



Tektronix Simplifying Test Automation With tm_ Devices And Python User Guide

[Home](#) » [Tektronix](#) » Tektronix Simplifying Test Automation With tm_ Devices And Python User Guide 

Tektronix® Simplifying Test Automation with tm_devices and Python HOW-TO GUIDE



Contents

- 1 Simplifying Test Automation With tm_ Devices And Python
- 2 What is a Programmatic Interface?
- 3 What is the tm_devices Package?
- 4 Setting up your Environment
- 5 Example Code
- 6 Using IntelliSense/Code Completion
- 7 Docstring Help
- 8 Extra Resources
- 9 Troubleshooting
- 10 Appendix A – Offline Installation of tm_devices
- 11 Documents / Resources
 - 11.1 References

Simplifying Test Automation With tm_ Devices And Python

HOW-TO GUIDE

Simplifying Test Automation with tm_devices and Python

Engineers across many industries use automation to extend the capabilities of their test instruments. Many engineers choose the free programming language Python to accomplish this. There are many significant advantages that make Python a great programming language for automation:

- Versatility
- Easy to teach and learn
- Code readability
- Widely available knowledge bases and modules

There are two main use cases for automation:

- Routines that mimic human behavior to automate the front panel and save time e.g., automated compliance testing.
Rather than sitting down at the scope, adding appropriate measurements, and writing down the results every time you need to test a new part, the engineer develops a script that does all of that and displays the result.
- Uses that extend the functionality of the instrument; for example: measurement logging, validation, or quality assurance.

Automation allows the engineer to execute complex tests without many of the downsides inherent to those tests. There's no need for an operator to set up the scope and manually record the results, and the test can be performed the same way every time.

This how-to guide will cover what you need to get started programming scopes in Python, including the basics of programmatic interfaces and how to download and run an example.

What is a Programmatic Interface?

A programmatic interface (PI) is a boundary or set of boundaries between two computing systems that can be programmed to execute specific behaviors. For our purposes, it's the bridge between the computer that runs every piece of Tektronix test equipment, and the application written by an end user. To narrow this even further, it is a set of commands that can be sent remotely to an instrument which then processes those commands and executes a corresponding task. The PI Stack (Figure 1) shows the flow of information from the host controller down to the

instrument. The application code written by the end user defines the behavior of the target instrument. This is usually written in one of the development platforms popular in the industry such as Python, MATLAB, LabVIEW, C++, or C#. This application will send data using the Standard Commands for Programmable Instrumentation (SCPI) format, which is a standard supported by most test and measurement equipment. SCPI commands are often sent through a Virtual Instrument Software Architecture (VISA) layer, which is used to facilitate the transfer of data by including additional robustness (e.g., error checking) to the communication protocol. In some cases, applications may call a driver which will then send one or more SCPI commands to the VISA layer.

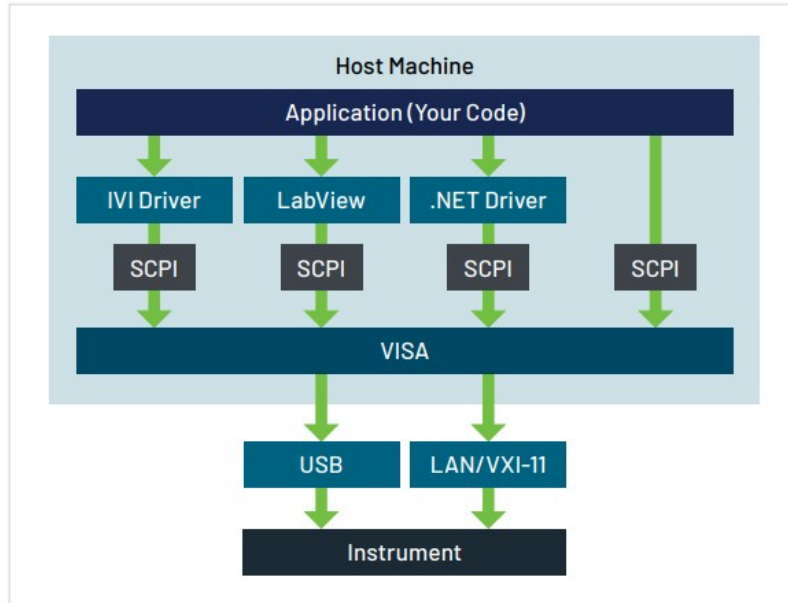


Figure 1. The programmatic interface (PI) stack shows the flow of information between a host controller and instrument.

