PAUL SCHERRER INSTITUT	Project PROSCAN
Title: pshell testing list	Document No.: toBeDef
Author: Marco Boccioli Co-Author(s):	Date: 16/02/2016

Summary

This document gives an overview of the QA testing tool being implemented as possible replacement of the current QA tool used PROSCAN Collimators (KMA3 and KMA5).

Keywords (max. 5): QA, Test

Org.	Distribution List	Expl.	Org.	Distribution List	Expl.	Approved by	Signature
						Pages	29
						Attachments	
						Signature Author:	
PSI							

pshell-based QA testing tool System Implementation Overview

Paul-Scherrer-Institute

Version 1 - DRAFT

16/02/2016

Table of Content

Revision History

Ver	Date	Name	Reason For Changes	Status
0	02.12.2015	Marco Boccioli	Document created	Draft

1. Introduction

1.1 pshell

pshell is a Java-based tool developed by GFA to supply a generic platform that allows to control any HW relevant to the experiments, and to help in commissioning new devices and beamlines [1].

The platform allows to launch python-based scripts that can sequence a set of operations (for example, testing a motor) followed by analysis of the result. It provides useful set of tools such as EPICS channel access, plotting, logging, archiving. It also supports plug/ins.

1.2 pshell testing list

pshell-testing-list comes in form of a pshell plug-in and adds the possibility of visualising a set of tests, grouped per device. In this way tests can be repeated with same settings and same sequence, allowing collection of data for quality assurance.

1.3 About this document

This document is targeted to those who need to run tests or create new tests using the tool “pshell testing list”.

How to use the platform pshell is not a topic of this document.

1.4 Definitions

The following terms appear in the testing list, inspired by the testing tool FMT/Equate [3]:

Test Suite	A group of tests. A device can be assigned to one test suite. This means, that all the tests under the assigned test suite can be run for the device. A test suite can be assigned to one or more devices.
Test	The test that will run for a specific device. A test can be run for one or for many devices. A device can be assigned to one or many tests (see Test Suite).
Device	The device for which the test will be performed. Each device is assigned to one test suite. A test suite can be assigned to one or more than one device.

Part one - Use of the testing tool

This part describes how to use pshell testing list as testing tool, from the point of view of the end user who just needs to run the predefined tests.

2. Overview of pshell testing list

On a shell box, type `./runpshell` in order to open pshell. The application will show immediately the Testing List. The layout is as in Figure 1.

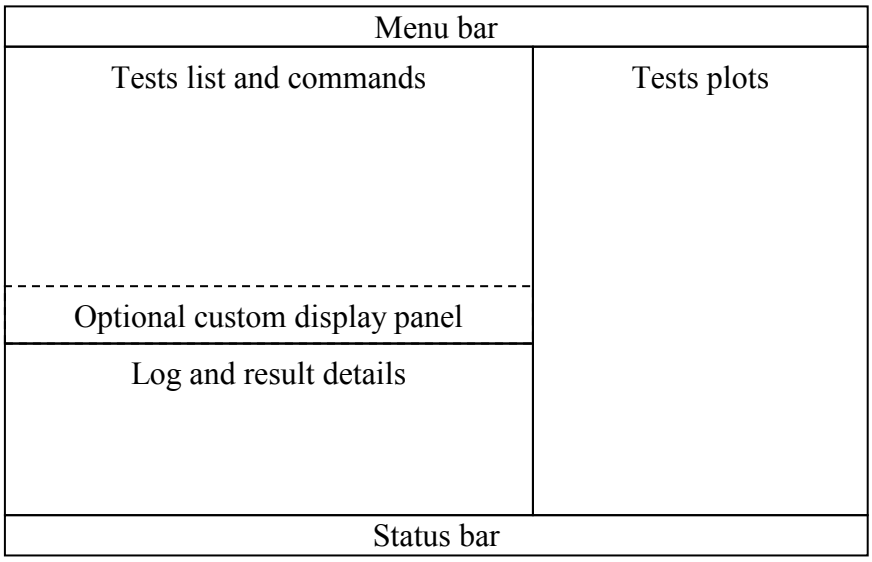

Figure 1 - pshell testing list layout

Tests list and commands	This area includes: the list of available tests, where each line represents a test, associated to a specific device; the commands box, allowing to execute the tests, save the tests sequence and reload it. Settings are available as well.
Tests plots	The plots showing the actual values of the items being tested. The features depend strictly on pshell.
Optional custom display panel	If necessary, specific test suites can come with a custom panel that shows extra information. The panel can be shown/hidden by the user.
Log and result details	Several tabs are available here, providing `Logs` (pshell system messaging), `Scan`, `Data` (history of the tests outputs), `Output` (immediate messages from the test).
Status bar	Shows the status of pshell.
Menu bar	pshell menu.

