# AutoTest Studio Getting started tutorials 7: Data Dictionary

AutoTest Studio provides a data dictionary module with excellent performance. Unlike other test frameworks that use text to store data, AutoTest Stduio uses Sqlite as its internal storage engine. The performance and stability of Sqlite are recognized in the industry. According to the official introduction, AutoTest Stduio can still provide good read performance even if it stores millions of records.

It should be pointed out that the data dictionary of AutoTest Studio is read-only during the running of the task, that is, users cannot write or modify data into the data dictionary through the function interface, and can only write or modify data in advance through the Data Dictionary tool. This design ensures that the test data will not be polluted during the running of the task.

The data dictionary of AutoTest Studio organizes the data hierarchical relationship in a directory structure. This design makes the data dictionary organization relationship with good readability. In the data dictionary, there are two concepts, data set (Dataset) and data item (Dataitem).

**Dataset**, a collection of the same type of data, such as the basic information of a test user, the Dataset is globally unique, that is, in the same data dictionary, the name of the data set cannot be repeated, even in different directories.

**Dataitem**, a data record under the data set, that is, a key-value record.

When the project is created, AutoTest Studio creates an empty data dictionary for each project by default.

The data dictionary module provides two operation interfaces.

Read the entire data set and return a python dictionary structure.

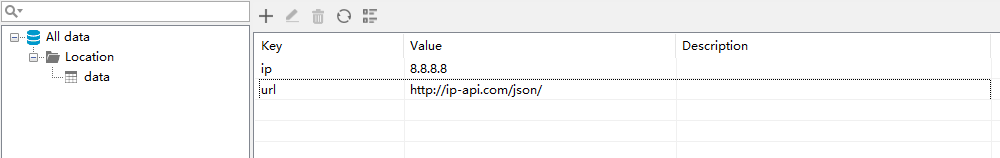
GetDataset (dataset)

Read the value corresponding to the key in the data set.

GetDataValue(dataset, key)

Let's take the demo in the previous chapter as an example, store the values ​​of the two variables directly defined in the demo, ip and ipQueryUrl, in the data dictionary, and then read the values ​​of these two variables through the data dictionary interface.

First, we first create a directory (Location), then create a data set (data), create two new data items ip, ipQueryUrl under "data", as shown in the figure below.



Modify the original code code.

Original code:

import requests

from autotest import \*

#set test case information

SetCase("TEST-1","Get location by ip","1")

ip="8.8.8.8"

ipQueryUrl="http://ip-api.com/json/{0}".format(ip)

LogInfo("ipQueryUrl:{0}".format(ipQueryUrl))

response=requests.get(ipQueryUrl)

AssertEqual(response.status\_code,200,"Query ip is successful.")

Modified code:

import requests

from autotest import \*

#set test case information

SetCase("TEST-1","Get location by ip","1")

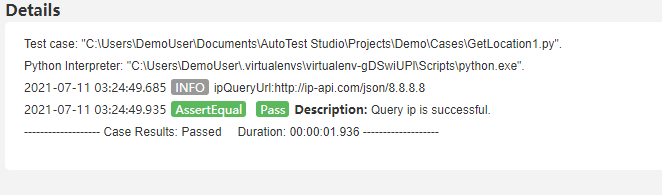
ipQueryUrl="{0}{1}".format(GetDataValue("data","url"),GetDataValue("data","ip"))

LogInfo("ipQueryUrl:{0}".format(ipQueryUrl))

response=requests.get(ipQueryUrl)

AssertEqual(response.status\_code,200,"Query ip is successful.")

The result of the operation is as follows. From the log, it is consistent with the original code.



Modify the code again, this time, we read the entire data set.

import requests

from autotest import \*

#set test case information

SetCase("TEST-1","Get location by ip","1")

data=GetDataset("data")

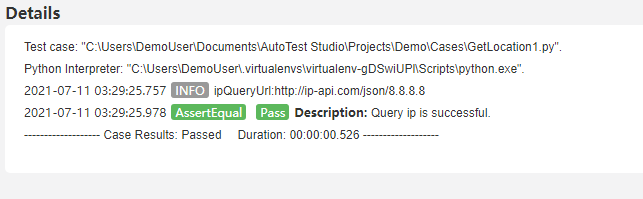
ipQueryUrl="{0}{1}".format(data["url"],data["ip"])

LogInfo("ipQueryUrl:{0}".format(ipQueryUrl))

response=requests.get(ipQueryUrl)

AssertEqual(response.status\_code,200,"Query ip is successful.")

Run result:



As can be seen from the demo above, the use of the data dictionary is very simple.