Tektronix®

# Simplifying Test Automation with tm_devices for Python

HOW-TO GUIDE

```
from tm_devices import DeviceManager
from tm_devices.drivers import MSO6B
from tm_devices.helpers import PYVISA_PY_BACKEND


with DeviceManager(verbose=True) as device_manager:
    # Disabling resetting the devices when connecting
    device_manager.setup_cleanup_enabled = False
    # Disabling to see the output after the execution of program.
    device_manager.teardown_cleanup_enabled
    # Establishing Connection
    device_manager.visa_library = PYVISA_PY
    
    # Instantiate the model as an MSO6B
    scope :MSO6B = device_manager.add_scope
    print(scope.commands.idn.query())
```

(property) xincr: WfmoutpreXincr

Return the `WFMOutpre:XINcr` command.

**Description:**

- This query-only command returns the horizontal point spacing in units of `WFMOutpre:XUNit` for the waveform specified by the `DATA:SOURCE` command. This value corresponds to the sampling interval. An error is reported if the `DATa:SOUrce` waveform does not exist.
- Using the `.verify(value)` method will send the `WFMOutpre:XI`… query and raise an AssertionError if the returned value does not match `value`.

```
    acq_record = scope.commands.horizontal.recordlength.query()
    scope.commands.data.stop.write("{}".format(acq_record))
    xincr = float(scope.commands.wfmoutpre.xincr.query())
    ymult = float(scope.commands.wfmoutpre.ymult.query())
    yoff = float(scope.commands.wfmoutpre.yoff.query())
    yzero = float(scope.commands.wfmoutpre.yzero.query())
    Hscope = float(scope.commands.horizontal.scale.query())
    HDelay = float(scope.commands.horizontal.delay.time.query())
    HPos = float(scope.commands.horizontal.position.query())

    # Setting up Channel 1 as data source
    scope.commands.data.source.write("CH1")
    scope.commands.data.width.write("1")
    scope.commands.data.encdg.write("RIBinary")

    # Generate afg signal to channel 1
    # scope.set_and_check(":OUTPUT1:STATE", "CH1")
    scope.commands.afg.frequency.write(10e6)
    scope.commands.afg.offset.write(0.2)
    scope.commands.afg.square.duty.write(25)
    scope.commands.afg.function.write("SQUARE")
    scope.commands.afg.output.load.impedance.write("FIFTY")
    scope.commands.ch[1].scale.write(0.5, verify=True)
    scope.commands.afg.output.state.write(1)
    scope.commands.esr.query()
    scope.commands.acquire.stopafter.write("SEQUENCE")

    # Save the setup
    scope.commands.save.session.write("saveSession.tss")
    # Save the waveform produced on channel 1
    scope.commands.save.waveform.write("CH1, "ch2afg.wfm"")

    # recalling the saved sessions and waveforms
    scope.recall_session("saveSession.tss")
    # recall ch2afg as REF1
    scope.recall_reference("ch2afg.wfm", 1)

    ###SCREENSHOT###
    # configure scope for screenshot
    scope.commands.save.image.write("C:\\screenshots\\tek.png")
    # transfer the screenshot to pc
    scope.commands.filesystem.readfile.write("C:\\screenshots\\tek.png")
    imageData = scope.read_raw()
    file = open("C:\screenshots\MSO64Screenshot.png", "wb")
    file.write(imageData)
    file.close()
```

## Table of Contents

## Introduction

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 technical brief 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.

## What Is the tm_devices Package?

Tektronix's tm_devices is a device management package developed by Tektronix that allows users to control and automate tests on Tektronix and Keithley products with the programming language Python. It is easily installed using pip, Python's package-management system. This package includes a multitude of commands and functions to help users easily automate tests on Tektronix and Keithley products. The package 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.

## Setting up your Environment

This section will guide you through the prerequisites and installations to prepare you to do development work with tm_devices. We made a conscious choice to include instructions that support virtual environments in Python (venvs) because we believe it makes your projects easier to manage and maintain, especially if you are just trying this package out before committing to its usage.

### Installation and Prerequisites Overview

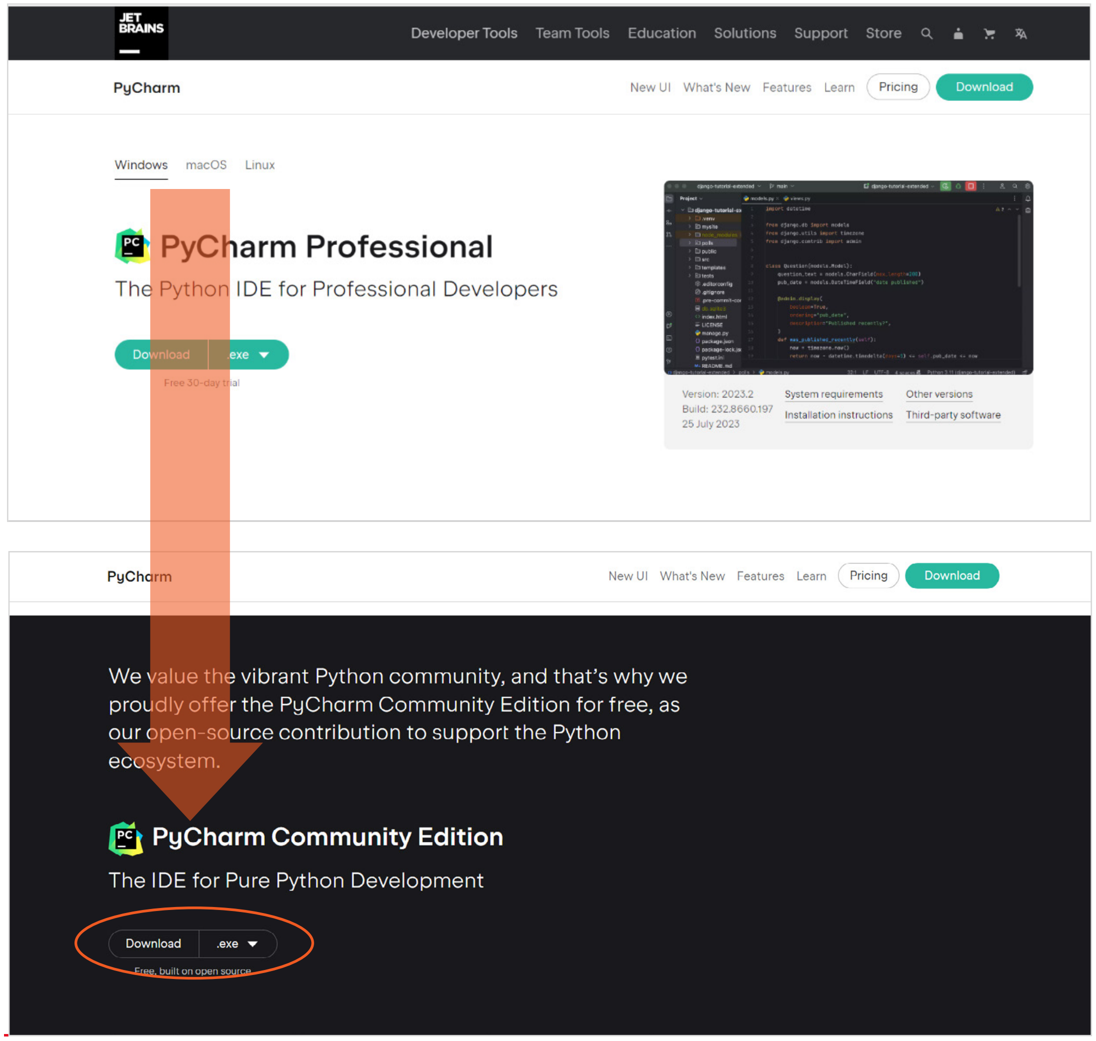
1. Install Python.
   a. Python >=3.8
2. PyCharm – PyCharm Installation, Starting a project, and tm_devices installation
3. VSCode – VSCode Installation, Starting a project, and tm_devices installation

## PyCharm Community (free) edition

PyCharm is a 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.

