

Export Python Scripts for

TEST AUTOMATION

Objectives

At the end of this section, you will be able to

- Create a Python script for an existing set of test vectors
- Execute the Python script from the command line
- Configure the script to generate HTML and/or JUnit reports

testIDEA

SCRIPTING FOR TEST AUTOMATION

| | | |
|---|---|-------|
| 1 | Back to the EC-LIB® Project | 3 |
| 2 | Three steps of automation | 4 |
| 3 | Step 1 - Generate a Python script | 5-6 |
| 4 | Step 2 - Call Python script for test automation | 7-9 |
| 5 | Step 3 - Configure test report creation | 10-19 |
| 6 | Summary | 20-21 |

testIDEA

1 BACK TO THE EC-LIB[©] PROJECT



Continuous Integration (CI) tools, such as Jenkins and Bamboo, make it possible to check out code from a repository, build it and automatically test it, saving the results for later review.

testIDEA, with its Script Export functionality, makes it possible to export test cases in Python, which is easy to integrate into various automation tools or processes.

To demonstrate the export functionality of testIDEA we will reuse the EC-LIB[©] testing example as seen in Unit 05.

The screenshot shows the testIDEA interface with a table of test results. The table has columns for 'func', 'params', 'retVal', 'testTimeout', 'coreId', 'isExpectException', and a final column for the result. The 'func' column lists 'ECLIB_Sqr_16' for 16 test cases. The 'params' column shows various input values. The 'retVal' column shows the return values. The 'testTimeout' and 'coreId' columns are empty. The 'isExpectException' column shows '0' for all tests. The final column shows the result, which is 'myResult===' for all tests.

| | func | params | retVal | testTimeout | coreId | isExpectException | |
|----|--------------|-------------------------------|--------|-------------|--------|-------------------|------------|
| 0 | ECLIB_Sqr_16 | myResult myResult_sf 2 0 | | | | | myResult== |
| 1 | ECLIB_Sqr_16 | myResult myResult_sf 0 0 | | | | | myResult== |
| 2 | ECLIB_Sqr_16 | myResult myResult_sf 0 0 | | | | | myResult== |
| 3 | ECLIB_Sqr_16 | myResult myResult_sf 0 0 | | | | | myResult== |
| 4 | ECLIB_Sqr_16 | myResult myResult_sf 13 3 | | | | | myResult== |
| 5 | ECLIB_Sqr_16 | myResult myResult_sf 0 0 | | | | | myResult== |
| 6 | ECLIB_Sqr_16 | myResult myResult_sf 5 0 | | | | | myResult== |
| 7 | ECLIB_Sqr_16 | myResult myResult_sf 32766 0 | | | | | myResult== |
| 8 | ECLIB_Sqr_16 | myResult myResult_sf 32767 0 | | | | | myResult== |
| 9 | ECLIB_Sqr_16 | myResult myResult_sf -32766 0 | | | | | myResult== |
| 10 | ECLIB_Sqr_16 | myResult myResult_sf -32767 0 | | | | | myResult== |
| 11 | ECLIB_Sqr_16 | myResult myResult_sf -32768 0 | | | | | myResult== |
| 12 | ECLIB_Sqr_16 | myResult myResult_sf 0 41 | | | | | myResult== |
| 13 | ECLIB_Sqr_16 | myResult myResult_sf 5 41 | | | | | myResult== |
| 14 | ECLIB_Sqr_16 | myResult myResult_sf 32766 41 | | | | | myResult== |
| 15 | ECLIB_Sqr_16 | myResult myResult_sf 32767 41 | | | | | myResult== |



The import functionality of testIDEA shown in this unit requires the testIDEA Pro license.



2 THREE STEPS OF AUTOMATION

In this unit we will go through three steps to create a scripted unit test.

Firstly a Python script will be generated from the testIDEA GUI. This will be created using an existing set of test cases.

Next, the script will be executed from the command line.

