Blender User Manual

Release 0.1

Blender Community

March 17, 2015

CONTENTS

CHAPTER

ONE

INSTALLATION

1.1 Install BlenderVR

In order to install Blender-VR you need this guide.

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

1.1.1 Document Sections

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

1.1.2 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).

`//blender-vr/` *Blender-VR Source Code*

`//blender/` *Blender Binaries*

`//samples/` *Blender-VR Samples*

`//venv/` *Python Virtual Environment*

1.1.3 Acquiring Blender

Blender-VR requires Blender 2.74 or newer. Optionally you can use a patched version of Blender 2.73a available here for all the supported platforms.

- Mac OSX 64 bit: Blender 2.73a Patched. [1]
- Windows 32 bit: Blender 2.73a Patched. [2]
- Windows 64 bit: Blender 2.73a Patched. [2]
- Linux 32 bit: Blender 2.73a Patched. [3]
- Linux 64 bit: Blender 2.73a Patched. [3]

1.1.4 Acquiring BlenderVR

- Blender-VR Sources (download and unzip in the top folder, rename it **blender-vr**)

1.1.5 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/simple samples/
```

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

1.1.6 Install dependecies

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".

1.1.7 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:

[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

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

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

On Windows:

```
$ cd $INSTALL_DIR
$ pip3 install virtualenv
$ virtualenv venv
$ .\venv\Scripts\activate
$ pip3 install -r blender-vr\requirements.txt
$ python3 .\venv\Scripts\pyside_postinstall.py -install
$ python3 .\blender-vr\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)

1.1.8 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
$ ./blender-vr/blenderVR
```

On Windows:

