

NX12 NXOpen Python with Eclipse/PyDev

Table of Contents

1	Overview.....	2
2	Installing the software.....	3
2.1	Python.....	3
2.2	Eclipse	5
2.3	PyDev	6
3	Configuring the Python Interpreter	8
3.1	Add/Select Python Interpreter.....	8
3.2	Add Interpreter Libraries	10
3.3	Forced Builtins.....	11
3.4	Code Analysis Setting.....	12
4	Creating NXOpen Python projects	13
4.1	Create a new PyDev project.....	13
4.2	Create a Python source code file.....	17
4.3	Typing Python code	18
4.3.1	The 'import' statement	18
4.3.2	Namespace notation.....	19
4.3.3	Type Hinting.....	19
5	Using the PyDev Console.....	21
5.1	Open the PyDev Console.....	21
5.2	Typing Python Code in Console.....	22
6	Debugging NXOpen Python	23
6.1	Debugging with NX GUI.....	24
6.2	Debugging without NX GUI (Batch)	25
7	Caveats	25

1 Overview

This document will guide through the installation of an external Python Distribution with Eclipse and PyDev to provide a featured IDE for writing and debugging NXOpen Python applications, see also related topics:

"Release Notes - Programming Tools – Product Notes":

https://docs.plm.automation.siemens.com/tdoc/nx/12.0.2/release_notes/#uid:index_xid920453:xid920496:xid385122

"Use external Python distribution":

https://docs.plm.automation.siemens.com/tdoc/nx/12/nx_api/#uid:xid1162445:index_nxopen_prog_guide:xid1198245:xid1124918:xid1124926

"Debugging Python applications":

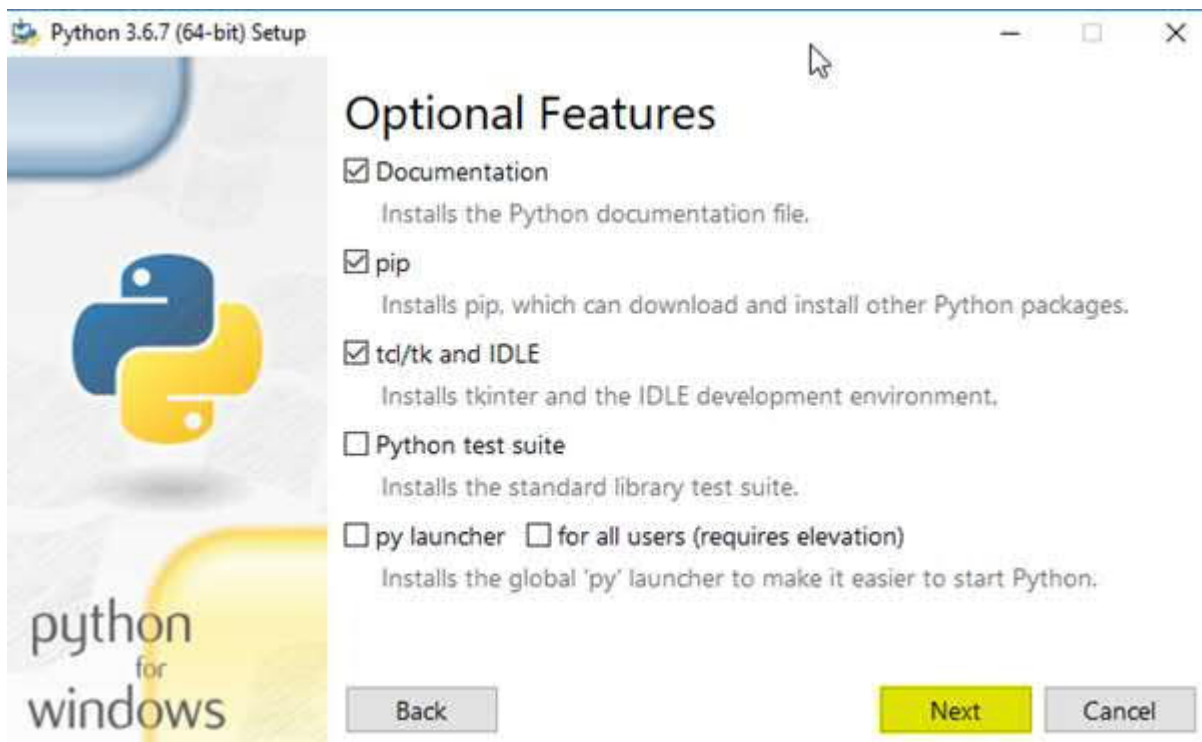
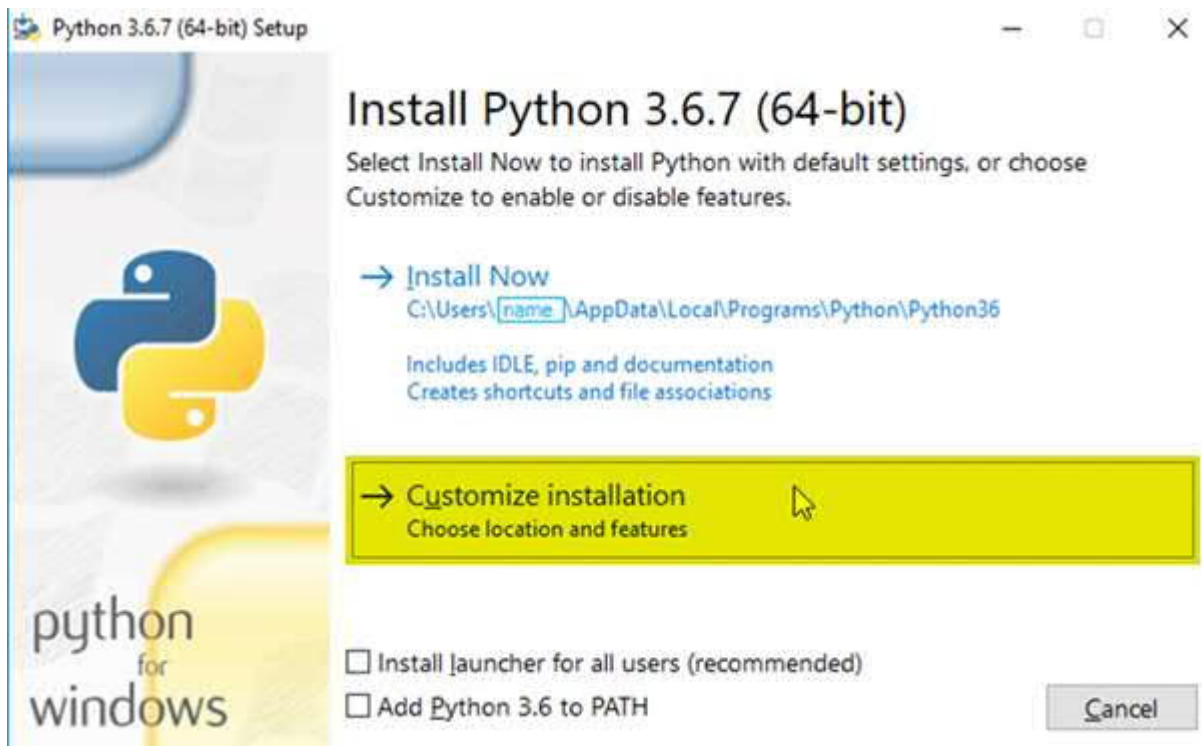
https://docs.plm.automation.siemens.com/tdoc/nx/12/nx_api/#uid:xid1162445:index_nxopen_prog_guide:id1142061:xid1124957

2 Installing the software

2.1 Python

Download and install Python 3.6.x (64bit) from <https://www.python.org/downloads/>.

Choose “Customize Installation” to check and modify the default settings.



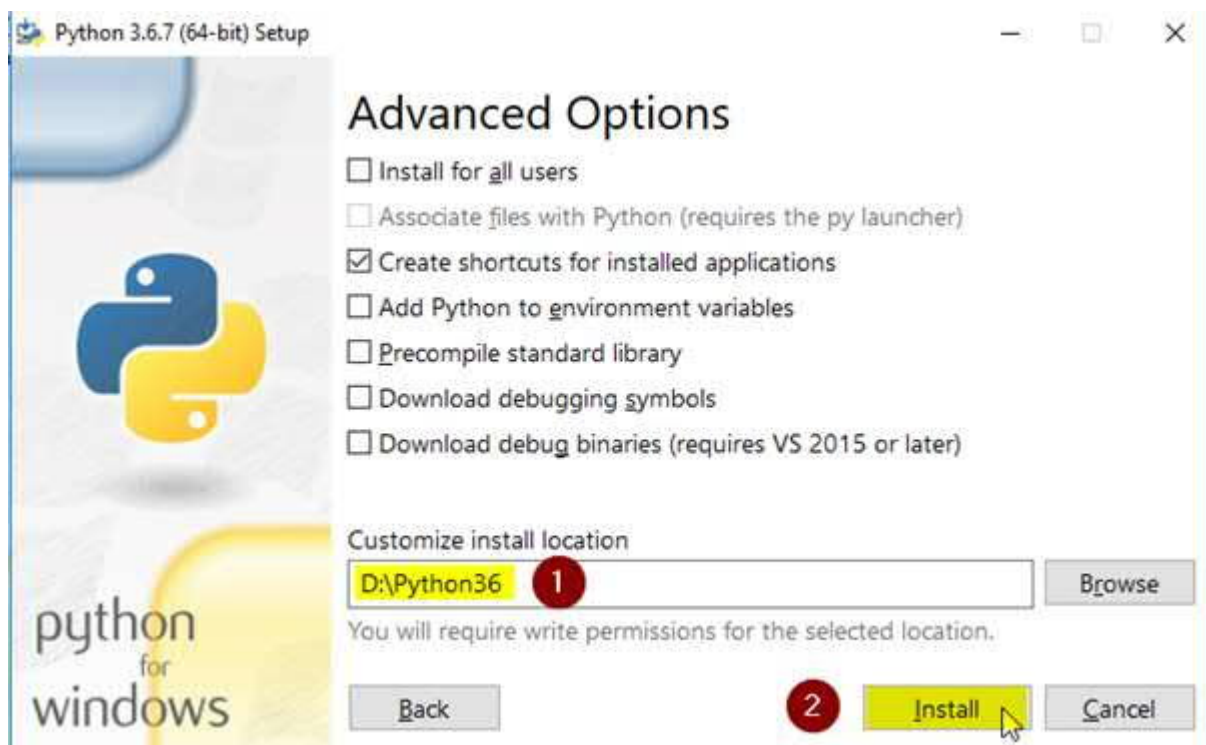
Make sure the option "Install for all users" is turned off to install all files into the same folder.

Otherwise the Python shared library will be copied to "%SystemRoot%\system32\python36.dll" with "system" as owner rather than into the Python installation folder (%PYTHONHOME%).

Defining an external Python distribution(s) is then not straight forward by multiple locations.

Since Python 3.5, "Download debugging symbols" and "Download debug binaries" allow to install symbol PDB files immediately to be used with Visual Studio 2015 and PTVS (not covered in this document).

For this document, we will use the folder D:\Python36 as installation target and this external distribution will be added as Python Interpreter in Eclipse.



2.2 Eclipse

Download the current release of “Eclipse IDE for Java Developers” (64bit) as ZIP package from <https://www.eclipse.org/downloads/packages/> which is recommended due to its extended features.

Extract the ‘eclipse’ folder found in the ZIP file to your desired path, e.g. “D:\eclipse”.

