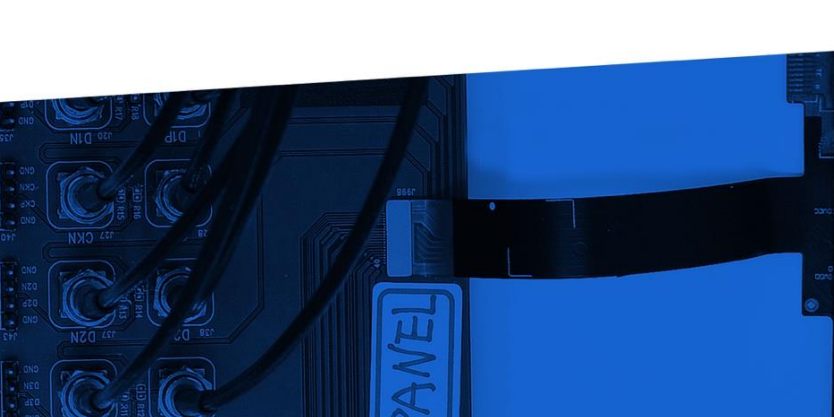
introspect technology

TECHNICAL INSIGHTS

Importing and Exporting Python Modules



Published in Canada on October 7, 2019

INTROSPECT.CA

Table of Contents

Introduction

It is sometimes useful to import external Python modules into Test Procedures developed within the Introspect ESP Software. Similarly, it might be useful to export test methods or entire Test Procedures from the Introspect ESP Software for use in external Python scripts. This Technical Insights brief describes ways to achieve both of these requirements.