```
$ cd $INSTALL_DIR
$ .\venv\Scripts\activate
$ python3 .\blender-vr\blenderVR
```

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

Once you are done running Blender-VR you can end the virtual environment running the command:

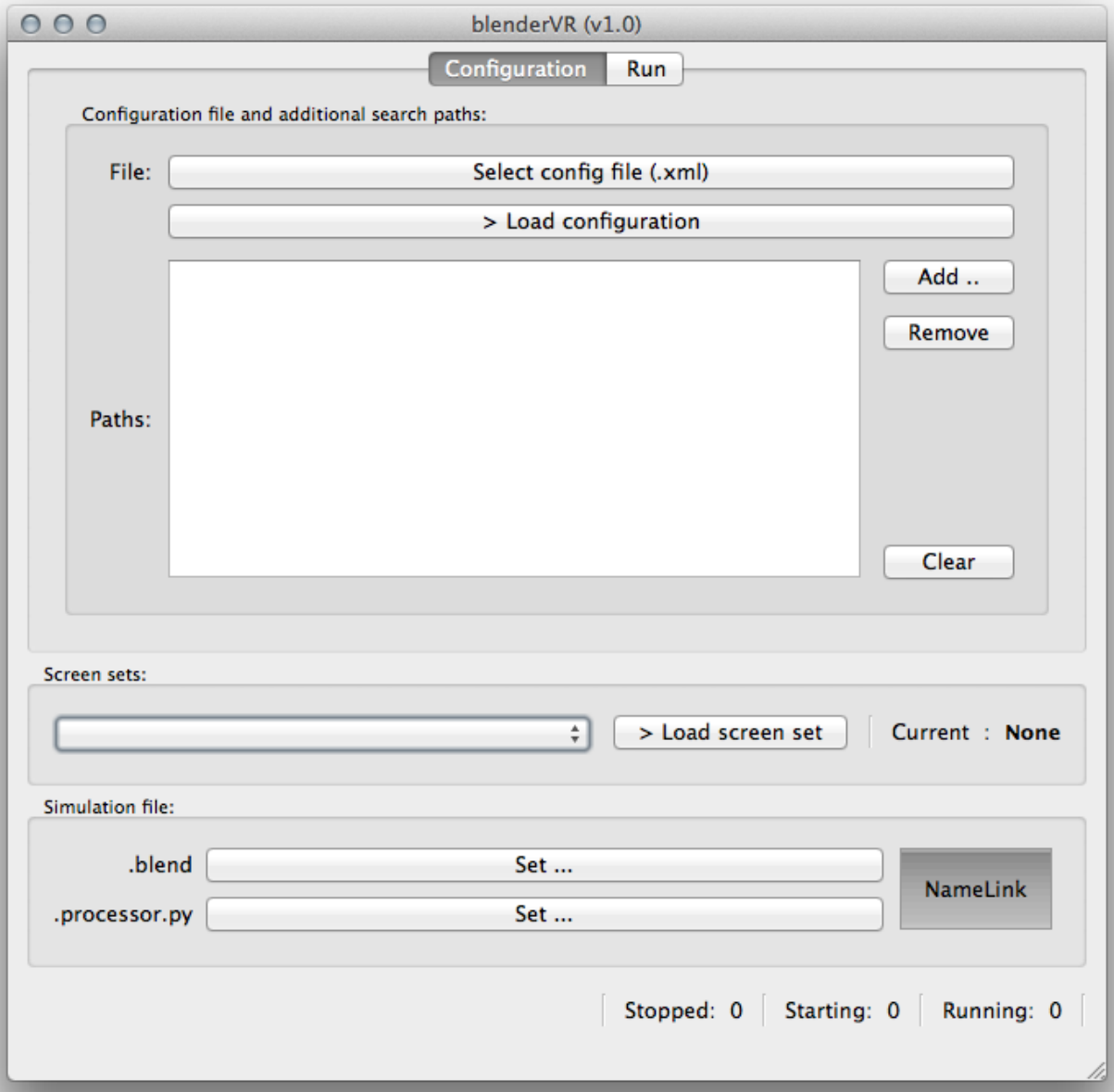
```
$ deactivate
```

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

1.2 Install for Future blenderVR Developers

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.


blenderVR (v1.0)
Configuration
Run
Configuration file and additional search paths:
File:
Select config file (.xml)
> Load configuration
Add ..
Remove
Paths:
Clear
Screen sets:
> Load screen set
Current : None
Simulation file:
.blend
Set ...
.processor.py
Set ...
NameLink
Stopped: 0
Starting: 0
Running: 0

The install is the same than 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.

1.2.1 Document Sections

- Acquiring Blender
- Acquiring BlenderVR
- Download Samples Scenes
- Requirements
- Quick Setup and Running

1.2.2 Acquiring Blender

Blender-VR requires a vanilla Blender 2.74 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.

1.2.3 Acquiring BlenderVR

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

```
$ git clone https://github.com/BlenderVR/blender-vr.git
```

If you do not intend to modify blenderVR source code, simply download the Blender-VR Sources zipfile.

1.2.4 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
```

Or for an individual sample folder:

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

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

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

1.2.5 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

1.2.6 Quick Setup and Running

see Quick Setup and Running in the Install blenderVR Section.

CHAPTER

TWO

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.

2.1 Document Sections

- Launch Blender-VR
- Open the Simulation File
- Edit the Configuration File
- Run

2.2 Launch Blender-VR

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.

2.3 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.

2.4 Edit the Configuration File

You now need a valid Configuration File to run Blender-VR. 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.

2.5 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 Blender-VR.

CHAPTER

THREE

HOW TO USE

3.1 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 Blender-VR. 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.

3.1.1 Document Sections

3.1.2 Console

The so called `console` is the GUI of Blender-VR. 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 informations in its internal data store path (see above). So you have to set these informations the first time you run Blender-VR and they remain active (across different running) until you change it with the GUI.

