

User Manual

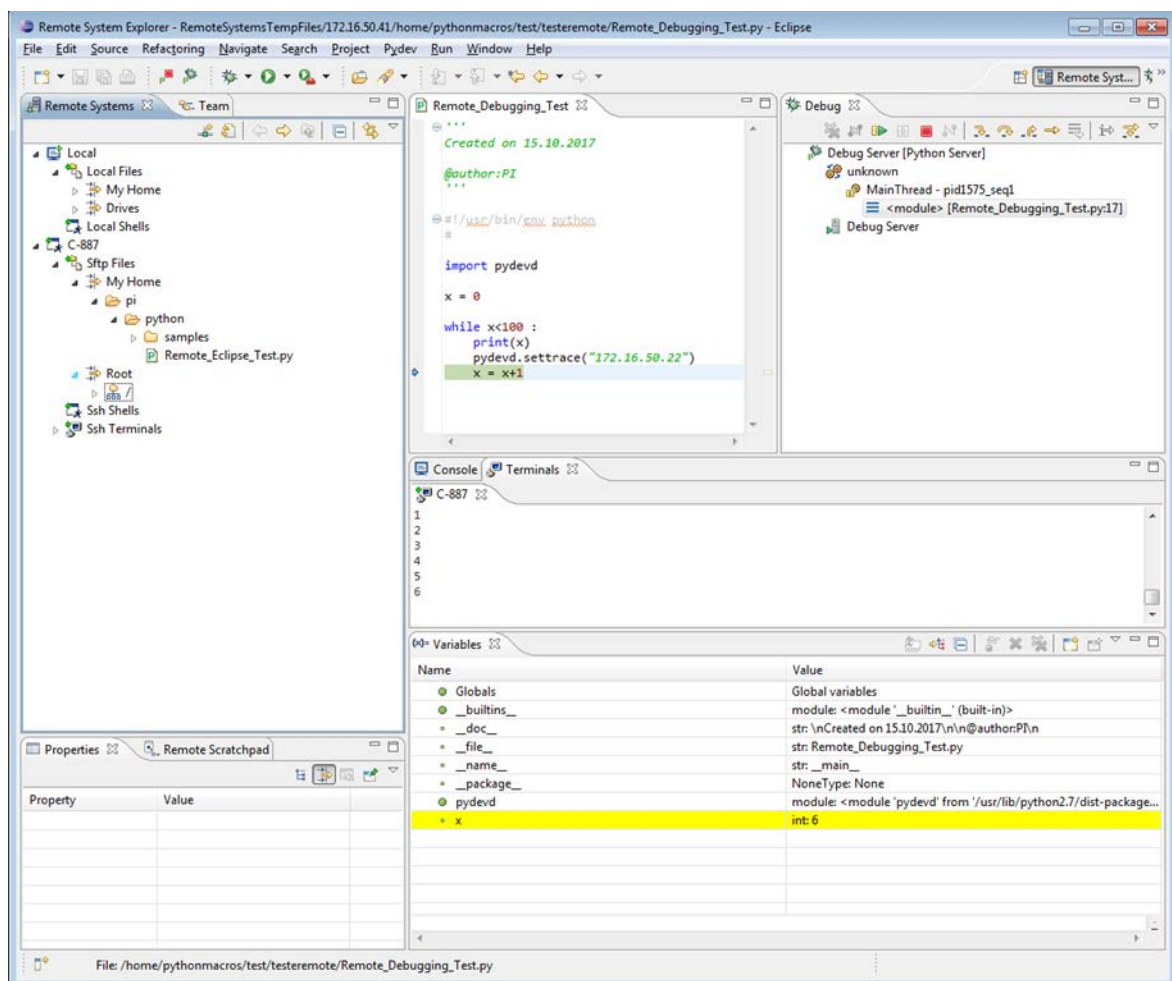
SM157D, applies to PIPython

MFr, ASt, 11/7/2017

PI

PIPython

Programming PI Controllers with Python



Contents

About this Document	3
Symbols and Typographic Conventions.....	3
Notes on Copyright and Trademarks.....	3
Disclaimer	3
PIPython Overview	4
Intended Use	4
Directory Structure on the Controller	4
Login Data for PIPython.....	4
Requirements on the PC	5
Installing PIPython.....	5
Using a Development Environment.....	5
Using Eclipse	6
Installing Eclipse	6
Configuring Eclipse	7
Using Eclipse and PIPython on the PC.....	9
Creating a project.....	9
Creating and Running Scripts	11
Using Eclipse and PIPython on the Controller	12
Configuring the Connection to the Controller	12
Establishing Connection to the Controller	14
Using the Remote Debugger in Eclipse	16
Using PyCharm Professional	18
Installing PyCharm Professional	18
Configuring PyCharm Professional	18
Using PyCharm and PIPython on the PC.....	18
Creating a project.....	18
Creating and Running Scripts	19
Using PyCharm and PIPython on the Controller	21
Configuring the Connection to the Controller for the Remote Interpreter	22
Running Scripts on the Controller	25
Using the Remote Debugger in PyCharm	27
Executing Scripts on the Controller	29
Automatic Execution of Scripts during System Start.....	29
Executing Scripts Manually.....	29

About this Document

This document describes the work with Python on Controllers from PI.

In addition to conventional macros with GCS commands, it is also possible to use Python scripts for automatic control of controllers that are equipped with PIPython.

Symbols and Typographic Conventions

The following symbols and typographic conventions are used in this document:

Symbol	Meaning
1.	Action consisting of several steps whose sequential order must be observed
2.	
➤	Action consisting of one or several steps whose sequential order is irrelevant
▪	Lists
p. 5	Cross-reference to page 5
Start > Settings	Menu path in the PC software (example: to open the menu, the Start and Settings buttons must be clicked in succession)
<code>print</code>	Command line or command in a script
5	Value that must be entered or selected via the PC software

Notes on Copyright and Trademarks

Software components from third-party suppliers can be integrated into or used with software products from PI. Further information is available at [Third Party Software Note](#) on our website.

Also available on our website [General Software License Agreement](#).

Disclaimer

We are not liable for content or links from third-party websites that are used in this document. PI expressly dissociates itself from any externally linked websites, their design, data and contents as well as from any liability connected thereto. The contents of the linked websites falls exclusively within the responsibility of their operators.

PI is not responsible for the accuracy of the links to third-party websites.

PIPython Overview

Intended Use

PIPython is made available by several PI controllers with the Linux operating system. The purpose of PIPython is automatic control of controllers and the connected positioning systems via scripts.

The use of Python as programming language offers a variety of possibilities. PIPython from PI provides a collection of Python modules for convenient use of PI devices and GCS data. These modules can be used with Python 2.7+ and 3.4+ on Windows, Linux, and OS X, as well as via sockets on other operating systems of a PC.

Similar to GCS macros, it is possible to use Python scripts both on the PC and directly on the controller.

PIPython may not be used for purposes other than those stated in this user manual.

Directory Structure on the Controller

The following directories on the controller are relevant for PIPython:

- Home directory: /home/piuser/
- Directory for Python scripts: /home/piuser/python/

There is also a directory for sample scripts: /home/piuser/python/samples/

The "PIPython" module is in the following directory: /home/piuser/python/pipython/

Login Data for PIPython

For the transfer and management of files (scripts) for PIPython the SFTP protocol is used. The following login data is required (pay attention to upper/lower case):

- User name: piuser
- Password: PI2020user!

Requirements on the PC

To use PIPython, Python must be installed on the PC (e.g., download via <https://www.python.org/downloads/>). PIPython must also be installed.

Installing PIPython

- Install PIPython onto the PC.

The module is on the product CD of the controller. Notes on installing are in the "read-me.rst" file.

For PC-based working, it is recommended to install PI_GCS2_DLL.dll as well:

- Install PI_GCS2_DLL.dll onto the PC.

The corresponding setup is on the product CD of the controller.

Using a Development Environment

To make working with PIPython more convenient, it is recommended to use a development environment (IDE, Integrated Development Environment).

In this document, working with PIPython is explained using two widely used IDEs:

- **Eclipse** (p. 6)
as open-source solution
- **PyCharm Professional** (p. 18)
as commercial product
(Note: The free PyCharm Community Edition does not offer the remote debugging option.)

It is possible to develop and run Python scripts easily with both IDEs. It is also possible to transfer the scripts to the controller and debug them from the PC.

