{})
103
104          // It should not Init with more args than 1
105          stub = shim.NewMockStub("tokens_init_test", cc)
106          checkInitFail(t, stub, [][]byte{[]byte("1"), []byte("1")})
107
```

```
108         // It should not Init with empty arg
109         stub = shim.NewMockStub("tokens_init_test", cc)
110         checkInitFail(t, stub, [][]byte{[]byte("")})
111 }
112
113 func Test_InvokeFail(t *testing.T) {
114         cc := new(Chaincode)
115         stub := shim.NewMockStub("invoke_fail_test", cc)
116         args := [][]byte{[]byte("NoFunction"), []byte("test")}
117         expectedMessage := "Received unknown function invocation"
118         checkInvokeResponseFail(t, stub, args, expectedMessage)
119 }
120
121 func Test_createAccount(t *testing.T) {
122         cc := new(Chaincode)
123         stub := shim.NewMockStub("create_acc_test", cc)
124
125         // Init 1 account with 10 000 tokens
126         checkInit(t, stub, [][]byte{[]byte("10000")})
127
128         // It should create account
129         args := [][]byte{[]byte("createAccount"), []byte("2"), []byte("acc_name")}
130         expectedPayload := "Account created"
131         checkInvokeResponse(t, stub, args, expectedPayload)
132
133         // It should fail to create an account with ID that already exists
134         args = [][]byte{[]byte("createAccount"), []byte("2"), []byte("acc_name")}
135         expectedMessage := "This account already exists: 2"
136         checkInvokeResponseFail(t, stub, args, expectedMessage)
137
138         // It should fail with empty string arg
```

```
   139          args = [][]byte{[]byte("createAccount"), []byte(""), []byte("acc_name")}
   140          expectedMessage = "Argument at position 1 must be a non-empty string"
   141          checkInvokeResponseFail(t, stub, args, expectedMessage)
   142
   143          // It should fail with empty string arg
   144          args = [][]byte{[]byte("createAccount"), []byte("3"), []byte("")}
   145          expectedMessage = "Argument at position 2 must be a non-empty string"
   146          checkInvokeResponseFail(t, stub, args, expectedMessage)
   147
   148          // It should fail with less than 2 args
   149          args = [][]byte{[]byte("createAccount"), []byte("3")}
   150          expectedMessage = "Incorrect number of arguments. Expecting account Id and name"
   151          checkInvokeResponseFail(t, stub, args, expectedMessage)
   152
   153          // It should fail with more than 2 args
   154          args = [][]byte{[]byte("createAccount"), []byte("3"), []byte("acc_name"),
[]byte("acc_name")}
   155          expectedMessage = "Incorrect number of arguments. Expecting account Id and name"
   156          checkInvokeResponseFail(t, stub, args, expectedMessage)
   157  }
   158
   159  func Test_deleteAccountByID(t *testing.T) {
   160          cc := new(Chaincode)
   161          stub := shim.NewMockStub("tokens_init_test", cc)
   162
   163          // Init 1 account with 10 000 tokens
   164          checkInit(t, stub, [][]byte{[]byte("10000")})
   165          // create account with 0 tokens
   166          args := [][]byte{[]byte("createAccount"), []byte("2"), []byte("acc_name")}
   167          expectedPayload := "Account created"
   168          checkInvokeResponse(t, stub, args, expectedPayload)
```

```
169
170         // it should delete account that have 0 tokens
171         args = [][]byte{[]byte("deleteAccountByID"), []byte("2")}
172         expectedPayload = "Account deleted"
173         checkInvokeResponse(t, stub, args, expectedPayload)
174
175         // it should not be possible to delete an account that have some tokens
176         args = [][]byte{[]byte("deleteAccountByID"), []byte("1")}
177         expectedMessage := "Account cannot be deleted. Amount of tokens is not 0."
178         checkInvokeResponseFail(t, stub, args, expectedMessage)
179
180         // It should fail with empty string arg
181         args = [][]byte{[]byte("deleteAccountByID"), []byte("")}
182         expectedMessage = "Argument at position 1 must be a non-empty string"
183         checkInvokeResponseFail(t, stub, args, expectedMessage)
184
185         // It should fail with more than 2 args
186         args = [][]byte{[]byte("deleteAccountByID"), []byte("2"), []byte("2")}
187         expectedMessage = "Incorrect number of arguments. Expecting AccountID."
188         checkInvokeResponseFail(t, stub, args, expectedMessage)
189 }
190
191 func Test_getAccountByID(t *testing.T) {
192         cc := new(Chaincode)
193         stub := shim.NewMockStub("get_account_test", cc)
194
195         // Init 1 account with 10 tokens
196         checkInit(t, stub, [][]byte{[]byte("10")})
197
198         // It should get account with ID "1" that was Init
199         args := [][]byte{[]byte("getAccountByID"), []byte("1")}
```