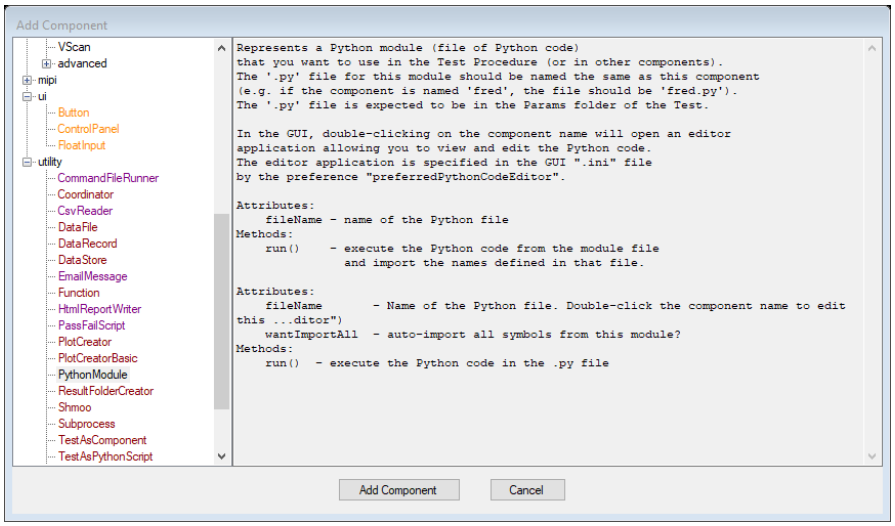
Importing Modules into a Test Procedure

USING THE PYTHONMODULE COMPONENT CLASS

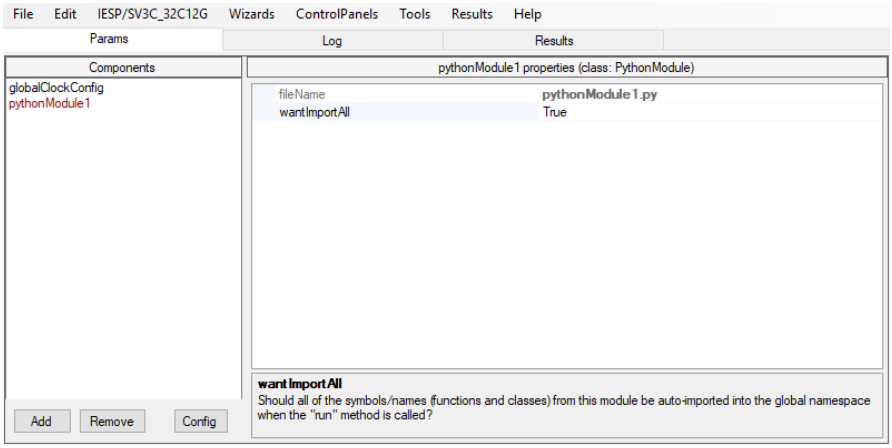
BASIC CONCEPT

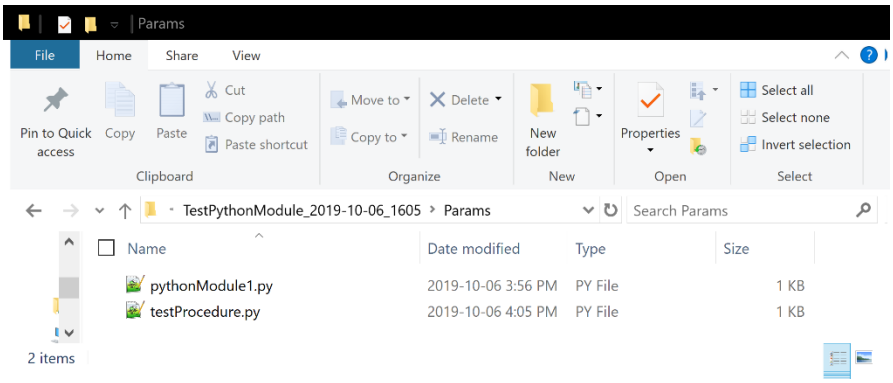
The Introspect ESP Software has a built-in component class called **PythonModule**, and this is the recommended way of importing external Python functions or modules into Test Procedures being executed from within the Introspect ESP Software. This section describes the basic concept of this component class and illustrates real-life examples of how it is used.

A **PythonModule** component can be instantiated by adding it to the Introspect ESP Software Test Procedure using the "Add Component" menu. Note that this component class is listed under the "utility" category of the menu as shown in the following image.



When the **PythonModule** component is instantiated, the Introspect ESP Software takes a couple of automated actions. First, it adds the module to the Components tab as is the case with all other instantiated components. Second, it creates a blank .py file inside the Params sub-folder of the Test Procedure folder in Windows. The name of this file matches the name of the component class that was instantiated. That is, referring to the two following images, the software creates a module called pythonModule1 and creates a corresponding file called pythonModule1.py.





By default, the pythonModule1.py file is blank, and the Introspect ESP software typically launches a code editor to allow the user to edit this file. This is when external code can be added. In the following example, we show two function declarations: one to create a custom print method and the second to compute a Fibonacci sequence. As can be seen, any Python code can be placed in this file, including function declarations, function calls, and library imports.

