Open for Innovation
KNIME

KNIME Python Integration Guide

KNIME AG, Zurich, Switzerland
Version 4.4 (last updated on 2021-10-25)

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 is available since 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.

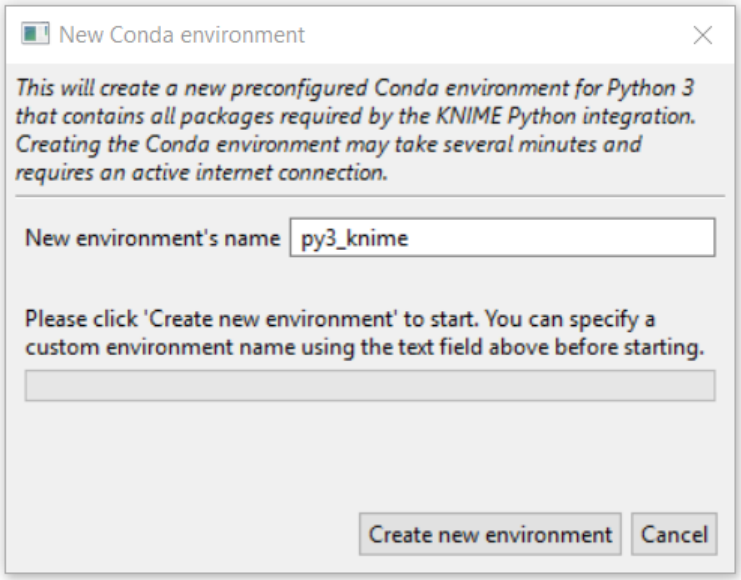
The KNIME Python Integration makes use of an existing Python, which is installed alongside KNIME Analytics Platform. As the KNIME Python Integration depends on certain Python packages, the Python installation needs to have these packages installed. Our recommended way to set up such a Python environment 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 a more thorough explanation, please refer to the more detailed Anaconda Setup Section.

1. First, install the KNIME Python Integration. In KNIME Analytics Platform, go to *File* → *Install KNIME Extensions*. The KNIME Python Integration can be found under *KNIME & Extensions* or by entering *Python Integration* into the search box.
2. Next, install Anaconda. It is used to manage Python environments. Anaconda can be downloaded here (choose Anaconda with Python 3).
3. Finally, configure 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, select **Conda** under *Python environment configuration*. Next, provide the path to your `Anaconda` installation folder (the default installation path is documented here). Once a valid path has been entered, the `conda` version number is shown. Below the `conda` version number you can choose which `conda` environment to be used for Python 3 and Python 2 by selecting it from a combo box. If you have already set up a Python environment, containing all the necessary dependencies for the KNIME Python Integration, just select it from the list and you are ready to go. If you do not have a suitable environment available, click the **New environment…** button. This opens the

following dialog:



Provide a name for the new environment and click the **Create new environment** button. This creates a new `conda` environment containing all the required dependencies for the KNIME Python Integration.

Depending on your internet connection, the environment creation may take a while as all packages need to be downloaded and extracted.

Once the environment is successfully created, the dialog closes and the new environment is selected automatically.

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 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 installed packages. This means you can 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; 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 dependencies needed for the KNIME Python Integration.

Anaconda installation

First, you need to install the latest Anaconda version (`Anaconda ≥ 2019.03, conda ≥ 4.6.2`). 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). Therefore, 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. There are two options to do this:

Option 1: Automatic (recommended)

A Python environment containing all required dependencies can be automatically created in the KNIME Python Integration Preference page. If you do not explicitly want to create an environment manually, please continue with Setting up the KNIME Python Integration.

Option 2: Manual

If you do not want to create a `conda` environment automatically, you can create one manually after Anaconda is installed. Do this with 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). They list all of the dependencies needed for the KNIME Python Integration:

py3_knime.yml

```
name: py3_knime       # Name of the created environment
channels:             # Repositories to search for packages
- defaults
- anaconda
- conda-forge
dependencies:         # List of packages that should be installed
- nbformat=4.4        # Notebook support
- scipy=1.1           # Notebook support
- pillow=5.3          # Image inputs/outputs
- cairo=1.14          # SVG support
- ipython=7.1         # Notebook support
- numpy=1.16.1        # N-dimensional arrays
- python=3.6          # Python
- matplotlib=3.0      # Plotting
- jpype1=0.6.3        # Databases
- pyarrow=0.11        # Arrow serialization
- jedi=0.13           # Python script autocompletion
- python-dateutil=2.7 # Date and Time utilities
- pandas=0.23         # Table data structures
- libiconv=1.15       # MDF Reader node
- asammdf=5.19.14     # MDF Reader node
```

py2_knime.yml

```
name: py2_knime       # Name of the created environment
channels:             # Repositories to search for packages
- defaults
- anaconda
- conda-forge
dependencies:         # List of packages that should be installed
- python=2.7          # Python
- pandas=0.23         # Table data structures
- jedi=0.13           # Python script autocompletion
- parso=0.7.1         # Jedi dependency this is the last version compatible with 2.7
- 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
- pyarrow=0.11        # Arrow serialization
- IPython=5.8         # Notebook support
- nbformat=4.4        # Notebook support
- scipy=1.1           # Notebook support
- jpype1=0.6.3        # Databases
- protobuf=3.5        # Serialization for deprecated Python nodes
```