```
   200          expectedPayload := "{\"RecordType\":\"ACCOUNT\",\"AccountID\":\"1\",\"Name
\":\"Init_Account\",\"OwnerID\":\"\",\"Tokens\":10}"
   201          checkInvokeResponse(t, stub, args, expectedPayload)
   202
   203          // It should fail with empty string arg
   204          args = [][]byte{[]byte("getAccountByID"), []byte("")}
   205          checkInvokeFail(t, stub, args)
   206
   207          // It should fail with more than one arg
   208          args = [][]byte{[]byte("getAccountByID"), []byte("1"), []byte("a")}
   209          checkInvokeFail(t, stub, args)
   210
   211          /*
   212                  // This cannot be tested because of the limitations of MockStub implementation
   213                  // It should fail to get account that is not created
   214                  args = [][]byte{[]byte("getAccountByID"), []byte("2")}
   215                  expectedMessage := ""
   216                  checkInvokeResponseFail(t, stub, args, expectedMessage)
   217          */
   218  }
   219
   220  func Test_getAccountHistoryByID(t *testing.T) {
   221          cc := new(Chaincode)
   222          stub := shim.NewMockStub("get_history_test", cc)
   223
   224          // Init 1 account with 10 tokens
   225          checkInit(t, stub, [][]byte{[]byte("10")})
   226          /*
   227                  // This cannot be tested because of the limitations of MockStub implementation
   228                  // It should return history for account ID "1"
   229                  args := [][]byte{[]byte("getAccountHistoryByID"), []byte("1")}
```

```
230                 expectedPayload := ""
231                 checkInvokeResponse(t, stub, args, expectedPayload)
232         */
233
234         // It should fail with empty string arg
235         args := [][]byte{[]byte("getAccountHistoryByID"), []byte("")}
236         expectedMessage := "Argument at position 1 must be a non-empty string"
237         checkInvokeResponseFail(t, stub, args, expectedMessage)
238
239         // It should fail with more than one args
240         args = [][]byte{[]byte("getAccountHistoryByID"), []byte("1"), []byte("lol")}
241         expectedMessage = "Incorrect number of arguments. Expecting AccountID"
242         checkInvokeResponseFail(t, stub, args, expectedMessage)
243 }
244
245 func Test_getAccountByName(t *testing.T) {
246         cc := new(Chaincode)
247         stub := shim.NewMockStub("get_acc_test", cc)
248
249         // Init 1 account with 10 tokens
250         checkInit(t, stub, [][]byte{[]byte("10")})
251
252         // It should return one account
253         args := [][]byte{[]byte("getAccountByName"), []byte("Init_Account")}
254         expectedPayload := "[{\"RecordType\":\"ACCOUNT\",\"AccountID\":\"1\",\"Name
\":\"Init_Account\",\"OwnerID\":\"\",\"Tokens\":10}]"
255         checkInvokeResponse(t, stub, args, expectedPayload)
256
257         // create second account
258         args = [][]byte{[]byte("createAccount"), []byte("2"), []byte("Init_Account")}
259         expectedPayload = "Account created"
```

```
260         checkInvokeResponse(t, stub, args, expectedPayload)
261
262         // It should return JSON array of two accounts
263         args = [][]byte{[]byte("getAccountByName"), []byte("Init_Account")}
264         expectedPayload = "[{\"RecordType\":\"ACCOUNT\",\"AccountID\":\"1\",\"Name
\":\"Init_Account\",\"OwnerID\":\"\",\"Tokens\":10}" +
265                 "," +
266                 "{\"RecordType\":\"ACCOUNT\",\"AccountID\":\"2\",\"Name\":\"Init_Account
\",\"OwnerID\":\"\",\"Tokens\":0}]"
267         checkInvokeResponse(t, stub, args, expectedPayload)
268
269         // It should fail with empty string arg
270         args = [][]byte{[]byte("getAccountByName"), []byte("")}
271         expectedMessage := "Argument at position 1 must be a non-empty string"
272         checkInvokeResponseFail(t, stub, args, expectedMessage)
273
274         // It should fail with more than one args
275         args = [][]byte{[]byte("getAccountByName"), []byte("1"), []byte("lol")}
276         expectedMessage = "Incorrect number of arguments. Expecting name of account holder"
277         checkInvokeResponseFail(t, stub, args, expectedMessage)
278 }
279
280 func Test_sendTokensFast(t *testing.T) {
281         cc := new(Chaincode)
282         stub := shim.NewMockStub("tokens_init_test", cc)
283
284         // Init 1 account with 10 000 tokens
285         checkInit(t, stub, [][]byte{[]byte("10000")})
286
287         // create another acc without tokens
288         args := [][]byte{[]byte("createAccount"), []byte("2"), []byte("acc_name")}
```

