
BlenderVR User Manual

Release 0.1

BlenderVR Team

June 26, 2017

1	Installation	2
1.1	Install BlenderVR (manual)	2
1.2	Install BlenderVR (automatic)	6
1.3	Install for BlenderVR Development	6
1.4	Install Plugins	9
2	First Run	12
2.1	Document Sections	12
2.2	Launch BlenderVR	12
2.3	Open the Simulation File	12
2.4	Edit the Configuration File	12
2.5	Run	13
3	How to Use	14
3.1	User Interface	14
3.2	Processor File	17
3.3	Configuration File	21
3.4	Virtual Reality Private Network (VRPN)	34
3.5	Open Sound Control (OSC)	35
3.6	Oculus Rift DK2	37
4	Architecture	38
4.1	Master and Slaves	38
4.2	Notion of Vehicle	39
4.3	UI - Daemon Network Protocol	39
4.4	Configuration File	40
5	Development	49
5.1	Building Documentation	49
5.2	Build Windows Executable	50
6	Frequently Asked Questions	53
6.1	FAQ - BlenderVR Install	53
6.2	FAQ - BlenderVR Run	54

The BlenderVR Manual is part of the [BlenderVR](#) project.

Installation

Install BlenderVR (manual)

In order to install BlenderVR you need this guide.

Tip: Windows “standard” (non-developers) users are invited to download the BlenderVR Install Executable for Windows 7 (though you are still advised to read through the following installation and how to use guides). See the [Install BlenderVR \(automatic\)](#) page.

Note: If you need the full development setup make sure to follow the Development Environment guide.



Document Sections

- Folder Structure
- Acquiring Blender
- Acquiring BlenderVR
- Download Samples Scenes
- Install Dependencies
- Quick Setup
- Running

Folder Structure

After all the downloads and installations you should end up with the following folder structure. This is a recommendation, and it will be used as reference along this manual (the directory holding these files is referred as \$INSTALL_DIR in the next sections).

```
//source/ BlenderVR Source Code
```

```
//blender/ Blender Binaries
```

```
//samples/ BlenderVR Samples
```

```
//venv/ Python Virtual Environment
```

Acquiring Blender

BlenderVR requires Blender 2.75 or newer. Optionally you can use a patched version of Blender 2.74 available here for all the supported platforms.

- Mac OSX 64 bit: Blender 2.74 Patched. ¹
- Windows 32 bit: Blender 2.74 Patched. ²
- Windows 64 bit: Blender 2.74 Patched. ²
- Linux 32 bit: Blender 2.74 Patched. ³
- Linux 64 bit: Blender 2.74 Patched. ³

Acquiring BlenderVR

- BlenderVR Sources (download and unzip in the top folder, rename it **source**)
- Oculus Library (download and unzip it inside the **source/libs** folder, rename it **python-ovrSdk**)

Download Samples Scenes

Before getting started, you'll probably want to take a look at the available BlenderVR ".blend" sample scenes.

- Download All Samples

You can also download an individual sample folder. For that you will need [SVN](#) or [SVN Tortoise](#) (Windows only). To check which samples are available visit the [Samples Repository](#).

```
$ cd $INSTALL_DIR
$ mkdir -p samples
$ svn checkout https://github.com/BlenderVR/samples/trunk/basic/basic samples/
```

(Or simply *svn checkout* the required sample with SVN Tortoise).

Install Dependencies

Install those packages or make sure you have them in your system.

All Time Mandatory

- Python 3.4

Future developments will make the following packages optional:

- QT 4.8
- PIP (the Python install should have installed pip, try to use it in the next Section before installing)

Note: MacOS: open Qt.mpkg with mouse right click -> Open, to avoid popup window "can't install, non identified developer".

¹ Requires Mac OSX 10.6+

² Compatible with Windows 8, 7, Vista. If Blender reports an error on startup, please install the [Visual C++ 2013 Redistributable Package](#).

³ Requires glibc 2.11. Suits most recent GNU/Linux distributions

Quick Setup

Type the following commands in your terminal. If you are in Windows we recommend you to use [Power Shell](#) or similar.

On OSX/Linux:

```
$ cd $INSTALL_DIR
$ pip3 install virtualenv
$ pyvenv venv
$ source venv/bin/activate
$ pip3 install -r source/requirements.txt
```

At that point, if the console does not show any trace of `pyside_postinstall.py` script execution, type in:

```
$ python ./venv/bin/pyside_postinstall.py -install
```

Note: MacOS: running these lines may popup window “download the command line developer tools”, go for it.

Linux: If the “`pyvenv venv`” command fails, you can try the command “`pyvenv-3.4 venv`”.

Linux: If `pyvenv` command fails due to missing `ensurepip` module, try [this script](#).

On Windows:

```
$ cd $INSTALL_DIR
$ pip3 install virtualenv
$ virtualenv venv
$ .\venv\Scripts\activate
$ pip3 install -r source\requirements.txt
$ python3 .\venv\Scripts\pyside_postinstall.py -install
$ python3 .\source\blendervr
```

You may have to add the path to the python binary, e.g.

```
$ [Environment]::SetEnvironmentVariable("Path", "$env:Path;C:\Python34\;C:\Python34\Scripts\")
```

(For PowerShell to automatically add this path at startup, add this line to a file named e.g. `profile.ps1` that you'll place in your `WindowsPowerShell` directory)

Running

Type the following commands in your terminal. If you are in Windows we recommend you to use [Power Shell](#) or similar.

On OSX/Linux:

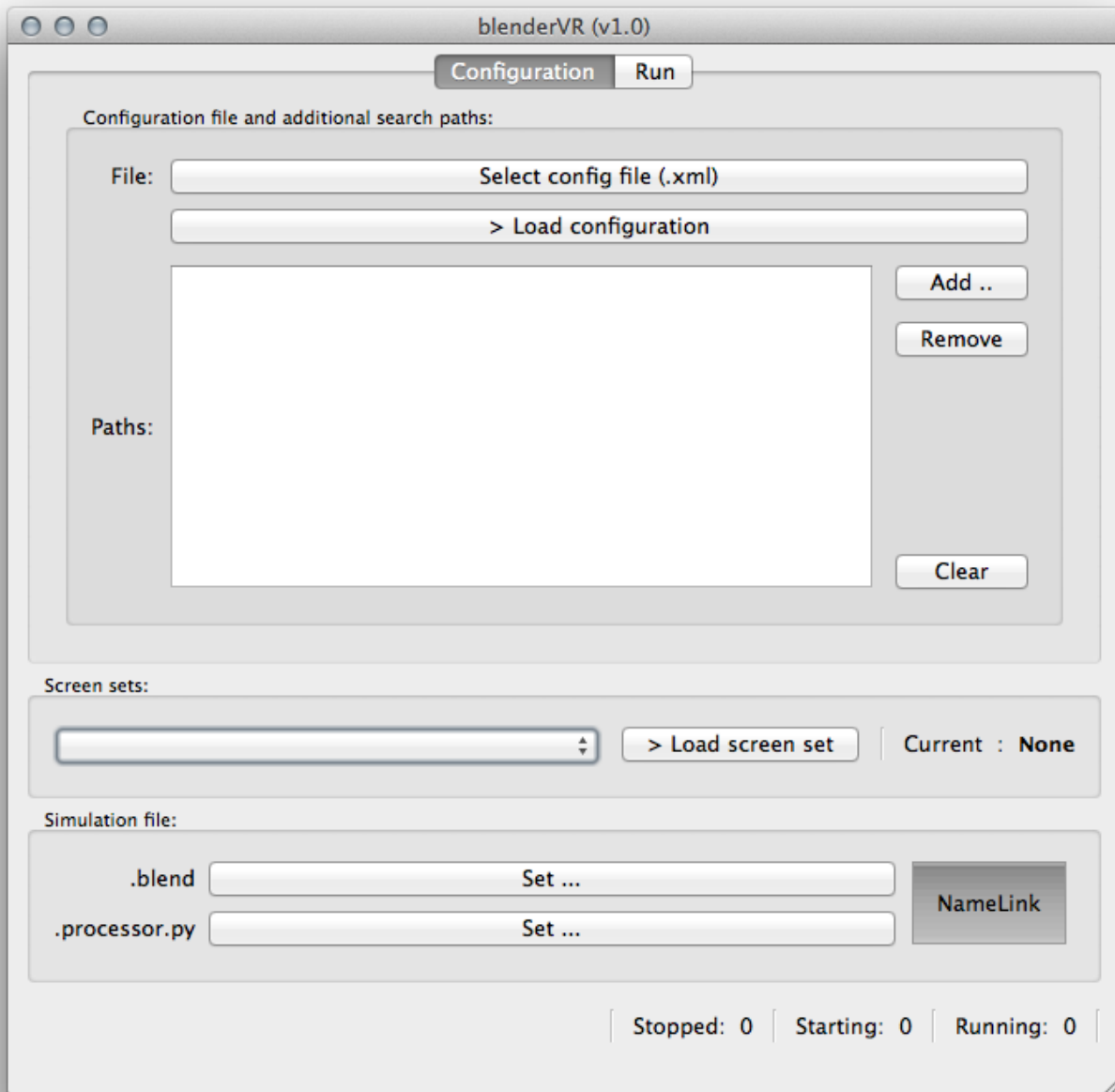
```
$ cd $INSTALL_DIR
$ source venv/bin/activate
$ ./source/blendervr
```

On Windows:

```
$ cd $INSTALL_DIR
$ .\venv\Scripts\activate
$ python3 .\source\blendervr
```

You should now see the BlenderVR window popping up (see figure below). Congratulations your installation was a success!

Once you are done running BlenderVR you can end the virtual environment running the command:



§ deactivate

For your convenience it is recommended to create a bash script to help re-launching the BlenderVR environment.

Install BlenderVR (automatic)

This section details the install of BlenderVR based on an executable file, automation of the procedure described in the Install BlenderVR page.

Once BlenderVR installed, verify the installation by opening the newly created BlenderVR application (you should see the BlenderVR interface pop-up) and move on to the First Run page.

Document Sections

- Windows 7 (64bits)

Note: If none of the provided executable is compliant / work on your machine, fallback to the Install BlenderVR page.

Windows 7 (64bits)

Download executable:

- BlenderVR Win7 64bits WebInstall
- BlenderVR Win7 64bits Standalone



Warning: Even the Standalone install requires an internet connection (to install non-bundled python libs).

First, launch the executable, that will require administrator's rights. Move on the installation procedure and, when asked, select Python Install for all users:

Near the end of the installation, accept the run of BlenderVR-install.bat to complete the installation.