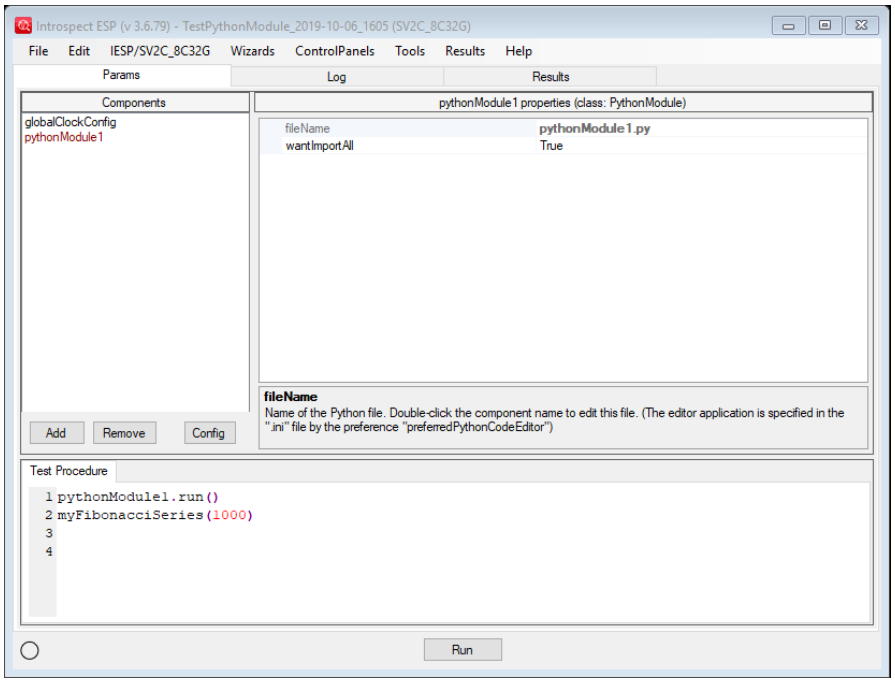
pythonModule1.py

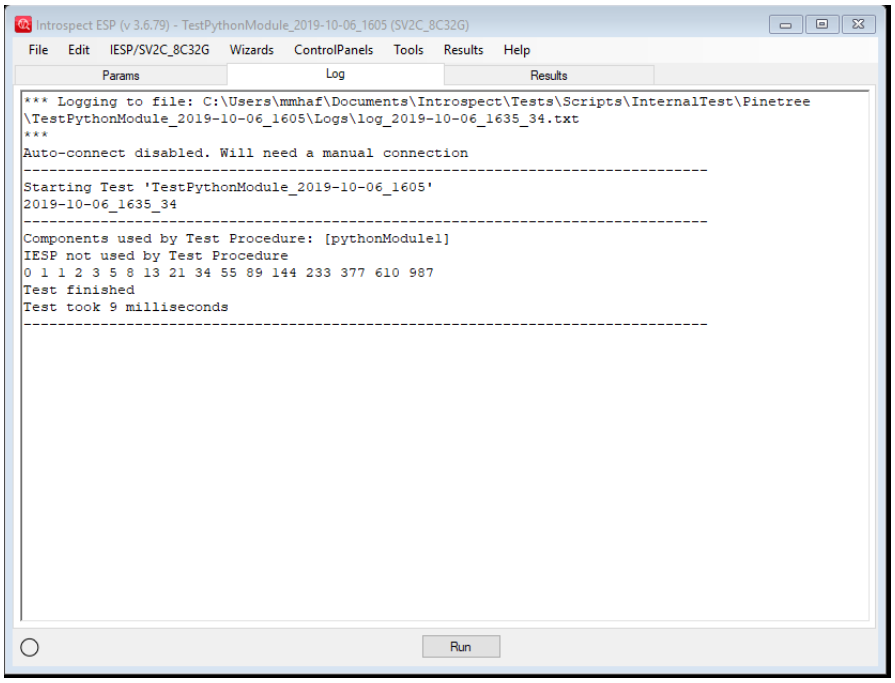
```
#!/usr/bin/env python

def myprint():
  print("hello")


def myFibonacciSeries(n):
  a, b = 0, 1
  while a < n:
    print(a, end=' ')
    a, b = b, a+b
  print()
```

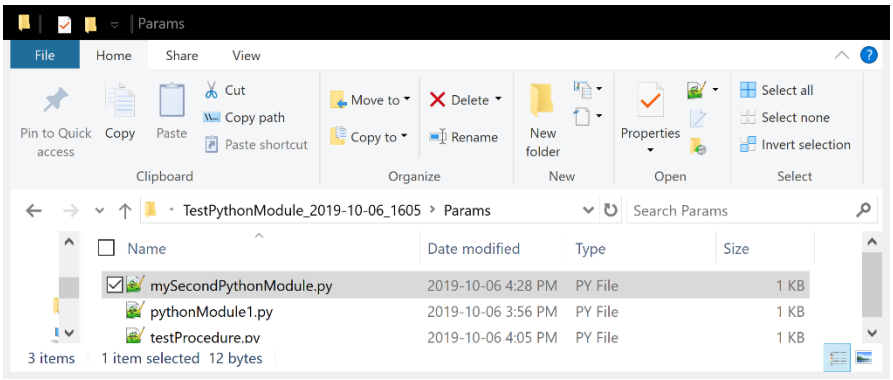
After saving the pythonModule1.py file, we can go back to the Introspect ESP Software and never have to worry about the source code anymore. Instead, we import the file's contents by adding the call pythonModule1.run() in the main Test Procedure pane. Once the file is imported this way, we can call any function that was declared inside it. For example, the following image shows how to call the Fibonacci Series calculator that was declared in the above example; and the image after it shows the execution log of the Test Procedure, confirming that the external Python file was indeed imported and that the Fibonacci Series function was executed.