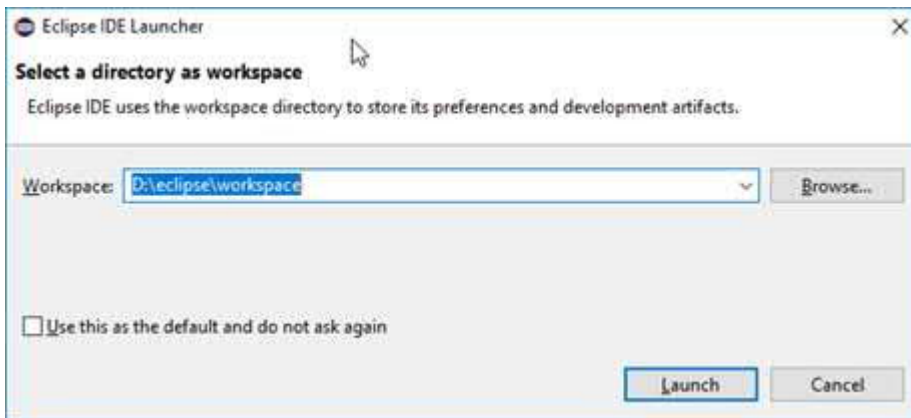
Or for a featured installer instead, refer to <https://www.eclipse.org/downloads/packages/installer> .

Open a NX 12 Command Prompt from the Windows menu to ensure the correct UGII_BASE_DIR and PATH environment variables are set.

Note: Don’t define and set the PYTHONPATH variable to the NXBIN\python folder here to prevent a conflict with the “ctypes” libraries later in the Python Interpreter configuration.

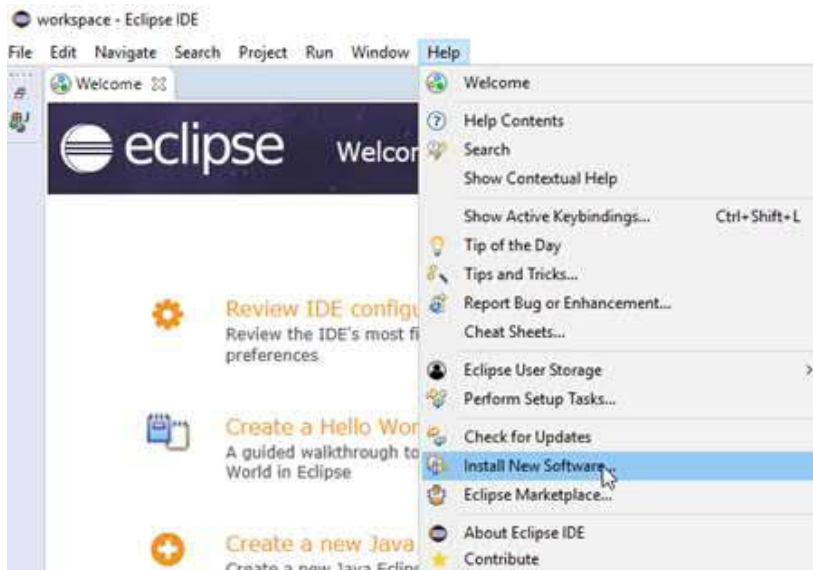
Start the Eclipse IDE by executing the D:\eclipse\eclipse.exe command.

Specify a directory as workspace, e.g. D:\eclipse\workspace and Launch

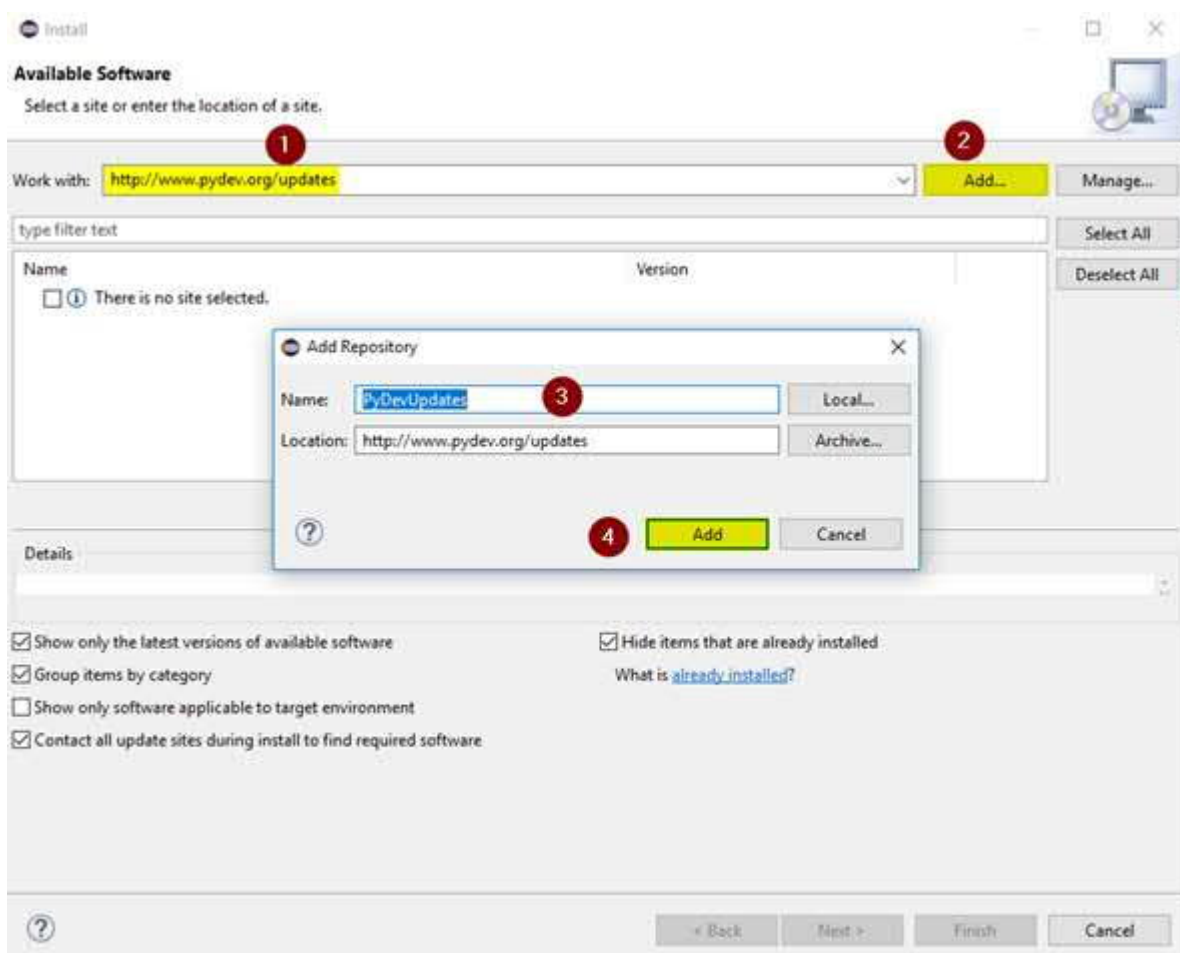


2.3 PyDev

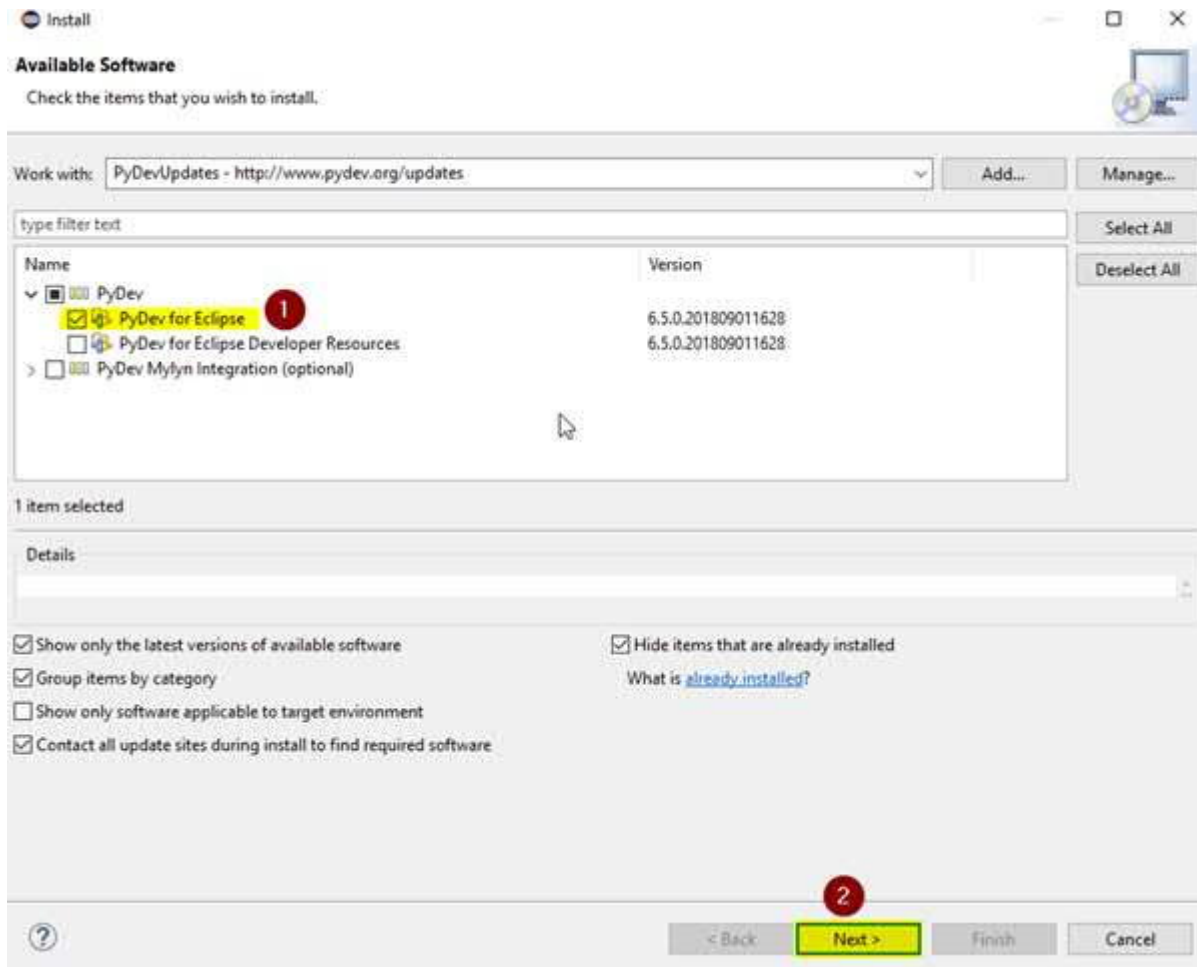
Eclipse -> Help -> Install New Software, see also http://www.pydev.org/manual_101_install.html



Add <http://www.pydev.org/updates> as repository to search for available extensions:



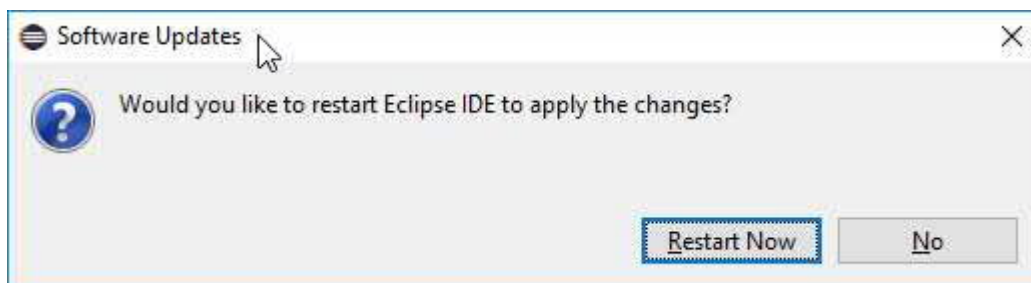
Select the item PyDev for Eclipse -> Next:



Go through the next pages:

- "Review the items to be installed" -> Next
- Turn on "I accept the terms of the license agreement" -> Finish
- Monitor the status line on the lower right side saying "Installing the Software"...