Using Eclipse

Installing Eclipse

If you want to use Eclipse for working with PIPython, install the following onto your PC:

1. IDE: **Eclipse** (e.g., download via <https://eclipse.org/downloads/>)

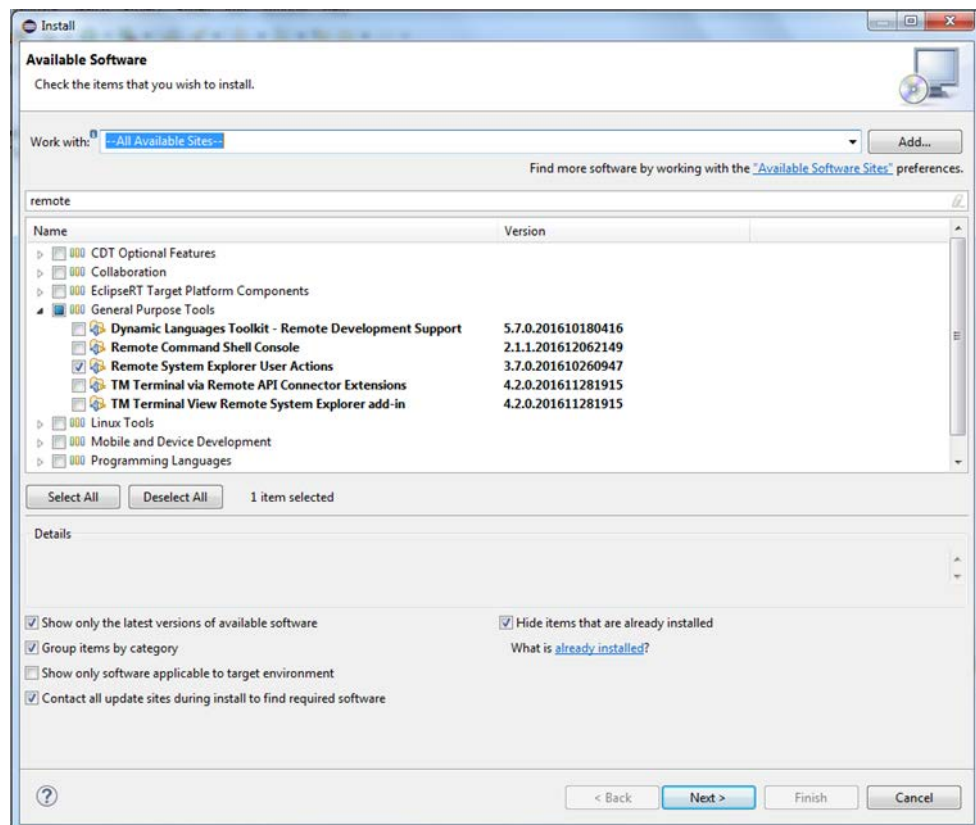
All of the following descriptions and figures refer to the Neon.2 version (4.6.2) in Windows.

2. Plug-in: **PyDev** (e.g., download via <http://www.pydev.org/>)

A detailed description of this open-source plug-in as well as information on installing is on the corresponding website (e.g., http://www.pydev.org/manual_101_install.html).

3. **Remote System Explorer (RSE)** for Eclipse must be installed to take full advantage of PIPython on the controller.

This plug-in is not available directly from the "Eclipse marketplace" but via "Available Software Sites" with the "remote" filter (**Help > Install New Software**).



4. Depending on the version of Eclipse used, it is also recommended to install two further plug-ins: **TM Terminal** and the **TM Terminal View Remote System Explorer add-in**.

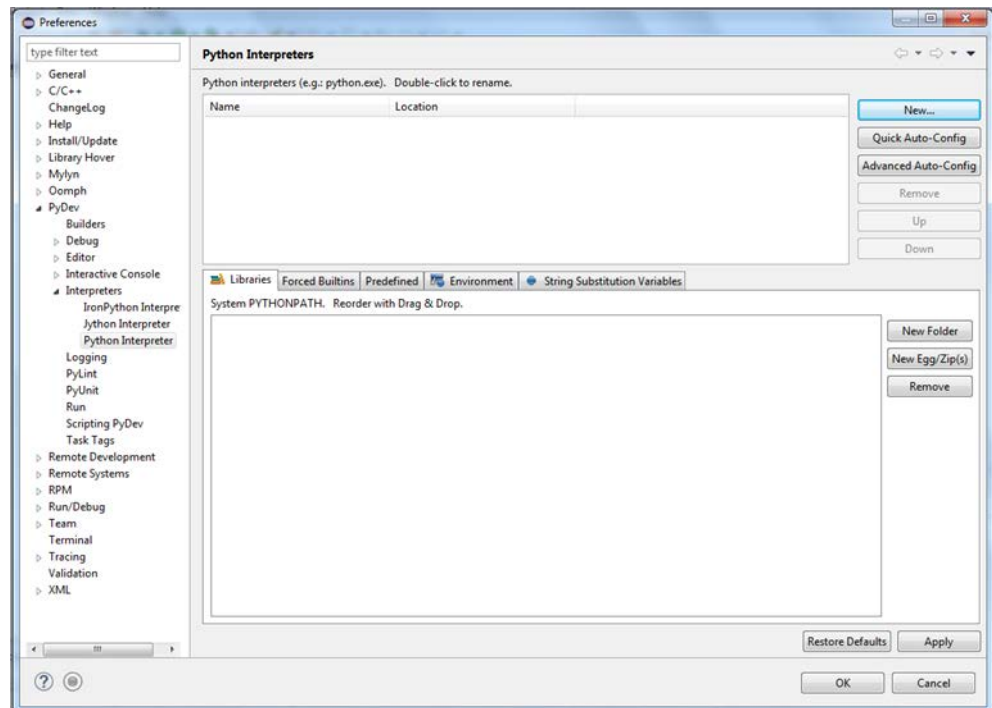
Both can be installed directly from Eclipse.

Eclipse must be restarted after installing.

Configuring Eclipse

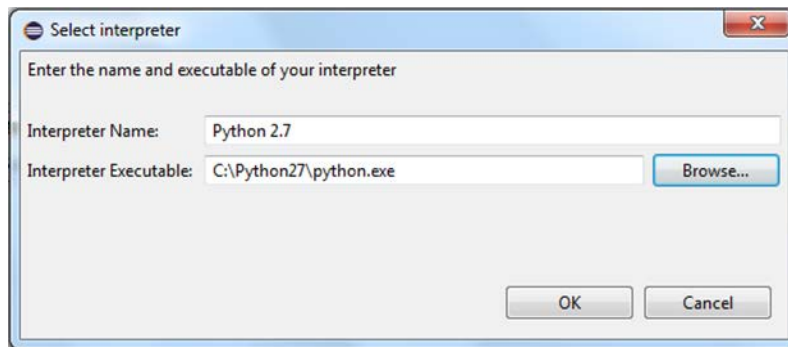
The path to the Python installation must be entered first in Eclipse:

1. Start **Eclipse**.
2. Open the **Preferences** dialog via **Window > Preferences**.
3. In the **Preferences** dialog, click **PyDev > Interpreters > Python Interpreter** in the list on the left-hand side of the window.



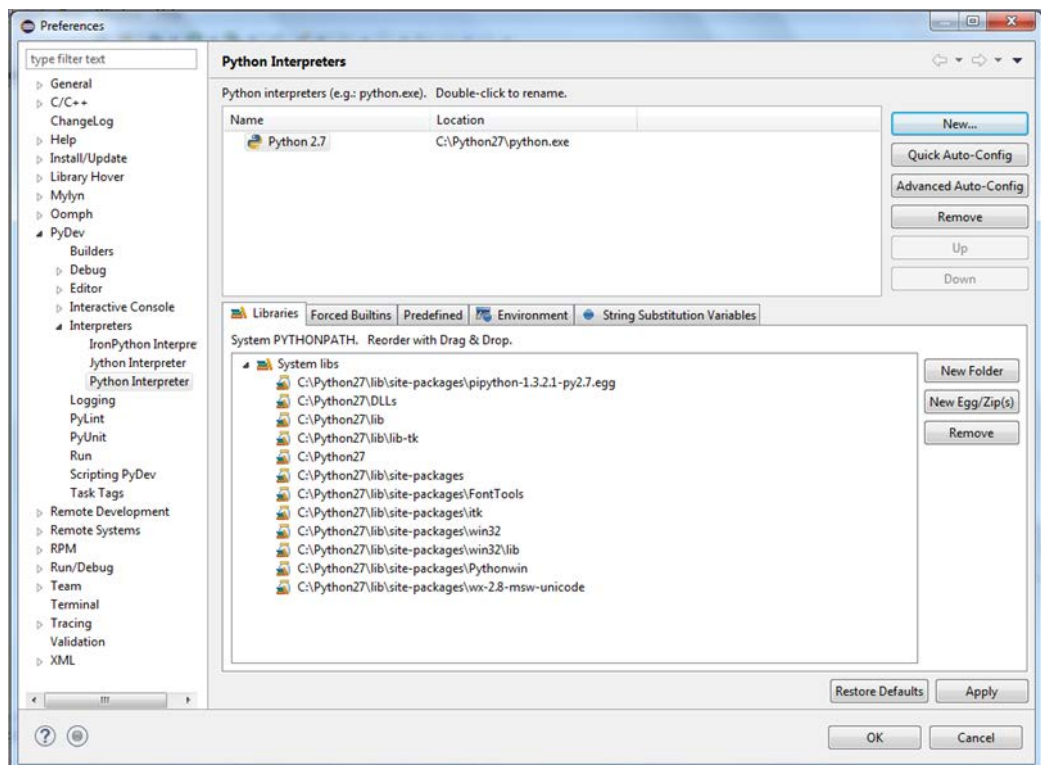
4. Click **New**.