```
   289          expectedPayload := "Account created"
   290          checkInvokeResponse(t, stub, args, expectedPayload)
   291          // It should transfer tokens that are not for data purchase immediatelly
   292          args = [][]byte{[]byte("sendTokensFast"), []byte("1"), []byte("2"), []byte("1"),
[]byte("false")}
   293          expectedPayload = "1"
   294          checkInvokeResponse(t, stub, args, expectedPayload)
   295          // Check the result
   296          args = [][]byte{[]byte("getAccountTokens"), []byte("1")}
   297          expectedPayload = "9999"
   298          checkInvokeResponse(t, stub, args, expectedPayload)
   299          args = [][]byte{[]byte("getAccountTokens"), []byte("2")}
   300          expectedPayload = "1"
   301          checkInvokeResponse(t, stub, args, expectedPayload)
   302
   303          // It should transfer tokens that are for data purchase and create pendingTx
   304          args = [][]byte{[]byte("sendTokensFast"), []byte("1"), []byte("2"), []byte("1"),
[]byte("true")}
   305          expectedPayload = "2"
   306          res := stub.MockInvoke("2", args)
   307          if res.Status != shim.OK {
   308                  fmt.Println("Invoke", args, "failed", string(res.Message))
   309                  t.Fail()
   310          }
   311          if string(res.Payload) != expectedPayload {
   312                  fmt.Println("Expected payload:", expectedPayload)
   313                  fmt.Println("Instead got this:", string(res.Payload))
   314                  t.Fail()
   315          }
   316          // Check the result
   317          args = [][]byte{[]byte("getAccountTokens"), []byte("1")}
```

```
   318          expectedPayload = "9998"
   319          checkInvokeResponse(t, stub, args, expectedPayload)
   320          // It is as pending therefore not available for account 2
   321          args = [][]byte{[]byte("getAccountTokens"), []byte("2")}
   322          expectedPayload = "1"
   323          checkInvokeResponse(t, stub, args, expectedPayload)
   324
   325          // It should not transfer tokens if tokens limit for fast transfer is exceeded
   326          args = [][]byte{[]byte("sendTokensFast"), []byte("1"), []byte("2"), []byte("11"),
[]byte("false")}
   327          expectedMessage := "Exceeded max number of tokens for fast transaction. Use safe token
transfer instead."
   328          checkInvokeResponseFail(t, stub, args, expectedMessage)
   329
   330          // It should not transfer negative amount of tokens
   331          args = [][]byte{[]byte("sendTokensFast"), []byte("1"), []byte("2"), []byte("-1"),
[]byte("false")}
   332          expectedMessage = "Expecting positive integer as number of tokens to transfer."
   333          checkInvokeResponseFail(t, stub, args, expectedMessage)
   334
   335          // It should not transfer tokens if sender and recipient acc is the same
   336          args = [][]byte{[]byte("sendTokensFast"), []byte("1"), []byte("1"), []byte("1"),
[]byte("false")}
   337          expectedMessage = "From account and to account cannot be the same."
   338          checkInvokeResponseFail(t, stub, args, expectedMessage)
   339
   340          // It should not transfer tokens if amount is not a number
   341          args = [][]byte{[]byte("sendTokensFast"), []byte("1"), []byte("2"), []byte("lol"),
[]byte("false")}
   342          expectedMessage = "Expecting positive integer as number of tokens to transfer."
   343          checkInvokeResponseFail(t, stub, args, expectedMessage)
```

