Read the Docs Template Documentation

Release 3.3.0

Read the Docs

Sep 25, 2023

Contents

Read The Funny Manuals!

Project Home Page

Latest changes: *v3.3.1*

Last edit: Sep 25, 2023

Contents:

CHAPTER 1

Downloads

Browse all files and versions:

https://sourceforge.net/projects/cgru/files

1.1 MS Windows

ZIP Archive:

https://sourceforge.net/projects/cgru/files/3.3.1/cgru.3.3.1.windows.zip

64Bit, Python 3.10 PySide2, Qt 5.15.2, MSVC 2019.

1.2 Linux

- **CentOS 7 (RHEL)** https://sourceforge.net/projects/cgru/files/3.3.1/cgru.3.3.1.CentOS-7_x86_64.tar.gz
- **Debian 10** https://sourceforge.net/projects/cgru/files/3.3.1/cgru.3.3.1.debian10_amd64.tar.gz
- **Debian 11** https://sourceforge.net/projects/cgru/files/3.3.1/cgru.3.3.1.debian11_amd64.tar.gz
- **Fedora 36** https://sourceforge.net/projects/cgru/files/3.3.1/cgru.3.3.1.Fedora-37_x86_64.tar.gz
- **Open SUSE 15.4** https://sourceforge.net/projects/cgru/files/3.3.1/cgru.3.3.1.openSUSE-15.4_x86_64.tar.gz
- **Rocky Linux 8.6 (RHEL)** https://sourceforge.net/projects/cgru/files/3.3.1/cgru.3.3.1.Rocky-8.7_x86_64.tar.gz
- **Rocky Linux 9.0 (RHEL)** https://sourceforge.net/projects/cgru/files/3.3.1/cgru.3.3.1.Rocky-9.1_x86_64.tar.gz
- **Ubuntu 20.04 LTS** https://sourceforge.net/projects/cgru/files/3.3.1/cgru.3.3.1.ubuntu20.04_amd64.tar.gz
- **Ubuntu 22.04 LTS** https://sourceforge.net/projects/cgru/files/3.3.1/cgru.3.3.1.ubuntu22.04_amd64.tar.gz

All binaries (packages) are 64 bit.

1.3 Mac OS X

There is no special release for this platform yet, but it is fully supported by sources. So you can build Afanasy manually.

1.4 Sources

Latest release snapshot:

https://sourceforge.net/projects/cgru/files/3.3.1/cgru.3.3.1.zip

Repository:

https://github.com/CGRU/cgru

CHAPTER 2

Installation

2.1 Linux Packages

Download the latest release and unpack.

- **Server**: You need to install **afanasy-server** package. Also you can to install **afanasy-render** package to monitor its resources, and should limit some heavy tasks not to run on server machine.
- **Workstation**: You need to install **cgru** package.
- **Render**: You need to install **afanasy-render** package.

CGRU will be installed in **/opt/cgru** folder.

Structure	Description	Depends
• cgru – afanasy-render ∗ afanasy-common · cgru-common – afanasy-qtgui ∗ cgru-common • afanasy-server – afanasy-common ∗ cgru-common	**cgru**: Start menu item.	PySide(1-2) or PyQt(4-5)
	cgru-common: All files, except Afanasy binaries.	
	afanasy-common: Afanasy binaries, except GUI.	PostgreSQL libraries (libpq)
	afanasy-render: Afanasy render startup scripts.	
	afanasy-server: Afanasy server startup scripts.	*PostgreSQL server, apache, php, php-pgsql*
	afanasy-qtgui: Afanasy Qt GUI binary.	Qt libraries

- *PostgreSQL server and Apache+PHP needed for afserver to store and view statistics only.*

Installation Methods:

- Install and uninstall scripts provided with the packages.
- Various GUI utilities native for each Linux distribution, can install it.

- Commands like dpkg -i for .deb's and rpm -i for .rmp's.
- The best way is to put this packages in you local company Linux repository. And to use native Linux ways to install and update software. In this way Linux system solves packages dependences itself.

2.2 MS Windows Archives

Download the latest release and unpack. Use Keeper to launch applications.

- **Server**: Launch afserver.
- **Render**: Launch afrender.
- **Workstation**: Use Keeper to launch render client and other CGRU applications.

2.3 Mac OS X

There is no release for this platform, yet. But you can build project yourself.

2.4 Manual Project Build

You can build project using cmake.

The project is hosted on GitHub: https://github.com/CGRU/cgru

If you want just to compile (not to develop) better to use tags.

You can simple *Download* an archive with the latests release sources.

Needed libraries:

- Python: >= 2.6
- Qt: Needed for Qt-GUI only
- PostgreSQL: Optional, needed by server for statistics only

2.4.1 Linux

All needed libraries can be installed by a script:

```
cd cgru/utilities
./install_depends_devel.sh
```

Run build script:

```
cd cgru/afanasy/src/project.cmake
./build.sh
```

2.4.2 MS Windows

You need MS Visual Studio 2015 and cmake. Cmake will generate Visual Studio project.

Go to `cgru/afanasy/src/project.cmake` and run `win_build_msvc.cmd`.

2.4.3 Mac OS X

You should be familiar with building projects on Mac.

You can use *macports* or *homebrew* to install needed libraries and cmake.

```
cd cgru/afanasy/src/project.cmake
./build.sh
```

CHAPTER 3

Configuration

Create config.json file in a CGRU root folder:

```
{"cgru_config":{
        "af_servername":"afanasy",
        "":""
}}
```

It should at least contain Afanasy server hostname or IP address. Read cgru/config_default.json parameters comments for help.

3.1 Config files

CGRU configuration files based JSON. One config file can include other, where parameters can be overridden. Config file can contain OS specific section, which parameters will be read only if client platform parameters matches OS section name.

At first `cgru/config_default.json` is read, where global parameters are set. It includes `cgru/afanasy/config_default.json` which configures Afanasy specific parameters. At last it tries to load `cgru/config.json`, where you should override some your company specific parameters. At least you should specify Afanasy server location in it.

CGRU Keeper and AfWatch stores settings in `$HOME/.cgru/config.json` on UNIX and `%APPDATA%/cgru/config.json` on MS Windows OS. It is read with the same rules as the main config. User can use it to set its own properties and override global config settings.

Each parameter should be preset only once in a config file. If there will several parameters with the same name in a same file, which will be chosen is undefined. Parameters order plays no role. Included files are read after the end of file, no matter where include line is. If in the next included file there will be a parameter with the same name, it value will be overridden.

See also:

JSON syntax reference: http://json.org

Better to check config with afcmd. Just run this command with no arguments. On error it will output error message, position and some text around it. You can also view configuration in Keeper, but it can not to start with bad config at all. Keeper Configuration window will display result configuration and each config file contents in the order they where read (included).

Note: There are no comments in JSON syntax. But some ways to comment JSON files exist. For example you can create an object with no name: `"":"Some comment text."` Or create an object with some unused name. To disable (comment) some parameter you can change it name to unused. For example you can just prefix name with "-".

3.2 Environment Variables

Common user does not need to setup environment variables manually. Use Keeper or start scrips they will setup environment. To setup a console you should go to CGRU root folder and source setup script.

```
cd cgru
source ./setup.sh
```

3.2.1 Manual Environment Setup

3.2.1.1 Required Variables

- **CGRU_LOCATION** - CGRU root folder.
- **AF_ROOT** - AFANASY root folder (`cgru/afanasy`).
- **PYTHONPATH** - To import CGRU and AFANASY Python modules:
  - `cgru/lib/python` - CGRU Python general library path.
  - `cgru/afanasy/python` - Afanasy Python library path.

3.2.1.2 Optional Variables

- **CGRU_VERSION** - CGRU version string, will be shown in AfWatch and Browser. May be any. You can add some useful info to real version at work.
- **AF_HOSTNAME** - Override hostname for Afanasy. Useful to run several clients on the same machine.
- **AF_USERNAME** - Override user name for Afanasy.
- **PATH** - Run commands.
  - `cgru/bin` - CGRU tools.
  - `cgru/afanasy/bin` - AFANASY applications.

3.3 Config Overrides

You can override any config parameter.

By Command Argument:

Use `--[param_name] param_value` arguments to an Afanasy executable. For example to make afwatch to connect to other afanasy server type:

```
afwatch --af_servername otherserver
```

By Environment Variable:

Set `CGRU_[PARAM_NAME]` environment variable. For example to setup console to use other server port type:

```
export CGRU_AF_SERVERPORT=51111
```

Python API will take environment overrides too.

3.4 Config Path Override

You can override the path to a custom config file using the `CGRU_CUSTOM_CONFIG` environment variable. This will be loaded last, but before `~/.cgru/config.json`

CHAPTER 4

Afanasy

4.1 Server

4.1.1 Launch Methods

- MS Windows script

  `start\AFANASY\_afserver.cmd`

- UNIX script

  `start/AFANASY/_afserver.sh`

- Linux daemon when Linux packages are installed

  `sudo systemctl start afserver`

- Setup CGRU environment and launch a command:

```
cd cgru
source ./setup.sh
afserver
```

4.1.2 System Job

System job is designed to execute system tasks on render farm (such as post commands). When server needs to execute some command it appends system job with a task.

Note: Your farm should be configured to execute have system services to execute job post commands (remove rendered scenes).

You can explore system job by Watch GUI in super user mode, to manipulate it's parameters to control its running.

Fig. 1: System Job

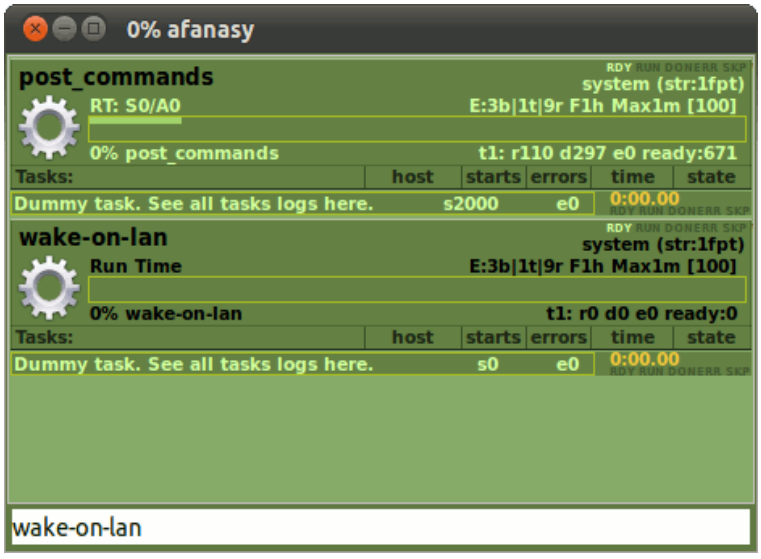


Fig. 2: System Job Tasks

If error system task can't be restarted (a number of error retries reached the maximum value) it will be deleted. It needed to prevent the growth of system tasks number.

You can watch system job log and its task log. When error occurs the log will be appended with the command output.

To reset system commands queue you can restart block or task.

4.1.2.1 Configuration

- `"af_sysjob_tasklife": 1800`

  Maximum system task age in seconds. If task age will equal to this number it will be treated as an error task. It needed to prevent the growth of tasks number, if some task(s) can't be executed (restarted).

- `"af_sysjob_tasksmax": 1000`

  Maximum number of running or ready tasks. If number of tasks will equal to this number, no new tasks will be created. But commands will not be lost, they will be stored in special list, to wait for some tasks will be done. It needed to prevent the growth of tasks number, if system job running will be stopped for some time (may be all hosts appeared in black lists). Tasks need more memory and CPU time then a simple commands list.

- `"af_sysjob_postcmd_service": "postcmd"`

  Service type for Post Commands system block.

- `"af_sysjob_events_service":"events"`

  Service type for Events system block.

- `"af_sysjob_wol_service": "wakeonlan"`

  Service type for Wake-On-LAN system block.

- `"af_render_cmd_wolsleep": "wolsleep"`

  Sleep command performed by a render client.

- `"af_render_cmd_wolwake": "wolwake"`

  Wake command constructed by a server and performed on a online client by the system job.

4.1.3 Post Commands

Post commands are executed on a job deletion. It is designed to clean up temporary files, that are not needed w/o the job. In a most common case, it is a temporary scene file to render.

Most submission scripts copy (save) current scene to some temporary file. This way artist can continue to make and save modifications in the current opened scene during render. Scene will be rendered at the state it was submitted.

Post commands are executed by renders via server system job *post_commands* block.

4.1.4 Wake-On-LAN

You can setup Afanasy to Wake-On-LAN machines.

Wake-On-LAN work-flow:

- Server sends a message to client to ask him to sleep.
- Client receives message from server to sleep.
- Client executes a `wolsleep` command which can be customized in Afanasy configuration.

- Client falls a sleep.
- Server does not receive updates from client and make it offline.
- Server "decides" to wake a render up.
- Server adds a task `wolwake mac1 .. macN` to system job *wake-on-lan* block. Command can be customized in Afanasy configuration.
- Another online and ready render executes the task.
- This task sends magic packet for each mac address of a sleeping render to a broadcast address. It is a small Python script provided with CGRU.
- Render wakes up.

You can wake and sleep renders by afwatch GUI and afcmd command.

4.1.5 Events

Events are generated by server. When event happened, job and user data is pushed to event service as a command by JSON. If event is emitted by render, render and all parent pools will be written too. Event service Python class reads its command - JSON data and can generate any command to execute. So event task receives data by a command, do something with this data and can construct a real command to execute as a task process.

4.1.5.1 JOB_DONE

Some job became done.

4.1.5.2 JOB_ERROR

Some job task produced an error.

4.1.5.3 JOB_DELETED

Job has been deleted.

4.1.5.4 RENDER_ZOMBIE

Render stopped to send updates to server for zombie_time seconds.

4.1.5.5 RENDER_SICK

Render produced sick_errors_count errors from different users in a row and got *SICK* state.

4.1.5.6 RENDER_NO_TASK

Render has no task for no_task_event_time seconds.

4.1.5.7 RENDER_OVERLOAD

Render has low free memory or disk or swap. How much resources considered as low, you can configure by JSON config parameters:

- `af_render_overflow_mem` - percentage of a free memory.
- `af_render_overflow_swap` - percentage of a free swap.
- `af_render_overflow_hdd` - percentage of a free disk space.

By default this parameters are equal to `-1` and this means that the resource check is disabled. Practically good free percentage to emit event is `1`, as an overloaded machine never reaches zero free memory or hdd.

The next time event will be emitted after overload_event_time seconds.

There is already default Python service class: `cgru/afanasy/python/services/events.py` It designed to send emails.

Example of a custom data to send emails:

```
{
    "emails":["some@email.com"],
    "events":
    {
        "JOB_ERROR":{"methods":["email"]},
        "JOB_DONE":{"methods":["email"]}
    }
}
```

User and job custom data objects are simple merged. So user can have information about email and job about events. If user will have email and events in *custom data* all it jobs will send emails.

You can write any custom Python service class, for example: `cgru/afanasy/python/services/events_local.py`

And set it as System job events block service name in your configuration file: `"af_sysjob_events_service":"events_local"`

4.1.6 Statistics

Afanasy server can store jobs and tasks statistics in SQL database. It uses PostgreSQL engine. On job deletion and task finish (with any result) server insert some job and task data into database tables.

4.1.6.1 Database Schema

```
afanasy=# \d jobs;
                          Table "public.jobs"
     Column     |          Type          | Collation | Nullable | Default
----------------+------------------------+-----------+----------+---------
 annotation     | character varying(512) |           |          |
 blockname      | character varying(512) |           |          |
 capacity       | integer                |           |          | 0
 description    | character varying(512) |           |          |
 folder         | character varying(512) |           |          |
 jobname        | character varying(512) |           |          |
 hostname       | character varying(512) |           |          |
 service        | character varying(512) |           |          |
```

(continues on next page)

(continued from previous page)

```
 tasks_done     | integer                |           |          | 0
 tasks_quantity | integer                |           |          | 0
 run_time_sum   | bigint                 |           |          | 0
 time_done      | bigint                 |           |          | 0
 time_started   | bigint                 |           |          | 0
 username       | character varying(512) |           |          |
 serial         | bigint                 |           |          | 0
 id_block       | integer                |           |          | 0

afanasy=# \d tasks;
                         Table "public.tasks"
    Column     |          Type           | Collation | Nullable | Default
---------------+-------------------------+-----------+----------+---------
 annotation    | character varying(512)  |           |          |
 blockname     | character varying(512)  |           |          |
 capacity      | integer                 |           |          | 0
 command       | character varying(4096) |           |          |
 description   | character varying(512)  |           |          |
 error         | integer                 |           |          | 0
 errors_count  | integer                 |           |          | 0
 folder        | character varying(512)  |           |          |
 frame_pertask | bigint                  |           |          | 0
 hostname      | character varying(512)  |           |          |
 jobname       | character varying(512)  |           |          |
 resources     | character varying(4096) |           |          |
 service       | character varying(512)  |           |          |
 starts_count  | integer                 |           |          | 0
 time_done     | bigint                  |           |          | 0
 time_started  | bigint                  |           |          | 0
 username      | character varying(512)  |           |          |
 serial        | bigint                  |           |          | 0
 id_block      | integer                 |           |          | 0
 id_task       | integer                 |           |          | 0
```

4.1.6.2 Database Setup

- Edit Postgre SQL client authentication configuration file pg_hba.conf.

  Its location depends on Linux distributive. For example:

  Debian, Ubuntu: `/etc/postgresql/ [version] /main/pg_hba.conf`

  CentOS, Fedora, openSUSE: `/var/lib/pgsql/data/pg_hba.conf`

  make install: `/usr/local/pgsql/data/pg_hba.conf`

  Add this line: `local afanasy afadmin password` Read comments in this file to know what does it mean. (If problems with authentication try trust for all methods.)

- Restart database

- Create afanasy database and user

```
sudo su - postgres
createdb afanasy
psql afanasy
CREATE USER afadmin PASSWORD 'AfPassword';
```

4.1.6.3 Create Tables

- Go into CGRU root folder: `cd /opt/cgru`
- Source setup: `source ./setup.sh`
- Check database connection: `afcmd db_check`
- Program should output an error or print "Database connection is working" if everything is ok.
- Create required tables: `afcmd db_reset_all`
- This command also delete old tables if they exists.

4.1.6.4 Server setup

You need to install a web server with PHP and PGSQL modules. Any Linux distribution have this packages.

In most Linux-es all this can be provided by packages: `apache2 libapache2-mod-php php php-pgsql`

The site is located in `cgru/afanasy/statistics` folder.

4.1.6.5 Web Page

There is a Web page to view Afanasy SQL statistics database.