The **Select interpreter** dialog is opened for selecting the interpreter.



5. In the **Select interpreter** dialog, select the Python interpreter that is installed on the PC and click **OK**.

The interpreter is loaded. The result should look as follows:



6. Close the **Preferences** dialog by clicking **OK**.

The Eclipse configuration is completed. You are now in the main window of the program.

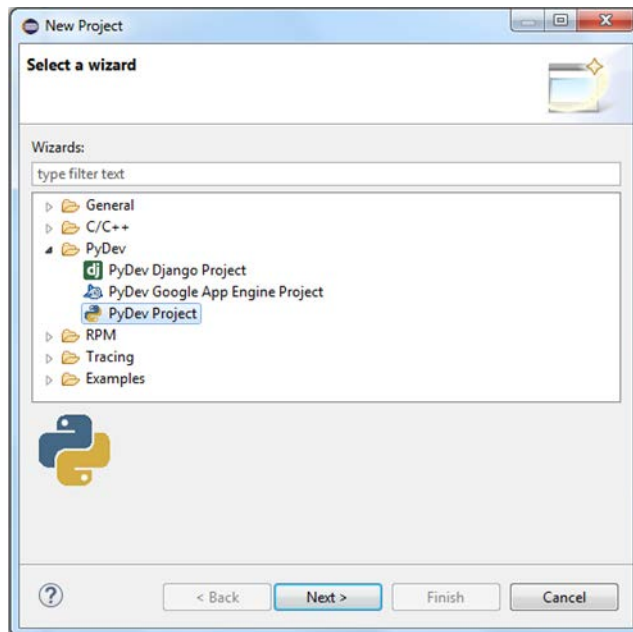
Using Eclipse and PIPython on the PC

Creating a project

The first project can be created after Eclipse has been configured correctly. Proceed as follows:

1. In the main window of the program, select **File > New > Project > PyDev > PyDev Project**.

The **New Project** dialog is opened for selecting the Project Wizard.

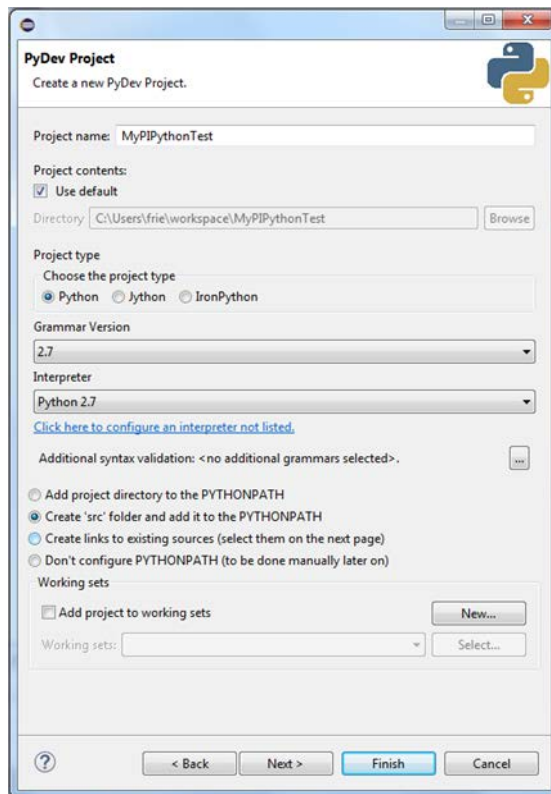


2. Mark **PyDev > PyDevProject** and click **Next**.

The **PyDev Project** dialog is opened for creating the project.

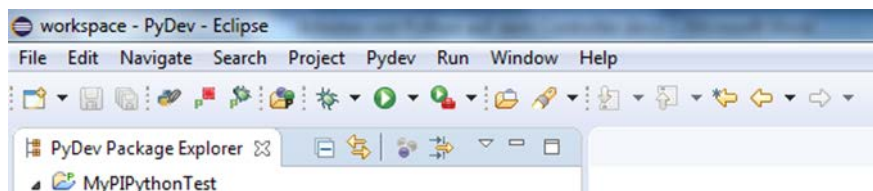
3. Enter the name of the project into the **PyDev Project** dialog and select the Python interpreter installed.

In the example shown, "MyPIPythonTest" is entered as **Project name** and Python 2.7 is selected as the **Interpreter**.



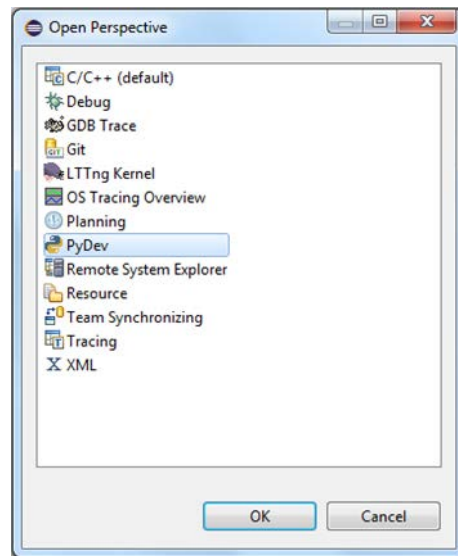
4. Click **Finish** to create the project.

The project creation dialog is closed and the newly created project is opened in the PyDev view in the main window of Eclipse.



If the PyDev view is not opened automatically:

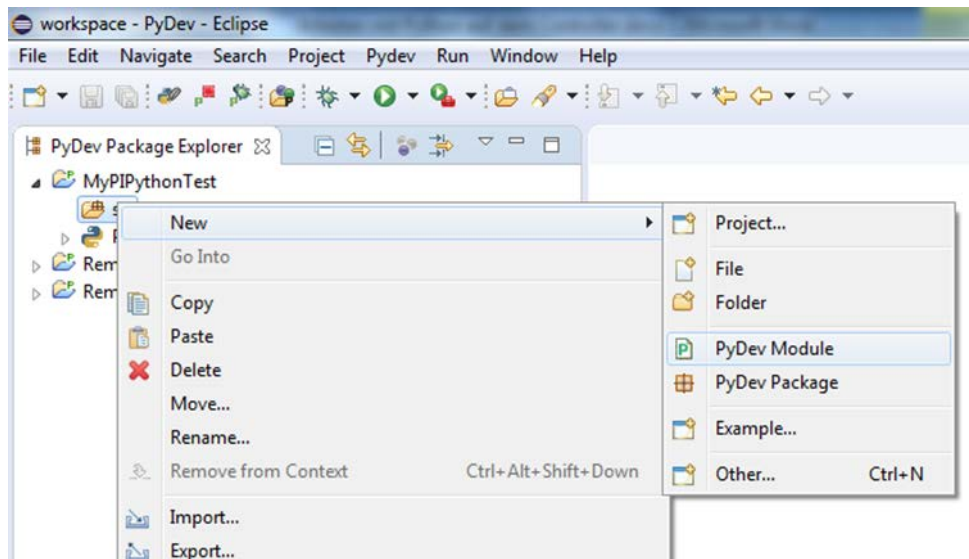
- a) In the main window of Eclipse, open the **Open Perspective** dialog via **Window > Perspective > Open Perspective > Other**.
- b) Mark the **PyDev** entry in the **Open Perspective** dialog and click **OK**.



Creating and Running Scripts

After creating the project, it is possible to create the first script in the PyDev view opened in Eclipse:

1. Right-click the **src** directory below the newly created project to open the context menu and call the **New > PyDevModule** option.

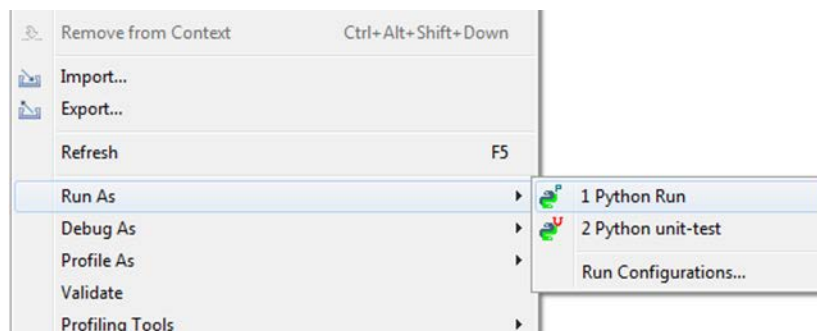


2. You can now write the script.

The following example script opens a connection dialog and queries the identification string of the controller. The script uses the PI_GCS2_DLL.dll.