The above configuration files only contain the Python packages that the KNIME Python Integration depends on. If you want to use more 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 `py3_knime.yml` and download it 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 Windows Search), change the directory to the folder that contains the configuration file and execute the following command:

```
conda env create -f py3_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). It also 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.

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

Manually installing additional Python packages

The Anaconda configuration files listed above only contain the packages to be installed so that the KNIME Python Integration works 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 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.

Troubleshooting

Mac Matplotlib

On Mac, there may be issues with the `matplotlib` package. The following error:

```
libc++abi.dylib: terminating with uncaught exception of type NSException
```

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 haven't set up Anaconda and/or the recommended Python environment yet, please refer to the Anaconda Setup guide.

Installation

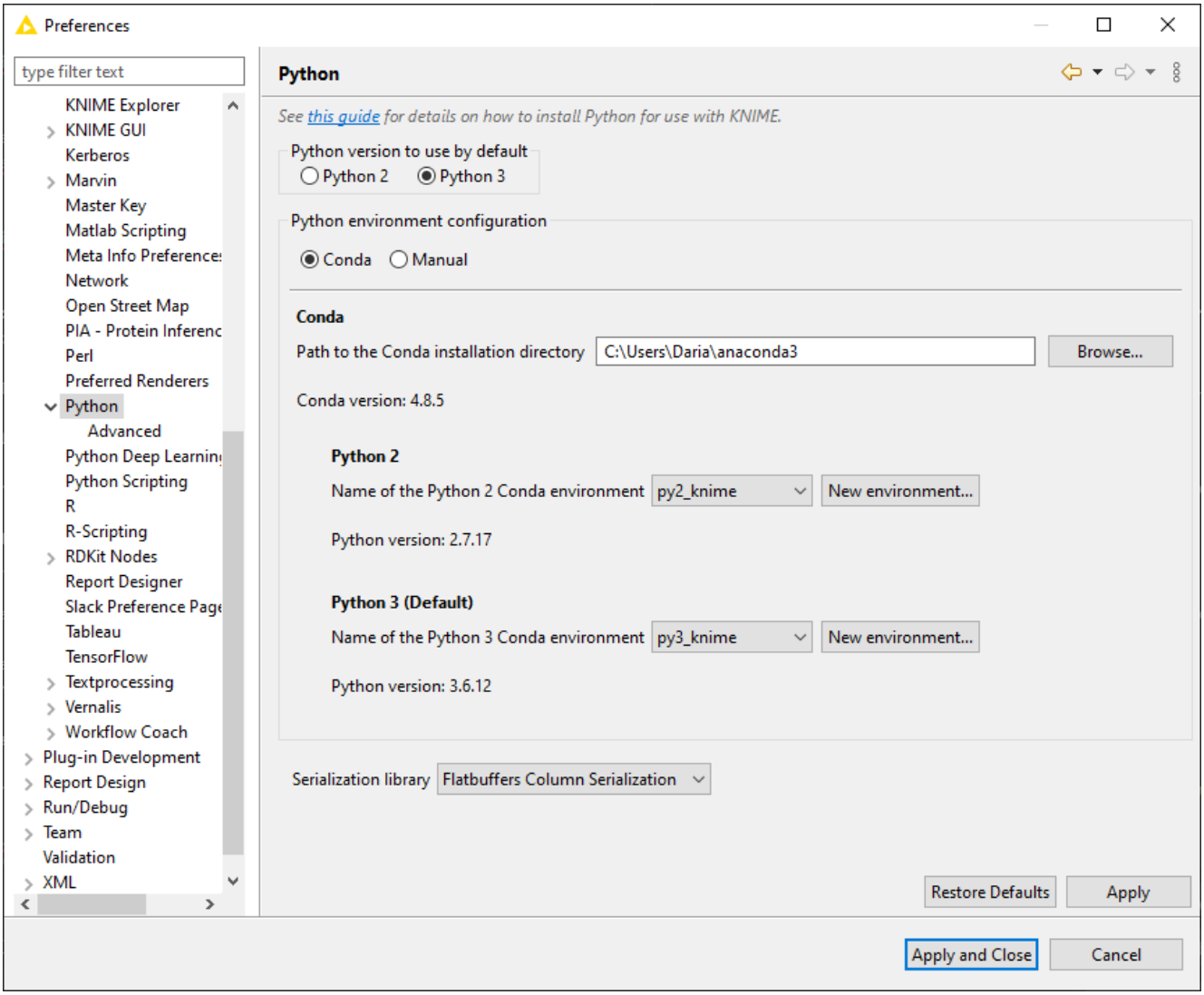
From KNIME Analytics Platform, install the KNIME Python Integration by going to *File* → *Install KNIME Extensions*. The KNIME Python Integration can be found under *KNIME & Extensions* or by entering *Python Integration* into the search box.

Configure the KNIME Python Integration

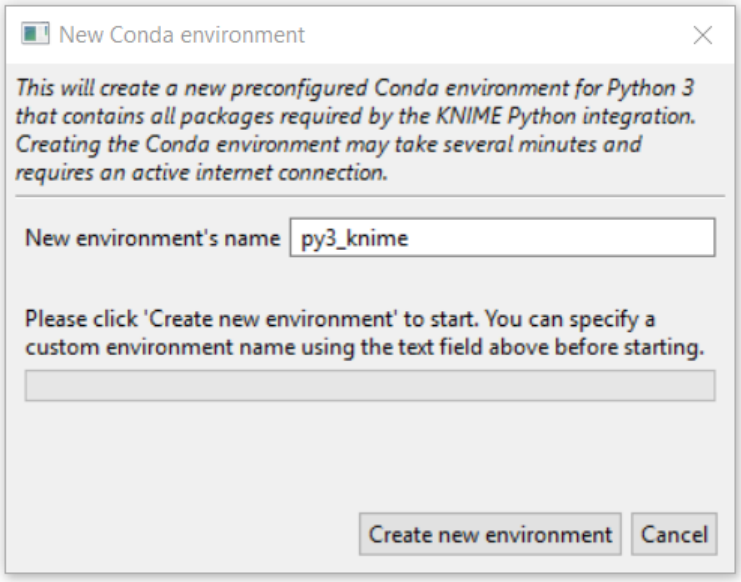
Now tell KNIME which Python environment should be used. Go to the Preference page of the KNIME Python Integration located at *File* → *Preferences*, and then select *KNIME* → *Python* from the list on the left. A dialog opens giving you two options for configuring the Python environment:

Option 1: Conda (recommended)

Select **Conda** under *Python environment configuration*. The dialog should look like the screenshot shown below.