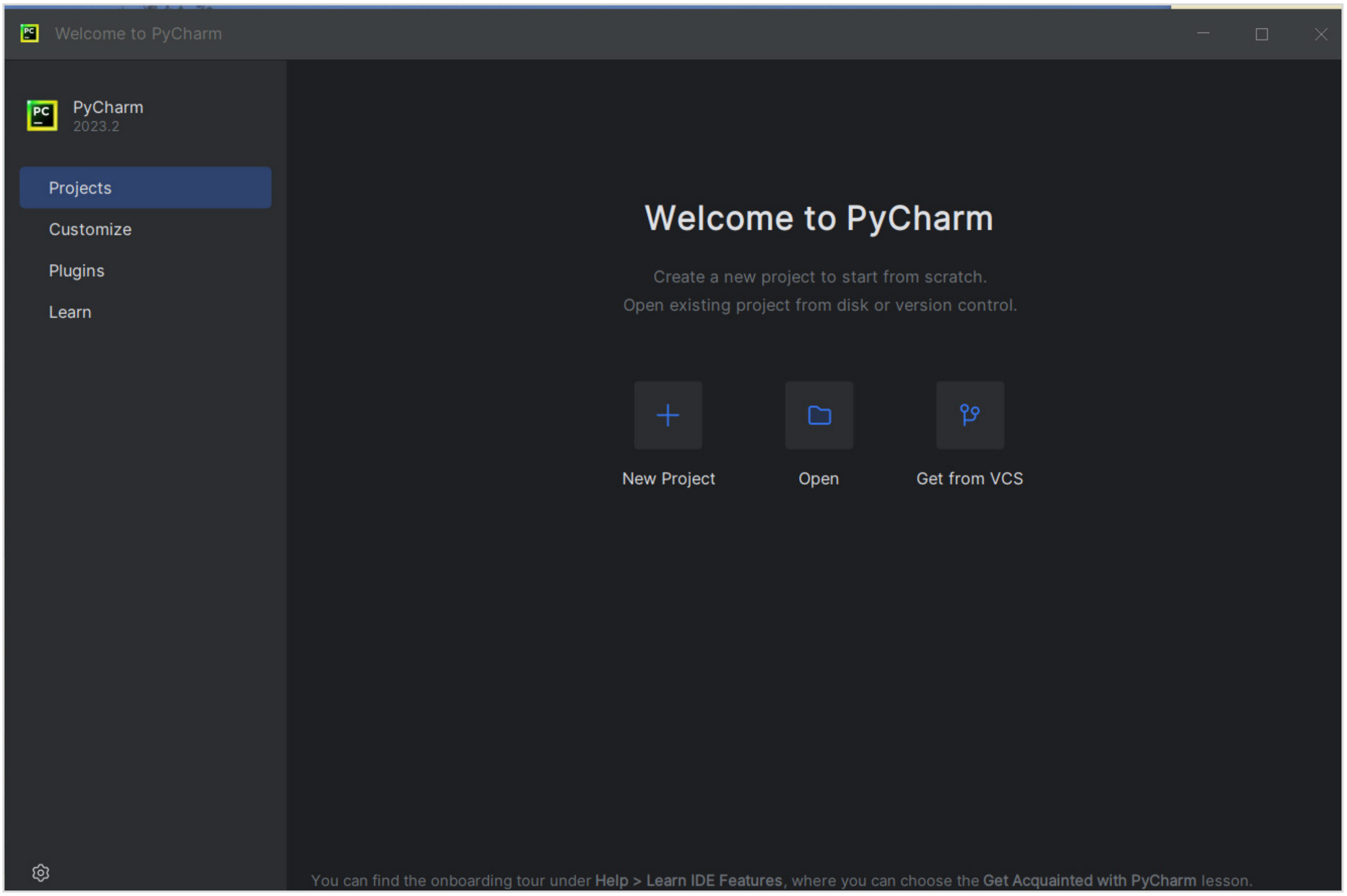
We will walk through the installation PyCharm community edition (free), followed by installing tm_devices in the IDE and setting up a virtual environment to develop in.

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!



Welcome to PyCharm

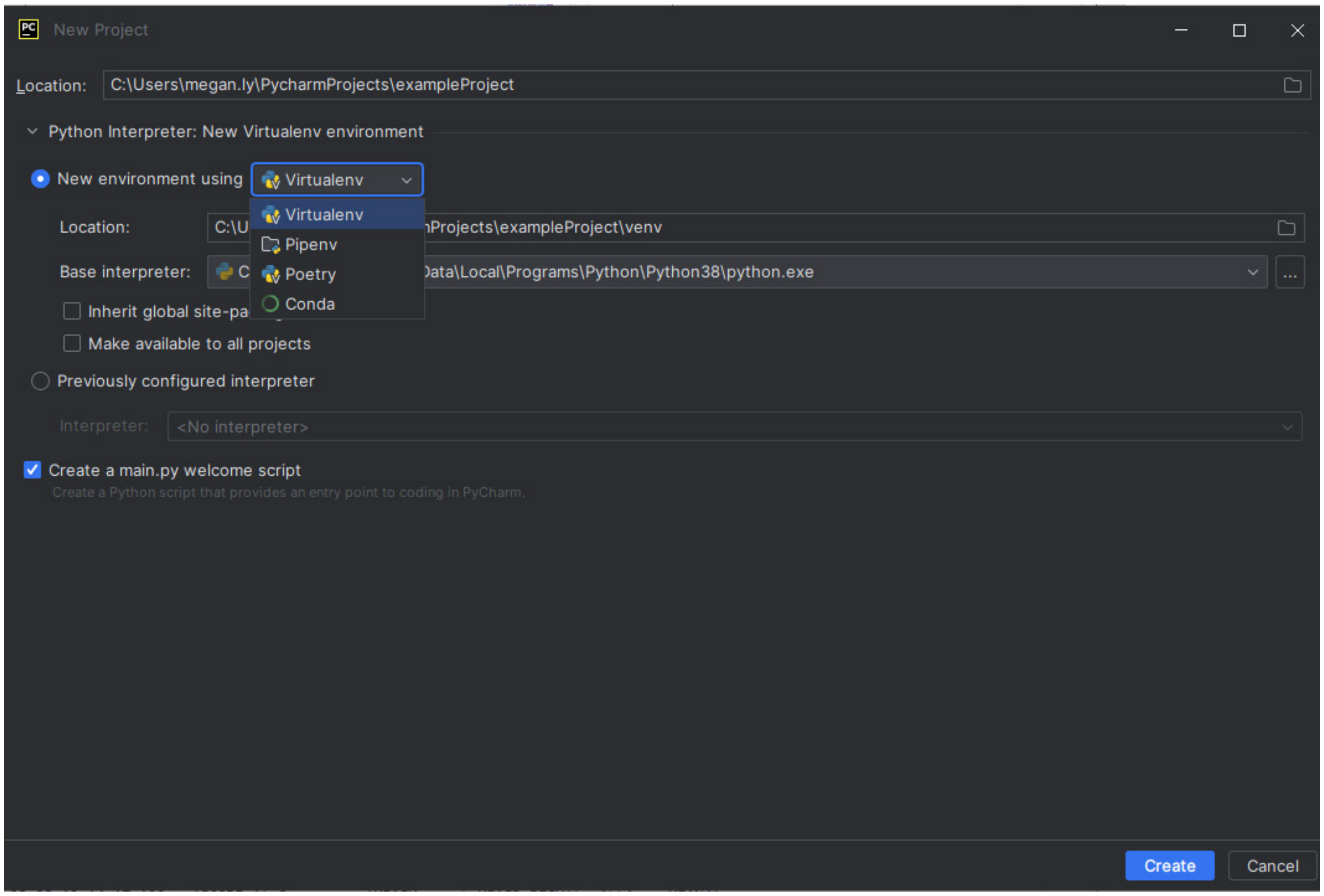
Create a new project to start from scratch.
Open existing project from disk or version control.