What is the `tm_devices` Package?

`tm_devices` is a device management package developed by Tektronix that includes a multitude of commands and functions to help users easily automate tests on Tektronix and Keithley products using the programming language Python. It can be used in the most popular IDEs for Python and supports code-completion aids. This package makes coding and test automation simple and easy for engineers with software skills of any level. Installation is also simple and uses `pip`, Python's package-management system.

Setting up your Environment

This section will guide you through the prerequisites and installations to prepare you to do development work with `tm_devices`. It also includes instructions that support virtual environments in Python (venvs) to make your projects easier to manage and maintain, especially if you are just trying this package out before committing to its usage.

Note: If you have an environment without direct access to the internet you will have to modify your steps using the commands in the appendix. If you are having problems feel free to post in the [github discussions](#) for assistance.

Installation and Prerequisites Overview

1. Install Python
 - a. Python ≥ 3.8
2. PyCharm – PyCharm Installation, Starting a project, and `tm_devices` installation
3. VSCode – VSCode Installation, Starting a project, and `tm_devices` installation

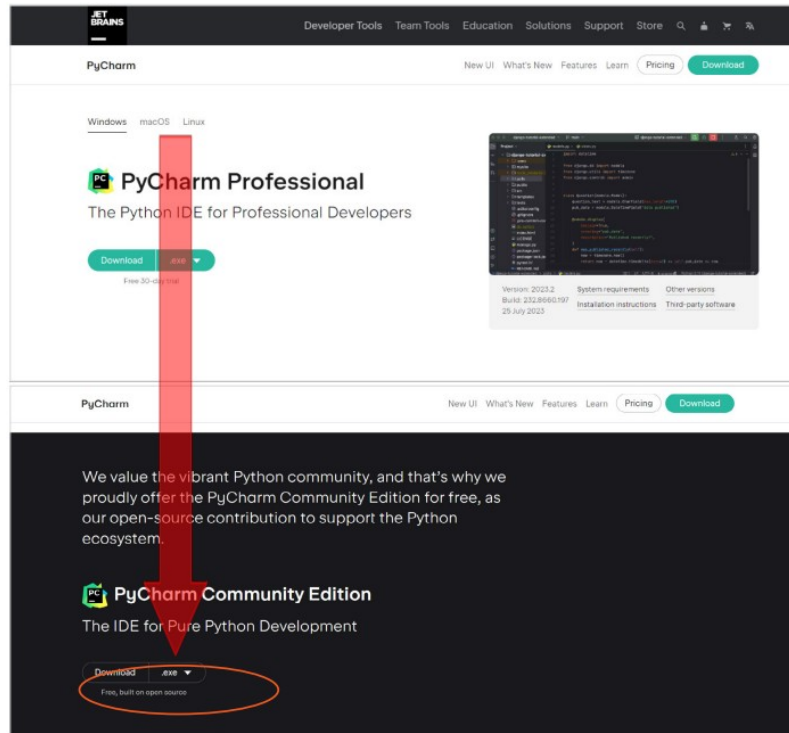
PyCharm Community (free) edition

PyCharm is a popular Python IDE used by software developers across all industries. PyCharm has an integrated unit tester which allows users to run tests by file, class, method, or all tests within a folder. Like most modern IDE's it has a form of code completion that speeds up your development tremendously over a basic text