4.1.7 TIME-WAIT

TIME-WAIT is a special socket state, needed to ensure that all packages will not be lost. If server calls close() function first, its socket will fall into this state. To ensure that the connection last package is processed, it will wait:

TIME-WAIT = 2 * MSL (Maximum Segment Lifetime)

This is the reason why server should not call close() first. On a big amount of clients (~1000), application can reach 2^16 ports limit. Afanasy waits for about 2sec for client to close socket first. To check socket connected state we just try to write in it. *SIGPIPE is ignored by Afanasy*

To check sockets state you can:

```
netstat -nat | grep 51000 | wc -l
netstat -nat | egrep ':51000.*:.*TIME_WAIT' | wc -l
ss -tan state time-wait | wc -l
ss -tan 'sport = :51000' | awk '{print $(NF)" "$(NF-1)}' | sed 's/:[^ ]*//g' | sort |␣
↪uniq -c
```

4.2 Render

Render is a client application. It runs on a remote host and communicates with server. Server sends tasks to render to run.

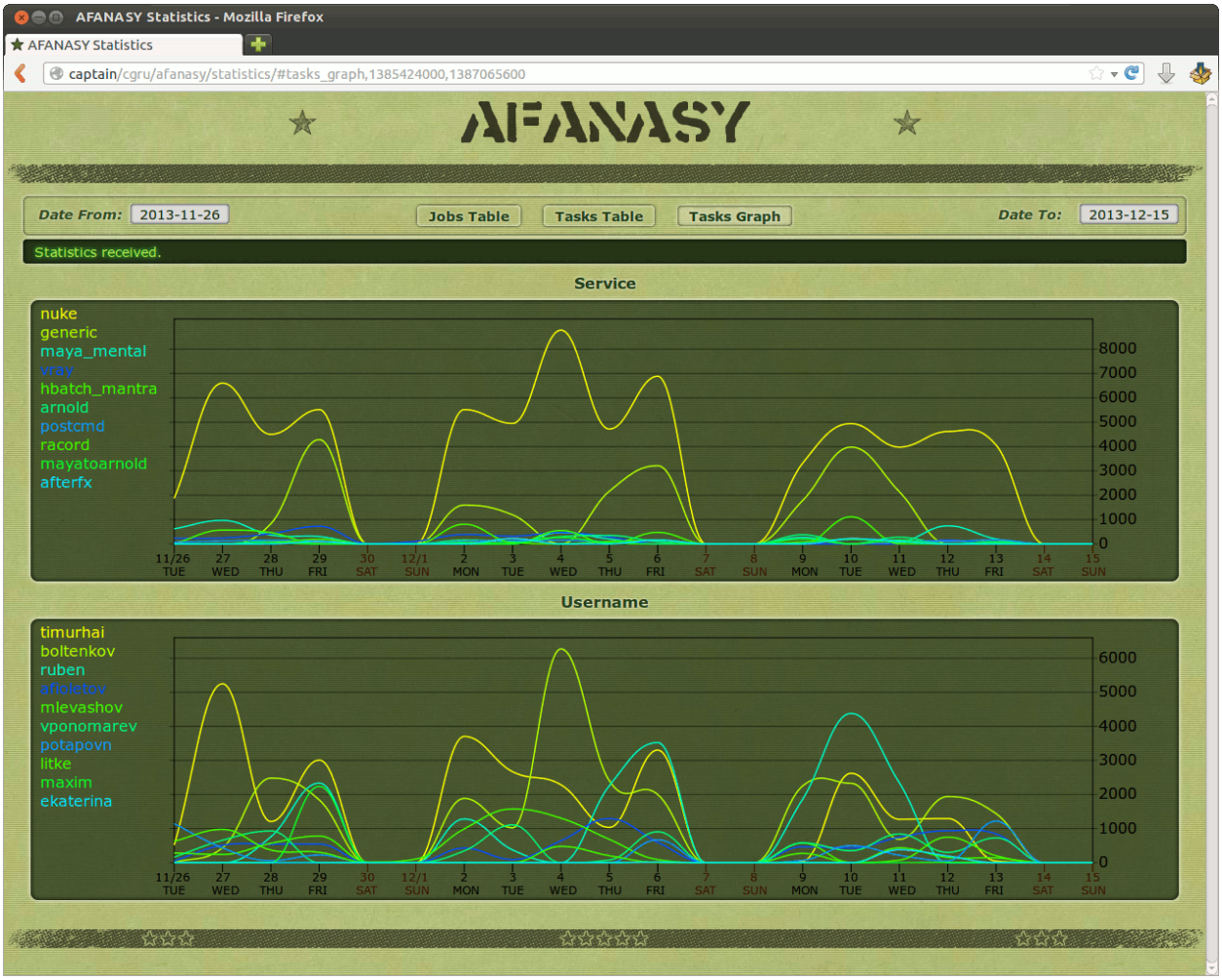


Fig. 3: Statistics Tasks Graph Page

4.2.1 Launch Methods

- MS Windows script

  `start\AFANASY\99.render.cmd`

- UNIX script

  `start/AFANASY/_afrender.sh`

- Linux daemon when packages are installed

  `sudo systemctl start afrender`

- Setup CGRU environment and launch a command:

```
cd cgru
source ./setup.sh
afrender
```

To register a render already in [nimby|NIMBY] state use command:

```
afrender [nimby|NIMBY]
```

4.2.2 Attributes

4.2.2.1 name

Each render has an unique name. If new render comes to server with the name which already exists, server will not register it. Users and jobs hosts masks are based on render names and Regular Expressions (they are Perl-like). To launch another render on the same host use AF_HOSTNAME environment variable to override render name.

4.2.2.2 address.family

4.2.2.3 address.ip

IP address with family (IPv4 or IPv6).

4.2.2.4 netifs[]

Network interfaces information (name, mac and ips).

4.2.2.5 time_launch

Time the application was launched.

4.2.2.6 time_register

Time the render was registered on server.

4.2.2.7 time_update

Last time the render has update its resources usage.

4.2.2.8 wol_operation_time

Time the last any Wake-On-LAN operation was performed.

4.2.2.9 tasks[]

Running tasks number and some its attributes (user, job, etc.).

4.2.2.10 capacity_used

Capacity used by running tasks.

4.2.3 Editable Parameters

4.2.3.1 user_name

Who launched a render. Can be changed only by administrators. Note that process do not change uid. This value for afanasy only.

4.2.3.2 priority

Render with greater priority get task first.

4.2.3.3 capacity

You can override host farm setup capacity.

4.2.3.4 max_tasks

You can override host farm setup maximum running tasks.

4.2.3.5 services

Services that render can run. If empty it it configured by pool.

4.2.3.6 services_disabled

You can disable some services.

4.2.3.7 annotation

Annotate render GUI item.

4.2.4 State

Online	ONL	Is online.
Offline	OFF	Is offline.
nimby	Nby	Is taken by his user. Only render user can render on it.
NIMBY	NBY	Is taken by his user and he don't want to render on it.
Busy	RUN	Executing one or more tasks.
Dirty	DRT	Capacity changed or some service disabled.
WOLFalling	WFL	Is falling a sleep. It was asked to sleep, but still online.
WOLSleeping	WSL	Is sleeping.
WOLWaking	WWK	Is waking up. It was asked to wake up but still is not online.
Paused	PAU	Is paused, like super-Nimby, never will be free automatically.
Sick	SIC	Is seek, produced errors only from different users.

4.2.5 Resources

4.2.5.1 cpu_num

CPUs x Cores number.

4.2.5.2 cpu_mhz

Fist Processor frequency.

4.2.5.3 cpu_loadavg[3]

Load average.

4.2.5.4 cpu_user

User usage percentage.

4.2.5.5 cpu_nice

User 'nice' usage percentage (low priority processes).

4.2.5.6 cpu_system

System usage percentage.

4.2.5.7 cpu_idle

Idle percentage (CPU free).

4.2.5.8 cpu_iowait

Waiting for I/O complete percentage.

4.2.5.9 cpu_irq

Interrupts servicing percentage.

4.2.5.10 cpu_softirq

Soft interrupts servicing percentage.

4.2.5.11 mem_total_mb

Total amount of memory in megabytes.

4.2.5.12 mem_free_mb

Free memory in megabytes.

4.2.5.13 mem_cached_mb

Cached memory in megabytes.

4.2.5.14 mem_buffers_mb

Buffered memory in megabytes.

4.2.5.15 swap_total_mb

Total swap space in megabytes.

4.2.5.16 swap_used_mb

Used swap in megabytes.

4.2.5.17 hdd_total_gb

Total disk space in gigabytes.

4.2.5.18 hdd_free_gb

Available free disk space in gigabytes (in root - '/').

4.2.5.19 hdd_rd_kbsec

Disk reading in kilobytes per second.

4.2.5.20 hdd_wr_kbsec

Disk writing in kilobytes per second.

4.2.5.21 hdd_busy

Percentage of system ticks spend for the disk IO.

4.2.5.22 net_recv_kbsec

Network receiving traffic in kilobytes per second.

4.2.5.23 net_send_kbsec

Network sending traffic in kilobytes per second.

4.2.5.24 gpu_string

The official product name of the GPU. It will be empty, if GPU is not detected. You can check it length for zero to find out GPU exists.

- NVIDIA cards are processed via nvidia-smi command line utility. So `nvidia-smi` must be in `PATH`.
  - **Linux**: Any NVIDIA driver installation from some repository or official site adds this binary to `PATH`, you should do nothing to get it work.
  - **Windows**: NVIDIA driver installation should add `nvidia-smi.exe` to `C:\Program Files\NVIDIA Corporation\NVSMI` folder. The folder can differ on some custom installation. You should add that folder to `PATH`. You can do this via system environment settings. Better not to modify the system environment permanently, but to do it by demand via scripting. You can create a `setup_gpu.cmd` script in CGRU root folder with a following content:

    ```
    set "PATH=C:\Program Files\NVIDIA Corporation\NVSMI;%PATH%"
    ```

  - **macOS**: Was not checked. Like on other platforms `nvidia-smi` should be in `PATH`.

For now only NVIDIA cards are supported.

4.2.5.25 gpu_gpu_util

GPU utilization of core GPU.

4.2.5.26 gpu_gpu_temp

GPU temperature of core GPU.

4.2.5.27 gpu_mem_total_mb

GPU total memory. You can check for zero, to find out that GPU was detected.

4.2.5.28 gpu_mem_used_mb

GPU used memory.

4.2.6 Paths Map

CGRU has an ability to map paths. Every client can have own paths map file to translate paths to server and from server.

Paths map is described in config files by pathsmap object. It is an arrays of `["CLIENT","SERVER"]` paths pairs:

```
{
    "pathsmap":[
        ["//server/projects/","/mnt/prj/"],
        ["//server/tools/","/mnt/tools/"]
    ],
}
```

When job constructs (on the client side) all commands and working directories are translated from client to server. When task starts (on the client side) all commands and working directories are translated from server to client. Server does know nothing about paths map.

4.2.6.1 MS Windows platform issues

- You can write only `/` slashes in a config. It will try both slashes directions. Some applications allows client to use and `\` and `/` slashes, so pattern will be matched in any case.
- When client searches a pattern it converts paths in lower case. So no matter how client wrote a path `//server/projects/`, `//SERVER/PROJECTS/`, `//SERVER/projects/` or `//server/PROJECTS/`. It will work any way.
- Module (Python Class) can works in *UnixSeparators* mode. During translation from server to client it uses `/` slashes for client paths. For example NUKE uses only `/` slashes on any platform.

A part of a real working `config.json` with map example:

```
{
    "OS_windows":{
        "pathsmap":[
            ["P:/",              "/ps/prj/"],
            ["//box/project/",   "/ps/prj/"],
            ["Q:/",              "/ps/prj2/"],
            ["//box2/project/",  "/ps/prj2/"],
            ["//sun/libs/",      "/ps/lib/"],
            ["//sun/vault/",     "/ps/vault/"],
            ["T:/",              "/ps/etc/"],
            ["c:/ps/",           "/ps/"],
            ["c:/temp/",         "/tmp/"]
        ]
    }
}
```

4.2.7 Services

Service is a Python class that will be instanced by render on each incoming task.

Python classes stored in

`cgru/afanasy/python/services`

and based from

`cgru/afanasy/python/services/service.py`

The class stands for:

- Define default service parser, that you can override.
- Instance needed parser and pass task output data it.
- Method to fill in numeric block pattern with frames.
- Method to transfer commands and paths from server to client (different OS-es can have different paths).
- Check rendered files.
- Generate thumbnails.
- Check exit status for tasks that can return non zero exit status on success.
- Method to insert in task command variable capacity coefficient.
- Method to fill in multi-host task command with captured hosts.

You can write custom service class based on `service.py` to override any functions for customization.

4.2.8 Parsers

Parser read task output and calculate running percentage and frame (for multiply frames per render).

Python classes stored in

`cgru/afanasy/python/parsers`

and based from

`cgru/afanasy/python/parsers/parser.py`

Parser class stands for:

- Parse task progress frame and percent of a current frame and a total(all frames) percentage.
- Parse output for rendered file to make thumbnails, that render will send to server.
- Stop task on bad output.
- Produce a warning just for user notification.
- Mark success finish as error on bad output.
- Append some string to task log for some useful info.
- Make some job report what will be shown in GUI job item as something important.
- Parse some resources, for example triangles count or pear memory usage.

To write a custom parser you should inherit base parser class and override main function:

4.2.8.1 do

```
def do(self, i_args):
```

Input arguments are passed via dictionary:

- **i_args['mode']** (str)
  - `RUN`: Task is running.

  - `EXIT_CODE:STOP_TIME`: Task is not running, process exit status and stop time if task was asked to stop (zero if was not).

- **i_args['pid']** (int)

  Task process identifier.

- **i_args['data']** (str)

  Current portion of a task process output.

- **i_args['resources']** (str)

  Host resources JSON. Designed to query/modify for **self.resources** construction.

This method can return nothing or a string. In string case this string will be stored instead of incoming data. You can use it to produce some message, by appending incoming data with your information. Or you can cut some useless information.

All parser notifications and actions are transferred by setting class members:

4.2.8.2 self.percent(int)

Task execution percentage.

4.2.8.3 self.frame(int)

Task execution frame. May set for multiply frame tasks to show current frame in GUI.

4.2.8.4 self.percentframe(int)

Task execution current frame percentage.

4.2.8.5 self.progress_changed(False/True)

Whether the task progress has changed. By default is `True` when any output with a not zero length was produced.

4.2.8.6 self.warning(False/True)

Some warning. To notify user only.

4.2.8.7 self.error(False/True)

Error. Render will try to terminate a task and later kill if task ignored termination.

4.2.8.8 self.badresult(False/True)

Error. Task will finish with an error. In this case render will not try to kill it, sometimes you don't need kill and want to wait finish. You can use at the end of task execution, at final result check.

4.2.8.9 self.finishedsuccess(False/True)

Success. Task will finish with a success. Parser can consider that task is already done and should not to continue. Render will terminate(kill) task process and send "done" status to server(not an "error").

4.2.8.10 self.activity(str)

Some string to inform user about task running stage. For example: Nuke current rendering view when stereo, Movie Maker convert or encode stage.

4.2.8.11 self.resources(str)

Any custom resources string. For example: `triangles:155000000`.

4.2.8.12 self.log(str)

Some string to append to the task server log. For example when server or parser noticed some error, you can specify it here.

4.2.8.13 self.report(str)

Some info string for an entire job. GUIs will show it a job item(not task). Some most important info should be here. Most suitable for a job with one or several big tasks. For example you can put big file on FTP and show speed here.

4.2.9 Thumbnails

Thumbnails are small previews of a task rendered files. They can be generated by render and shown by GUI.

If task (block) has files parameter or parser finds images thumbnail will be generated. Thumbnails are generated by afrender after task process finish. Python service doPost function returns commands for it. Thumbnail files binary data is send by afrender to afserver along with task output. Server stores all files that afrender sends on task finis. You can get tasks thumbnails from afserver by HTTP GET method.

If parser found some image during output parsing it can call a special function:

4.2.9.1 appendFile(i_file, i_onthefly)

- **i_file**

  Path to the image file to append.

- **i_onthefly**
  - **False** Thumbnail will be generated after task process finish. This is a most common method.
  - **True** Thumbnail will be generated just after this function call. Task probably will be still running in this case. This can be useful for a long time task that process many images. Good example is a movie encoding or dailies creation.

4.2.9.2 Configuration

```
{
    "af_thumbnail_extensions":["exr","dpx","jpg","jpeg","png","tif","tiff","tga"],
    "af_thumbnail_cmd":"convert -identify \"%(image)s\" -thumbnail \"100x100^\" -
↪gravity center -extent 100x100 \"%(thumbnail)s\"",
}
```

4.2.10 Custom Resources

You can write custom resources meter(s) on Python. Render instances a class and runs update method periodically (each time the Render updates). You can inherit base resbase class and set its properties.

There are some custom resources meters in Afanasy:

4.2.10.1 example

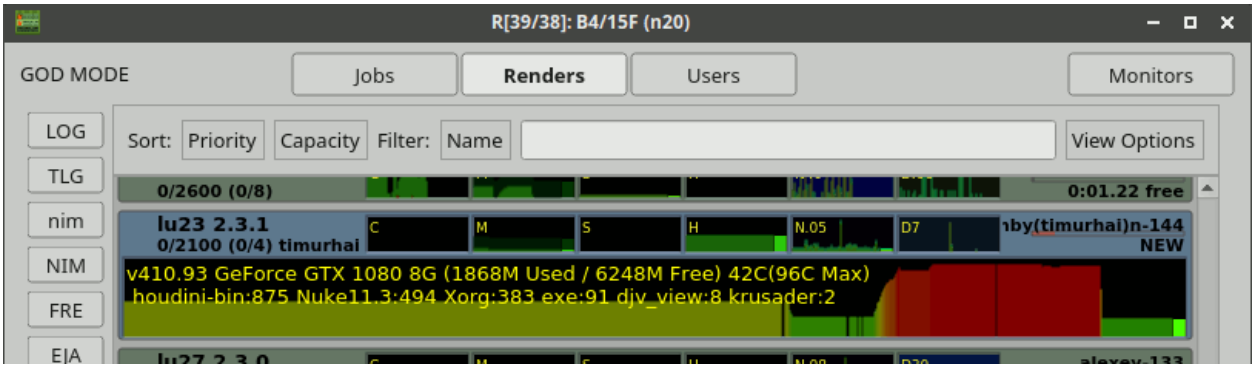
Just an example. It increments a value from 0 to 100, changes label text, plotter and label size, graph and back color.

