Open for Innovation
KNIME®

KNIME Python Integration Installation Guide

KNIME AG, Zurich, Switzerland
Version 3.7 (last updated on 2019-02-05)

Table of Contents

Introduction

This guide describes how to install the KNIME Python Integration to be used with KNIME Analytics Platform.

> This guide refers to the KNIME Python Integration that was part of the v3.4 release of KNIME Analytics Platform (not to be confused with the KNIME Python Scripting Extension). The integration is the recommended and most recent way to use arbitrary Python™ scripts in KNIME Analytics Platform and supports both Python 2 as well as Python 3.

The KNIME Python Integration makes use of an existing Python, which is installed alongside KNIME Analytics Platform. As the KNIME Python Integration depends on certain Python packages, the Python installation needs to have these packages installed. Our recommended way to set up such a Python environment is to use the Anaconda Python distribution from Continuum Analytics. In this guide we describe how to install Python and the necessary packages using Anaconda, as well as how to configure the KNIME Python Integration.

Quickstart

This quickstart guide shows you the basic steps required to install the KNIME Python Integration and its prerequisites with Python. We do not provide any further details. If you'd like are more thorough explanation, please refer to the full Anaconda Setup Guide.

1. Install Anaconda first. It is used to manage Python environments. Anaconda can be downloaded here (choose Anaconda with Python 3).
2. Next, create a `conda` environment. This is a folder that contains a specific Python version and the packages that are installed. Creating such an environment can be easily done using a configuration YAML file, which lists all the packages to be installed in the newly created environment. Download the following configuration file (*Right-Click → Save As…*) for Python 3 and execute the command below using Anaconda *prompt* (the Anaconda *prompt* can be found by entering `anaconda` in the Windows search), Linux *shell* or Mac *terminal*:

   ```
   conda env create -f <PATH-TO-FOLDER>/py36_knime.yml
   ```

   Replace `<PATH-TO-FOLDER>` with the path to the folder that contains the configuration file you just downloaded. This will create a new `conda` environment named `py36_knime`, containing all necessary packages.

3. Now, create a start script for the conda environment. This script makes sure that the correct environment is loaded. On Linux and Mac the script looks like this:

```
#! /bin/bash
# Start by making sure that the anaconda folder is on the PATH
# so that the source activate command works.
# This isn't necessary if you already know that
# the anaconda bin dir is on the PATH
export PATH="<PATH_WHERE_YOU_INSTALLED_ANACONDA>/bin:$PATH"

conda activate py36_knime
python "$@" 1>&1 2>&2
```

   On Windows, the script looks like the following:

```
@REM Adapt the folder in the PATH to your system
@SET PATH=<PATH_WHERE_YOU_INSTALLED_ANACONDA>\Scripts;%PATH%
@CALL activate py36_knime || ECHO Activating python environment failed
@python %*
```

   Create a file with the appropriate script for your system (e.g. call it *py36.bat* on Windows and *py36.sh* on Linux) and replace `<PATH_WHERE_YOU_INSTALLED_ANACONDA>` with the path to your Anaconda installation. On Linux/Mac, you additionally need to make the script executable using:

```
chmod gou+x py36.sh
```

4. Finally, point the KNIME Python Integration to the start script you just created. After installing the KNIME Python Integration go to the Python Preference page located at *File → Preferences*. Select *KNIME → Python* from the list on the left. In the page that opens, specify the path to the sh/bat file you created for Python 3.

   Assuming everything is correct, the Python version is now shown in the dialog and you're ready to go.

The above guide installs a Python 3 environment. If you want to install Python 2, you'll find the respective configuration file in the Anaconda Setup guide.

If you want to use the Python Script (DB) node, please see the additional instructions listed here.

Anaconda Setup

This section describes how to install and configure Anaconda to be used with the KNIME Python Integration. Anaconda allows you to manage several so called `conda` environments, which can contain different different Python versions and different sets of packages, also using different versions. A `conda` environment is essentially a folder that contains a specific Python version and the packages that are installed. Hence, it enables you to have several different Python versions installed on your system at the same time in a clean and easy to maintain manner. For KNIME, this is especially useful as it allows you to use Python 3 and Python 2 at the same time without running into version issues as Anaconda keeps each environment nicely encapsulated and independent of all others. Furthermore, Anaconda is able to create predefined environments with a single command and makes it easy to add Python packages to existing ones.

Next, you will learn how to set up an environment that contains the needed dependencies for the KNIME Python Integration.

Anaconda installation

First, you need to install the latest Anaconda version. On the Anaconda download page you can choose between Anaconda with Python 3.x or Python 2.x, however this only affects the root `conda` environment, which we will not use (as we are creating our own). Hence, you can choose either one (if you're not sure, we suggest selecting Python 3).