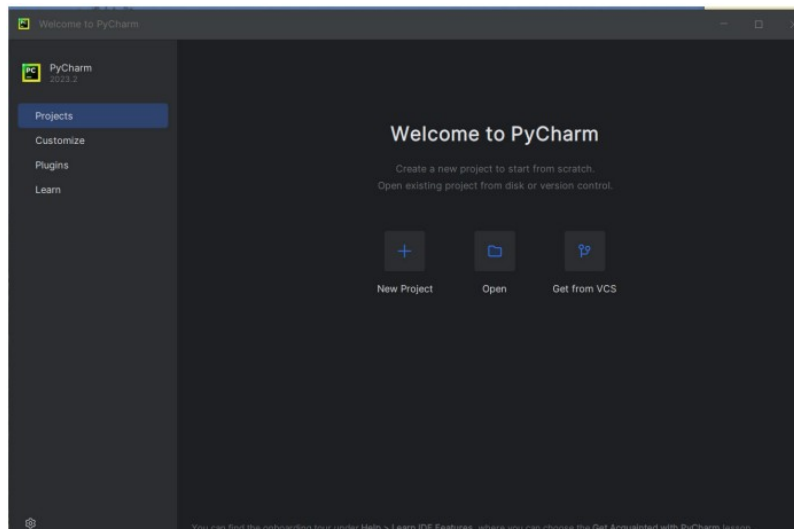
editor.

We will walk through the installation PyCharm community edition (free), followed by installing tm_devices in the IDE and setting up a virtual environment to develop in.

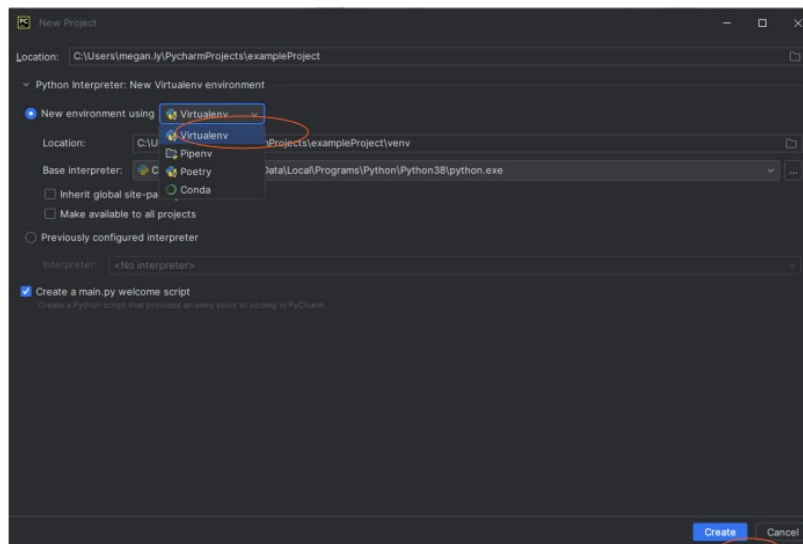
1. Go to <https://www.jetbrains.com/pycharm/>
2. Scroll past PyCharm Professional to PyCharm Community Edition, click download



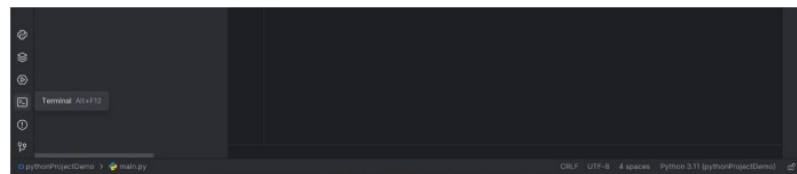
3. You should be able to proceed with just the default installation steps. We do not require anything unique.
4. Welcome to PyCharm!



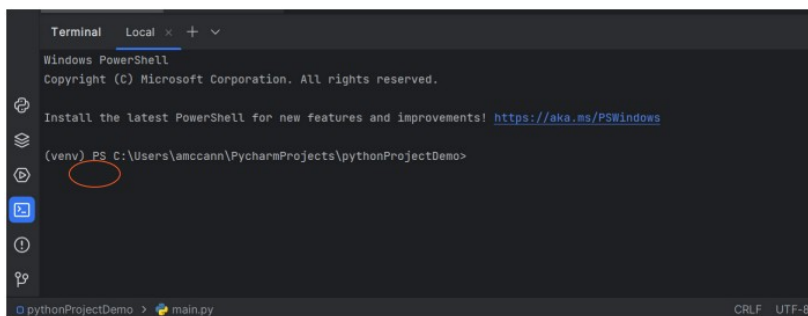
5. Now you will need to create a new project and make sure to set up a virtual environment. Click "New Project"
6. Confirm path for project, make sure "Virtualenv" is selected