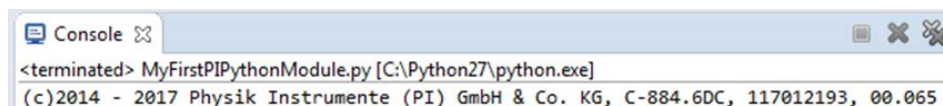
```
from pipython import GCSDevice
gcs = GCSDevice ('C-884')
gcs.InterfaceSetupDlg ()
print gcs.qIDN ()
gcs.CloseConnection ()
```

3. To start the module, right-click the script created to open the context menu and call the **Run As > 1 Python Run** option.



The script is run.

A connection dialog opens after our example script starts running. The identification string of the controller is output to the console window of Eclipse after successful connection.



Using Eclipse and PIPython on the Controller

The settings described under "Installing Eclipse" (p. 6) and "Configuring Eclipse" (p. 7) are required to take full advantage of PIPython on the controller.

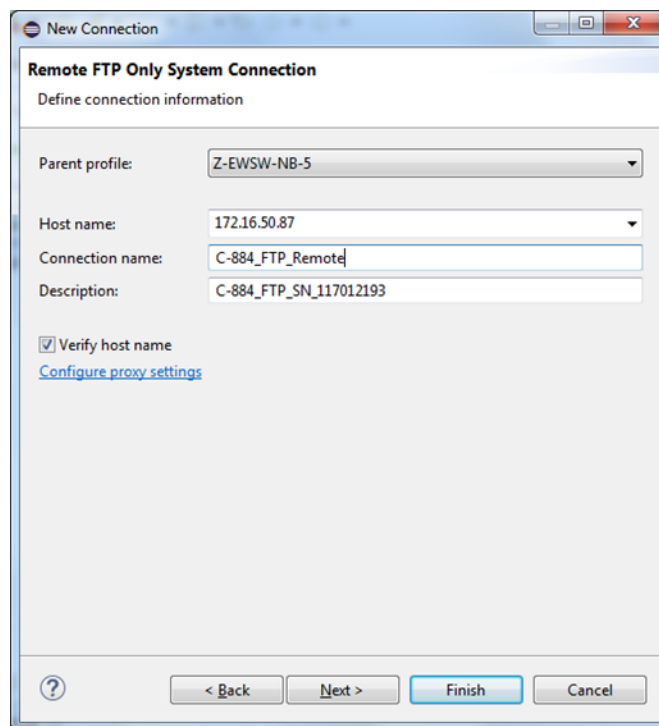
Configuring the Connection to the Controller

1. The RSE view (Remote System Explorer) must be opened in Eclipse to establish connection (via SSH) to the controller:
 - a) In the main window of Eclipse, open the **Open Perspective** dialog via **Window > Perspective > Open Perspective > Other**.

- b) Mark the **Remote System Explorer** entry in the **Open Perspective** dialog and click **OK**.
2. Right-click in the RSE view to open the context menu and call the **New > Connection... > FTP Only** option.

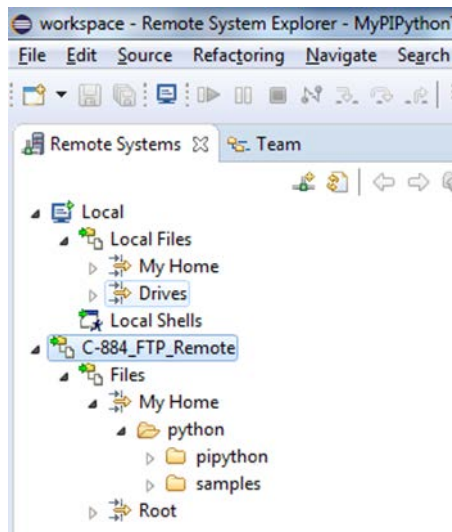
The **New Connection** dialog is opened for configuring a remote connection.

3. Enter the following information for the connection into the **New Connection** dialog:
 - **Host name:** IP address of the controller that is to be connected
 - **Connection name:** User-defined name of the connection to be saved in Eclipse
 - **Description:** User-defined description of the connection



4. Close the dialog via **Finish**.

The configured connection is shown in the RSE view in Eclipse:



Establishing Connection to the Controller

- In the RSE view in Eclipse, mark the item of the newly configured remote connection and right-click to open the context menu and call the **Connect** option to open the connection.

For the user name and password, please refer to "Login Data for PIPython" (p. 4).

Now it is possible to access the file system of the controller in the directory structure. By default, user scripts should be in the directory `/piuser/python` underneath the home directory. There is also a subdirectory which already contains some example scripts.

User scripts can be created directly on the controller by right-clicking the corresponding directory, opening the context menu, and calling the **New > File** (+ file extension ".py") option. It is also possible to copy scripts from the local hard disk to the controller.

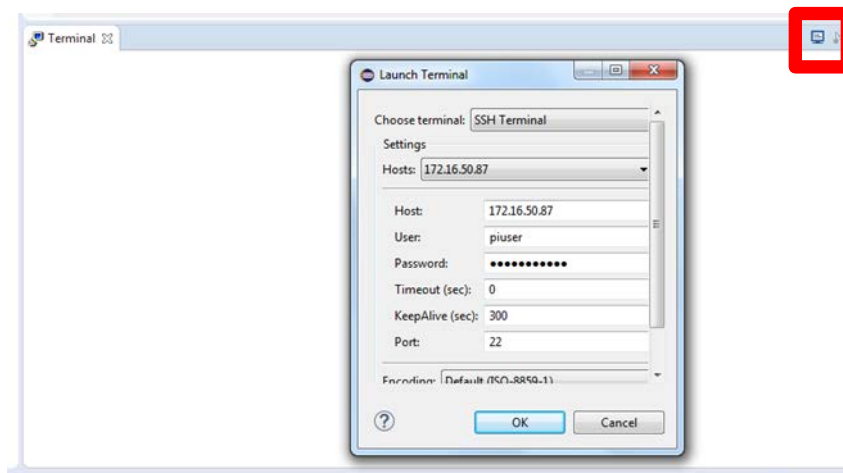
In contrast to running scripts on the PC, DLLs are not used when running scripts on the controller. Therefore, import of the required components is slightly different, as can be seen in the following example script. For this reason, the connection cannot be made via a dialog; instead, it is made via a socket.

The following example script makes a connection to the controller firmware, queries the identification string of the controller, and outputs it.

```
from pipython.gcscmds import GCSCmds
from pipython.gcsmsgs import GCSMsgs
from pipython.interfaces.pisocket import PISocket
gateway = PISocket (host='127.0.0.1')
messages = GCSMsgs (gateway)
```

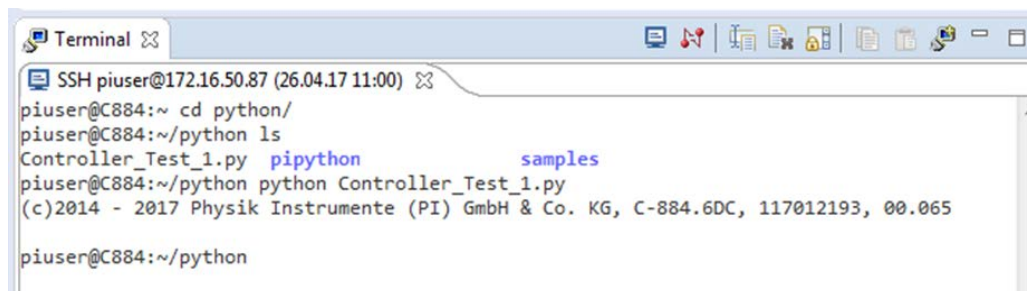
```
gcs = GCSCommands (messages)
print gcs.qIDN ( )
gateway.close ( )
```

To make the connection to the controller, it is possible to display the terminal window in Eclipse via **Window > Show View**. A connection can be made here via the corresponding symbol (top right, see figure).



After changing to the corresponding directory (home/piuser/python), it is possible to start Python scripts by calling the `python <Skriptname>` command.

In the example shown in the following figure, the "Controller_Test_1.py" script is started. It opens a connection dialog and queries the identification string of the controller. The call is: `python Controller_Test_1.py`. The figure shows a script that was run successfully.



Using the Remote Debugger in Eclipse

Remote debugging of Python scripts is also possible. Here for example, it is possible to monitor variables in the Eclipse IDE during execution of the script on the controller.