Creating a Conda environment

After Anaconda is installed, you need to create a new `conda` environment. This can be done using a YAML configuration file, which lists all of the packages to be installed in the newly created environment. We have provided two such configuration files below (one configuration file to create a new Python 3 environment and one file for Python 2) that list all needed dependencies for the KNIME Python Integration:

py36_knime.yml

```
name: py36_knime      # Name of the created environment
channels:             # Repositories to search for packages
- defaults
- anaconda
dependencies:         # List of packages that should be installed
- python=3.6          # Python
- pandas=0.23         # Table data structures
- jedi=0.13           # Python script autocompletion
- python-dateutil=2.7 # Date and Time utilities
- numpy=1.15          # N-dimensional arrays
- cairo=1.14          # SVG support
- pillow=5.3          # Image inputs/outputs
- matplotlib=3.0      # Plotting
- pyarrow=0.11        # Arrow serialization
- IPython=7.1         # Notebook support
- nbformat=4.4        # Notebook support
- scipy=1.1           # Notebook support
```

py27_knime.yml

```
name: py27_knime      # Name of the created environment
channels:             # Repositories to search for packages
- defaults
- anaconda
dependencies:         # List of packages that should be installed
- python=2.7          # Python
- pandas=0.23         # Table data structures
- jedi=0.13           # Python script autocompletion
- python-dateutil=2.7 # Date and Time utilities
- numpy=1.15          # N-dimensional arrays
- cairo=1.14          # SVG support
- pillow=5.3          # Image inputs/outputs
- matplotlib=2.2      # Plotting
- protobuf=3.5        # Serialization for deprecated Python nodes
- IPython=5.8         # Notebook support
- nbformat=4.4        # Notebook support
- scipy=1.1           # Notebook support
```


The above configuration files only contain the Python packages that the KNIME Python Integration depends on. If you want to use further Python packages in KNIME you can either add the name of the package at the end of the configuration file or add them after the environment has been created.

For example, for Python 3 you can use the `py36_knime.yml` and download it (*Right click* → *Save As…*) to any folder on your system (e.g. your home folder). In order to create an environment from this file, open a *shell* (Linux), *terminal* (Mac), or Anaconda *prompt* (Windows, can be found by entering `anaconda` in the Windows search), change the directory

to the folder that contains the configuration file and execute the following command:

```
conda env create -f py36_knime.yml
```

This command creates a new environment with the name provided at the top of the configuration file (of course you can change the name), and downloads and installs all of the listed packages (depending on your internet speed, this may take a while).

If you want to use both Python 3 and Python 2 at the same time, just repeat the above steps using the respective configuration file.

The list of dependencies for Python 3 and Python 2 is almost the same, however version numbers change.

After Anaconda has successfully created the environment, Python is all set up and you are ready to proceed with Setting up the KNIME Python Integration.

If you want to use the Python Script (DB) node, please see the additional instructions listed in the section Dependency for Python Script (DB) node.

Further information on how to manage Anaconda environments can be found here.

Manually installing additional Python packages

The Anconda configuration files listed above only contain the packages that have to be installed in order for the KNIME Python Integration to work properly. Hence, if you want to use Python packages other than the ones listed in the configuration files, these can be easily added manually after the environment has already been created. E.g. if you want to use functionality from `scikit-learn` in KNIME Python nodes, you can use the following command:

```
conda install --name <ENV_NAME> scikit-learn
```

Just replace `<ENV_NAME>` with the name of the environment in which you want to install the package.

You can easily specify a specific version of the package with e.g. `scikit-learn==0.20.2`

Further information on how to manage Anaconda packages can be found here.

Dependency for Python Script (DB) node

If you plan to use the Python Script (DB) node, one additional dependency (`jpype1`) is required that is not provided by Anaconda. However, it can be easily added to a `conda` environment using the standard Python package manager `pip`.
First activate your environment using the following command on Windows:

```
activate <ENV_NAME>
```

Or the following command on Linux/Mac:

```
conda activate <ENV_NAME>
```

Next execute:

```
pip install jpype1
```

Troubleshooting

Mac Matplotlib

On Mac, there may be issues with the `matplotlib` package, which can be resolved by executing the following commands:

```
mkdir ~/.matplotlib
echo "backend: TkAgg" > ~/.matplotlib/matplotlibrc
```

Setting up the KNIME Python Integration

This section describes how to install and configure the KNIME Python Integration using an existing Anaconda environment. If you did not yet set up Anaconda and/or the recommended Python environment, please refer to the Anaconda Setup guide.