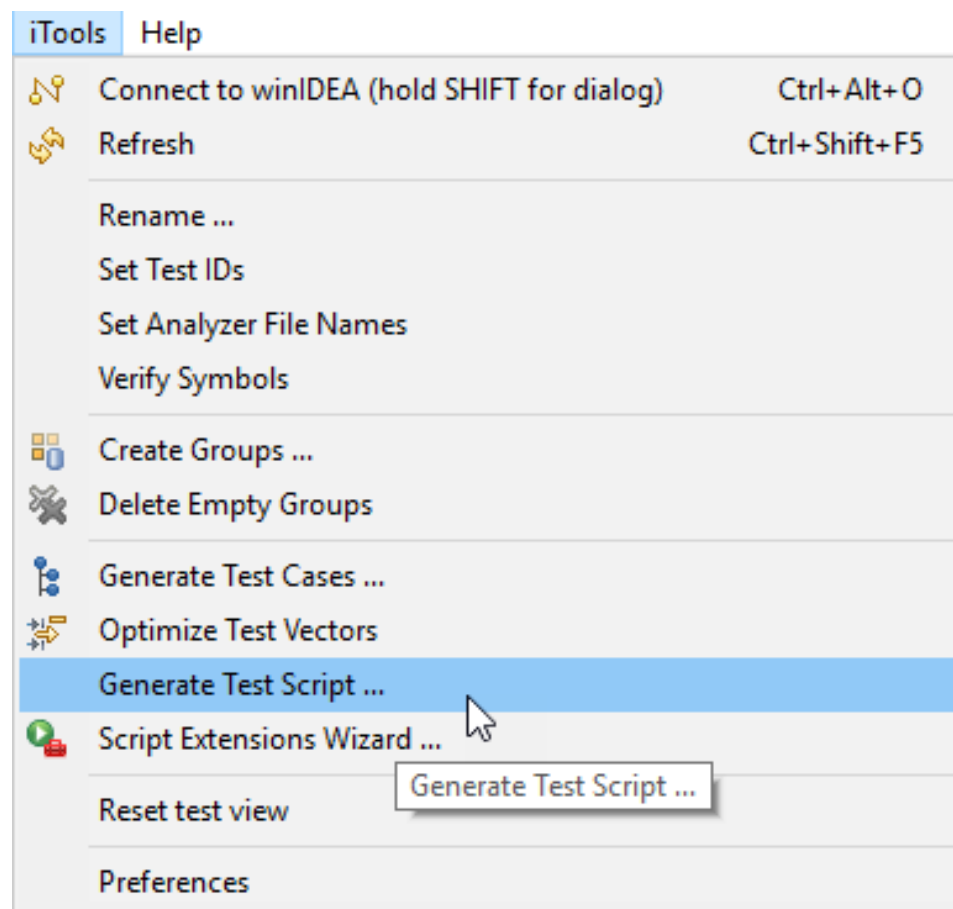
Finally, the script will be updated to automatically generate a test report in either HTML format or a format that can be used by CI tools such as Jenkins.

1. Generate a Python test script
Go to [“Generate a Python test script”](#)
2. Call a Python test script
Go to [“Call a Python test script for test automation”](#)
3. Automate test report creation
Go to [“Automate test report creation”](#)

3 STEP 1 - GENERATE A PYTHON TEST SCRIPT

Start by creating a set of test cases or, as here, opening an existing iYAML file. Here we are using the imported tests vectors generated for the EC-LIB[®] library, the version without the generation of code coverage, which will save us some time during test execution.

To generate a Python test script, start by opening the *iTools* menu and select the *Generate Test Script...* option.



3 STEP 1 - GENERATE A PYTHON TEST SCRIPT



Start by naming your Python script file and, if required, modifying the path where it is to be saved.

Initially we will not use the reporting feature, so we can uncheck the box *Save test report in testIDEA format* in the section *Report*.

The screenshot shows the 'Generate Test Script' dialog box with the following configuration:

- Files**
 - Generated script file: C:\Users\babkinje\Desktop\TestIdeaWorkspace\Exercise03\bsc0002-03.py
 - Use custom script template: ☐ (Browse button)
 - Use custom test spec. file: C:\Users\babkinje\Desktop\TestIdeaWorkspace\Exercise03\bsc0002-03.iyar (Browse button)
 - Use custom winIDEA worksp.: ☐ (Browse button)
- Execution configuration**
 - Imports:
 - Use custom init sequence: ☐ (Edit script init sequence button)
 - Use filter ID: (dropdown arrow)
 - Use progress monitor: ☒
 - Use default monitor: ☒
 - Monitor class: Monitor
 - Path to isystem.connect dll:
- Report**
 - Save test report in testIDEA format: ☐ (highlighted with a red box)
 - Save test report in JUnit format (for Jenkins): ☐
 - Open report in browser: ☐
 - Use custom report configuration: ☐ (Edit script report configuration button)

