Using µManager supported devices in Zen

Jérôme Mutterer[1*], Julien Kissenberger[2] and Fabrice Schmitt[2]

[1]C.N.R.S., Strasbourg, France.
[2]Carl Zeiss S.A.S., Marly-le-Roi, France.

*Corresponding author(s). E-mail(s): jerome.mutterer@cnrs.fr;
Contributing authors: julien.kissenberger@zeiss.com;
fabrice.schmitt@zeiss.com;

Abstract

Microscopy setups can include common or custom devices for which no driver is available in the Zen software. In this white paper, we propose a method and show an example of creating a basic graphical user interface for driving those peripherals from Zen, provided a matching µManager device adapter is available.

Keywords: drivers, adapters, devices, µManager, pycromanager, Zen

1 Introduction

Advanced microscopy experiments conducted in core imaging facilities typically require systems of two very distinct types. First, we find complete commercially available systems. Those usually consist of official manufacturer hardware parts like the microscope stand, beam path mirrors, objectives or reflector turrets. Motorized versions of those core elements are of course well integrated into the company's provided microscope control software. For advanced experiments, these core elements can be complemented by so-called third party provided pieces of hardware, as is often the case for motorized stage, specific illuminators, additional filter wheels, or environmental control devices. The ability of the main control software to communicate with those third party devices will make user training and workflows execution easier, as a single software can be used throughout the experiment.

Home-made, or custom microscopy setups make up the second type of core facilities imaging systems. Those can be anything ranging from 3D printed structural parts to a mix of up cycled outdated stands, standalone filter wheels and Arduino driven illuminators or motion controllers. µManager, the Open Source microscope control and automation software [1, 2] is often used to try and integrate those diverse hardware parts, given its extensive list of so called device adapters supporting direct communication with the devices themselves or talking to the system level device drivers.

While the current version of µManager is impressively capable for advanced acquisition protocols, commercially available control software like Zen [3] still benefit from smoother user experience, with greater development efforts targeted to user experience.

Here we present a simple step-by-step method to create a Zen graphical user interface that drives devices configured to work in µManager, enabling creating imaging setups comprising of both vendor supported and µManager supported hardware, using the `pycromanager`[4] Python library.

2 Overview

1. Prerequisite: Zen 3.4 (blue edition), with Macro Environment Module
2. Install µManager and configure devices
3. Install Python 3.6
4. Install and test pycromanager
5. Create Python scripts to query and alter device state
6. Create Zen Macro that invokes scripts from step 5

3 Walk-through

3.1 Prerequisite: Zen 3.4 (blue edition), with Macro Environment Module

This tutorial is intended for a working imaging setup running Zen 3.4. It uses Zen OAD Macros, so make sure the Macro Environment Module is activated. To do this, navigate to Zen>Tools>Modules Manager...

Fig. 1 Zen optional modules manager

3.2 Install µManager and configure devices

In this section, we install µManager and configure a sample device according to instructions available from the µManager wiki at https://micro-manager.org/

Retrieve the latest µManager software from the download page. For this tutorial, we chose the Windows 64-bits version version 2.0.0.

Click through the installer steps and launch µManager.

In the "Micro-Manager Startup Configuration" dialog, select "Default User" profile and "(none)" as "Hardware Configuration File" as we're going to add just the devices we need.

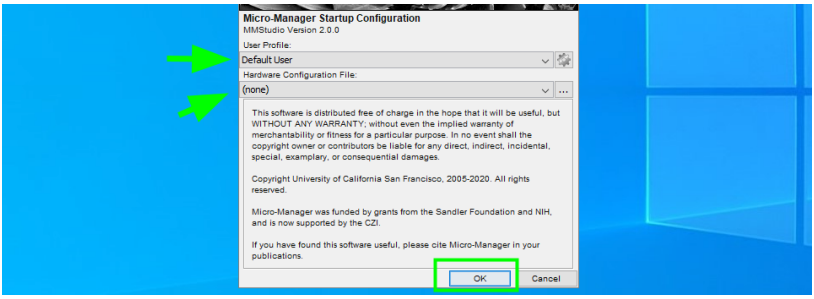

Fig. 2 µM Startup Configuration Dialog

µManager starts with no devices configured.