In this dialog, provide the path to the folder containing your Anaconda installation (the default installation path is documented here). Once you have entered a valid path, the installed conda version is displayed and KNIME automatically checks for all available conda environments. Underneath the conda version number, you can choose which conda environment should be used for Python 3 and Python 2 by selecting it from a combo box. If you have already set up a Python environment containing all the necessary dependencies for the KNIME Python Integration, just select it from the list and you are ready to go. If you do not have a suitable environment available, click the **New environment…** button. This opens the following dialog:



Provide a name for the new environment and click the **Create new environment** button. This creates a new `conda` environment containing all required dependencies for the KNIME Python Integration.

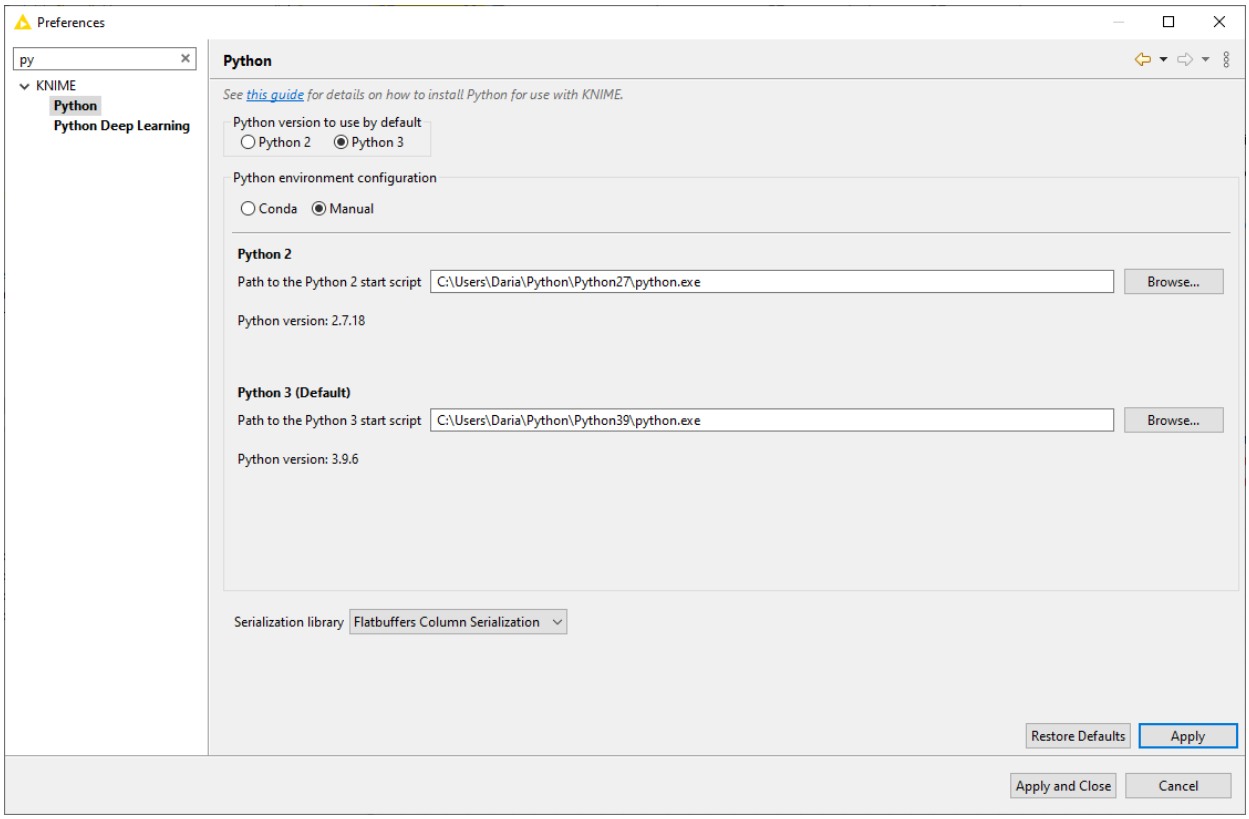
Depending on your internet connection, the environment creation may take a while as all packages need to be downloaded and extracted.

Once the environment is successfully created, the dialog closes and the new environment is selected automatically. If everything worked out fine, the Python version is now shown below the environment selection and you are ready to go.

Option 2: Manual

If you choose the manual setup, you have the following options:

1. Point KNIME to a Python executable of your choice



2. Point KNIME to a start script which activates the environment you want to use for Python 2 and Python 3 respectively. This option assumes that you have created a suitable Python environment earlier with a Python virtual environment manager of your choice.
   In order to use the created environment for the KNIME Python Integration, you need to create a start script (shell script on Linux and Mac, bat file on Windows).
   The script has to meet the following requirements:

   - It has to start Python with the arguments given to the script (please make sure that spaces are properly escaped)
   - It has to output standard and error out of the started Python instance
   - It must not output anything else.

   Here we provide an example shell script for the Python environment on Linux and Mac. Please note that on Linux and Mac you additionally need to make the file executable (i.e. `chmod gou+x py3.sh`).


