# Source file src/chaincode/chaincode_data/chaincode_data_test.go

Documentation: **chaincode/chaincode_data**

```
 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 checkInvoke(t *testing.T, stub *shim.MockStub, args [][]byte) {
    27          res := stub.MockInvoke("1", args)
    28          if res.Status != shim.OK {
    29                  fmt.Println("Invoke", args, "failed", string(res.Message))
    30                  t.Fail()
    31          }
    32  }
    33
    34  func checkInvokeFail(t *testing.T, stub *shim.MockStub, args [][]byte) {
    35          res := stub.MockInvoke("1", args)
    36          if res.Status == shim.OK {
    37                  fmt.Println("Invoke", args, "should fail but did not", string(res.Payload))
    38                  t.Fail()
    39          }
    40  }
    41
    42  func checkInvokeResponse(t *testing.T, stub *shim.MockStub, args [][]byte, expectedPayload
string) {
    43          res := stub.MockInvoke("1", args)
    44          if res.Status != shim.OK {
    45                  fmt.Println("Invoke", args, "failed", string(res.Message))
    46                  t.Fail()
    47          }
    48          if string(res.Payload) != expectedPayload {
    49                  fmt.Println("Expected payload:", expectedPayload)
    50                  fmt.Println("Instead got this:", string(res.Payload))
```

```
    51                  t.Fail()
    52          }
    53  }
    54
    55  func checkInvokeResponseFail(t *testing.T, stub *shim.MockStub, args [][]byte, expectedMessage
string) {
    56          res := stub.MockInvoke("1", args)
    57          if res.Status == shim.OK {
    58                  fmt.Println("Invoke", args, "should fail")
    59                  fmt.Println("Instead got payload:", string(res.Payload))
    60                  t.Fail()
    61          }
    62          if res.Message != expectedMessage {
    63                  fmt.Println("Expected message:", expectedMessage)
    64                  fmt.Println("Instead got this:", res.Message)
    65                  t.Fail()
    66          }
    67  }
    68  func Test_init(t *testing.T) {
    69          cc := new(Chaincode)
    70          stub := shim.NewMockStub("init_test", cc)
    71
    72          // Init should always success
    73          checkInit(t, stub, [][]byte{[]byte("1")})
    74  }
    75
    76  func Test_InvokeFail(t *testing.T) {
    77          cc := new(Chaincode)
    78          stub := shim.NewMockStub("invoke_fail_test", cc)
    79          args := [][]byte{[]byte("NoFunction"), []byte("test")}
    80          expectedMessage := "Received unknown function invocation"
```

```
 81          checkInvokeResponseFail(t, stub, args, expectedMessage)
 82  }
 83
 84  func Test_createData(t *testing.T) {
 85          cc := new(Chaincode)
 86          stub := shim.NewMockStub("init_test", cc)
 87
 88          // Init
 89          checkInit(t, stub, [][]byte{[]byte("20")})
 90          // create test data
 91          args := [][]byte{[]byte("createData"),
 92                  []byte("1"), []byte("test_data"), []byte("10"), []byte("Unit"),
 93                  []byte("20181212152030"), []byte("pub_name")}
 94          checkInvokeResponse(t, stub, args, "")
 95
 96          // Check if it is in the state
 97          args = [][]byte{[]byte("getDataByIDAndTime"), []byte("1"), []byte("20181212152030")}
 98          expectedPayload := "{\"RecordType\":\"DATA_ENTRY\",\"DataEntryID\":\"1\"" +
 99                  ",\"Description\":\"test_data\",\"Value\":\"10\",\"Unit\":\"Unit\"," +
100                  "\"CreationTime\":20181212152030,\"Publisher\":\"pub_name\"}"
101          checkInvokeResponse(t, stub, args, expectedPayload)
102
103          // create test data that has the same ID and creationTime Shoult fail
104          args = [][]byte{[]byte("createData"),
105                  []byte("1"), []byte("test_data"), []byte("50"), []byte("Unit"),
106                  []byte("20181212152030"), []byte("pub_name")}
107          expectedMessage := "This data entry already exists: 1~20181212152030"
108          checkInvokeResponseFail(t, stub, args, expectedMessage)
109
110          // It should fail to createData that have one empty arg