You can find the onboarding tour under **Help > Learn IDE Features**, where you can choose the **Get Acquainted with PyCharm** lesson.

**Creating a new project + setting up virtual environment in PyCharm**

5. Click "New Project".

6. Confirm path for project, make sure "Virtualenv" is selected.



Location: C:\Users\megan.ly\PycharmProjects\exampleProject

Python Interpreter: New Virtualenv environment

New environment using: Virtualenv

Location: C:\U…Projects\exampleProject\venv

Base interpreter: C…Data\Local\Programs\Python\Python38\python.exe

Inherit global site-pa…

Make available to all projects

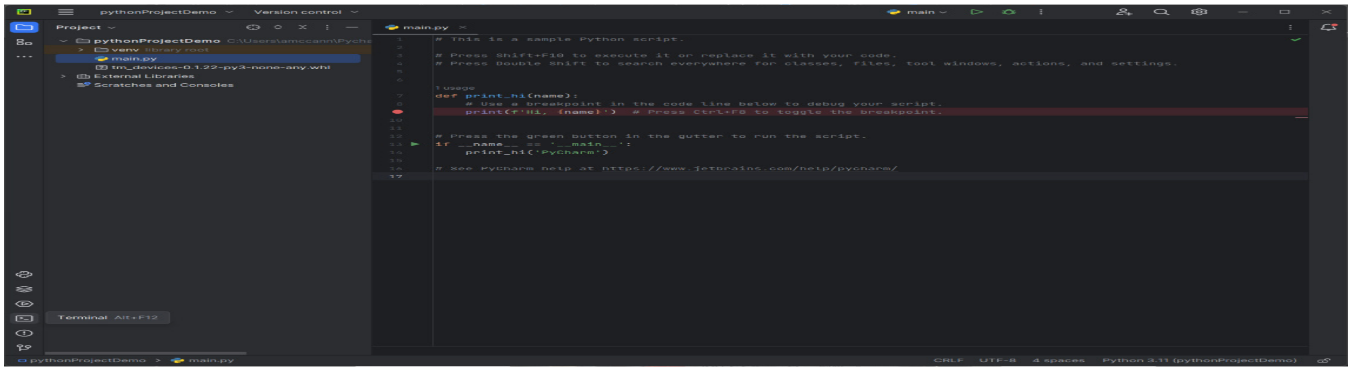
Previously configured interpreter

Interpreter: <No interpreter>

Create a main.py welcome script

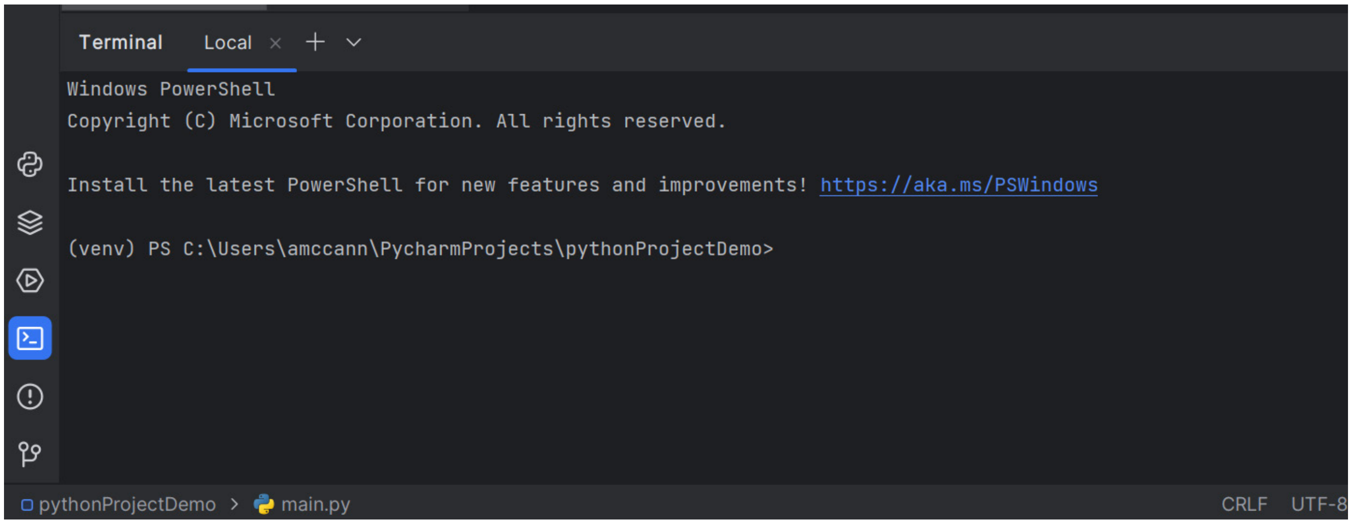
Create a Python script that provides an entry point to coding in PyCharm.

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



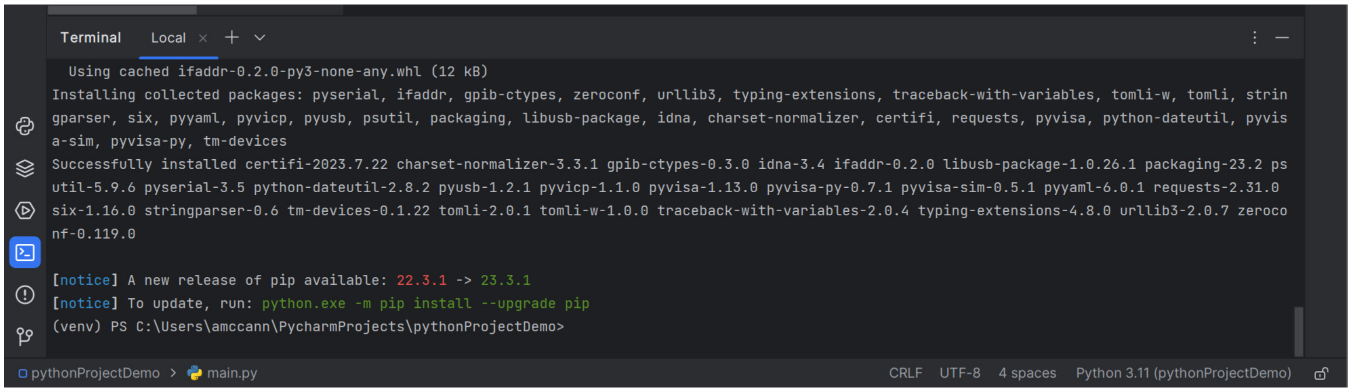
```
# This is a sample Python script.

# Press Shift+F10 to execute it or replace it with your code.
# Press Double Shift to search everywhere for classes, files, tool windows, actions, and settings.


def print_hi(name):
    # Use a breakpoint in the code line below to debug your script.
    print(f'Hi, {name}')  # Press Ctrl+F8 to toggle the breakpoint.


# Press the green button in the gutter to run the script.
if __name__ == '__main__':
    print_hi('PyCharm')

# See PyCharm help at https://www.jetbrains.com/help/pycharm/
```

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



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

(venv) PS C:\Users\amccann\PycharmProjects\pythonProjectDemo>
```

9. Install driver from the terminal.
   Type: **`pip install tm _ devices`**


Terminal
Local
pythonProjectDemo > main.py
CRLF UTF-8 4 spaces Python 3.11 (pythonProjectDemo)