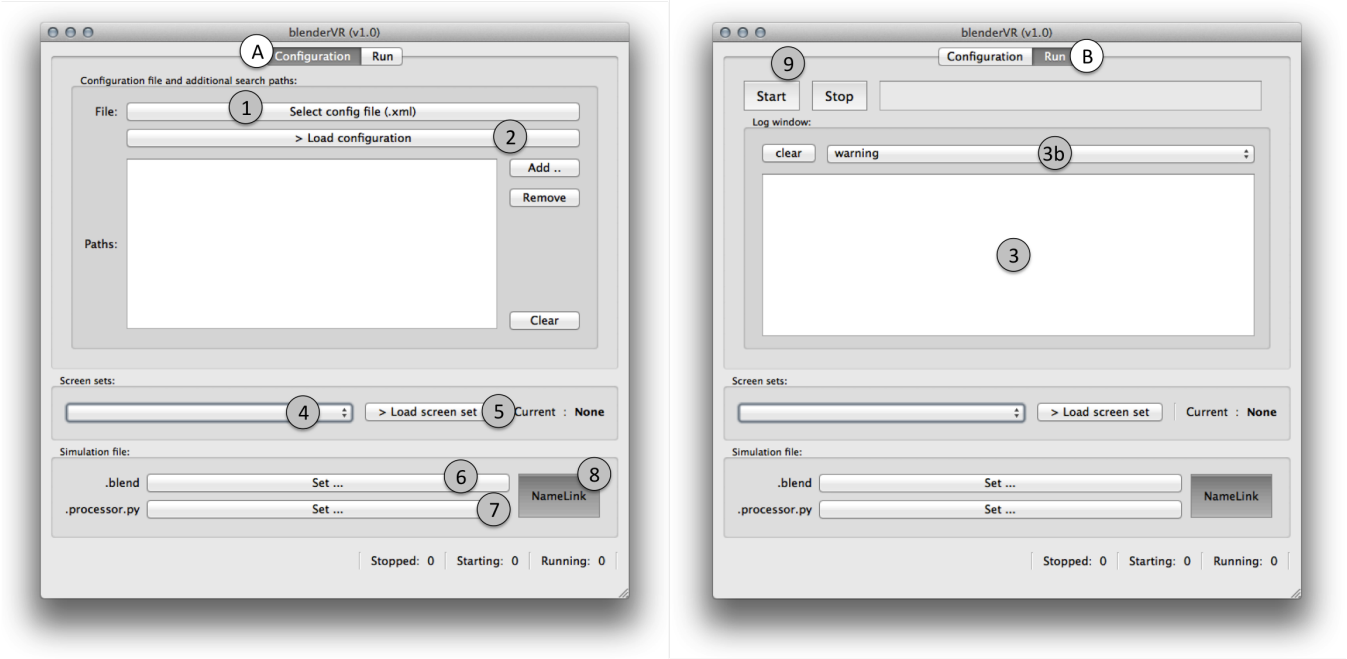


Figure 3.1: Console Graphical User Interface (left: `Configuration` tab, right: `Run` tab)

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 Blender-VR 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 Blender-VR 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 though 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`.

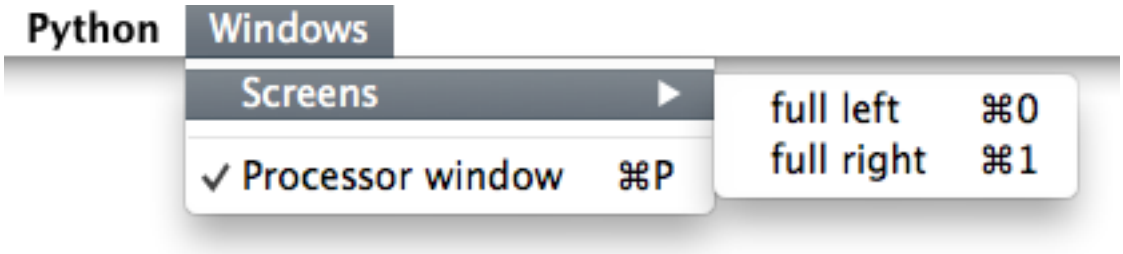


Figure 3.2: Top screen `Windows` menu.