The main window is shown in Figure 2. In this example, the tests list contains 3 tests. The tests results were successful (green icons) and their related plots are shown on the tabs on the right side of the window (one tab per test). On the left-bottom side, the Output tab shows details of the tests run. In this example, a custom panel called Kollimators is also shown (left-centre).

△ Note: when launching the application, tests cannot run while the Status bar shows `Initialising`.

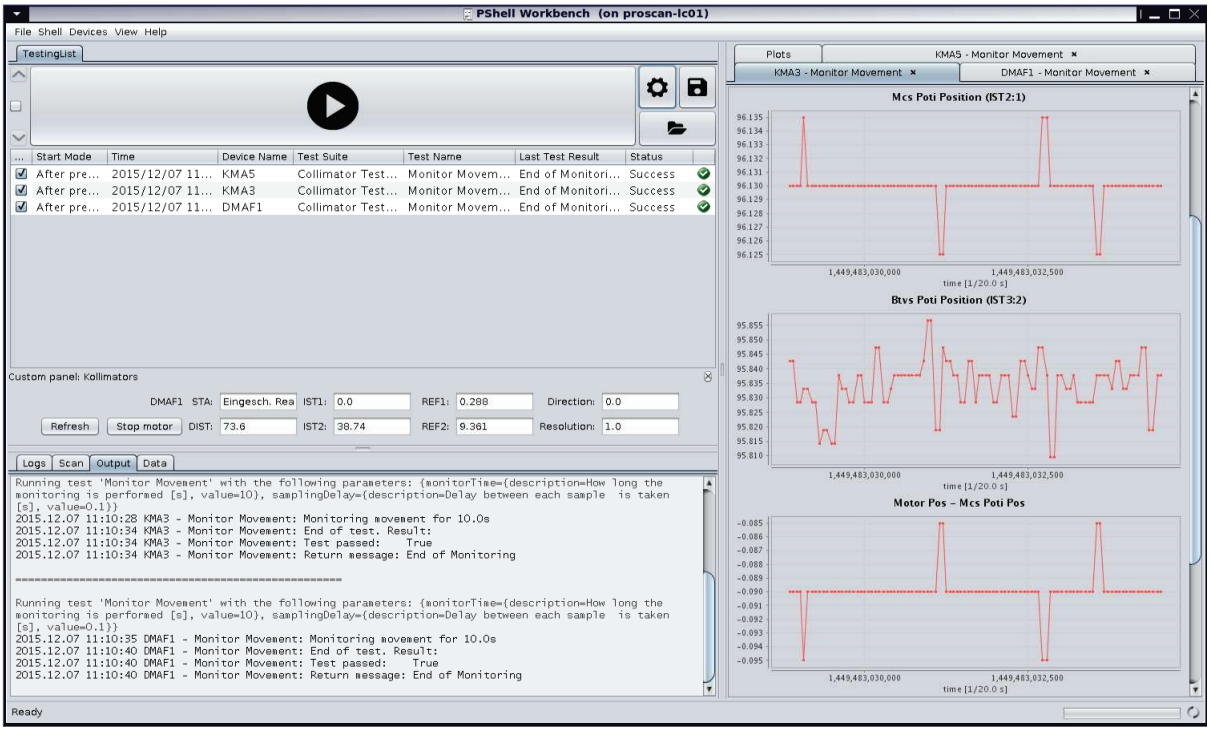

Figure 2 - pshell testing list main window

3. Running predefined tests

3.1 Workflow

In brief, an existing test can be run as follows (each step will be described in the following paragraphs):

- Open a predefined list of tests.
- Optionally, change the tests sequence.
- Run the tests.
- Evaluate the results.

3.2 Opening a predefined list of tests

- Click on 🗁. A list of predefined test sequences appears. It might be empty (i.e. no sequence ever saved).
- Select a predefined test list.
- Click on `Open`.

Loading one test sequence will:

- Order the tests in the predefined saved sequence.
- Enable all the predefined tests.
- Hide all the other tests that are not enabled.

3.3 Running the tests

Once the tests are set in the desired order, they can be launched by clicking on ▶. From this moment, the enabled tests will be run from top to bottom, and in parallel/sequence according to their selected `Start Mode`.

A single test can be launched by right-clicking on it, and selecting `Run this test now`.