```
   344
   345          // It should not transfer tokens if dataPurchase param is not bool value
   346          args = [][]byte{[]byte("sendTokensFast"), []byte("1"), []byte("2"), []byte("1"),
[]byte("lol")}
   347          expectedMessage = "Expecting boolean value. If this transfer is for data purchase or
not."
   348          checkInvokeResponseFail(t, stub, args, expectedMessage)
   349
   350          // It should fail with empty string arg
   351          args = [][]byte{[]byte("sendTokensFast"), []byte(""), []byte("2"), []byte("1"),
[]byte("false")}
   352          expectedMessage = "Argument at position 1 must be a non-empty string"
   353          checkInvokeResponseFail(t, stub, args, expectedMessage)
   354
   355          // It should fail with empty string arg
   356          args = [][]byte{[]byte("sendTokensFast"), []byte("1"), []byte(""), []byte("1"),
[]byte("false")}
   357          expectedMessage = "Argument at position 2 must be a non-empty string"
   358          checkInvokeResponseFail(t, stub, args, expectedMessage)
   359
   360          // It should fail with empty string arg
   361          args = [][]byte{[]byte("sendTokensFast"), []byte("1"), []byte("2"), []byte(""),
[]byte("false")}
   362          expectedMessage = "Argument at position 3 must be a non-empty string"
   363          checkInvokeResponseFail(t, stub, args, expectedMessage)
   364
   365          // It should fail with empty string arg
   366          args = [][]byte{[]byte("sendTokensFast"), []byte("1"), []byte("2"), []byte("1"),
[]byte("")}
   367          expectedMessage = "Argument at position 4 must be a non-empty string"
   368          checkInvokeResponseFail(t, stub, args, expectedMessage)
```

```
   369
   370          // It should fail with more than 4 args
   371          args = [][]byte{[]byte("sendTokensFast"), []byte("1"), []byte("2"), []byte("1"),
[]byte("false"), []byte("lol")}
   372          expectedMessage = "Incorrect number of arguments. Expecting FromAccountId, ToAccountId,
Amount, dataPurchase"
   373          checkInvokeResponseFail(t, stub, args, expectedMessage)
   374
   375          // It should fail with less than 4 args
   376          args = [][]byte{[]byte("sendTokensFast"), []byte("1"), []byte("2"), []byte("1")}
   377          expectedMessage = "Incorrect number of arguments. Expecting FromAccountId, ToAccountId,
Amount, dataPurchase"
   378          checkInvokeResponseFail(t, stub, args, expectedMessage)
   379
   380          /*
   381                  // These tests cannot be executed in Mock environment yet.
   382                  // They create panic even though chaincode is ok and should return error.
   383                  // It is because of the implementation of Mock state as map and it dereference
0 pointer
   384                  // It should not transfer tokens to account that does not exist
   385                  args = [][]byte{[]byte("sendTokensFast"), []byte("1"), []byte("3"),
[]byte("100")}
   386                  checkInvokeFail(t, stub, args)
   387
   388                  // It should not transfer tokens from account that does not exist
   389                  args = [][]byte{[]byte("sendTokensFast"), []byte("3"), []byte("1"),
[]byte("100")}
   390                  checkInvokeFail(t, stub, args)
   391
   392          */
   393  }
```

```
   394
   395  func Test_sendTokensSafe(t *testing.T) {
   396          cc := new(Chaincode)
   397          stub := shim.NewMockStub("tokens_init_test", cc)
   398
   399          // Init 1 account with 10 000 tokens
   400          checkInit(t, stub, [][]byte{[]byte("10000")})
   401
   402          // create another acc without tokens
   403          args := [][]byte{[]byte("createAccount"), []byte("2"), []byte("acc_name")}
   404          expectedPayload := "Account created"
   405          checkInvokeResponse(t, stub, args, expectedPayload)
   406          // It should transfer tokens
   407          args = [][]byte{[]byte("sendTokensSafe"), []byte("1"), []byte("2"), []byte("100"),
[]byte("false")}
   408          expectedPayload = "1"
   409          checkInvokeResponse(t, stub, args, expectedPayload)
   410          // Check the result
   411          args = [][]byte{[]byte("getAccountTokens"), []byte("1")}
   412          expectedPayload = "9900"
   413          checkInvokeResponse(t, stub, args, expectedPayload)
   414          args = [][]byte{[]byte("getAccountTokens"), []byte("2")}
   415          expectedPayload = "100"
   416          checkInvokeResponse(t, stub, args, expectedPayload)
   417
   418          // It should transfer tokens that are for data purchase and create pendingTx
   419          args = [][]byte{[]byte("sendTokensSafe"), []byte("1"), []byte("2"), []byte("100"),
[]byte("true")}
   420          expectedPayload = "2"
   421          res := stub.MockInvoke("2", args)
   422          if res.Status != shim.OK {
```