Introspect ESP (v 3.6.79) - TestPythonModule_2019-10-06_1605 (SV2C_8C32G)
File Edit IESP/SV2C_8C32G Wizards ControlPanels Tools Results Help
Params Log Results
Components
globalClockConfig
pythonModule1
pythonModule1 properties (class: PythonModule)
fileName pythonModule1.py
wantImportAll True
fileName
Name of the Python file. Double-click the component name to edit this file. (The editor application is specified in the ".ini" file by the preference "preferredPythonCodeEditor")
Add Remove Config
Test Procedure
1 pythonModule1.run()
2 myFibonacciSeries(1000)
3
4
Run


Introspect ESP (v 3.6.79) - TestPythonModule_2019-10-06_1605 (SV2C_8C32G)
File Edit IESP/SV2C_8C32G Wizards ControlPanels Tools Results Help
Params Log Results
*** Logging to file: C:\Users\mmhaf\Documents\Introspect\Tests\Scripts\InternalTest\Pinetree
\TestPythonModule_2019-10-06_1605\Logs\log_2019-10-06_1635_34.txt
***
Auto-connect disabled. Will need a manual connection
Starting Test 'TestPythonModule_2019-10-06_1605'
2019-10-06_1635_34
Components used by Test Procedure: [pythonModule1]
IESP not used by Test Procedure
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
Test finished
Test took 9 milliseconds
Run

PLACING A PYTHON FILE IN THE PARAMS FOLDER

If you want to import an already existing Python file without having to paste its contents into a blank file, then you can simply place this file in the Params sub-folder of your target Introspect ESP Software folder. When you do so, the Introspect ESP Software automatically creates an instance of the **PythonModule** component class. In the following example, we place a file called mySecondPythonModule.py in the Params folder as shown in the following image.

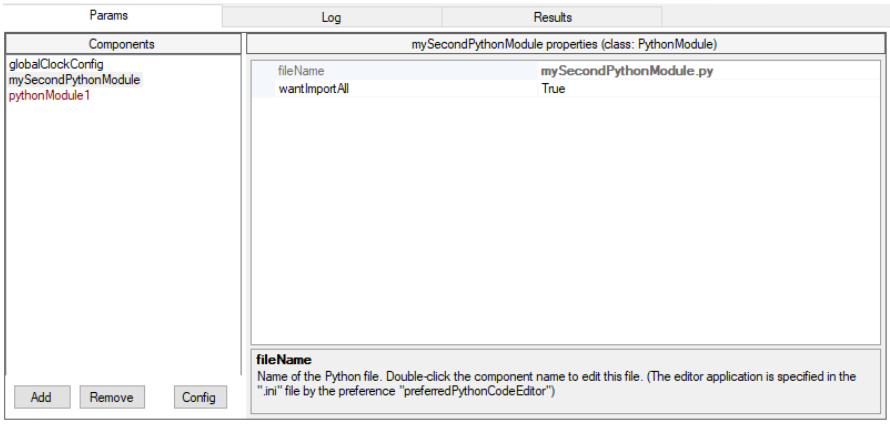


The file itself has one line, which is an import call for the built-in Python uuid library, included here only for exemplary reasons.

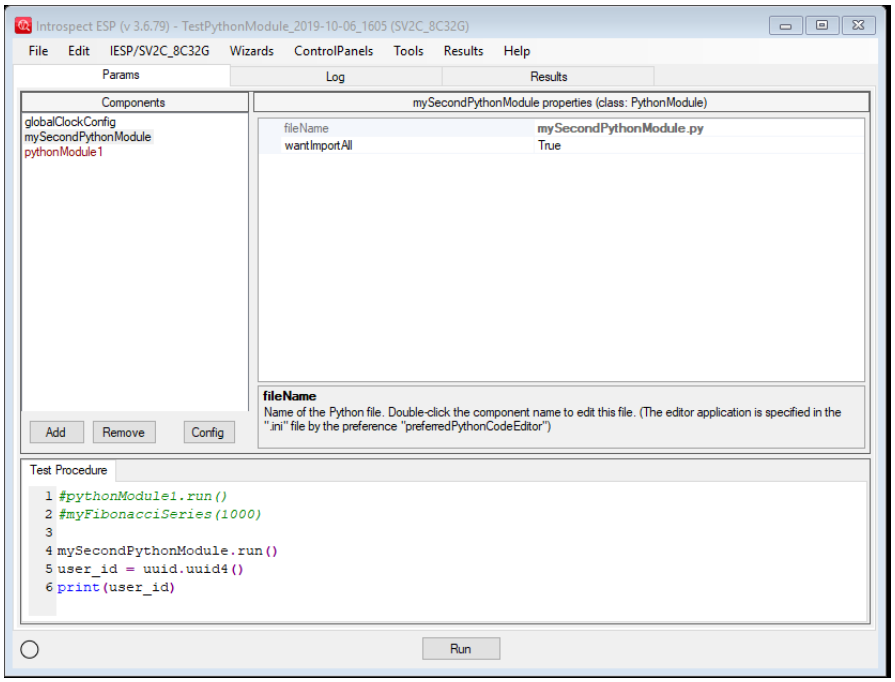
mySecondPythonModule.py

```
import uuid
```