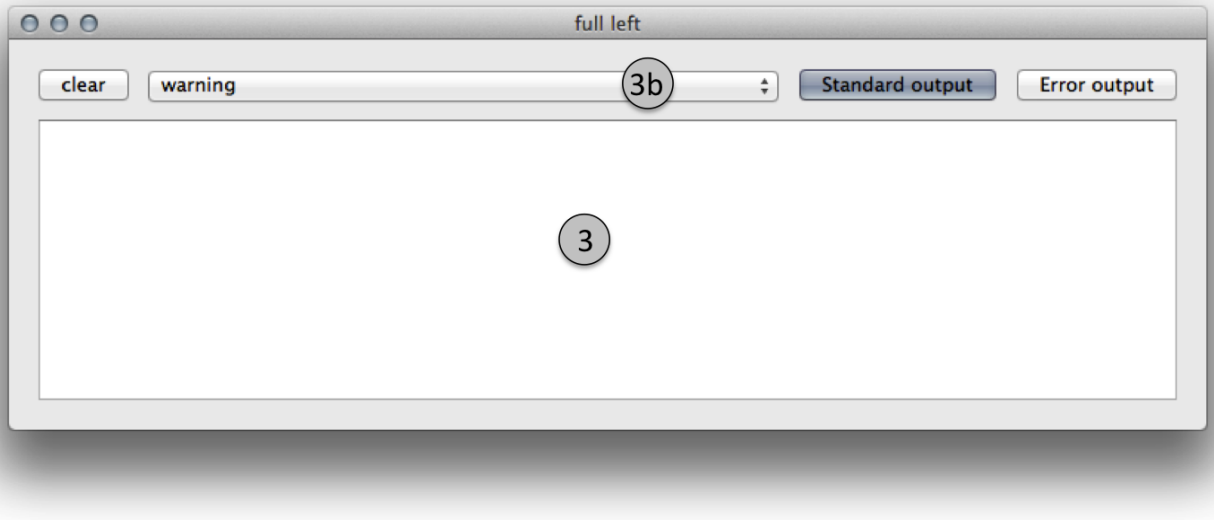


Figure 3.3: Screen window for screen named `full left` in the configuration file.

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 usefull when blenderplayer runs properly. It can display errors of your processor file in the log window (3).

3.1.3 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 Blender-VR - you don't have to tweak it.

3.2 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.

Blender-VR 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 funcionality.

3.2.1 Document Sections

3.2.2 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)
```

3.2.3 Basic Processor File

Unlike the Minimum Processor File, this one actually does something (in this case it syncronizes all the objects between the master and the slaves. This is file is fully explained in the Basic Example of the Blender-vR 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
```

3.2.4 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 Blender-VR runs.

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

You should instead use the Blender-VR 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 `login` 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 Blender-VR 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.

3.2.5 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.

3.2.6 Choose Objects to Synchronize

By default, Blender-VR 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`).

3.2.7 Processor Inheritance

> We commonly use the same interactions on different scenes. For instance, the Head Control Navigation system is useful on most scenes. Blender-VR 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 Blender-VR) 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.

3.2.8 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.

3.2.9 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`.

3.2.10 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.

3.2.11 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.

3.2.12 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.

3.3 Configuration File

The Blender-VR XML configuration file is loaded by the console to get the architecture related information to run Blender-VR 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.

3.3.1 Document Sections

3.3.2 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 *rendundant* as many entries will use the same information.

3.3.3 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).

3.3.4 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 Update 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.