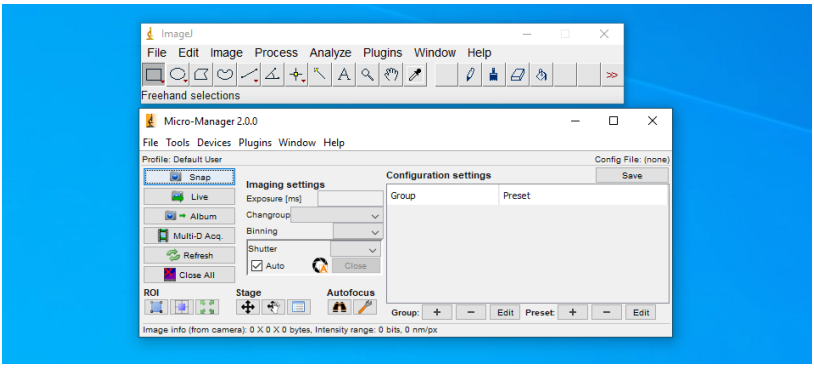

Fig. 3 Main µM user interface

The example device we will use is a custom made device emulating a PrecisExcite LED illuminator. It is built around an Arduino micro-controller and a NeoPixels RGB LED ring. It is connected to the host computer using a serial over USB port, and the micro-controller firmware implements a subset of the official PrecisExcite protocol. This allows µManager hardware configuration tool to recognize the device as a 3 channels illumination device with shutter. Further examples of emulated custom devices and their Arduino firmwares can be found at https://github.com/mutterer/MM_CustomArduino

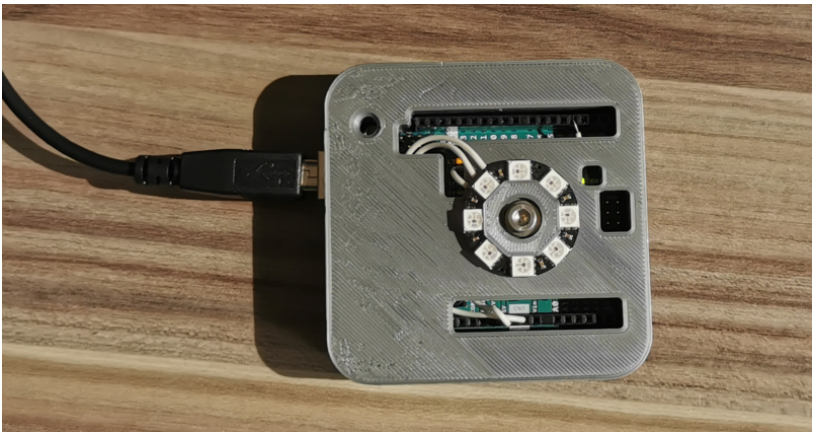

Fig. 4 A custom illumination device built around an Arduino micro-controller

Now start the configuration assistant by navigating to Micro-Manager>Devices>Hardware Configuration Wizard...

1. Chose "Create new configuration" and press "Next"
2. From the devices list, open the "PrecisExcite" folder, select the "PrecisExcite: PrecisExcite LED illuminator" device and add it using the "Add..." button

3. In the "device initialisation properties dialog" specify the COM port to which the device is connected, change "Answer Timeout" to 2000. You can leave all other default value. Proceed by pressing "OK" button.
4. Click through the rest of the Hardware Wizard, and at the last step, give this configuration a file name and click the "Finish" at step 6/6.

µManager loads and applies the configuration we just created. We can now test this device by going to Micro-Manager>Devices>Devices Properties Browser, a dialog in which all device properties are listed and can be controlled.

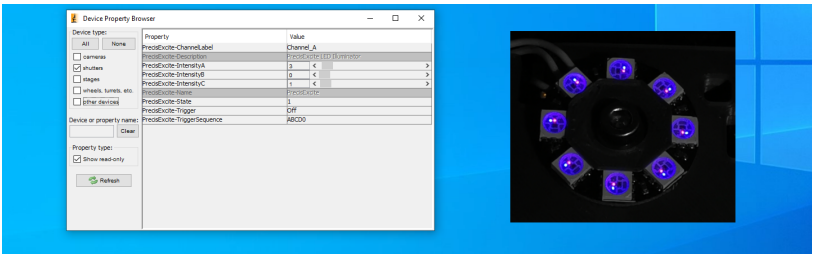

Fig. 5 Play with the IntensityA,B,C sliders and State properties to assess your device is working properly

Last step in µManager, we need to open a server port to allow external software control. Go to Micro-Manager>Tools>Options... and activate the "Run Server on port 4827" checkbox. Everything is now set for control from the outside.