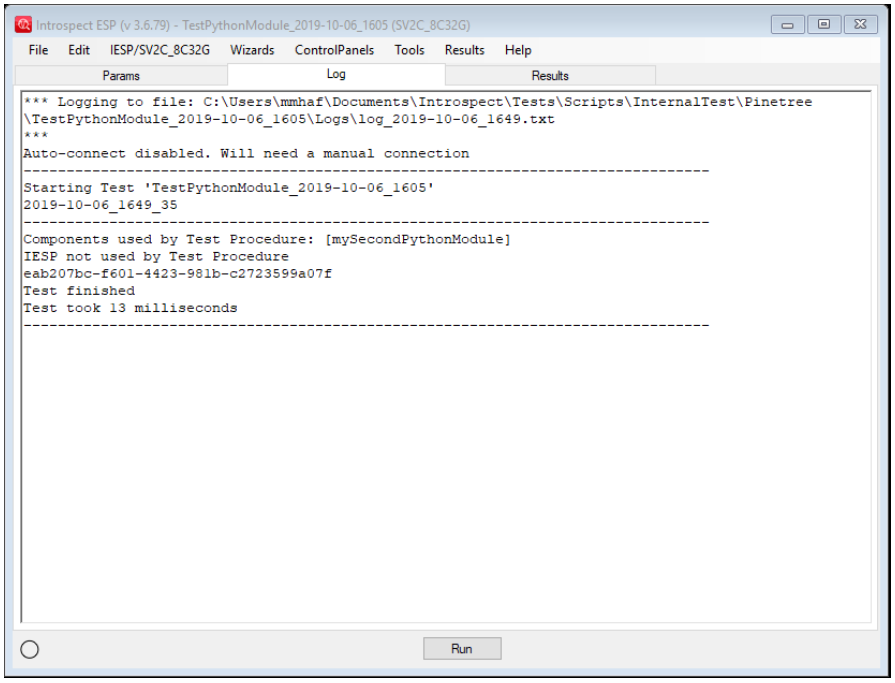
Opening the Test Procedure in the Introspect ESP Software, we see that the **PythonModule** class is automatically instantiated as in the following image.



We now proceed to using this module in the Test Procedure pane. Specifically, we import the module by executing the method mySecondPythonModule.run() as before. Then, we are able to use any built-in function within the uuid library that we have just imported. In the following example, we use it to create a unique user ID and then print this ID to the log window.



The result of executing the above Test Procedure is shown in the following image. As can be seen, the Python module was successfully imported without cluttering the Test Procedure window.