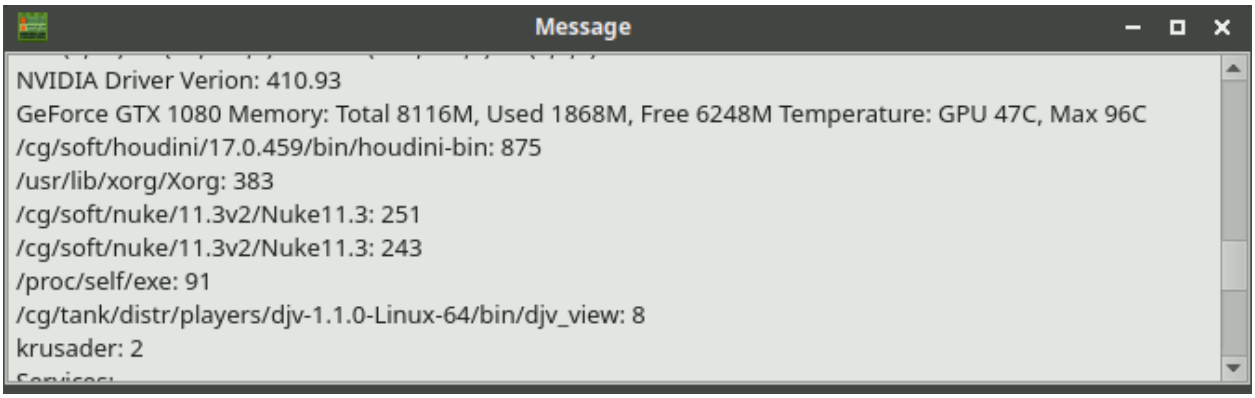
4.2.10.2 iostat

Shows parsed output of Linux iostat command. Graph value is utilization percentage (or %busy).

4.2.10.3 nvidia-smi

Shows parsed output of Linux nvidia-smi command. It shows NVIDIA driver version, product name, total and used memory, temperature, running processes.





Python classes stored in

`cgru/afanasy/python/resources`

and based from

`cgru/afanasy/python/resources/resbase.py`

Information is passed within class properties:

4.2.10.4 Properties

self.value(int)

The resource value to watch.

self.valuemax(int)

Maximum resource value for graph scale.

self.height(int)

Preferred plotter widget height for GUI.

self.width(int)

Preferred plotter widget width for GUI.

self.graphr(int)

self.graphrg(int)

self.graphb(int)

Graph color.

self.label(str)

Label text.

self.labelsize(int)

Label text font size.

self.labelr(int)

self.labelg(int)

self.labelb(int)

Label text font color.

self.bgcolorr(int)

self.bgcolorg(int)

self.bgcolorb(int)

Plotter background color.

self.tooltip(str)

Widget tooltip.

self.valid(False|True)

Resource meter validness. Should be set to True in constructor, or update function will not be called. Set False if resource meter initialization failed.

4.2.11 Windows Must Die

Farm based on MS Windows OS can produce some 'bad' windows. If process crashed, Windows OS can launch a window with apologizes, and 'hung' the process until someone closes this window.

Afanasy Render client periodically finds and closes windows listed in `af_render_windowsmustdie` configuration parameter.

It closes them by sending `WM_CLOSE` signal.

`af_render_windowsmustdie` parameter example:

```
{
    "af_render_windowsmustdie":[
        "ImageMagick Studio library and utility programs",
        "Microsoft Visual C++ Runtime Library",
        "QuickTimeHelper-32.exe - Application Error",
        "Visual Studio Just-In-Time Debugger"
    ]
}
```

4.3 Pools

Pool consists of renders and other child pools. It designed to manipulate properties (abilities) of group of render clients.

- Add/Remove/Enable/Disable services that pool renders are able to run.
- Set tickets to control how much pool renders can run together and each at the same time.
- Configure renders automatic *Nimby* and *Free*.
- Specify renders *capacity* and *max_tasks*.
- Make new renders to register in *Nimby* or *Paused* state to have some time for software installation.

Fig. 4: AfWatch pools services and tickets

4.3.1 Creation

You can add child pool to any parent pool using GUI. Root pool will be automatically created.

4.3.2 Attributes

4.3.2.1 name

Pool name represents its path.

4.3.2.2 parent

Pool parent name (path).

4.3.2.3 time_creation

Time, when pool was created.

4.3.2.4 pools_num

Number of direct child pools.

4.3.2.5 pools_total

Total number of child and pools with grand childs.

4.3.2.6 renders_num

Number of renders in this pool.

4.3.2.7 renders_total

Total number of renders in this pool and renders in all child pools.

4.3.2.8 run_tasks

Number of total running tasks of all renders in the pool.

4.3.2.9 run_capacity

Total capacity of all running tasks in the pool.

4.3.2.10 task_start_finish_time

Time when first task was started or last task was finished.

4.3.3 Editable Parameters

4.3.3.1 annotation

Just some informative string that will be shown in GUI.

4.3.3.2 capacity_host

Pool hosts capacity.

4.3.3.3 exit_no_task_time

Render will exit having no task for this time. By default it is `-1`, feature is disabled. When this parameter is changed on pool, all its renders will receive it.

4.3.3.4 heartbeat_sec

Renders heart beat period in seconds. Each heart beat render receives events from server, launches new tasks and processes running tasks output, sends an update message to server and receives a new events as an answer. When this parameter is changed on pool, all its renders will receive it.

4.3.3.5 max_tasks_host

Pool hosts maximum running tasks.

4.3.3.6 new_nimby

New render will be registered in *NIMBY* state. Useful when new host was created, Afanasy installed, but render software is not. Afanasy can be used to install render software. *Maintenance* job can ignore *NIMBY* and *PAUSED* render state. *NIMBY* state can turned off automatically.

4.3.3.7 new_paused

New render will be registered in *PAUSED* state for maintenance purposes.

4.3.3.8 no_task_event_time

If render has no task for this time (seconds), server will emit this event `RENDER_NO_TASK`. Next time event will be repeated after twice longer duration. By default it is `-1`, feature is disabled.

4.3.3.9 overload_event_time

Event `RENDER_OVERLOAD` will be emitted if it has no free memory, disk or swap. Next time event will be repeated after twice longer duration (seconds). By default it is `-1`, feature is disabled.

4.3.3.10 power_host

Pool hosts *power*. This is just any custom integer. Job can filter renders for some minimum power.

4.3.3.11 properties_host

Pool hosts *properties*. This is just any custom string. Job can filter renders for matches this string.

4.3.3.12 resources_update_period

Render updates resources periodically, this is number of heart beats to do it. AfWatch farm monitor will ask for renders resources according to this parameter. When this parameter is changed on pool, all its renders will receive it.

4.3.3.13 sick_errors_count

Number of errors from different users render considered as *SICK*. On any error render remembers task job user and counts them. On any success task finish this count will be reset. *RENDER_SICK* event can be used to notify admin that some machine can't render.

4.3.3.14 services

Services names list that pool renders can run.

4.3.3.15 services_disabled

Disabled services names list that pool renders can not run. If some parent pool allows to run some service, you can disallow to run in child pool. Also it is useful for temporary service disabling, to not to delete service and remember that it is just disabled for some time.

4.3.3.16 tickets_pool

Total tickets the pool has. For example, to limit licenses,you can set `NUKE:20` tickets to the root pool. And nuke tasks should have `NUKE:1` ticket.

4.3.3.17 tickets_host

Each render in the pool have such tickets. For example, to limit RAM, you can set `MEM:64` tickets to some pool with renders which have 64GB RAM. And each render in the pool can run only one task with `MEM:64` tickets, or 2 tasks with `MEM:32` tickets, or 1 with `MEM:32` and 3 with `MEM:10` at the same time.

4.3.3.18 zombie_time

If server will not receive an update message from render for this time, render is considered as zombie (connection is lost) and goes to offline state.

4.3.3.19 idle_wolsleep_time

Time in seconds to put an idle machine to sleep. If this value is set to zero, machines will never put to sleep automatically.

4.3.3.20 idle_free_time

Time in seconds set an idle machine with Nimby to free. Zero or negative value disables the feature.

4.3.3.21 busy_nimby_time

Time in seconds set a machine with busy CPU and no Afanasy task to Nimby. Zero or negative value disables the feature.

4.3.3.22 idle_cpu

CPU usage percentage machine considered as idle.

4.3.3.23 busy_cpu

CPU usage percentage machine considered as busy.

4.3.3.24 idle_mem

Memory used percentage machine considered as idle.

4.3.3.25 busy_mem

Memory used percentage machine considered as busy.

4.3.3.26 idle_swp

Swap used percentage machine considered as idle.

4.3.3.27 busy_swp

Swap used percentage machine considered as busy.

4.3.3.28 idle_hddgb

Free disk space in Gigabytes machine considered as idle.

4.3.3.29 busy_hddgb

Free disk space in Gigabytes machine considered as busy.

4.3.3.30 idle_hddio

Disk I/O usage percentage machine considered as idle.

4.3.3.31 busy_hddio

Disk I/O usage percentage machine considered as busy.

4.3.3.32 idle_netmbs

Network send plus receive speed in Megabytes per second machine considered as idle.

4.3.3.33 busy_netmbs

Network send plus receive speed in Megabytes per second machine considered as busy.

4.3.4 State

Busy	At least one pool render runs some task
Paused	Pool renders does not accept any tasks

4.4 Tickets

Ticket is some named counter.

Pool can have *host* and *pool* tickets. *Host* means that each pool host has such tickets. *Pool* means that an entire pool has this tickets.

If job block has tickets it can run only on pools and renders that has such tickets enough. Each block task will be produced with block tickets. When render starting task with tickets, it counts tickets usage. Pool counts total tickets usage.

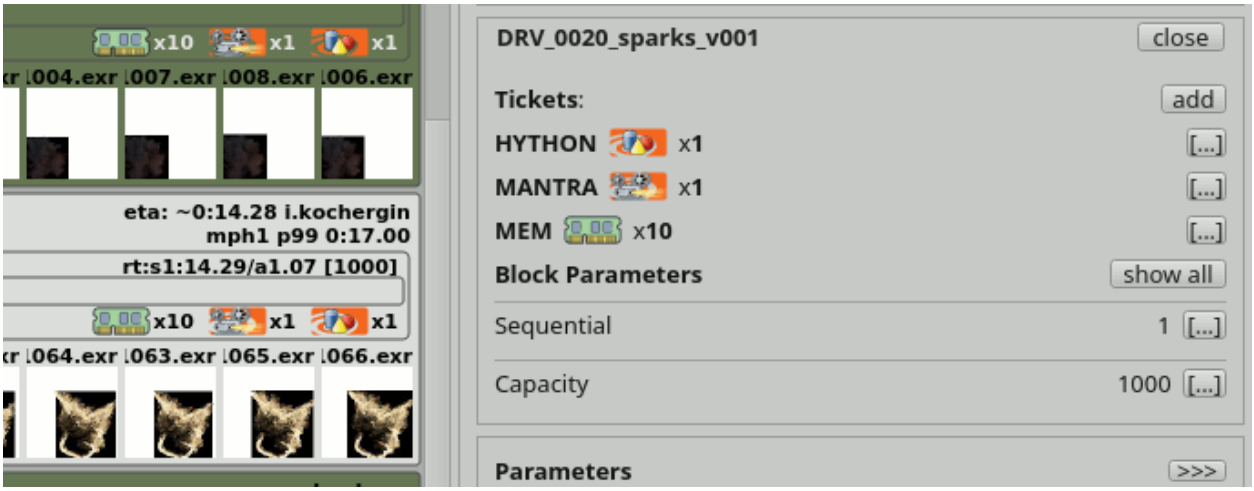


Fig. 5: AfWatch job block tickets

For example if you want to limit licenses on the entire farm, you can set `NUKE:20` *pool* tickets on the root pool. If you want to limit RAM usage on each pool host, You can set `MEM:64` *host* tickets on a pool which renders has 64GB RAM. And each pool render will be able to run one task with `MEM:64` tickets, or 2 tasks with `MEM:32`, or 1 `MEM:32` and 3 `MEM:10` tasks at the same time. Just one side effect will appear in this case, pools will count total `MEM` tickets and renders will count `NUKE` tickets too.

Ticket can be displayed as some custom image, if a png file with the same name exists in the directory:

```
cgru/icons/tickets
```

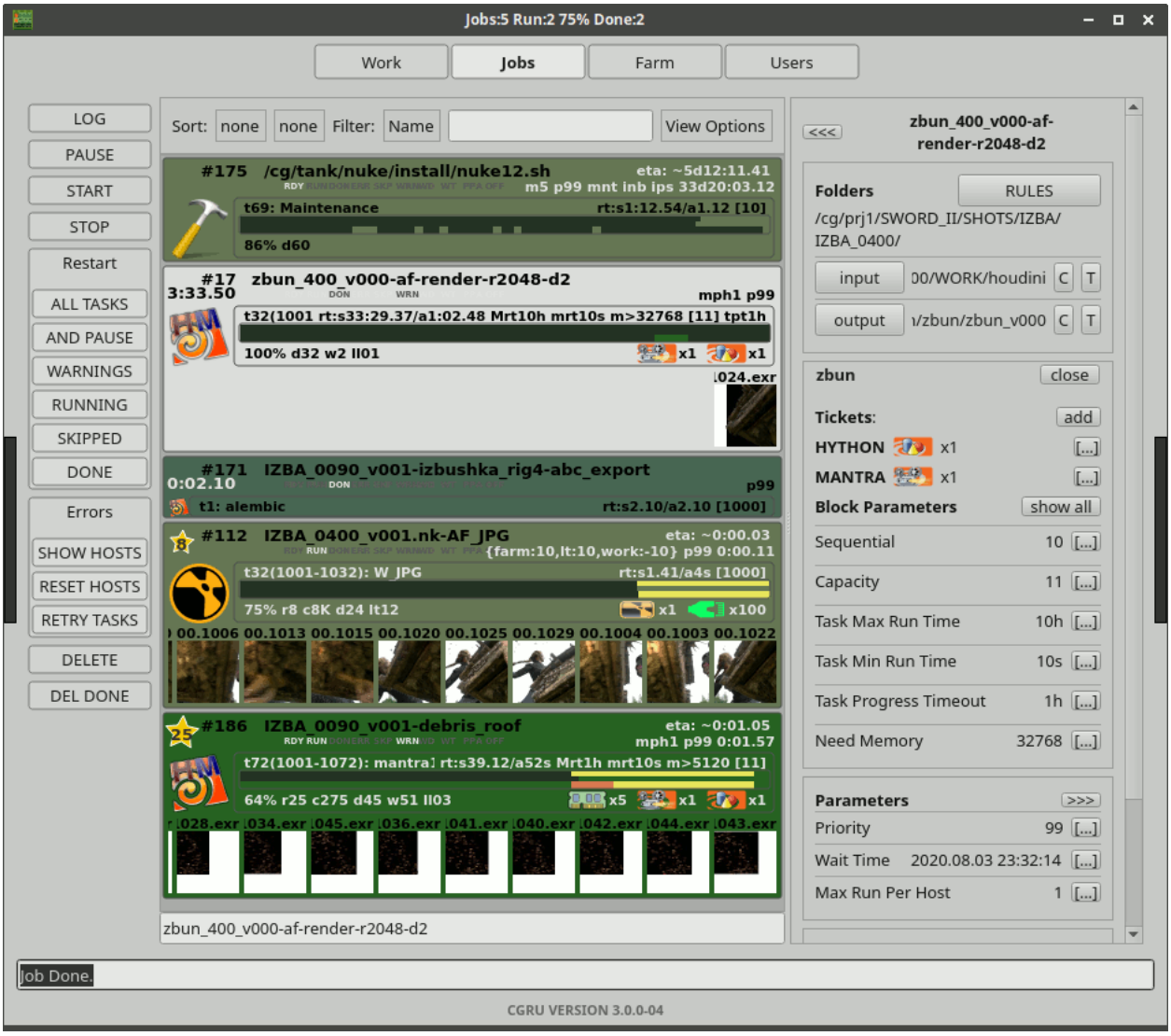
If a ticket icon file exists, GUI will replace ticket name with its icon. Unlike serivce, where icon is painted along with name. Also ticket icon will not be resized to a square image, so ticket can be painted as a rectangle. This is done to make tickets and services more differ in a GUI, to not to mess them. And they will be differ more, if tickets names will be uppercase, unlike services, that are all lowercase.

Also pool tickets has a maximum hosts limit. This is mostly needed for licence hosts limits. There is a common type of licensing where you can run multiple instances of software on same host, occupying only single license.

4.5 Watch

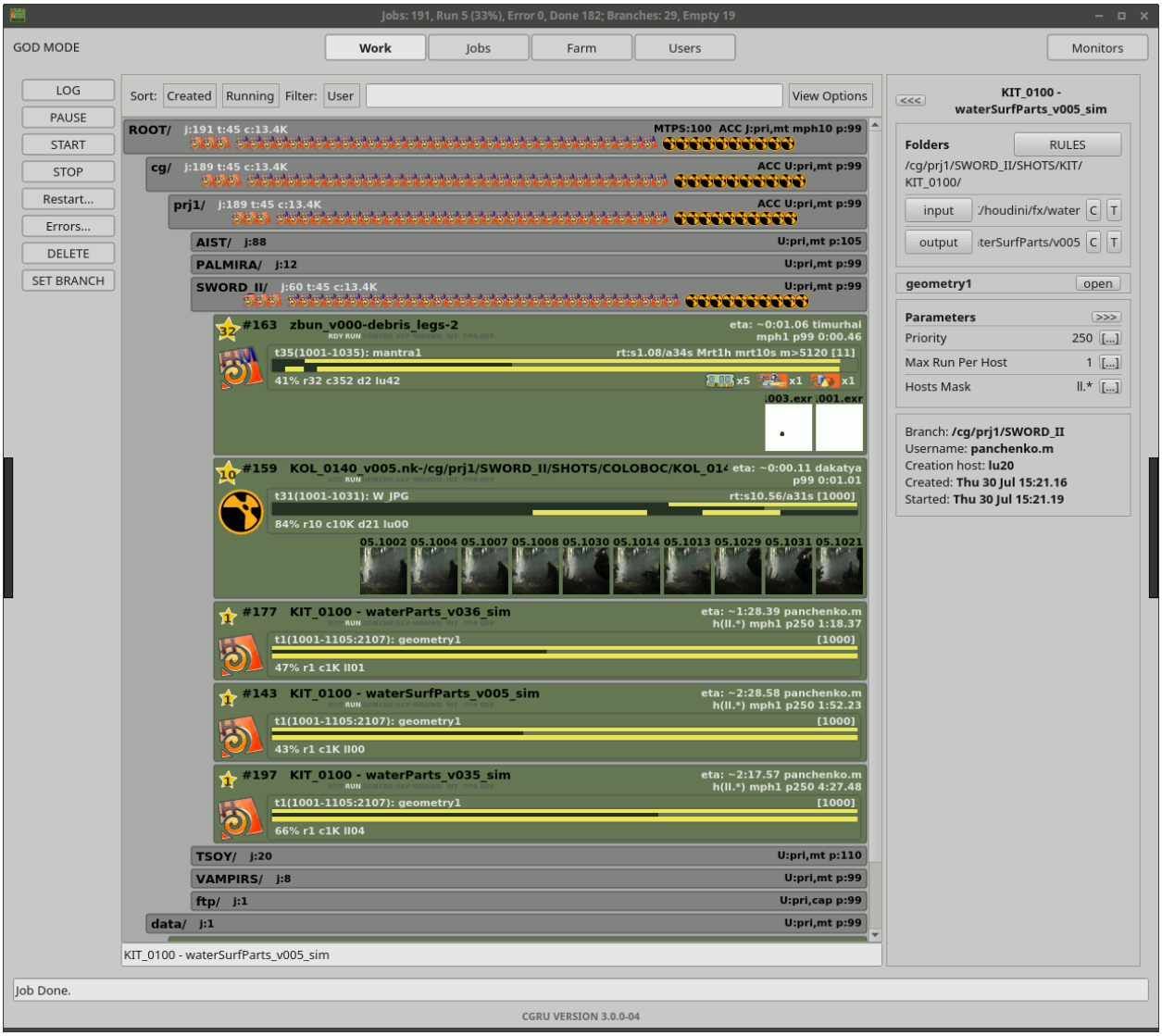
Watch is a Qt GUI for Afanasy.