```
#! /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 are example scripts for conda. You may need to adapt them for other tools by replacing the conda-specific parts and you will need to edit them in order to point to the location of your environment manager installation and to `activate` the correct environment.

After creating the start script you have to point KNIME to its location. In the Python Preferences page of KNIME select the option *Manual* and add the path to the script.

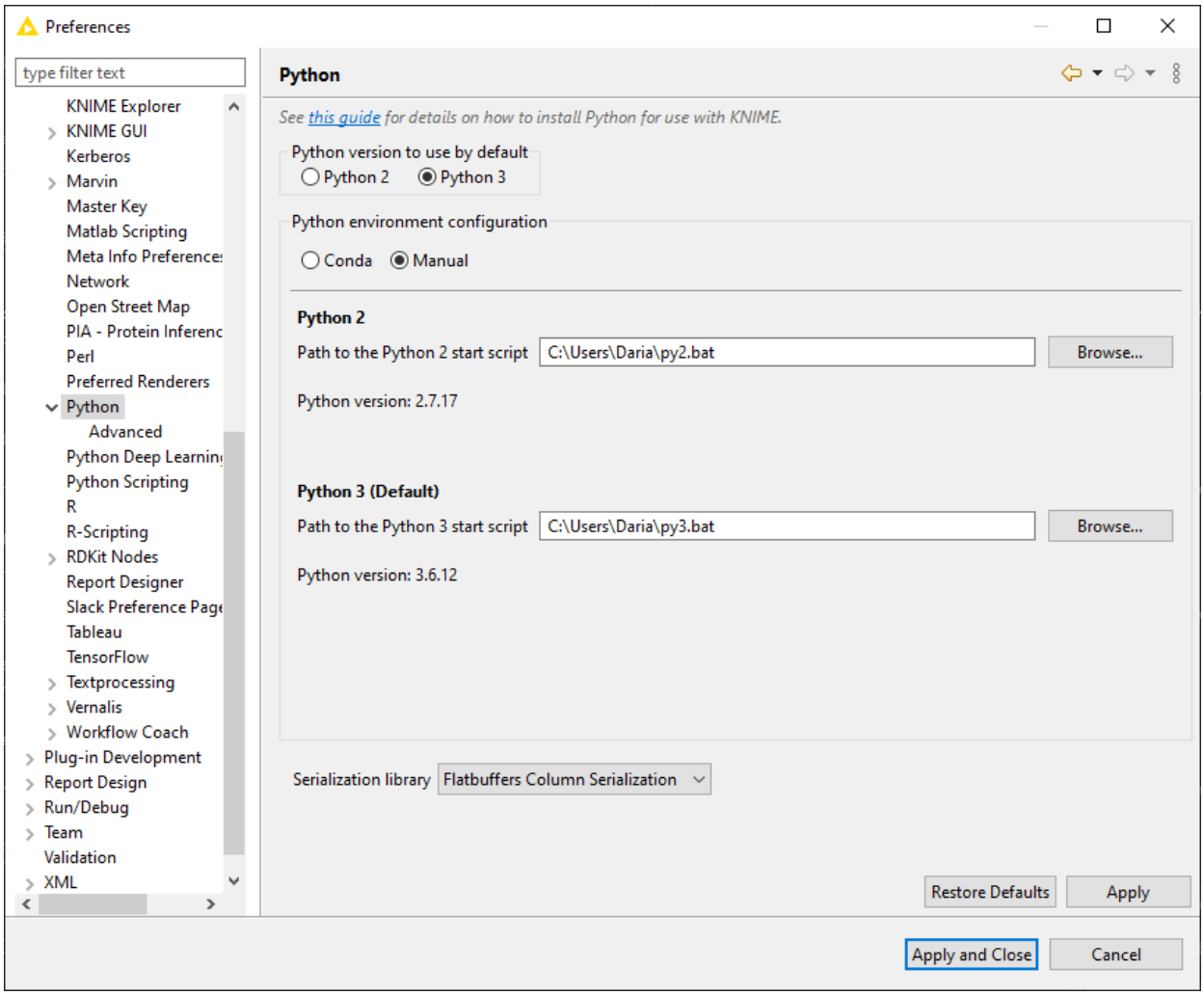


Figure 1. KNIME Python Preferences page. Here you can set the path to the executable script that launches your Python environment.