Install for BlenderVR Development

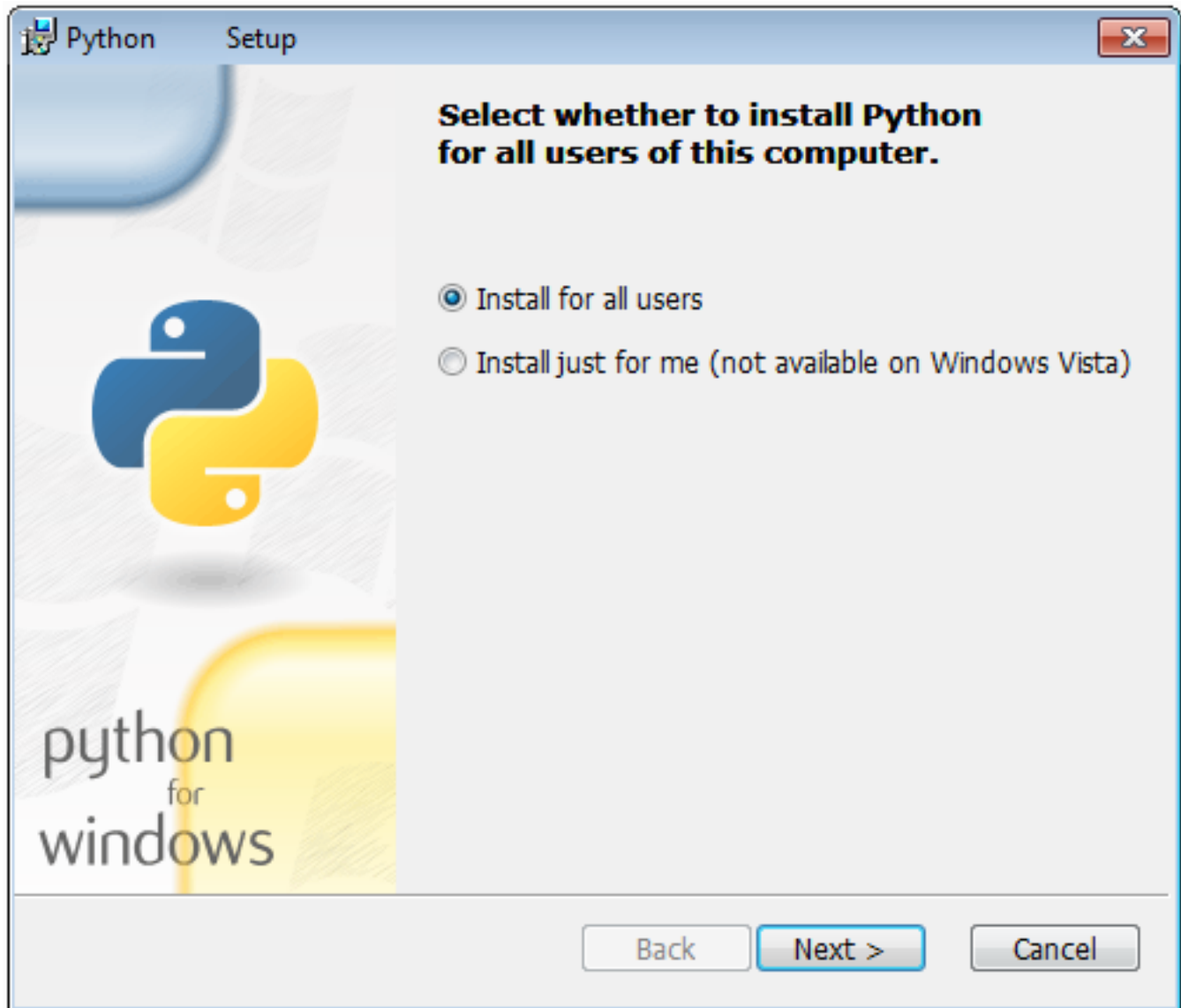
This guide walks you over the basic steps of setting up a development environment for BlenderVR.

Note: For casual BlenderVR users, please refer to the Install BlenderVR page.

The install is the same as described in Install BlenderVR but for:

- git clone of the BlenderVR repository, to freely modify and eventually commit your modifications.
- svn/git clone of the BlenderVR samples, to eventually add your own demo scenes to the BlenderVR samples repository
- manual compilation of Blender, if you need to modify its source code.

Most of the time you won't need to modify and rebuild Blender, so those instructions are specified separately.



Document Sections

- [Acquiring Blender](#)
- [Acquiring BlenderVR](#)
- [Download Samples Scenes](#)
- [Requirements](#)
- [Quick Setup and Running](#)

Acquiring Blender

BlenderVR requires a vanilla Blender 2.75 or newer.

If you ever need to modify and rebuild Blender for further customizations, please consult the [Blender's official documentation](#). Else, download the sources provided in [Acquiring Blender](#).

Acquiring BlenderVR

To download the latest BlenderVR git version (master HEAD):

```
$ git clone https://github.com/BlenderVR/source.git
$ cd source
$ git submodule update --init --recursive --remote
```

Note: In old versions of git (1.6x, 1.7x) the `--remote` option doesn't exist and thus should be left out

Download Samples Scenes

Regarding BlenderVR samples, Git is not a good system to work on binary files, so it's recommended to use the SVN protocol to interact with the samples repository instead:

```
$ cd $INSTALL_DIR
$ svn checkout https://github.com/BlenderVR/samples/trunk samples
```

where `$INSTALL_DIR` is the root BlenderVR folder (see the [Install BlenderVR](#) page for more details).

Alternatively, to fetch an individual sample folder, you can use:

```
$ svn checkout https://github.com/BlenderVR/samples/trunk/basic/basic
```

Alternatively if you want to access the repository via GIT you can do:

```
$ cd $INSTALL_DIR
$ git clone https://github.com/BlenderVR/samples.git
```

Requirements

Install those packages or make sure you have them in your system.

All Time Mandatory

- GIT
- Python 3.4

Required for Interface Development

At this moment the following packages are always required, but the plans are to make them optional.

- PIP
- QT 4.8

Note: Unix users may want to install these libraries via their package manager. If so, note that for PySide binding build, you'll need Qt libraries along with *development headers* that may come from a separate package (for example on Ubuntu 14.04 where `libqt4-dev` is required). See [PySide build scripts](#) for more information.

Quick Setup and Running

see Quick Setup and Running in the Install BlenderVR Section.

Install Plugins

BlenderVR comes with plugins, e.g. to support the Oculus Rift DK2 or to use VRPN devices. This page will lead you through their installation, while none of them is mandatory to run BlenderVR.

Document Sections

- Folder Structure
- VRPN
- Oculus Rift DK2

Folder Structure

Add a `plugins` directory to your `$INSTALL_DIR` (see Folder Structure, this is once more a recommendation as it will be used as reference along this manual).

VRPN

This section briefly exposes the [VRPN](#), install. In a nutshell, you will have to build VRPN, launch a VRPN server and BlenderVR then will act as a VRPN client to fetch the data from your VRPN devices. once the VRPN server launched on your machine/network, any device defined in your `vrpn.cfg` (input of vrpn server) will be handled by the server and its related `infos` pulled by the BlenderVR VRPN client. See [VRPN Getting Started](#) page for more information.

1. Install

- Download [VRPN 07.33.zip](#) and unzip it (e.g. into `$INSTALL_DIR/plugins/vrpn`), or directly clone from the git repository: `git clone https://github.com/vrpn/vrpn.git`.

- Follow the compilation instructions from the [VRPN Getting Started](#) page.

Warning: As for today, VRPN default compilation flags will build the `vrpn.so` (shared object) for **Python2.7**. As BlenderVR runs with **Python3.X**, you'll need to add the following compilation flags to the `cmake` instruction (adapt hereabove paths to your architecture):

```
$ -DVRPN_BUILD_PYTHON=OFF
$ -DVRPN_BUILD_PYTHON_HANDCODED_2X=OFF
$ -DVRPN_BUILD_PYTHON_HANDCODED_3X=ON
$ -DPYTHON_INCLUDE_DIR=/<adapt_to_your_architecture>/Headers
$ -DPYTHON_LIBRARY=/<adapt_to_your_architecture>/lib/libpython3.4.dylib
```

2. Test the installation (VRPN itself and its shared object python module)

- Test the installation with the binaries you just compiled, using the pair `vrpn_print_devices` and `vrpn_server` (see [VRPN Getting Started](#) for more details).
- Test the compilation of the `vrpn.so` (Unix) `vrpn.pyd` (Windows) module: launch `python3` and type:

```
import sys
sys.path.append('<path-to-the-folder-hodling-vrpn.so>')
import vrpn
for elmt in dir(vrpn): print(elmt)
```

The output should look like:

```
__doc__
__file__
__loader__
__name__
__package__
__spec__
error
quaternion
receiver
sender
```

3. First run in BlenderVR

- Add the path of the VRPN python directory (that contains `vrpn.so`) to your configuration file `library` path.
- Grab a VRPN device, find it's name in the VRPN server, and define it's associated callback name in the `vrpn` sub-section of the configuration file (said name is yours to chose).
- Define the callback method in the `blender-name-scene.processor.py` attached to your scene (said method being named after the callback declared in the configuration file).

Use the `basic-vrpn.blend` scene in `$INSTALL_DIR/samples/plugin/vrpn/basic-vrpn/` and its associated processor file for an example of how to use VRPN in BlenderVR, modifying the VRPN callbacks (methods) names `space_navigator_analog` and `space_navigator_button` to fit yours.

Oculus Rift DK2

This section guides you through the installation required to run the [Oculus Rift DK2](#) with BlenderVR. The install procedure involves to:

- Make sure you followed the instructions to add the Oculus Python Library to BlenderVR sources

- Install and test the [Oculus Runtime](#) for your architecture: version 0.6 on Windows (to support Extended mode) and 0.5 on Linux/OSX (last version available on these architectures).

Note: OSX: check on the web to setup your screen configuration for the rift (mirrored display, 90 rotation, etc.). For example: [\[here\]](#)

- Modify BlenderVR configuration file for dk2 support:

Modify your configuration file to add the plugin users. Check the Oculus configuration examples for Desktop Oculus DK2 and Dual Oculus DK2 setups.

Once done, launch the `basic-dk2.blend` scene in `$INSTALL_DIR/samples/plugin/hmd/` to check installation.

First Run

Note: After the installation you should make sure everything is working before going on your own. For this first run you will need at least the basic sample from the samples repository.

Document Sections

- Launch BlenderVR
- Open the Simulation File
- Edit the Configuration File
- Run

Launch BlenderVR

Start by opening the BlenderVR GUI (see Running BlenderVR in the Install section). Although in the future you can launch it via a shortcut, for the first run it's better to do it via command-line, to catch any unexpected error. It is advised to understand how to manipulate the User Interface before going any further.

Open the Simulation File

In the Simulation File select the `basic.blend` file. Mark `NameLink` for the `basic.processor.py` to be automatically selected as the Processor File.

Edit the Configuration File

You now need a valid Configuration File to run BlenderVR. Make sure to follow the correct instructions and set a correct `blender` and `blenderplayer` paths.

Remember to select and load the configuration file.

Note: For the initial test it's recommended to create a single screen with a mono buffer setup.

Run

If there are no errors in the configuration tab, change to the `Run` tab and hit `Start` .

Congratulations, you can now try the other sample files, configuration options and finally bring your own `.blend` files into BlenderVR.

User Interface

We dissociate the *controlling interface* from the *virtual environment*.

The *controlling interface* (here called `console`) is the graphical user interface (GUI) that controls BlenderVR. The *virtual environment* is the part of the simulation that runs on each node inside `blenderplayer`.

To simplify, the `console` is run by the user, use PySide but cannot import `bge` python module whereas the *virtual environment* is run by `blenderplayer`, don't have any GUI and can import `bge` python module.

From the processor file perspective there is even a third mode, the `update loader`. This mode is a glue between the previous ones. In the `update loader` the `.blend` file is changed on-the-fly so when it runs into the *virtual environment* it interacts with the `console` and with eventual interaction devices.

Document Sections

- Console
 - Configuration File
 - Active Screen Set
 - Simulation File
 - Start/Stop
 - Debug Window per Screen
 - Standard/Error Outputs
 - Log Level
- Daemons

Console

The so called `console` is the GUI of BlenderVR. It allows you to choose the configuration file, the screen set to use, the simulation file (`.blend`) or to run BlenderVR.

By default, the `console` does not “know” anything. You have to manually set configuration file, active screen-set, simulation file ... However, it stores these relevant information in its internal data store path (see above). So you have to set these information the first time you run BlenderVR and they remain active (across different running) until you change it with the GUI.

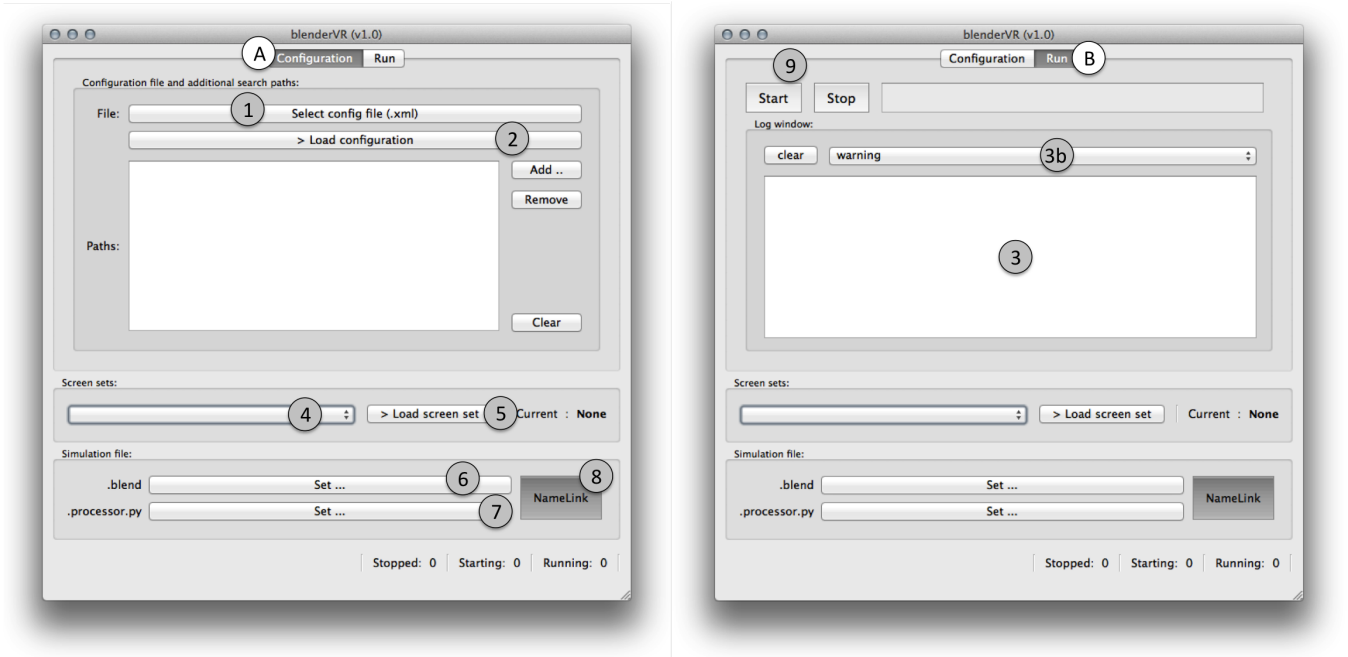


Figure 3.1: Console Graphical User Interface (left: Configuration tab, right: Run tab)

Configuration File

You can specify the XML file inside the configuration tab (A-1). Don't forget to click on `> Load configuration` (A-2) to ask BlenderVR to read the configuration file (and store it inside its internal data file) ! You should go to run tab (B) and select debug (3b) inside the log window (3) to see if there is bug inside your configuration file.