Restart Now finally to apply the changes and restart the Eclipse IDE:



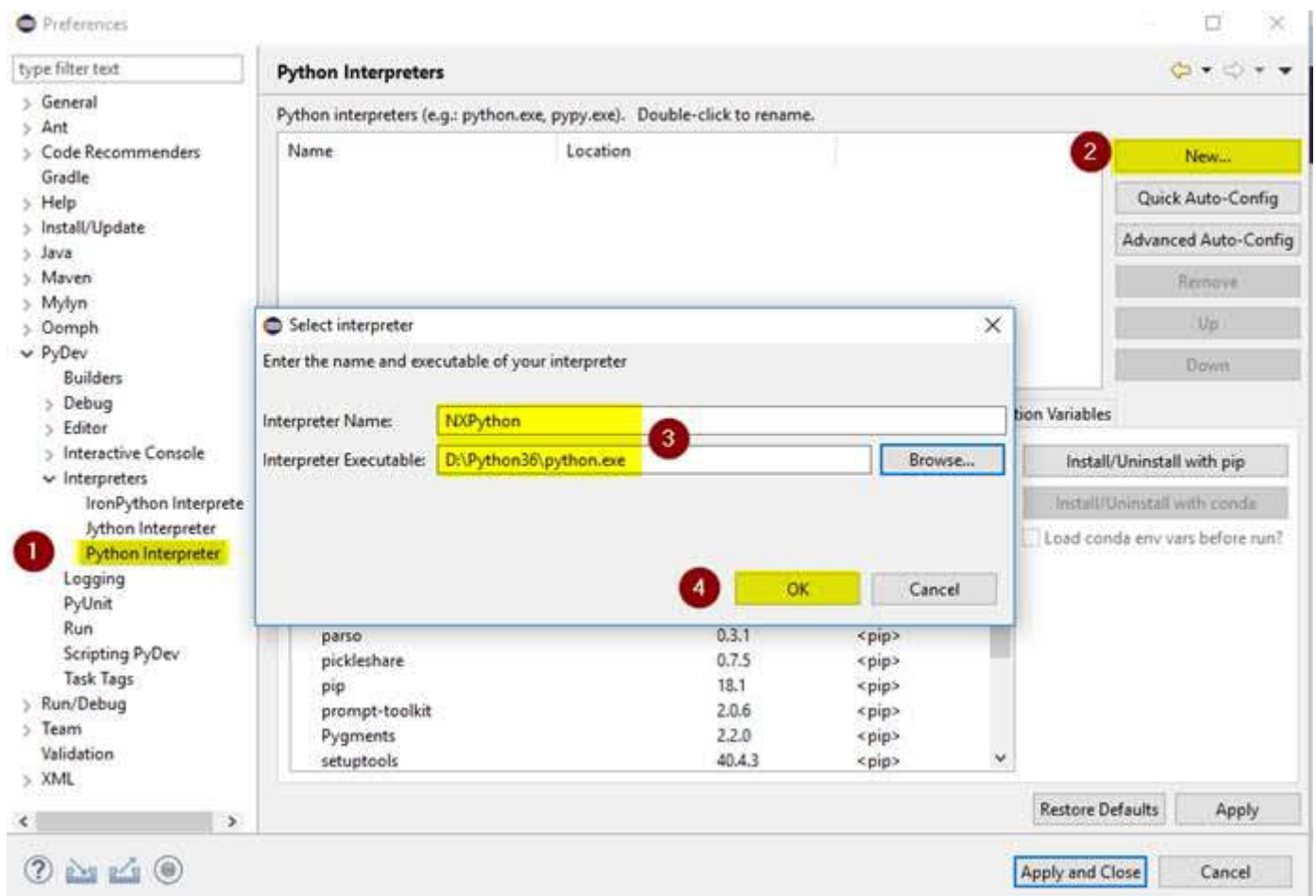
3 Configuring the Python Interpreter

3.1 Add/Select Python Interpreter

The “Quick Auto-Config” command searches for installed Python interpreters in your system and adds them automatically but we want to add it manually with a custom name.

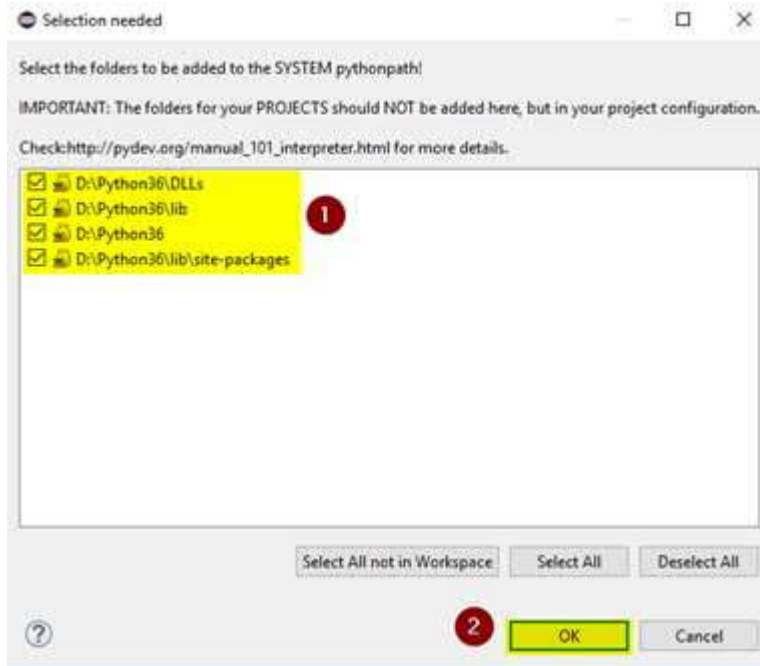
Window -> Preferences -> PyDev -> Interpreters -> Python Interpreter -> New

Enter an interpreter name like “NXPython”, browse to the folder which holds the “python.exe” executable -> OK



PyDev will ask you to add the interpreter sub folders found to the System PYTHONPATH.

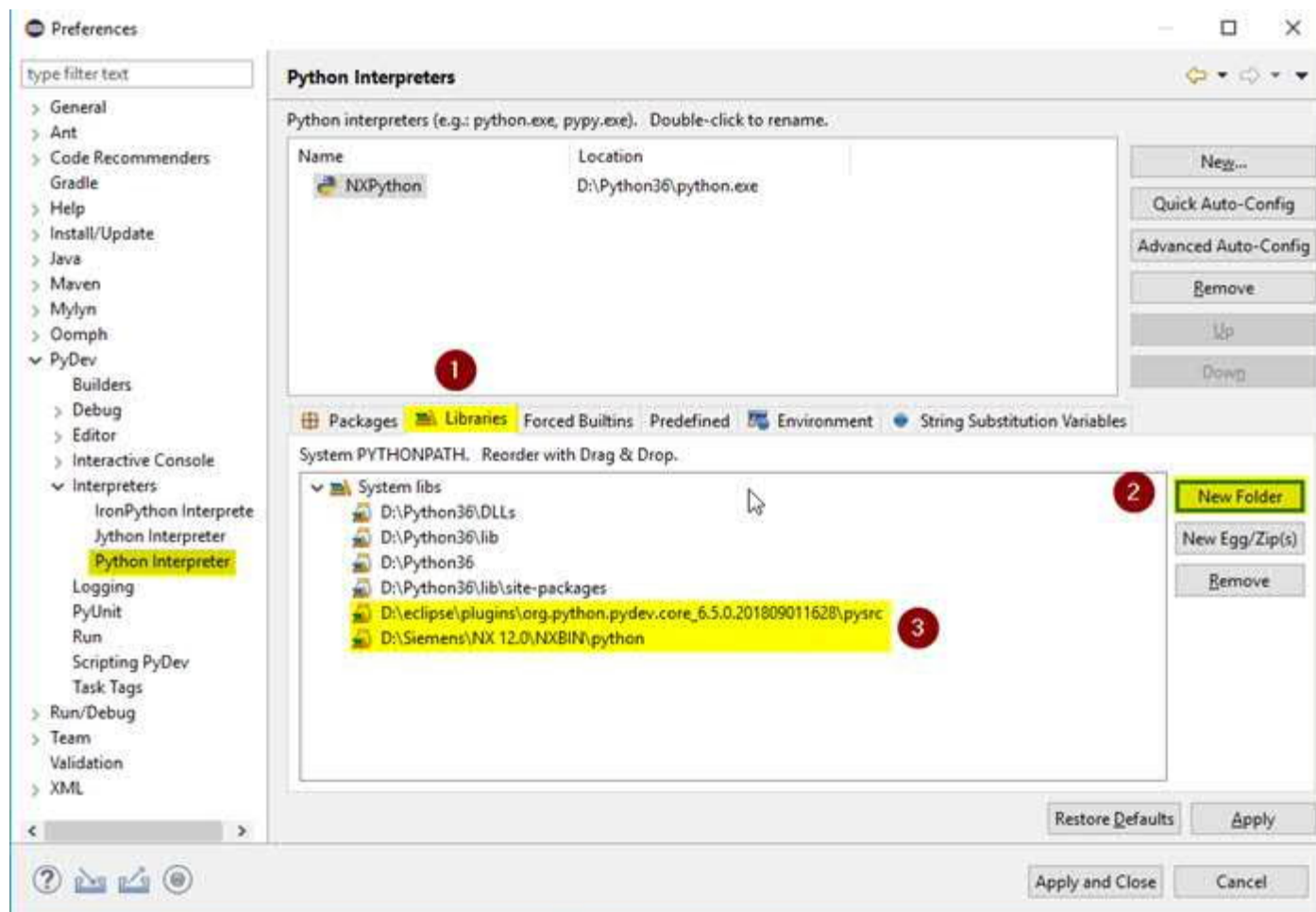
Accept all folders found -> OK



3.2 Add Interpreter Libraries

In the Python Interpreters -> Libraries tab, search and add the following folders:

- The Eclipse plugin PyDev core folder containing the pydevd.py file (for PyDev Debug and Console)
- The NX / NXOpen python installation folder (for NXOpen APIs)



Note: make sure that the NXOpen Python folder is added at the end of the folder list or at least after the Python DLL folder, otherwise the PyDev Console may run into errors due to conflicting "ctypes" class definitions.

"Apply" will finally collect all modules and built the internal PYTHONPATH setting.