4.5.1 Jobs



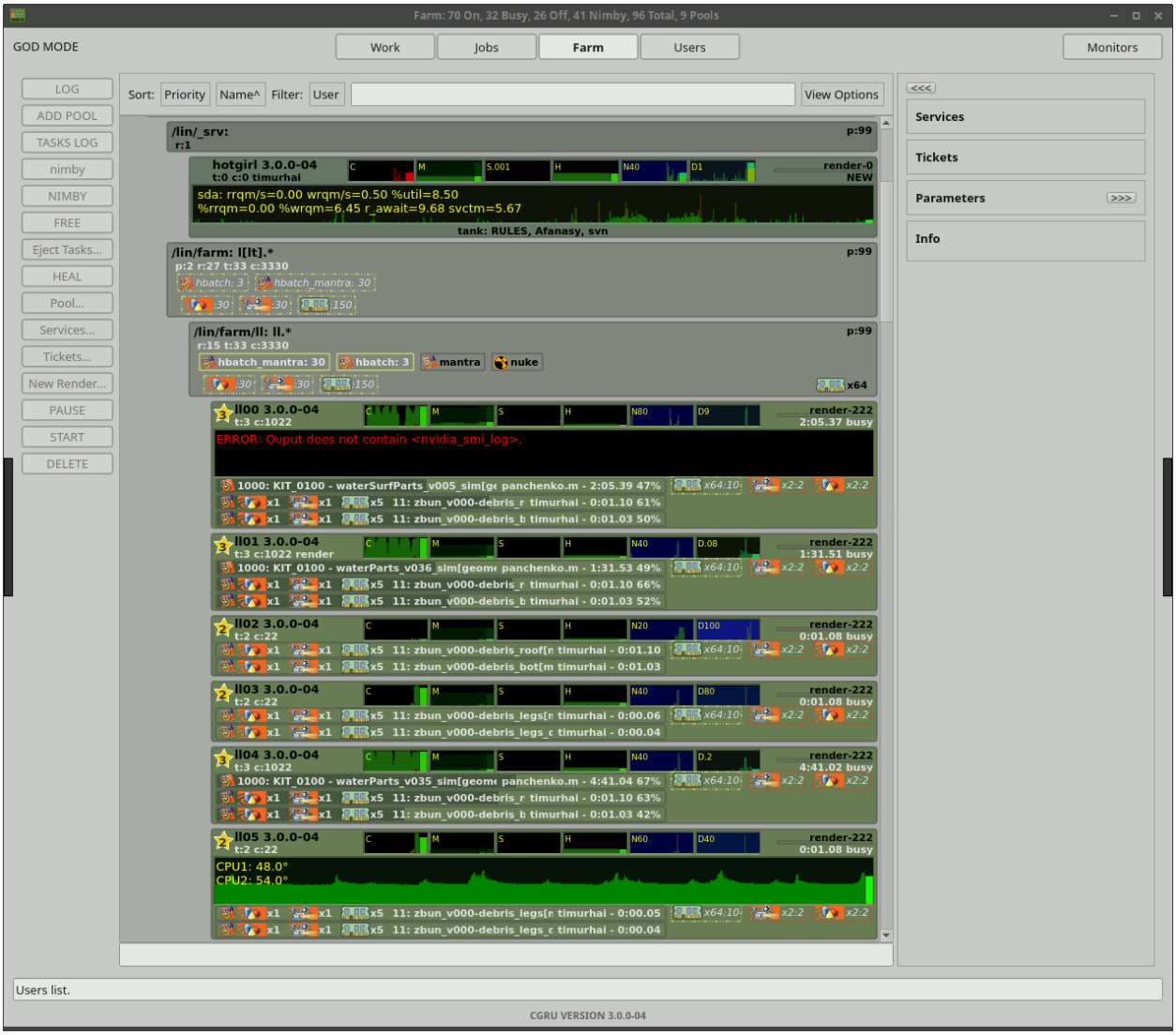
This is a list of user jobs.

4.5.2 Work



This is a hierarchy of branches and jobs from all users. Only VISOR can make changes here. For a common user this list is read only.

4.5.3 Farm



This is a hierarchy of pools and renders. Only admin can make changes here. For a common user this list is read only.

4.5.4 Users

This is a list of all Afanasy users. User can manipulate only own node. Admin can manipulate any user node.

4.5.5 Modes

It has several user modes for farm administration purposes:

- **USER** user mode:

  User can change his parameters and operate his jobs only.

- **VISOR** super user mode:

  Can change parameter of any user and operate jobs of any user.

- **GOD** admin mode:

  Can do anything, operate any user, any job and farm (renders and pools).

To switch user mode you need to type password in active Watch window (you do not need to re-login to switch between modes). To reset super user mode you can type password again.

You can configure passwords in a CGRU config JSON files (see *Configuration* page). Passwords are defined by configuration variables:

4.5.5.1 pswd_visor

Visor passwords *md5* sum. Password must be 5 characters length.

Default value is *1832116180fdc61b64fd978401e462e9*

It is `idkfa`

4.5.5.2 pswd_god

God password *md5* sum Password must be 5 characters length.

Default value is *73bcaaa458bff0d27989ed331b68b64d*

It is `iddqd`

4.5.6 UI Levels

At first Watch was designed as a minimalistic GUI, like hand watches. With a lots of abbreviated words, to fit lots of information on a small screen. But later it begin to grow and got several UI levels to display brief or full info.

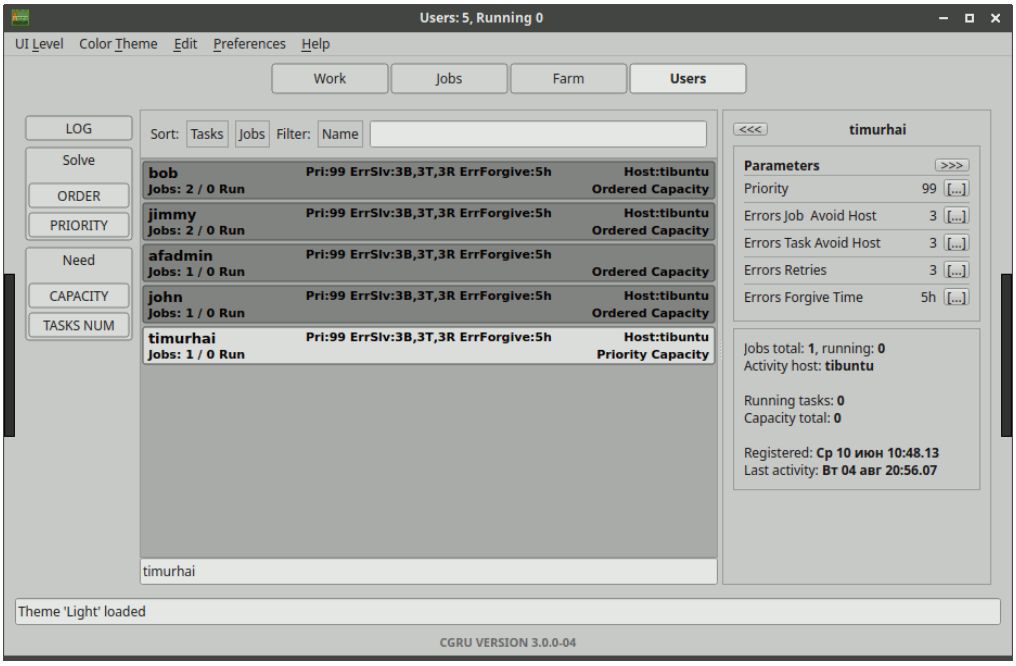
- **Padawan** You will see parameters with a full names. Useful when you just started to use Afanasy.
- **Jedi** You will see parameters with a brief names. Useful when you know most parameters.
- **Sith** You will see parameters with a abbreviated names. It is like on hand watches, where you see "TU 4", instead of "Today it is Tuesday, 4th August 2020 Year", as you are definitely know that it is August 2020, "TU" is Tuesday, and next "4" number means the day of month.

  Main window menu will disappear for space economy. It will be available on RMB menu at window top.

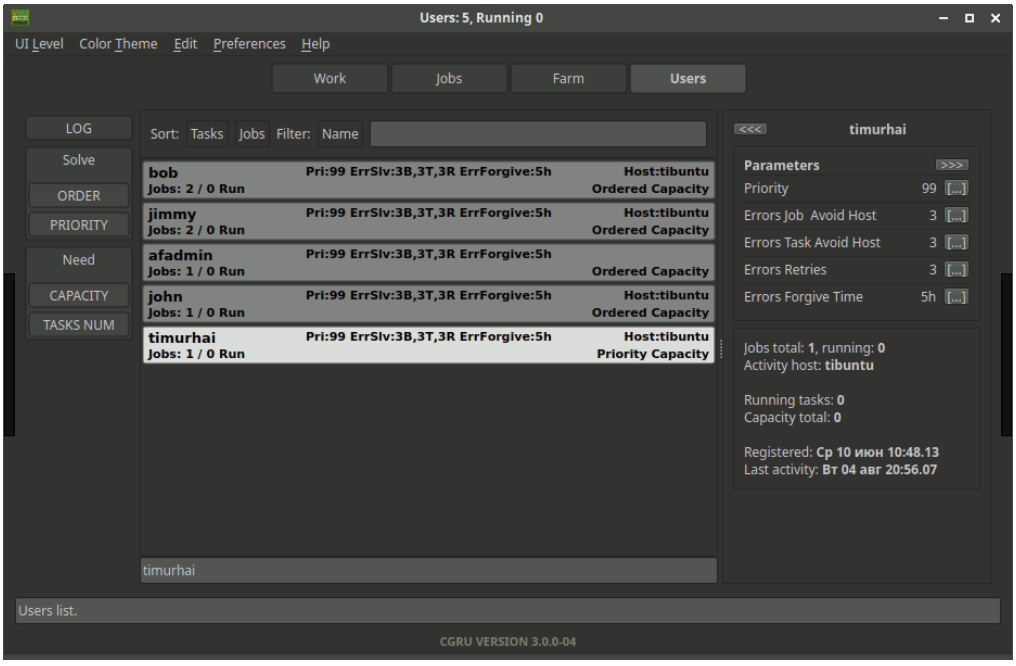
4.5.7 Styles

Watch GUI has a several Color Themes:

4.5.7.1 Light



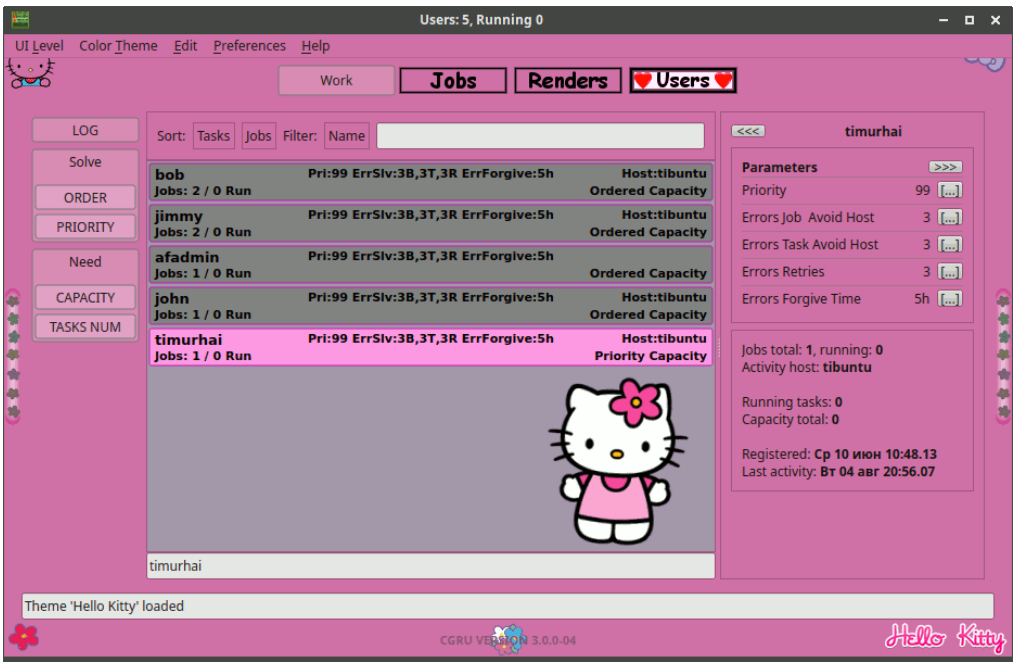
4.5.7.2 Dark



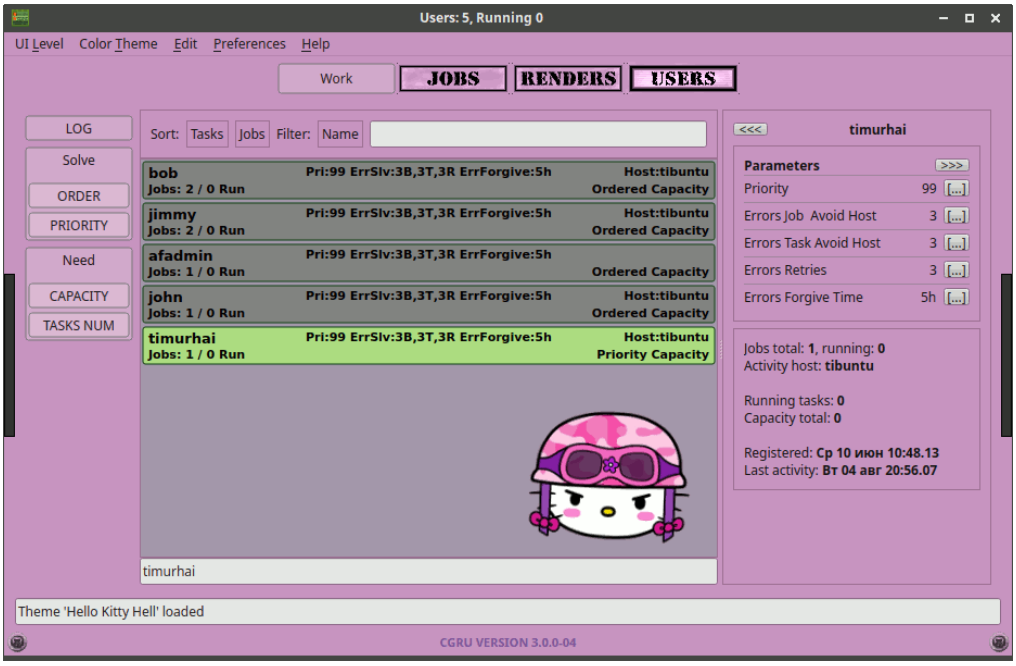
4.5.7.3 Military

First versions has no Color Themes and UI Levels. AfWatch GUI always had Military theme and Jedi level.

4.5.7.4 Hello Kitty



4.5.7.5 Hello Kitty Hell



4.6 Web GUI

There is a Web GUI for Afanasy.

Afanasy server supports some web server functions to be able to run this GUI. No other software or plugins needed.

Too see Afanasy page, type `server_name:51000` in a browser address bar, where 5100 is the default Afanasy server port.

Web GUI is useful for administrators to manage farm from some special machines (devices) where CGRU can't be installed.

If you setup some VPN, you can monitor jobs from smartphone.

Important: Try not to use Web GUI as a common Afanasy GUI instead of Watch. Web GUI consumes much more resources and on client and on server side. Also it needs much more traffic. On a large number of GUI items browser can take lots of RAM. If browser tries to show more that 10000 items it can hung.

4.6.1 Online Version

Online version is designed to let users to try Afanasy via WEB GUI. You can change parameters, restart jobs. You even can to delete everything and it will be empty till the next demo start.

http://afanasy.cgru.info:51000

- This is not a real working server at CG company.
- All render clients and server are running on the same host.
- Server does not accept new jobs.
- Server does not allow to change existing tasks commands.
- All services (tasks) starts a simple python script that just sleeps for some time.

4.6.2 HTTP Server Configuration

There is some Afanasy server settings stands for HTTP serving. This settings can help you to setup some custom web GUI.

- **`af_http_serve_dir`** Afanasy server HTTP serve folder. If not set or empty CGRU root folder will be used.

- **af_http_site_index** Afanasy server HTTP response on an empty GET request. Default: `/afanasy/browser/index.html`
- **af_http_directory_index** Afanasy server HTTP response on a directory GET request. Default: `index.html`

You can even make server to serve several GUIs and let artists to choose one.

4.7 Job

Afanasy Job can have one or more blocks. Blocks have tasks. Blocks are needed to store some same parameters for all tasks it consists of. For example all block tasks have the same working directory, capacity, service and parser type. But commands are differ.

Jobs are created by Python API by some submission script:

```
import af

job = af.Job(job_name)
```

4.7.1 Attributes

4.7.1.1 name

`af.Job(str)`

`af.Job.setName(str)`

Each job has an unique name. If new a job comes to server with the name which already exists, server change it's name by adding a number. Jobs dependences bases on their names and depend mask to match it. Afanasy uses standard Regular Expressions. The same expressions are in Python, Perl, JavaScript, PHP and other languages.

4.7.1.2 user_name

`af.Job.setUserName(str)`

User name who has created the Job. Python API constructs a job with the current user name.

