Configure Interpreter

After installing it, the first thing you must do is configure the **Python and/or Jython and/or IronPython** interpreter.
To configure the interpreter:

1. Go to: **window > preferences > PyDev > Interpreter - (Python/Jython/IronPython)**.

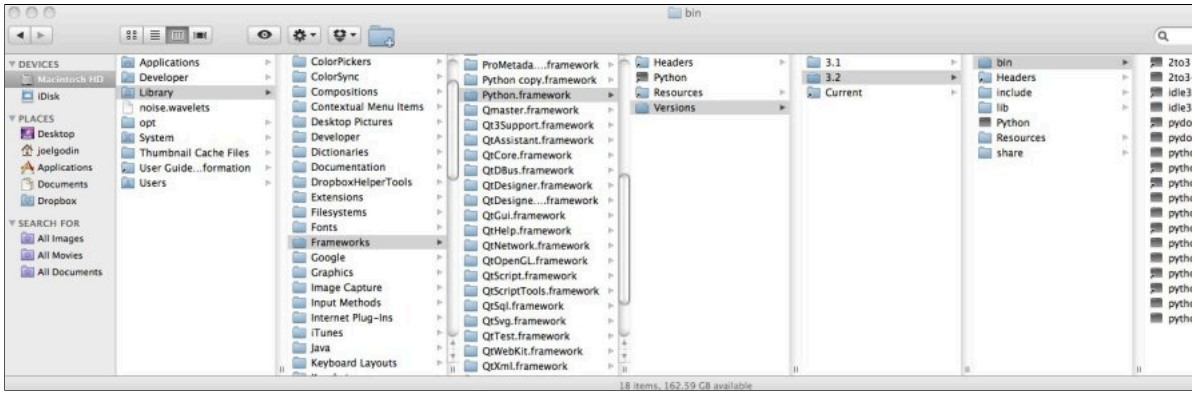
2. Choose the interpreter you have installed in your computer (such as python.exe, jython.jar or ipy.exe).

Note that the **Auto Config** will try to find it in your PATH, but it can fail if it's not there (or if you want to configure a different interpreter).

On **Windows** it'll also search the registry and provide a choice based on the multiple interpreters available in your computer (searching in the registry).

On **Linux/Mac**, usually you can do a 'which python' to know where the python executable is located.

On **Mac** it's usually at some place resembling the image below (so, if you want to configure a different version of the interpreter manually, that's where you'd wa