4 STEP 2 - CALL PYTHON SCRIPT FOR TEST AUTOMATION



Next, open a command line shell and go to the directory where we saved our script.

```
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\babkinje>cd Desktop\TestIdeaWorkspace\Exercise03

C:\Users\babkinje\Desktop\TestIdeaWorkspace\Exercise03>dir /s *.py
Datenträger in Laufwerk C: ist OS
Volumeserienummer: E823-7EA2

Verzeichnis von C:\Users\babkinje\Desktop\TestIdeaWorkspace\Exercise03

25.07.2017  16:13                17.436 bsc0002-03.py
               1 Datei(en),               17.436 Bytes

Anzahl der angezeigten Dateien:
               1 Datei(en),               17.436 Bytes
               0 Verzeichnis(se), 378.273.763.328 Bytes frei

C:\Users\babkinje\Desktop\TestIdeaWorkspace\Exercise03>
```

4 STEP 2 - CALL PYTHON SCRIPT FOR TEST AUTOMATION



winIDEA installs its own version of Python with the necessary isystem.connect libraries to support scripted use of iSYSTEM's BlueBox™ On-Chip Analyzers. For beginners we recommend to use this version of Python.

In this example, Python is called from the installation path of winIDEA as follows:

```
C:\iSYSTEM\winIDEA9\  
Python\python.exe
```

The names of the Python script, *bsc0002-03.py*, is passed as a parameter.

```
Microsoft Windows [Version 10.0.15063]  
(c) 2017 Microsoft Corporation. Alle Rechte vorbehalten.  
  
C:\Users\babkinje>cd Desktop\TestIdeaWorkspace\Exercise03  
  
C:\Users\babkinje\Desktop\TestIdeaWorkspace\Exercise03>dir /s *.py  
Datenträger in Laufwerk C: ist OS  
Volumeseriennummer: E823-7EA2  
  
Verzeichnis von C:\Users\babkinje\Desktop\TestIdeaWorkspace\Exercise03  
  
25.07.2017  16:24                17.436 bsc0002-03.py  
             1 Datei(en),               17.436 Bytes  
  
Anzahl der angezeigten Dateien:  
1 Datei(en),               17.436 Bytes  
0 Verzeichnis(se), 378.276.311.040 Bytes frei  
  
C:\Users\babkinje\Desktop\TestIdeaWorkspace\Exercise03>C:\iSYSTEM\winIDEA9\Python\python.exe bsc0002-03.py_
```


4 STEP 2 - CALL PYTHON SCRIPT FOR TEST AUTOMATION

Upon execution, the same process is performed as you would expect from executing the tests in the testIDEA GUI.

The test results are shown at the end of the command line output, as shown opposite.



This Python script can be executed directly from the CI tool Jenkins in the same manner as shown here. Later slides will show how to generate reports suitable for display in that platform.

```

Description:
  Executed test: 117 / 119
Executing test: ECLIB_Sqr_16.0118 /
  Description:
    Executed test: 118 / 119
Executing test: ECLIB_Sqr_16.0119 /
  Description:
    Executed test: 119 / 119
...
reportStatistic:
  noOfTests: 119
  allErrors: 0
  exceptionErrors: 0
  failures: 0
  expressionErrors: 0
  coverageErrors: 0
  codeProfilerErrors: 0
  dataProfilerErrors: 0
  scriptErrors: 0
  stubErrors: 0
  testPointErrors: 0
  stackUsageErrors: 0
...
OK, no errors in test execution detected!

C:\Users\Stuart\sw-dev\learnIDEA\BSC0002\SAM3X\E03-Square>

```

5 STEP 3 - CONFIGURE TEST REPORT CREATION



In order to create test reports, the report format must be configured

Start by selecting *Test* menu and clicking *Configure test report* to open the dialog seen here.