Activating the Debug Function in the Script

For remote debugging, the script to be debugged must contain the following:

1. In the script to be debugged, import the "pydevd" module by entering the following command: `import pydevd`
2. Call the pydevd function "settrace" with the IP address of the PC, e.g.:
`pydevd.settrace('172.17.130.84')`

Example of a simple script that was prepared for remote debugging:

```
import sys
sys.path.append('/home/piuser/python/pysrc')
import pydevd
x = 0
while x < 100 :
    print (x)
    pydevd.settrace ('172.17.130.84')
    x = x + 1
```

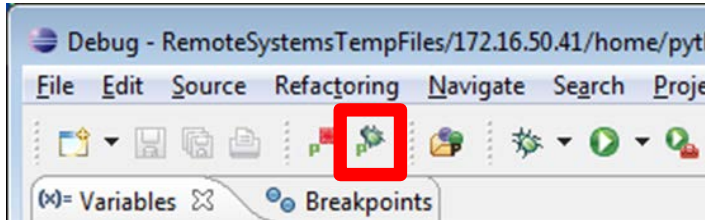
Configuring the Remote Debugger

If you want to use the Python remote debugger in Eclipse, proceed as follows:

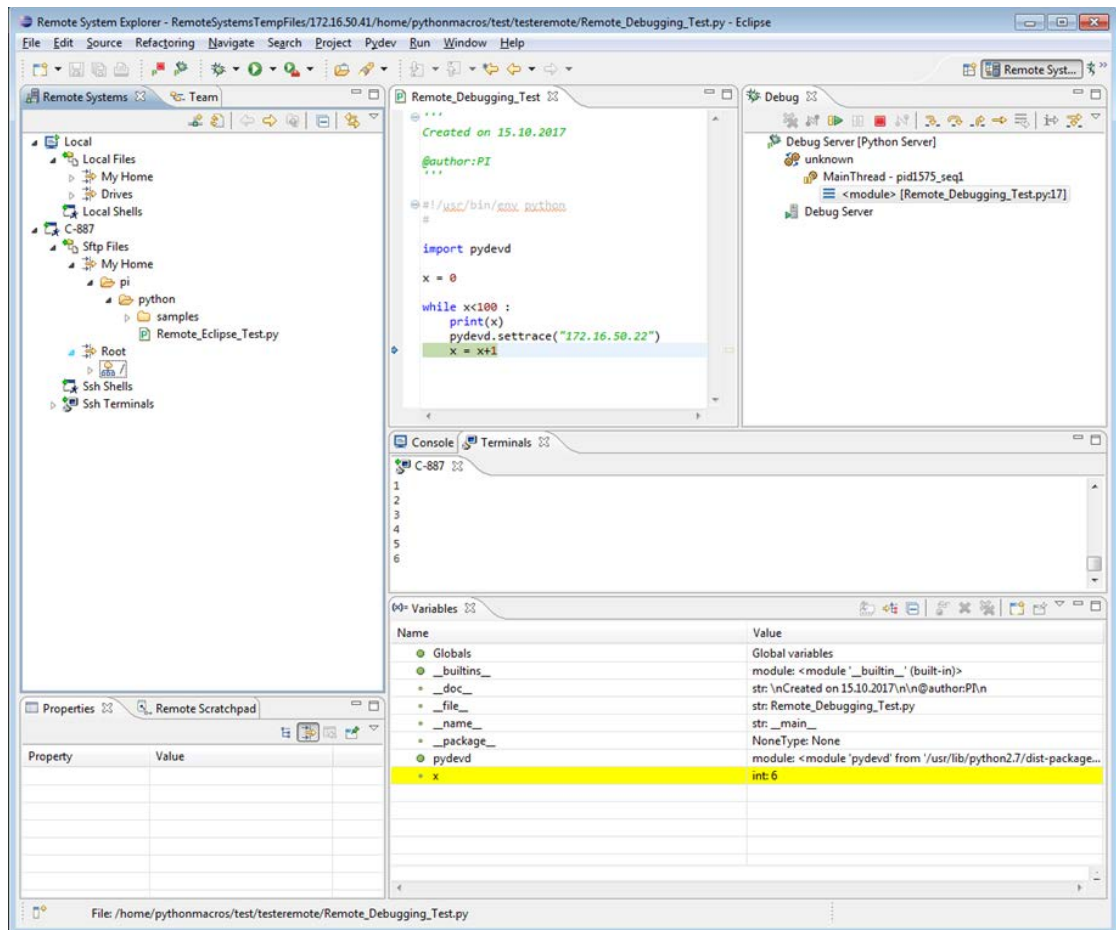
1. Copy the entire contents of the directory that includes the **pydevd.py** file into the following directory on the controller: `/home/piuser/python/pysrc`
Depending on the version of Eclipse, the pydevd.py file can be in different directories, e.g.:
`C:\Program Files (x86)\Eclipse\eclipse\plugins\org.python.pydev_VERSION\pysrc`
2. To make Python remote debugging more convenient, a number of views should be added to the current view in Eclipse:
 - a) Add the buttons for starting and stopping the PyDev server to the toolbar by calling **Window > Customize perspective > Command groups availability > PyDev debug**.
 - b) Add the "Debug" and "Variables" views to Eclipse by calling **Window > Show View > Other > Debug**.

Using the Debug Function

Now, the PyDev server can be started in Eclipse via the green button (see figure). Finally, the script can be run via the terminal as before.



The script can be run step-by-step by pressing the **F8** or the **Resume** button.



A warning that appears in some versions of the pydev debugger can be ignored.

For further information on the configuration of the remote debugger with Eclipse/Pydev, see http://www.pydev.org/manual_adv_remote_debugger.html.

Using PyCharm Professional

Installing PyCharm Professional

If you want to use PyCharm Professional for working with PIPython, install the following onto your PC:

- IDE: PyCharm Professional (e.g., download via <https://www.jetbrains.com/pycharm/>)

All of the following descriptions and figures refer to the Version 2017.1.2 in Windows.

Configuring PyCharm Professional

Apart from selecting the Python interpreter after installing PyCharm, no further settings need to be made to use Python and PIPython on the PC.

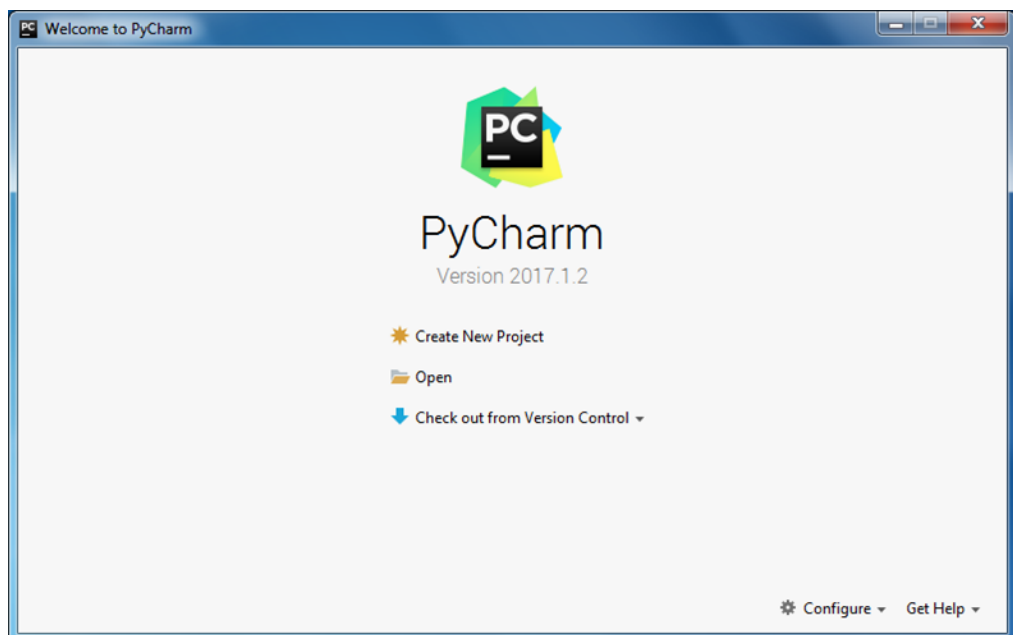
It is possible to define selection of the interpreter globally via **File > Default Settings > Project Interpreter** or specifically by project via **File > Settings > Project: Name > Project Interpreter**.

Using PyCharm and PIPython on the PC

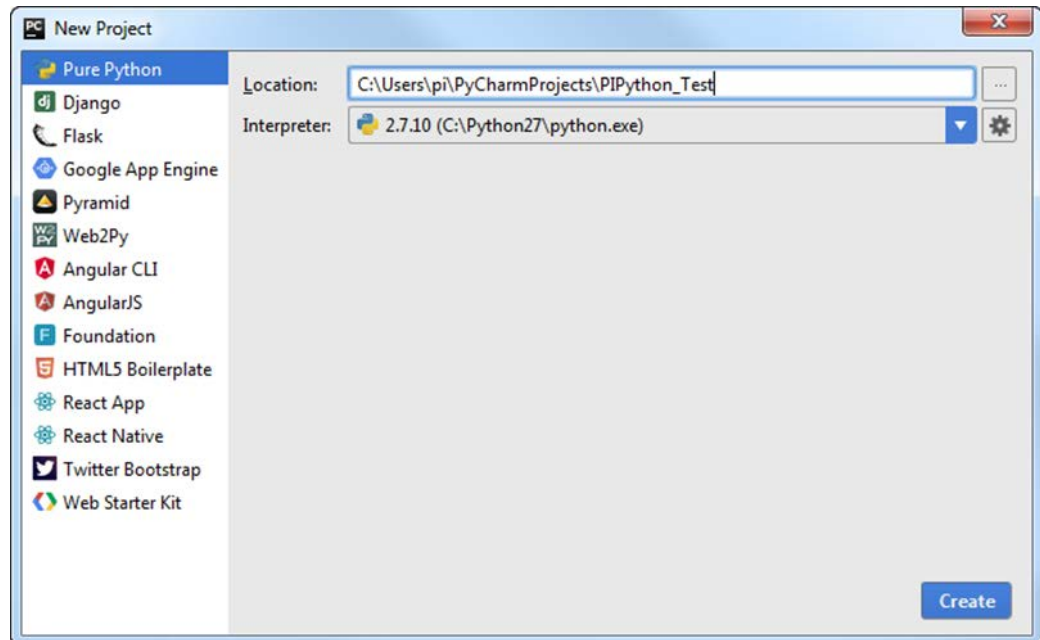
Creating a project

The first project can be created after installing and configuring PyCharm. Proceed as follows:

1. Select the **Create New Project** option in the main window of the program.



The **New Project** dialog is opened.