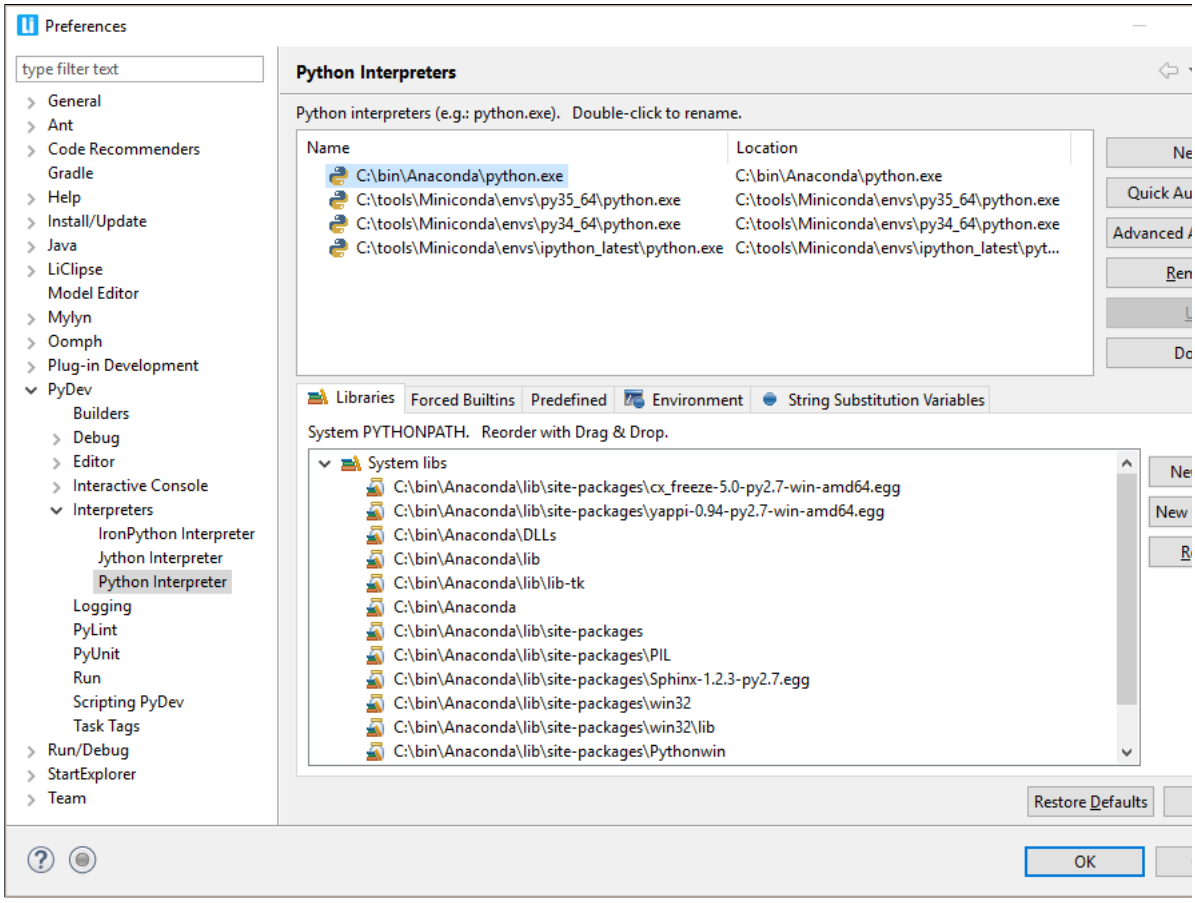


3. Select the paths that will be in your **SYSTEM PYTHONPATH**.

IMPORTANT: Select only folders that will **NOT be used as source folders for any project** of yours (those should be later configured as source folders in the

IMPORTANT for Mac users: The Python version that usually ships with Mac doesn't seem to have the .py source files available, which are required for PyDev, interpreter is recommended (i.e.: Download it from http://python.org). If you don't want to use a different interpreter, get the source files for the Python **/Lib** f the system installation.

After those steps, you should have a screen as presented below:



How to check if the information was correctly gotten

The **System libs** must contain at least the Lib and the Lib/site-packages directory.

The **Forced builtin libs** must contain the modules built into the interpreter (and others whose analysis should be done dynamically. See: Forced Builtins).

How to work with virtual environments (virtualenv, conda, etc)

For **PyDev**, a virtual environment works as any other **regular Python executable**, so, just click the **New...** button and select the Python executable in the virt take care to **double check the folders** which are selected when the virtual environment executable is used to make sure that you leave checked the **/Lib** folde interpreter **if** the virtual environment inherits the **PYTHONPATH** from a base installation).

venv is the recommended way of managing virtual environments starting from Python 3.5. By default, venv will only symlinks to Python interpreter, and this wil created virtual environments in Pydev showing all system installed site-packages. To get it displayed correctly, you will need to append "--copies" to the "python /path/to/venv"

What if it is not correct?

The most common error is having a problem in the environment variables used from the shell that spawned Eclipse, in a way that for some reason when getting interpreter, it gathers the info from another interpreter (thus mixing the interpreter and the actual libraries).

Usually running (from the command prompt) the file that gives that info for PyDev can help you discovering the problem in your configuration (interpreterInfo.p

That file is usually located at: eclipse\plugins\org.python.pydev_$version$\pysrc\interpreterInfo.py, but it can be at other location depending on how you install

In Python: python.exe interpreterInfo.py

In Jython: java.exe -cp c:\path\to\jython.jar org.python.util.jython interpreterInfo.py

In IronPython: ipy.exe interpreterInfo.py

If you're unable to find out what's going on, please open an issue in the tracker (https://www.brainwy.com/tracker/PyDev (giving the output obtained from exec in your machine).

What if I add something new in my System PYTHONPATH after configuring it?

Since PyDev 3.0, such modifications should be automatically detected by PyDev and require no further steps (so, if you **pip**-installed something or manually add is already in the **PYTHONPATH** -- such as **Lib/site-packages** -- just wait a bit and PyDev should start considering it in code-analysis and code-completion).

Libraries

The **System libs** are the libraries that will be added to the PYTHONPATH of any project that is using this interpreter.

For **Python and IronPython**, it's composed of **folders, zip files and egg files**. Note that if dlls should be added to the PYTHONPATH, the folders actually cont be added, and they must have the same name to be imported in the code (the case is important). I.e.: if you want to import **myDllModule**, it **must** be called m that .pyd and .so extensions are also accepted).

For **Jython**, it's composed of **folders and jars**.