In turn, each test running will:

- Show its status as `Running`.
- Print information on the box under the tab `Output`.
- Plot measured data on the tab named as `<Device Name> - <Test Name>`.

The tests can be stopped at any time by clicking on ■.

△ Note: while tests are running, no other action is allowed.

3.4 Evaluating the results

At the end of each test, its result will be shown as summary in the `Status` column and with more details in the column `Last Test Result`. More details are also available under the tab `Output` and in the `Device Details`.

To open the `Device Details`, double-click on the related test line on the table.

The data acquired is usually plotted in the Tests plots area, under a dedicated tab named `<device name> - <test name>`. Plots are pshell-native, and feature several tools available by right-clicking on them.

The results, as well as the data acquired, are automatically saved and can be accessed from the tab `Data` (Figure 3). Archived tests results are organised by date and time.

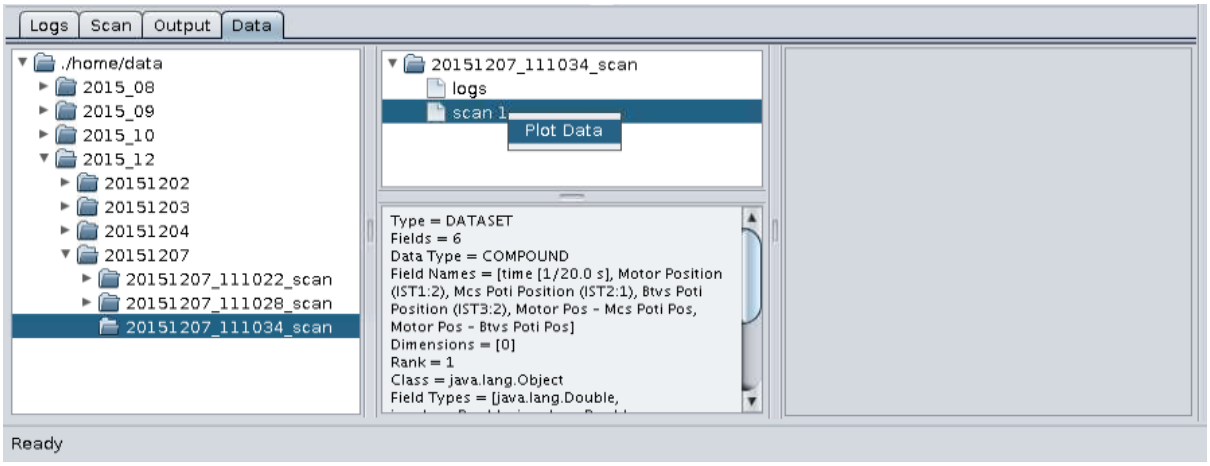

Figure 3 - Browsing the archived tests results

4. Organising the predefined tests

4.1 Modifying the sequence of the tests to run

When pshell is loaded, it will automatically load the predefined testing list last used.

The list of tests can be modified at any time:

- Enable/disable a test by checking/unchecking its checkbox. Batch actions can be done by right-clicking on a selection (Figure 4). If a test is enabled to be run, the status `Pending` will appear under the column `Status`.
- Set a test to run together with the previous test or only after the previous test is completed (see in Figure 5 the combo boxes in the column `Start Mode`).
    - `With previous`: the test will start at the same time as the enabled test above.
    - `After previous`: the test will start right after the end of the test above.
- See all the available tests (regardless from the predefined list of tests open), even those not appearing on the predefined testing list loaded: click on ⚙ and enable the checkbox `Show enabled tests only`.
- Reorder the sequence of the tests: select one or more tests, and click on the buttons ^ or v

△ Note: reordering is available only if the option `Show enabled tests only` is unchecked.