111          args = [][]byte{[]byte("createData"),
```

```
112                 []byte(""), []byte("test_data"), []byte("10"), []byte("Unit"),
113                 []byte("20181212152030"), []byte("pub_name")}
114         // it should not save to the state and it should fail
115         expectedMessage = "Argument at position 1 must be a non-empty string"
116         checkInvokeResponseFail(t, stub, args, expectedMessage)
117
118         // It should fail to createData that have one empty arg
119         args = [][]byte{[]byte("createData"),
120                 []byte("2"), []byte(""), []byte("10"), []byte("Unit"),
121                 []byte("20181212152030"), []byte("pub_name")}
122         // it should not save to the state and it should fail
123         expectedMessage = "Argument at position 2 must be a non-empty string"
124         checkInvokeResponseFail(t, stub, args, expectedMessage)
125
126         // It should fail to createData that have one empty arg
127         args = [][]byte{[]byte("createData"),
128                 []byte("2"), []byte("test_data"), []byte(""), []byte("Unit"),
129                 []byte("20181212152030"), []byte("pub_name")}
130         // it should not save to the state and it should fail
131         expectedMessage = "Argument at position 3 must be a non-empty string"
132         checkInvokeResponseFail(t, stub, args, expectedMessage)
133
134         // It should fail to createData that have one empty arg
135         args = [][]byte{[]byte("createData"),
136                 []byte("2"), []byte("test_data"), []byte("10"), []byte(""),
137                 []byte("20181212152030"), []byte("pub_name")}
138         // it should not save to the state and it should fail
139         expectedMessage = "Argument at position 4 must be a non-empty string"
140         checkInvokeResponseFail(t, stub, args, expectedMessage)
141
142         // It should fail to createData that have one empty arg
```

```
143         args = [][]byte{[]byte("createData"),
144                 []byte("2"), []byte("test_data"), []byte("10"), []byte("Unit"),
145                 []byte(""), []byte("pub_name")}
146         // it should not save to the state and it should fail
147         expectedMessage = "Argument at position 5 must be a non-empty string"
148         checkInvokeResponseFail(t, stub, args, expectedMessage)
149
150         // It should fail to createData that have one empty arg
151         args = [][]byte{[]byte("createData"),
152                 []byte("2"), []byte("test_data"), []byte("10"), []byte("Unit"),
153                 []byte("20181212152030"), []byte("")}
154         // it should not save to the state and it should fail
155         expectedMessage = "Argument at position 6 must be a non-empty string"
156         checkInvokeResponseFail(t, stub, args, expectedMessage)
157
158         // It should fail to createData with less than 6 args
159         args = [][]byte{[]byte("createData"),
160                 []byte("2"), []byte("test_data"), []byte("10"), []byte("Unit"),
161                 []byte("20181212152030")}
162         // it should not save to the state and it should fail
163         expectedMessage = "Incorrect number of arguments. Expecting 6"
164         checkInvokeResponseFail(t, stub, args, expectedMessage)
165
166         // It should fail to createData with more than 6 args
167         args = [][]byte{[]byte("createData"),
168                 []byte("2"), []byte("test_data"), []byte("10"), []byte("Unit"),
169                 []byte("20181212152030"), []byte("pub_name"), []byte("lol")}
170         // it should not save to the state and it should fail
171         expectedMessage = "Incorrect number of arguments. Expecting 6"
172         checkInvokeResponseFail(t, stub, args, expectedMessage)
173
```

```
174         // It should fail to createData for negative creationTime
175         args = [][]byte{[]byte("createData"),
176                 []byte("2"), []byte("test_data"), []byte("10"), []byte("Unit"),
177                 []byte("-20181212152030"), []byte("pub_name")}
178         // it should not save to the state and it should fail
179         expectedMessage = "Expecting positiv integer or zero as creation time."