Forced Builtins

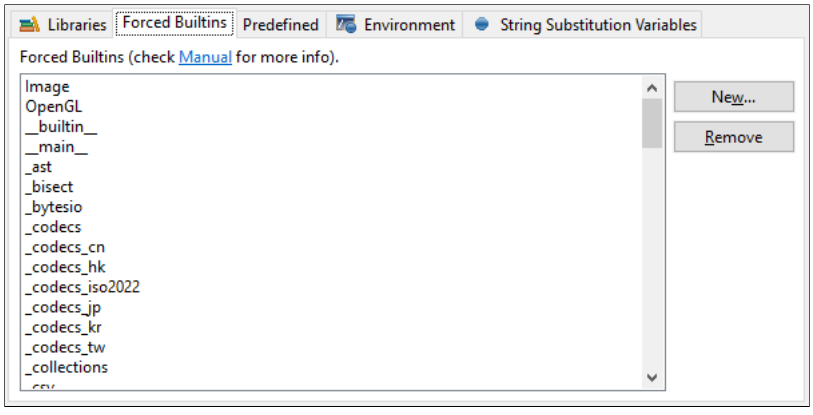
The Forced builtin libs are the libraries that are built-in the interpreter, such as ***__builtin__*, sha, etc** or libraries that should forcefully analyzed through a shell (i. in this list, PyDev will spawn a shell and do a dir() on the module to get the available tokens for completions and code-analysis) – still, sometimes even that is n which case, Predefined Completions may be used to let PyDev know about the structure of the code.

For **Python**, you should have around **50** entries

For **Jython** around **30** entries.

For **IronPython** more than **100** entries. All the packages built into .NET should be included here – e.g.: Microsoft, Microsoft.Windows.Themes, System, System

Additionally, you may add other libraries that you want to treat as builtins, such as **os, wxPython, OpenGL, etc**. This is very important, because PyDev works with static information, but some modules don't have much information when analyzed statically, so, PyDev must create a shell to get information on those. Ano that they **must** be on your System PYTHONPATH (otherwise, the shell will be unable to get that information).



Predefined Completions

Predefined completions are completions acquired from sources that provide only the interfaces for a given Python module (with Python 3.0 syntax).

A predefined completion module may be created by having a module with the extension **.pypredef** with regular Python 3.0 contents, but with attributes having methods having as the body a sole return statement – and the docstring may have anything.

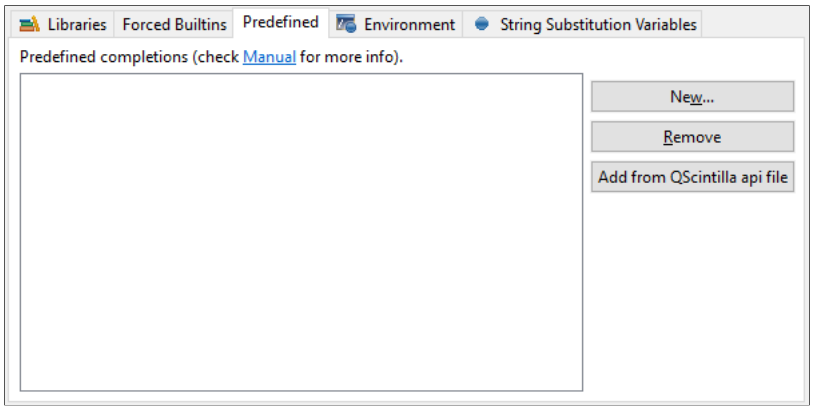
Example for a **my.source.module** (must be declared in a **my.source.module.pypredef** file):

```
MyConstantA = int
MyConstantB = int
```

Note 1: the name of the file is the exact name of the module

Note 2: .pypredef files are not searched in subfolders

Optionally a QScintilla .api file may be added. When this is done, PyDev will try to create .pypredef files from that .api file and will add the folder containing PYTHONPATH. Note that this conversion is still in beta and the file may not be correctly generated, so, keep an eye for errors logged when a code-completion tha modules (while it will not fail, those completions won't be shown using the .pypredef files). In those situations, please create a bug-report with the .api file that code.



Environment

The variables defined at the environment will be set as environment variables when running a script that uses the given interpreter (note that it can still be over configuration)

String substitution variables

Strings defined here may be used in:

- project configuration for source folders and external libraries
- launch configuration for the main module

They can be used in those places in the format: ${DECLARED_VARIABLE}

Cygwin users

PyDev currently has no support for cygwin. Currently you'll be able to configure the interpreter with cygwin, but there are still other related problems (mostly on windows and cygwin paths as needed).