4.7.1.3 host_name

Host name where job was created.

4.7.1.4 time_creation

Time when the job was created.

4.7.1.5 time_started

Time when the job was started (produced the first task).

4.7.1.6 time_done

Time when the job was done (last task finished).

4.7.1.7 description

`af.Job.setDescription(str)`

Any custom description. For statistics database QSL queries only.

4.7.1.8 blocks[]

`af.Job.blocks[] (array)`

Job consists of block(s).

4.7.2 Editable Parameters

4.7.2.1 priority

`af.Job.setPriority(int)`

Job with a greater priority will run first.

4.7.2.2 max_running_tasks

`af.Job.setMaxRunningTasks(int)`

Maximum number of running tasks at the same time.

4.7.2.3 max_running_tasks_per_host

`af.Job.setMaxRunTasksPerHost(int)`

Maximum number of running tasks at the same time at the same host.

4.7.2.4 hosts_mask

`af.Job.setHostsMask(str)`

Job run only on renders which host name matches this mask.

4.7.2.5 hosts_mask_exclude

`af.Job.setHostsMaskExclude(str)`

Job can not run on renders which host name matches this mask.

4.7.2.6 pools

```
af.Job.setPools(dict)
```

Pools is a string and number pairs (map<sting,int>). Each pair represents pool name string and pool priority number.

4.7.2.7 depend_mask

```
af.Job.setDependMask(str)
```

Job will wait other user jobs which name matches this mask.

4.7.2.8 depend_mask_global

```
af.Job.setDependMaskGlobal(str)
```

Job will wait other jobs from any user which name matches this mask.

4.7.2.9 time_wait

```
af.Job.setWaitTime(seconds)
```

Time to wait to start a job.

4.7.2.10 ppa

```
af.Job.setPPApproval()
```

Preview Pending Approval parameter plays role only when job block(s) has a non-sequential tasks solving. When PPA is turned on, job renders only non-sequential tasks (for example just each 10 frame). Then job state falls into PPA and it stops to solve any tasks. Artist can check each 10 job frames. And, depending on the results, continue job or not. To continue job, you can turn PPA parameter off.

4.7.2.11 maintenance

```
af.Job.setMaintenance()
```

Job will run on tasks which name matches render name. Useful for "Maintenance" jobs, when you want some command run only once on each render. For example you can install software this way.

4.7.2.12 ignorenimby

```
af.Job.setIgnoreNimby()
```

Job tasks will run on render even it has "Nimby" state. Useful for "Maintenance" jobs.

4.7.2.13 ignorepaused

```
af.Job.setIgnorePaused()
```

Job tasks will run on render even it has "Paused" state. Useful for "Maintenance" jobs.

4.7.2.14 need_os

`af.Job.setNeedOS(str)`

`af.Job.setNativeOS()`

Job will run only on hosts which name contains this mask. Python *setNativeOS()* function will automatically set needed OS the same that it run.

4.7.2.15 need_properties

`af.Job.setNeedProperties(str)`

Job will run only on hosts with custom properties contains this mask. It's custom host parameter can be defined in farm description.

4.7.2.16 command_pre

`af.Job.setCmdPre(str)`

Command to execute on job registration. Note, that this command is executed by server, and not from tasks working directory. Use absolute paths here or even transfer paths if you server has another file system than renders. If somebody executes 'sleep 1000', other commands execution (and jobs registration) will be delayed on 1000 seconds (only delayed, not lost). Try not use Pre Command at all. You always can create one more task(block) and make other tasks(blocks) depend on it.

4.7.2.17 command_post

`af.Job.setCmdPost(str)` Command executed on job deletion. Usually used to delete temporary render scene. This commands are executed on render farm hosts by special system job. Working directory of such system task will be the first block working folder.

4.7.2.18 time_life

`af.Job.setTimeLife(seconds)`

Maximum job age in seconds. When job age becomes greater then life time if will be automatically deleted in any case. It is useful for some technical jobs to prevent their amount rise. User can set default Life Time value for all its jobs.

4.7.2.19 annotation

`af.Job.setAnnotation(str)`

Job annotation. Does not influence anything. This string will be shown in a GUI item.

4.7.2.20 report

Job annotation. Does not influence anything. This string will be shown in a GUI item. It should be set from a task parser: self.report

4.7.3 State

Ready	RDY	Job is ready to produce a task.
Running	RUN	Job has running tasks.
Done	DON	All job tasks are done (may be some skipped).
Error	ERR	Job has some error tasks.
Skipped	SKP	Some job tasks are skipped.
Waiting Dependencies	WD	Job waits some other jobs to be done.
Waiting Time	WT	Job waits some time to start.
Preview Pending Approval	PPA	Job has rendered all non-sequential tasks and waits approval.
Offline	OFF	Flag to server not to solve a job.

4.8 Job Block

Afanasy Job Block keeps the same parameters for all its tasks. When task is generated (when it ready to run) it take such parameters as working folder, capacity, service from block.

Blocks can be *numeric* (most blocks in Afanasy are numeric). Numeric blocks does not have tasks at all. Such block keeps parameters for all its tasks itself. For example it has a command as

```
render -s @#@ -e @#@"
```

where `@#@` will be replaced with start and end frames for each task.

```
import af

job = af.Job(job_name)

block = af.Block(name, service)

job.blocks.append(block)
```

4.8.1 Attributes

4.8.1.1 name

`af.Block(name, service)`

`af.Block.setName()`

Each Block has an unique name. If new Block added to Job which the name already exists, Job change it's name by adding a number. Blocks dependence bases on their names and depend masks to match it.

4.8.1.2 tasks_num

The number of tasks in block.

4.8.1.3 frame_first

`af.Block.setNumeric(start, end, pertask, inc)`

First block frame.

4.8.1.4 frame_last

```
af.Block.setNumeric(start, end, pertask, inc)
```

Last block frame.

4.8.1.5 frames_inc

```
af.Block.setNumeric(start, end, pertask, inc)
```

In various software also known as frames step, jump, by frame. You can use it if you want to render only each third frame, for example.

4.8.1.6 frames_per_task

```
af.Block.setNumeric(start, end, pertask, inc)
af.Block.setFramesPerTask(int)
```

Number of frames in each task. When block is not numeric (has tasks with individual commands) used to compute blocks per task dependency. Negative values means sub-frame dependency. For example you can render tiles or generate shadows in one block and generate mantra 'ifd' files in another with 1 frame per task. And if task has 4 shadows or 2x2 tiles you can set -4 frames per task for depended block.

4.8.2 Editable Parameters

4.8.2.1 tasks_name

```
af.Block.setTasksName(str)
```

Block tasks names pattern.

- *Block - numeric, pattern - not empty*: generated task will fill this pattern with its first and last frames numbers.
- *Block - numeric, pattern - empty*: tasks will take first_frame - last_frame name.
- *Block - not numeric, pattern - not empty*: task fill this pattern with it name.
- *Block - not numeric, pattern - empty*: task simply take its name.

4.8.2.2 sequential

```
af.Block.setSequential(int)
```

By default, sequential is 1, tasks will be solved from the first to the last one by one. If this parameter is -1, tasks will be solved from the last to the first one by one. If this parameter is greater than 1 or less than -1, 10 for example, tasks with every 10 frame will be solved at first, than other tasks. If -10, every 10 frame but from the end. Important thing that task frame is used for sequential calculation, not task number. If sequential is 0, always middle task will be solved. For example if frame range is 1-100, tasks solving order will be: 1,100,50,25,75 and so on.

4.8.2.3 service

```
af.Block.setService(str)
```

The name of a block tasks service type.

4.8.2.4 parser

```
af.Block.setParser(str)
```

Block tasks output parser name.

4.8.2.5 working_directory

```
af.Block.setWorkingDirectory(str)
```

Tasks process working directory.

4.8.2.6 environment

```
af.Block.setEnv(name, value)
```

Tasks process extra environment. Each task in a block will automatically get the following environment variables:

```
AF_JOB_ID = tasks job id
AF_BLOCK_ID = tasks block id
AF_TASK_ID = tasks task id
```

4.8.2.7 command_post

```
af.Block.setCmdPost(str)
```

Like job "command_post" but for each block. Working directory of this command will be this block working folder.

4.8.2.8 capacity

```
af.Block.setCapacity(int)
```

Task checks available capacity on render to run on it. Capacity can be static (by default) and variable - base value and coefficients:

4.8.2.9 capacity_coeff_min

4.8.2.10 capacity_coeff_max

```
af.Block.setVariableCapacity(min, max)
```

Block can generate tasks with capacity*coefficient to fit free render capacity. Task commands will be searched for the special string to replace it with capacity coefficient number. This command replacement performs render by service python class instance. Any service can describe own rule for this replacement by implementation of base class method. By default, base service class performs command.replace('@AF_CAPACITY@', str(capacity)). You can specify number of CPUs to use for your applications (if it supports it by command line arguments).

4.8.2.11 multihost_min

4.8.2.12 multihost_max

4.8.2.13 multihost_max_wait

4.8.2.14 multihost_master_on_slave

4.8.2.15 multihost_service

4.8.2.16 multihost_service_wait

`af.Block.setMultiHost(min, max, wait, master_on_slave=False, service=None, service_wait=-1)`

A singe block task can run on several hosts. You can specify minimum and maximum number of hosts that task can take. Time in seconds to wait for maximum hosts. Whether the master host will be in slaves list. For example if task took 'r00', 'r01', 'r02', 'r03', 'r04' hosts master command will be executed on 'r00' and 'r00' will be in slaves list too. If tasks has multihost service you can't enable this parameter, because only one command can be executed master or slave. Command to execute on slaves hosts, if it is empty, no service will be executed. Time in seconds to wait for master execution after slaves execution.

4.8.2.17 max_running_tasks

`af.Block.setMaxRunningTasks(int)`

Maximum number of tasks block can run on the same time.

4.8.2.18 max_running_tasks_per_host

`af.Block.setMaxRunTasksPerHost(int)`

Maximum number of tasks block can run on the same time on the same host.

4.8.2.19 hosts_mask

`af.Block.setHostsMask(str)`

Block run only on Renders which host name matches this mask.

4.8.2.20 hosts_mask_exclude

`af.Block.setHostsMaskExclude(str)`

Block can not run on renders which host name matches this mask.

4.8.2.21 depend_mask

`af.Block.setDependMask(str)`

Block will wait other job blocks which name matches this mask.

4.8.2.22 tasks_depend_mask

`af.Block.setTasksDependMask(str)`

Block task will wait other job blocks task which name matches this mask.

4.8.2.23 errors_retries

`af.Block.setErrorsRetries(int)`

Number of task errors to retry it automatically. Value '-1' means take this value from user settings.

4.8.2.24 errors_avoid_host

`af.Block.setErrorsAvoidHost(int)`

Maximum number of errors on same host. Block begins to avoid render host name if number of errors on it greater or equal this value. Zero value means no limit. Value '-1' means take this value from user settings.

4.8.2.25 errors_task_same_host

`af.Block.setErrorsTaskSameHost(int)`

Maximum number of errors for task on same host. Task begin to avoid this host name of errors on it greater or equal this value. Zero value means no limit. Value '-1' means take this value from user settings.

4.8.2.26 errors_forgive_time

`af.Block.setErrorsForgiveTime(int)`

Time form last error to forgive error host (reset it's errors count). Zero value means no forgive. Value '-1' means take this value from user settings.

4.8.2.27 task_max_run_time

`af.Block.setTaskMaxRunTime(seconds)`

Task maximum time to run. After this time task will be set to error (and may be automatically restarted according to Error Retries value). If this value equals or less than zero, no task run time limit exists.

4.8.2.28 task_min_run_time

`af.Block.setTaskMinRunTime(seconds)`

Task minimum time to run. If task will finished with success for a time less this value, it will be treated as an error. If this value equals or less than zero, this limit will be disabled.

4.8.2.29 task_progress_change_timeout

`af.Block.setTaskProgressChangeTimeout(seconds)`

If running task progress (percentage) will be the same for this time, task will be stopped with error. If this value equals or less that zero, no such limit exists.

The default value can be set by `af_task_progress_change_timeout` config variable. It is a server side variable, you can ask server to reload config without restarting by `afcmd cload` command. See Configuration and afcmd sections.

4.8.2.30 need_power

`af.Block.setNeedPower(int)`

Minimum render host power needed. It's custom host parameter can be set by pool.

4.8.2.31 need_memory

`af.Block.setNeedMemory(int)`

Minimum render host free memory needed in megabytes.

4.8.2.32 need_gpu_mem_mb

`af.Block.setNeedGPUMemGB(float)`

Minimum render host GPU free memory needed in gigabytes. The function will convert it to integer megabytes.

4.8.2.33 need_cpu_freq_mgz

`af.Block.setNeedCPUFreqGHz(float)`

Minimum render host CPU frequency in gigahertz. The function will convert it to integer megahertz.

4.8.2.34 need_cpu_cores

`af.Block.setNeedCPUCores(int)`

Minimum render host CPU cores number.

4.8.2.35 need_cpu_freq_cores

`af.Block.setNeedCPUFreqCores(float)`

Minimum render host CPU frequency * cores in gigahertz. The function will convert it to integer megahertz.

4.8.2.36 need_hdd

`af.Block.setNeedHDD(int)`

Minimum render host free disk space needed in gigabytes.

4.8.2.37 need_properties

`af.Block.setNeedProperties(str)`

A mask to much render host properties to run on it. It's a custom host parameter can be set by pool.

4.8.2.38 command

`af.Block.setCommand(str)`

Tasks command pattern. When block produces a task it calculates an unique command from this pattern and other parameters, depend on block type - numeric or string, replacing @#@ pattern with a number. Padding is specified by the number of "#" symbols between "@" symbols.

String: block seek for "@#@" string in command and replace it by another string get from Task Command.

Example:

command: `myrender some.scene -camera @#@`

arguments = `['sun','sky','front','side','bottom']`

Result:

1st task command: `myrender some.scene -camera sun`

2nd task command: `myrender some.scene -camera sky`

3rd task command: `myrender some.scene -camera front`

If block command is empty Task Command is simply used.

Numeric block calculates first and last frame for the task according to task number, Frame First, Frame Last, Frame per Host and Frame Increment values. Each of @#@ pairs will be replaced with the start and end numbers.

Examples:

command: `myrender some.scene -s @#@ -e @#@`

frame_first: 1, frame_last: 10, frames_per_task: 4

Result:

1st task command: `myrender some.scene -s 1 -e 4`

2nd task command: `myrender some.scene -s 5 -e 8`

3rd task command: `myrender some.scene -s 9 -e 10`

command: `myrender something.@#####@.obj`

frame_first: 1, frame_last: 10, frames_per_task: 1

Result:

1st task command: `myrender something.0001.obj`

2nd task command: `myrender something.0002.obj`

last task command: `myrender something.0010.obj`

You can check numbers filling by command:

```
afcmd numcmd service frame_start frame_end command
```

4.8.2.39 files[]

```
af.Block.setFiles(str[])
```

Each task can have result file(s) pattern. Result file name will be constructed from this pattern by the same method as described before. Some another application, for example watch GUI, can execute your favorite image viewer program [file] and preview result frame.

Python function will extend an existing files array.

Example (numeric):

Block files: `images/back.@####@.exr`

Preview command: `nuke -v @ARG@`

Result for 57 frame: `nuke -v images/back.0057.exr`

Example (not numeric):

Block files: `images/back.@#@.exr`

Task files: `0057`

Preview command: `nuke -v @ARG@`

Result: `nuke -v images/back.0057.exr`

Task can have several files for preview, for example when several render passes or a stereo images pair.

If block is not numeric and block view command is empty only task view command is used.

Watch will execute command in a task block working directory.

4.8.3 Flags

4.8.3.1 numeric

```
1 << 0
```

Numeric

4.8.3.2 varcapacity

```
1 << 1
```

```
af.Block.setVariableCapacity(min, max)
```

4.8.3.3 multihost

```
1 << 2
```

```
af.Block.setMultiHost(min, max, wait, master_on_slave=False, service=None,
service_wait=-1)
```

4.8.3.4 masteronslave

```
1 << 3
```

```
af.Block.setMultiHost(min, max, wait, master_on_slave=False, service=None,
service_wait=-1)
```

4.8.3.5 dependsubtask

```
1 << 4
```

```
af.Block.setDependSubTask()
```

For tasks with several frames calculate sub task dependence. Useful for simulation and render when not all frames simulated.

4.8.3.6 skipthumbnails

```
1 << 5
```

```
af.Block.skipThumbnails()
```

Do not try to generate any thumbnails.

4.8.3.7 skipexistingfiles

```
1 << 6
```

```
af.Block.skipExistingFiles( size_min = -1, size_max = -1)
```

AfRneder can check files on client just before task start (in a Python service class initialization). It can skip task command launch if file(s) are exist. If size_min or(and) size_max are positive, it will check size too. Block(task) files parameter should be set properly.

4.8.3.8 checkrenderedfiles

```
1 << 7
```

```
af.Block.checkRenderedFiles(size_min = -1, size_max = -1)
```

AfRneder can check files on client just after task finish (in a Python service class). It can set task as error if file(s) are not exist. If size_min or(and) size_max are positive, it will check size too. Block(task) files parameter should be set properly.

4.8.3.9 slavelostignore

```
1 << 8
```

```
af.Block.setSlaveLostIgnore()
```

On a slave host missing, multi-host task will not restart. It will just ignore this.

4.8.4 State

Ready	RDY	Block is ready to produce a task.
Running	RUN	Block has running tasks.
Done	DON	All block tasks are done (or some skipped).
Error	ERR	Block has some error tasks.
Skipped	SKP	Some block tasks are skipped.
Waiting Dependencies	WD	Block waits some other blocks.

4.9 Job Task

Tasks are exists only in non-numeric blocks, where each task can have its own name and command. In numeric blocks tasks are generated on demand, as numeric block enough has information to generate any task. Most blocks are numeric, as tasks are differ only by few numbers in a command.

There are some cases when tasks commands differ by some strings, and block can't be described by frame numbers. For example *ffmpeg* converts various sequences and movies in a single job block (Rules constructs such jobs for previews).

```
import af

job = af.Job(job_name)

block = af.Block(name, service)

job.blocks.append(block)

task = af.Task(task_name)

block.tasks.append(task)
```

4.9.1 Attributes

If block is numeric all this attributes are generated on the fly by block.

4.9.1.1 name

`af.Task(str)`

Task name. Generated, if block is numeric.

4.9.1.2 command

`af.Task.setCommand(str)`

Command to execute. Generated, if block is numeric.

4.9.1.3 files[]

`af.Task.setFiles(str[])`

Files for preview. Generated, if block is numeric.