```
  Using cached ifaddr-0.2.0-py3-none-any.whl (12 kB)
Installing collected packages: pyserial, ifaddr, gpib-ctypes, zeroconf, urllib3, typing-extensions, traceback-with-variables, tomli-w, tomli, strin
gparser, six, pyyaml, pyvicp, pyusb, psutil, packaging, libusb-package, idna, charset-normalizer, certifi, requests, pyvisa, python-dateutil, pyvis
a-sim, pyvisa-py, tm-devices
Successfully installed certifi-2023.7.22 charset-normalizer-3.3.1 gpib-ctypes-0.3.0 idna-3.4 ifaddr-0.2.0 libusb-package-1.0.26.1 packaging-23.2 ps
util-5.9.6 pyserial-3.5 python-dateutil-2.8.2 pyusb-1.2.1 pyvicp-1.1.0 pyvisa-1.13.0 pyvisa-py-0.7.1 pyvisa-sim-0.5.1 pyyaml-6.0.1 requests-2.31.0
six-1.16.0 stringparser-0.6 tm-devices-0.1.22 tomli-2.0.1 tomli-w-1.0.0 traceback-with-variables-2.0.4 typing-extensions-4.8.0 urllib3-2.0.7 zeroco
nf-0.119.0

[notice] A new release of pip available: 22.3.1 -> 23.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip
(venv) PS C:\Users\amccann\PycharmProjects\pythonProjectDemo>
```

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
    ch1pk2pk = float(scope.commands.measurement.meas[1].results.allacqs.mean.query())
    # Printing the value onto the console
    print(f'Channel 1 pk2pk: {ch1pk2pk}')
```

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

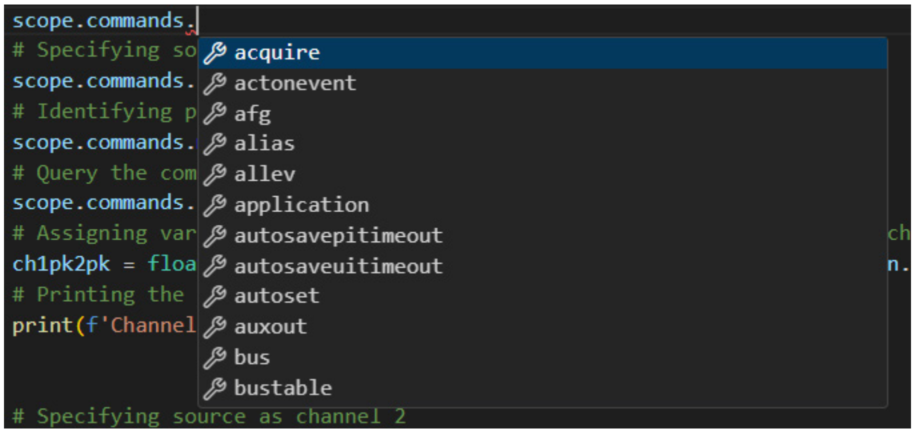
IntelliSense – Microsoft's name for Code Completion is a very powerful feature of IDEs we have tried to exploit as much as possible.

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) command structure so if you are familiar with SCPI you will be familiar with these.

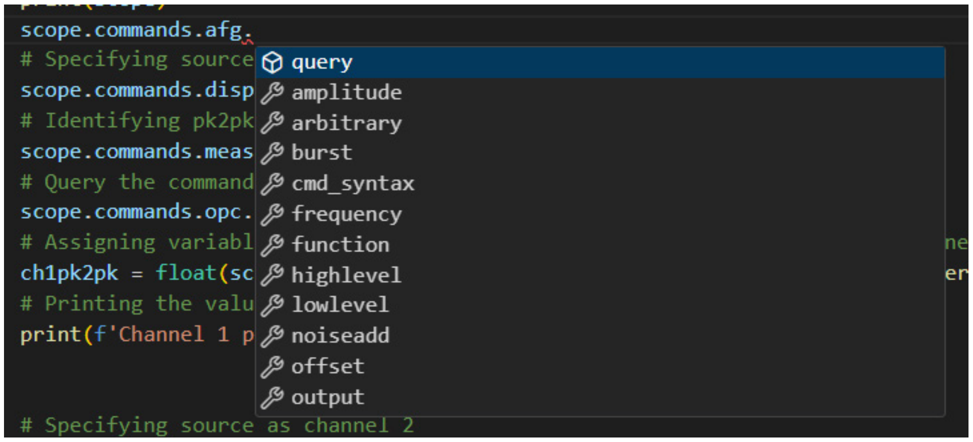
The following is an example of how IntelliSense shows all the commands available with the previously typed command.

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



```
scope.commands.
# Specifying so
scope.commands.
# Identifying p
scope.commands.
# Query the com
scope.commands.
# Assigning var                                                          ch
ch1pk2pk = floa                                                          n.
# Printing the
print(f'Channel

# Specifying source as channel 2
```