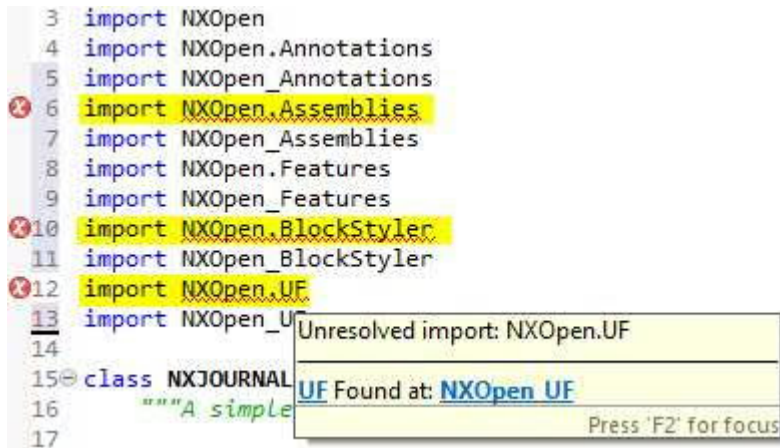
3.3 Forced Builtins

One known caveat in external IDEs is seen by some NXOpen namespace modules misleadingly presenting an “Unresolved import” status using the “.” (dot) notation while it doesn’t happen using the “_” (dash/underscore/underline) syntax.

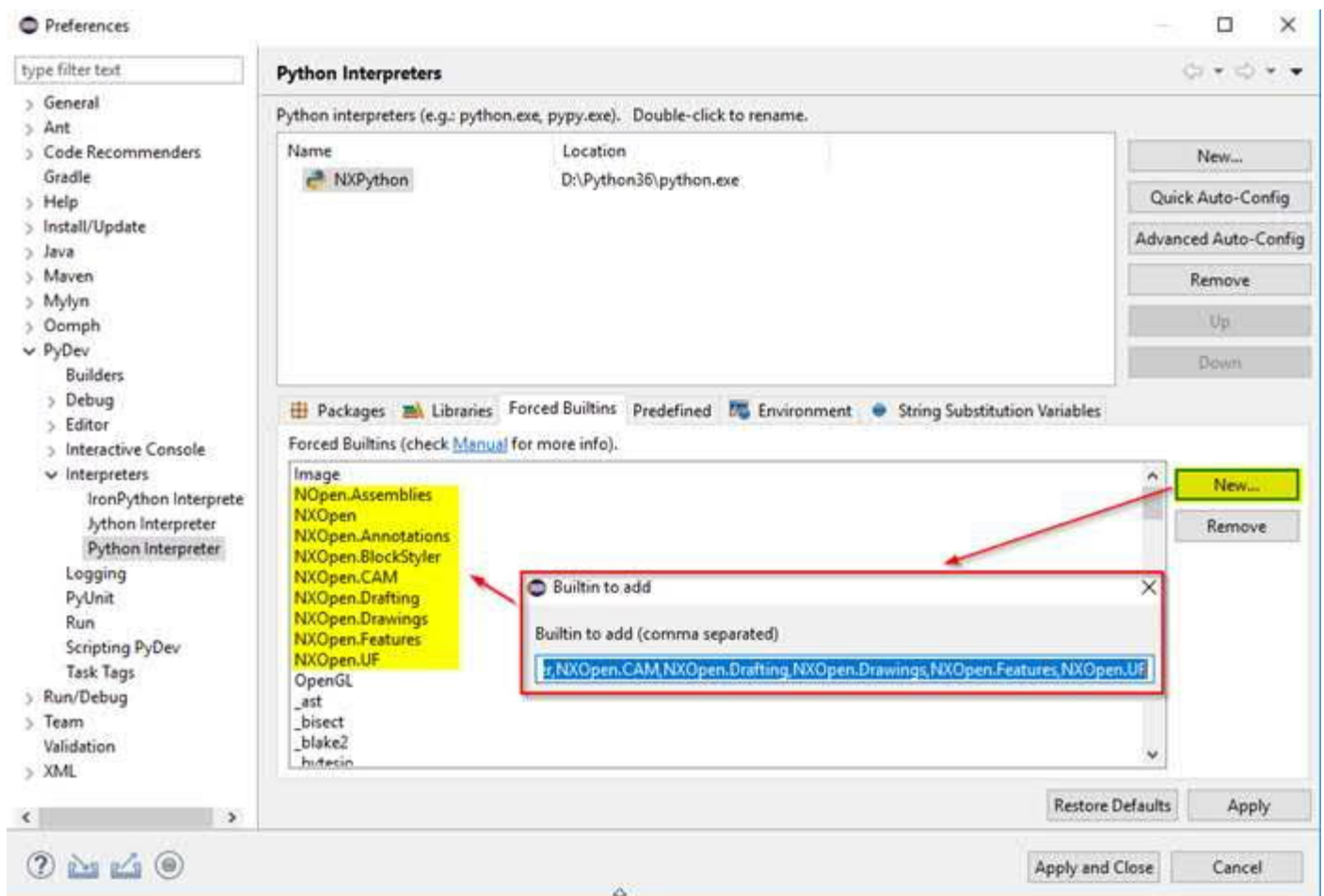
```

3 import NXOpen
4 import NXOpen.Annotations
5 import NXOpen.Annotations
6 import NXOpen.Assemblies
7 import NXOpen.Assemblies
8 import NXOpen.Features
9 import NXOpen.Features
10 import NXOpen.BlockStyler
11 import NXOpen.BlockStyler
12 import NXOpen.UF
13 import NXOpen.UF
14
15 class NXJOURNAL
16     """A simple
17

```

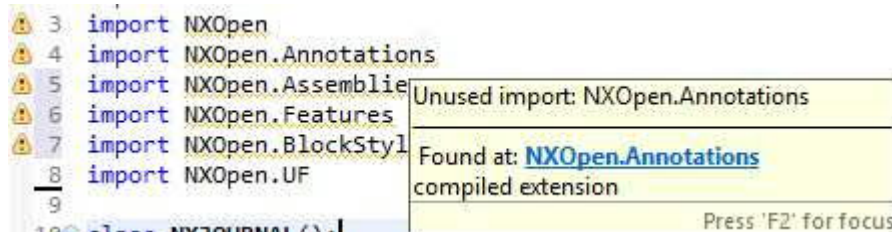


To prevent this issue for the Eclipse Editor and Console using the ‘import’ statement, add the desired NXOpen namespace modules to the “Forced Builtins” list.

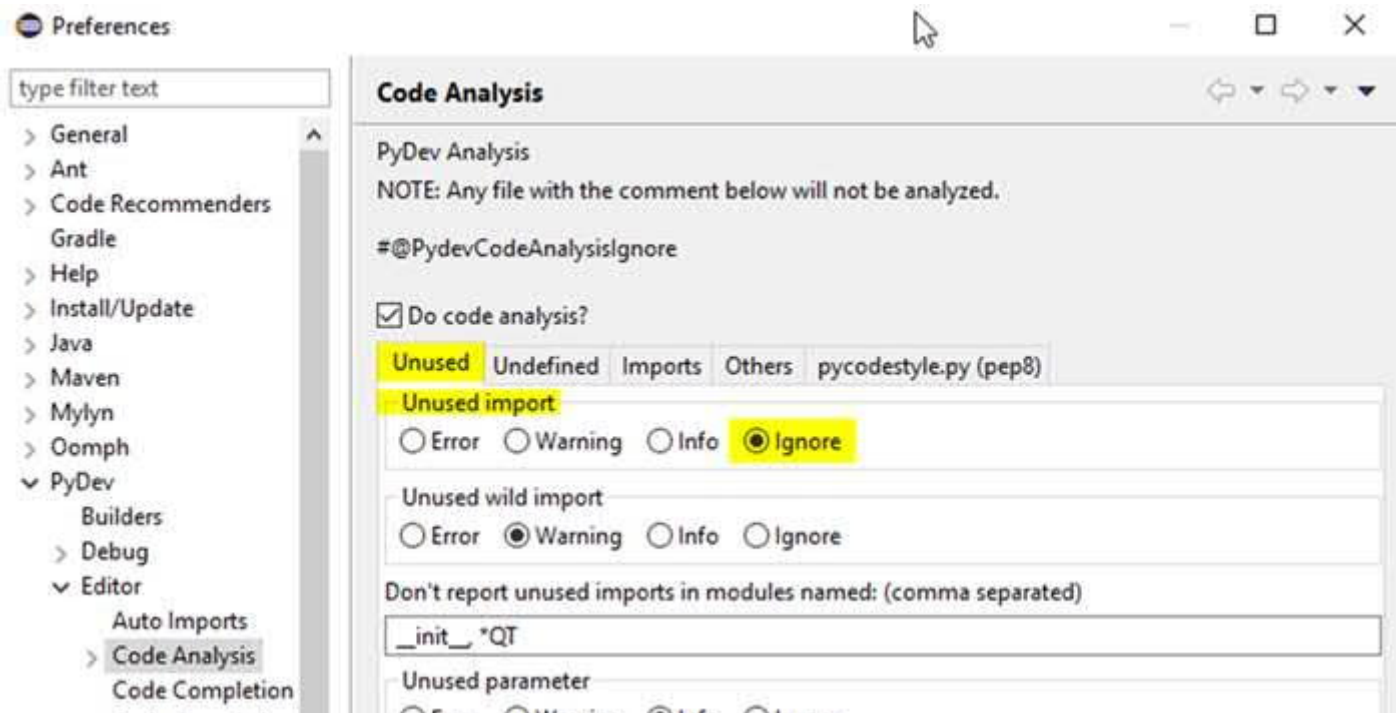


3.4 Code Analysis Setting

Another minor caveat is the “Unused import” warning, for example on “import NXOpen” caused by also importing a sub-module which implies that the parent module will already be imported.



Turn this off by Window -> Preferences -> PyDev -> Editor -> Code Analysis -> Unused -> Unused import -> Ignore



If the Text Editor doesn't update the display with the modified settings fast enough, you may force it on demand.