Active Screen Set

You can choose any screen set (4) that is defined inside your XML configuration file. You also must click on `> Load screen set` (5) to make it active (and register it for further BlenderVR usage). The current active screen set is displayed on the right.

Simulation File

Here, you must select the `.blend` file you want to load (6). For the beginning, you should try the basic `.blend`, that you can get from the samples repository. You can manually select a processor file (7) or activate the `NameLink` (8) for BlenderVR to automatically look for a `<name_of_blender_scene>.processor.py` file in the directory of the `.blend` file. You will learn to create your own `.blend` scenes and processor files via the samples and going through the [BlenderVR API](#).

Start/Stop

When everything is defined you can try to start/stop (9) by going to the Run tab. Have a look at the main log window below the Start and the Stop buttons.

Debug Window per Screen

Once the configuration file and the screen set are loaded, you can also have a look at the per screen log window : top screen menu `Windows > Screens` and select the screen that you want to debug. We suggest, at the beginning, to debug your XML

configuration file, to set it to debug mode and activate `Standard output` and `Error output`.

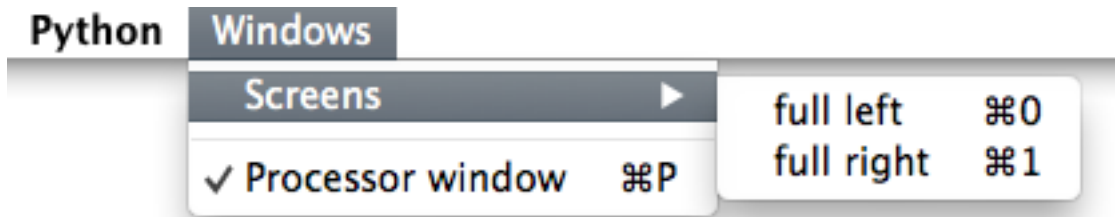


Figure 3.2: Top screen Windows menu.

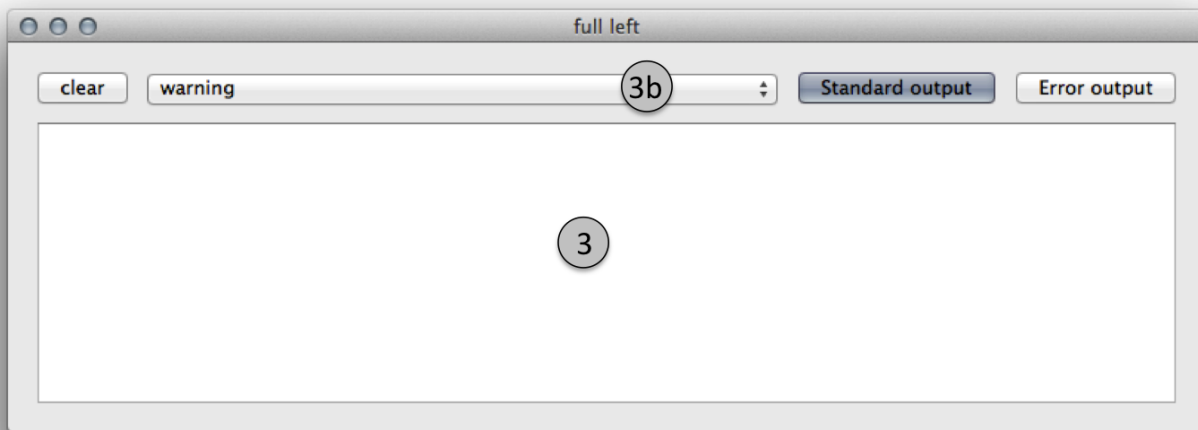


Figure 3.3: Screen window for screen named `full left` in the configuration file.

Standard/Error Outputs

They will display `stdout` and `stderr` of the instance of `blenderplayer`. Thus, you will see if there is a bug while running it. When `blenderplayer` runs correctly, you should disable these options.

Log Level

The log level (3b) is useful when `blenderplayer` runs properly. It can display errors of your processor file in the log window (3).

Daemons

The `console` use one daemon per screen. The daemon is a python script that:

- Connects by network to the `console` and interact with it.
- Start the instance of `blenderplayer` (for the “virtual environment”) when required.

- Catch blenderplayer’s `stdout` and `stderr` to address them to the `console` if requested.
- Kill blenderplayer if the “virtual environment” don’t gently stop on `console` request.
- ...

In other words, the daemon manages blenderplayer. It runs on the computer that will run the blenderplayer instance.

Under Linux, this daemon becomes a real UNIX daemon (fork, close input and output ...).

Note: The daemon script is included inside BlenderVR - you don’t have to tweak it.

Processor File

We want to reduce the impact of BlenderVR on the blender file (`.blend`). For instance, all the interactions issued from the plugins (VRPN, OSC ...) don’t have to be defined inside the `.blend` file, since they do not exists outside BlenderVR development frame. Moreover, elements to synchronize interaction from master to slaves cannot be defined inside `.blend` file.

BlenderVR thus introduces the notion of processor file. It is a Python file associated with the `.blend` that contains all the interactions required to use the `.blend` file within BlenderVR. By default (and you should not change it) this file is in the same folder than the `.blend` file and its name is the name of the blender file minus `.blend`, but post-fixed by `.processor.py`. For instance, the processor file of `simple.blend` is `simple.processor.py`.

Refer to the [Complete API](#) for all the available commands and functionality.

Document Sections

- Minimal Processor File
- Basic Processor File
- Debugging Processor Through Log Messages
- Keyboard and Mouse
- Choose Objects to Synchronize
- Processor Inheritance
- Master-Slaves Communication
- Stream: Processor as Synchronized Object
- One-Shot: Specifically Send a Data
- Run() Method
- Console-“Virtual Environment” Communication

Minimal Processor File

The minimal processor file contains:

```
import blendervr

if blendervr.is_virtual_environment():
    class Processor(blendervr.processor.getProcessor()):
```

```

def __init__(self, parent):
    super(Processor, self).__init__(parent)

elif blendervr.is_console():
    class Processor(blendervr.processor.getProcessor()):
        def __init__(self, console):
            super(Processor, self).__init__(console)

```

Basic Processor File

Unlike the Minimum Processor File, this one actually does something (in this case it synchronizes all the objects between the master and the slaves. This is file is fully explained in the [Basic Example](#) of the BlenderVR API.

```

import blendervr

if blendervr.is_virtual_environment():
    import bge

    class Processor(blendervr.processor.getProcessor()):
        def __init__(self, parent):
            super(Processor, self).__init__(parent)

            if self.BlenderVR.isMaster():
                self.BlenderVR.getSceneSynchronizer().\
                    getItem(bge.logic).activate(True, True)

elif blendervr.is_creating_loader():
    import bpy

    class Processor(blendervr.processor.getProcessor()):
        def __init__(self, creator):
            super(Processor, self).__init__(creator)

elif blendervr.is_console():
    class Processor(blendervr.processor.getProcessor()):
        def __init__(self, console):
            global try_wait_user_name, try_chooser, try_console_arc_balls
            super(Processor, self).__init__(console)

        def useLoader(self):
            return True

```

Debugging Processor Through Log Messages

As you have probably seen in Debug window per screen , the output of blenderplayer is not displayed by default in the console during BlenderVR runs.

Thus, you cannot use basic `print` python commands to help you while debugging.

You should instead use the BlenderVR standard logger usable inside any BlenderVR object (due to inheritance):

```
self.logger.debug("blah blah ...")
```

`blah blah ...` is whatever you want, comma separated, as long as there is a “stringification” method (`__str__`) for each element. The `logger` object inherits from python `logging` module. Thus, you can replace `debug` by `info`, `warning`, `error`,

`critical`. Depending on the log window level selection (see the screen window of the Run tab of the Console), you will see your message or not.

You can also use `self.logger.log_traceback(False)` to display the traceback of your program. `True` in parenthesis means an error, then BlenderVR will stop running in “Virtual Environment”. This traceback is available inside as well as outside an exception.

There is also `self.logger.log_position()` that simply displays the position of the calling method in debug level.

Keyboard and Mouse

You can get access to keyboard and mouse information of the master node by defining the `keyboardAndMouse` method. The `info` provided has the same format than any provided through the VRPN plugin.

You can use a `logger` to see what is contained inside the `info` argument. You can also have a look at the `simple.processor.py` file inside `simple` sample folder to get an example of how to use this method.

Choose Objects to Synchronize

By default, BlenderVR doesn’t synchronize scene objects (blacklisting for efficiency issues). You must specify the elements you want to synchronize by explicitly flagging the objects to synchronize by the master node:

```
# synchronizer.objects.getItem(enable, recursive = True)
# synchronizer.objects.item_base.Base.activate(enable, recursive = True)
if self.BlenderVR.isMaster():
    self.BlenderVR.getSceneSynchronizer().getItem(bge.logic).activate(True, True)
```

This method will synchronize (first `True` as `activate` parameter) all elements recursively (second `True` as `activate` parameter) from the `bge.logic` (that is the root of the `.blend` file). In other words, it will activate all the objects of the scene. You can also synchronize only a few objects by applying this call to each item (the objects as parameter of `getItem`).

Processor Inheritance

We commonly use the same interactions on different scenes. For instance, the Head Control Navigation system is useful on most scenes. BlenderVR allows the developer to have a “generic” processor that all other processors will be able to use by inheritance. You can add an intermediate processor by adding a line at the beginning of your processor:

```
blendervr.processor.appendProcessor(os.path.join(BlenderVR_root, 'samples', 'processors.py'))
```

This line specifically adds the `processors.py` (from folder `samples` of BlenderVR) processor to each processor in the `sample` folder. This processor proposes:

Inside the `virtual environment`:

- **Head control navigation system** to navigate through the world just with your head as joystick (see `mountain sample`)
- **Laser** interaction, useful when you want to select objects from your scene (see `chess sample`)
- **Viewpoint manipulation** in the same way than blender uses in its graphic window (see `simple sample` and press ‘v’ to use it)
- **‘Q’ to quit** clean quit the “Virtual Environment”

Inside the `console`:

- **User interface** that can include buttons for Head control navigation system

We suggest you to have a look at the processor files inside the sample folder before you write your owns.

Master-Slaves Communication

Inside the processors, you can send data from the master to the slaves.

Note: There is no solution to send data from any slave to the master nor any other slave !

There are two mechanisms to send data to the slaves: `stream` and `one-shot`.

Stream: Processor as Synchronized Object

You can register your processor as a synchronized object. As such, at each frame, the synchronizer will ask the master's processor (through `getSynchronizerBuffer()` method) the buffer to send to the slaves. Then, if the buffer is not empty (`getSynchronizerBuffer()` doesn't return `None`), each slave, *in the same frame*, will receive it through its `processSynchronizerBuffer()` method.

To register your processor, you must call from the constructor of your “virtual environment” processor:

```
self.BlenderVR.addObjectToSynchronize(self, 'main processor')
```

The argument in single quote is the name of the processor used by the synchronizer to disambiguate between all synchronized objects. You can use anything else than `main processor`, but this is a good default choice.