MANUALLY IMPORTING PYTHON FILES

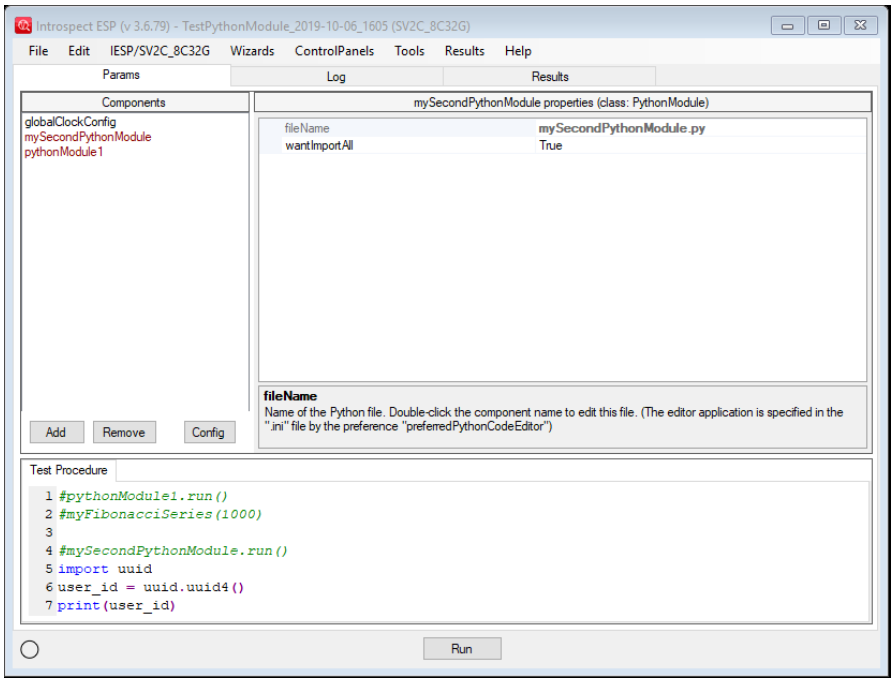
Apart from the **PythonModule** class, it is possible to manually import files into Test Procedures created within the Introspect ESP Software. This is done using common Python language constructs. For example, the following image illustrates importing the uuid library directly from within the Test Procedure.

Note that the Introspect ESP Software automatically searches the following path for external Python files:

<User Account>\Documents\Introspect\PythonCode

Any file stored in this directory can simply be imported using the call

```
import fileName
```

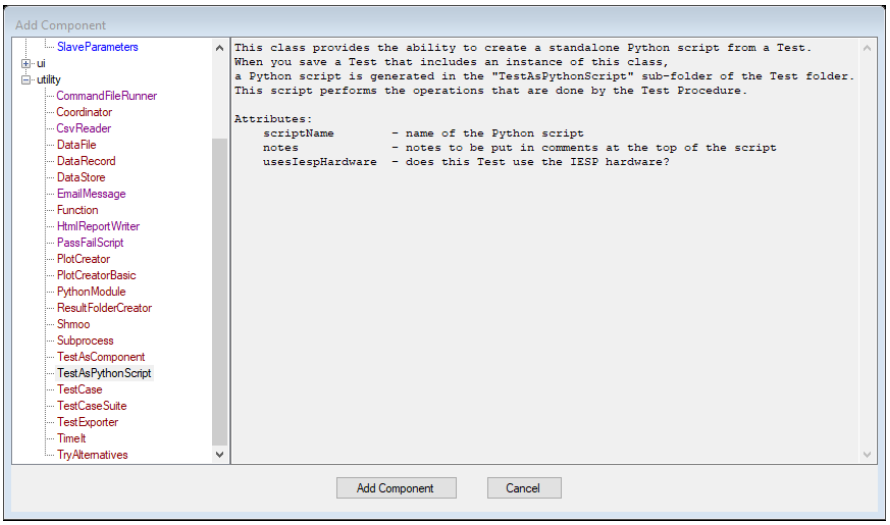


Exporting a Test Procedure

USING THE TESTASPYTHONSCRIPT COMPONENT CLASS

The Introspect ESP Software has a built-in component class called **TestAsPythonScript**, and this is an automated code-generation utility that allows you to export algorithms developed inside the Introspect ESP Software for use in external Python scripts. The advantage of this tool is that it automatically takes care of initializing form factors, creating component contexts, and connecting to the hardware. This section describes the basic concept of this component class.