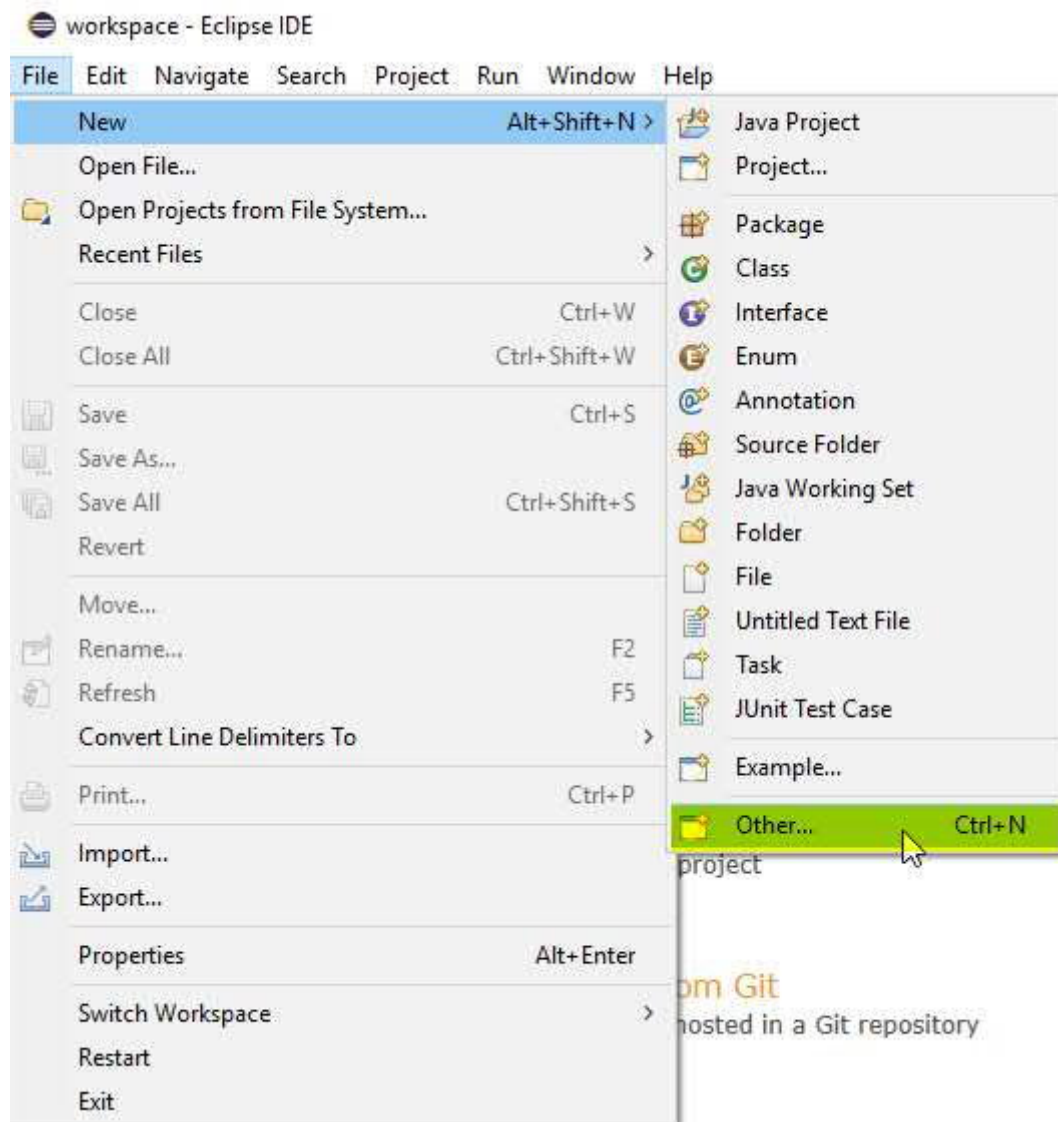
The command MB3 -> PyDev -> Code Analysis can be found by right clicking within the editor code area or on the project name in the PyDev Package Explorer.



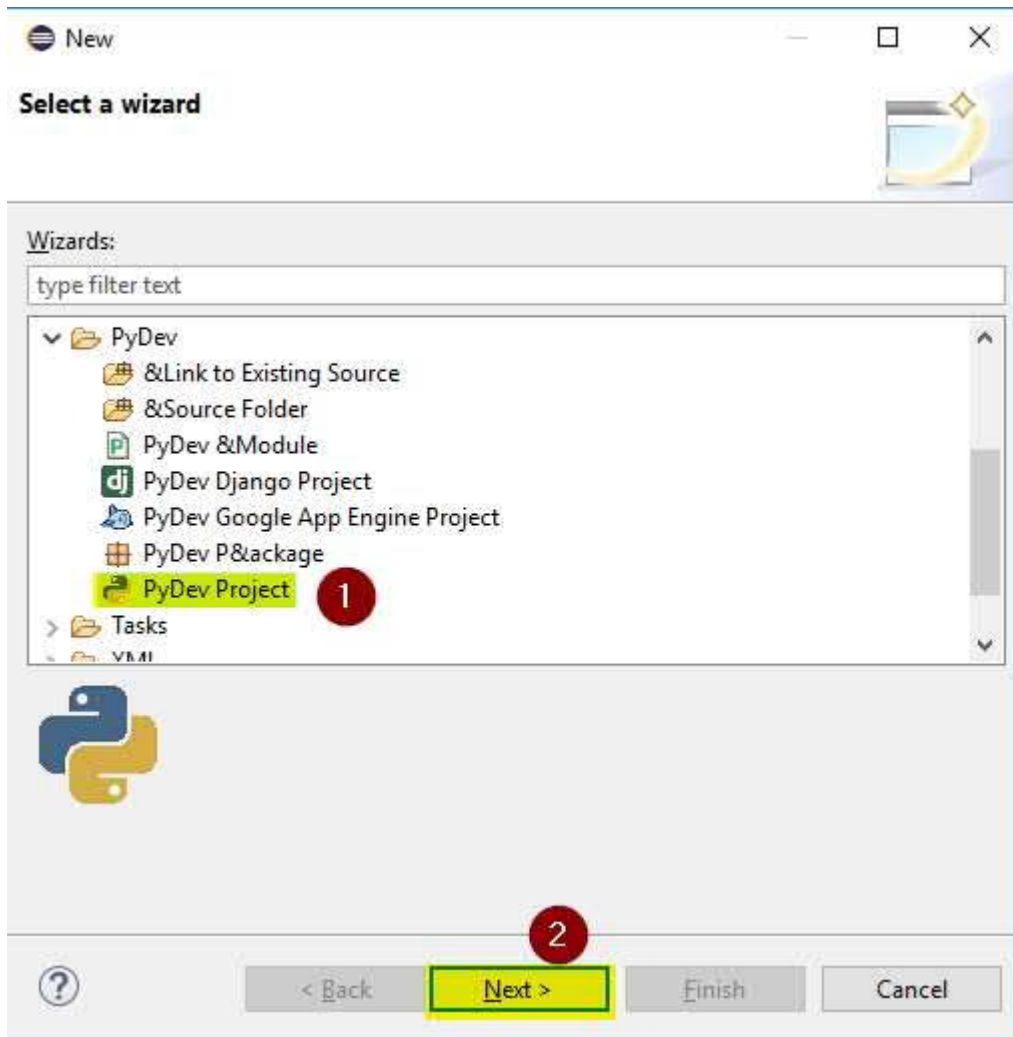
4 Creating NXOpen Python projects

4.1 Create a new PyDev project

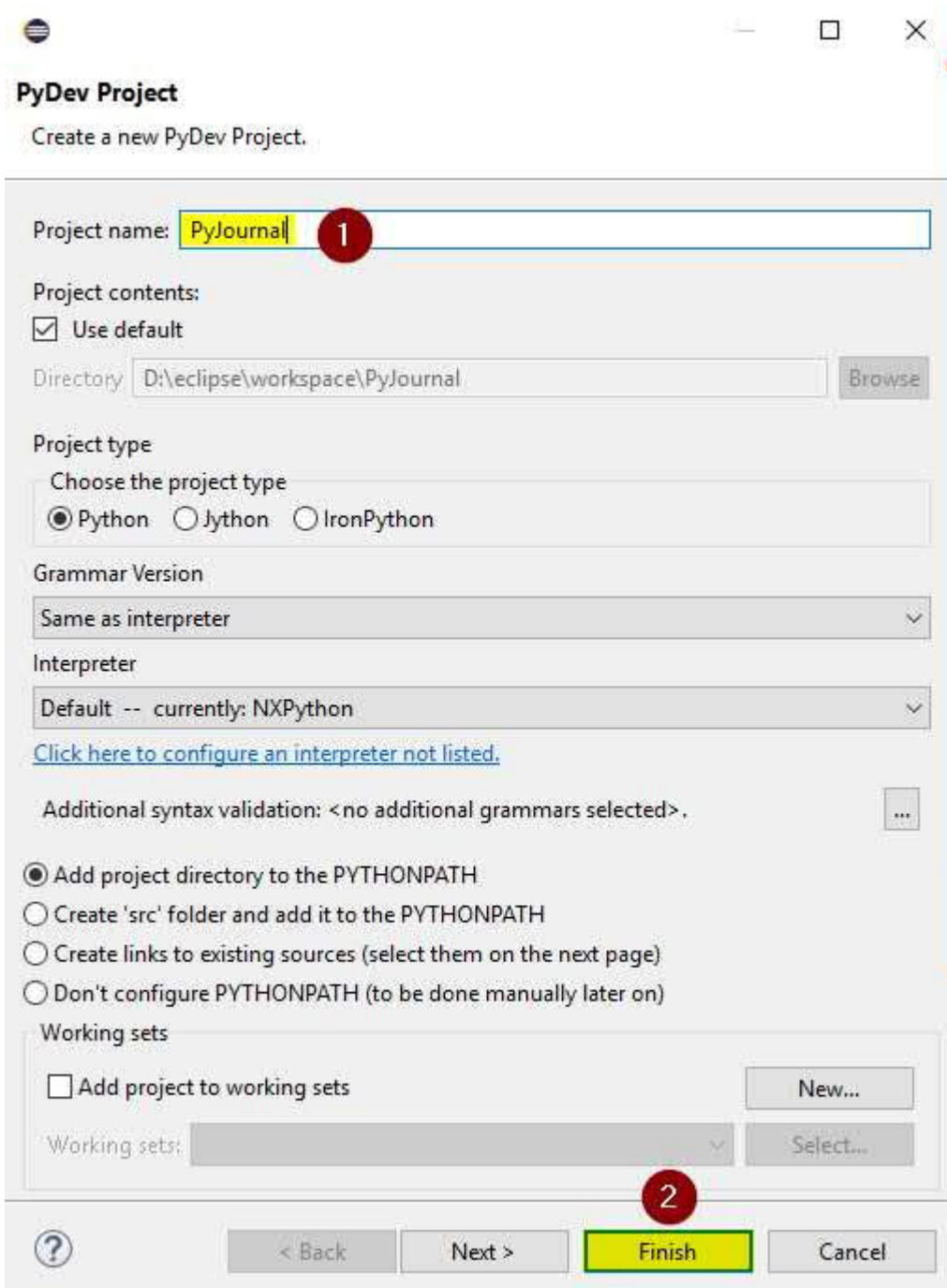
File -> New -> Other



PyDev -> PyDev Project -> Next



Enter a project name, e.g. "PyJournal" -> Finish



PyDev Project
Create a new PyDev Project.

Project name: 1

Project contents:
☒ Use default
 Directory: Browse

Project type
 Choose the project type
☒ Python ☐ Jython ☐ IronPython

Grammar Version

Interpreter

[Click here to configure an interpreter not listed.](#)

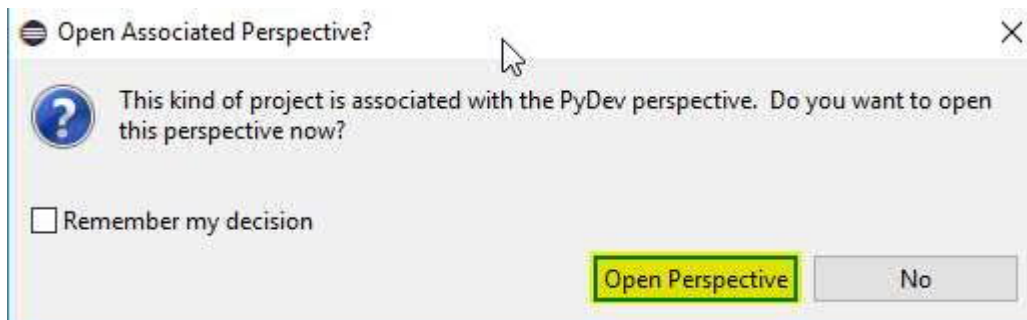
Additional syntax validation: <no additional grammars selected> ...

☒ Add project directory to the PYTHONPATH
☐ Create 'src' folder and add it to the PYTHONPATH
☐ Create links to existing sources (select them on the next page)
☐ Don't configure PYTHONPATH (to be done manually later on)

Working sets
☐ Add project to working sets New...
 Working sets: Select...