4.9.1.4 environment

`af.Task.setEnv(name, value)`

Tasks process extra environment. It will be merged with a block extra environment. Each task will automatically get the following environment variables:

```
AF_JOB_ID = tasks job id
AF_BLOCK_ID = tasks block id
AF_TASK_ID = tasks task id
```

4.9.1.5 tst

Time when task was started (last start).

4.9.1.6 tdn

Time when task was done (last finish).

4.9.1.7 str

Number of times task has started (it can be manually or automatically restarted).

4.9.1.8 per

Running task progress percentage.

4.9.1.9 frm

Running frame for multiframe tasks which can be produced by numeric blocks when frames per render parameter > 1.

4.9.1.10 pfr

Running percentage of current running frame for multiframe tasks which can be produced by numeric blocks when frames per render parameter > 1.

4.9.1.11 err

Number of times the task produced an error.

4.9.1.12 hst

Host name where the task was started last time.

4.9.1.13 act

Last task activity. This is a sting to informate user only, does not influence anything. Activity can be parsed from task process output by Python parser class.

4.9.2 State

Read	RDY	Task can be executed.
Running	RUN	Task is running.
Done	DON	Task is done.
Error	ERR	Task finished with error or failed to start.
Skipped	SKP	Task skipped.
Waiting Dependencies	WD	Warning dependent tasks to be done.
Warning	WRN	Warning from parser.
Parser Error	PER	Error from parser.
Parser Bad Result	PBR	Bad result from parser.
Restated Error Ready	RER	Automatically restarted ERR task.

4.10 Branch

A branch is like a folder in a file-system. A branch can contain child branches (folders) and jobs (files), so there is a hierarchy of branches (folders) and jobs (files). Branches are designed to combine and manipulate a set of similar (department, project, scene, asset) jobs.

4.10.1 Creation

The first ROOT branch will be created by the system job.

Any job has a sting attribute branch. When a job comes to the server, it looks whether the job branch exists. If the branch exists, this branch becomes the new job parent. If the branch does not exist, server tries to find the parent branch of the new job branch. Then it tries to find the parent of the parent in a cycle (recursion). When it find a matching parent branch (it must do it, as a root branch always exists), it tries to create a child branch if the Auto Create Child ACC flag is set. If the flag is not set, the job will be parented to the most base branch in the branches hierarchy. And the job branch will be updated to the actual parent that the server could find/create.

4.10.2 Example

For example, you have projects mounted in /prj folder. So, you have such folders structure:

- /
  - prj/
    - bus/
    - car/
    - plane/
    - train/

Where *bus*, *car*, *plane* and *train* are project names.

Lets imagine that you have some scene file to render:

`/prj/train/shots/scene_a/work/scene.sc`

When you render that scene for the first time, the */prj* branch will be created. As the root branch has Auto Create Child (ACC) flag set by default. No more deeper branch(es) will created on this stage. As the auto-created branch does not have ACC flag set. At this stage job solving will be the same as there are no branches in Afanasy at all. But if you set ACC flag on a new */prj* branch, each project will create it's own branch. So you can manipulate jobs that belongs to some project. For example give some project more priority.

4.10.3 Attributes

4.10.3.1 name

Branch name, that represents branch full path. Root branch name is always `/`.

4.10.3.2 parent_path

Parent branch path (name). It is an empty string for the root branch.

4.10.3.3 time_creation

Time when branch was created.

4.10.3.4 branches_num

Number of child branches (direct childs, not childs of childs).

4.10.3.5 branches_total

Total number of child branches and all their sub-childs.

4.10.3.6 jobs_num

Number of child jobs (direct childs, not childs of childs).

4.10.3.7 jobs_total

Total number of child jobs and all sub-child branches jobs.

4.10.3.8 running_tasks_num

Number of tasks that branch jobs running.

4.10.3.9 running_capacity_total

Total capacity of all (total) running tasks.

4.10.4 Editable Parameters

4.10.4.1 priority

Branch solving priority.

4.10.4.2 max_tasks_per_second

Maximum tasks limit that branch can produce per second.

4.10.4.3 max_running_tasks

Maximum tasks limit that branch can run at the same time.

4.10.4.4 max_running_tasks_per_host

Maximum tasks limit that branch can run at the same time on the same host.

4.10.4.5 hosts_mask

Branch can be solved only on machines that name matches this mask (regular expression).

4.10.4.6 hosts_mask_exclude

Branch can not be solved on machines that name matches this mask (regular expression).

4.10.5 Flags

4.10.5.1 create_childs

Branch will create a child branch automatically, when a new job asks for it.

4.10.5.2 solve_jobs

By default, branch solves its jobs users by priority. But if this flag is set, branch will solve its jobs directly.

4.10.5.3 solve_method

Solve child nodes by priority or order.

4.10.5.4 solve_need

Solve child nodes by running capacity total or tasks number.

4.11 User

User is created on a new job from user that does not exist.

Or you can create user manually using CLI:

```
afcmd uadd username
```

4.11.1 Attributes

4.11.1.1 name

Each user should have an unique name. If some artists has the same login name, you can use `AF_USERNAME` environment variable to override it.

4.11.1.2 host_name

Host name where the last job was send from.

4.11.1.3 jobs_num

The number of jobs the user has.

4.11.1.4 running_jobs_num

The number of currently running jobs.

4.11.1.5 running_tasks_num

The number of currently running tasks.

4.11.1.6 time_register

Time when the user was registered.

4.11.1.7 time_activity

The last user activity time.

4.11.2 Editable Parameters

4.11.2.1 priority

User with greater priority can have more running tasks number (get more render hosts).

```
need = pow( 1.1, priority) / (running_tasks_num + 1.0)
```

Each priority point gives 10% hosts bonus (running tasks number).

4.11.2.2 max_running_tasks

Maximum number of running tasks user can have.

4.11.2.3 hosts_mask

User can run only on renders which host name matches this mask.

4.11.2.4 hosts_mask_exclude

User can not run on renders which host name matches this mask.

4.11.2.5 errors_retries

Default Error Retries value for user jobs.

4.11.2.6 errors_avoid_host

Default Errors Avoid Host value for user jobs.

4.11.2.7 errors_task_same_host

Default Errors Task Same Host value for user jobs.

4.11.2.8 errors_forgive_time

Default Errors Forgive Time value for user jobs.

4.11.2.9 jobs_life_time

Default Life Time value for user jobs.

4.11.2.10 annotation

Annotate user GUI item.

4.12 API

Afanasy can communicate via JSON protocol. And only JSON protocol. Any GUI, CLI or Python script (API) constructs JSON objects to send to server.

4.12.1 Python API

You can create jobs within Python, that exists in most common CG software. Afanasy Python module helps you to construct a valid JSON job object for server. Also it sends json data to server.

4.12.1.1 Example

```
# Import afanasy python module (must be in PYTHONPATH)
import af

# Create a job
job = af.Job('somejob')

# Set job depend mask
job.setDependMask('another_job_name')

# Set maximum tasks that can be executed simultaneously
job.setMaxRunningTasks(15)

# Set job hosts mask
job.setHostsMask('render.*')

# Start job paused
job.offLine()

# Create a block with provided name and service type
block = af.Block('back', 'nuke')

# Set block tasks command
block.setCommand('nuke -i -X WriteBack -x scene.nk.tmp.nk @#@,@#@')

# Set block tasks preview command arguments
block.setFiles(['jpg/img.@####@.jpg'])

# Set block to numeric type, providing first, last frame and frames per host
block.setNumeric(1, 100, 10)

# Add block to the job
job.blocks.append(block)

# Set command to execute by server after a job is deleted.
job.setCmdPost('rm /projects/test/nuke/scene.nk.tmp.nk')

# Send job to Afanasy server
job.send()
```

4.12.1.2 Job Class

- Constructor:
  - `job = af.Job(job_name = None)`

    Takes job name as a parameter (optional).
- Variables:
  - `job.blocks = []`

    Blocks list.
- Some Functions:
  - `job.offline()`

    Set job to Offline state.

- `job.output(output_blocks = True)`

  Print job information. If True print job blocks information too.

- `job.send()`

  Send job to Afanasy server.

4.12.1.3 Block Class

- Constructor:
  - `block = af.Block(block_name, service_name)`

    Construct a new block and return it.
- Variables:
  - `block.tasks = []`

    Tasks list. Used for not numeric blocks.

4.12.1.4 Task Object

- Constructor:
  - `task = af.Task(task_name)`

    Construct a new task and return it.

4.12.2 JSON Protocol

You can use `afcmd` CLI to send JSON objects (files) to Afanasy server:

- `afcmd json [file|pipe]`: Test JSON syntax, output an error and position.
- `afcmd v json [file|pipe]`: Same as previous and output parsed JSON document structure.
- `afcmd json send [file]`: Send JSON data after successfully parsed.

4.12.2.1 Job

Here is an example of a minimum JSON object to send to server to construct a job:

```
{
    "job":
    {
        "name"                  : "job name",
        "user_name"             : "jimmy",
        "host_name"             : "host",
        "blocks":[
        {
            "name"              : "Nuke",
            "tasks_name"        : "frames @#@-@#@",
            "service"           : "nuke",
            "parser"            : "nuke",
            "frame_first"       : 1,
            "frame_last"        : 100,
```

(continues on next page)

(continued from previous page)

```
            "frames_per_task"    : 10,
            "frames_inc"         : 2,
            "command"            : "nuke -F@#@,@#@ -x scene.nk -X Write1",
            "working_directory"  : "/home/jimmy/work",
            "files"              : ["folder/img_L.@#####@.jpg","folder/img_R.@#####@.jpg
↪"]
        }
        ]
    }
}
```

4.12.2.2 Get

Get request are used to get information from server.

Here are some examples:

- Get a list with all jobs:

```
{
    "get":
    {
        "type" : "jobs"
    }
}
```

- Get jobs list from users with specified ids:

```
{
    "get":
    {
        "type" : "jobs",
        "uids" : [1,2]
    }
}
```

- Get renders by host names pattern:

```
{
    "get":
    {
        "type" : "renders",
        "mask" : "farmhost.*"
    }
}
```

- Get users list with special ids:

```
{
    "get":
    {
        "type" : "users",
        "ids"  : [1,2]
    }
}
```

4.12.2.3 Actions

Actions are used to edit parameters and perform operations.

Any action should have `host_name` and `user_name` fields for logs.

Here are some examples:

- Set render nimby

```
{
    "action":
    {
        "user_name"   : "jimmy",
        "host_name"   : "pc01",
        "mask"        : "pc02",
        "type"        : "renders",
        "params"      :
        {
            "nimby"      : true
        }
    }
}
```

- Set user priority

```
{
    "action":
    {
        "user_name"   : "jimmy",
        "host_name"   : "pc01",
        "mask"        : "bob",
        "type"        : "users",
        "params"      :
        {
            "priority"   : 50
        }
    }
}
```

- Exit render

```
{
    "action":
    {
        "user_name"   : "jimmy",
        "host_name"   : "pc01",
        "mask"        : "pc02",
        "type"        : "renders",
        "operation"   :
        {
            "type"       : "exit"
        }
    }
}
```

- Delete job

```
{
    "action":
    {
        "user_name"   : "jimmy",
        "host_name"   : "pc01",
        "mask"        : "my3drender",
        "type"        : "jobs",
        "operation"   :
        {
            "type"       : "delete"
        }
    }
}
```

4.13 afcmd

`afcmd` is a command line interface (CLI) to Afanasy server and statistics SQL database.

`afcmd [command]` execute a command.

`afcmd` without any arguments to see output of afanasy environment initialization.

`afcmd h` display help - list all commands.

`afcmd h [command]` display help for specified command.

`afcmd v [other arguments]` put program in verbose mode.

4.13.1 afcmd cload

Ask Afanasy server to reread config files. It can be reconfigured without restart.

4.13.2 afcmd db_check

Check database connection.

4.13.3 afcmd db_reset_all

Drop tables in Afanasy database if any, and create new. This command should be executed before first Afanasy server start to create needed tables for statistics (if you need any).

CHAPTER 5

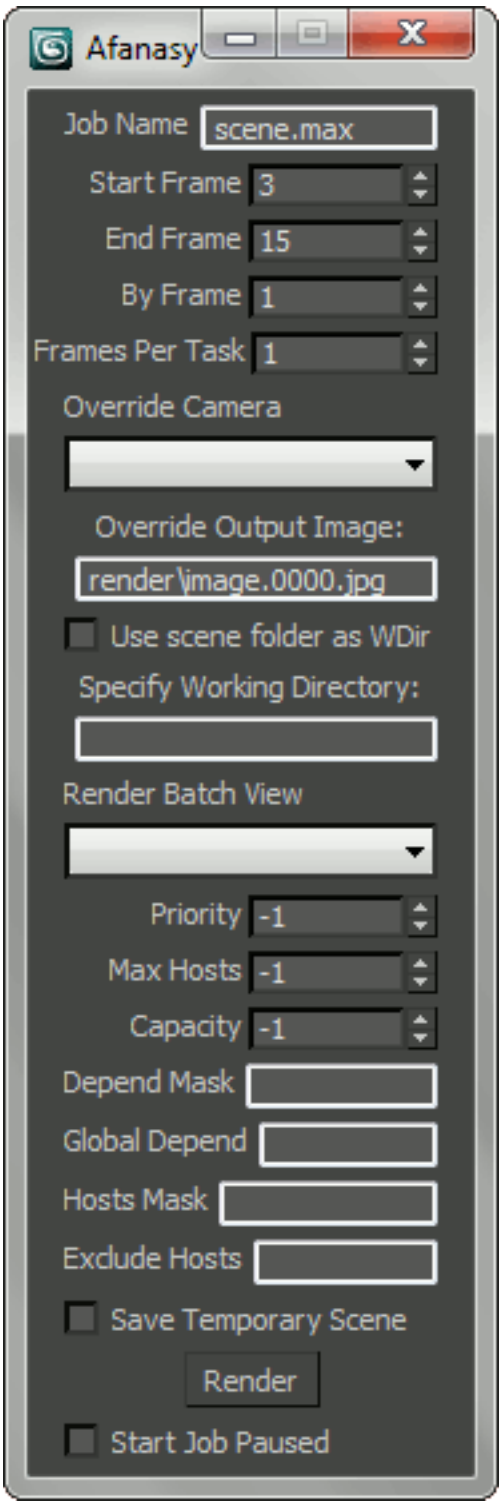
Software Integration

5.1 3D Studio Max

CGRU creates a menu in main window menus panel. Menu creation script file is `cgru\plugins\max\startup.ms`. To setup MAX to launch this script automatically you can add this folder in PATH. On every start MAX searches PATH for `startup.ms` scripts and launches them.

5.1.1 Submission Dialog

CGRU -> Afanasy...


Afanasy
Job Name
scene.max
Start Frame
3
End Frame
15
By Frame
1
Frames Per Task
1
Override Camera
Override Output Image:
render\image.0000.jpg
Use scene folder as WDir
Specify Working Directory:
Render Batch View
Priority
-1
Max Hosts
-1
Capacity
-1
Depend Mask
Global Depend
Hosts Mask
Exclude Hosts
Save Temporary Scene
Render
Start Job Paused

- **Job Name** Job name. Scene name by default.
- **Start Frame** First frame to render.
- **End Frame** Last frame to render.
- **By Frame** Render every Nth frame.
- **Frames Per Task** Number of frames in one task.
- **Override Camera** Select camera to render.
- **Override Output Image** Specify output image.
- **Use Scene Folder As WDir** Use scene file folder as render process working directory. 3D Studio Max, its plugins sometimes change working directory. With this option it will be always the scene folder.
- **Specify Working Directory** Set custom working directory.
- **Render Batch View** Select batch view to render.
- **Priority** Job order in user job list, '-1' - use default priority.
- **Max Hosts** Maximum number of hosts job can capture (running tasks limit), '-1' no limit.
- **Capacity** Job tasks capacity, '-1' - use default value.
- **Depend Mask** Wait same user jobs names pattern.
- **Global Depend** Wait any user jobs names pattern.
- **Hosts Mask** Job can run only on hosts which names match this pattern.
- **Exclude Hosts** Job can not run on hosts which names match this pattern.
- **Save Temporary Scene** Copy scene to temporary file to render. It allows user to continue working with original file.
- **Start Job Paused** Send job in offline state.

