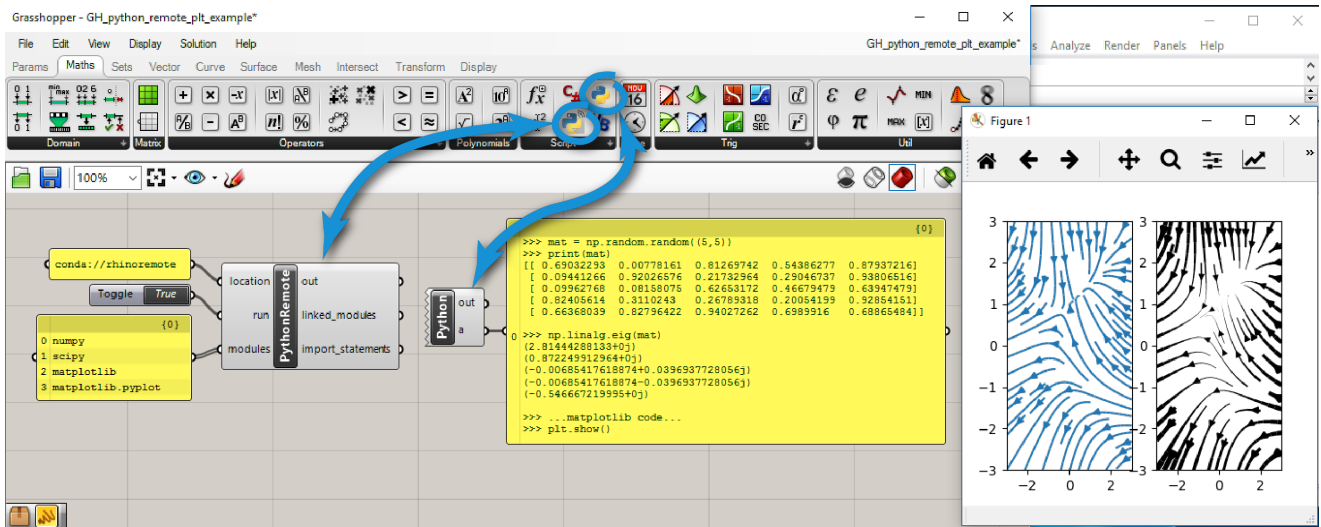


# gh-python-remote

## Connect an external python instance to Grasshopper, and vice-versa.

This lets you run any Python package directly from Grasshopper, including numpy and scipy!



# Installation

Requires a Python 2.7 installation, not compatible with Python 3. Currently Windows only.

## 1. Install the software dependencies:

Before installing gh-python-remote in **Rhino 6**, you will need to install Python 2, Rhino 6, and open Grasshopper in Rhino 6 at least once.

Before installing gh-python-remote in **Rhino 5**, you will need to install Python 2, Rhino 5, Grasshopper and GHPython, and drop the GHPython component on the Grasshopper canvas in Rhino 5 at least once.

Install the following:

**Python 2.7:** gh-python-remote was developed with the [Anaconda](#) distribution in mind, but any Python distribution works. If you already have Anaconda installed with Python 3, do not reinstall, instead just read the next paragraph.

*If you want to be able to name virtual environments in gh-python-remote by their conda name, select "Add conda to my PATH" when prompted during Anaconda's installation.*

Python **virtual environment** (optional):

Isolate dependencies for each project by creating a new virtual environment. If you use Anaconda, open the Windows command prompt (or the Anaconda prompt if you chose not to add conda to your PATH) and type:

```
conda create --name rhinoremove python=2.7 numpy scipy
```

This will set you up with a new virtual environment named `rhinoremove`, and install numpy and scipy in it.