Installation

From KNIME Analytics Platform, install the KNIME Python Integration by going to *File* → *Install KNIME Extensions*. The KNIME Python Integration can be found under *KNIME &*

Extensions or by entering *Python Integration* into the search box.

Creating a start script for Python

In order to use the previously created Anaconda environment for the KNIME Python Integration, you need to create a start script (shell script on Linux and the Mac, bat file on Windows).

If you are using Linux or the Mac, here's an example shell script for the Python environment:

```
#! /bin/bash
# Start by making sure that the anaconda folder is on the PATH
# so that the source activate command works.
# This isn't necessary if you already know that
# the anaconda bin dir is on the PATH
export PATH="<PATH_WHERE_YOU_INSTALLED_ANACONDA>/bin:$PATH"

conda activate <ENVIRONMENT_NAME>
python "$@" 1>&1 2>&2
```

On Windows, the script looks like this:

```
@REM Adapt the folder in the PATH to your system
@SET PATH=<PATH_WHERE_YOU_INSTALLED_ANACONDA>\Scripts;%PATH%
@CALL activate <ENVIRONMENT_NAME> || ECHO Activating python environment failed
@python %*
```

These example scripts need to be edited in order to point to the location of your Anaconda installation and to `activate` the correct Anaconda environment. I.e. replace the `<PATH_WHERE_YOU_INSTALLED_ANACONDA>` with the location of your Anaconda installation and `<ENVIRONMENT_NAME>` with the name of the `conda` environment the script should start and which you created in the Anaconda Setup guide (e.g. in this case `py36_knime` for Python 3.6 or `py27_knime` for Python 2.7).

For example on Windows, create a new bat script named e.g. `py36.bat` (`py36.sh` on Linux or Mac) and paste the corresponding script to the file.

On Linux/Mac you additionally need to make the file executable (i.e. `chmod gou+x py36.sh`).

Configure the KNIME Python Integration

Once you have created the start script, you will be almost done setting up Python. The last thing to do, is to point KNIME to start script you just created. This can be done in the Preference page of the KNIME Python Integration located at *File → Preferences*, then on the list on the left select *KNIME → Python*. The dialog should look like the screenshot shown below.