5.2 Adobe After Effects

5.2.1 Installation

- Put a script from CGRU plugins folder

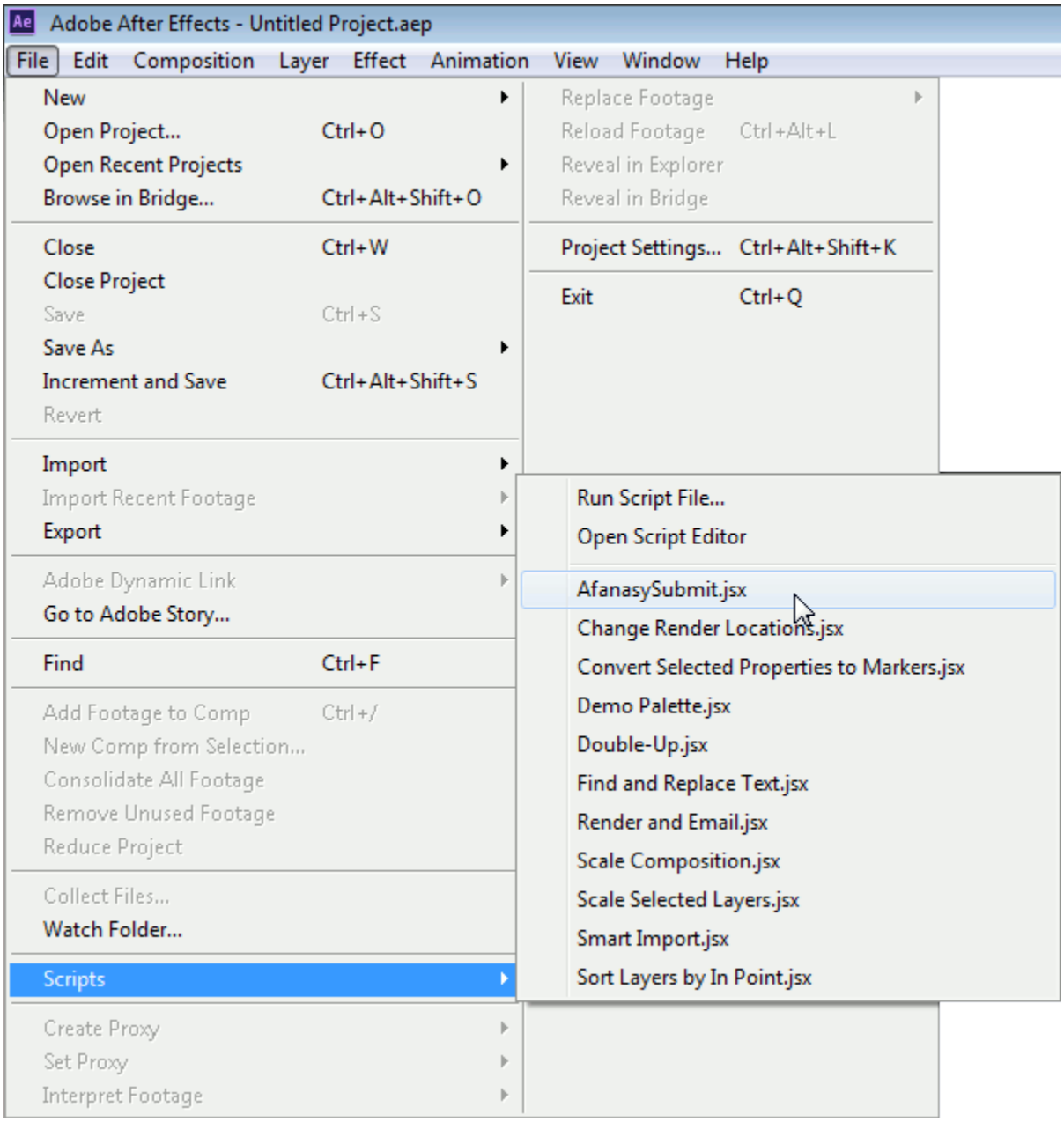
  `C:\cgru\plugins\afterfx\Afanasy.py`

  to AfterFX scripts folder

  `C:\Program Files\Adobe\Adobe After Effects CS6\Support Files\Scripts`

- Allow script to connect network

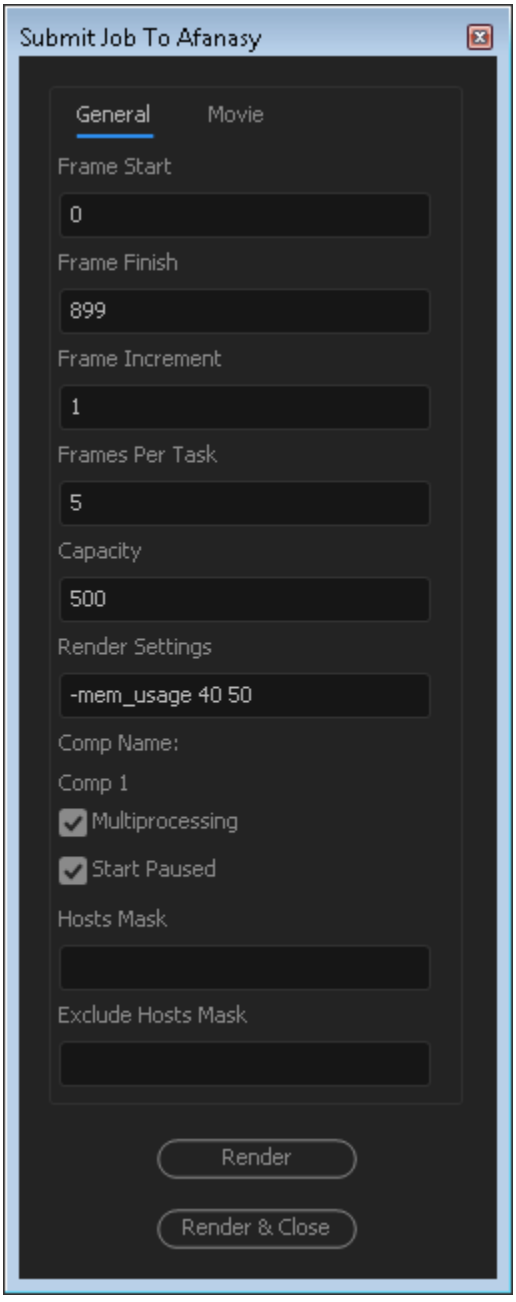
  Preferences - General - Allow scripts to Write Files and Access Network



5.2.2 Tool Dialog

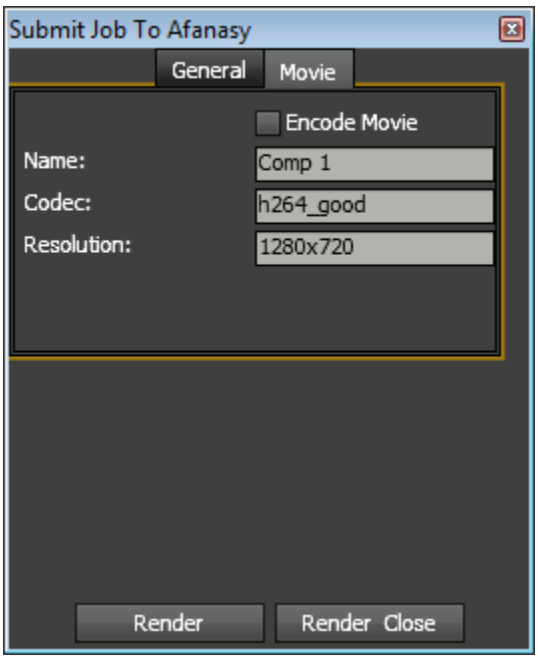
5.2.2.1 General Tab

Here are general job parameters.



5.2.2.2 Movie Tab

Here you can ask Afanasy to create a small preview movie.



5.2.3 Watch Job

This is how a job will be displayed in the Watch GUI.

5.2.4 Shared Script Location

You can put a CGRU plugin script to some shared (network) location. In this case you should create a script:

`C:\Program Files\Adobe\Adobe After Effects CS6\Support Files\Scripts\Afanasy.jsx`

```
var scriptFile = new File("\\\\server\\share\\cgru\\plugins\\afterfx\\Afanasy.jsx");
scriptFile.open();
eval(scriptFile.read());
scriptFile.close();
```

5.3 Blender

5.3.1 Setup

If you run Blender from CGRU Keeper, it addon will be added automatically.

You can setup CGRU addon manually. It is located in:

`/opt/cgru/plugins/blender`

Or in some custom location you have unpacked CGRU in.

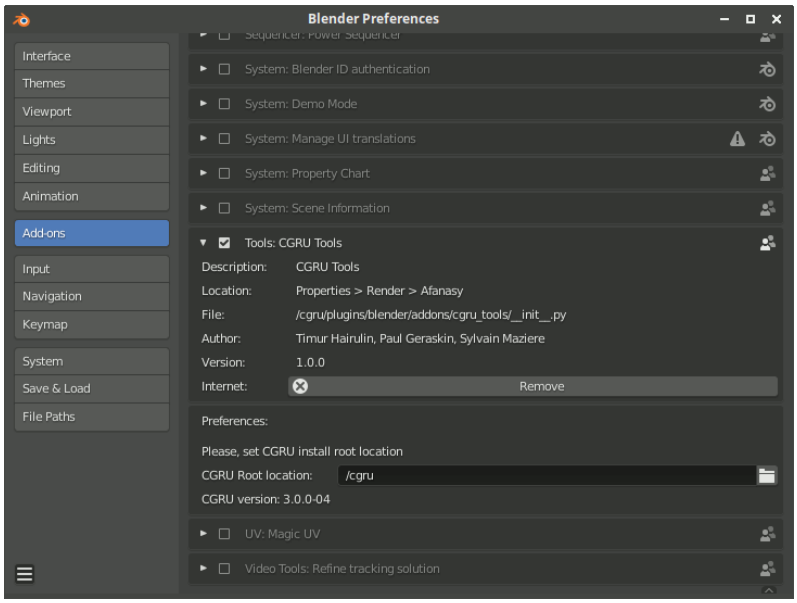


Fig. 1: Blender Preferences Window

If you run blender not from CGRU and installed addon manually, you should setup CGRU location for the addon. It needed as the addon uses other python scripts from CGRU.

5.3.2 Properties

- **Job Name** Afanasy job name. If empty scene name will be used.
- **File Path** If not empty, override output images to render.

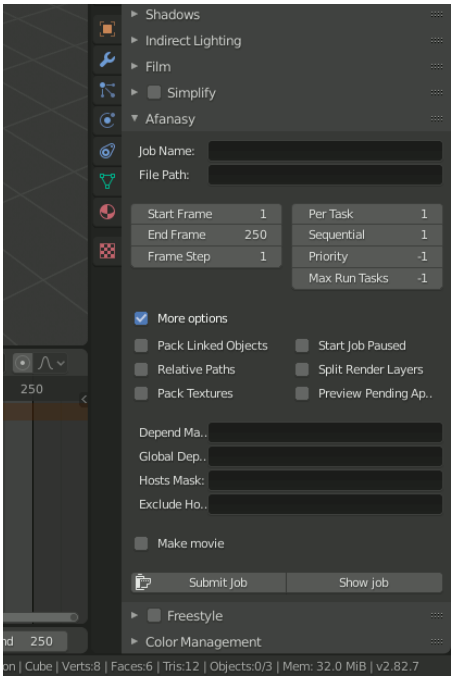


Fig. 2: Blender Afanasy Properties

- **Start Job Paused** Start job paused (send in off-line state).
- **Split Render Layers** Render different layers in a separate job blocks. Warning: this option disable post-processing passes (compositing nor seqeuncer are execute)
- **Pack Linked Objects** Make local all linked groups and objects.
- **Pack Textures** Pack all textures into the blend file.
- **Start** First frame to render.
- **End** Last frame to render.
- **By** Frames "jump" or increment.
- **Per Task** Number of frames in each task.
- **Priority** Job priority (execution order), `-1` means default.
- **Max Run Tasks** Maximum number of running at the same time tasks, `-1` means no limit.
- **Depend Mask** Other job(s) name pattern to wait. Empty value means not wait any job.
- **Global Depend** Other job(s) pattern of any user to wait.
- **Hosts Mask** Hosts names pattern job tasks can run on. If empty job can run on any host.
- **Exclude Hosts** Hosts names pattern job tasks can not run on.
- **Submit Job** Construct a job and send it to server.

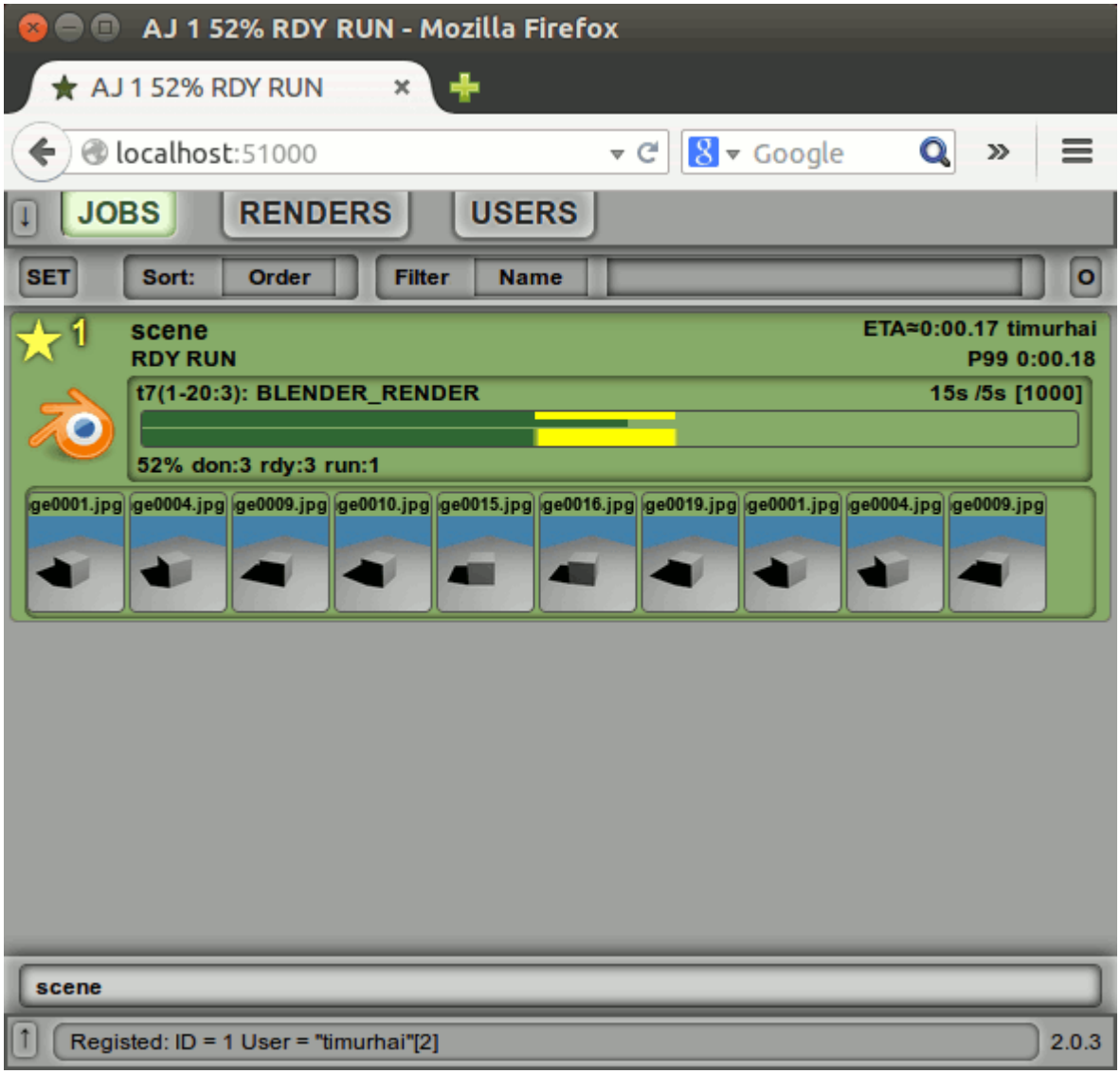


Fig. 3: Web GUI Jobs List.