<b>Rhinoceros3D:</b>	Version 5 and 6 on Windows are supported. Rhino 6 for Mac might be supported in a later release.
----------------------	--

<b>Grasshopper:</b>	On Rhino 6, it is already installed. On Rhino 5, install version 0.9.0076. <b>Open it at least once before continuing.</b>
---------------------	--

<b>GH Python:</b>	On Rhino 6, it is already installed. On Rhino 5, install version 0.6.0.3. <b>On Rhino 5, drop it on the Grasshopper canvas at least once before continuing.</b>
-------------------	---

## 2. Install gh-python-remote:

From the Windows command prompt (or the special Anaconda or Python prompt if pip is not in your path by default), run:

(If you are using a virtual environment, remember to **activate** it first. With the conda virtual environment from above, you would need to run `conda activate rhinoremote` in the Windows or Anaconda prompt)

```
pip install gh-python-remote --upgrade
python -m ghpythonremote._configure_ironpython_installation
```

This will install gh-python-remote for Rhino 6, and install the gh-python-remote UserObject in all Grasshopper versions.

The `ghpythonremote._configure_ironpython_installation` script takes an optional location argument that can be `5`, `6` (default), or the path to a target IronPython package directory.

For example, to install for Rhino 5, replace the second command with:

```
python -m ghpythonremote._configure_ironpython_installation 5
```

To install to another location, like for Rhino 7:

```
python -m ghpythonremote._configure_ironpython_installation ^
"%APPDATA%\McNeel\Rhinoceros\7.0\Plug-ins\^
IronPython (814d908a-e25c-493d-97e9-ee3861957f49)\settings\lib"
```

## Usage

All the examples files are copied in the `%APPDATA%\Grasshopper\UserObjects\gh-python-remote\examples` folder. You can also download them from the [github repo](#).

## From Grasshopper to Python

### Step-by-step

1. Open the example file `GH_python_remote.ghx` in Grasshopper, or drop the gh-python-remote component on the canvas.
2. Use the `location` input to define the location of the Python interpreter you want to connect to.
3. Use the `modules` input to define the modules you want to access in the GHPython component.
4. Change `run` to `True` to connect.

In the GHPython component, the imported modules will now be available via the sticky dictionary. For example if you are trying

5. to use Numpy:

```
import scriptcontext
np = scriptcontext.sticky['numpy']
```

### Notes

Creating remote array-like objects from large local lists is slow. For example, `np.array(range(10000))` takes more than 10 seconds. To solve this, you need to first send the list to the remote interpreter, then create the array from this remote object:

```
import scriptcontext as sc
import ghpythonremote
np = sc.sticky['numpy']
rpy = sc.sticky['rpy']

r_range = ghpythonremote.deliver(rpy, range(10000))
np.array(r_range)
```

Additionally, Grasshopper does not recognize remote list objects as lists. They need to be recovered to the local interpreter first:

```
import scriptcontext as sc
import ghpythonremote
from ghpythonlib.treehelpers import list_to_tree # Rhino 6 only!
np = sc.sticky['numpy']

a = np.arange(15).reshape((3,5))
a = ghpythonremote.obtain(a.tolist())
a = list_to_tree(a, source=[0,0])
```

`ghpythonlib.treehelpers` is Rhino 6 only, see the [treehelpers gist](#) for an equivalent implementation if you need it on Rhino 5.

## Quick-ref:

*\* marks an input that is only available by editing the `gh-python-remote` `UserObject`, or in `GH_python_remote.ghx`.*