```
   423                 fmt.Println("Invoke", args, "failed", string(res.Message))
   424                 t.Fail()
   425         }
   426         if string(res.Payload) != expectedPayload {
   427                 fmt.Println("Expected payload:", expectedPayload)
   428                 fmt.Println("Instead got this:", string(res.Payload))
   429                 t.Fail()
   430         }
   431         // Check the result
   432         args = [][]byte{[]byte("getAccountTokens"), []byte("1")}
   433         expectedPayload = "9800"
   434         checkInvokeResponse(t, stub, args, expectedPayload)
   435         // It is as pending therefore not available for account 2
   436         args = [][]byte{[]byte("getAccountTokens"), []byte("2")}
   437         expectedPayload = "100"
   438         checkInvokeResponse(t, stub, args, expectedPayload)
   439
   440         // It should not transfer tokens that are not available
   441         args = [][]byte{[]byte("sendTokensSafe"), []byte("2"), []byte("1"), []byte("101"),
[]byte("false")}
   442         expectedPayload = "Not enough tokens on the sender's account"
   443         checkInvokeResponseFail(t, stub, args, expectedPayload)
   444
   445         // It should not transfer tokens if sender and recipient acc is the same
   446         args = [][]byte{[]byte("sendTokensSafe"), []byte("1"), []byte("1"), []byte("10"),
[]byte("false")}
   447         expectedMessage := "From account and to account cannot be the same."
   448         checkInvokeResponseFail(t, stub, args, expectedMessage)
   449
   450         // It should not transfer negative amount of tokens
   451         args = [][]byte{[]byte("sendTokensSafe"), []byte("1"), []byte("2"), []byte("-10"),
```

```
[]byte("false")}
   452          expectedMessage = "Expecting positive integer as number of tokens to transfer."
   453          checkInvokeResponseFail(t, stub, args, expectedMessage)
   454
   455          // It should not transfer tokens if amount is not a number
   456          args = [][]byte{[]byte("sendTokensSafe"), []byte("1"), []byte("2"), []byte("lol"),
[]byte("false")}
   457          expectedMessage = "Expecting positive integer as number of tokens to transfer."
   458          checkInvokeResponseFail(t, stub, args, expectedMessage)
   459
   460          // It should not transfer tokens if dataPurchase param is not bool value
   461          args = [][]byte{[]byte("sendTokensSafe"), []byte("1"), []byte("2"), []byte("1"),
[]byte("lol")}
   462          expectedMessage = "Expecting boolean value. If this transfer is for data purchase or
not."
   463          checkInvokeResponseFail(t, stub, args, expectedMessage)
   464
   465          // It should fail with empty string arg
   466          args = [][]byte{[]byte("sendTokensSafe"), []byte(""), []byte("2"), []byte("10"),
[]byte("false")}
   467          expectedMessage = "Argument at position 1 must be a non-empty string"
   468          checkInvokeResponseFail(t, stub, args, expectedMessage)
   469
   470          // It should fail with empty string arg
   471          args = [][]byte{[]byte("sendTokensSafe"), []byte("1"), []byte(""), []byte("10"),
[]byte("false")}
   472          expectedMessage = "Argument at position 2 must be a non-empty string"
   473          checkInvokeResponseFail(t, stub, args, expectedMessage)
   474
   475          // It should fail with empty string arg
   476          args = [][]byte{[]byte("sendTokensSafe"), []byte("1"), []byte("2"), []byte(""),
```

```
[]byte("false")}
   477          expectedMessage = "Argument at position 3 must be a non-empty string"
   478          checkInvokeResponseFail(t, stub, args, expectedMessage)
   479
   480          // It should fail with empty string arg
   481          args = [][]byte{[]byte("sendTokensSafe"), []byte("1"), []byte("2"), []byte("1"),
[]byte("")}
   482          expectedMessage = "Argument at position 4 must be a non-empty string"
   483          checkInvokeResponseFail(t, stub, args, expectedMessage)
   484
   485          // It should fail with more than 4 args
   486          args = [][]byte{[]byte("sendTokensSafe"), []byte("1"), []byte("2"), []byte("100"),
[]byte("false"), []byte("lol")}
   487          expectedMessage = "Incorrect number of arguments. Expecting FromAccountId, ToAccountId,
Amount, dataPurchase"
   488          checkInvokeResponseFail(t, stub, args, expectedMessage)
   489
   490          // It should fail with less than 4 args
   491          args = [][]byte{[]byte("sendTokensSafe"), []byte("1"), []byte("2"), []byte("1")}
   492          expectedMessage = "Incorrect number of arguments. Expecting FromAccountId, ToAccountId,
Amount, dataPurchase"
   493          checkInvokeResponseFail(t, stub, args, expectedMessage)
   494
   495          /*
   496                  // These tests cannot be executed in Mock environment yet.
   497                  // They create panic even though chaincode is ok and should return error.
   498                  // It is because of the implementation of Mock state as map and it dereference
0 pointer
   499                  // It should not transfer tokens to account that does not exist
   500                  args = [][]byte{[]byte("sendTokensSafe"), []byte("1"), []byte("3"),
[]byte("100")}
```