7. Open a terminal. If your view does not include the labeled button at the bottom look for this:

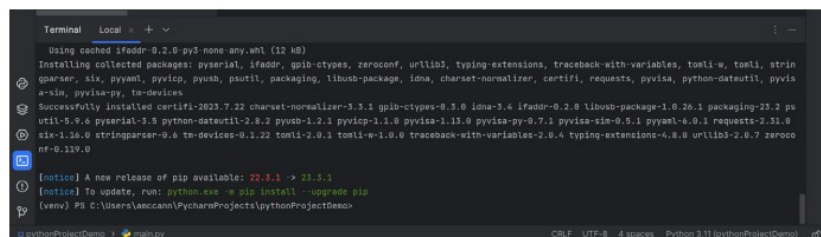


8. Confirm virtual environment is set up by checking for (venv) before the prompt in your terminal



9. Install driver from the terminal

Type: `pip install tm_devices`



10. Your terminal should be error free! Happy hacking!

Visual Studio Code

Visual Studio Code is another popular free IDE that software developers across all industries use. It is great for most languages and has extensions for most languages that make coding in this IDE very convenient and efficient. Visual Studio Code provides IntelliSense which is an extremely useful tool when developing as it aids in code completion, parameter information, and other information regarding objects and classes. Conveniently, `tm_devices` supports code completion that describes the command tree of the objects and classes.

We have an excellent guide on the installation of both Python and Visual Studio Code, including information on virtual environment setup [here](#).

Example Code

In this section we will step through pieces of a simple code example and highlight some necessary components to use `tm_devices` effectively.

Imports

```
from tm_devices import DeviceManager
from tm_devices.drivers import MSO5B
```

These two lines are critical to the effective usage of `tm_devices`. In the first line we import the `DeviceManager`. This will handle the boilerplate connecting and disconnecting of multiple device classes. In the second line we import a specific driver, in this case the `MSO5B`. We setup a context manager with the `DeviceManager`:

```
with DeviceManager(verbose=True) as device_manager:
```

And then when we use the device manager and driver together:

```
scope :MSO5B= device_manager.add_scope("192.168.1.1")
```

We can instantiate an instrument with a specific command set that matches its model. Just input your instrument's IP address (other VISA addresses work as well).

With these four lines complete, we are able to start writing meaningful and specific automation for the `MSO5B`!

Code Snippets

Let's take a look at a few simple actions –

Setting the Trigger type to Edge

```
# Setting Trigger A to Edge
scope.commands.trigger.a.type.write("EDGE")
```

Here's how you would add and query a peak-to-peak measurement on CH1:

```
# Specifying source as Channel 1
scope.commands.display.select.source.write("CH1")
# Identifying pk2pk as the measurement we would like to make
scope.commands.measurement.addmeas.write('PK2Pk')
# Make sure the operation is complete using the opc command
scope.commands.opc.query()
# Store the value locally before we print
chipk2pk = float(scope.commands.measurement.meas[1].results.allacqs.mean.query())
# Printing the value onto the console
print(f'Channel 1 pk2pk: {chipk2pk}')
```

If you wanted to take an amplitude measurement on CH2:

```
# Specifying source as channel 2
scope.commands.display.select.source.write("CH2")
# Identifying amplitude as the measurement we would like to make
scope.commands.measurement.addmeas.write('AMPLITUDE')
# Make sure the operation is complete using the opc command
scope.commands.opc.query()
# Store the value locally before we print
ch2amp = float(scope.commands.measurement.meas[2].results.allacqs.mean.query())
# Print the value onto the console
print(f'amplitude: {ch2amp}')
```

Using IntelliSense/Code Completion

IntelliSense – Microsoft's name for Code Completion is a very powerful feature of IDE's we have tried to exploit as much as possible.

One of the core barriers to automation with test and measurement devices is the SCPI command set. It is a dated structure with syntax not widely supported in the development community.

What we have done with `tm_devices` is create a set of Python commands for each SCPI command. This allowed us to generate Python code from existing command syntax to avoid manual development of the drivers, as well as create a structure that is familiar to existing SCPI users. It also maps to the lower-level code that might require intentional debugging during your program creation. The structure of the Python commands mimics the SCPI (or in some Keithley cases TSP) commands structure so if you are familiar with SCPI you will be familiar with these.