2

Confirm Open Associated Perspective to open the PyDev layout immediately

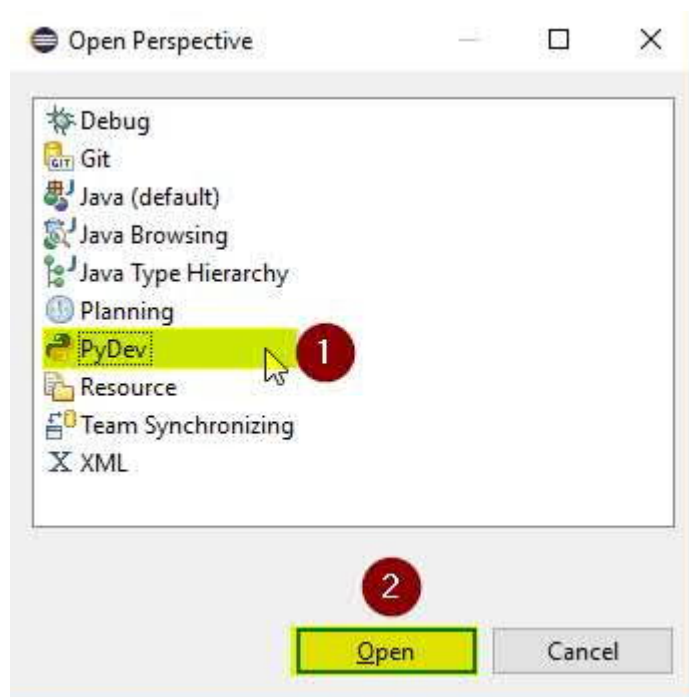
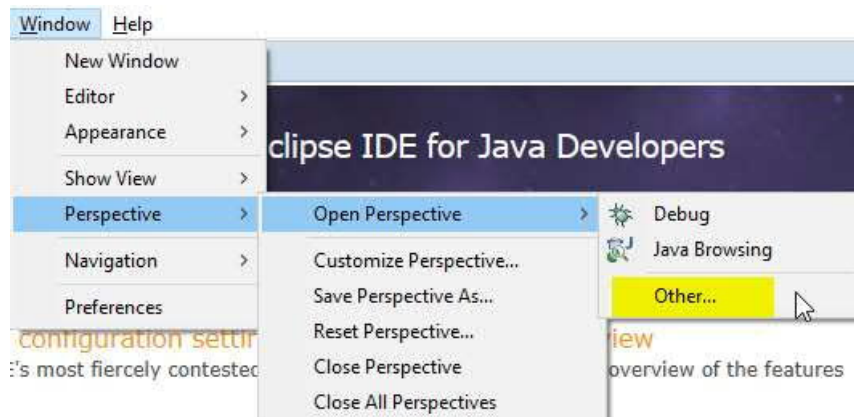


Open Associated Perspective?

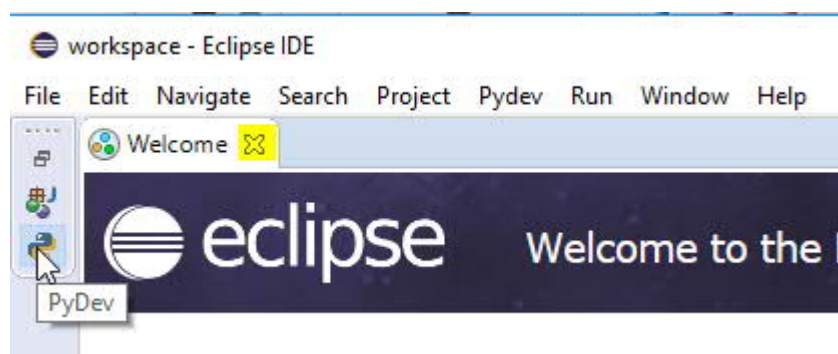
☒ This kind of project is associated with the PyDev perspective. Do you want to open this perspective now?

☐ Remember my decision

Or open the Perspective later by Window -> Perspective -> Open Perspective -> Other -> PyDev

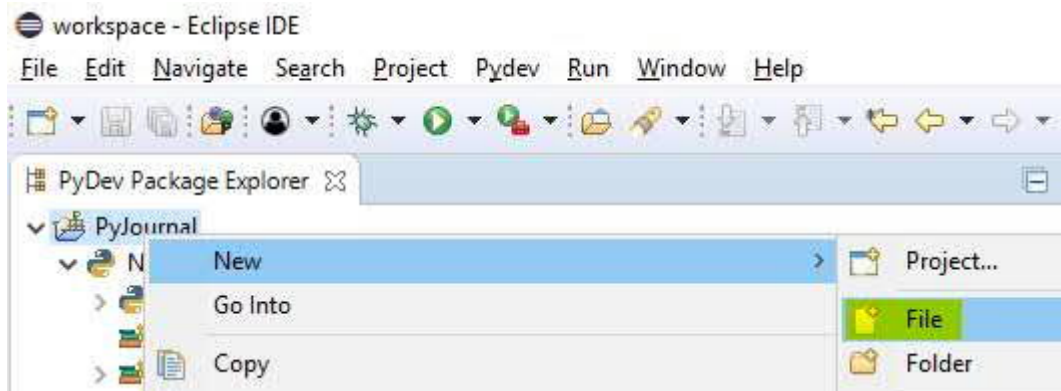


The Perspective Toolbar appears but if you don't see the PyDev Package Explorer layout yet then minimize or close the Welcome page:

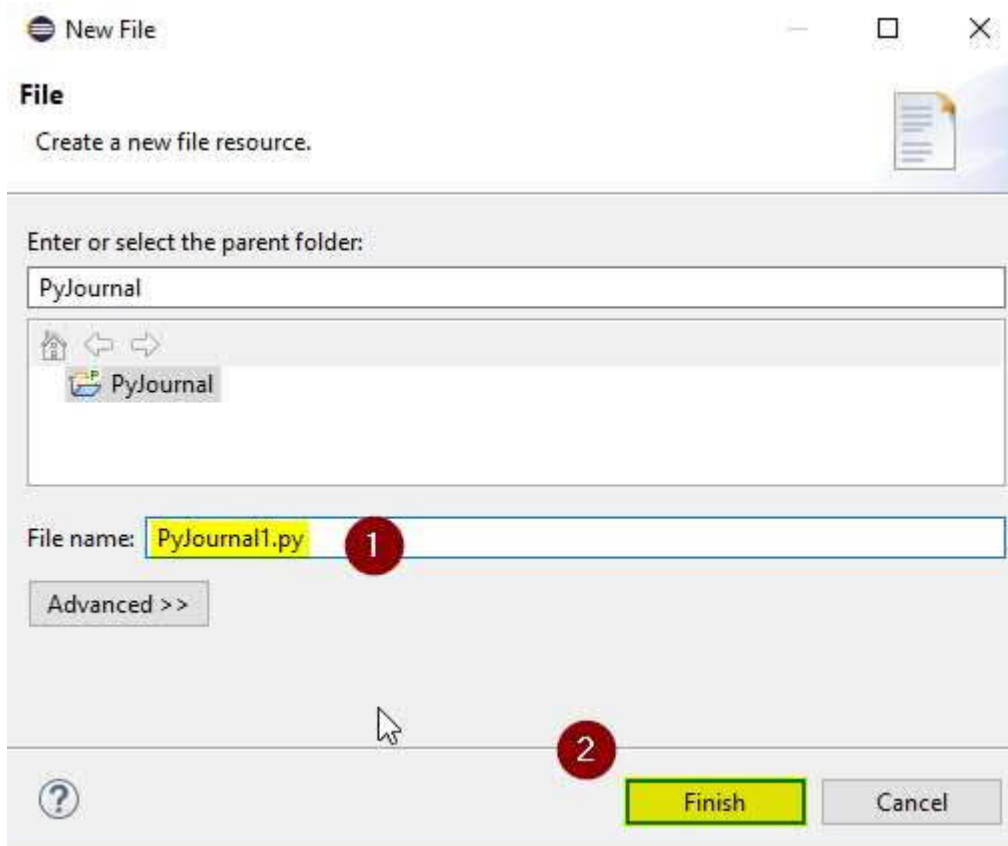


Create a Python source code file

PyDev Package Explorer -> Project name -> MB3 -> New -> File



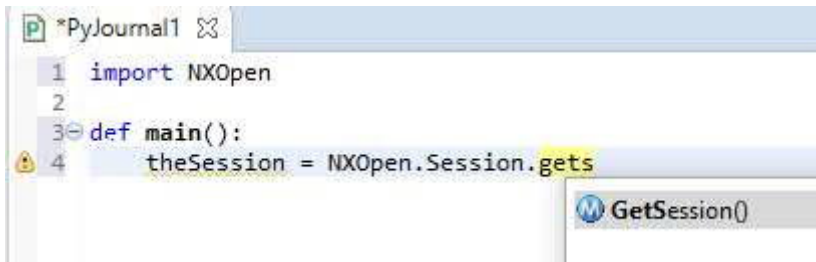
Enter a File Name -> Finish



4.2 Typing Python code

4.2.1 The 'import' statement

The 'import NXOpen' statement imports the NXOpen package but not any of the contained classes yet. Hence, we have to specify the full qualifier 'NXOpen.Session' to see Auto Completion initially.



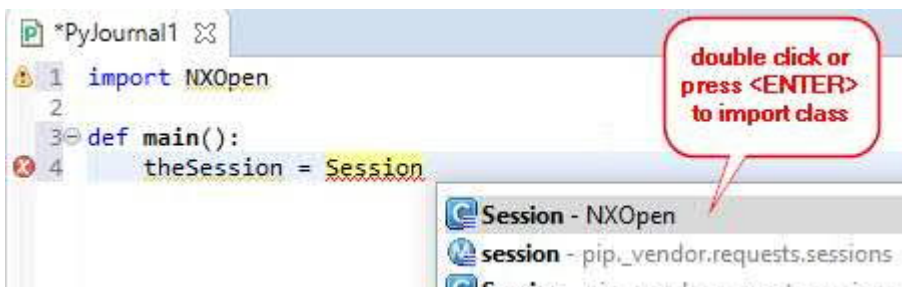
```

1 import NXOpen
2
3 def main():
4     theSession = NXOpen.Session.gets

```

GetSession()

Using the 'Session' class name cannot be resolved yet and a trailing "." (dot) will not provide any Auto Completion. But as far as Python detects known classes from PYTHONPATH, it provides a list of those classes and a double click on the class name (or a keyboard <ENTER>) will import this class from the package.



```

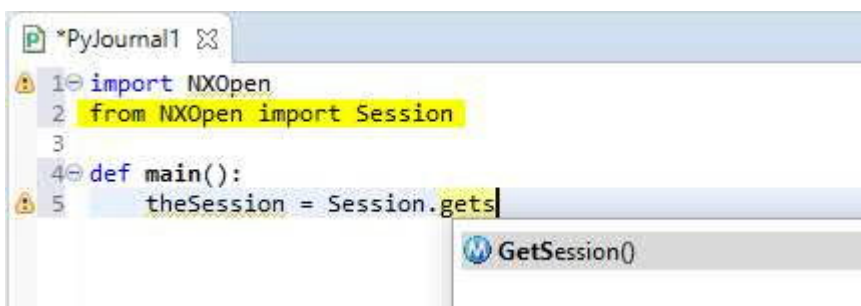
1 import NXOpen
2
3 def main():
4     theSession = Session

```

double click or press <ENTER> to import class

Session - NXOpen
session - pip_vendor.requests.sessions
Session - ...