```
   501                  checkInvokeFail(t, stub, args)
   502
   503                  // It should not transfer tokens from account that does not exist
   504                  args = [][]byte{[]byte("sendTokensSafe"), []byte("3"), []byte("1"),
[]byte("100")}
   505                  checkInvokeFail(t, stub, args)
   506
   507          */
   508  }
   509
   510  func Test_getTxDetails(t *testing.T) {
   511          cc := new(Chaincode)
   512          stub := shim.NewMockStub("tokens_init_test", cc)
   513
   514          // Init 1 account with 10 000 tokens
   515          checkInit(t, stub, [][]byte{[]byte("10000")})
   516
   517          // create another acc without tokens
   518          args := [][]byte{[]byte("createAccount"), []byte("2"), []byte("acc_name")}
   519          expectedPayload := "Account created"
   520          checkInvokeResponse(t, stub, args, expectedPayload)
   521          // It should transfer tokens
   522          args = [][]byte{[]byte("sendTokensFast"), []byte("1"), []byte("2"), []byte("1"),
[]byte("false")}
   523          res := stub.MockInvoke("2", args)
   524          if res.Status != shim.OK {
   525                  fmt.Println("Invoke", args, "failed", string(res.Message))
   526                  t.Fail()
   527          }
   528          // get the TxID and participants
   529          args = [][]byte{[]byte("getTxDetails"), res.Payload}
```

```
530         expectedPayload = "1->2->1->ValidTx"
531         checkInvokeResponse(t, stub, args, expectedPayload)
532
533         // It should fail with random TxID
534         args = [][]byte{[]byte("getTxDetails"), []byte("-4863asfaebh")}
535         expectedMessage := "Transaction was not found."
536         checkInvokeResponseFail(t, stub, args, expectedMessage)
537
538         // It should fail with empty string arg
539         args = [][]byte{[]byte("getTxDetails"), []byte("")}
540         expectedMessage = "Argument at position 1 must be a non-empty string"
541         checkInvokeResponseFail(t, stub, args, expectedMessage)
542
543         // It should fail with more than one args
544         args = [][]byte{[]byte("getTxDetails"), []byte("1"), []byte("lol")}
545         expectedMessage = "Incorrect number of arguments. Expecting TxID"
546         checkInvokeResponseFail(t, stub, args, expectedMessage)
547 }
548
549 // For this function we cannot test more because of the MockStub limitations
550 func Test_changePendingTx(t *testing.T) {
551         cc := new(Chaincode)
552         stub := shim.NewMockStub("tokens_init_test", cc)
553
554         // Init 1 account with 10 000 tokens
555         checkInit(t, stub, [][]byte{[]byte("10000")})
556
557         // It should fail to changePendingTx with less than 3 args
558         args := [][]byte{[]byte("changePendingTx"), []byte("channel1"), []byte("chaincode_ad")}
559         expectedMessage := "Incorrect number of arguments. Expecting 3"
560         checkInvokeResponseFail(t, stub, args, expectedMessage)
```

```
561
562         // It should fail to changePendingTx with more than 3 args
563         args = [][]byte{[]byte("changePendingTx"), []byte("channel1"), []byte("chaincode_ad"),
564                 []byte("TxID-1"), []byte("extra_arg")}
565         expectedMessage = "Incorrect number of arguments. Expecting 3"
566         checkInvokeResponseFail(t, stub, args, expectedMessage)
567
568         // It should fail to changePendingTx with empty string arg
569         args = [][]byte{[]byte("changePendingTx"), []byte(""), []byte("chaincode_ad"),
570                 []byte("TxID-1")}
571         expectedMessage = "Argument at position 1 must be a non-empty string"
572         checkInvokeResponseFail(t, stub, args, expectedMessage)
573
574         // It should fail to changePendingTx with empty string arg
575         args = [][]byte{[]byte("changePendingTx"), []byte("channel1"), []byte(""),
576                 []byte("TxID-1")}
577         expectedMessage = "Argument at position 2 must be a non-empty string"
578         checkInvokeResponseFail(t, stub, args, expectedMessage)
579
580         // It should fail to changePendingTx with empty string arg
581         args = [][]byte{[]byte("changePendingTx"), []byte("channel1"), []byte("chaincode_ad"),
582                 []byte("")}
583         expectedMessage = "Argument at position 3 must be a non-empty string"
584         checkInvokeResponseFail(t, stub, args, expectedMessage)
585 }
586
587 func Test_pruneAccountTx(t *testing.T) {
588         cc := new(Chaincode)
589         stub := shim.NewMockStub("tokens_init_test", cc)
590
591         // Init 1 account with 10 000 tokens
```