1 Define Output format

1 **Output format:** ☒ XML ☐ YAML ☐ CSV ☐ XLS ☐ XLSX ⁱ

Output format configuration

XSLT: i **Browse**

CSS: i **Browse**

Logo image URL: i **Browse**

Report title: i

☒ Create HTML ⁱ ☐ Embed XSLT/CSS ⁱ HTML content: i

Output file: i **Browse**

☐ Use absolute links to export files ⁱ ☐ Include test specifications ⁱ

☒ Open default browser after save

Test Environment

| Attribute | Value |
|-----------|-------|
| | |

5 STEP 3 - CONFIGURE TEST REPORT CREATION



In order to create test reports, the report format must be configured

Start by selecting *Test* menu and clicking *Configure test report* to open the dialog seen here.

1 Define Output format

2 Define a XSLT and a CSS template, otherwise the report can't be displayed properly

1

2

Configure test reports

Output format: ☒ XML ☐ YAML ☐ CSV ☐ XLS ☐ XLSX ⁱ

Output format configuration

XSLT: <built-in> isystemTestReport.xslt ⁱ Browse

CSS: <built-in> blue.css ⁱ Browse

Logo image URL: ⁱ Browse

Report title: ⁱ

☒ Create HTML ⁱ ☐ Embed XSLT/CSS ⁱ HTML content: All results ⁱ

Output file: C:\Users\babkinje\Desktop\TestIdeaWorkspace\Exercise03\Report.xml ⁱ Browse

☐ Use absolute links to export files ⁱ ☐ Include test specifications ⁱ

☒ Open default browser after save

Test Environment

| Attribute | Value |
|-----------|-------|
| | + |

5 STEP 3 - CONFIGURE TEST REPORT CREATION



In order to create test reports, the report format must be configured

Start by selecting *Test* menu and clicking *Configure test report* to open the dialog seen here.

- 1 Define Output format
- 2 Define a XSLT and a CSS template, otherwise the report can't be displayed properly
- 3 Logo can be pasted in the report as well as a report title

The screenshot shows the 'Configure test reports' dialog box. It has a title bar with a close button. The main area contains several sections:

- Output format:** A group box with radio buttons for XML (selected), YAML, CSV, XLS, and XLSX. An information icon is next to XLSX.
- Output format configuration:** A section with two rows: 'XSLT:' and 'CSS:'. Each has a text field with a built-in value ('<built-in> isystemTestReport.xslt' and '<built-in> blue.css' respectively) and a 'Browse' button.
- Logo image URL:** A text field with a 'Browse' button.
- Report title:** A text field with an information icon.
- HTML options:** A checked checkbox for 'Create HTML', an unchecked checkbox for 'Embed XSLT/CSS', and a dropdown for 'HTML content' set to 'All results'.
- Output file:** A text field showing a file path and a 'Browse' button.
- Checkboxes:** 'Use absolute links to export files' (unchecked), 'Open default browser after save' (checked), and 'Include test specifications' (unchecked).
- Test Environment:** A table with two columns: 'Attribute' and 'Value'. It has a green plus icon in the 'Value' column for adding new entries.

Numbered annotations on the left side of the dialog:

- 1** points to the 'Output format' group box.
- 2** points to the 'XSLT:' and 'CSS:' rows.
- 3** points to the 'Logo image URL:' and 'Report title:' rows.

5 STEP 3 - CONFIGURE TEST REPORT CREATION



In order to create test reports, the report format must be configured

Start by selecting *Test* menu and clicking *Configure test report* to open the dialog seen here.

- 1 Define Output format
- 2 Define a XSLT and a CSS template, otherwise the report can't be displayed properly
- 3 Logo can be pasted in the report as well as a report title
- 4 "Create HTML"

The screenshot shows the 'Configure test reports' dialog box with the following configuration:

- Output format:** XML (selected), YAML, CSV, XLS, XLSX
- Output format configuration:**
 - XSLT:** <built-in> isystemTestReport.xslt
 - CSS:** <built-in> blue.css
 - Logo image URL:** (empty)
 - Report title:** (empty)
- 4** ☒ Create HTML, ☐ Embed XSLT/CSS, HTML content: All results
- Output file:** C:\Users\babkinje\Desktop\TestIdeaWorkspace\Exercise03\Report.xml
- ☐ Use absolute links to export files, ☐ Include test specifications
- ☒ Open default browser after save
- Test Environment**

| Attribute | Value |
|-----------|-------|
| | |

5 STEP 3 - CONFIGURE TEST REPORT CREATION



In order to create test reports, the report format must be configured

Start by selecting *Test* menu and clicking *Configure test report* to open the dialog seen here.