This is an example of how IntelliSense shows all the commands available with the previously typed command:

In the scrollable list that appears after the dot on `scope` we can see an alphabetical list of scope command categories:


```
scope.commands.
# Specifying source
scope.commands.
# Identifying p
scope.commands.
# Query the com
scope.commands.
# Assigning var
ch1pk2pk = float
# Printing the
print(f'Channel
# Specifying source as channel 2
```

Choosing afg we are able to then see a list of AFG categories:

```
scope.commands.afg.
# Specifying source
scope.commands.disp
# Identifying pk2pk
scope.commands.meas
# Query the command
scope.commands.opc.
# Assigning variabl
ch1pk2pk = float(sc
# Printing the valu
print(f'Channel 1 p
# Specifying source as channel 2
```

Final command written with the help of IntelliSense:

```
scope.commands.afg.amplitude.write(10e6)
```

Docstring Help

As you code, or as you are reading someone else's code, you can hover over different parts of the syntax to get that level's specific help documentation. The closer you are to the full command syntax the more specific it will get.

```
(property) commands: MSO5BCommands
Return the MSO5B commands.
This provides access to all the commands for the MSO5B device. See the documentation of each sub-property for more usage
information.
Sub-properties:
  • .acquire: The ACQUIRE command.
  • .actonevent: The ACTONEVENT command tree.
  • .afg: The AFG command tree.
  • .alias: The ALIAS command.
  • .allev: The ALLEV command.
  • .application: The APPLICATION command tree.
  • .autosavepertimeout: The AUTOSAVEPETIMEOUT command.
  • .autosaveuitimeout: The AUTOSAVEUITIMEOUT command.
  • .autoset: The AUTOSet command.
  • .auxout: The AUXout command tree.
  • .bus: The BUS command tree.
  • .bustable: The BUSTABLE command tree.
# Spec
scope.commands.
```

Depending on your IDE conditions you can display both IntelliSense and docstring help at the same time.

```
(property) afg: Afg
Return the AFG command tree.
**Usage**
• Using the .query() method will send the AFG? query.
• Using the .verify(value) method will send the AFG? query and raise an AssertionError if the returned value does not match value.
Sub-properties:
• .amplitude: The AFG:AMPLitude command.
• .arbitrary: The AFG:ARbitrary command tree.
• .burst: The AFG:BURST command tree.
• .frequency: The AFG:FREQUENCY command.
• .function: The AFG:FUNCTION command.
• .highlevel: The AFG:HIGHELevel command.
• .lowlevel: The AFG:LOWLevel command.
• .noiseadd: The AFG:NOISEAdd command tree.
• .offset: The AFG:OFFSet command.
• .output: The AFG:OUTPut command tree.
# Specifying
scope.commands.afg.
query
  amplitude
  arbitrary
  burst
  cmd_syntax
  frequency
  function
  highlevel
  lowlevel
  noiseadd
  offset
  output

(property) amplitude: AfgAmplitude
Return the AFG:AMPLitude command.
**Description**
• Sets (or queries) the AFG amplitude in volts, peak to peak.
**Usage**
• Using the .query() method will send the AFG:AMPLitude? query.
• Using the .verify(value) method will send the AFG:AMPLitude? query and raise an AssertionError if the returned value does not match value.
• Using the .write(value) method will send the AFG:AMPLitude value command.
**SCPI Syntax**
- AFG:AMPLitude <NR3>
- AFG:AMPLitude?
**Info**
• <NR3> is a floating point number that represents the AFG amplitude, peak to peak, in volts.
# Specifying
scope.commands.afg.amplitude
```

With this guide you have seen some of the benefits of Tek's python driver package `tm_devices` and can start your automation journey. With the easy setup, code completion, and built-in help you will be able to learn without leaving your IDE, speed up your development time, and code with higher confidence.

There are contribution guidelines in the Github repo if you wish to improve the package. There are plenty of more advanced examples highlighted in the documentation and within the package contents in the Examples folder.