```
   592          checkInit(t, stub, [][]byte{[]byte("10000")})
   593
   594          // create another acc without tokens
   595          args := [][]byte{[]byte("createAccount"), []byte("2"), []byte("acc_name")}
   596          expectedPayload := "Account created"
   597          checkInvokeResponse(t, stub, args, expectedPayload)
   598          // It should transfer tokens
   599          args = [][]byte{[]byte("sendTokensFast"), []byte("1"), []byte("2"), []byte("1"),
[]byte("false")}
   600          res := stub.MockInvoke("2", args)
   601          if res.Status != shim.OK {
   602                  fmt.Println("Invoke", args, "failed", string(res.Message))
   603                  t.Fail()
   604          }
   605          args = [][]byte{[]byte("sendTokensFast"), []byte("1"), []byte("2"), []byte("1"),
[]byte("false")}
   606          res = stub.MockInvoke("3", args)
   607          if res.Status != shim.OK {
   608                  fmt.Println("Invoke", args, "failed", string(res.Message))
   609                  t.Fail()
   610          }
   611          // prune Tx for acc ID
   612          args = [][]byte{[]byte("pruneAccountTx"), []byte("2")}
   613          res = stub.MockInvoke("4", args)
   614          if res.Status != shim.OK {
   615                  fmt.Println("Invoke", args, "failed", string(res.Message))
   616                  t.Fail()
   617          }
   618
   619          // get the TxID and participants
   620          args = [][]byte{[]byte("getTxDetails"), []byte("4")}
```

```
621         expectedPayload = "pruneTx->2->2->ValidTx"
622         checkInvokeResponse(t, stub, args, expectedPayload)
623
624         // It should fail with empty string arg
625         args = [][]byte{[]byte("pruneAccountTx"), []byte("")}
626         expectedMessage := "Argument at position 1 must be a non-empty string"
627         checkInvokeResponseFail(t, stub, args, expectedMessage)
628
629         // It should fail with more than one args
630         args = [][]byte{[]byte("pruneAccountTx"), []byte("1"), []byte("lol")}
631         expectedMessage = "Incorrect number of arguments. Expecting account ID"
632         checkInvokeResponseFail(t, stub, args, expectedMessage)
633 }
634
635 func Test_updateAccountTokens(t *testing.T) {
636         cc := new(Chaincode)
637         stub := shim.NewMockStub("tokens_init_test", cc)
638
639         // Init 1 account with 10 000 tokens
640         checkInit(t, stub, [][]byte{[]byte("10000")})
641
642         // create another acc without tokens
643         args := [][]byte{[]byte("createAccount"), []byte("2"), []byte("acc_name")}
644         expectedPayload := "Account created"
645         checkInvokeResponse(t, stub, args, expectedPayload)
646
647         // send tokens to account 2
648         args = [][]byte{[]byte("sendTokensSafe"), []byte("1"), []byte("2"), []byte("100"),
[]byte("false")}
649         expectedPayload = "1"
650         checkInvokeResponse(t, stub, args, expectedPayload)
```

```
   651
   652          // accounts should have the initial value because they were not updated yet
   653          args = [][]byte{[]byte("getAccountByID"), []byte("1")}
   654          expectedPayload = "{\"RecordType\":\"ACCOUNT\",\"AccountID\":\"1\",\"Name
\":\"Init_Account\",\"OwnerID\":\"\",\"Tokens\":10000}"
   655          checkInvokeResponse(t, stub, args, expectedPayload)
   656          args = [][]byte{[]byte("getAccountByID"), []byte("2")}
   657          expectedPayload = "{\"RecordType\":\"ACCOUNT\",\"AccountID\":\"2\",\"Name\":\"acc_name
\",\"OwnerID\":\"\",\"Tokens\":0}"
   658          checkInvokeResponse(t, stub, args, expectedPayload)
   659
   660          // Check if Tx was successful
   661          args = [][]byte{[]byte("getAccountTokens"), []byte("1")}
   662          expectedPayload = "9900"
   663          checkInvokeResponse(t, stub, args, expectedPayload)
   664          // Check if Tx was successful
   665          args = [][]byte{[]byte("getAccountTokens"), []byte("2")}
   666          expectedPayload = "100"
   667          checkInvokeResponse(t, stub, args, expectedPayload)
   668
   669          // Update the amount of tokens on account 1
   670          args = [][]byte{[]byte("updateAccountTokens"), []byte("1")}
   671          expectedPayload = "{\"RecordType\":\"ACCOUNT\",\"AccountID\":\"1\",\"Name
\":\"Init_Account\",\"OwnerID\":\"\",\"Tokens\":9900}"
   672          checkInvokeResponse(t, stub, args, expectedPayload)
   673          // Update the amount of tokens on account 2
   674          args = [][]byte{[]byte("updateAccountTokens"), []byte("2")}
   675          expectedPayload = "{\"RecordType\":\"ACCOUNT\",\"AccountID\":\"2\",\"Name\":\"acc_name
\",\"OwnerID\":\"\",\"Tokens\":100}"
   676          checkInvokeResponse(t, stub, args, expectedPayload)
   677
```