180         checkInvokeResponseFail(t, stub, args, expectedMessage)
181 }
182
183 func Test_getDataByIDAndTime(t *testing.T) {
184         cc := new(Chaincode)
185         stub := shim.NewMockStub("init_test", cc)
186
187         // Init
188         checkInit(t, stub, [][]byte{[]byte("1")})
189         // create test data
190         args := [][]byte{[]byte("createData"),
191                 []byte("1"), []byte("test_data"), []byte("10"), []byte("Unit"),
192                 []byte("20181212152030"), []byte("pub_name")}
193         // it should save to the state
194         checkInvokeResponse(t, stub, args, "")
195
196         expectedPayload := "{\"RecordType\":\"DATA_ENTRY\",\"DataEntryID\":\"1\"" +
197                 ",\"Description\":\"test_data\",\"Value\":\"10\",\"Unit\":\"Unit\"," +
198                 "\"CreationTime\":20181212152030,\"Publisher\":\"pub_name\"}"
199
200         // It should get the same expected payload
201         args = [][]byte{[]byte("getDataByIDAndTime"), []byte("1"), []byte("20181212152030")}
202         checkInvokeResponse(t, stub, args, expectedPayload)
203
204         // It should fail to get data that have one empty arg
```

```
205         args = [][]byte{[]byte("getDataByIDAndTime"), []byte(""), []byte("20181212152030")}
206         expectedMessage := "Argument at position 1 must be a non-empty string"
207         checkInvokeResponseFail(t, stub, args, expectedMessage)
208
209         // It should fail to get data that have one empty arg
210         args = [][]byte{[]byte("getDataByIDAndTime"), []byte("1"), []byte("")}
211         expectedMessage = "Argument at position 2 must be a non-empty string"
212         checkInvokeResponseFail(t, stub, args, expectedMessage)
213
214         // It should fail to get data that have less than 2 args
215         args = [][]byte{[]byte("getDataByIDAndTime"), []byte("1")}
216         expectedMessage = "Incorrect number of arguments. Expecting data entry Id to get"
217         checkInvokeResponseFail(t, stub, args, expectedMessage)
218
219         // It should fail to get data that have more than 2 args
220         args = [][]byte{[]byte("getDataByIDAndTime"), []byte("1")}
221         expectedMessage = "Incorrect number of arguments. Expecting data entry Id to get"
222         checkInvokeResponseFail(t, stub, args, expectedMessage)
223 }
224
225 func Test_getAllDataByID(t *testing.T) {
226         cc := new(Chaincode)
227         stub := shim.NewMockStub("init_test", cc)
228
229         // Init
230         checkInit(t, stub, [][]byte{[]byte("1")})
231         // create test data
232         args := [][]byte{[]byte("createData"),
233                 []byte("1"), []byte("test_data"), []byte("10"), []byte("Unit"),
234                 []byte("20181212152030"), []byte("pub_name")}
235         // it should save to the state
```

```
236         checkInvokeResponse(t, stub, args, "")
237         expectedPayload := "{\"RecordType\":\"DATA_ENTRY\",\"DataEntryID\":\"1\"" +
238                 ",\"Description\":\"test_data\",\"Value\":\"10\",\"Unit\":\"Unit\"," +
239                 "\"CreationTime\":20181212152030,\"Publisher\":\"pub_name\"}"
240
241         // It should get the same expected payload
242         args = [][]byte{[]byte("getAllDataByID"), []byte("1")}
243         checkInvokeResponse(t, stub, args, "["+expectedPayload+"]")
244
245         // create second test data
246         args = [][]byte{[]byte("createData"),
247                 []byte("1"), []byte("test_data"), []byte("100"), []byte("Unit"),
248                 []byte("20181212152031"), []byte("pub_name")}
249         // it should save to the state
250         checkInvoke(t, stub, args)
251         expectedPayload2 := "{\"RecordType\":\"DATA_ENTRY\",\"DataEntryID\":\"1\"" +
252                 ",\"Description\":\"test_data\",\"Value\":\"100\",\"Unit\":\"Unit\"," +
253                 "\"CreationTime\":20181212152031,\"Publisher\":\"pub_name\"}"
254
255         // It should get both entry as JSON array
256         args = [][]byte{[]byte("getAllDataByID"), []byte("1")}
257         expectedPayload3 := "[" + expectedPayload + "," + expectedPayload2 + "]"
258         checkInvokeResponse(t, stub, args, expectedPayload3)
259
260         // It should not find entry that is not in state
261         args = [][]byte{[]byte("getAllDataByID"), []byte("2")}
262         // expected only empty array
263         expectedPayload = "[]"
264         checkInvokeResponse(t, stub, args, expectedPayload)
265
266         // It should fail with empty arg
```

```
267         args = [][]byte{[]byte("getAllDataByID"), []byte("")}
268         expectedPayload = "Argument at position 1 must be a non-empty string"
269         checkInvokeResponseFail(t, stub, args, expectedPayload)
270
271         // It should fail with empty more than one arg
272         args = [][]byte{[]byte("getAllDataByID"), []byte("1"), []byte("1")}