2. Enter the name to be used for saving the project into the **Location** field and if necessary, select the Python interpreter to be used in the **Interpreter** field.

In the example shown, "PIPython_Test" was selected as the project name and Python 2.7 as the interpreter.

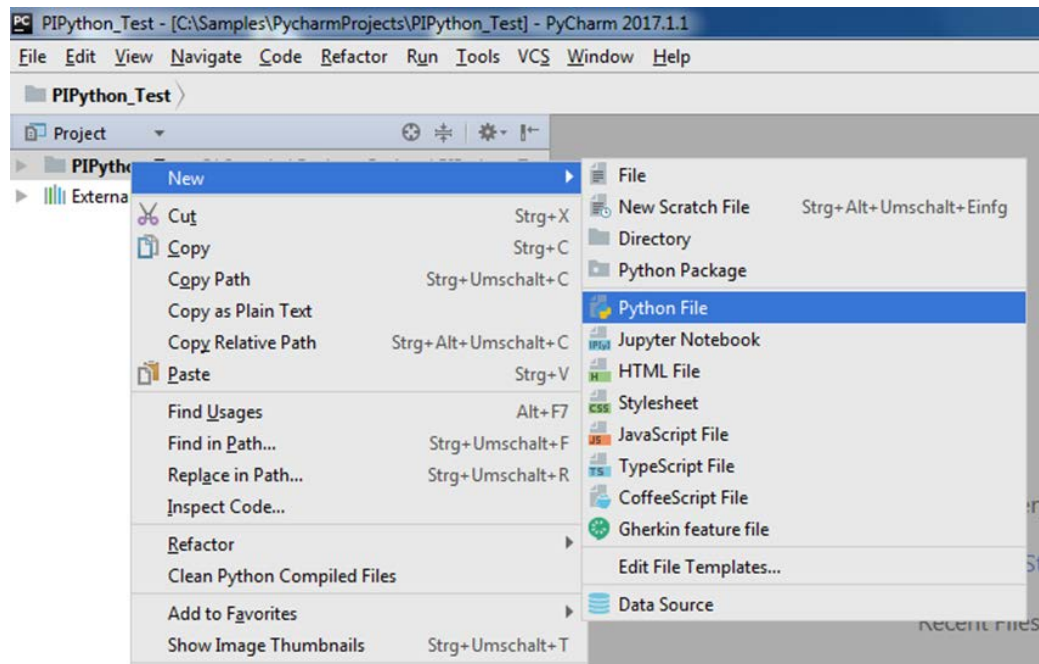
3. Click **Create** to create the project.

The project creation dialog is closed and the newly created project is opened in the main window of PyCharm.

Creating and Running Scripts

After creating the project, it is possible to create the first script in PyCharm:

1. Right-click the newly created project to open the context menu and call the **New > Python File** option.

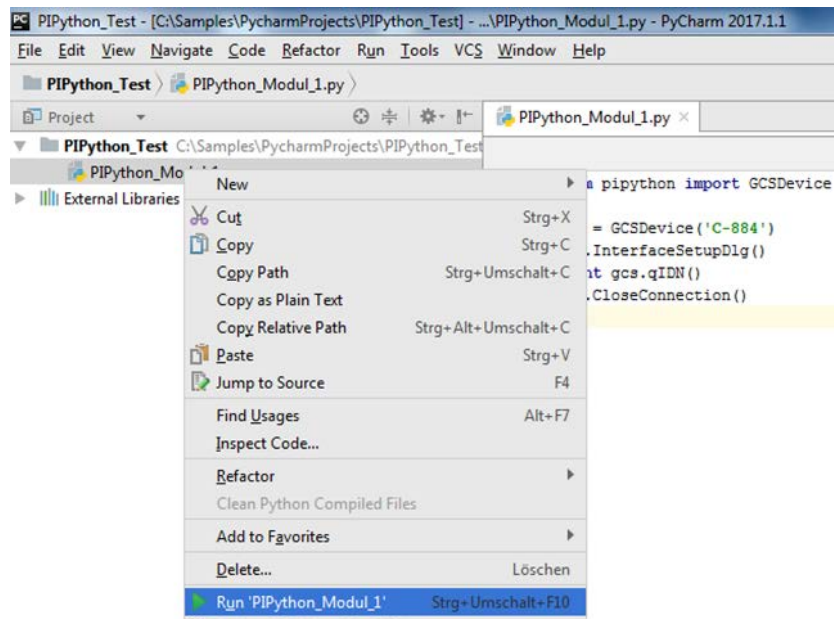


2. You can now write the script.

The following example script opens a connection dialog and queries the identification string of the controller. The script uses the PI_GCS2_DLL.dll.

```
from pipython import GCSDevice
gcs = GCSDevice ('C-884')
gcs.InterfaceSetupDlg ()
print gcs.qIDN ()
gcs.CloseConnection ()
```

3. To start the script, right-click the script created to open the context menu and call the **Run <script name>** option.



The script is run.

A connection dialog opens after the example script starts running. The identification string of the controller is output to the console window of PyCharm after successful connection.




Using PyCharm and PIPython on the Controller

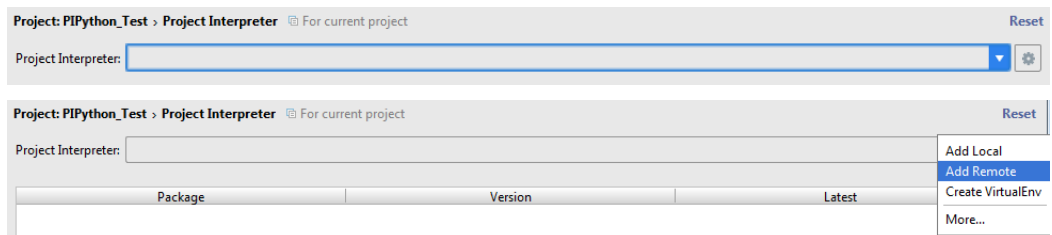
The "SSH Remote Run" plug-in from PyCharm must be activated to work with the Python interpreter on the controller. This can be checked under **File > Settings > Plugins** and activated if necessary.

The settings described under "Configuring PyCharm Professional" (p. 18) are required to take full advantage of PIPython on the controller.

Configuring the Connection to the Controller for the Remote Interpreter

Carry out the following steps in PyCharm Professional:

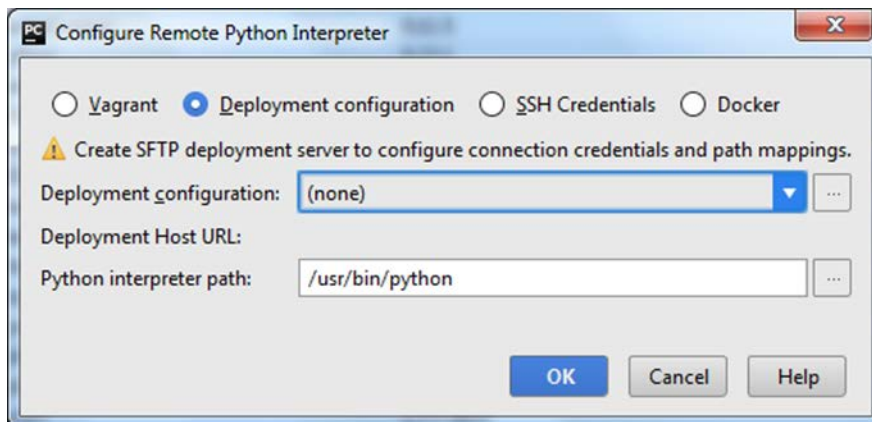
1. Add a remote interpreter to your project by calling **File > Settings > Project: Name > Project Interpreter** (Name = the name of your project).
2. Right-click on the  button behind the **Project Interpreter** input field and select the **Add remote** option in the context menu.