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



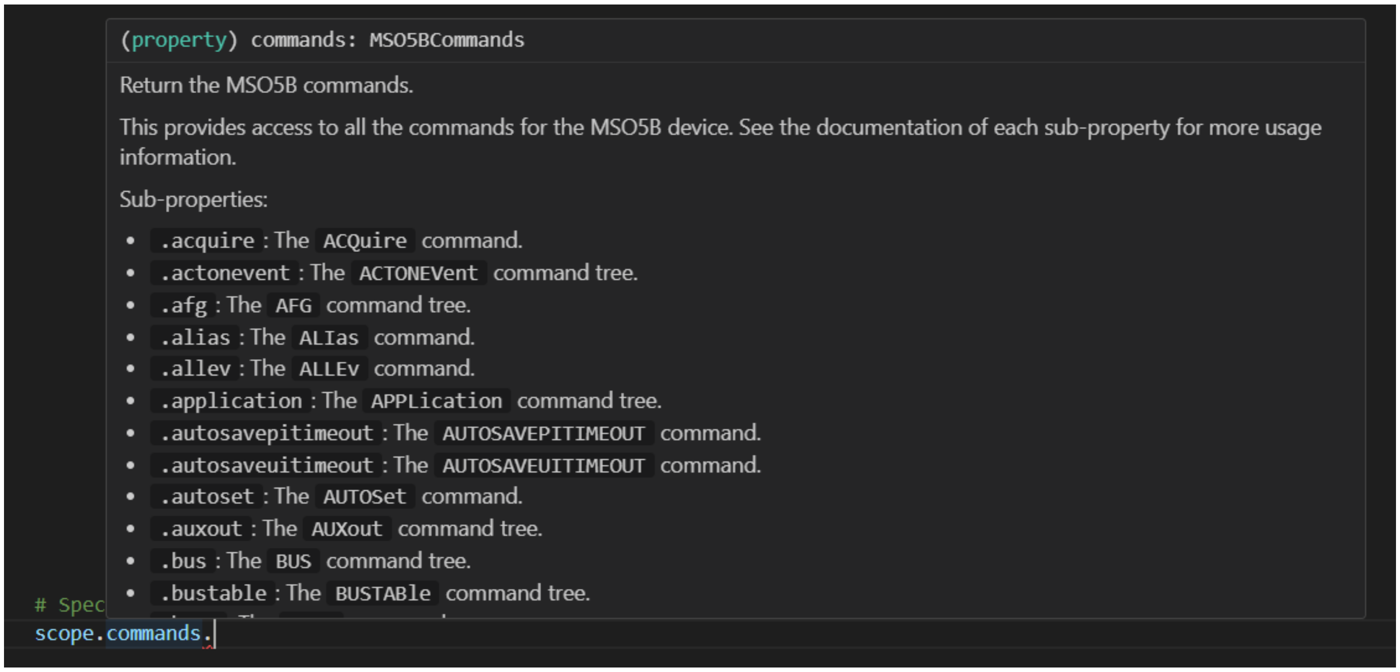
```
scope.commands.afg.
# Specifying source
scope.commands.disp
# Identifying pk2pk
scope.commands.meas
# Query the command
scope.commands.opc.
# Assigning variabl
ch1pk2pk = float(sc
# Printing the valu
print(f'Channel 1 p


# 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.

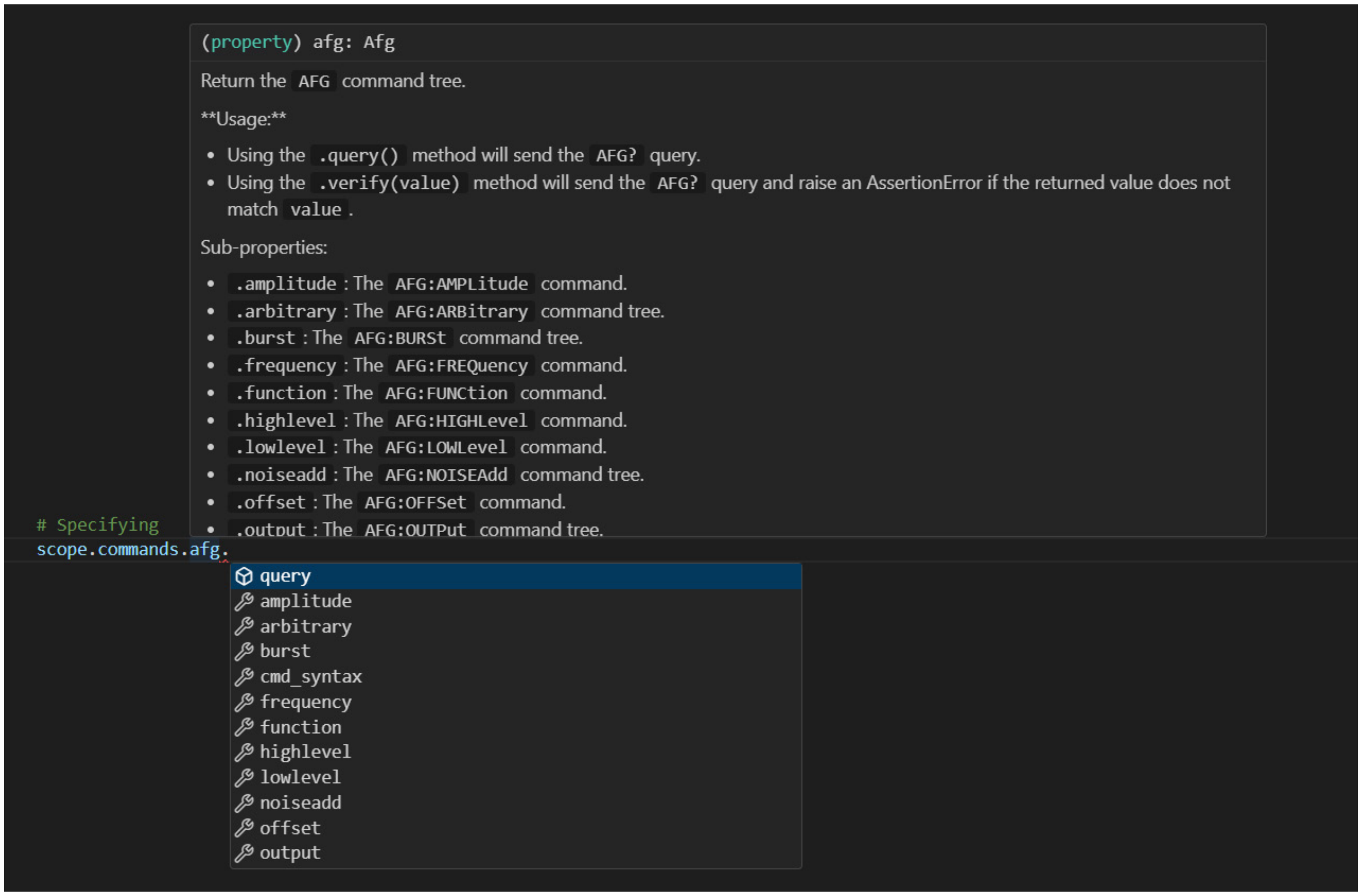
This provides access to all the commands for the MSO5B device. See the documentation of each sub-property for more usage information.

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.
- `.autosavepitimeout` : The `AUTOSAVEPITIMEOUT` 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.

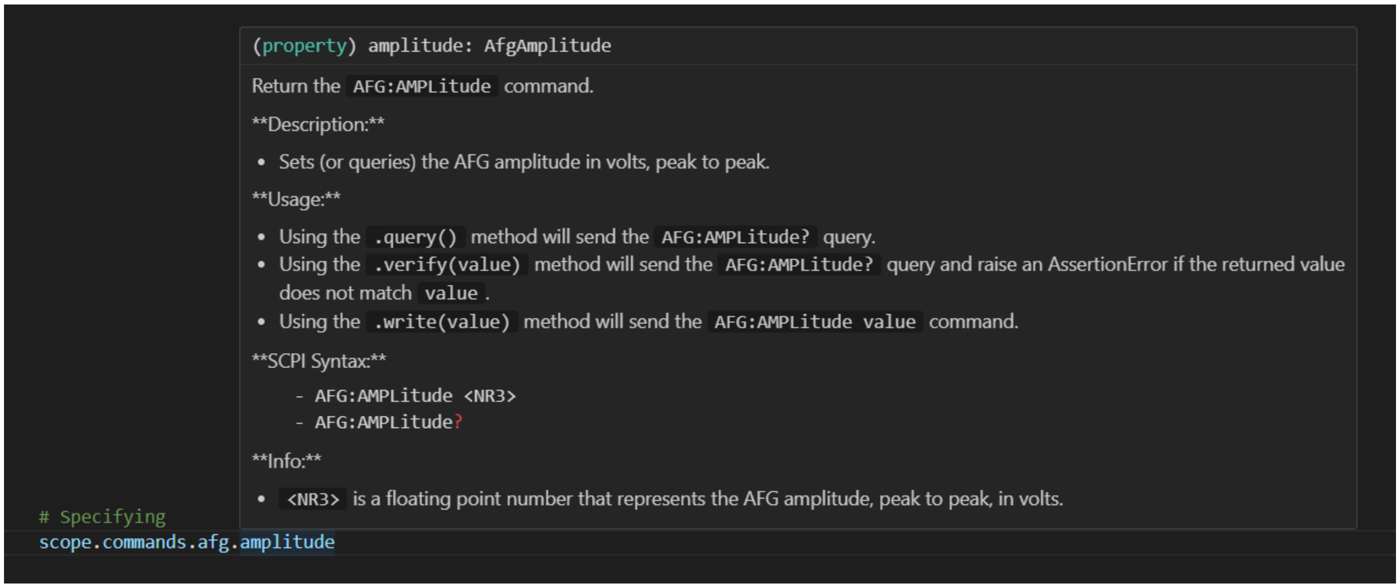
**Usage:**

- 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:HIGHLevel` 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.
```



```
(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

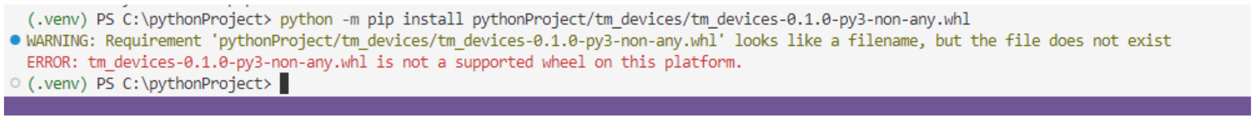
https://github.com/tektronix/tm_devices#documentation – Online Documentation

## 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.

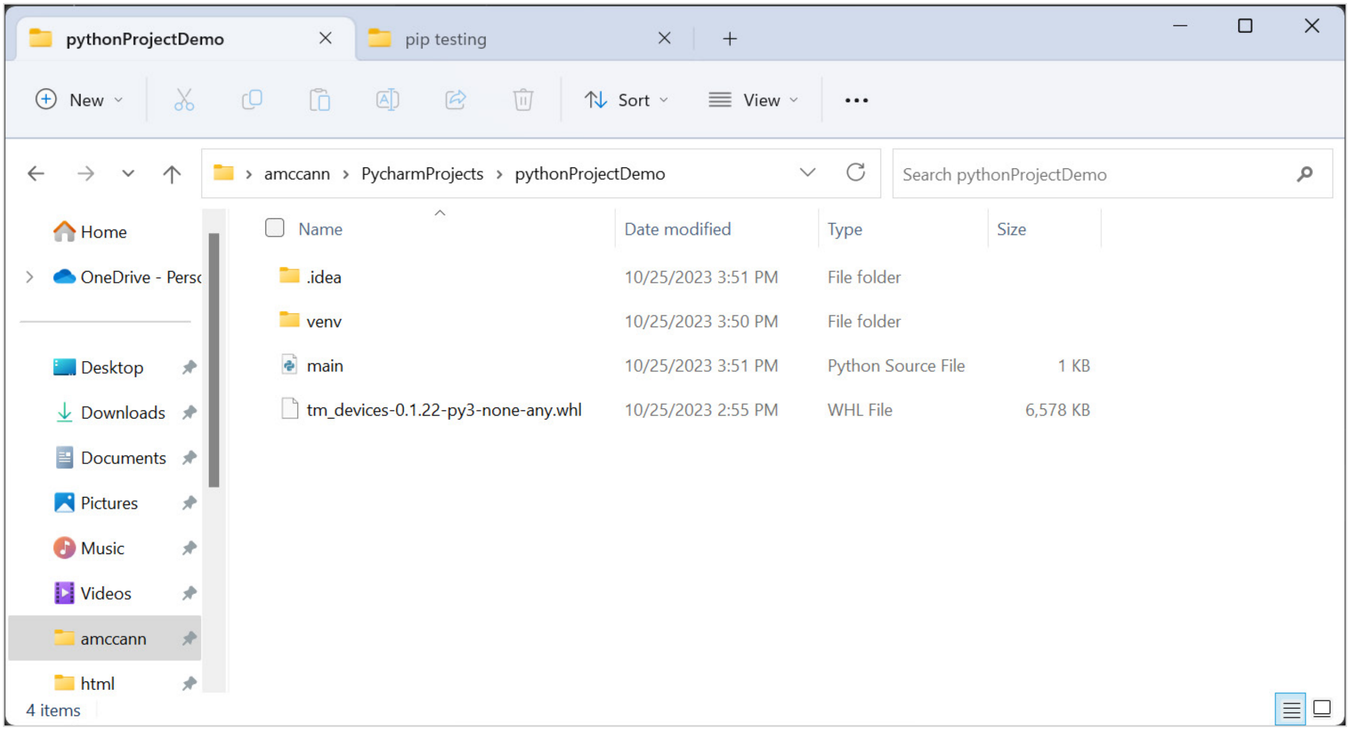