273         expectedPayload = "Incorrect number of arguments. Expecting data entry Id to get"
274         checkInvokeResponseFail(t, stub, args, expectedPayload)
275 }
276
277 func Test_getLatestDataByID(t *testing.T) {
278         cc := new(Chaincode)
279         stub := shim.NewMockStub("init_test", cc)
280
281         // Init
282         checkInit(t, stub, [][]byte{[]byte("1")})
283         // create test data
284         args := [][]byte{[]byte("createData"),
285                 []byte("1"), []byte("test_data"), []byte("10"), []byte("Unit"),
286                 []byte("20181212152030"), []byte("pub_name")}
287         // it should save to the state
288         checkInvokeResponse(t, stub, args, "")
289
290         // create second test data that are produced later in time
291         args = [][]byte{[]byte("createData"),
292                 []byte("1"), []byte("test_data"), []byte("100"), []byte("Unit"),
293                 []byte("20181212152031"), []byte("pub_name")}
294         // it should save to the state
295         checkInvoke(t, stub, args)
296         expectedPayload := "{\"RecordType\":\"DATA_ENTRY\",\"DataEntryID\":\"1\"" +
297                 ",\"Description\":\"test_data\",\"Value\":\"100\",\"Unit\":\"Unit\"," +
```

```
298                 "\"CreationTime\":20181212152031,\"Publisher\":\"pub_name\"}"
299
300         // It should get both entry as JSON array
301         args = [][]byte{[]byte("getLatestDataByID"), []byte("1")}
302         checkInvokeResponse(t, stub, args, expectedPayload)
303
304         // It should fail with empty arg
305         args = [][]byte{[]byte("getLatestDataByID"), []byte("")}
306         expectedPayload = "Argument at position 1 must be a non-empty string"
307         checkInvokeResponseFail(t, stub, args, expectedPayload)
308
309         // It should fail with empty more than one arg
310         args = [][]byte{[]byte("getLatestDataByID"), []byte("1"), []byte("1")}
311         expectedPayload = "Incorrect number of arguments. Expecting data entry Id to get"
312         checkInvokeResponseFail(t, stub, args, expectedPayload)
313 }
314
315 func Test_getDataByPub(t *testing.T) {
316         cc := new(Chaincode)
317         stub := shim.NewMockStub("init_test", cc)
318
319         // Init
320         checkInit(t, stub, [][]byte{[]byte("1")})
321         // create test data
322         args := [][]byte{[]byte("createData"),
323                 []byte("1"), []byte("test_data"), []byte("10"), []byte("Unit"),
324                 []byte("20181212152030"), []byte("pub_name")}
325         // it should save to the state
326         checkInvokeResponse(t, stub, args, "")
327         expectedPayload := "{\"RecordType\":\"DATA_ENTRY\",\"DataEntryID\":\"1\"" +
328                 ",\"Description\":\"test_data\",\"Value\":\"10\",\"Unit\":\"Unit\"," +
```

```
329                 "\"CreationTime\":20181212152030,\"Publisher\":\"pub_name\"}"
330
331         // It should get the same expected payload
332         args = [][]byte{[]byte("getDataByPub"), []byte("pub_name")}
333         checkInvokeResponse(t, stub, args, "["+expectedPayload+"]")
334
335         // create second test data
336         args = [][]byte{[]byte("createData"),
337                 []byte("2"), []byte("test_data"), []byte("100"), []byte("Unit"),
338                 []byte("20181212152030"), []byte("pub_name")}
339         // it should save to the state
340         checkInvoke(t, stub, args)
341         expectedPayload2 := "{\"RecordType\":\"DATA_ENTRY\",\"DataEntryID\":\"2\"" +
342                 ",\"Description\":\"test_data\",\"Value\":\"100\",\"Unit\":\"Unit\"," +
343                 "\"CreationTime\":20181212152030,\"Publisher\":\"pub_name\"}"
344
345         // It should get both entry as JSON array
346         args = [][]byte{[]byte("getDataByPub"), []byte("pub_name")}
347         expectedPayload3 := "[" + expectedPayload + "," + expectedPayload2 + "]"
348         checkInvokeResponse(t, stub, args, expectedPayload3)
349
350         // It should not find entry that is not in state
351         args = [][]byte{[]byte("getDataByPub"), []byte("pub_name2")}
352         // expected only empty array
353         expectedPayload = "[]"
354         checkInvokeResponse(t, stub, args, expectedPayload)
355
356         // It should fail with empty arg
357         args = [][]byte{[]byte("getDataByPub"), []byte("")}
358         expectedPayload = "Argument at position 1 must be a non-empty string"
359         checkInvokeResponseFail(t, stub, args, expectedPayload)
```

```
360
361         // It should fail with empty more than one arg
362         args = [][]byte{[]byte("getDataByPub"), []byte("pub_name"), []byte("pub_name")}
363         expectedPayload = "Incorrect number of arguments. Expecting publisher to get"
364         checkInvokeResponseFail(t, stub, args, expectedPayload)
365 }
366
```

View as plain text