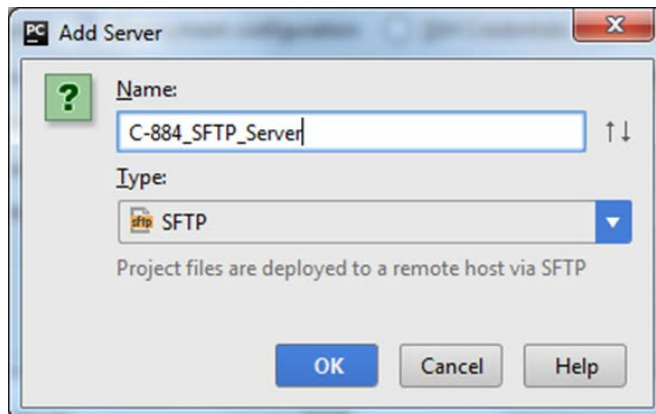
The configuration dialog for the remote interpreter is opened.

Note: When adding a Python remote interpreter, the PyCharm helper files are copied into the controller automatically. These are used for example, for debugging. The version information of the helper files is checked by PyCharm during each remote run and updated if necessary.

3. Mark the **Deployment configuration** option in the configuration dialog.



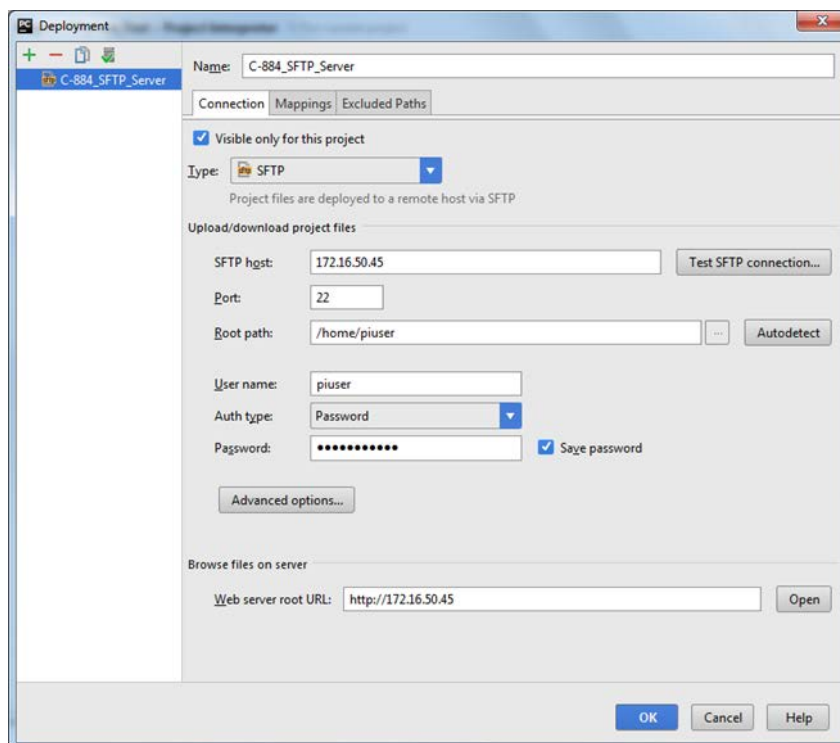
4. Click the selection button (...) behind the **Deployment configuration** field to open the **Add Server** dialog for defining a server.



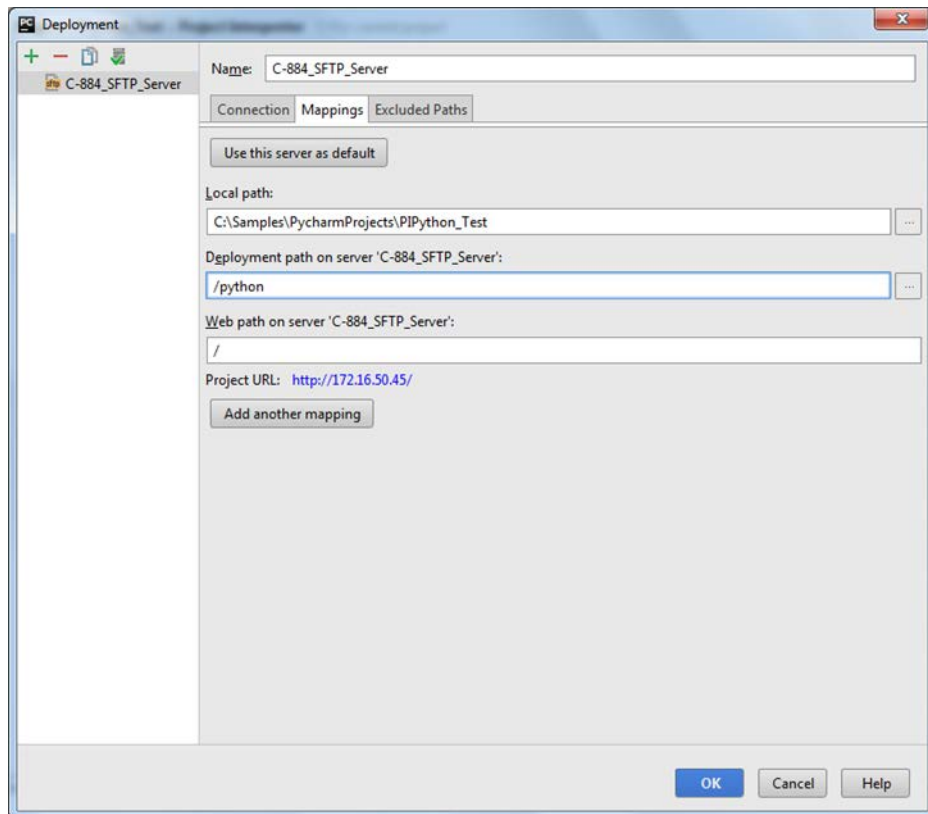
5. Enter a name (user defined) for the server into the **Name** field in the **Add Server** dialog and select **SFTP** in the **Type** field.
6. Close the **Add Server** dialog with **OK**.

The **Deployment** dialog is opened.

7. Configure the connection settings in the **Connection** tab as shown in the following figure. For the user name and password, please refer to "Login Data for PIPython" (p. 4).



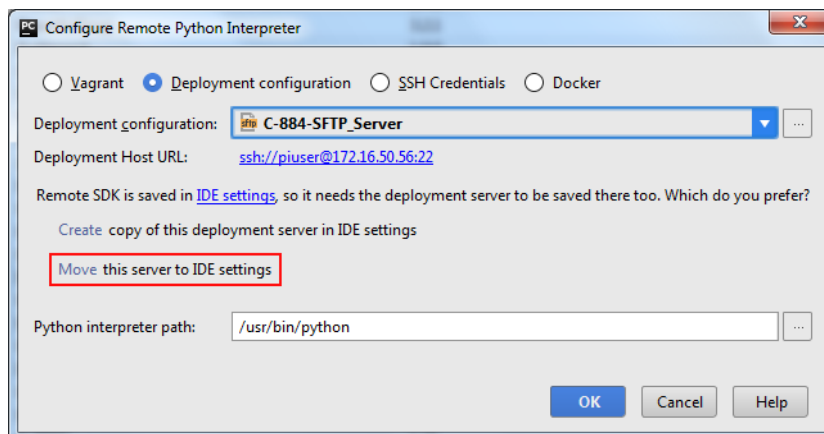
8. Set the deployment path on the controller in the **Mappings** tab as shown on the following figure. The scripts are transferred to this location as required.



9. Close the deployment configuration with **OK**.

The **Deployment** dialog is closed and you are returned to the **Configure Remote Python Interpreter** dialog.

10. Activate the **Move** function (marked in the figure) to accept the configuration for all projects. Finally, all PyCharm helper files are copied to the controller automatically.



11. Close the **Configure Remote Python Interpreter** dialog with **OK**.

The configuration of the connection to the controller for the remote interpreter is complete.

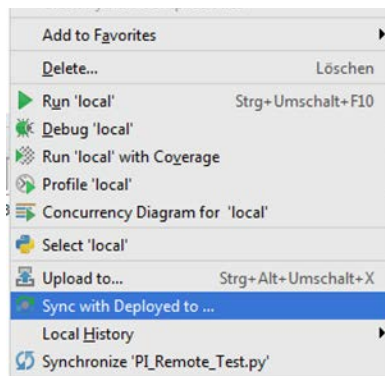
Running Scripts on the Controller

In contrast to running scripts on the PC, DLLs are not used when running scripts on the controller. Therefore, import of the required components is slightly different, as can be seen in the following example script. For this reason, the connection cannot be made via a dialog; instead, it is made via a socket.