Arguments:	<b>*code (string):</b>	Path to the <code>GH_to_python.py</code> code file.
	<b>location (string):</b>	
		Path to a python executable, or to a folder containing <code>python.exe</code> , or the name of a conda-created virtual environment prefixed by <code>conda://</code> ( <code>conda://env_name</code> , requires <code>conda</code> available in your PATH). If empty, finds python from your windows <code>%PATH%</code> .
	<b>run (boolean):</b>	Creates the connection, and imports new modules, when turned to True. Kills the connection, and deletes the references to the imports, when turned to False.
	<b>modules (string list):</b>	
		List of module names to import in the remote python. They will be added to the <code>scriptcontext.sticky</code> dictionary, allowing them to be reused from other python components in the same Grasshopper document. Submodules (for example <code>numpy.linalg</code> ) have to be added explicitly to this list to be available later, and importing the parent package is also required even if only the submodule is used.
	<b>*log_level (string from ['NOTSET', 'DEBUG', 'INFO', 'WARNING', 'ERROR', 'CRITICAL']):</b>	
		Logging level to use for the local IronPython and the remote python instance.
	<b>*working_dir (string):</b>	
		Working directory for the remote python instance.
Returns:	<b>out (string):</b>	Console output with DEBUG information.
	<b>linked_modules (string list):</b>	
		List of imported module names.
	<b>rpy (rpyc connection object):</b>	
		The object representing the remote Python interpreter.
	<b>import_statements (string):</b>	
		What to use in the GHPython component to actually use the imported modules.

## From Python to Grasshopper

You can also use `gh-python-remote` to programmatically control a Rhinoceros instance, and connect to it via Python. Have a look at `examples/python_to_GH.py` for a full working example.

## License

Licensed under the [MIT license](#).

## Changelog

### 1.3.1 (2020-04-01)

## Changes

- Allow access to GH user objects from CPython. Access them from a PythonToGrasshopperRemote object as `py2gh.gh_remote_userobjects.<UO Name>`
- Remove `GhCompService.get_component`. Access remote GH components from a PythonToGrasshopperRemote object as `py2gh.gh_remote_components.<Component Name>`

## Fix

- Fix CPython -> Rhino Python connections for Rhino 6.

## 1.3.0 (2020-03-20)

---

## New

- Add option to select Rhino version in CPy to GH (#29).

## Changes

- Better Rhino path finder.
- Catch conda env list failures.
- Upgrade to RPyC 4.1.4

## Fix

- Fix missing file at installation.
- Fix async request timeout (#14).
- Repair log level passing to `pythonservice.py`.

## 1.2.1 (2019-07-24)

---

## Fix

- Cleanup examples directory, and fix missing file at installation (#26).
- Update logger names in `.ghuser` file

## 1.2.0 (2019-07-08)

---

## New

- Now fully compatible with Rhino 6.

## Changes

- Revert using a special RPyC distribution, now explicitly using upstream.
- Upgrade to RPyC 4.1.0.
- Hide `rpyc` behind a special `ghpythonremote` import. `rpyc.utils` functions `deliver` and `obtain` are available as `ghpythonremote.deliver` and `ghpythonremote.obtain`; `rpyc` should be imported with `from ghythonremote import rpyc` if ever needed.
- Expand the environment path in CPython subprocess, trying to match what conda would do, helping Numpy finds its DLLs.

## Fix

- Repair pip install command on recent pip with pip.main deprecated.
- Repair incompatibilities with IronPython 2.7.0.
- Repair incompatibilities with RPyC 4.1.0, using monkey-patched compatibilty fixes for IronPython.
- Remove unnecessary pip install parameters.

## 1.1.4 (2018-02-28)

---

### Fix

- Stop using the *exposed* prefix to remote attributes, per rpyc v4.

## 1.1.3 (2018-02-27)

---

### Fix

- Rename `async` import from rpyc to `async_`.
- Bump rpyc version to [pilcru/rpyc@3.4.6](#) to fix IronPython 2.7.0 dump bytes issues.

## 1.1.2 (2018-02-23)

---

### Fix

- Use https file location for the dependencies, to remove the need for git when installing.

## 1.1.0 (2018-02-14)

---

### New

- Documented `obtain` and `deliver` features of rpyc to speedup remote array-like objects creation and retrieval.

### Changes

- Use the v4.0.0 pre-release of rpyc to fix IronPython <-> CPython `str` unpickling issues.
- Improve error messages when connection is lost.

### Fix

- Repair the GH to python example, where argument passing (for the port configuration) was broken.

## 1.0.4 (2017-10-06)

---

### Fix

- Fix quote escaping issue in pip install command for IronPython.

## 1.0.3 (2017-10-02)

---

First public release.