- 1 Define Output format
- 2 Define a XSLT and a CSS template, otherwise the report can't be displayed properly
- 3 Logo can be pasted in the report as well as a report title
- 4 "Create HTML"
- 5 Output file: Where the report should be saved

The screenshot shows the 'Configure test reports' dialog box with the following configuration:

- Output format:** XML (selected), YAML, CSV, XLS, XLSX
- Output format configuration:**
 - XSLT: <built-in> isystemTestReport.xslt
 - CSS: <built-in> blue.css
 - Logo image URL: (empty)
 - Report title: (empty)
- Output:** ☒ Create HTML, ☐ Embed XSLT/CSS, HTML content: All results
- Output file:** C:\Users\babkinje\Desktop\TestIdeaWorkspace\Exercise03\Report.xml
- ☐ Use absolute links to export files, ☐ Include test specifications
- ☒ Open default browser after save
- Test Environment:** (Table with columns Attribute and Value)

Numbered steps in the image:

1. Output format selection
2. XSLT and CSS selection
3. Logo image URL and Report title input fields
4. 'Create HTML' checkbox and HTML content dropdown
5. Output file path and 'Browse' button

5 STEP 3 - CONFIGURE TEST REPORT CREATION



In order to create test reports, the report format must be configured

Start by selecting *Test* menu and clicking *Configure test report* to open the dialog seen here.

6 Set “*Open default browser after save*” check-box. This ensures that the report file will be opened in the browser for viewing upon test completion.

The screenshot shows the 'Configure test reports' dialog box with the following configuration:

- 1. Output format: ☒ XML ☐ YAML ☐ CSV ☐ XLS ☐ XLSX
- 2. Output format configuration:
 - XSLT: <built-in> isystemTestReport.xslt
 - CSS: <built-in> blue.css
- 3. Logo image URL: (empty field)
- 4. Report title: (empty field)
- 5. ☒ Create HTML ☐ Embed XSLT/CSS HTML content: All results
- 6. ☒ Open default browser after save
- Output file: C:\Users\babkinje\Desktop\TestIdeaWorkspace\Exercise03\Report.xml
- ☐ Use absolute links to export files ☐ Include test specifications
- Test Environment table:

| Attribute | Value |
|-----------|-------|
| | |

5 STEP 3 - CONFIGURE TEST REPORT CREATION



Now we can create a new Python script as before (*iTools* -> *Generate Test Script...*) and execute from the command line in the same manner as before.



If you plan to use the Python script together with a Continuous Integration tool such as Jenkins, it is recommended to leave the “Open report in Browser” option unchecked.

Generate Test Script

Files

Generated script file:

☐ Use custom script template:

☐ Use custom test spec. file:

☐ Use custom winIDEA worksp.:

Execution configuration

Imports:

☐ Use custom init sequence

☐ Use filter ID:

☒ Use progress monitor

☒ Use default monitor

Monitor class:

Path to isystem.connect dll:

Report

☒ Save test report in testIDEA format

☐ Save test report in JUnit format (for Jenkins)

☐ Export coverage for Jenkins. Analyzer (trd) file:

☒ Open report in browser

☐ Use If checked, test report is opened in the system default browser, after testing is finished. This setting overrides the one in report configuration dialog.

5 STEP 3 - CONFIGURE TEST REPORT CREATION



Once the tests are finished the test report will be automatically generated and displayed in the default browser.

| | |
|----------------------|---|
| Stub failures | 0 |
| Test point failures | 0 |
| Stack usage failures | 0 |

| Test Cases With Failures and Errors | | |
|-------------------------------------|----------|---------------|
| Test ID | Function | Failure/Error |
| | | |