With the 'from NXOpen import Session' statement we now get Auto Completion on the Session class name.



```

1 import NXOpen
2 from NXOpen import Session
3
4 def main():
5     theSession = Session.gets

```

GetSession()

4.2.2 Namespace notation

See also

- PR-8396897 (PyDev syntax discrepancy for import NXOpen modules with dot and underscore)
- [Forced Builtins](#) (current workaround)

```

1 import NXOpen
2 from NXOpen import Session
3
4 import NXOpen.Annotations
5 import NXOpen_Annotations
6 import NXOpen.Assemblies
7 import NXOpen_Assemblies
8 import NXOpen.Drafting
9 import NXOpen_Drafting
10 import NXOpen.BlockStyler
11 import NXOpen_BlockStyler
12 import NXOpen.Features
13 import NXOpen_Features
14 import NXOpen
15
16 def main():
17     theSession = Session.Session()

```

4.2.3 Type Hinting

In Python a local variable doesn't require a type specifier and the PyDev Text Editor is not able to detect type binding and class association right away.

For example, the following will not activate Auto Completion and not provide a list of possible choices when typing the last period/point after the class instance variable name:

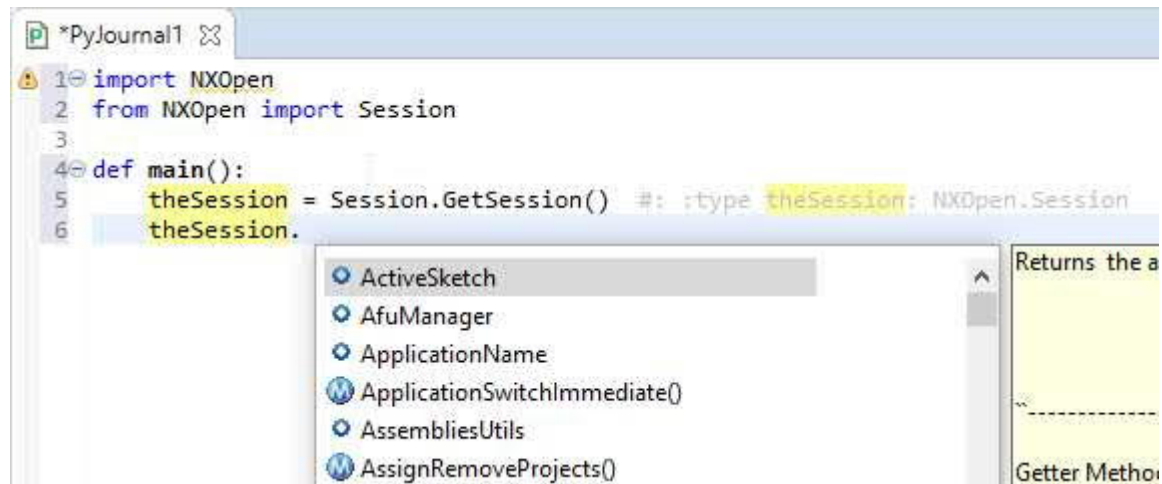
```

1 import NXOpen
2 from NXOpen import Session
3
4 def main():
5     theSession = Session.Session()
6     theSession.

```

To overcome this limitation, the necessary information for code completion can be added manually by providing "type hinting" with docstrings/comments as described at http://www.pydev.org/manual_adv_type_hints.html

```
theSession = NXOpen.Session.GetSession() #: :type theSession: Session
```



The screenshot shows a Python IDE window titled '*PyJournal1'. The code editor contains the following Python code:

```
1 import NXOpen
2 from NXOpen import Session
3
4 def main():
5     theSession = Session.GetSession() #: :type theSession: NXOpen.Session
6     theSession.
```

A lookup window is open below the code, showing a list of attributes and methods for the `Session` class. The list includes:

- ActiveSketch
- AfuManager
- ApplicationName
- ApplicationSwitchImmediate()
- AssembliesUtils
- AssignRemoveProjects()

On the right side of the lookup window, there is a description: "Returns the a" (likely "Returns the active sketch") and "Getter Method".

5 Using the PyDev Console

5.1 Open the PyDev Console

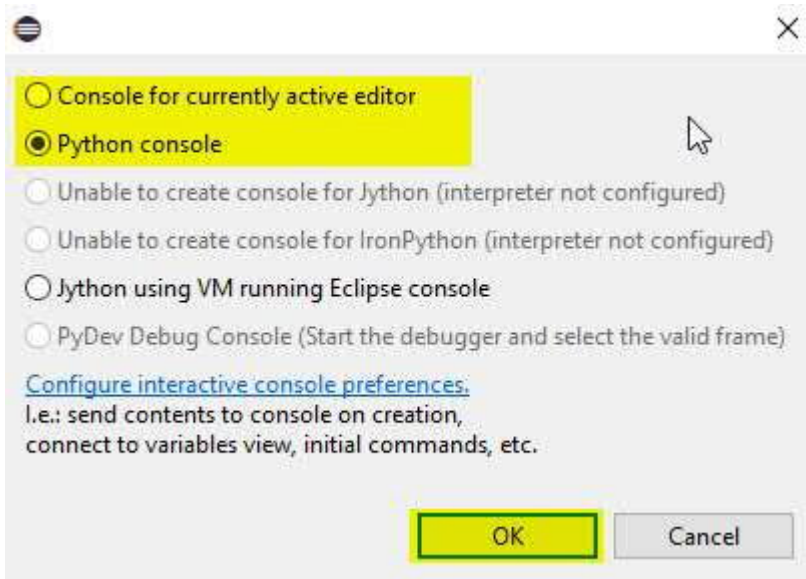
Window -> Show View -> Console



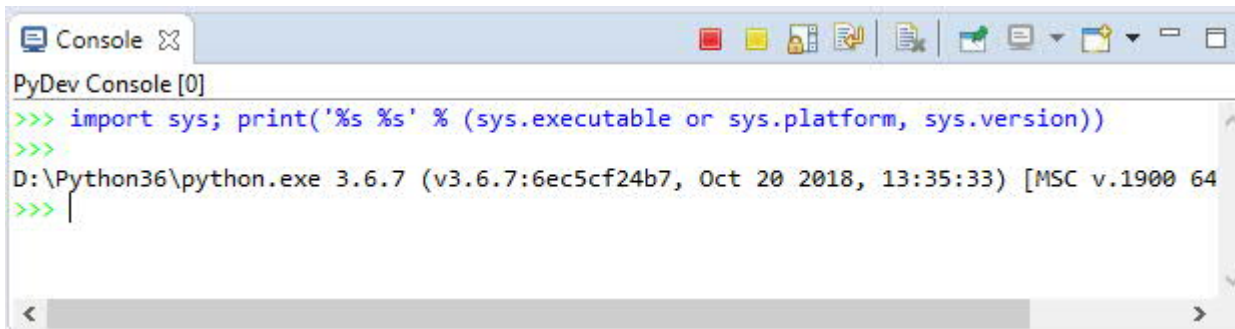
Console icons -> Open Console -> PyDev Console



Select the scope/content to start the PyDev Console with (Python Console will be independent from project):



Confirm that the PyDev Console is initialized correctly

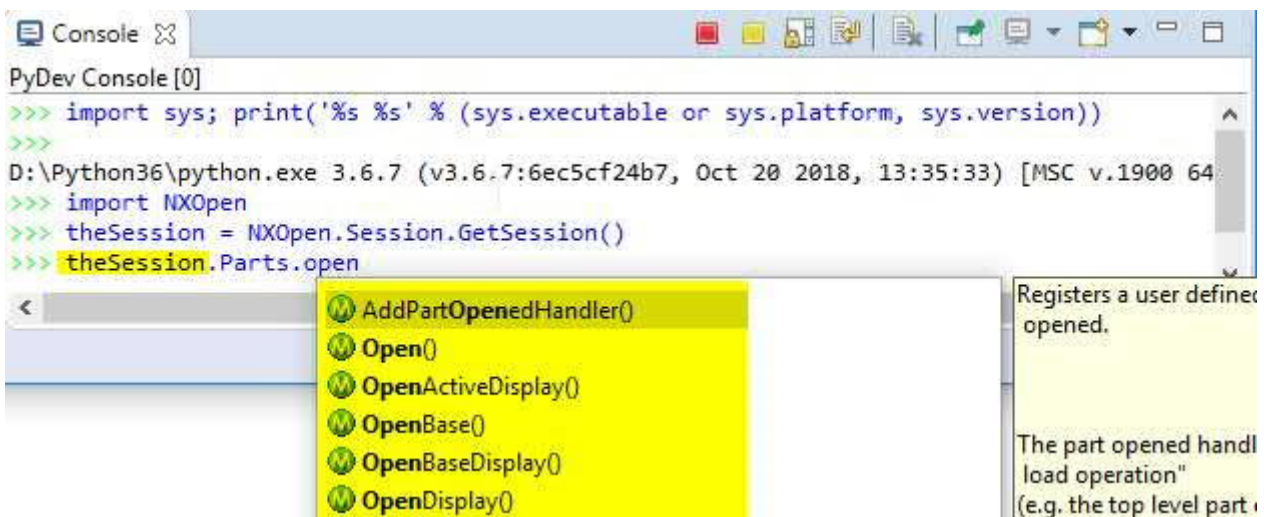


```

Console
PyDev Console [0]
>>> import sys; print('%s %s' % (sys.executable or sys.platform, sys.version))
>>>
D:\Python36\python.exe 3.6.7 (v3.6.7:6ec5cf24b7, Oct 20 2018, 13:35:33) [MSC v.1900 64
>>>
  
```

5.2 Typing Python Code in Console

In opposite to the limitation seen with the Python Text Editor, the type of a variable is known because it will be evaluated (interpreted) immediately which allows Auto Completion as desired:



```

Console
PyDev Console [0]
>>> import sys; print('%s %s' % (sys.executable or sys.platform, sys.version))
>>>
D:\Python36\python.exe 3.6.7 (v3.6.7:6ec5cf24b7, Oct 20 2018, 13:35:33) [MSC v.1900 64
>>> import NXOpen
>>> theSession = NXOpen.Session.GetSession()
>>> theSession.Parts.open
  
```

Auto-completion suggestions for `theSession.Parts.open`:

- AddPartOpenedHandler()
- Open()
- OpenActiveDisplay()
- OpenBase()
- OpenBaseDisplay()
- OpenDisplay()

Registers a user defined opened.