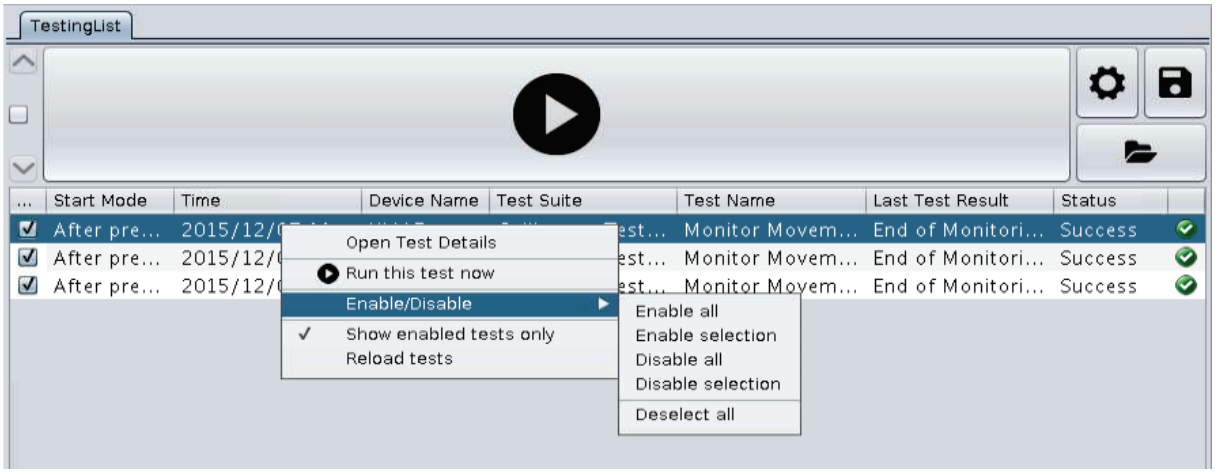

Figure 4 - Enable/disable menu

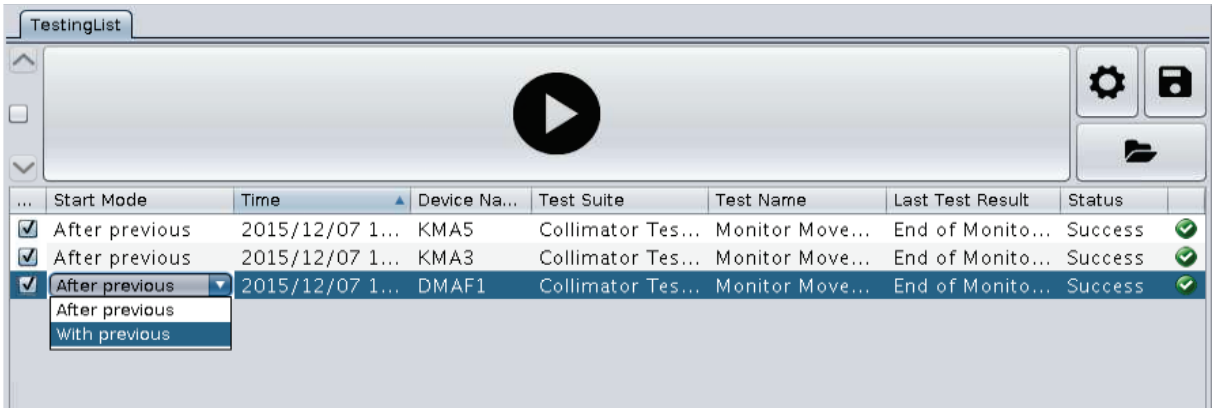

Figure 5 - Setting the start sequence of tests

4.2 Saving a predefined list of tests

The tests that are currently enabled can be saved as a predefined test sequence. The tests currently enabled and their sequence will be saved under a custom name.

- Click on .
- Give a name to the new predefined list of tests.
- Click on `Save`.

4.3 Modifying the parameters of a test

Tests parameters come with default values and can be modified.

Test parameters can be modified from the panel.

- Double-click on the test row to modify. The panel `Device Details` will appear. The parameters appear on the table at the bottom.
- The only element that can be modified is the value of each parameter. On `Test Parameters`, select the parameter to change and click on the cell corresponding to the column `Value`, change the value (see Figure 6).
- Click on `Save` and close the panel.

△ Note: the parameters are associated to the test itself, and not to the device to be tested. The same parameters will be used by the test for all the associated devices.