As an example, you can have a look at the `simple.processor.py` in `sample` folder, where `try_use_stream_between_master_and_slave` is set to `True`.

One-Shot: Specifically Send a Data

When you don't need to send data through a stream (ie.: each frame), you can send one data sometime with `sendToSlaves/receivedFromMaster` methods. The first argument is a string describing your data whereas the second argument is the data.

Beware that this processing use encapsulation and **JSON** to encode and decode the data. That is heavier than the stream mechanisms and must be applied to data with a low update rate only.

As an example, you can have a look at the `simple.processor.py` in `sample` folder, when you press 's' (see method `keyboardAndMouse`) on the master.

Run() Method

The `run` method will be called at each frame on the master node. Thus, if you need to process something (register a data, update a value, etc.), you can add whatever you want here. To process something on the slaves, you should unlock it with previous mechanisms to send data from the master to the slaves.

Console-“Virtual Environment” Communication

You can send data from the master “virtual environment” node to the console (`sendToConsole/receivedFromVirtualEnvironment`). You can also do the opposite, from console to “virtual environment” (`sendToVirtualEnvironment/receivedFromConsole`).

As usual, the `simple.processor.py` file shows the use of this mechanism. If you set `try_wait_user_name` to `True`, then the “virtual environment” is paused. To unlock it, you must type a name in the processor window from the console and you click on `Set user name`. Then, the name will be sent to the master node that will display it and answer the console.

Configuration File

The BlenderVR XML configuration file is loaded by the console to get the architecture related information to run BlenderVR and send it to each virtual environment rendering node (see processor file templates).

Note: For a comprehensive overview of the file specification access its architecture documentation.

Note: Example configuration files can be found in the `$INSTALL_DIR/source/configurations` directory.

Document Sections

- [Operating System Disclaimer](#)
- [Desktop Basic](#)
- [Desktop Stereo 3D](#)
- [Desktop Networked](#)
- [Desktop Oculus DK2](#)
- [Dual Oculus DK2](#)
- [CAVE](#)

Operating System Disclaimer

To avoid redundancy and unsynced and outdated documents the instructions to all supported operating system are condensed together in this page.

In order to use any of the example configuration files in this page you need to first modify it according to the platform of each of your machines.

- [Windows](#)
- [Mac](#)
- [Linux](#)

Windows

All samples in this page were written for Windows. All that is required to change is the corresponding path of blender, your system and the blenderplayer executable.

In those examples we assume the BlenderVR files are in `C:/BlenderVR/`. So the source is in `C:/BlenderVR/source/` and the samples in `C:/BlenderVR/samples/`.

Change the folders accordingly to your installation.

Mac

All samples in this page were written for Windows. There are sections that are not needed for Mac, besides you need to change the corresponding paths of your system.

The first thing that needs to be adapted is the filepaths. The following are assorted examples of filepaths in Windows and Mac.

Windows:

```
<starter blender='C:/BlenderVR/blender/blender.exe' anchor='C:/BlenderVR/samples'>
<system root='C:/BlenderVR/source' anchor='C:/BlenderVR/samples'>
<blenderplayer executable='C:/BlenderVR/blender/blenderplayer.exe' />
<login remote_command="ssh windows_login@192.168.0.1" python="C:/Python3.4/Python.exe" />
<library path="C:/BlenderVR/venv/lib/python3.4/site-packages" />
```

Mac:

```
<starter blender='/Users/MYUSER/BlenderVR/blender/blender.app/Contents/MacOS/blender' anchor='/Users/MYU
<system root='/Users/MYUSER/BlenderVR/source' anchor='/Users/MYUSER/BlenderVR/samples'>
<blenderplayer executable='/Users/MYUSER/BlenderVR/blender/blenderplayer.app/Contents/MacOS/blenderplaye
<login remote_command="ssh mac_login@192.168.0.1" python="/Users/MYUSER/BlenderVR/venv/bin/python3.4" />
<library path="/Users/MYUSER/BlenderVR/venv/lib/python3.4/site-packages" />
```

Notice that in Mac the Blender binary is inside the bundle (blender.app), as well as the Blenderplayer (blenderplayer.app). Also /Users/MYUSER/ is to be replaced by the location of your BlenderVR installation.

The second thing is the environment XML element which is not required in Mac. So the lines below are to be suppressed in the Mac configuration files.

```
<daemon transmit='True'>
  <environment>SystemRoot=C:/Windows</environment>
</daemon>
```

Linux

All samples in this page were written for Windows. There are sections that are slightly different for Linux. Besides you need to change the corresponding paths of your system.

The first thing that needs to be adapted is the filepaths. The following are assorted examples of filepaths in Windows and Linux.

Windows:

```
<starter blender='C:/BlenderVR/blender/blender.exe' anchor='C:/BlenderVR/samples'>
<system root='C:/BlenderVR/source' anchor='C:/BlenderVR/samples'>
<blenderplayer executable='C:/BlenderVR/blender/blenderplayer.exe' />
<login remote_command="ssh windows_login@192.168.0.1" python="C:/Python3.4/Python.exe" />
<library path="C:/BlenderVR/venv/lib/python3.4/site-packages" />
```

Linux:

```
<starter blender='/home/MYUSER/BlenderVR/blender/blender' anchor='/home/MYUSER/BlenderVR/samples'>
<system root='/home/MYUSER/BlenderVR/source' anchor='/home/MYUSER/BlenderVR/samples'>
<blenderplayer executable='/home/MYUSER/BlenderVR/blender/blenderplayer' />
<login remote_command="ssh linux_login@192.168.0.1" python="/home/MYUSER/BlenderVR/venv/bin/python3.4"/>
<library path="/home/MYUSER/BlenderVR/venv/lib/python3.4/site-packages" />
```

The location /home/MYUSER/ is to be replaced by the location of your BlenderVR installation.

The second thing is the environment XML element which is not required in Linux. So the lines below are to be suppressed in the Mac configuration files.


```
<daemon transmit='True'>
  <environment>SystemRoot=C:/Windows</environment>
</daemon>
```

Finally, Linux allows you to specify a unique `<environment>DISPLAY=:0.0</environment>` element to specify in which display a screen should run. For example:

```
(...)
<screen name="console" computer="Any">
  <display options="-w 400 400">
    <environment>DISPLAY=:0.0</environment>
    <graphic_buffer buffer="mono" user='user A' eye="middle"/>
  </display>
</screen>
(...)
```

Desktop Basic

This is a very basic configuration file. There is only one computer and one user defined, and there are three screens:

1. **Fullscreen:** plays the .blend file in fullscreen.
2. **Console:** plays the .blend file in a small window.
3. **Split:** plays the .blend file in two small windows, side-by-side, completing each other.

```
<?xml version="1.0"?>
<blendervr>

  <starter blender='C:/BlenderVR/blender/blender.exe'>
    <config name='Fullscreen'>fullscreen</config>
    <config name='Console'>console</config>
    <config name='Split'>console half left, console half right</config>
  </starter>

  <users>
    <user name="user A"/>
  </users>

  <computers>
    <system>
      <daemon transmit='True'>
        <environment>SystemRoot=C:/Windows</environment>
      </daemon>
      <blenderplayer executable='C:/BlenderVR/blender/blenderplayer.exe' />
    </system>
    <computer name='Any' hostname='*' />
  </computers>

  <screens>
    <screen name="fullscreen" computer="Any">
      <display options="-f">
        <graphic_buffer buffer="mono" user='user A' eye="middle"/>
      </display>
      <wall>
        <corner name="topRightCorner">1.0, 1.0, -1.0</corner>
        <corner name="topLeftCorner">-1.0, 1.0, -1.0</corner>
        <corner name="bottomRightCorner">1.0, -1.0, -1.0</corner>
      </wall>
    </screen>
  </screens>
```

```

</screen>

<screen name="console" computer="Any">
  <display options="-w 400 400">
    <graphic_buffer buffer="mono" user='user A' eye="middle"/>
  </display>
  <wall>
    <corner name="topRightCorner">1.0, 1.0, -1.0</corner>
    <corner name="topLeftCorner">-1.0, 1.0, -1.0</corner>
    <corner name="bottomRightCorner">1.0, -1.0, -1.0</corner>
  </wall>
</screen>

<screen name="console half left" computer="Any">
  <display options="-w 400 400 200 300">
    <graphic_buffer user='user A' />
  </display>
  <wall>
    <corner name="topRightCorner">0.0, 1.0, -1.0</corner>
    <corner name="topLeftCorner">-1.0, 1.0, -1.0</corner>
    <corner name="bottomRightCorner">0.0, -1.0, -1.0</corner>
  </wall>
</screen>

<screen name="console half right" computer="Any">
  <display options="-w 400 400 600 300">
    <graphic_buffer user='user A' />
  </display>
  <wall>
    <corner name="topRightCorner">1.0, 1.0, -1.0</corner>
    <corner name="topLeftCorner">0.0, 1.0, -1.0</corner>
    <corner name="bottomRightCorner">1.0, -1.0, -1.0</corner>
  </wall>
</screen>

</screens>

<plugins>
</plugins>

</blendervr>

```

Desktop Stereo 3D

This is a very basic configuration file. There is only one computer and one user defined, and there are three screens:

1. **Fullscreen 2D**: plays the `.blend` file in fullscreen without stereo 3d.
2. **Stereo 3D - Side by Side**: plays the `.blend` file in a stereo 3d fullscreen in side by side mode.
3. **Stereo 3D - Quadbuffer**: plays the `.blend` file in a stereo 3d fullscreen with shutter glasses.

The only differences between those three modes are the `display` options. A screen need as many `display_buffer` items as eyes being rendered.

Simply said, the stereo 3d screens will need the `left` and `right` buffers, while the 2d screen only needs the `mono` buffer.

Note: For more advanced `display_buffer` arrangements check the [CAVE](#) example.

Apart from the `display_buffers`, the display options are considerably different between the screens.

- **Fullscreen 2D:** `<display options="-f 1920 1080">`
- **Stereo 3D - Side by Side:** `<display options="-f 1920 1080 -s sidebyside">`
- **Stereo 3D - Quadbuffer:** `<display options="-f 1920 1080 24 120 -s hwpageflip">`

Those options are passed straight as command-line arguments to the `blenderplayer`. For a comprehensive list of arguments run `blenderplayer` with the `--help` option.

For *Fullscreen 2D* all you need to do is to specify the fullscreen mode `-f`, and the screen resolution.

For *Stereo 3D - Side by Side*, besides the above, you need to specify the stereo 3d mode, `-s sidebyside`.

For *Stereo 3D - Quadbuffer* we specify the stereo 3d mode, `-s hwpageflip`, and force the screen bits per pixel, 24, and the frequency, 120. This is the shuttering speed of the active shutter glasses.

You can't specify the frequency without defining the bits first.

Note: In order to use the `hwpageflip` mode your graphic card must support Quadbuffer natively.

```
<?xml version="1.0"?>
<blendervr>

  <starter blender='C:/BlenderVR/blender/blender.exe'>
    <config name='Fullscreen 2D'>fullscreen</config>
    <config name='Stereo 3D - Side by Side'>sidebyside</config>
    <config name='Stereo 3D - Quadbuffer'>quadbuffer</config>
  </starter>

  <users>
    <user name="user A"/>
  </users>

  <computers>
    <system>
      <daemon transmit='True'>
        <environment>SystemRoot=C:/Windows</environment>
      </daemon>
      <blenderplayer executable='C:/BlenderVR/blender/blenderplayer.exe' />
    </system>
    <computer name='Any' hostname='*' />
  </computers>

  <screens>

    <screen name="fullscreen" computer="Any">
      <display options="-f 1920 1080">
        <graphic_buffer buffer="mono" user='user A' eye="middle"/>
      </display>
      <wall>
        <corner name="topRightCorner">1.0, 1.0, -1.0</corner>
        <corner name="topLeftCorner">-1.0, 1.0, -1.0</corner>
        <corner name="bottomRightCorner">1.0, -1.0, -1.0</corner>
      </wall>
    </screen>
```

```

<screen name="sidebyside" computer="Any">
  <display options="-f 1920 1080 -s sidebyside">
    <graphic_buffer buffer="left" user='user A' eye="left" />
    <graphic_buffer buffer="right" user='user A' eye="right" />
  </display>
  <wall>
    <corner name="topRightCorner">1.0, 1.0, -1.0</corner>
    <corner name="topLeftCorner">-1.0, 1.0, -1.0</corner>
    <corner name="bottomRightCorner">1.0, -1.0, -1.0</corner>
  </wall>
</screen>

<screen name="quadbuffer" computer="Any">
  <display options="-f 1920 1080 24 120 -s hwpageflip">
    <graphic_buffer buffer="left" user='user A' eye="left" />
    <graphic_buffer buffer="right" user='user A' eye="right" />
  </display>
  <wall>
    <corner name="topRightCorner">1.0, 1.0, -1.0</corner>
    <corner name="topLeftCorner">-1.0, 1.0, -1.0</corner>
    <corner name="bottomRightCorner">1.0, -1.0, -1.0</corner>
  </wall>
</screen>

</screens>

<plugins>
</plugins>

</blendervr>

```

Desktop Networked

This is an extension of the [Desktop Basic](#) with basic network functionality. There are two computers (the master and the slave) and either is tied to a user. The screens are analog to the previous ones:

1. **Fullscreen Dual:** plays the .blend file in fullscreen in both computers.
2. **Fullscreen Left / Right:** plays the .blend file in fullscreen in either computer.
3. **Console Dual:** plays the .blend file in a small window in both computers.
4. **Console Left / Right:** plays the .blend file in a small window in either computer.