3.3 Install Python 3.6 and pycromanager

We're going to use pycromanager to send device state commands to µManager. We can't, however, do this directly from Zen, as Zen Macros make use of Python 2.7 whereas pycromanager requires Python 3.6.

3.3.1 Python 3.6

We start by installing Python 3.6 and follow the steps described at:

`https://www.python.org/downloads/release/python-360/`

3.3.2 pycromanager

pycromanager [4] is a Python package that allows controlling µManager using the Python programming language. Installation is straight forward. Simply run:

```
pip install pycromanager
```

The extensive documentation at pycromanager website provides a test program to make sure it can communicate with µManager. Open a Python command line window, breath, and type the following code:

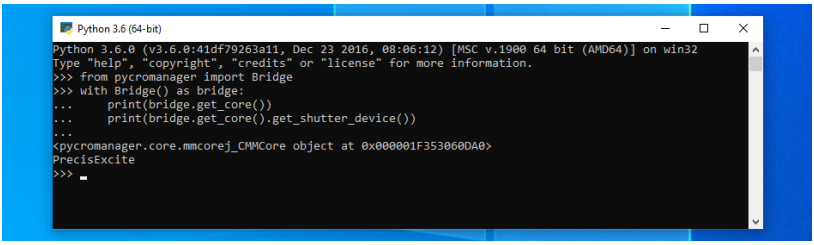

Fig. 6 Testing pycromanager successful installation

We were able to programmatically get a reference to µManager core and from this the name of the current shutter device we installed earlier.

3.4 Python scripts

It is unfortunate that we cannot directly use pycromanager from Zen macros, so we first need to create two intermediate scripts.

3.4.1 Device query script

The first one queries the current illuminator states, the 3 channels intensities and the shutter state and replies with a simple text string containing those values. The following code is basic and can easlily be improved.

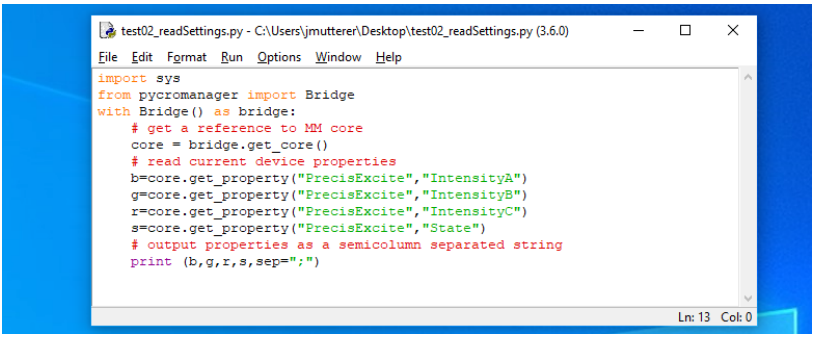

Fig. 7 A python script that queries device state and reports it

You can test this script by opening a Windows command line and typing:

```
python Desktop/test02_readSettings.py
10;20;8;1
```

The replied string means intensities for channels A,B,C are 10,20,8, and the final 1 means the shutter is open (LEDs are shining).

3.4.2 Device driving script

The second script is in charge of setting device properties. It must be called with arguments and uses pycromanager to send the matching commands to µManager.

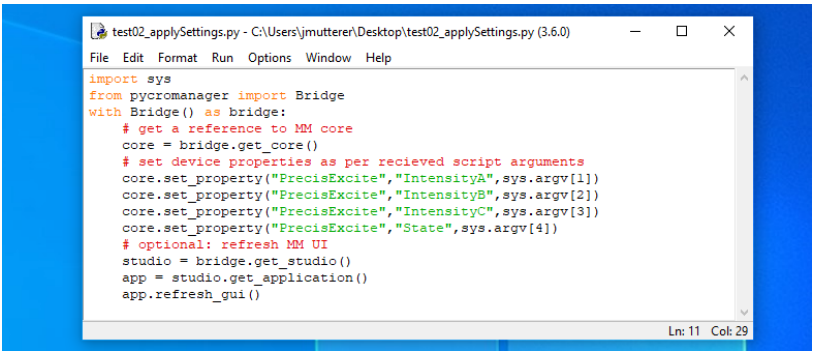

Fig. 8 A python script that sets device states from arguments it receives

You can test this script by opening a Windows command line and typing:

```
python Desktop/test02_applySettings.py 50 0 50 1
```

There's no reply for this script, but the device should display a bright purple color with 50% blue, 0% green, 50% red, and an open shutter state (1). Congratulations, we have all pieces to put everything together in a Zen macro!