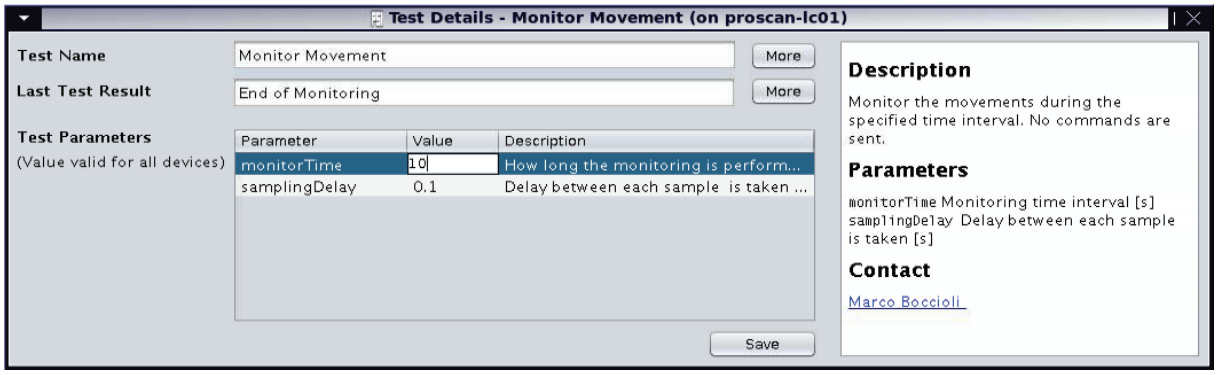

Figure 6 - Test Details panel

4.4 Opening/closing a custom panel

If available, a custom panel can be opened and shown while tests are running. The custom panel will be eventually developed by the test developer.

The custom panel will be animated with the values corresponding to the device currently being tested.

If the testing list includes tests related to only one device, the custom panel will be animated even while no tests are running.

To load a custom panel:

- Click on ⚙ | `Advanced` | `Load Custom Panel`.
- Click on the custom panel to be loaded.

To close a custom panel:

- Click on the button × on the right side of the custom panel.

△ Note: custom panels are designed for specific devices only.

Part two - Administration of the testing tool

This part describes how to maintain pshell testing list, how to configure and to create new tests.

5. Testing structure

A test is a generic program that accepts as only external parameter the name of a device. The pshell testing list picks the device, sees which test suites are associated to a device, then runs the tests contained in that test suite, passing them the name of that device. Figure 7 shows the UML of the testing structure.

Figure 7 - The testing structure

An example of a test-device relationship is depicted on Figure 8. This example translates into the following list:

Test A for Device I

Test B for Device I

Test C for Device I

Test A for Device II

Test B for Device II

Test C for Device II

Test D for Device III

Test E for Device III

All those test cases above would appear on the pshell testing list.

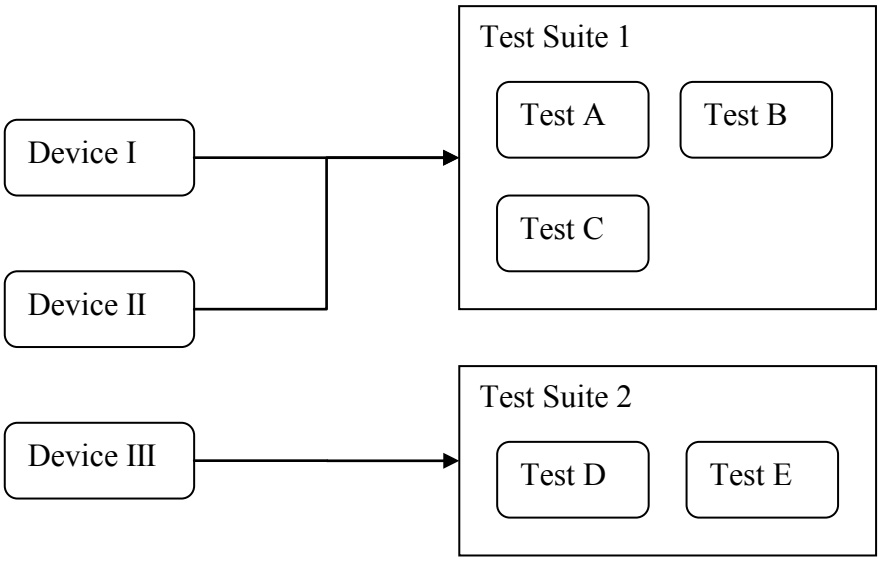

Figure 8 - Example of a testing structure

5.1 Structure of a device

A device must be inside a folder called as the device name, and is composed of the following files:

Type	Example	Description
Config	.config	Device configuration containing: device name, device description, name of the test suite.
Folder	DMAF1	The device folder containing the file above.

The device folder must be inside the following directory:

```
/pshell/home/scripts/tests/devices/
```

All those files are created automatically by the pshell testing list tools (paragraph 8.1).

5.2 Structure of a test suite

A test suite is simply a folder containing one or several tests. For consistency, here is the files list:

Type	Example	Description
Folder	Collimators Tests	The device folder containing the file above.

The test suite folder must be inside the following directory:

```
/pshell/home/scripts/tests/tests/
```

The test suite folder is created automatically by the pshell testing list tools.

5.3 Structure of a test

A test must be inside a folder called as the test name, and is composed of the following files:

Type	Example	Description
Python	Monitor Movement.py	The code containing the test itself.
Config	.config	Test configuration containing: name, description, parameters list with default values.
Help	help.html	Instructions explaining what the test does.
Folder	Monitor Movement	The test folder containing all the files above.