It's important to make sure the master computer can connect to the slave and to itself using the specified `ssh` command. Also, don't underestimate the console screens, they are great for debugging.

```

<?xml version="1.0"?>
<blendervr>

  <starter blender='C:/BlenderVR/blender/blender.exe' anchor='C:/BlenderVR/samples'>
    <config name='Fullscreen Dual'>full left, full right</config>
    <config name='Fullscreen Left'>full left</config>
    <config name='Fullscreen Right'>full right</config>
    <config name='Console Dual'>console left, console right</config>
    <config name='Console Left'>console left</config>
    <config name='Console Right'>console right</config>
  </starter>

```

```

<users>
  <user name="user A"/>
  <user name="user B"/>
</users>

<computers>

  <computer name='Left' hostname='192.168.0.1'>
    <system root='C:/BlenderVR/source' anchor='C:/BlenderVR/samples'>

      <daemon transmit='True'>
        <environment>SystemRoot=C:/Windows</environment>
      </daemon>

      <blenderplayer executable='C:/BlenderVR/blender/blenderplayer.exe' />
      <login remote_command="ssh master@192.168.0.1" python="C:/Python3.4/Python.exe" />
    </system>
  </computer>

  <computer name='Right' hostname='192.168.0.2'>
    <system root='Z:/BlenderVR/source' anchor='Z:/BlenderVR/samples'>

      <daemon transmit='True'>
        <environment>SystemRoot=C:/Windows</environment>
      </daemon>

      <blenderplayer executable='Z:/BlenderVR/blender/belnderplayer.exe' />
      <login remote_command="ssh slave@192.168.0.2" python="D:/MyPython/Python.exe" />
    </system>
  </computer>

</computers>

<screens>

  <screen name="console left" computer="Left">
    <display options="-w 720 450 720 450">
      <graphic_buffer buffer="mono" user='user A' eye="middle"/>
    </display>
    <wall>
      <corner name="topRightCorner">2.16, 1.35, -1.0</corner>
      <corner name="topLeftCorner">-2.16, 1.35, -1.0</corner>
      <corner name="bottomRightCorner">2.16, -1.35, -1.0</corner>
    </wall>
  </screen>

  <screen name="console right" computer="Right">
    <display options="-w 720 450 720 450">
      <graphic_buffer buffer="mono" user='user B' eye="middle"/>
    </display>
    <wall>
      <corner name="topRightCorner">2.16, 1.35, -1.0</corner>
      <corner name="topLeftCorner">-2.16, 1.35, -1.0</corner>
      <corner name="bottomRightCorner">2.16, -1.35, -1.0</corner>
    </wall>
  </screen>

  <screen name="full left" computer="Left">

```

```

<display options="-w 720 900 720 900">
  <graphic_buffer user='user A' />
</display>
<wall>
  <corner name="topRightCorner">1.0, 1.0, -1.0</corner>
  <corner name="topLeftCorner">0.0, 1.0, -1.0</corner>
  <corner name="bottomRightCorner">1.0, -1.0, -1.0</corner>
</wall>
</screen>

<screen name="full right" computer="Right">
  <display options="-w 720 900 0 900">
    <graphic_buffer user='user B' />
  </display>
  <wall>
    <corner name="topRightCorner">0.0, 1.0, -1.0</corner>
    <corner name="topLeftCorner">-1.0, 1.0, -1.0</corner>
    <corner name="bottomRightCorner">0.0, -1.0, -1.0</corner>
  </wall>
</screen>

</screens>

<plugins>
</plugins>

</blendervr>

```

Desktop Oculus DK2

This configuration is composed of three screens: the main one to be used for deployment, and two others used for debugging and testing:

1. **Oculus DK2 Fullscreen:** plays the .blend file in fullscreen in Oculus DK2 mode.
2. **Oculus DK2 Debug:** plays the .blend file in a small window in Oculus DK2 mode.
3. **Console:** plays the .blend file in a small window in the computer.

Besides that, the configuration file example below now defines the Oculus DK2 plugin:

```

<oculus_dk2>
  <user viewer='user A' computer='Any' processor_method="user_position" />
</oculus_dk2>

```

with parameters such as `viewer` (which user is concerned) or `processor_method` (optional callback that processes oculus tracking data in the processor file, see [sample files](#) for example implementations).

A computer can control only a single Oculus, for a multiple Oculus installation you need networked computers as explained in the [Dual Oculus DK2](#) example.

Note: If you experience a “non full-screen” with the `fullscreen` configuration proposed above, or can’t drag the blender-player window to the Oculus screen, try to replace `<display options="-f -s sidebyside">` with `<display options="-w 1920 1080 0 0 -s sidebyside">`. If it works, try changing the two last values (0 0) - corresponding to blenderplayer window left/top coordinate - to values that will position the blenderplayer window on the Oculus screen at BlenderVR start.

Note: On Windows, the shortcut Shift + Windows + Left/Right enables you to switch the monitor on which is displayed the current window (even a fullscreen one).

```
<?xml version="1.0"?>
<blendervr>

  <starter blender='C:/BlenderVR/blender/blender.exe' >
    <config name='Oculus DK2 Fullscreen'>oculus dk2 full</config>
    <config name='Oculus DK2 Debug'>oculus dk2 debug</config>
    <config name='Console'>console</config>
  </starter>

  <users>
    <user name="user A"/>
  </users>

  <computers>

    <system>
      <daemon transmit='True'>
        <environment>SystemRoot=C:/Windows</environment>
      </daemon>
      <blenderplayer executable='C:/BlenderVR/blender/blenderplayer.exe' />
    </system>
    <computer name='Any' hostname='*' />

  </computers>

  <screens>

    <screen name="oculus dk2 full" computer="Any">
      <display options="-f -s sidebyside">
        <graphic_buffer buffer="left" user='user A' eye="left"/>
        <graphic_buffer buffer="right" user='user A' eye="right"/>
      </display>
      <hmd model="oculus_dk2">
        <left>
          <corner name="topRightCorner">1.0, 1.0, -1.0</corner>
          <corner name="topLeftCorner">-1.0, 1.0, -1.0</corner>
          <corner name="bottomRightCorner">1.0, -1.0, -1.0</corner>
        </left>
        <right>
          <corner name="topRightCorner">1.0, 1.0, -1.0</corner>
          <corner name="topLeftCorner">-1.0, 1.0, -1.0</corner>
          <corner name="bottomRightCorner">1.0, -1.0, -1.0</corner>
        </right>
      </hmd>
    </screen>

    <screen name="oculus dk2 debug" computer="Any">
      <display options="-w 720 450 720 450 -s sidebyside">
        <graphic_buffer buffer="left" user='user A' eye="left"/>
        <graphic_buffer buffer="right" user='user A' eye="right"/>
      </display>
      <hmd model="oculus_dk2">
        <left>
          <corner name="topRightCorner">1.0, 1.0, -1.0</corner>
          <corner name="topLeftCorner">-1.0, 1.0, -1.0</corner>
        </left>
      </hmd>
    </screen>

  </screens>
</blendervr>
```

```

        <corner name="bottomRightCorner">1.0, -1.0, -1.0</corner>
    </left>
    <right>
        <corner name="topRightCorner">1.0, 1.0, -1.0</corner>
        <corner name="topLeftCorner">-1.0, 1.0, -1.0</corner>
        <corner name="bottomRightCorner">1.0, -1.0, -1.0</corner>
    </right>
</hmd>
</screen>

<screen name="console" computer="Any">
    <display options="-w 400 400">
        <graphic_buffer buffer="mono" user='user A' eye="middle"/>
    </display>
    <wall>
        <corner name="topRightCorner">1.0, 1.0, -1.0</corner>
        <corner name="topLeftCorner">-1.0, 1.0, -1.0</corner>
        <corner name="bottomRightCorner">1.0, -1.0, -1.0</corner>
    </wall>
</screen>

</screens>

<plugins>

    <oculus_dk2>
        <user viewer='user A' computer='Any' processor_method="user_position" />
    </oculus_dk2>

</plugins>
</blendervr>

```

Dual Oculus DK2

This is a mix of the [Desktop Networked](#) with the [Desktop Oculus DK2](#) examples. We now have a server which is running in Mac, while the client is in Windows.

Each computer has an Oculus DK2 device connected to it. And each device controls a `user` point of view. We skipped the debug and console configurations in this example, but they can be copied from the previous ones.

It's important to make sure the master computer can connect to the slave and to itself using the specified `ssh` command.

Note: The same configuration file can be used by both computers by changing only the `starter` section for each corresponding master station.

```

<?xml version="1.0"?>
<blendervr>
    <starter blender='/Users/MYUSER/BlenderVR/blender/blender.app/Contents/MacOS/blender' hostname='192.16
        <config name='Oculus DK2 Dual1 Dual1'>oculus dk2 left, oculus dk2 right</config>
    </starter>

    <users>
        <user name="user A"/>
        <user name="user B"/>
    </users>

```



```

<computers>

  <computer name='Left' hostname='192.168.0.1'>
    <system root='/Users/MYUSER/BlenderVR/source' anchor='/Users/MYUSER/BlenderVR/samples'>
      <blenderplayer executable='/Users/MYUSER/BlenderVR/blender/blenderplayer.app/Contents/MacOS/
        <login remote_command="ssh MYUSER@192.168.0.1" python="/Users/MYUSER/BlenderVR/venv/bin/pyth
      </system>
    </computer>

  <computer name='Right' hostname='192.168.0.2'>
    <system root='C:/BlenderVR/source' anchor='C:/BlenderVR/samples'>
      <daemon transmit='True'>
        <environment>SystemRoot=C:/Windows</environment>
      </daemon>
      <blenderplayer executable='C:/BlenderVR/blender/blenderplayer.exe' />
      <login remote_command="ssh slave@192.168.0.2" python="C:/Python3.4/Python.exe" />
    </system>
  </computer>

</computers>

<screens>

  <screen name="oculus dk2 left" computer="Left">
    <display options="-f -s sidebyside">
      <graphic_buffer buffer="left" user='user A' eye="left"/>
      <graphic_buffer buffer="right" user='user A' eye="right"/>
    </display>
    <hmd model="oculus_dk2">
      <left>
        <corner name="topRightCorner">1.0, 1.0, -1.0</corner>
        <corner name="topLeftCorner">-1.0, 1.0, -1.0</corner>
        <corner name="bottomRightCorner">1.0, -1.0, -1.0</corner>
      </left>
      <right>
        <corner name="topRightCorner">1.0, 1.0, -1.0</corner>
        <corner name="topLeftCorner">-1.0, 1.0, -1.0</corner>
        <corner name="bottomRightCorner">1.0, -1.0, -1.0</corner>
      </right>
    </hmd>
  </screen>

  <screen name="oculus dk2 right" computer="Right">
    <display options="-f -s sidebyside">
      <environment>DISPLAY=:0.0</environment>
      <graphic_buffer buffer="left" user='user B' eye="left"/>
      <graphic_buffer buffer="right" user='user B' eye="right"/>
    </display>
    <hmd model="oculus_dk2">
      <left>
        <corner name="topRightCorner">1.0, 1.0, -1.0</corner>
        <corner name="topLeftCorner">-1.0, 1.0, -1.0</corner>
        <corner name="bottomRightCorner">1.0, -1.0, -1.0</corner>
      </left>
      <right>
        <corner name="topRightCorner">1.0, 1.0, -1.0</corner>
        <corner name="topLeftCorner">-1.0, 1.0, -1.0</corner>
        <corner name="bottomRightCorner">1.0, -1.0, -1.0</corner>
      </right>
    </hmd>
  </screen>

```

```

        </right>
    </hmd>
</screen>

</screens>

<plugins>

    <oculus_dk2>
        <user viewer='user A' computer='Left' />
        <user viewer='user B' computer='Right' />
    </oculus_dk2>

</plugins>
</blendervr>

```

CAVE

This more advanced configuration has a few screens but two modes:

1. **Console:** plays the `.blend` file in a small window in the current computer (for debugging).
2. **CAVE:** plays the `.blend` file in a CAVE (floor, front, left and right screens).