In your terminal type: `pip install wheel`

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

### Local Installation in PyCharm

11. On a computer with internet access, download the latest tm_devices package here:
    a. tm-devices · PyPI
12. Move the .whl file to your project folder.



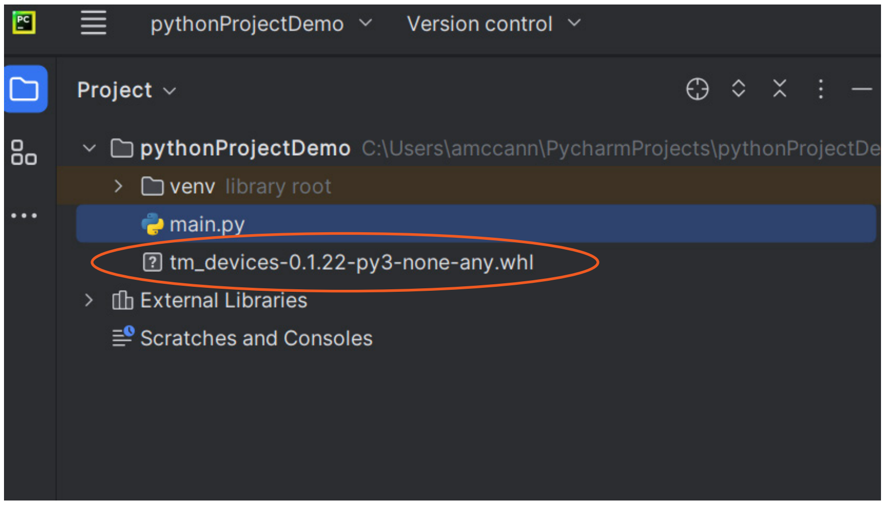
pythonProjectDemo | pip testing

amccann › PycharmProjects › pythonProjectDemo

| Name | Date modified | Type | Size |
|---|---|---|---|
| .idea | 10/25/2023 3:51 PM | File folder | |
| venv | 10/25/2023 3:50 PM | File folder | |
| main | 10/25/2023 3:51 PM | Python Source File | 1 KB |
| tm_devices-0.1.22-py3-none-any.whl | 10/25/2023 2:55 PM | WHL File | 6,578 KB |

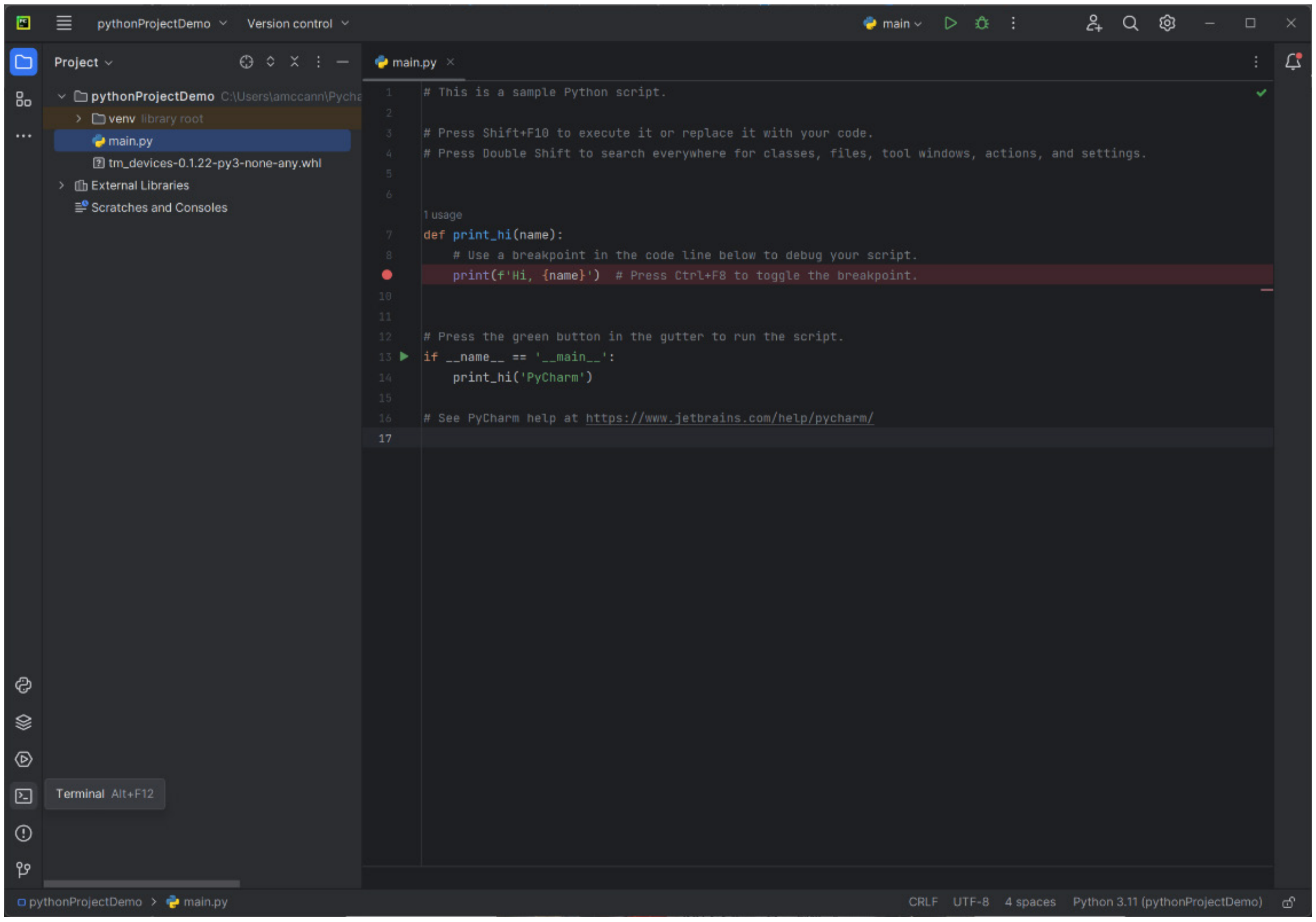
4 items

13. Move back to PyCharm, ensure the .whl file is in the project folder.



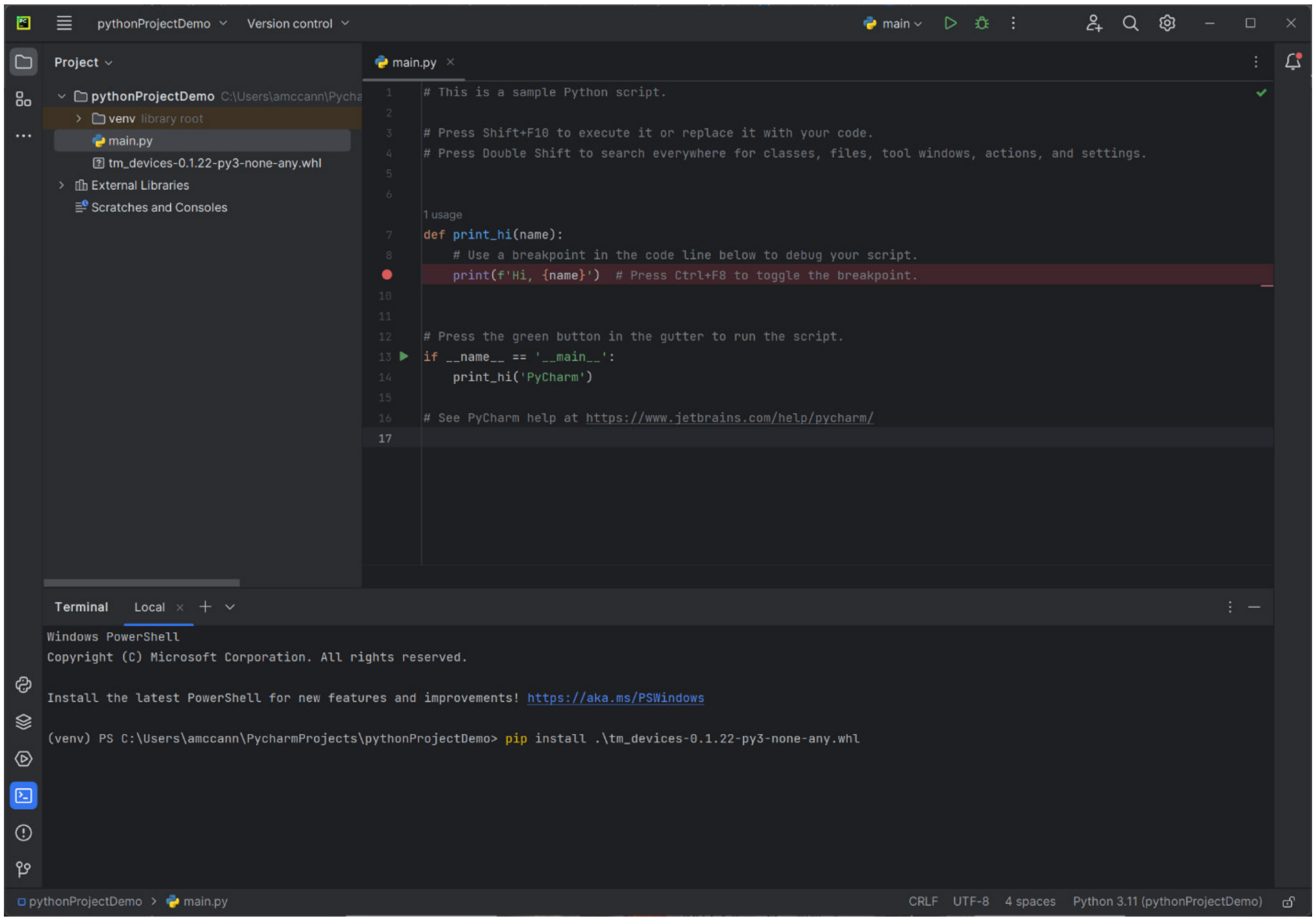
```
pythonProjectDemo  C:\Users\amccann\PycharmProjects\pythonProjectDe
  venv  library root
  main.py
  tm_devices-0.1.22-py3-none-any.whl
External Libraries
Scratches and Consoles
```

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



