



Importing and Exporting Python Modules





© Introspect Technology, 2019 Published in Canada on October 7, 2019

INTROSPECT.CA



Table of Contents

Introduction	3
Importing Modules into a Test Procedure	3
Using the PythonModule Component Class	3
Manually Importing Python Files	10
Exporting a Test Procedure	11
Using the TestAsPythonScript Component Class	11
Manually Exporting Component Classes	14



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.

Add Component			
→ VScan ⊕-advanced ⊕-advanced ⊕-mpi ←Uton ←CortrolPanel ←CortrolPanel ←CortrolPanel ←CortrolPanel ←CortrolPanel ←Cortrolator ←CortPanel ←DataFle ←DataFle ←DataFle ←DataStore ←TaitMessage ←Fundtion ←throtPeontWriter ←PassFalScript ←PlotCreator PhotCreator PhotCreator FythonModule ←Result FoldeCreator ←Sutprocess ←Sutprocess ←Sutprocess ←Sutprocess ←TestAsPythonScript	< >	<pre>Represents a Python module (file of Python code) that you want to use in the Test Procedure (or in other components). The '.py' file for this module should be named the same as this component (e.g. if the component is named 'fred', the file should be 'fred.py'). The '.py' file is expected to be in the Params folder of the Test. In the GUI, double-clicking on the component name will open an editor application allowing you to view and edit the Python code. The editor application is specified in the GUI ".in" file by the preference "preferredPythonCodeEditor". Attributes: fileHame - name of the Python file Methods: run() - execute the Python code from the module file and import the names defined in that file. Attributes: fileHame - Name of the Python file. Double-click the component name to edit thisditor") wantImportAll - auto-import all symbols from this module? Methods: run() - execute the Python code in the .py file</pre>	< >
		Add Component Cancel	



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.

File	Edit	IESP/SV3C_32C12G	Wizards	ControlPanels	Tools	Results	Help			
		Params		Log			Results			
Components pythonModule1 properties (class:						ass: PythonModul	e)			
	(ClockCo nModule			fileName wantImportAll			pythonM True	łodule 1.py		
Ac	dd	Remove	Sho	nt Import All uld all of the symbols n the "run" method i		unctions and	l classes) from this	s module be auto	imported into the global nan	nespace