This **CAVE** setup is focused on the Linux platform, but it can be adapted for other operating systems as well. The head-tracking system is using the VPRN Plugin system.

The dimensions of this CAVE is 4.8m (width) x 3.0m (height) x 2.7m (depth) (i.e., x, y, z).

The parameters to define in the screens walls, are all relate to the head reference frame looking forward. Meaning, the screen walls corners coordinates are as width, height, depth (i.e., x, z, y). Also the origin of the system is at 0.0 x 0.0 x 1.60 in this particular case.

This also impacts the settings of the head-tracking system (in the plugins `vrpn` tracker element). In this example we are converting the data from the VPRN server so that the translation is also in the head reference frame.

Finally, for an ortostereoscopy experience, the `.blend` file should mimics this - The scene camera initial position should be at 1.6 height looking forward (rotation: 90, 0, 0).

Note: The head-tracker device expects the processor method `user_position`. Since this is also the name of the fallback routine, it doesn't need to be implemented in the Processor File.

```

<?xml version="1.0"?>
<blendervr>

    <starter blender="/mnt/software/blendervr/blender/blender">
        <config name="Console">console</config>
        <config name="CAVE">floor, front, left, right</config>
    </starter>

    <!-- Users -->
    <users>
        <user name="user A"/>
    </users>

    <!-- System -->
    <system>
        <blenderplayer executable="/mnt/software/code/64/tools/blender-git/compile/bin/blenderplayer">

```

```

    <environment>PATH=/mnt/softwares/code/64/bin:/mnt/softwares/bin.sh:/usr/bin:/bin</environment>
    <environment>PYTHONPATH=/mnt/softwares/code/64/python3.2mu</environment>
    <environment>HOME='os.environ["HOME"]'</environment>
  </blenderplayer>
</system>

<!-- Computers -->
<computers>
  <system>
    <login remote_command="ssh `self._attributs_inheritance["hostname"]`"/>
  </system>

  <computer name="Control" hostname="localhost" />
  <computer name="Node 1" hostname="node-`cluster.name`-1" />
  <computer name="Node 2" hostname="node-`cluster.name`-2" />
  <computer name="Node 3" hostname="node-`cluster.name`-3" />
  <computer name="Node 4" hostname="node-`cluster.name`-4" />
</computers>

<!-- Screens -->
<screens>
  <screen name="console" computer="Control">
    <display options="-w 400 400">
      <graphic_buffer buffer="mono" user="user A" eye="middle"/>
    </display>

    <wall>
      <corner name="topRightCorner" 1.0, 1.0, -1.0</corner>
      <corner name="topLeftCorner" -1.0, 1.0, -1.0</corner>
      <corner name="bottomRightCorner" 1.0, -1.0, -1.0</corner>
    </wall>
  </screen>

  <display options="-f -s hwpageflip">
    <graphic_buffer buffer="left" user="user A" eye="left"/>
    <graphic_buffer buffer="right" user="user A" eye="right"/>
    <environment>DISPLAY=:0.0</environment>
  </display>

  <screen name="floor" computer="Node 1">
    <wall>
      <corner name="topRightCorner" 2.4, -1.6, -1.35</corner>
      <corner name="topLeftCorner" -2.4, -1.6, -1.35</corner>
      <corner name="bottomRightCorner" 2.4, -1.6, 1.35</corner>
    </wall>
  </screen>

  <screen name="front" computer="Node 2">
    <wall>
      <corner name="topRightCorner" 2.4, 1.4, -1.35</corner>
      <corner name="topLeftCorner" -2.4, 1.4, -1.35</corner>
      <corner name="bottomRightCorner" 2.4, -1.6, -1.35</corner>
    </wall>
  </screen>

  <screen name="left" computer="Node 3">
    <display><viewport>420, 0, 1500, 1080</viewport></display>
    <wall>

```

```

    <corner name="topRightCorner">  -2.4,  1.4, -1.35</corner>
    <corner name="topLeftCorner">   -2.4,  1.4,  1.35</corner>
    <corner name="bottomRightCorner">-2.4, -1.6, -1.35</corner>
  </wall>
</screen>

<screen name="right" computer="Node 4">
  <display><viewport>420, 0, 1500, 1080</viewport></display>
  <wall>
    <corner name="topRightCorner">   2.4,  1.4,  1.35</corner>
    <corner name="topLeftCorner">   2.4,  1.4, -1.35</corner>
    <corner name="bottomRightCorner"> 2.4, -1.6,  1.35</corner>
  </wall>
</screen>

</screens>

<!-- Plugins -->
<plugins>

  <vrpn>
    <tracker device="HeadCap" host="VRPN_SERVER">
      <transformation>
        <post_translation z="-1.6" />
        <pre_rotation x="1.0" y="1.0" z="1.0" angle="\2.0 * math.pi / 3.0\" />
        <post_rotation x="1.0" y="1.0" z="1.0" angle="\-2.0 * math.pi / 3.0\" />
      </transformation>
      <sensor id="0" processor_method="user_position" users="user A" />
    </tracker>
  </vrpn>

</plugins>

</blendervr>

```

Virtual Reality Private Network (VRPN)

VRPN is a protocol used in Virtual Reality to exchange data with external devices. See <http://www.cs.unc.edu/Research/vrpn/>.

BlenderVR behaves like a VRPN client. At the other end, a VRPN server will host different tracker or sensors that would be as many haptic arms, tracked stereoscopic glasses or Wiimote devices. The server will associate a name to these device, along with a variable “info” that holds the useful information about the considered device.

In BlenderVR, the receiving of VRPN messages and definition of associated methods is done in the `<blender_scene_name>.processor.py` script attached to a scene (see examples in the samples folder).

Document Sections

- [Interaction Setup](#)
- [Example with a Nintendo Wii Controller](#)

Interaction Setup

To be able to interact in your BlenderVR scene with a VRPN compatible interface you will have to:

1. Define the interface in your `vrpn.cfg` script
2. Define the related processor method in the BlenderVR `.xml` configuration script
3. Define the processor method in the `<blender_scene_name>.processor.py` script attached to your BlenderVR scene

Example with a Nintendo Wii Controller

1. In your `vrpn.cfg` file, add:

```
vrpn_WiiMote WiiMote0 1 0 0 1
```

2. In the BlenderVR `.xml` configuration script (e.g. `single.xml`), add:

```
<processor>
  (...)
  <plugins>
    <vrpn>
      <analog device="WiiMote0" host="localhost" processor_method="wiiAnalog"/>
      <button device="WiiMote0" host="localhost" processor_method="wiiButton"/>
    </vrpn>
  </plugins>
</processor>
```

Analog will receive accelerometer data from the WiiMote, button only the pressed button states.

Note: You also need to specify the folder containing your vrpn library in the configuration file.

3. In the `<blend_file_name>.processor.py` script (e.g. `BlenderVR_API.processor.py`), add:

```
import blendervr

if blendervr.is_virtual_environment():
    import bge

    class Processor(blendervr.processor.getProcessor()):
        (...)

        def wiiAnalog(self, info):
            self.logger.info("Analog from Wii through VPRN ", info)

        def wiiButton(self, info):
            self.logger.info("Button from Wii through VPRN ", info)
```

Here, both functions will be executed whenever the VRPN server receives data from the WiiMote (the `wiiButton` when your touch a button, the `wiiAnalog` when you move the WiiMote).

Open Sound Control (OSC)

Routines based on the [OSC protocol](#) (based on UDP) have been added to BlenderVR ([dedicated API](#)) to be able to easily send data to a third party application that would take care of the audio components related to a given VR scene.

You're welcome to use BlenderVR [Sound Rendering Engine](#) (SRE, open source, developed in [Max/MSP](#)) if you don't want to develop your own sound server. While the OSC API allows to easily send OSC messages, said SRE has been designed to receive an process these flags.

Note: BlenderVR [OSC API](#) can easily be adapted to any other OSC client to fathom your own interactions.

Document Sections

- [Interaction Setup](#)

Interaction Setup

This section details how to setup BlenderVR configuration file to work with its [Max/MSP Sound Rendering Engine](#).

In the `<osc>` subsection of the `<plugins>` section in the configuration file are defined the parameters related to OSC and sound rendering, typically:

```
<plugins>

  <-- (...) -->

  <-- general parameters -->
  <osc host='localhost' port='3819' configuration='Laptop SPAT' max_audio_objects='8'>
    <-- user parameters -->
    <user listener='Binaural 1' viewer='user A' />
    <user listener='Binaural 2' viewer='user B' />
    <user listener='Ambisonic' />
    <user listener='Stereo' />
  </osc>
</plugins>
```

with:

- `host / port`: IP adress and port of the OSC server (computer on which the [Max/MSP Sound Rendering Engine](#) is opened).
- `configuration`: flag used to dynamically adapt the [Sound Rendering Engine](#) to a given architecture specific (e.g. load the patches adapted to a given speaker configuration, rendering technique, etc.)
- `max_audio_objects`: flag used to pre-allocate N audio objects in the SRE. BlenderVR will use and reuse (once released) these pre-allocated audio objects to add sound to your scenes. This mecanism allows to limit the maximum DSP usage on the OSC server.
- `user listener / viewer`: these flag allow to associate BlenderVR users (viewers) and [Max/MSP](#) sound rendering techniques (listeners). Here, `<user listener='Binaural 1' viewer='user A' />` associates BlenderVR viewer `user A` to [Max/MSP](#) listener `Binaural 1` (i.e. to a headset in [Max/MSP](#)), e.g. for head-tracking automatic update.

Once you've opened the `BlenderVR_Sound_Rendering_Engine_<version>.maxpat` on your OSC server, the rest of the sound adding process takes place in BlenderVR processor file through the [OSC API](#).

Tip: If you downloaded BlenderVR samples, see the scenes in the `samples/plugin/osc` folder for a direct insight on how to use the OSC API.

Oculus Rift DK2

Setup with one Oculus Rift, without networking

See the Install Plugin Oculus Section for installation notes.

Basic setup

Note: Currently, Blender VR does not support direct mode for the Oculus Rift. To run Oculus Rift with Blender VR, use extended mode (Tools, Rift Display Mode, Extend the HMD to Desktop, Apply in the Windows Oculus Rift Config Utility).

Select main_dk2.xml as a basic configuration file in BlenderVR, making sure you understand what's new there by taking a look at the Oculus configuration file section and select, for example, the basic-dk2.blend in the `samples/plugin/hmd` folder.

Additional info relevant for Linux environments

You need to run `ovrd` in order to have the Oculus communicating with the applications. `ovrd` is part of the Oculus SDK.

All this communication is done via the `/dev/hidraw2` device. The `hidraw` driver provides the RAW interface for HID (Human Interface Devices) USB and Bluetooth.

How to install it? All you need is to run `make install` as indicated in the `README.Linux` file of the SDK.

Additionally, if you run into problems, you may be required to load the OpenGL library from a recent driver. For example by adding `export LD_PRELOAD="/usr/lib/nvidia-346/libGL.so` to your `.bashrc` file (replace-346 by your driver version).

Architecture

Master and Slaves

Communications inside BlenderVR are organized through a master/slaves structure. Although inside virtual environment, all nodes are equivalent, one node is the master. The master computer is the console from the configuration file.

Document Sections

- [Master](#)
- [Slaves](#)

Master

The master node is the one that computes all scene interactions, updates the animations and dispatches them to the slaves. This node have extra possibilities regarding slaves:

- You can access to keyboard and mouse of its graphic window,
- It connects to VRPN to get the information issued by VR devices,
- Dialog from the console are send to it,
- OSC is running on the master,
- ...

Slaves

Playing blender animations on slaves has been reported to conflict with the update coming from the master and may produces flicking.

To avoid that (and restrict calculation of scene updates), slave nodes are suspended (`bge.logic.getCurrentScene().suspend()`) during BlenderVR runs.