```
# This is a sample Python script.

# Press Shift+F10 to execute it or replace it with your code.
# Press Double Shift to search everywhere for classes, files, tool windows, actions, and settings.


def print_hi(name):
    # Use a breakpoint in the code line below to debug your script.
    print(f'Hi, {name}')  # Press Ctrl+F8 to toggle the breakpoint.


# Press the green button in the gutter to run the script.
if __name__ == '__main__':
    print_hi('PyCharm')

# See PyCharm help at https://www.jetbrains.com/help/pycharm/
```

15. In your terminal type: `pip install <specific filename and version of tm_devices>.whl`
    It should look similar to: `tm_devices-1.0.0-py3-none-any.whl`


pythonProjectDemo
Version control
main
Project
pythonProjectDemo C:\Users\amccann\Pycha
venv library root
main.py
tm_devices-0.1.22-py3-none-any.whl
External Libraries
Scratches and Consoles
1 usage
Terminal
Local
pythonProjectDemo > main.py
CRLF UTF-8 4 spaces Python 3.11 (pythonProjectDemo)

```python
# This is a sample Python script.

# Press Shift+F10 to execute it or replace it with your code.
# Press Double Shift to search everywhere for classes, files, tool windows, actions, and settings.


def print_hi(name):
    # Use a breakpoint in the code line below to debug your script.
    print(f'Hi, {name}')  # Press Ctrl+F8 to toggle the breakpoint.