The test folder must be inside the following directory:

```
/pshell/home/scripts/tests/tests/<test suite>/
```

All those files are created automatically by the pshell testing list tools (paragraph 8.3). The job of the developer is to fill the python script with the code needed for the test. This task will be described in the next chapters.

6. Access Control

pshell provides Access Control feature. In this way it is possible to hide/show, enable/disable features depending on who is logged in. For more details on pshell Access Control, please refer to the pshell documentation.

6.1 Default user

By default, the tool loads the `Default` user. For use with the Testing List, the `Default` user is configured with standard privileges. This makes the interface easier to use by hiding some of the pshell advanced features, and by disabling the testing list advanced commands (for example, modification of a test).

In case the user is not the `Default`, the current user is visible on the right-top corner of pshell. How to switch back to `Default` user:

- Select the menu `File | Change User…`
- Select `Default` and press `Ok`.

6.2 Admin user

For administration purposes, the user can be switched to `Admin`. This will enable full access to all pshell features and to advanced function of the testing list, i.e. modify or create tests and devices.

How to switch to `Admin` user:

- Select the menu `File | Change User…`
- Select `Admin` and press `Ok`.

The user `Admin` will stay visible on the top-right corner of the pshell window.

△ Note: all the actions explained in this Part of the manual need `Admin` user.

7. Modifying a test

Be sure to have switched to `Admin` user (see paragraph 6.2).

The code of a test can be modified by directly editing its python file. A direct way can be the following:

- In the tests list, select the test to modify.
- Click on ⚙ `| Advanced | Edit selected test.` The python file of the test will appear on the editor.
- Once finished editing, click on the menu `File | Save`
- Click on the button × on the right side of the tab corresponding to the test.

Once saved, the modified test is already available to be run.

8. Creating new tests and devices

A new test is always contained into a test suite. A device that needs to be tested for that test has to be associated to the test suite containing the test.

The actions described in the following paragraphs will create all the files listed in Chapter 5. Before continuing, be sure to have switched to `Admin` user (see paragraph 6.2).

8.1 New device

A device name is simply a part of an EPICS record name. For example, if `DMAF1:IST:1` is the record name, the device name is `DMAF1`. For EPICS basics, refer to [4].

A new device can be created in the following way:

- Click on ⚙ | `Advanced` | `New Device…`
- On the form `New Device` (Figure 9), fill all the fields marked with an asterisk *:
  - `Device Name` must contain the name of the device to be tested.
  - `Test Case/Suite` is the Test Suite to be assigned to the device. You can create a new one or select from the list of existing Test Suites. In the latter case, the new device will be immediately ready to be tested with all the pool of tests that are contained under the selected Testing Suite.
  - `Device Description` is the description of the new device.
- Click on `Generate`.
- Close the dialog box.

Figure 9 - The form New Device

△ Note: a Device associated to a new Testing Suite will appear on the Testing List only after having created at least one new Test in the same Testing Suite.

8.2 New test

A new test can be created in the following way:

- Click on ⚙ | `Advanced` | `New Test…`
- On the form `New Test` (Figure 10), fill all the fields marked with an asterisk *:
  - `Name` must contain the name of the test.
  - `Test Case/Suite` is the Test Suite to be assigned to the test. You can create a new one or select from the list of existing Test Suites. In the latter case, the new test will be added to the pool of tests that are under the specified Test Suite, and will be automatically available for all the devices that are assigned to that Testing Suite.
  - `Description` is the description of the new test.
  - `Contact Person` is the name of the person to be contacted for questions concerning this test.

△ Note: it is recommended to provide a user name: in this case the Testing List will search for the person information from LDAP. Alternatively, the full name can be given.

  - `Test Parameters` is an optional list of parameters that the test will use and that can be easily changed before running the test. Each parameter consists of the following compulsory fields:
    - `Parameter`: the name of a parameter. One only word allowed. This name will also appear as variable in the test script.
    - `Value`: the value of the parameter: it can be a number or a string.
    - `Description`: description of the parameter, including unit, if any.

    Parameters can be added or removed by right-clicking on the `Test Parameters` table and selecting `Add Row` or `Delete Row` (see Figure 10).

    The test parameters values can be modified at any time before running the test.

△ Note: once the test has been created, test parameters can be added or deleted only manually by editing the test .py file and the test .config file.

- Click on `Generate`.
- A dialog box will ask “Do you want to edit the test script now?”: click `Yes`.
- Edit the python script with the code necessary for the test.