Even if you `resume()` the scene, the next execution of BlenderVR will `suspend()` it on the slaves.

Notion of Vehicle

We can see the Virtual Environment as a vehicle: each device is an item of the vehicle (wheel, brake pedal, etc.), the screens are the windows of the vehicle opening on outside (virtual) world, you can “scale” your vehicle to the objects of the scene (microscopes or telescopes are kind of vehicle ...).

In Virtual Environments, each tracker, device, screen, etc. of the real world is defined in its own reference frame. However, everything resides in the same space. So we have to introduce a reference frame change between each device. Instead of device inter-related position, BlenderVR uses a single reference frame in which all device, screen, tracker, etc. will be defined.

This “center” of the real world defines the origin of a vehicle: a bridge between real and virtual worlds. As such, In the virtual world, the vehicle should be attached to blender virtual camera. Hence, if you move the camera, you move the vehicle inside the virtual world.

From another point of view, you can move yourself inside the vehicle without it moving in the virtual world: if you come back at the same place in the Virtual Environment you will have exactly the same display on the screens.

UI - Daemon Network Protocol

First of all, to avoid problem of paths resolutions, the UI must run in the same context (computer and user), than the daemon.

Document Sections

- Base Protocol
- Get/Set Settings
- Simulation

Base Protocol

The communication relies on encapsulation of (command, argument) messages. Each stage of encapsulation is responsible to compose/decompose its information and parse the provided commands. In charge of the central stage to use `blendervr.tools.connector.Client` class to interact with the daemon and use its `send(command, argument = ‘’)` method.

The method `blendervr.tools.connector.Common.composeMessage(command, argument)` must be use to compose a message sent to the peer. On the other side, the method `blendervr.tools.connector.Common.decomposeMessage(message)` must be use to decompose a message and analyze its content.

For instance, if the UI request the blender file name, it should send (supposing client is a `blendervr.tools.connector.Client` object and composer is an import of `blendervr.tools.connector.Common`): `client.send(‘get’, composer.composeMessage(‘simulation’, ‘blender file name’))` For commodity, in the remaining specs, we will write: `(‘get’, (‘simulation’, ‘blender file name’))`

Unless specified, the daemon will reply to request with the same keywords to acknowledge the value. Thus, if the UI request for the current blender file name, the dialog will be (supposing the current blender file is `/home/blender/samples/mountain/mountain.blend`):

- **UI:** `(‘get’, (‘simulation’, (‘blender file name’)))`
- **Daemon:** `(‘get’, (‘simulation’, (‘blender file name’, ‘/home/blender/samples/mountain/mountain.blend’)))`

Get/Set Settings

The simulation can request/set many information from the daemon. For instance, to define the processor file, it must use ('set', ('simulation', ('blender processor name', '/home/blender/samples/spider/spider.blend'))) (and the daemon will answer : ('set', ('simulation', ('blender processor name')))).

Simulation

- blender file name [string]: name of the .blend file
- processor file name [string]: name of the processor file.
- link processor to blender [boolean]: do we have to link the processor file name to the blender file name ?

Configuration File

The BlenderVR XML configuration file is loaded by the console to get the architecture related information to run BlenderVR and send it to each virtual environment rendering node.

This file must contain at least four sections, plus the `plugins` section. It also includes a `blendervr` section which only option is the network port used for the synchronization between the rendering nodes.

Note: Use of space in `screen` name should work. Beware still Windows users.

Document Sections

- Redundant Sections
- Code Execution
- Starter Section
- Anchor
- Users Section
- Computers Section
- Screens Section
- Plugin Section
- Sample Configuration File

Redundant Sections

Some elements can be specific to one node, other shared. For instance, the `blenderplayer` executable can be the same for all rendering nodes or different on some nodes. In such case, there will be a section called `system` that can be inherited by each `computer` sub-section:

```
<computers>
  <system>
    <blenderplayer executable='/usr/local/blender/2.74/bin/blenderplayer' />
  </system>
  <computer name='front computer' hostname='front.fqdn'>
```

```

    <system>
      <blenderplayer executable='/usr/bin/blenderplayer' />
    </system>
  </computer>
  <computer name='left computer' hostname='left.fqdn' />
  <computer name='right computer' hostname='right.fqdn'>
    <system>
      <library path='/usr/local/lib/vrpn/' />
    </system>
  </computer>
</computers>

```

In this example, left computer and right computer nodes will use /usr/local/blender/2.74/bin/blenderplayer whereas front computer node will use /usr/bin/blenderplayer.

The system section is called *redundant* as many entries will use the same information.

Code Execution

In the XML file, you can use back-quote to execute code. First, the XML parser will try to execute this code as python code in BlenderVR environment system (with all variables and import present in the BlenderVR XML parser). If it fails, then, it tries as bash code and take the stdout result. If none is valid it raises an error.

For instance,

```
<environment>HOME=`os.environ['HOME']`</environment>
```

will define an environment variable (passed to the daemon or blenderplayer) called HOME that contains the current value of HOME operating system environment variable (with `os.environ` python code).

You can even use inherited values from redundant section:

```
<login remote_command="ssh `self._attributs_inheritance['hostname']`">
```

used inside the system redundant section will specify that `remote_command` will include the hostname as given in the computer entry.

If uncertain, we suggest you to simply print the `self._attributs_inheritance` python dictionary:

```
<login remote_command="`print(self._attributs_inheritance)`">
```

that will raise an exception (which is the point, since your purpose here is to create your configuration file, not to run BlenderVR).

Starter Section

This section only concerns the console. It contains all screen sets definitions.

```

<starter blender='/usr/bin/blender'>
  <config name='console'>console</config>
  <config name='virtual environment'>console, front screen, left screen, right screen</config>
  ...
</starter>

```

You can also add a `hostname` attribute in case of `socket.gethostname()` python function returns wrong hostname. This hostname is used by all *virtual environment* nodes to contact the console for network connection control.

The `blender` attribute is required in most of the cases for the `is_creating_loader` process.

Each `config` sub-section must list all screens, separated by commas, used by this screen set.

Note: De facto, the first screen listed here is the master node.

Anchor

On some devices, the paths are not homogeneous: the root path (repository) of `.blend` files on the console is not the same than on the master and/or on the slaves.

To fix that, BlenderVR uses the notion of **Anchor**: it is a node specific absolute path on all nodes that prefixes each relative path for blender and processor files.

It is a kind of least common multiple path. For instance, with two computers:

- **console** blender files repository: `/home/me/blender_files`
- **master node** blender files repository: `/remote_home/me/blender_files`

This least common path is `/home` on the console and `/remote_home` on the master node (`me/blender_files` are common on both systems).

In such case, the starter section (console specific section) will start by:

```
<starter anchor='/home'>
```

Whereas system section for the master node will start by:

```
<system anchor='/remote_home'>
```

Users Section

Each user must be listed here. Several users will e.g. enable you to attach a head tracker to adapt stereoscopic rendering to different points of view inside the virtual environment.

The behavior `redundant` section can define the `default_position` (0.0, 0.0, 0.0 by default) or the `eye_separation` (6 centimeters by default) of the user.

```
<!-- users section with default values -->
<users>
  <behavior eye_separation='0.06'>
    <default_position>0.0, 0.0, 0.0</default_position>
  </behavior>
  <user name="user A" />
</users>
```

Computers Section

We must describe how each rendering node (computer) works: each computer can have a specific configuration to run blender-player (paths, environment variables ...). However, most of the time, all computers are equivalent. Redundant section is useful!

Computer itself must have a name and a hostname. The name will be used by the screen.

```
<computers>
  <system>
    . . . <!-- computers global information -->
  </system>
  <computer name='front computer' hostname='front.fqdn'>
```

```

<system>
  . . . <!-- front computer specific information -->
</system>
</computer>
<computer name='left computer' hostname='left.fqdn' />
</computers>

```

System Section

The `system` redundant section defines many things:

```

<system root='C:\\program\\BlenderVR' anchor='U:\\blender_files'>
  <login remote_command="ssh `self._attributs_inheritance['hostname']`"/>
  <daemon>
    <environment>SystemRoot=C:\\Windows</environment>
  </daemon>
  <blenderplayer executable='C:\\blenderCave\\blender\\v2.70a\\blenderplayer.exe'>
    <environment>PYTHONPATH=C:\\Python33\\Lib;</environment>
  </blenderplayer>
</system>

```

The `root` parameter specifies the root path of BlenderVR (where resides the BlenderVR python script, the `modules` folder, etc.). By default, it is set to BlenderVR root path on the console computer. However, due to [not homogeneous paths between nodes](#), you may have to define it for each system.

See [Anchor](#) to know the purpose of anchor parameter.

Library Path Sub-Section

Plugins often relies on external libraries. If the library is not bundled in the `blenderplayer` python folder, the library folder can be specified with the `library` element. If any library is defined in a system section, they all must be defined.

In the example below, the VRPN library folder is specified for the OSX system, while the Linux stations share the same system as defined in the top of the `computers` section.

```

<computers>
  <system>
    <library path="/usr/local/lib/vrpn/" />
  </system>
  <computer name='OSX station' hostname='mac'>
    <system>
      <!-- define path towards VRPN python shared object-->
      <library path="/User/dev/vrpn/build/python/" />
    </system>
  </computer>
  <computer name='Linux station A' hostname='linux_a' />
  <computer name='Linux station B' hostname='linux_b' />
</computers>

```

Login Sub-Section

This section explains how to connect console and hosts computers.

```

<login remote_command="ssh me@host" python="/usr/bin/python3"/>

```

or

```
<login remote_command="psexec -d \\host" python="C:\\python33\\python.exe"/>
```

- **remote_command** specifies the command, from the computer running the console to connect to the remote host.
- **python** contains the path and the name of the python3 executable.

Generally, we use redundant system section with code execution to create this section (see example of the redundant section upper).

Daemon Sub-Section

The daemon sub-section explains how to run the daemon (now that we know how to connect to the remote computer).

```
<daemon transmit='True'>
  <environment>SystemRoot=C:\\Windows</environment>
</daemon>
```

- **transmit** parameter specifies if the daemon must transmit the environment variables to blenderplayer while it runs it.
- **environment** sub-section adds some specific environment variable to the daemon.

Note: On Windows, you must at least, set the `SystemRoot` variable to points towards the path of your Windows installation (generally: `C:\\Windows`)

Blenderplayer Sub-Section

This section defines how to run blenderplayer.

```
<blenderplayer executable='C:\\BlenderVR\\blender\\v2.74\\blenderplayer.exe'>
  <environment>PYTHONPATH=C:\\Python33\\Lib;C:\\Python33\\DLLs;C:\\Python33\\Lib\\site-packages</environment>
</blenderplayer>
```

- The **executable** parameter contains the path and the binary name of patched version of blenderplayer.
- The **environment** sub-sections allows you to add specific environment variables for blenderplayer. You can add `PYTHONPATH` environment to specify paths for optional modules (such as for VRPN).

Screens Section

The screen is the unit of rendering: there is a bijection between screen and instance of blenderplayer. Each screen has a name and a computer (actually the name of the computer section, above).

```
<screens>
  <display>
    . . . <!-- screens global information -->
  </display>
  <screen name='front screen' computer='front computer'>
    <display>
      . . . <!-- front screen specific information -->
    </display>
    <wall>
      . . .
    </wall>
  </screen>
```

```
<screen name='left screen' computer='left computer'>
</screens>
```

The display `redundant` section defines several things:

- **options** passed as argument to `blenderplayer` (for instance, `-f -s hwpageflip` to request a stereoscopic full screen `blenderplayer` window).
- **environment** to pass specific environment variables to `blenderplayer`.
- **graphic_buffer** to associate:
 - `buffer` (`mono = no stereo`, `left graphic buffer` or `right graphic buffer`,
 - `user` (as given inside `users` section),
 - `eye of the user` (`left`, `middle` or `right`).
- **viewport** to reduce the screen (useful if you have occlusion).

```
<display options='-w 400 400'>
<viewport>420, 0, 1500, 1080</viewport>
<environment>DISPLAY=:0.0</environment>
<graphic_buffer buffer='mono' user='user A' eye='middle' />
</display>
```

Each screen must have one sub-section `wall` or `hmd`.

Wall or HMD differs in the way they manage the projection. Wall screens are fixed in the real world but HMD screen are attached to head of the user, moving along.