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 is set correctly, the Python version is now shown in the dialog window and you are ready to go.

Serialization library

You can choose which serialization library should be used by the KNIME Python Integration to transfer data from KNIME Analytics Platform to Python.

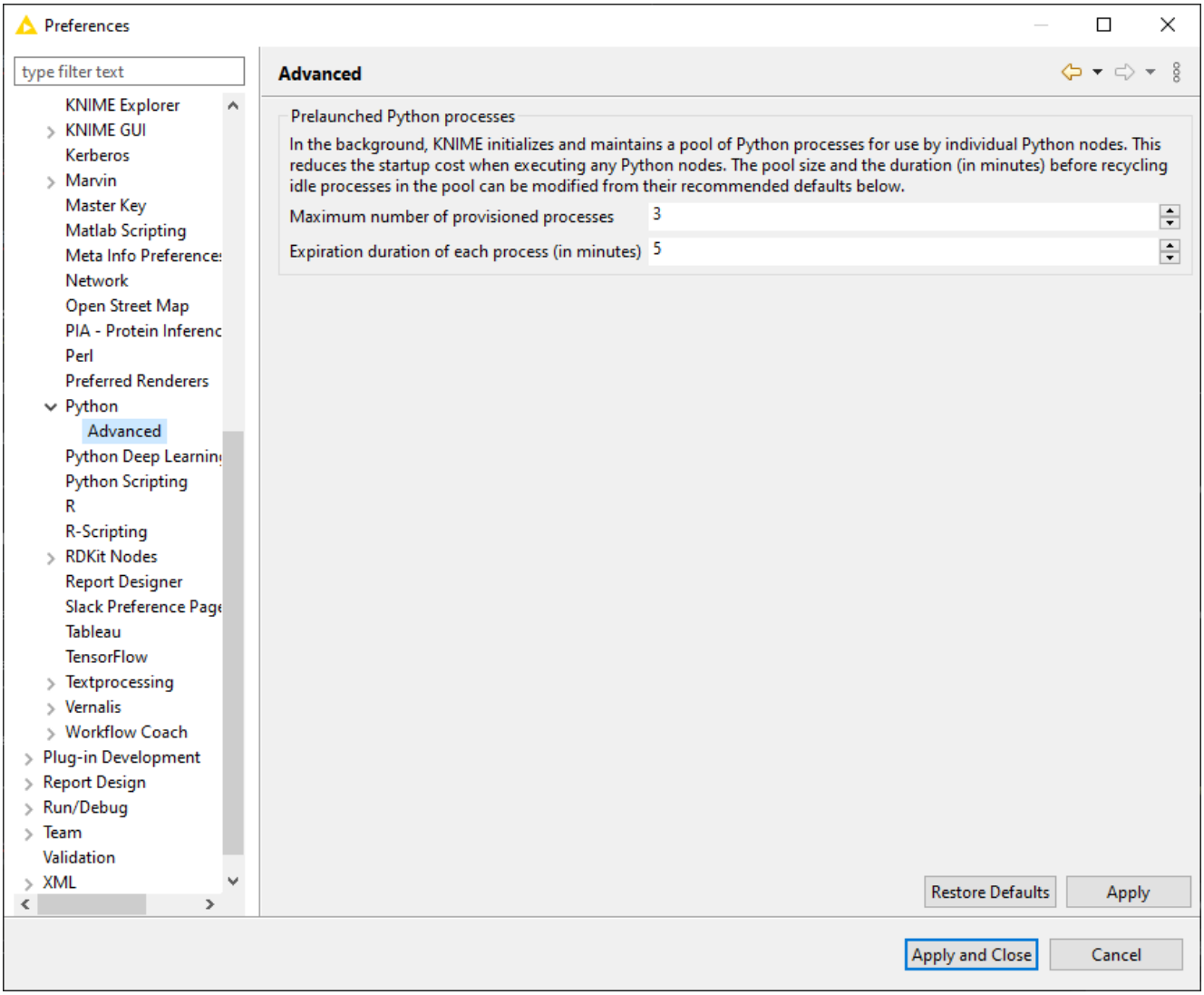
This option does not usually need to be changed and can be left as the default.

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 `4.0.1`
- *CSV (Experimental)*: depends on `pandas` version `0.23`

Advanced

A further *Advanced* option is also available to set up the options of the pre-launched Python processes. In the background, KNIME Analytics Platform initializes and maintains a pool of Python processes that can be used by individual Python nodes, reducing the startup cost when executing any Python nodes. Here, you can set up the pool size in the field *Maximum number of provisioned processes* and the duration in minutes before recycling idle processes in the pool in the field *Expiration duration of each process (in minutes)*.