△ Note: a Test associated to a new Testing Suite will appear on the Testing List only after having created at least one new Device with the same Testing Suite.

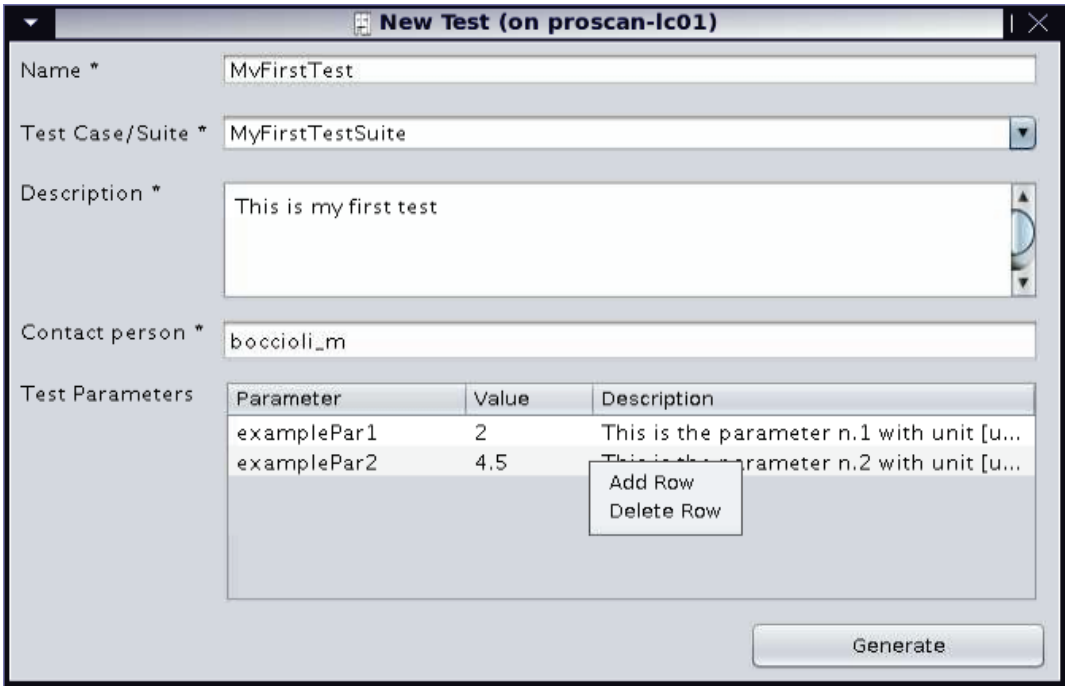

Figure 10 - The form New Test

The new test is now generated and the skeleton of its python script will be displayed for editing. The python file comes with instructions and a documented example that tries to include a typical use case. The test can be already run; nevertheless it will only be a demo, most probably returning an error because the Process Variables accessed by the example code might not exist. The developer must edit the .py file and add the necessary code for the test with the help of the example present in the test.

△ Note: For examples on how to write a test script, refer to the existing testing scripts in `/pshell/home/scripts/tests/tests`
For documentation on specific pshell built-in functions, refer to the pshell help.

8.3 New test suite

New Test Suite can be generated from either `New Device` form or `New Test` form. Since it is just a container for setting the relationship between a Device and a Test, there is no specific panel for it.

9. Creating a custom panel

A custom panel might be useful when some variables need to be constantly visualised, or when the features provided by pshell testing list do not cover the needs of a specific test. The panel must be a java class. This chapter will show how to develop a new panel.

9.1 Example

The custom panel Kollimators is very simple and can be used as starting point to create new panels, and in this chapter it will be used as example in order to help creating further panels:

```
/pshell/home/plugins/Kollimators.java
```

This panel provides real time visualisation of a set of EPICS channels specific for the Collimators (Figure 11, fields on the centre and on the right). It also allows stopping the collimator motor if needed (button `Stop motor` on Figure 11).

Figure 11 - The panel Kollimators

The class declaration is as follows:

```
1  public class Kollimators extends javax.swing.JPanel { …
```

The class must extend `javax.swing.JPanel` in order to be loaded by pshell.

9.2 Compulsory methods

On top of extending `javax.swing.JPanel`, the custom panel class needs only two compulsory, public methods that allow it to get commands from the pshell testing list:

- the constructor,
- the animation method.

The developer is free to implement any other method or subclass.