| Test ID | | | | Function | | | | | | Result | |
|--------------------|---------------|---|-----------|-------------------------------------|---|--|--|--|--|---------------------------------------|--|
| ECLIB_Sqr_16.0003 | | | | ECLIB_Sqr_16 | | | | | | Pass | |
| Tags | | | | Base tests | | | | | | | |
| | | | | /1 | | | | | | | |
| Assert expressions | | | | | | | | | | | |
| Expression | | | | Sub-expressions | | | | | | | |
| myResult == 0 | | | | myResult = 0x0000 (0) | | | | | | | |
| myResult_sf == 0 | | | | myResult_sf = \x00 (0x00) (0) | | | | | | | |
| Coverage | | | | | | | | | | | |
| Document | | 2017-07-1713_23_33.trd | | | | | | | | | |
| Export file | | vyh5lx18cej13_23_33.txt | | | | | | | | | |
| Function | Obj. code all | Src. lines all | Cond. all | CC (Outcomes) | Obj. code executed | Src. lines executed | Conditions any | Cond. true only | Cond. false only | Conditions both | |
| | | | | | measured (exp., abs.) | measured (exp., abs.) | measured (exp., abs.) | measured (exp., abs.) | measured (exp., abs.) | measured (exp., abs.) | |
| ECLIB_Sqr_16 | 142 | 10 | 5 | 20.0% (2/10) <div><div></div></div> | 32.4% (0.0%, 46) <div><div></div></div> | 50.0% (0.0%, 5) <div><div></div></div> | 40.0% (0.0%, 2) <div><div></div></div> | 20.0% (0.0%, 1) <div><div></div></div> | 20.0% (0.0%, 1) <div><div></div></div> | 0.0% (0.0%, 0) <div><div></div></div> | |

| Test ID | Function | Result |
|-------------------|--------------|--------|
| ECLIB_Sqr_16.0002 | ECLIB_Sqr_16 | Pass |
| Tags | Base tests | |
| | /1 | |

5 SUPPORTING CI TOOLS SUCH AS JENKINS



Test reports can be displayed automatically in tools such as Jenkins. Together with the *JUnit* plugin, the number of passed and failed tests can be shown directly on the project's landing page.

To generate such files, simply select the *Save test report in JUnit format* in the *Generate Test Script* dialog.

There is also some support for the *Cobertura* plugin that displays the progress in code coverage achieved. This is enabled by the *Export coverage for Jenkins* option.

To find out more, search for “*Jenkins*” in the testIDEA help.

The image shows two overlapping windows. The background window is the 'Generate Test Script' dialog, and the foreground window is the Jenkins project page for 'EPB Demo - nano-HIL New'.

Generate Test Script Dialog:

- Files:**
 - Generated script file: C:\Users\Stuart\sw-dev\learnIDEA\BSC0002\SAM3X\E03-Square\bs [Browse]
 - ☐ Use custom script template: [Browse]
 - ☐ Use custom test spec. file: C:\Users\Stuart\sw-dev\learnIDEA\BSC0002\SAM3X\E03-Square\bs [Browse]
 - ☐ Use custom winIDEA worksp.: C:\Users\Stuart\sw-dev\learnIDEA\BSC0002\SAM3X\E03-Square\bs [Browse]
- Execution configuration:**
 - Imports: []
 - ☐ Use custom init sequence [Edit script init sequence]
 - ☐ Use filter ID: []
 - ☒ Use progress monitor
 - ☒ Use default monitor
 - Monitor class: Monitor
 - Path to isystem.connect dll: []
- Report:**
 - ☐ Save test report in testIDEA format
 - ☒ Save test report in JUnit format (for Jenkins)
 - ☒ Export coverage for Jenkins. Analyzer (trd) file: []
 - ☐ Open report in browser
 - ☐ Use custom report configuration [Edit report configuration]

Jenkins Project Page:

- Project: EPB Demo - nano-HIL New
- Buttons: add description, Disable Project
- Workspace, Recent Changes, Latest Test Result (no failures)
- Test Result Trend:** A bar chart showing the count of test failures over time. The y-axis is labeled 'count' and ranges from 0 to 6. The x-axis shows build numbers from #18 to #73. The chart shows a significant spike in failures around build #54, reaching a count of 6.
- Permalinks:**
 - Last build (#73), 6 min 50 sec ago
 - Last stable build (#73), 6 min 50 sec ago
 - Last successful build (#73), 6 min 50 sec ago

SUMMARY

testIDEA

- To support automation tools, test vectors can be exported as Python scripts
- Python scripts can be executed from the command line or from within CI tools such as Python
- Scripts can also generate test reports in a variety of formats