3.5 Zen macro

3.5.1 A basic example

The last part to design is a Zen macro that will:

1. Execute the first script from 3.4.1 to query the device state
2. Build a user interface with sliders for channels intensities and a shutter checkbox
3. Execute the second script from 3.4.2 to alter the device state

Open Zen. Navigate to Zen>Macro>Macro Editor... and enter the following code:

We kept code to a minimum on purpose, but it will be easy to extend this model.

In a basic setting, you could easily create a Zen Toolbar button item to run this code by using the Macro Editor's "Toolbar Configuration" widget and associate a new toolbar button with the current macro.

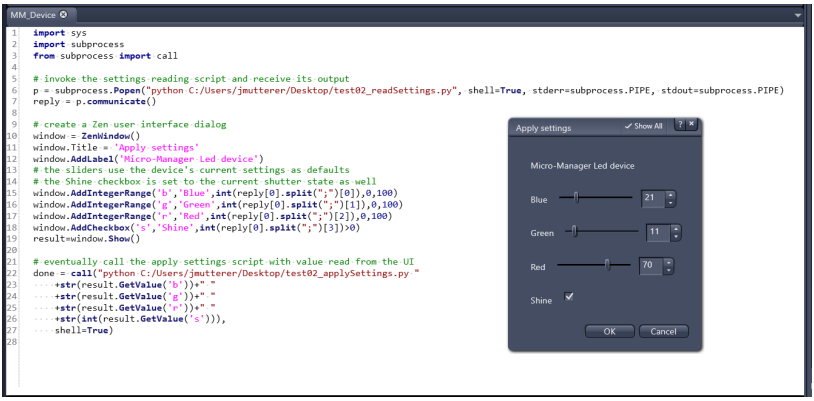

Fig. 9 The Zen macro wrapping calls to pycromanager scripts and displaying a user interface

3.5.2 Automation use case

For more complex experiment workflows, a first approach could be to use the Zen built-in "Automation" module. This module allows to execute given macros before and after experiments. A use case would be to pause fluid flow in a perfusion imaging chamber, driving an µManager configured peristaltic pump.

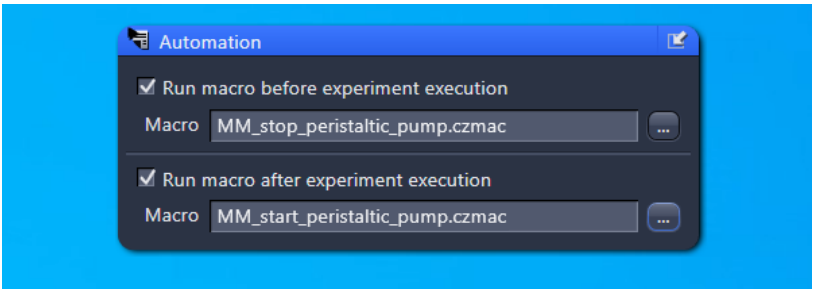

Fig. 10 The "Automation" dialog specifying pre- and post-experiment actions (macros)

3.5.3 Experiment feedback use case

Whenever actions should be taken during experiment, driving external devices could be integrated in the "Experiment Feedback" workflow design. Experiment feedback provides acquisition hack points at experiment startup and completion, but also allows Zen Macro code to be be upon events occurring at the "observable" level. Observables are system or environment properties whose values are monitored and can be used to trigger code execution. The following example will use those acquisition workflow entry points to adjust illumination level for each time point in an sample experiment and finish experiment by resetting the device after a given value was reached.

First in your "Acquisition" tab, enable "Time Series" and "Experiment Feedback" check boxes. In the matching dialogs, Set time series duration to "As long as possible". Set "Script runtime condition" to "Free Run".