# Press the green button in the gutter to run the script.
if __name__ == '__main__':
    print_hi('PyCharm')

# See PyCharm help at https://www.jetbrains.com/help/pycharm/
```

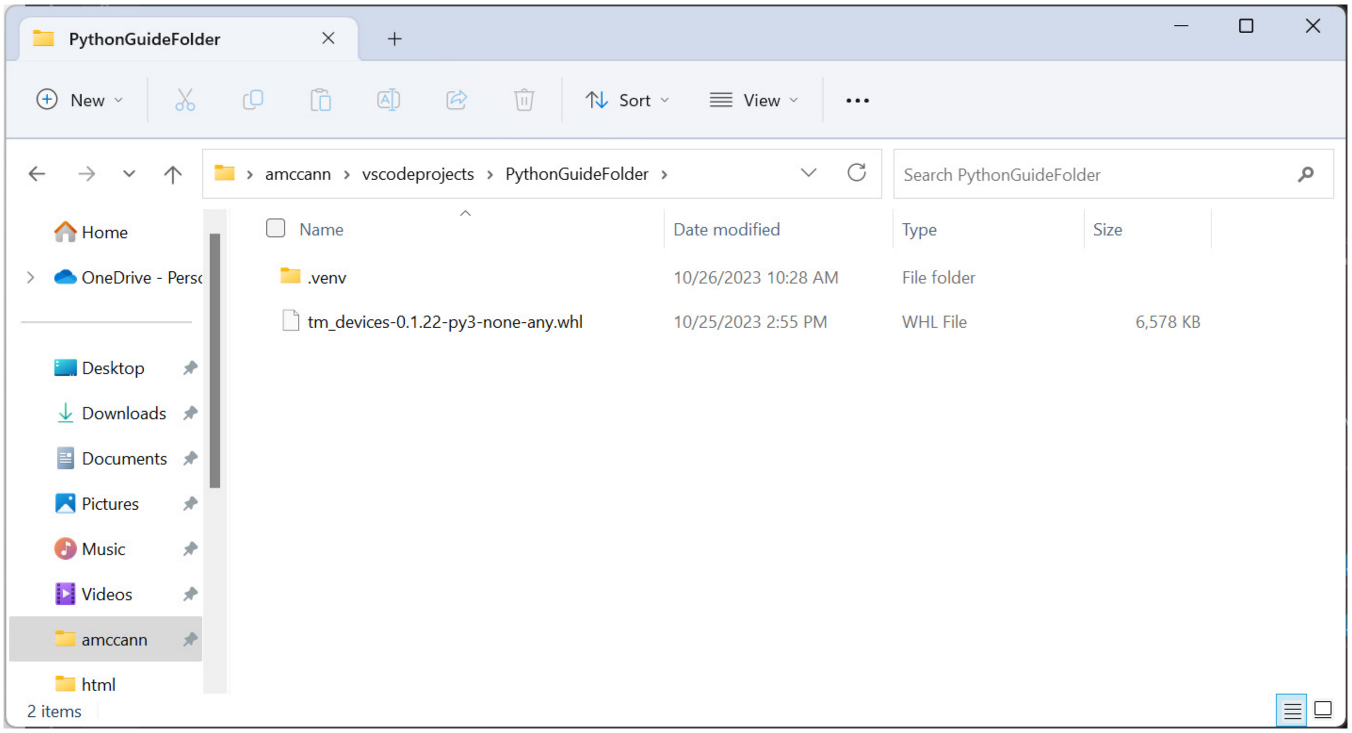
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

(venv) PS C:\Users\amccann\PycharmProjects\pythonProjectDemo> pip install .\tm_devices-0.1.22-py3-none-any.whl
```

## Local installation in Visual Studio Code

In this section we will detail how to install the tm_devices package without a local internet connection.

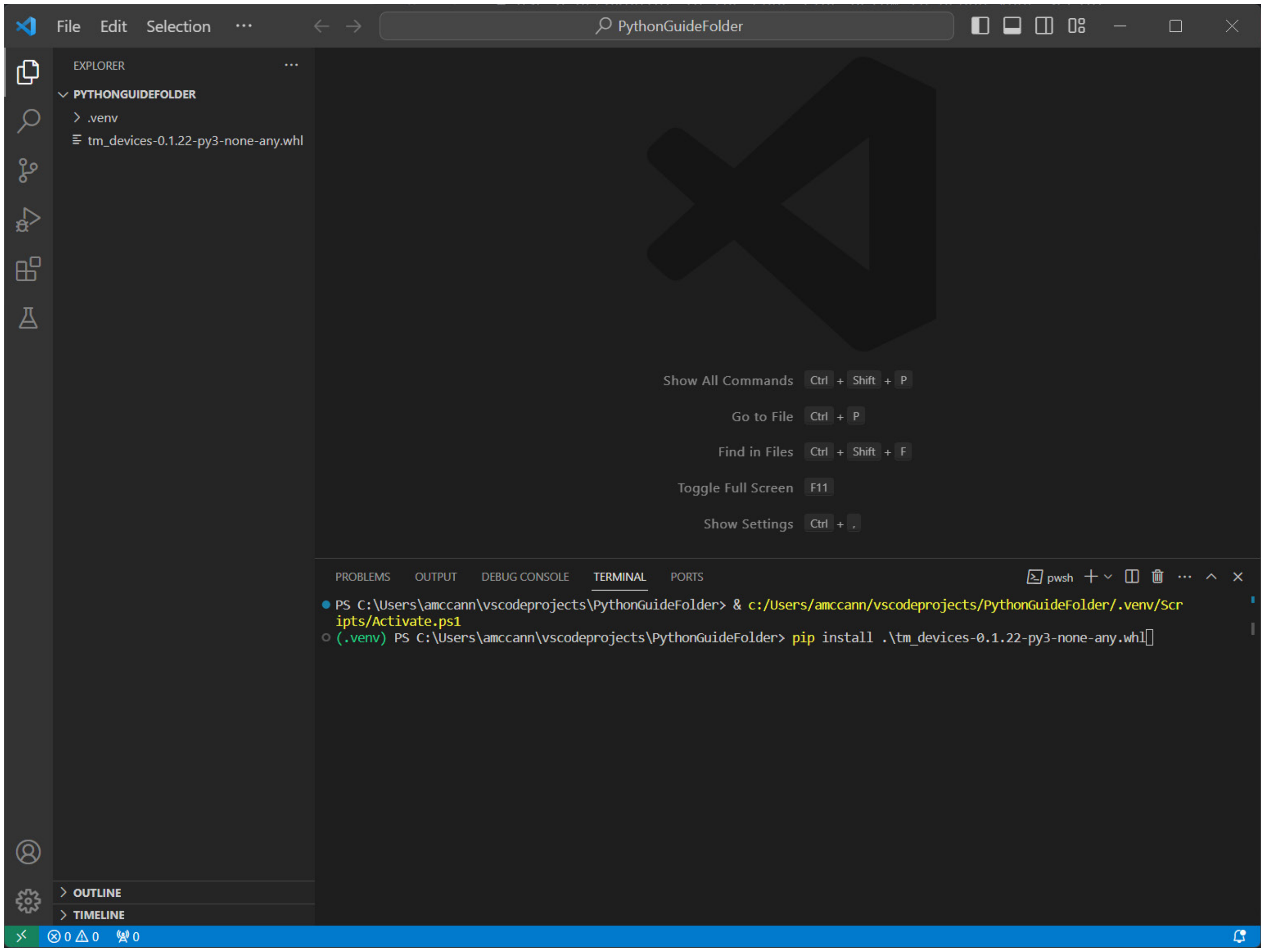
1. On a computer with internet access, download the latest tm_devices package here:
   a. tm-devices · PyPI

2. Move the .whl file to the project folder you are working in.



| Name | Date modified | Type | Size |
|---|---|---|---|
| .venv | 10/26/2023 10:28 AM | File folder | |
| tm_devices-0.1.22-py3-none-any.whl | 10/25/2023 2:55 PM | WHL File | 6,578 KB |

3. Navigate to terminal in Visual Studio Code (Ctrl+Shift+P -> Create New terminal).
   This example is using a virtual environment so it might look different if you are not.



```
PS C:\Users\amccann\vscodeprojects\PythonGuideFolder> & c:/Users/amccann/vscodeprojects/PythonGuideFolder/.venv/Scr
ipts/Activate.ps1
(.venv) PS C:\Users\amccann\vscodeprojects\PythonGuideFolder> pip install .\tm_devices-0.1.22-py3-none-any.whl
```

4. Make sure that the file is in your working directory and type:
   **`pip install <specific filename and version of tm_devices>.whl`**
   It should look similar to: **`tm_devices-1.0.0-py3-none-any.whl`**
5. Installation successful if tm_devices imports have no error.

## 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