The part opened handler load operation" (e.g. the top level part

6 Debugging NXOpen Python

Add these 3 lines on top of your source code and save the file:

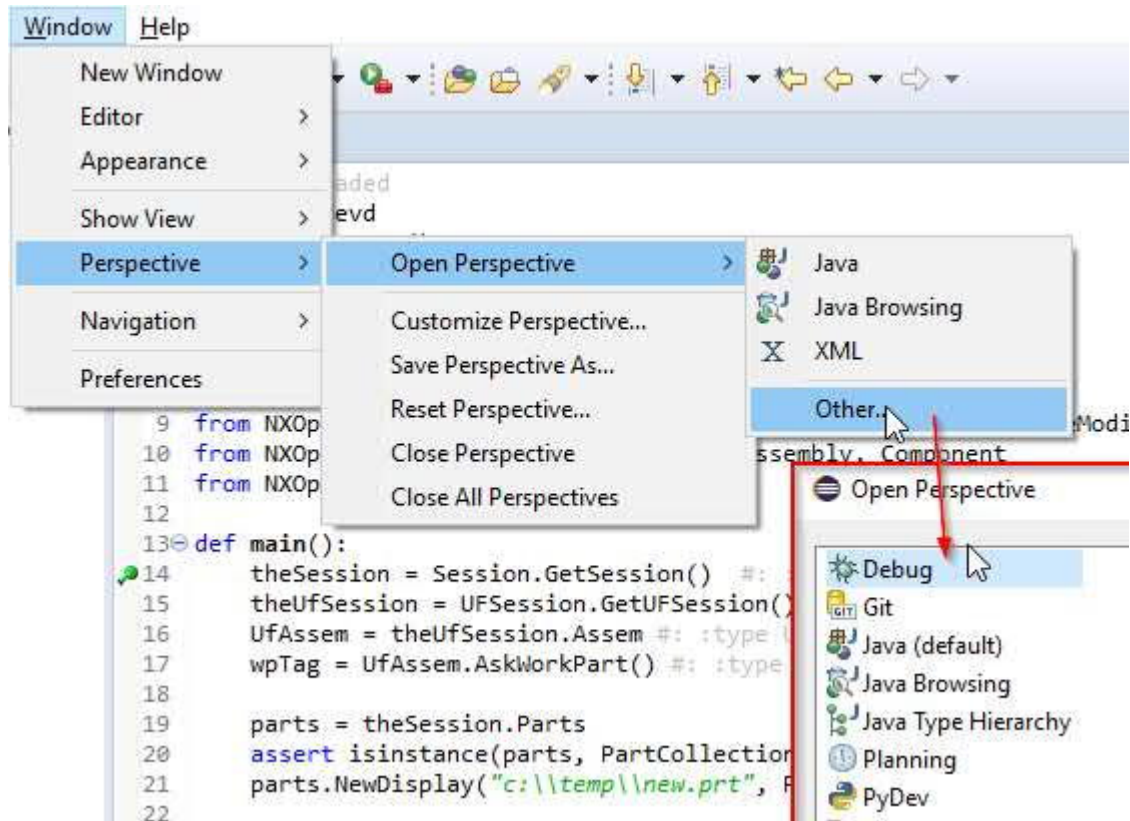
```
# nx:threaded
import pydevd
pydevd.settrace()
```

Note: The first flag is important but remove it after debugging if the journal is not using threaded extension modules.

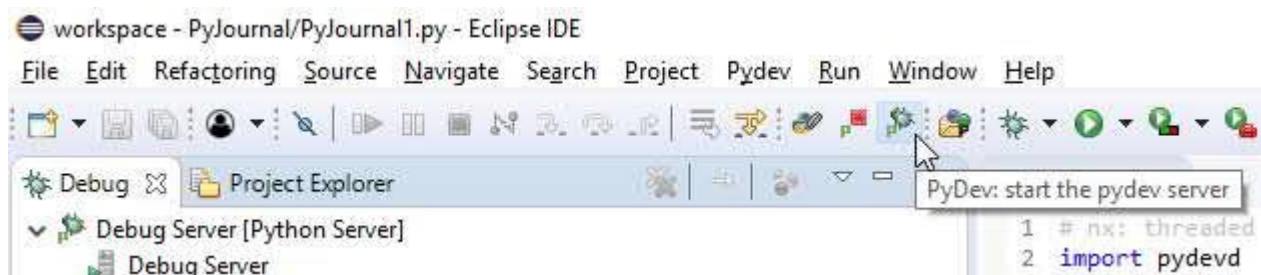
Open the Eclipse Debug Perspective from the toolbar or switch them using CTRL+F8:



If this icon is not displayed yet, use Window -> Perspective -> Open Perspective -> Other -> Debug



Start the PyDev Debug Server (see also PyDev -> Start Debug Server)

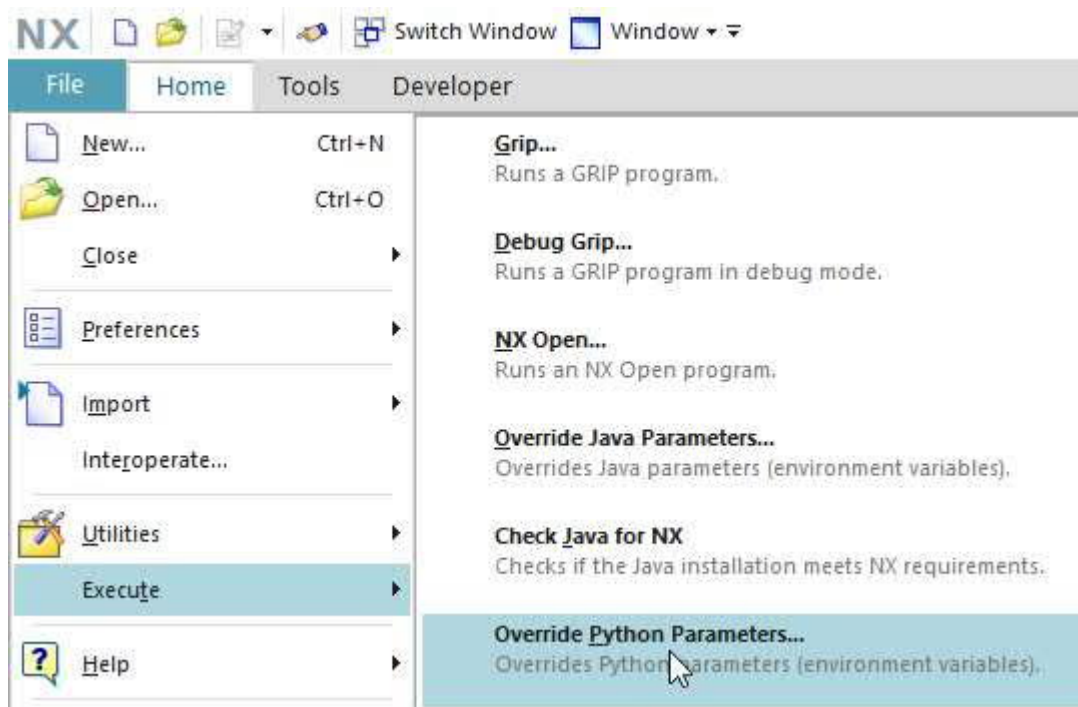


6.1 Debugging with NX GUI

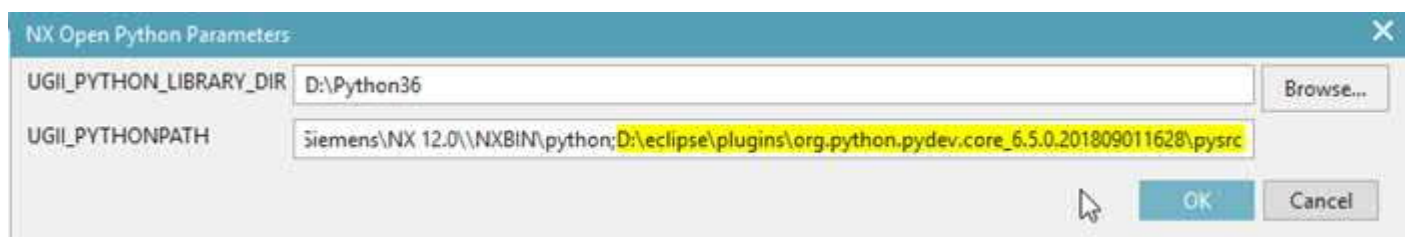
Start NX with the external Python distribution to ensure Python compatibility and extended capabilities:

```
set UGII_BASE_DIR=D:\Siemens\NX 12.0\
set UGII_PYTHON_HOME=D:\Python36
set UGII_PYTHON_DLL=python36.dll
set UGII_PYTHON_LIBRARY_DIR=%UGII_PYTHON_HOME%
set UGII_PYTHONPATH=%UGII_PYTHON_HOME%;D:\Python36\DLLs;D:\Python36\Lib;D:\Python36\Lib\site-packages;%UGII_BASE_DIR%\NXBIN\python;
set TCL_LIBRARY=%UGII_PYTHON_HOME%\tcl\tcl8.6
```

File -> Execute -> Override Python Parameters



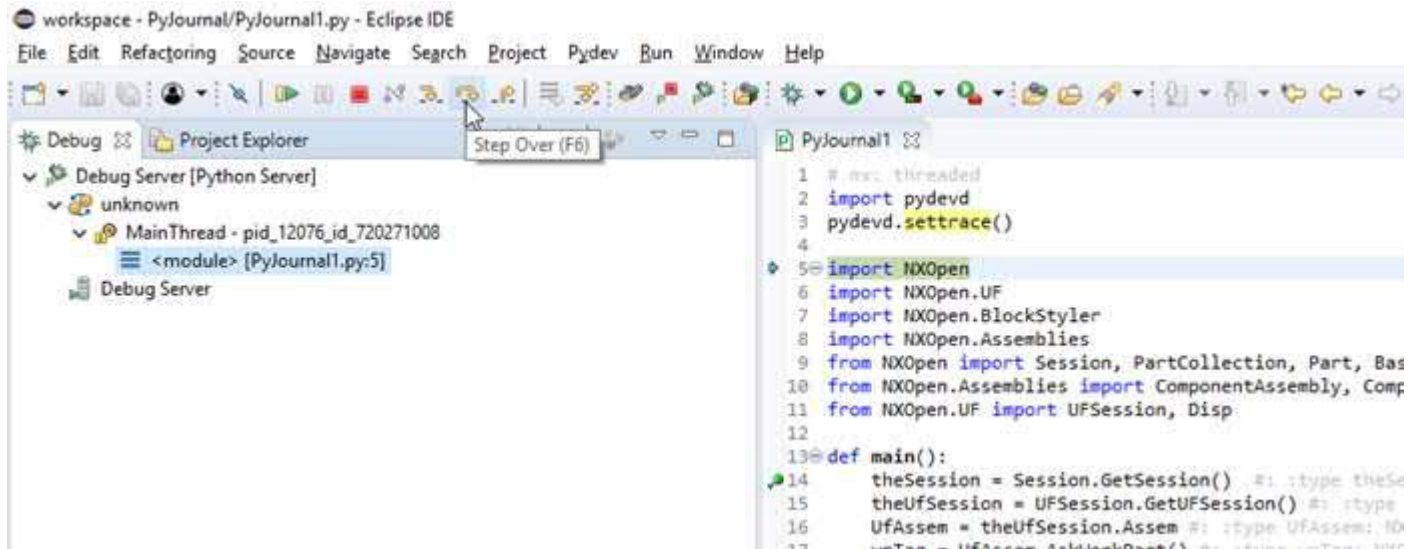
Add the Eclipse plugin PyDev core folder hosting the pydevd.py file to the UGII_PYTHONPATH parameter again:



In NX, Play the same Journal file, for example from the Journal Editor (ALT+F11):



Change to Eclipse (does not happen automatically like Visual Studio) which has the control now and step through the code as desired.



6.2 Debugging without NX GUI (Batch)

Without a connected interactive NX GUI instance, Eclipse doesn't know where to load the NX DLL libraries from. Hence, the external dependencies cannot be solved and debugging the code from within Eclipse will fail:

ImportError: DLL load failed: The specified module could not be found.

This is a common issue seen when debugging external NX applications and can be solved by simply starting Eclipse from a NX command prompt window which has the correct setting, e.g. including the NXBIN folder in %PATH%.

7 Caveats

If the Debugger or Debug Server had been stopped manually in Eclipse, playing the Journal in NX will not connect to Eclipse again.

Frank Berger
GTAC Programming Tools