Configure and export Python environments

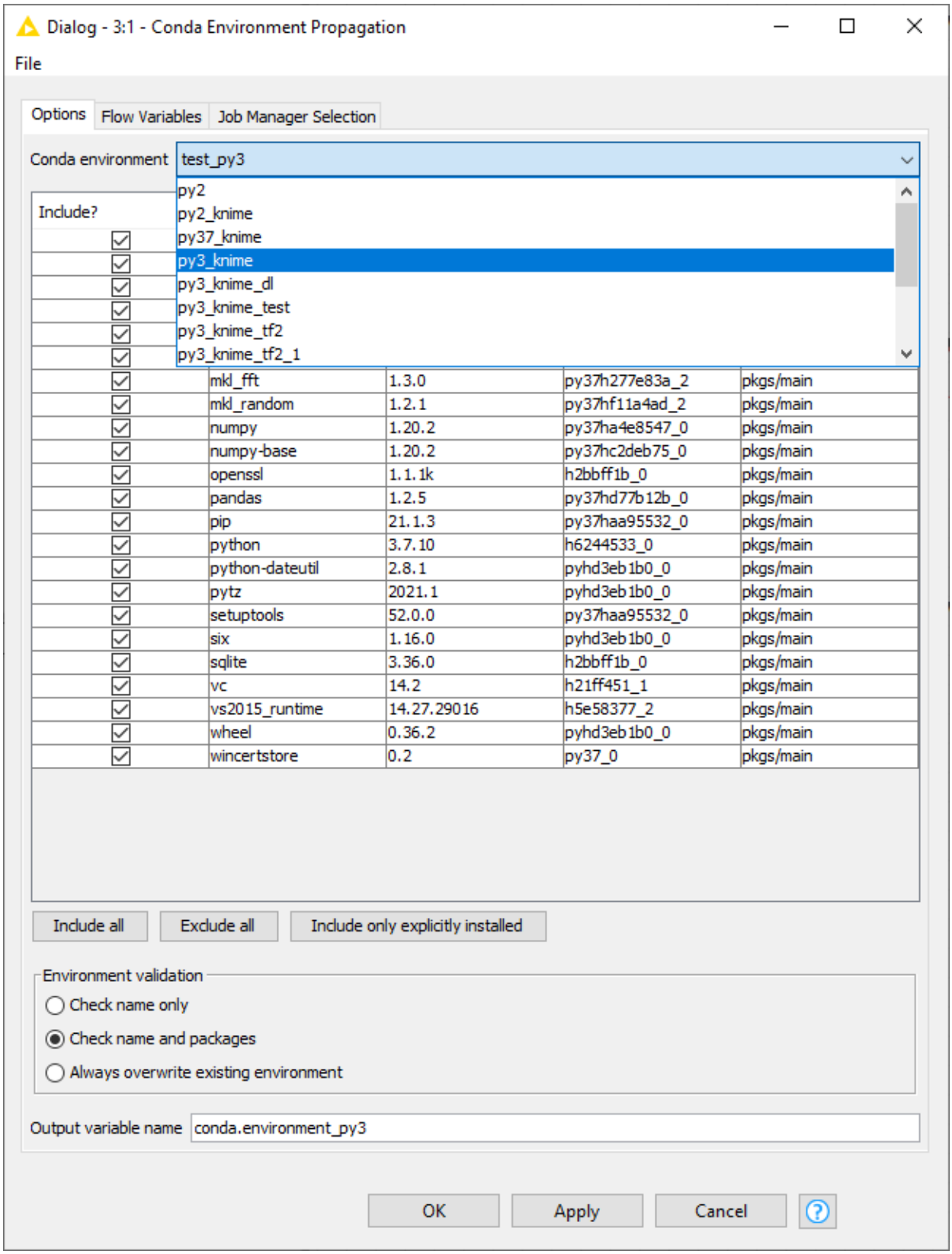
Besides setting up Python for your entire KNIME workspace via the Preference page, you can also use the Conda Environment Propagation node to set up specific Python environments to propagate the environment to downstream Python nodes. This node also allows you to export the specific Python environment together with your workflows.

This node is also useful to make workflows that contain Python nodes more portable by allowing to recreate the conda environment used on the source machine (for example your personal computer) on the target machine (for example a KNIME Server instance).

Configure the Python environment with Conda Environment Propagation node

To configure the Conda Environment Propagation node follow these steps:

1. On your local machine, you need to have conda set up and configured in the Preferences of the KNIME Python Integration as described in the Anaconda Setup section
2. Open the node configuration dialog and select the conda environment you want to propagate and the packages to include in the environment in case it will be recreated on a different machine

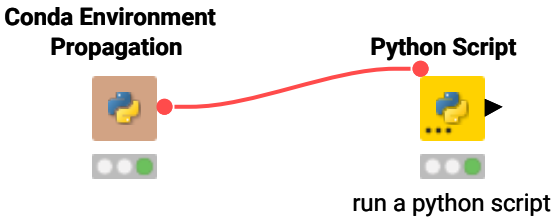


3. The Conda Environment Propagation node outputs a flow variable which contains the necessary information about the Python environment (i.e. the name of the environment and the respective installed packages and versions). The flow variable has `conda.environment` as default name but you can specify a custom name. In this way you can avoid name collisions that may occur when employing multiple Conda

Environment Propagation nodes in a single workflow.