Both require a screen definition: three corners (top right, top left and bottom right):

```
<wall> <!-- or <hmd> -->
<corner name="topRightCorner">1.0, 1.0, -1.0</corner>
<corner name="topLeftCorner">-1.0, 1.0, -1.0</corner>
<corner name="bottomRightCorner">1.0, -1.0, -1.0</corner>
</wall> <!-- or /<hmd> -->
```

For Wall, the screens are defined in vehicle reference frame. For HMD, the screens are defined in the reference frame of head tracker.

Plugin Section

So far, the plugin section can hold up to three sub-sections:

- `<vrpn>`: this section defines how to handle VRPN data sent by your VRPN server.
- `<osc>`: this section defines how to handle OSC data sent via OSC (e.g. with the [Sound Rendering Engine add-on](#) for BlenderVR) to an OSC client.
- `<oculus_dk2>`: this section defines how to handle data sent by the oculus rift dk2.

The following lines of code defines/assumes:

- `<vrpn>`: A VRPN server at `192.168.48.201` with a device named `DTrack`, saying that the callback `user_position_dtrack` (that has to be defined in the processor file) should be used to handle VRPN data coming from sensor 22 of `DTrack` device for user A. A VRPN server at `localhost` with a device named `device0`, saying that the callbacks `space_navigator_analog` and `space_navigator_button` should be used to handle VRPN data coming from `device0` device.

- <osc>: An OSC client at localhost listening to port 3819. The remaining flag will send init messages to the OSC client as defined in the [BlenderVR OSC API](#).
- <oculus_dk2>: An oculus rift dk2 plugged on computer localhost (named myComputer in computers section) saying that the callback user_position should be used to handle DK2 data (e.g. head orientation) for user A.

```
<plugins>
```

```
<vrpn>
```

```
<floor x='0.0' />
```

```
<!--! define DTrack parameters from VRPN server @ distant host -->
```

```
<tracker device="DTrack" host="192.168.48.201">
```

```
<transformation>
```

```
<post_translation z="0.0" />
```

```
<post_rotation x="1.0" y="1.0" z="1.0" angle="-2*math.pi/3" />
```

```
<pre_rotation x="1.0" y="1.0" z="1.0" angle="2*math.pi/3" />
```

```
</transformation>
```

```
<sensor id="22" processor_method='user_position_dtrack' users='user A' />
```

```
</tracker>
```

```
<!--! define device0 parameters from VRPN server @ local host -->
```

```
<analog device="device0" host="localhost" processor_method="space_navigator_analog" />
```

```
<button device="device0" host="localhost" processor_method="space_navigator_button" />
```

```
</vrpn>
```

```
<!--! define OSC parameters for OSC client @ localhost -->
```

```
<osc host='localhost' port='3819' configuration='Laptop SPAT'>
```

```
<user listener='Binaural 1' viewer='user A' />
```

```
<user listener='Binaural 2' viewer='user B' />
```

```
<user listener='Ambisonic' />
```

```
<user listener='Stereo' />
```

```
</osc>
```

```
<!--! define oculus dk2 parameters -->
```

```
<oculus_dk2>
```

```
<user host="localhost" processor_method="user_position" viewer='user A' computer='myComputer' />
```

```
</oculus_dk2>
```

```
</plugins>
```

Sample Configuration File

This sample configuration file can be used for a CAVE with three vertical square (2m x 2m) screens (left, front and right) plus a console computer with a single windowed screen.

```
<?xml version="1.0"?>
```

```
<blendervr>
```

```
<starter anchor='/tmp/console' blender='/usr/local/blender/2.74/bin/blender'>
```

```
<config name='console'>console screen</config>
```

```
<config name='virtual environment'>console screen, front screen, left screen, right screen</config>
```

```
</starter>
```

```
<users>
```

```
<user name='user A' />
```



```

</users>

<!-- Here, we define the console parameters -->
<computers>
  <computer name='console computer' hostname='console.fqdn' />
</computers>
<screens>
  <screen name='console screen' computer='console computer'>
    <display options='-w 600 600'>
<environment>DISPLAY=:0.0</environment>
<graphic_buffer user='user A' />
    </display>
    <wall>
<corner name='topRightCorner'>1.0, 1.0, -1.0</corner>
<corner name='topLeftCorner'>-1.0, 1.0, -1.0</corner>
<corner name='bottomRightCorner'>1.0, -1.0, -1.0</corner>
    </wall>
  </screen>
</screens>

<computers>
  <system root='/usr/local/blender/vr/1.0' anchor='/tmp/node'>
    <login remote_command="ssh 'self.__attributs_inheritance['hostname']" python='/usr/local/blender/2
    <daemon transmit='True'>
<environment>PATH=/usr/bin:/bin</environment>
    </daemon>
    <blenderplayer executable='/usr/local/blender/2.74/bin/blenderplayer' />
  </system>
  <computer name='front computer' hostname='front.fqdn' />
  <computer name='right computer' hostname='right.fqdn' />
  <computer name='left computer' hostname='left.fqdn' />
</computers>
<screens>
  <display options='-f -s hwpageflip'>
  <environment>DISPLAY=:0.0</environment>
  <graphic_buffer buffer='left' user='user A' eye='left' />
  <graphic_buffer buffer='right' user='user A' eye='right' />
</display>
  <screen name='front screen' computer='front computer'>
    <wall>
<corner name='topRightCorner'>1.0, 1.0, -1.0</corner>
<corner name='topLeftCorner'>-1.0, 1.0, -1.0</corner>
<corner name='bottomRightCorner'>1.0, -1.0, -1.0</corner>
    </wall>
  </screen>
  <screen name='left screen' computer='left computer'>
    <wall>
<corner name='topRightCorner'>-1.0, 1.0, -1.0</corner>
<corner name='topLeftCorner'>-1.0, 1.0, 1.0</corner>
<corner name='bottomRightCorner'>-1.0, -1.0, -1.0</corner>
    </wall>
  </screen>
  <screen name='right screen' computer='right computer'>
    <wall>
<corner name='topRightCorner'>1.0, 1.0, 1.0</corner>
<corner name='topLeftCorner'>1.0, 1.0, -1.0</corner>
<corner name='bottomRightCorner'>1.0, -1.0, 1.0</corner>
    </wall>

```

```
</screen>
</screens>

<plugins>
  <vrpn>
    <floor x='0.0' />
    <tracker device='GTK' host='localhost'>
<transformation>
  <post_translation z='-1.6' />
  <post_rotation x='1.0' y='1.0' z='1.0' angle="`-2*math.pi/3`"/>
  <pre_rotation x='1.0' y='1.0' z='1.0' angle="`2*math.pi/3`"/>
</transformation>
<sensor id='0' processor_method='user_position' users='user A' />
<sensor id='1' processor_method='tracker_1' />
<sensor id='2' processor_method='tracker_2' />
<sensor id='3' processor_method='tracker_3' />
  </tracker>
  <analog device='GTK' host='localhost' processor_method='movements' />
  <button device='GTK' host='localhost' processor_method='buttons' />
</vrpn>
</plugins>
</blendervr>
```

The BlenderVR project is open-source, opened to external collaboration.

Building Documentation

There are two parts of the project that are covered by this documents: Source Code API ¹ and the User Manual ². Even though those parts are independently hosted and maintained, they are both built on the same framework.

Document Sections:

- Language and Format
- Requirements
- Installation
- How to Build
- How to Edit

Language and Format

The documentation is written in `reSt` (reStructuredText). This is a markup language that is compiled to generate html (or pdf).

Requirements

Before working in the documentation you need to install all the requirements and the main repository from the installation guide.

Quick Setup

Type the following commands in your terminal. If you are developing in Windows we recommend you to use Power Shell or similar.

¹ code, compiled

² code, compiled

```
$ cd $INSTALL_DIR
$ git clone https://github.com/BlenderVR/manual.git
$ source venv/bin/activate
$ pip requirements -r source/docs/requirements.txt
$ pip requirements -r manual/requirements.txt
$ deactivate
```

How to Build

Note: While this generates a local copy of the latest documentation, the BlenderVR project is hooked up with the [ReadTheDocs](#) system. This auto-compile the documentation and make it available online everytime something is committed in the system.

User Manual

```
$ cd $INSTALL_DIR
$ source venv/bin/activate
$ cd manual
$ make
$ deactivate
```

This will output the documentation to `$INSTALL_DIR/manual/html`.

Source Code API

```
$ cd $INSTALL_DIR
$ source venv/bin/activate
$ cd source/docs/
$ make
$ deactivate
```

This will output the documentation to `$INSTALL_DIR/source/docs/html`.

How to Edit

The `.rst` files are simple plain text files that can be edited with any text editing tool. Once the file is ready it can be previewed with `make`, and eventually pushed back to the repository.

There are tools to preview the `.rst` file during the editing, but they are platform specific. In the Linux and OSX environment one can use [InstantRST](#) with `vim`. Sublime (for OSX) seems to have some tools as well.

- [ReST Quick Reference](#)
- [ReST API](#)

Build Windows Executable

This section is intended for developers willing to create a `BlenderVR.exe` (Windows executable) based on BlenderVR source code.

Note: For casual users, please download the current BlenderVR Executable, see the [Installation Guide](#).

This installer was build using [Inno Setup \(v5.5.5\)](#). Please download the [BlenderVR Installer source code](#) before reading the next section.

Document Sections:

- [Installer Folder Structure](#)
- [Build BlenderVR Windows Installer](#)

Installer Folder Structure

- Installer internal directories

//deps/ BlenderVR dependencies
//logo/ BlenderVR logos
//macros/ BlenderVR macros (post install & run)
//output/ Output directory after installer compilation
//doc/ Installer documentation

- BlenderVR base folders

(copy of the Folder Structure suggested during BlenderVR Installation)

//source/ BlenderVR Source Code
//blender/ Blender Binaries
//samples/ BlenderVR Samples
//configuration files/ BlenderVR configuration files

Build BlenderVR Windows Installer

- Build

Create .exe based on InnoSetup files:

//blendervr_online_install.iss/ install using the internet
//blendervr_offline_install.iss/ install without using the internet (packed Qt, git repository, etc.)

Warning: Do not forget to define the working directory with `#define MyBaseDir “\...sources-of-the-installer-Installer_BlenderVR”`.

- Updates

Replace BlenderVR base folders by current [BlenderVR Git](#) versions.

Note: To add new folders and files you need to perform specific modifications in the .iss files.

Project links (Repos, API, etc.)

- [BlenderVR Home](#)
- [BlenderVR API](#)

- All Repositories
- Source Code Repository
- Manual Repository
- Samples Repository
- Bug List
- Bug Management

Frequently Asked Questions

- FAQ - BlenderVR Install
 - Error: Building PySide
 - Error: Python executable not finding libpython shared library
- FAQ - BlenderVR Run
 - Error: Invalid Blender path
 - Error: Python permission denied
 - Error: Using VRPN library crashes Blenderpayer
 - Error: Cannot start daemon for screen <screenName>

FAQ - BlenderVR Install

Error: Building PySide

PySide compilation fails and returns:

```
[ 83%] Building CXX object libshiboken/CMakeFiles/libshiboken.dir/shibokenbuffer.cpp.o, Linking CXX shared library libshiboken.cpython-34m.so
```

Python was built without using `./configure` option `--enable-shared` and the default is disabled. The PySide build needs a python shared object (`python.X.x.so`) shared library and not a static (`.a`) library.

Error: Python executable not finding libpython shared library

Python run fails and returns:

```
error while loading shared libraries: libpythonX.x.so: cannot open shared object file: No such file or directory
```

See this [thread](#).

FAQ - BlenderVR Run

Error: Invalid Blender path

The main BlenderVR console displays:

Error: invalid blender path (<path-to-blender-binary>) (line 1 of <path-to-.xml-configuration-file>)

Check execution permissions on blender and blenderplayer executables (chmod).

Error: Python permission denied

The console displays:

*PermissionError: [Errno 13] Permission denied:
'path-to-blender-folder/python/lib/python3.4/encodings/__init__.py'*

Check execution permissions on python files (path-to-blender-folder/python and path-to-blender-folder/scripts (chmod).

Error: Using VRPN library crashes Blenderpayer

No log, the Blenderplayer simply crashes every time BlenderVR starts

Try to compile Blender setting the `WITH_PYTHON_FRAMEWORK` option to ON.

Error: Cannot start daemon for screen <screenName>

One of BlenderVR screen consoles displays:

Cannot start daemon for screen <screenName>

To run BlenderVR you need a password-less ssh access to all the computers defined in your configuration file, even if the only computer defined there is `*` or `localhost`. See e.g. these threads for [Unix](#) or [Windows](#) architectures.