```
   678          // accounts should have the updated value
   679          args = [][]byte{[]byte("getAccountByID"), []byte("1")}
   680          expectedPayload = "{\"RecordType\":\"ACCOUNT\",\"AccountID\":\"1\",\"Name
\":\"Init_Account\",\"OwnerID\":\"\",\"Tokens\":9900}"
   681          checkInvokeResponse(t, stub, args, expectedPayload)
   682          args = [][]byte{[]byte("getAccountByID"), []byte("2")}
   683          expectedPayload = "{\"RecordType\":\"ACCOUNT\",\"AccountID\":\"2\",\"Name\":\"acc_name
\",\"OwnerID\":\"\",\"Tokens\":100}"
   684          checkInvokeResponse(t, stub, args, expectedPayload)
   685
   686          // It should fail with empty string arg
   687          args = [][]byte{[]byte("updateAccountTokens"), []byte("")}
   688          expectedMessage := "Argument at position 1 must be a non-empty string"
   689          checkInvokeResponseFail(t, stub, args, expectedMessage)
   690
   691          // It should fail with more than one args
   692          args = [][]byte{[]byte("updateAccountTokens"), []byte("1"), []byte("lol")}
   693          expectedMessage = "Incorrect number of arguments. Expecting account ID"
   694          checkInvokeResponseFail(t, stub, args, expectedMessage)
   695  }
   696
   697  func Test_getAccountTokens(t *testing.T) {
   698          cc := new(Chaincode)
   699          stub := shim.NewMockStub("tokens_init_test", cc)
   700
   701          // Init 1 account with 10 000 tokens
   702          checkInit(t, stub, [][]byte{[]byte("10000")})
   703
   704          // create another acc without tokens
   705          args := [][]byte{[]byte("createAccount"), []byte("2"), []byte("acc_name")}
   706          expectedPayload := "Account created"
```

```
   707          checkInvokeResponse(t, stub, args, expectedPayload)
   708
   709          // send tokens to account 2
   710          args = [][]byte{[]byte("sendTokensSafe"), []byte("1"), []byte("2"), []byte("100"),
[]byte("false")}
   711          expectedPayload = "1"
   712          checkInvokeResponse(t, stub, args, expectedPayload)
   713
   714          // accounts should have the initial value because they were not updated yet
   715          args = [][]byte{[]byte("getAccountByID"), []byte("1")}
   716          expectedPayload = "{\"RecordType\":\"ACCOUNT\",\"AccountID\":\"1\",\"Name
\":\"Init_Account\",\"OwnerID\":\"\",\"Tokens\":10000}"
   717          checkInvokeResponse(t, stub, args, expectedPayload)
   718          args = [][]byte{[]byte("getAccountByID"), []byte("2")}
   719          expectedPayload = "{\"RecordType\":\"ACCOUNT\",\"AccountID\":\"2\",\"Name\":\"acc_name
\",\"OwnerID\":\"\",\"Tokens\":0}"
   720          checkInvokeResponse(t, stub, args, expectedPayload)
   721
   722          // Check if Tx was successful
   723          args = [][]byte{[]byte("getAccountTokens"), []byte("1")}
   724          expectedPayload = "9900"
   725          checkInvokeResponse(t, stub, args, expectedPayload)
   726          // Check if Tx was successful
   727          args = [][]byte{[]byte("getAccountTokens"), []byte("2")}
   728          expectedPayload = "100"
   729          checkInvokeResponse(t, stub, args, expectedPayload)
   730
   731          // It should fail with empty string arg
   732          args = [][]byte{[]byte("getAccountTokens"), []byte("")}
   733          expectedMessage := "Argument at position 1 must be a non-empty string"
   734          checkInvokeResponseFail(t, stub, args, expectedMessage)
```

```
735
736         // It should fail with more than one args
737         args = [][]byte{[]byte("getAccountTokens"), []byte("1"), []byte("lol")}
738         expectedMessage = "Incorrect number of arguments. Expecting account ID"
739         checkInvokeResponseFail(t, stub, args, expectedMessage)
740 }
741
```

View as plain text