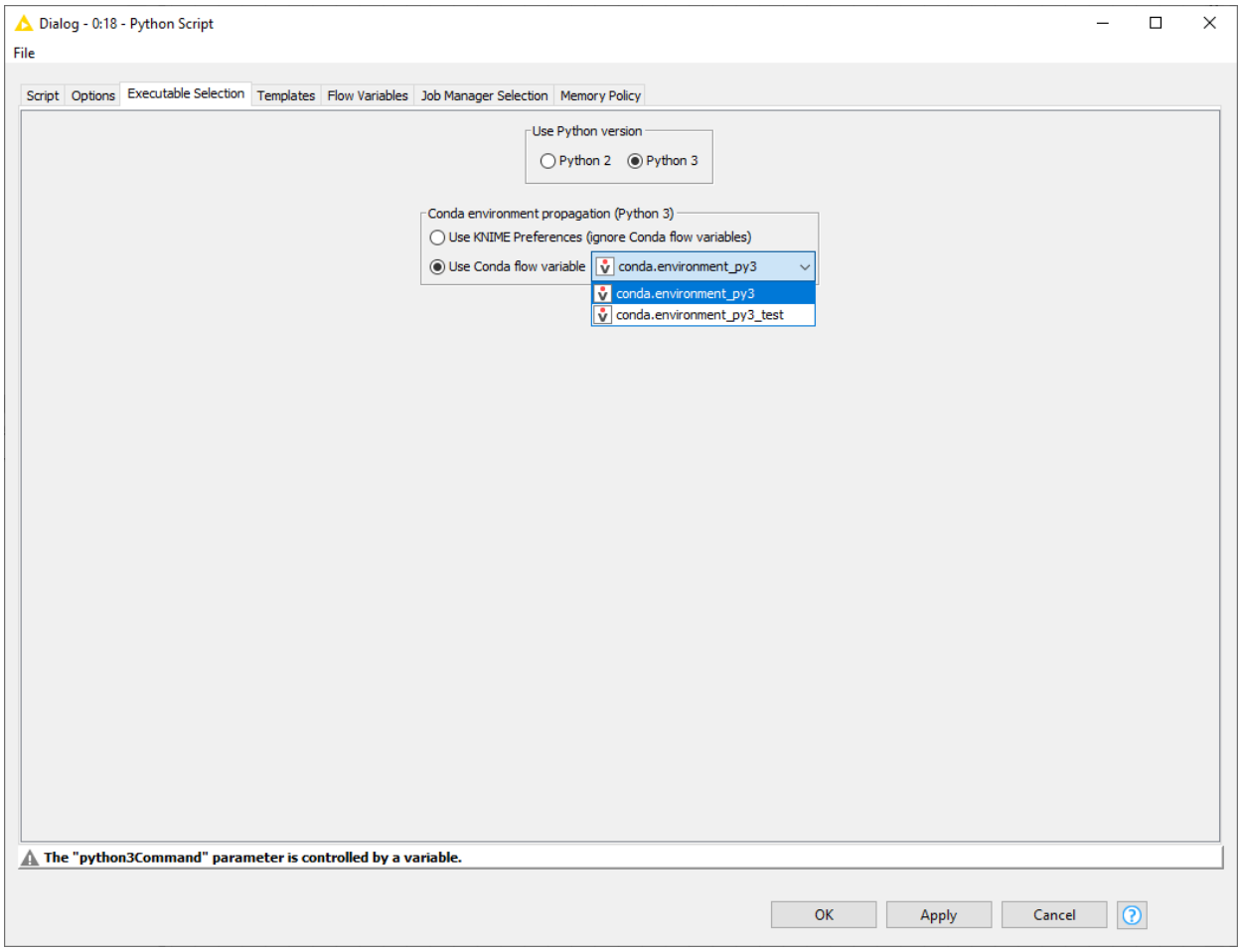
In order for any Python node in the workflow to use the environment you just created you need to:

1. Connect the flow variable output port of Conda Environment Propagation node to the input flow variable port of a Python node



> Please note that, since flow variables are propagated also through connections that are not flow variable connections, the flow variable propagating the conda environment you created with the Conda Environment Propagation node will be available also for all the downstream nodes.

2. Successively open the configuration dialogue of the Python nodes in the workflow that you want to make portable, go to the *Executable Selection* tab, and select:
    a. The Python version to be used by the current node
    b. Whether you want to use the conda flow variable and then select the name of the conda flow variable you want to use, or if you want the node to use the Python environment selected in the KNIME Preferences, which is the default behavior.