The constructor must contain one parameter `String params`. It must call all the methods necessary to initialise the class. In this example:

```
2  public Kollimators(String params) {
3      initComponents();
4   }
```

the constructor calls the function `initComponents()` that was automatically generated by the java graphical editor.

pshell Testing List will decide when to animate the panel. For this reason, it expects animating the panel by calling the panel method `animate()`. Therefore, the following method must also be present in the custom panel class:

```
5   public void animate(String deviceName) {
6      g_deviceName = deviceName; //set the global variable with the name of device
7      jLabelDeviceName.setText(deviceName); //show the name of device on the panel
8      connectString(deviceName+":STA:2", jTextSta); //connect (like camon) for Strings
9      connect(deviceName+":DIST:2", jTextDist); //connect (like camon) for Numbers
10     connect(deviceName+":IST1:1", jTextIst1);
11     connect(deviceName+":IST2:1", jTextIst2);
12     connect(deviceName+":REF1:1", jTextRef1);
13     connect(deviceName+":REF2:1", jTextRef2);
14  }
```

The method `animate()` contains all the code needed to animate the panel. The content is of course specific to the panel. In this example, the method contains connection calls that are needed to animate the panel test fields with several record values. See the next paragraph for more details.

9.3 Other functions in the example

The example class Kollimator contains other functions that might be useful to develop new panels. For this reason they are described here even though they are not compulsory.

The following method creates a link with an EPICS record, visualising its current value:

```
15  public void connect(String channelName, JTextField textField) {
16      try {
17          ChannelDouble channel = new ChannelDouble(channelName, channelName, 3);
18          channel.setMonitored(true);
19          channel.initialize();
20          channel.addListener(new DeviceAdapter() {
21              @Override
22              public void onValueChanged(Device channelName,Object value,Object former) {
23                  Double val = (Double)value;
24                  textField.setText(String.valueOf(val));
25                  textField.setToolTipText(channelName.getName());
26              }
27          });
28      } catch (Exception ex) {
29          System.out.println(ex.toString());
30      }
31  }
```

The method receives a string with the name of the EPICS channel to connect to, and a text field object where to show the current value of the record. After creating the channel, it adds a listener (line 20), creating the callback function (line 22). The callback function sets the value of the text field to the value received from the record. It also writes the record name to the text field tooltip text.

A very similar function is available: `connectString(). It` is identical to `connect(),` with the only difference on line 23, where the casting is done to `String` instead of `Double`, and the variable `val` is declared as `String` instead of `Double`.

The following method is called by the event `jButtonStopMotorActionPerformed()` on the button `Stop motor`:

```
32  public void stopMotor() {
33      try {
34          ChannelDouble channel = new ChannelDouble(g_deviceName+":COM:2",
35                                                    g_deviceName+":COM:2", 0);
36          Object stopVal = 0;
```

```
37          channel.write((Double) stopVal);
38      } catch (Exception ex) {
39          System.out.println(ex.toString());
40      }
41  }
```

In this method, the channel is created with the device name picked from the global variable `g_deviceName`, and the value `0` is written to the channel.

9.4 Enabling the panel

Once the panel class is created and saved (no build necessary), it must be enabled as plug-in into pshell:

- Copy the following files:

  `<yourNewPanel>.form`

  `<yourNewPanel>.java`

  into the directory:

  `/pshell/home/plugins/`

- Restart pshell. pshell will build the java class while starting.
- On pshell, select the menu entry `File | Plugins…`
- On the `Plugins` window, tab `Enabling`, check the box related to your new panel name.
- Click on `Save` and close the panel.
- Restart pshell.

10. Re-installing testing list into pshell

The Testing List comes as plug-in for pshell. After phell has been re-installed, here are the steps to re-install Testing List plug-in to pshell.

The source of the pshell testing list can be found on Git:

```
git@git.psi.ch:pshell_config/ncs.git
```

Its directory structure is identical to the one of pshell. Therefore the source directory always matches the destination directory.

- From the Git repository, get the file:

  ```
  pshell-plugin-testing-list.zip
  ```

- Close pshell.
- Extract the zip file into the pshell installation directory:

  ```
  pshell/home
  ```

- Confirm overwriting all files.
- Restart pshell.

References

[1] pshell presentation from Alexandre Gobbo: https://controls.web.psi.ch/TWiki-4.1.2/pub/Main/ControlsTalk151/ctdez2014-pshell.pptx

[2] FDA wiki: https://controls.web.psi.ch/cgi-bin/twiki/view/Main/FlexibleDataAcquisitionFDA

[3] FMT/Equate contact person: Simon Rees.

[4] EPICS Training at PSI : http://epics.web.psi.ch/training/handouts/e_EPICS_Training_at_PSI.ppt