3.3.5 Anchor

On some devices, the paths are not homogenous: 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'>
```

3.3.6 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>
```

3.3.7 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. owever, due to not homogenous 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 both OSC and VRPN library folders are specified for the OSX system, while the Linux stations shared the same system as defined in the top of the `computers` section.

```
<computers>
  <system>
    <library path="/usr/local/lib/vrpn/" />
    <library path="/usr/local/lib/osc/" />
  </system>
  <computer name='OSX station' hostname='mac'>
    <system>
      <library path="/User/dev/vrpn/build/python/" />
      <library path="/User/dev/osc/lib/" />
    </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</environ
</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).

3.3.8 Screens Section

The screen is the unit of rendering: there is 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 informations -->
  </display>
  <screen name='front screen' computer='front computer'>
    <display>
      . . . <!-- front screen specific informations -->
    </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 (usefull 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.

3.3.9 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>
```

3.4 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/.

Blender-VR 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 Blender-VR, 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).

3.4.1 Document Sections

- Interaction Setup
- Example with a Nintendo Wii Controller

3.4.2 Interaction Setup

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

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

3.4.3 Example with a Nintendo Wii Controller

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

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

2. In the Blender-VR `.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. Blender-VR_API.processor.py), add:

```
import blendervr

if blendervr.is_virtual_environment():
  import bge

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

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

    def wiiButton(self, info):
      print ("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).

3.5 Open Sound Control (OSC)

Note: Document need to be reviewed. Also the documentation need to be tested to see if it is still valid.

OSC is a protocol used to send / receive data through applications. See http://opensoundcontrol.org.

Blender-VR includes a MaxMSP (http://cycling74.com) Sound Rendering Engine available at Downloads. It is however possible (and advised) to make it work with any other OSC client and fathom it for other purposes.

3.5.1 Document Sections

- Interaction Setup
- Downloads

3.5.2 Interaction Setup

While the OSC API allows to easily send OSC (UDP) flags, the MaxMSP associated Sound Rendering Engine has been design to receive an process these flags. Once you've opened the `Blender-VR_Sound_Rendering_Engine_vX.maxpat` on the OSC server as defined in the `.xml` configuration file:

```
<processor>
  (...)
  <plugins>
    <osc host='serverName' port='3819'/>
      <user listener='Binaural 1' viewer='user A' />
      <user listener='Ambisonic' />
    </osc>
  </plugins>
</processor>
```

And modified it to fit to your needs (spatializer, speakers mapping, microphone inputs, etc.), the rest of the sound adding process takes place in Blender-VR.

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

See `samples/BlenderCave_OSC.blend` and `samples/BlenderCave_OSC_API.blend`. LIMSI members, see http://wikivenise.limsi.fr/index.php/Open_Sound_Control .

3.5.3 Downloads

- Blender-VR Sound Rendering Engine (.zip) version 4

This is an example of a flexible sound rendering engine developed under MaxMSP which is fully controlled from the OSC messages received from BlenderCave.

One may obviously use it with any other software (it's all about dynamic autonomous instantiations, should be modified to be used as a simple GUI Sound Rendering Engine).

CHAPTER

FOUR

ARCHITECTURE

4.1 Master and Slaves

Communications inside Blender-VR 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.

4.1.1 Document Sections

- Master
- Slaves

4.1.2 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,
- . . .

4.1.3 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 Blender-VR runs.

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

4.2 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.

4.3 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.

4.3.1 Document Sections

- Base Protocol
- Get/Set Settings
- Simulation

4.3.2 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 sended to the peer. On the other side, the method blendervr.tools.connector.Common.decomposeMessage(message) must be use to decompose a message and analyse 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')))

4.3.3 Get/Set Settings

The simulation can request/set many informations 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')))).

4.3.4 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 ?

CHAPTER

FIVE

DEVELOPMENT

The Blender-VR project is open-source and open for external collaboration.

5.1 Links

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

5.2 Development Documents

5.2.1 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).

[1] code, compiled
[2] code, compiled

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.

```
$ cd $INSTALL_DIR
$ git clone https://github.com/BlenderVR/manual.git
$ source venv/bin/activate
$ pip requirements -r blender-vr/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 Blender-VR project is hooked up with the ReadTheDocs system. This auto-compiles 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`.

Souce Code API

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

This will output the documentation to `$INSTALL_DIR/blender-vr/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