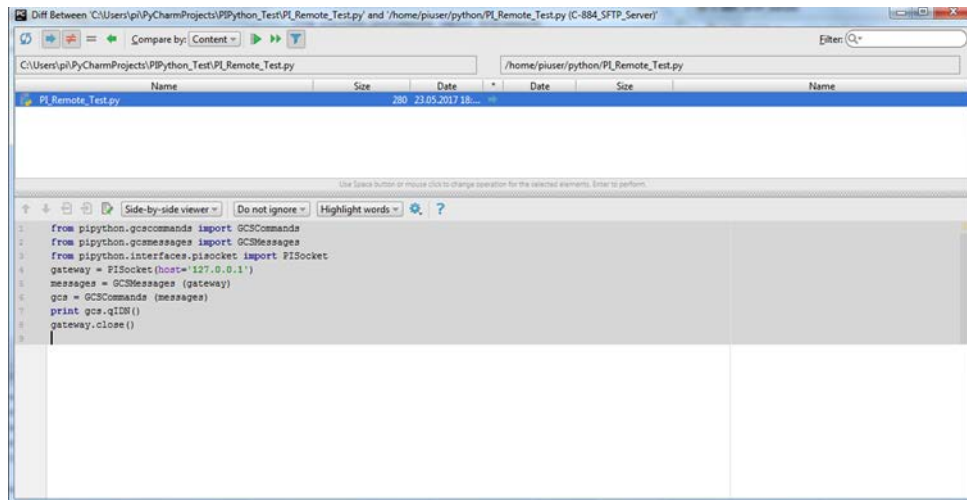
The following example script makes a connection to the controller firmware, queries the identification string of the controller, and outputs it.

```
from pipython.gcscmds import GCSCmds
from pipython.gcsmsgs import GCSMessages
from pipython.interfaces.pisocket import PISocket
gateway = PISocket (host='127.0.0.1')
messages = GCSMessages (gateway)
gcs = GCSCmds (messages)
print gcs.qIDN ()
gateway.close ()
```

Scripts can be transferred either directly into the specified directory on the controller via the Upload **to ...** context menu option or synchronized with it via the **Sync with Deployed to ...** context menu option.

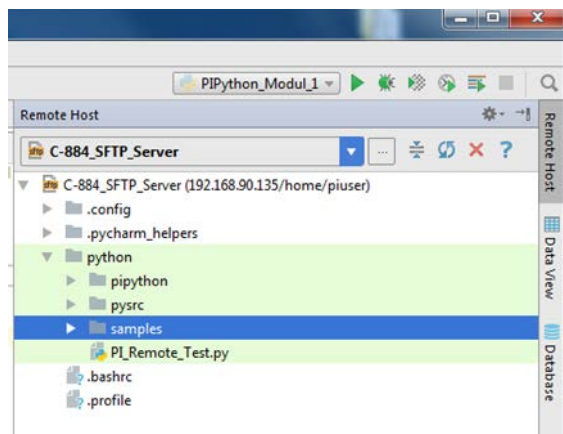


The following figure shows synchronization of scripts with the controller:

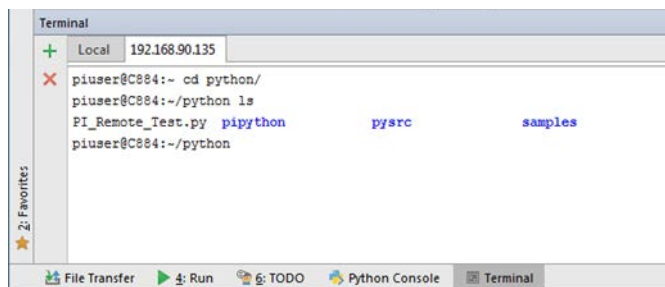


The selected file can be transmitted to the controller using the green arrow ("Synchronize Selected").

A file browser can be displayed on the controller to make working more convenient. For this purpose, call the **Tools > Deployment > Browse Remote Host** menu function in PyCharm.



If the script is already on the controller, a terminal window to the controller can be opened via **Tools > Start SSH session....** After changing to the corresponding directory (home/piuser/python), it is possible to start Python scripts by calling the `python <Skript-name>` command.



In the example shown in the following figure, the "PI_Remote_Test.py" is started. The call is: `python PI_Remote_Test.py`. The figure shows a script that was run successfully.



```
piuser@C884:~$ cd python/
piuser@C884:~/python$ ls
PI_Remote_Test.py  pipython  pysrc  samples
piuser@C884:~/python$ python PI_Remote_Test.py
(c)2014 - 2017 Physik Instrumente (PI) GmbH & Co. KG, C-884.6DC, 117012193, 00.070

piuser@C884:~/python$
```

For further information on configuring a remote interpreter in PyCharm, see <http://www.jetbrains.com>.

Using the Remote Debugger in PyCharm

Remote debugging of Python scripts is also possible. Here for example, it is possible to monitor variables in the PyCharm IDE during execution of the script on the controller.

Activating the Debug Function in the Script

For remote debugging, the script to be debugged must contain the following:

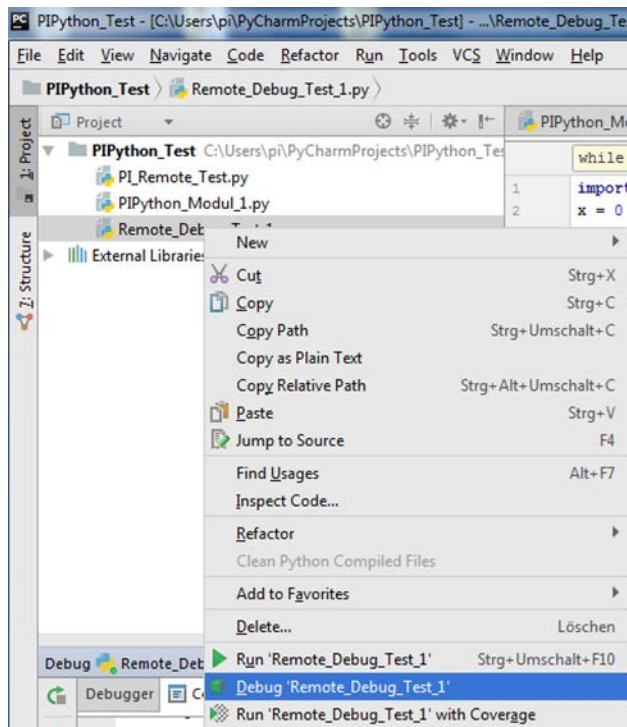
1. In the script to be debugged, import the "pydevd" module by entering the following command: `import pydevd`
2. Call the pydevd function "settrace" with the IP address of the PC, e.g.: `pydevd.settrace('172.17.130.84')`

Example of a simple script that was prepared for remote debugging:

```
import pydevd
x = 0
while x < 100 :
    print (x)
    pydevd.settrace ('172.17.130.84')
    x = x + 1
```

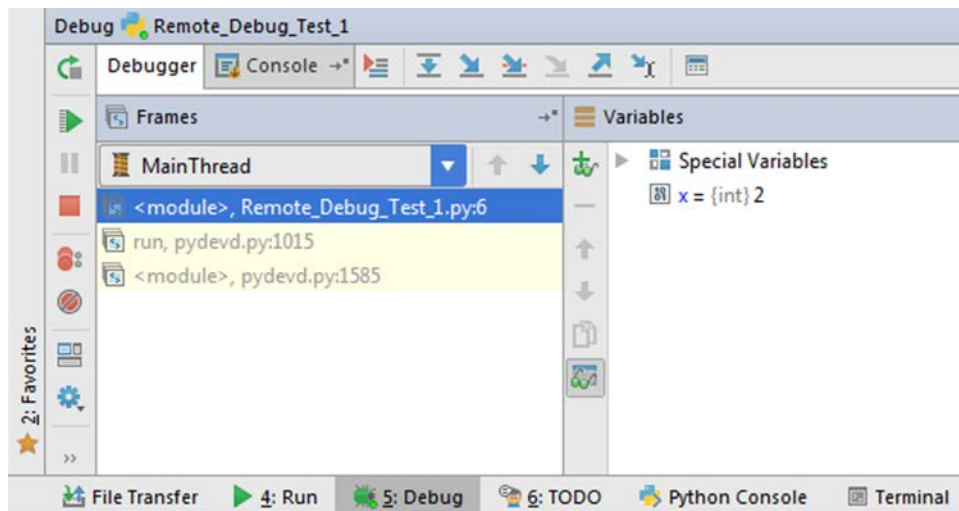
Using the Debug Function

Remote debugging can be started in PyCharm via the **Debug <script name>** context menu option.



The script is shown in the **Console** tab during execution. The variables are shown in the **Debugger** tab (in the following figure: in the right-hand window).

The script can be run step-by-step via the **F9** function key or via the **Resume Program** button.



Executing Scripts on the Controller

Automatic Execution of Scripts during System Start

If a Python script is to be run automatically during the start of the controller (comparable to the behavior of the autostart function of GCS macros), it must be named "autostart.py" and be saved in /home/piuser/python.

Executing Scripts Manually

Manual calling of a Python script on the controller can be done via an SSH connection (e.g., with a terminal program like, for example, PuTTY):

```
piuser@C884:~ python python/script.py
```

The script is stopped automatically when the connection is terminated. If this is not desired, the call must end with "&":

```
piuser@C884:~ python python/script.py &
```