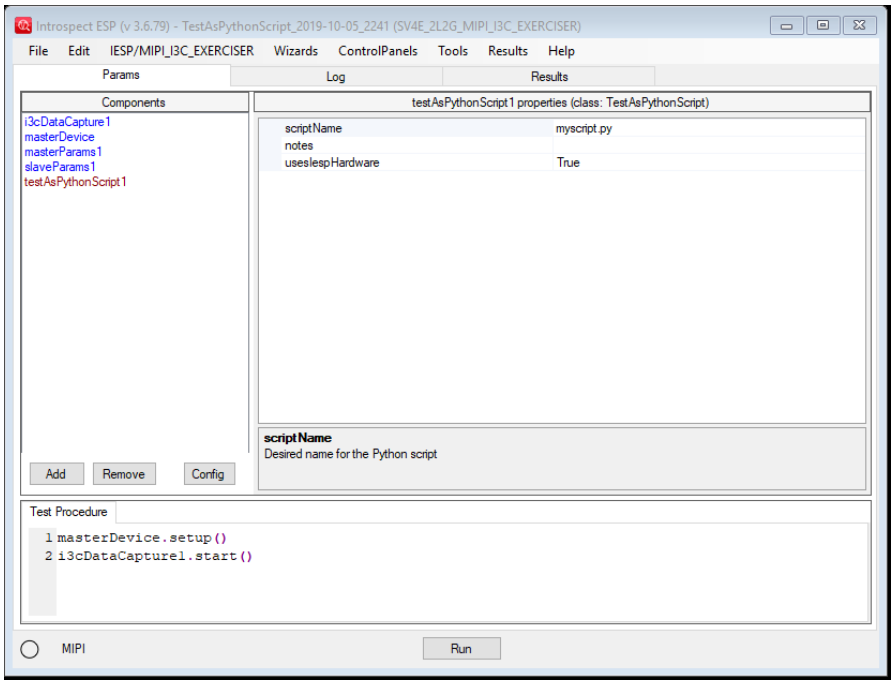
A **TestAsPythonScript** component can be instantiated by adding it to the Introspect ESP Software Test Procedure using the "Add Component" menu. Note that this component class is listed under the "utility" category of the menu as shown in the following image.



NOTE

The **TestAsPythonScript** has been introduced in version 3.6.79 of the Introspect ESP Software and is not available in earlier releases.

When the **TestAsPythonScript** component is instantiated, it is added to the Components pane just like any other component class. However, unlike other component classes, this class does not have any callable methods. As such, the Test Procedure pane is not modified as shown in the following screen shot. Instead, the Introspect ESP Software uses the existence of the testAsPythonScript1 component as a trigger to save an output Python file that can be used in external scripts. This output file is an exact representation of the Test Procedure that was edited from within the Software.



The automatically generated code for the above Test Procedure is shown in the next page. As can be seen, all aspects of external instantiation of Introspect components are taken care of automatically.

myScript.py

```
# Generated via SvtTestAsPythonScript from Test 'TestAsPythonScript_2019-10-
05_2241'
# 2019-10-05_2241

from dftm.svt import initFormFactor, createComponentContext, errorMsg
import dftm.fileUtil as fileUtil

formFactorName = 'SV4E_2L2G_MIPI_I3C_EXERCISER'
iesp = initFormFactor(formFactorName)

currentFolder = fileUtil.getCurrentFolder()
svtContextFolderName = 'myscriptFolder'
svtContextFolderPath = fileUtil.joinPaths(currentFolder, svtContextFolderName)
svtContext = createComponentContext(svtContextFolderPath)
svtNamesDict = svtContext.getNamesDict()
globalsDict = globals()
globalsDict.update(svtNamesDict)

connected = iesp.connectToHardware()
if not connected:
    errorMsg('Failed to connect to IESP hardware')

#------------------------------------------------------------------------------
# Components:
#------------------------------------------------------------------------------
i3cDataCapture1 = svtContext.createComponent('SvtMipiI3cDataCapture')

masterParams1 = svtContext.createComponent('SvtMipiI3cMasterParameters')

slaveParams1 = svtContext.createComponent('SvtMipiI3cSlaveParameters')

masterDevice = svtContext.createComponent('SvtMipiI3cDevice')
masterDevice.masterModeParams = masterParams1
masterDevice.slaveModeParams = slaveParams1
masterDevice.startupState = 'master'

#------------------------------------------------------------------------------
#------------------------------------------------------------------------------

def testProcedure():
    svtContext.initForRun() # re-init components for this run
    svtContext.createRunResultFolder() # create a dated sub-folder for results
    masterDevice.setup()
    i3cDataCapture1.start()
#------------------------------------------------------------------------------

if __name__ == '__main__':
    testProcedure()
#------------------------------------------------------------------------------
```

MANUALLY EXPORTING COMPONENT CLASSES

Introspect ESP component classes can be instantiated in external Python scripts by following the instructions in the application note:

UsingComponentsInExternalPythonScripts.pdf

This application note is included in the Doc folder of the Introspect ESP Installation.

Revision Number	History	Date
1.0	Document Release	October 7, 2019

The information in this document is subject to change without notice and should not be construed as a commitment by Introspect Technology. While reasonable precautions have been taken, Introspect Technology assumes no responsibility for any errors that may appear in this document.

Published in Canada on October 7, 2019

INTROSPECT.CA