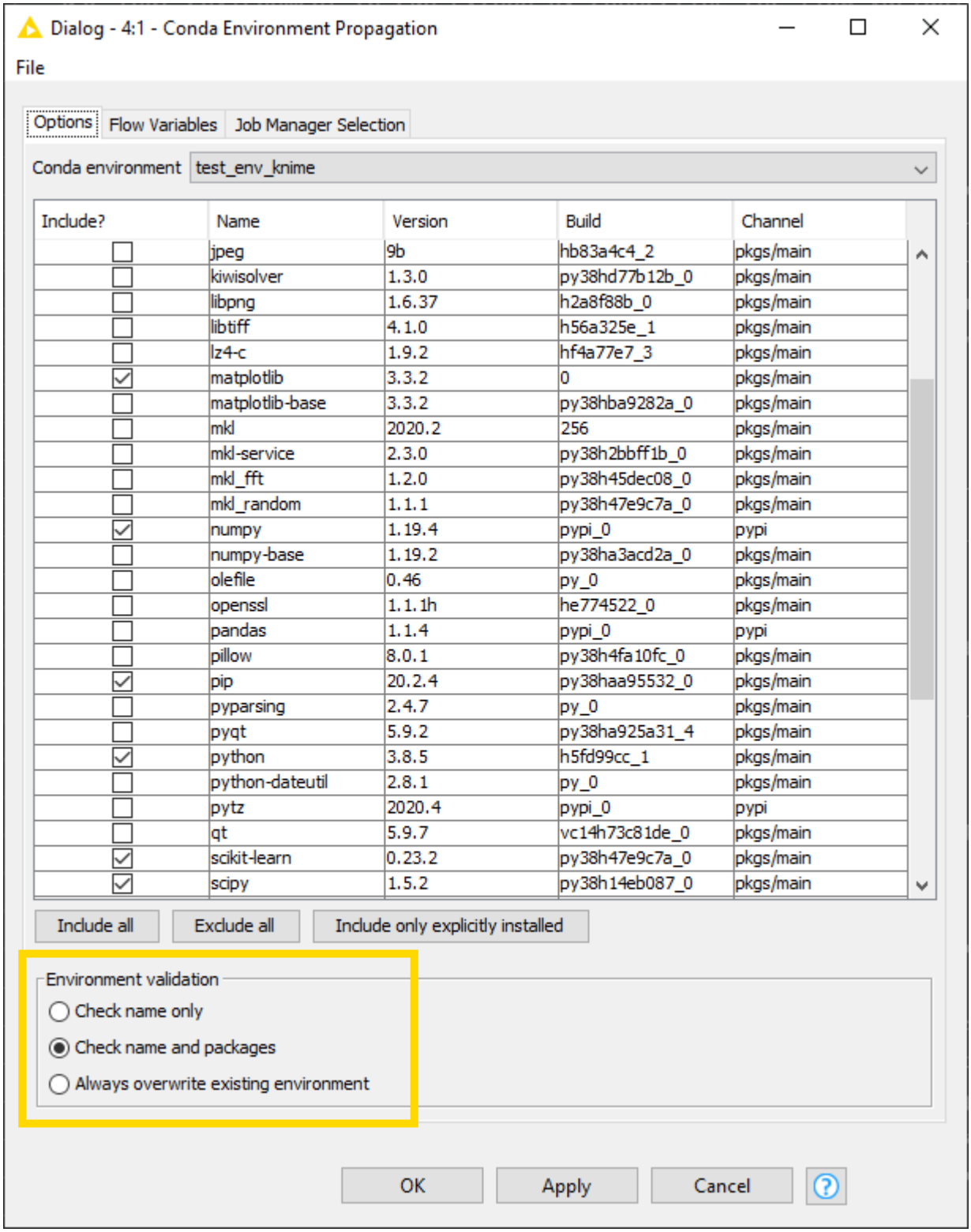
Export a Python environment with a workflow

Once you configured the Conda Environment Propagation node and set up the desired workflow, you might want to run this workflow on a target machine, for example a KNIME Server instance.

1. Deploy the workflow by uploading it to the KNIME Server, sharing it via the KNIME Hub, or exporting it. Make sure that the Conda Environment Propagation node is reset before or during the deployment process.
2. On the target machine, conda must also be set up and configured in the Preferences of the KNIME Python Integration. If the target machine runs a KNIME Server, you may need to contact your server administrator and/or refer to the Server Administration Guide in order to do this.
3. During execution (on either machine), the node will check whether a local conda environment exists that matches its configured environment. When configuring the node you can choose which modality will be used for the conda environment validation on the target machine. *Check name only* will only check for the existence of an environment with the same name as the original one, *Check name and packages* will check both name and requested packages to correspond, while *Always overwrite*

existing environment will disregard the existence of an equal environment on the target machine and will recreate it.

This option will affect the speed of execution of the node as an increasing amount of time is required if the check of the environment is based only on the name of the environment, or if a packages check is also requested.



Please be aware that exporting Python environments between systems that run different Operating Systems might cause some libraries to conflict.

Manual configuration of Python environments per node

In case you do not want to use the Conda Environment Propagation node's functionality, you can also configure individual nodes manually to use specific Python environments. This is done via flow variables `python2Command` and `python3Command` that each Python scripting node offers under the *Flow Variables* tab in its configuration dialog. Both variables accept the path to a Python start script like in the Manual case above. Which of the two flow variables is respected depends on whether a node is using Python 2 or Python 3. This can either be configured via option *Use Python Version* under the *Executable Selection* tab in the node's configuration dialog or via flow variable `pythonVersionOption` which accepts either *python2* or *python3* as value.

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 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.

You can find example workflows using the `knime_jupyter` Python module on our EXAMPLES server.

MDF Reader

Similar to the KNIME Deep Learning Integration, the MDF Reader node requires certain Python packages to be installed in the **Python 3** environment. Since the v4.1 release of

KNIME Analytics Platform, these will be automatically installed if you set up your Python environment via the Python Preference page (see here). The required packages are the following:

```
numpy=1.16.1
libiconv=1.15
asammdf=5.13.13
```

The MDF Reader node requires a newer `numpy` version (`1.16.1`) compared to the numpy version (`1.15`) required before.

KNIME AG
Talacker 50
8001 Zurich, Switzerland
www.knime.com
info@knime.com