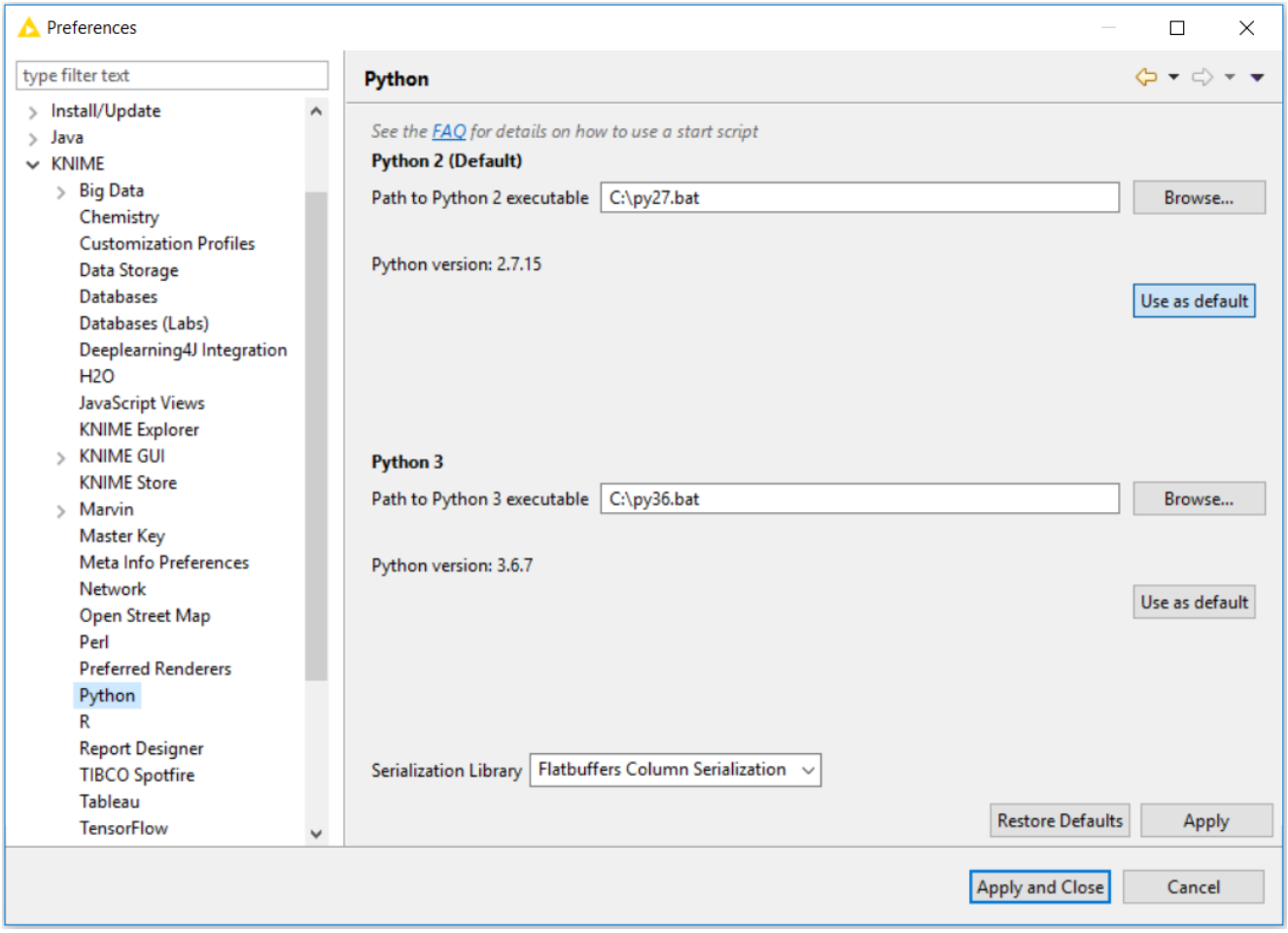

Figure 1. KNIME Python Preferences page. Here you can set the path to the executable script that launches your Python environment.

On this page you need to provide the path to the script/bat file you created to start Python. If you like, you can have configurations for both Python 2 and Python 3 (as is shown above). Just select the one that you would like to have as the default.

If everything worked out fine, the Python version is now shown in the dialog window and you are ready to go.

Advanced

You can choose which serialization library to be used by the KNIME Python Integration in

order to transfer data from KNIME Analytics Platform to Python.

This option does not usually need to be changed and can be left as the default.

Some of these serialization libraries have additional dependencies stated below, however if you followed the Anaconda Setup, all required dependencies are already included (see YAML configuration files on the Anaconda Setup guide). Currently, there are three options:

- *Flatbuffers Column Serialization* (**default** & **recommended**): no additional dependencies
- *Apache Arrow*: depends on `pyarrow` version `0.11`
- *CSV (Experimental)*: depends on `pandas` version `0.23`

Load Jupyter notebooks from KNIME

Existing Jupyter notebooks can be accessed within Python Script nodes using the `knime_jupyter` Python module (`knime_jupyter` will be imported automatically). Notebooks can be opened via the function `knime_jupyter.load_notebook`, which returns a standard Python module. The `load_notebook` function needs the path (path to the folder that contains the notebook file) and the name of the notebook (filename) as arguments. After a notebook has been loaded, you can call functions that are defined in the code cells of the notebook like any other function of a Python module. Furthermore, you can print the textual content of each cell of a Jupyter notebook using the function `knime_jupyter.print_notebook`. It takes the same arguments as the `load_notebook` function. An example script for a Python Script node loading a notebook could look like this:

```
# Path to the folder containing the notebook, e.g. the folder 'data' contained
# in my workflow folder
notebook_directory = "knime://knime.workflow/data/"

# Filename of the notebook
notebook_name = "sum_table.ipynb"

# Load the notebook as a Python module
my_notebook = knime_jupyter.load_notebook(notebook_directory, notebook_name)

# Print its textual contents
knime_jupyter.print_notebook(notebook_directory, notebook_name)

# Call a function 'sum_each_row' defined in the notebook
output_table = my_notebook.sum_each_row(input_table)
```

The `load_notebook` and `print_notebook` functions have two optional arguments:

- `notebook_version`: The Jupyter notebook format major version. Sometimes the version can't be read from a notebook file. In these cases, this option allows to specify the expected version in order to avoid compatibility issues. Should be an integer.
- `only_include_tag`: Only load cells that are annotated with the given custom cell tag (since Jupyter 5.0.0). This is useful to mark cells that are intended to be used in a Python module. All other cells are excluded. This is e.g. helpful to exclude cells that do visualization or contain demo code. Should be a string.

The Python nodes support code completion similar to an IDE. Just hit `ctrl-space` (`command-space` on the mac) e.g. after `knime_jupyter.` in order to show the available methods and documentation (`knime_jupyter` refers to the imported `knime_jupyter` Python module, e.g. see script example above).

The Jupyter notebook support for the KNIME Python Integration depends on the packages `IPython`, `nbformat`, and `scipy`, which are already included if you used the configuration files from the Anaconda Setup.

KNIME AG
Technoparkstrasse 1
8005 Zurich, Switzerland
www.knime.com
info@knime.com