Extra Resources

[tm_devices · PyPI – Package driver download and information](#)

[tm_devices Github – Source code, issue tracking, contribution](#)

[tm_devices Github – Online Documentation](#)

Troubleshooting

Upgrading pip is usually a good first step to troubleshooting:

In your terminal type: `Python.exe -m pip install -upgrade pip`

Error: whl looks like a filename, but file does not exist OR .whl is not a supported wheel on this platform.

```
(.venv) PS C:\pythonProject> python -m pip install pythonProject/tm_devices/tm_devices-0.1.0-py3-non-any.whl
WARNING: Requirement 'pythonProject/tm_devices/tm_devices-0.1.0-py3-non-any.whl' looks like a filename, but the file does not exist
ERROR: tm_devices-0.1.0-py3-non-any.whl is not a supported wheel on this platform.
(.venv) PS C:\pythonProject>
```

Solution: Pip installing wheel so that it recognizes the file format.

In your terminal type: `pip install wheel`

If you are needing to install wheel offline you can follow similar instructions as Appendix A, but it requires the tar.gz download instead of the .whl file.

Appendix A – Offline Installation of `tm_devices`

1. On a computer with internet, download the package along with all dependencies to the specified path location using:
`pip download --dest <path to destination> wheel setuptools tm_devices`
2. Copy the files to your computer that does not have internet access
3. Then, follow the instructions from the main guide for whichever IDE you are using but swap the install command for the following:
`pip install --no-index --find-links <new path of files> tm_devices`

Contact Information:

Australia 1 800 709 465
Austria* 00800 2255 4835
Balkans, Israel, South Africa and other ISE Countries +41 52 675 3777
Belgium* 00800 2255 4835
Brazil +55 (11) 3530-8901
Canada 1 800 833 9200
Central East Europe / Baltics +41 52 675 3777
Central Europe / Greece +41 52 675 3777
Denmark +45 80 88 1401
Finland +41 52 675 3777
France* 00800 2255 4835
Germany* 00800 2255 4835
Hong Kong 400 820 5835
India 000 800 650 1835
Indonesia 007 803 601 5249
Italy 00800 2255 4835
Japan 81 (3) 6714 3086
Luxembourg +41 52 675 3777
Malaysia 1 800 22 55835
Mexico, Central/South America and Caribbean 52 (55) 88 69 35 25
Middle East, Asia, and North Africa +41 52 675 3777
The Netherlands* 00800 2255 4835
New Zealand 0800 800 238
Norway 800 16098
People's Republic of China 400 820 5835
Philippines 1 800 1601 0077
Poland +41 52 675 3777
Portugal 80 08 12370
Republic of Korea +82 2 565 1455
Russia / CIS +7 (495) 6647564
Singapore 800 6011 473
South Africa +41 52 675 3777
Spain* 00800 2255 4835
Sweden* 00800 2255 4835
Switzerland* 00800 2255 4835
Taiwan 886 (2) 2656 6688
Thailand 1 800 011 931
United Kingdom / Ireland* 00800 2255 4835
USA 1 800 833 9200
Vietnam 12060128

* European toll-free number. If not accessible, call: +41 52 675 3777

Rev. 02.2022


Find more valuable resources at [TEK.COM](https://www.tek.com)

Copyright © Tektronix. All rights reserved. Tektronix products are covered by U.S. and foreign patents, issued and pending. Information in this publication supersedes that in all previously published material. Specification and




price change privileges reserved. TEKTRONIX and TEK are registered trademarks of Tektronix, Inc. All other trade names referenced are the service marks, trademarks or registered trademarks of their respective companies.
052124 SBG 46W-74037-1



Documents / Resources

	<p>Tektronix Simplifying Test Automation With tm_Devices And Python [pdf] User Guide 48W-73878-1, Simplifying Test Automation With tm_Devices And Python, Test Automation With tm_Devices And Python, Automation With tm_Devices And Python, tm_Devices And Python, Devices And Python, Python</p>
---	--

References

-  [Test and Measurement Equipment | Tektronix](#)
-  [Test and Measurement Equipment | Tektronix](#)
-  [PyCharm: the Python IDE for data science and web development](#)
- [User Manual](#)

[Manuals+](#), [Privacy Policy](#)

This website is an independent publication and is neither affiliated with nor endorsed by any of the trademark owners. The "Bluetooth®" word mark and logos are registered trademarks owned by Bluetooth SIG, Inc. The "Wi-Fi®" word mark and logos are registered trademarks owned by the Wi-Fi Alliance. Any use of these marks on this website does not imply any affiliation with or endorsement.