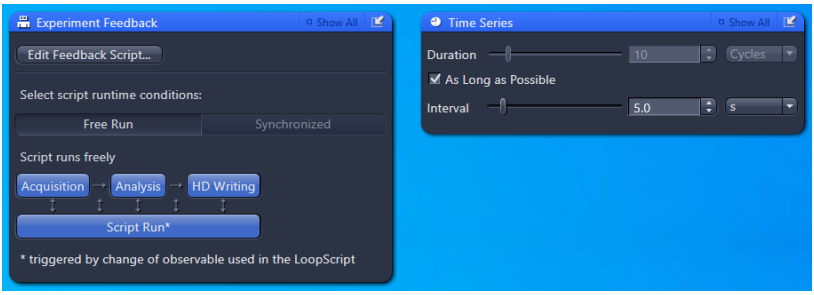

Fig. 11 Setting up a time lapse experiment with feedback

Now, open the "Feedback Script Editor" and enter the following pre-loop, loop, and post-loop scripts.

The PreLoop script initializes an intensity variable and paths to the Python 3.6 interpreter and pycromanager script responsible to drive our device through µManager.

The Loop script monitors the "CurrentTimePointIndex" observable and runs the pycromanager script whenever the observable changes. It also increments the intensity, and stops the acquisition when a given threshold is crossed.

Finally, the PostLoop script turns off our device after the acquisition.

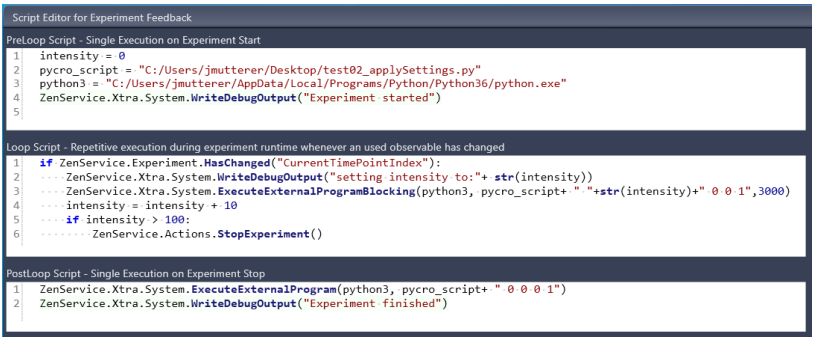

Fig. 12 Experiment Feedback scripts

4 Conclusion

Not all commercially available and basically none of custom made devices benefit from integration with Zen software. In this white paper, we present a way to drive devices from Zen, provided they can be configured to work with μManager in the first place. The extensive list of supported devices in μManager and the ability to easily create μManager device adapters, or emulate existing similar devices opens opportunities for using a wide range of such devices in Zen. The proposed method uses OAD Zen macros and interface python scripts using the pycromanager library to talk to μManager, and illustrate how Open Source and scriptable vendor software can cooperate to achieve better device support and improved user experience.

References

[1] Edelstein, A., Amodaj, N., Hoover, K., Vale, R., Stuurman, N.: Computer control of microscopes using μmanager. Current Protocols in Molecular Biology **92**(1), 14–201142017 (2010). https://doi.org/10.1002/0471142727.mb1420s92

[2] Edelstein, A.D., Tsuchida, M.A., Amodaj, N., Pinkard, H., Vale, R.D., Stuurman, N.: Advanced methods of microscope control using manager software. Journal of Biological Methods **1**(2), 10 (2014). https://doi.org/10.14440/jbm.2014.36

[3] Zen 3.4. Zeiss Efficient Navigation, Zeiss Microscopy (2021). https://www.zeiss.com/microscopy/int/products/microscope-software/zen.html

[4] Pinkard, H., Stuurman, N., Ivanov, I., Anthony, N., Ouyang, W., Li, B., Yang, B., Tsuchida, M., Chhun, B., Zhang, G., Mei, R., Anderson, M., Shepherd, D., Hunt-Isaak, I., Dunn, R., Jahr, W., Kato, S., Royer, L., Thiagarajah, J., Eliceiri, K., Lundberg, E., Mehta, S., Waller, L.: Pycromanager: open-source software for customized and reproducible microscope control. Nature Methods **18(3)**, 226–228 (2021). https://doi.org/10.1038/s41592-021-01087-6