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()</pre>
```

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.



🔯 In	trospect	ESP (v 3.6.79) - TestPytl	honMod	ule_2019-10-06_160	5 (SV2C_8						
File	e Edit	IESP/SV2C_8C32G	Wizards	ControlPanels	Tools	Results	Help				
	Params			Log			Results				
		Components				pythonMod	dule1 properties (clas	s: PythonMo	dule)		
pyth		Remove Config	N. "J	fileName wantImportAll eName ame of the Python file ni ^m file by the preferen	. Double-c	lick the con	pythonMo True	dule1.py		tion is specifie	d in the
0						Run					

🔯 Intr	ospect B	SP (v 3.6.79) - TestPyt	thonModule	≥_2019-10-06_160	5 (SV2C_8					
File	Edit	IESP/SV2C_8C32G	Wizards	ControlPanels	Tools	Results	Help			
		Params		Log			Results			
\Tes ***	tPyth	ng to file: C: onModule_2019- ect disabled. 	10-06_16	05\Logs\log	_2019-	10-06_1		ternalT	est\Pir	hetree
		Test 'TestPyth 6_1635_34		2019-10-06	-					
IESP 0 1 Test	not 123 fini	s used by Test P used by Test P 5 8 13 21 34 shed 9 millisecond	rocedure 55 89 14	•				 		
0						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.



Params	Log	Results					
Components	mySecondPythonModule properties (class: PythonModule)						
globalClockConfig mySecondPythonModule pythonModule1	fileName wantImportAll	mySecondPytho True	onModule.py				
	fil-Mara						
Add Remove Config	fileName Name of the Python file. Double ".ini" file by the preference "pre	e-click the component name to edit this file ferredPythonCodeEditor'')	e. (The editor application is specified in the				

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.

File Edit IESP/SV2C_8C326 Wizards ControlPanels Tools Results Components mySecondPythonModule properties (class: PythonModule) mySecondPythonModule.py mySecondPythonModule.py globalClockCorfig mySecondPythonModule mySecondPythonModule.py mySecondPythonModule.py pythonModule1 fileName mySecondPythonModule.py mySecondPythonModule.py fileName mySecondPythonModule.py mySecondPythonModule.py mySecondPythonModule.py fileName mySecondPythonModule.py mySecondPythonModule.py mySecondPythonModule.py fileName mySecondPythonModule.py mySecondPythonModule.py mySecondPythonModule.py fileName mySecondPythonModule.py mySecondPythonModule.py mySecondPythonModule.py Add Remove Config FileName Name of the Python file. Double-click the component name to edit this file. (The editor application is specified in the ''m'' file by the preference ''reference''preference'''reference''reference'''		lodule_2019-10-06_1605 (SV2							
Components mySecondPythonModule properties (class: PythonModule) globalClockCorfig mySecondPythonModule properties (class: PythonModule.py mySecondPythonModule1 mySecondPythonModule.py wartImportAll True fileName mySecondPythonModule.py Market PythonModule1 True fileName True Market PythonModule1 True fileName True Market PythonModule2 True fileName True Market PythonModule2 True fileName True Market PythonModule2 True fileName True Name of the Python file. Double-click the component name to edit this file. (The editor application is specified in the "in" file by the preference "preferedPythonCodeEditor") Test Procedure 1 1 fpythonModule1.run () 2 fmyFibonacciSeries (1000) 3 mySecondPythonModule.run () 5 user_id = uuid.uuid4() 6 print (user_id)	File Edit IESP/SV2C_8C32G Wiza	ards ControlPanels Too	s Results Help						
globalCockConfig mySecondPythonModule mySecondPythonModule.py pythonModule 1 True wartimportAll True fileName Name of the Python file. Double-click the component name to edit this file. (The editor application is specified in the "in" file by the preference "preferedPythonCodeEditor") Test Procedure 1 #pythonModule1.run () 2 #myFibonacciSeries (1000) 3 4 mySecondPythonModule.run () 5 user_id = uuid.uuid4 () 6 print (user_id)	Params	Log	Results						
impSecondPythonModule impSecondPythonModule.py wartmportAl True impSecondPythonModule.py impSecondPythonModule.py wartmportAl True impSecondPythonModule.py impSecondPythonModule.py impSecondPythonModule.py impSecondPythonModule.py impSecondPythonModule.py impSecondPythonModule.py impSecondPythonModule.py impSecondPythonModule.py impSecondPythonModule.py impSecondPythonModule.py impSecondPythonModule.py impSecondPythonModule.py impSecondPythonModule.run() impSecondPythonModule.run() suser_id = uuid.uuid4() impSecondPythonModule.run() fileName impSecondPythonModule.run() suser_id = uuid.uuid4() impSecondPythonModule.run()									
<pre>1 fpythonModule1.run() 2 fmyFibonacciSeries(1000) 3 4 mySecondPythonModule.run() 5 user_id = uuid.uuid4() 6 print(user_id)</pre>	mySecondPythonModule pythonModule1	wantImportAll fileName Name of the Pvthon file. Doub	True						
Run	<pre>1 #pythonModule1.run() 2 #myFibonacciSeries(1000 3 4 mySecondPythonModule.ru 5 user_id = uuid.uuid4()</pre>	Add Remove Config "in" file by the preference "preferedPythonCodeEditor") Test Procedure 1 #pythonModule1.run() 2 #myFibonacciSeries(1000) 3 4 mySecondPythonModule.run() 5 user_id = uuid.uuid4()							



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.

	Edit IESP/SV2C 8C	32G Wizards	ControlPanels	Tools	Results	Heln			
	Params	SEG TREATES	Log	10013	reserves	Results			
\Test ***	Logging to file tPythonModule_2 -connect disable	019-10-06_10	505\Logs\log	_2019-	10-06_1		ipts\Inter	nalTest\	Pinetree
	ting Test 'Test) -10-06_1649_35	PythonModule	2019-10-06	-					
eab2(Test	not used by Te: 07bc-f601-4423- finished took 13 millise	981b-c272359							



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.

Add Component	
SlaveParameters Gui Gui Gui CommandFileRunner CovmandFileRunner CavReader DataFile DataFlee DataRecord DataStore Function HmiRepotWiter PassaFalScript PictCreator PictCreator PictCreator Subprocess TestAcComponent TestCaseSuite TestCaseSuite TestCaseSuite TestExpoter Time t CommandFile Comma	This class provides the ability to create a standalone Python script from a Test. When you save a Test that includes an instance of this class, a Python script's sub-folder of the Test folder. This script performs the operations that are done by the Test Procedure. Attributes: scriptName - name of the Python script notes - notes to be put in comments at the top of the script usesIespHardware - does this Test use the IESP hardware? V
	Add Component Cancel

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.

🔯 Introspect ESP (v 3.6.79) - TestAsPytho	nScript_2019-10-05_2241 (SV4E_	2L2G_MIPI_I3C_EXERCISER)	
File Edit IESP/MIPI_I3C_EXERCISER	Wizards ControlPanels	Tools Results Help	
Params	Log	Results	
Components	tes	tAsPythonScript1 properties (class: TestAs	PythonScript)
i3cDataCapture1 masterDevice	scriptName	myscript.py	
masterParams1 slaveParams1	notes useslespHardware	True	
testAsPytionScript1			
Add Remove Config	scriptName Desired name for the Python scri	pt	
Test Procedure			
<pre>1 masterDevice.setup() 2 i3cDataCapturel.start()</pre>			
MIPI	[Run	

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.



© Introspect Technology, 2019 Published in Canada on October 7, 2019

INTROSPECT.CA