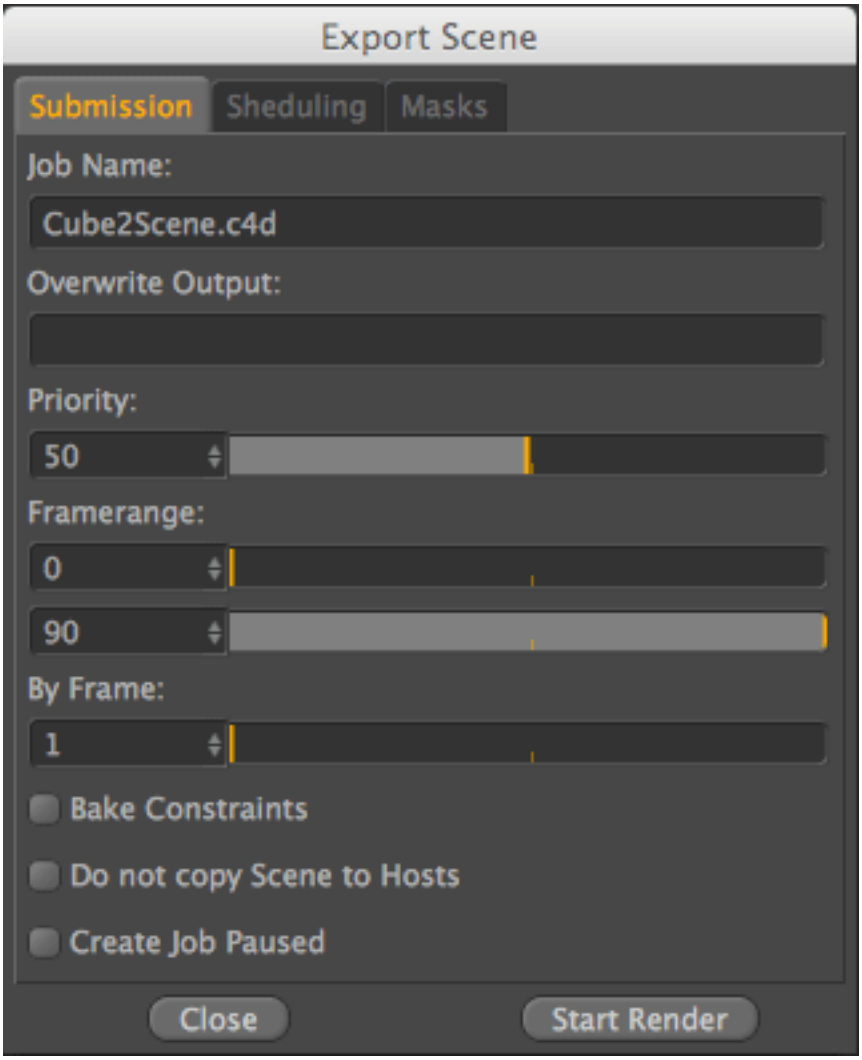
5.3.3 Job GUI

5.3.3.1 Job

5.3.3.2 Tasks

5.4 Cinema 4D

5.4.1 Afanasy Dialog



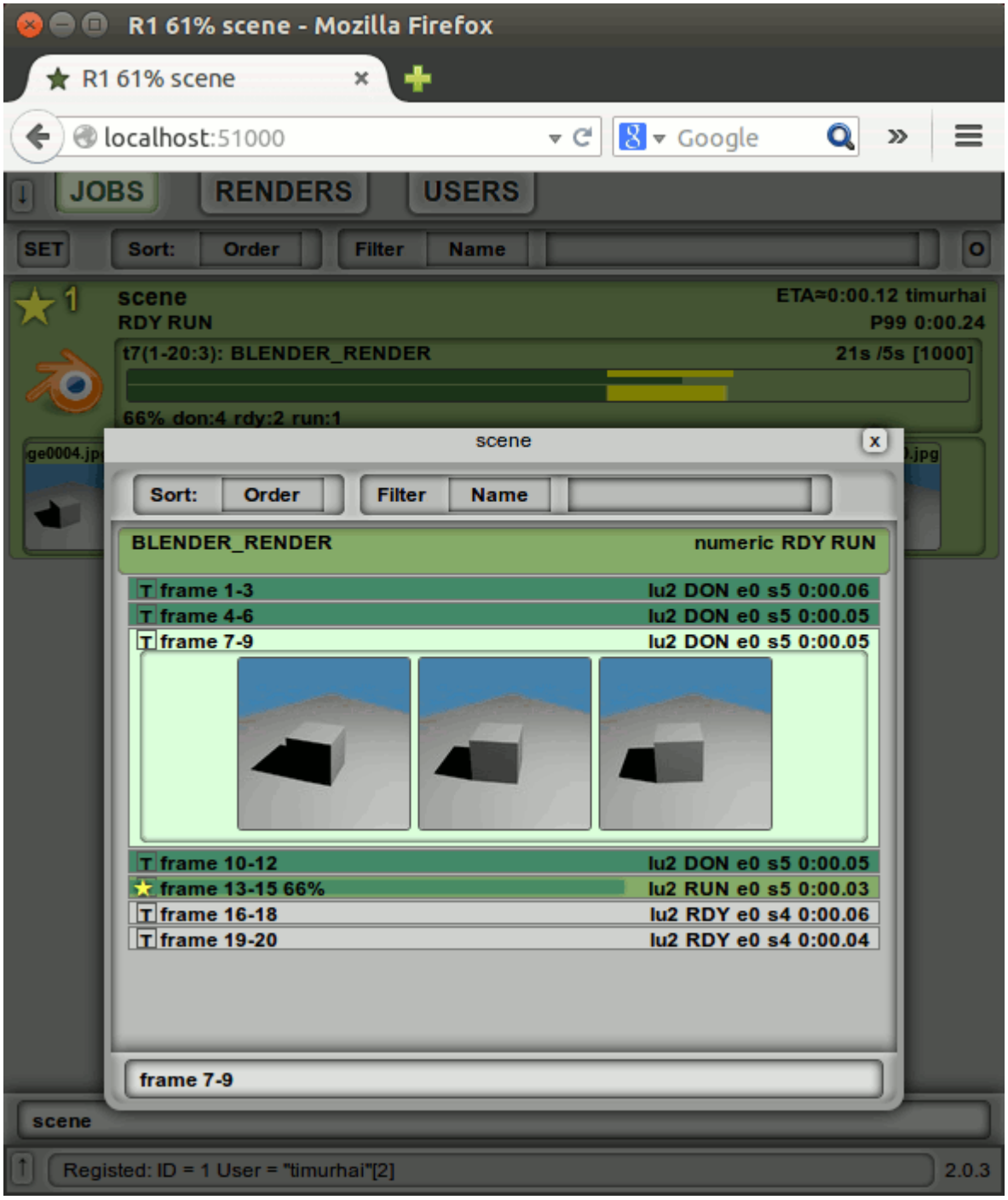
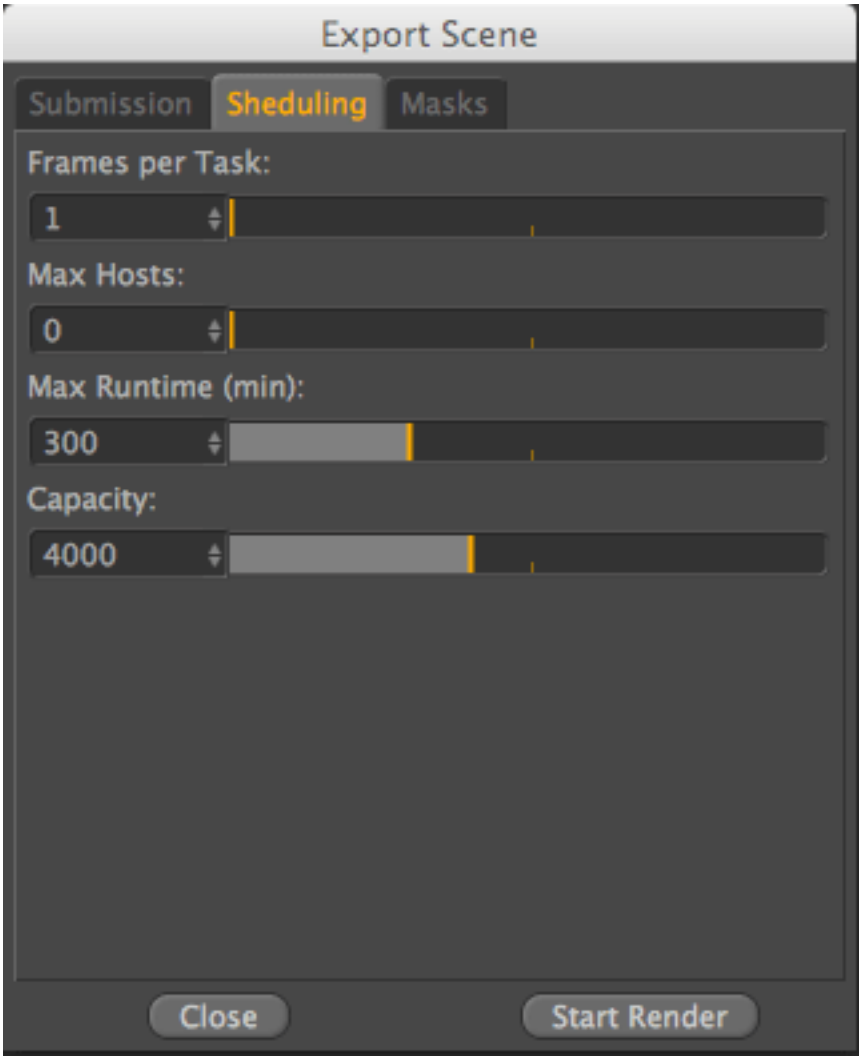
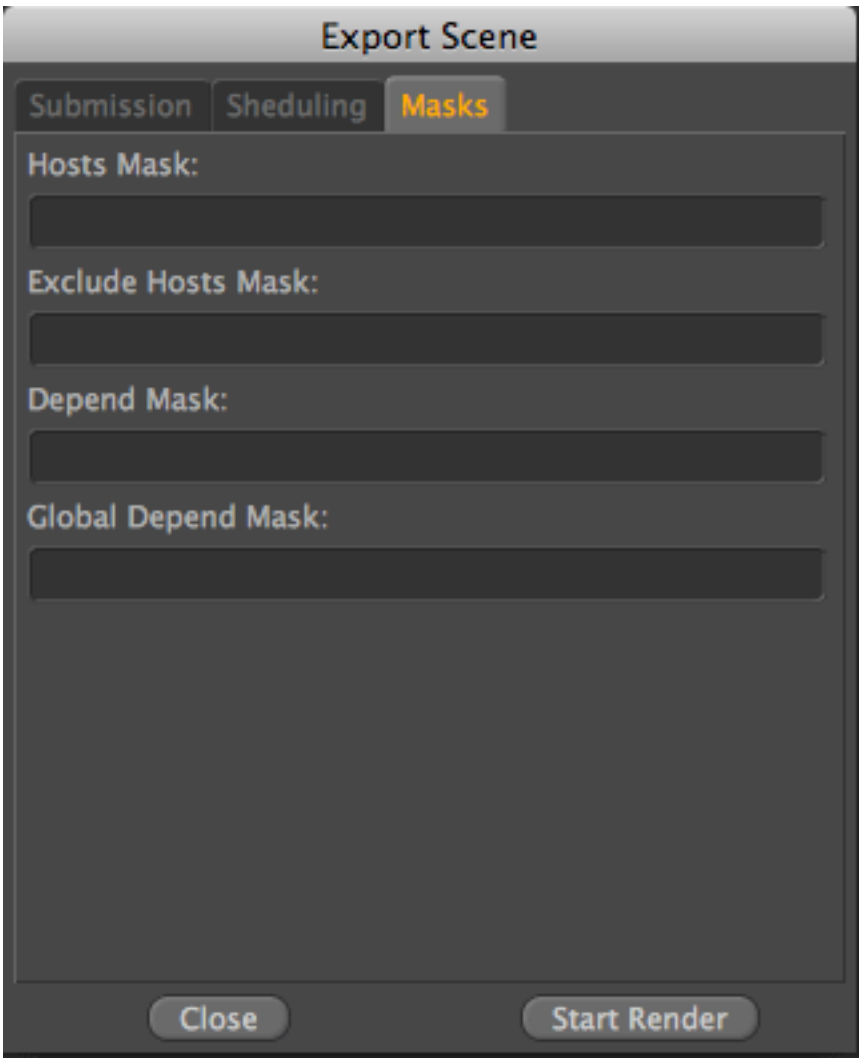


Fig. 4: Web GUI Jobs Tasks List.


Export Scene
Submission
Sheduling
Masks
Frames per Task:
1
Max Hosts:
0
Max Runtime (min):
300
Capacity:
4000
Close
Start Render



5.4.2 Submission

- **Job Name** Afanasy job name. Default is the scene-name
- **Overwrite Output** Overwrites the Render-Output-Path
- **Priority** Job order in user jobs list ('-1' means to keep this value default)
- **Framerange** Framerange to render on the farm
- **By Frame** Just renders every x frame. By default this is 1 so it renders every frame
- **Bake Constraints** Bakes the constraints and then saves a copy of the scene which gets rendered (currently it only bakes xxx)
- **Do not copy Scene to Hosts** By default it copies the scene and the textures locally to the farm and also renders the images locally. After it finished it copies then the finished renderings back to the server. If this is not wanted for some reason it can get deactivated here and it loads everything from the farm and saves directly to the farm
- **Create Job Paused** The submitted job is then paused and has to be started manually

5.4.3 Scheduling

- **Frames per Task** Number of frames in each task
- **Max Hosts** Maximum number of Hosts to use (0 means to keep the default value)
- **Max Runtime (min)** How long a frame is allowed to render maximum before it gets re-queued
- **Capacity** Tasks capacity value

5.4.4 Masks

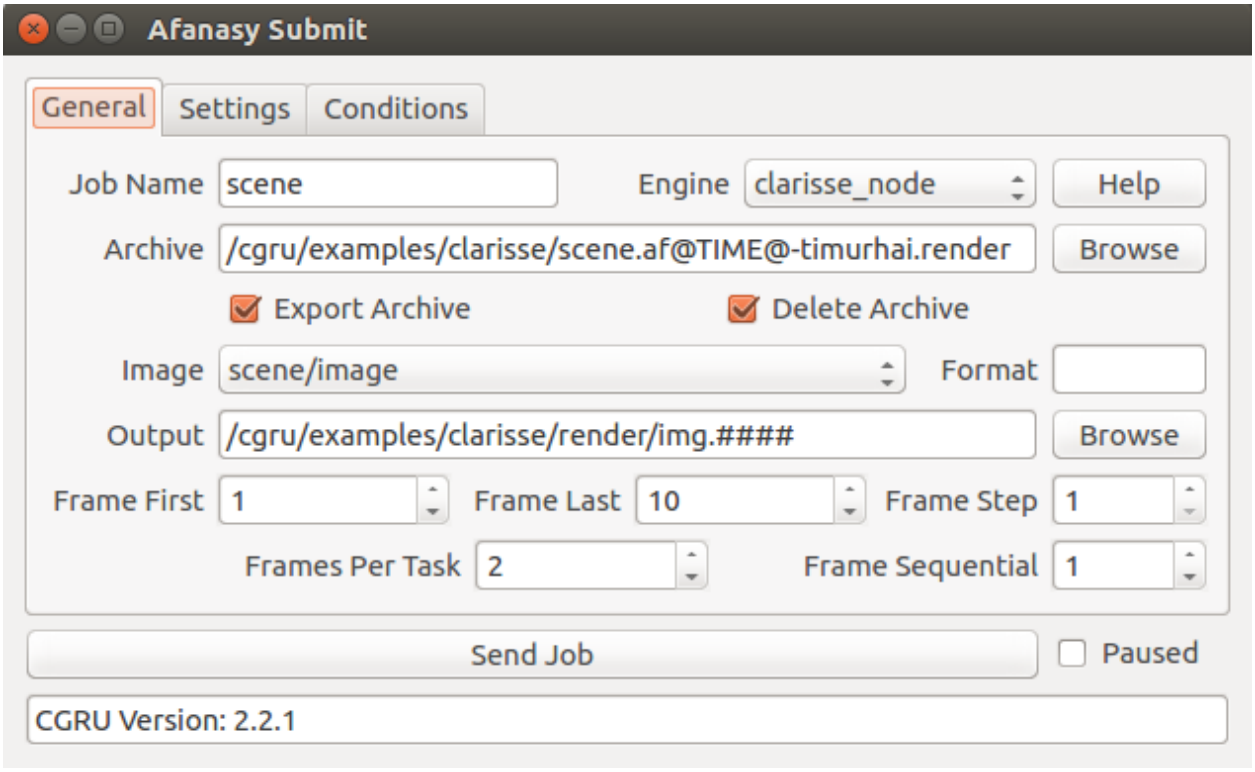
- **Hosts Mask** Hosts names pattern where job can run on (empty value means that job can run on host with any name)
- **Exclude Hosts Mask** Hosts names pattern where job can not run on
- **Depend Mask** Same user jobs names pattern to wait to be done to start (empty value means not to wait any job)
- **Global Depend Mask** Same as Depend Mask, but waits for a jobs from any user

5.5 Clarisse iFX

5.5.1 In-App Submission

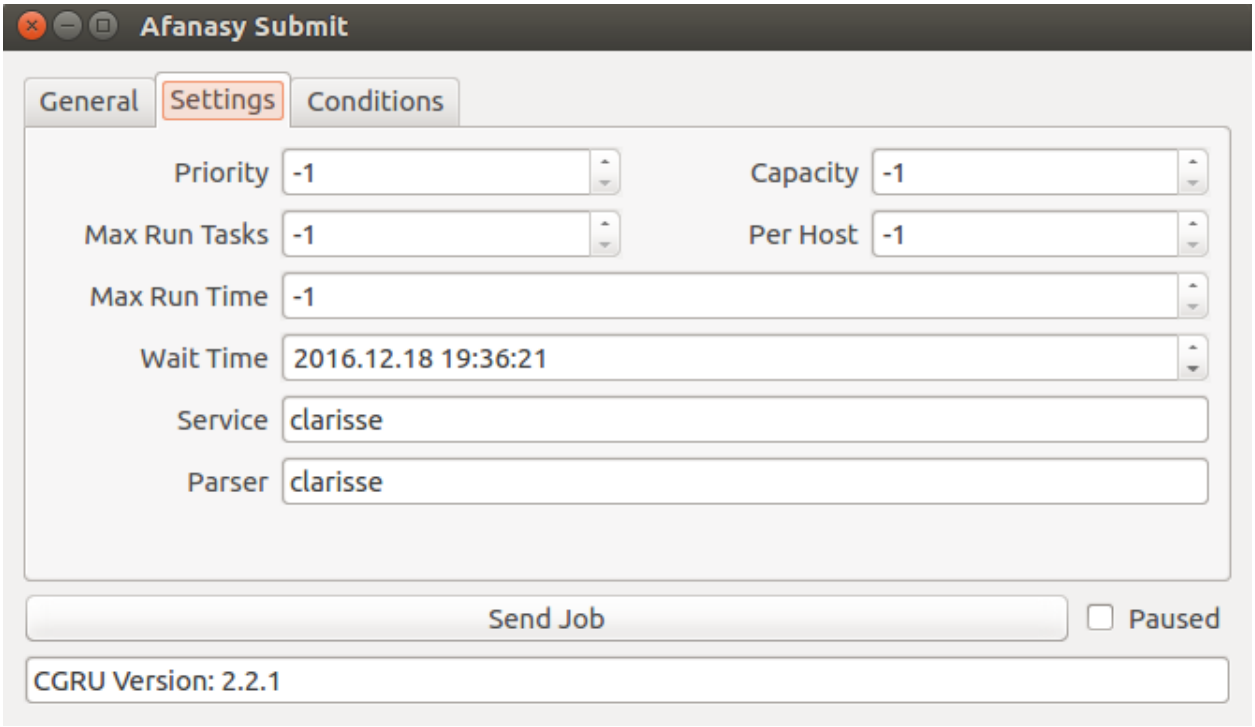
This a Python script. It get some project attributes and creates a dialog. In this dialog you can change parameters and send job to server. By default script exports project to render archive with some temporary name. And after render, when user deletes job, temporary archive will be deleted too.

5.5.1.1 General Tab



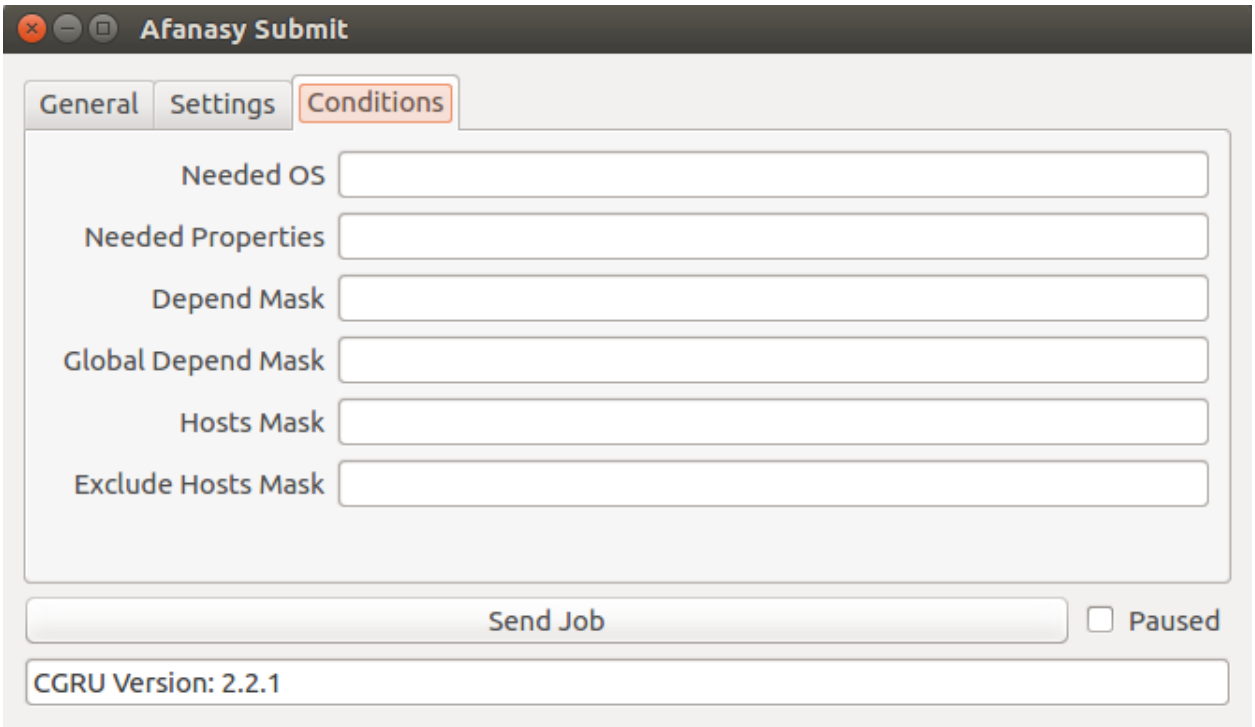
- **Engine**
  - **clarisse_node** CGRU command used to launch Clarisse cnode. By default it will launch the latest (alphabetically) version.
  - **clarisse_render** CGRU command used to launch Clarisse crender.
  - **cnode** System command will be searched in PATH environment.
  - **crender** System command will be searched in PATH environment.

5.5.1.2 Settings Tab



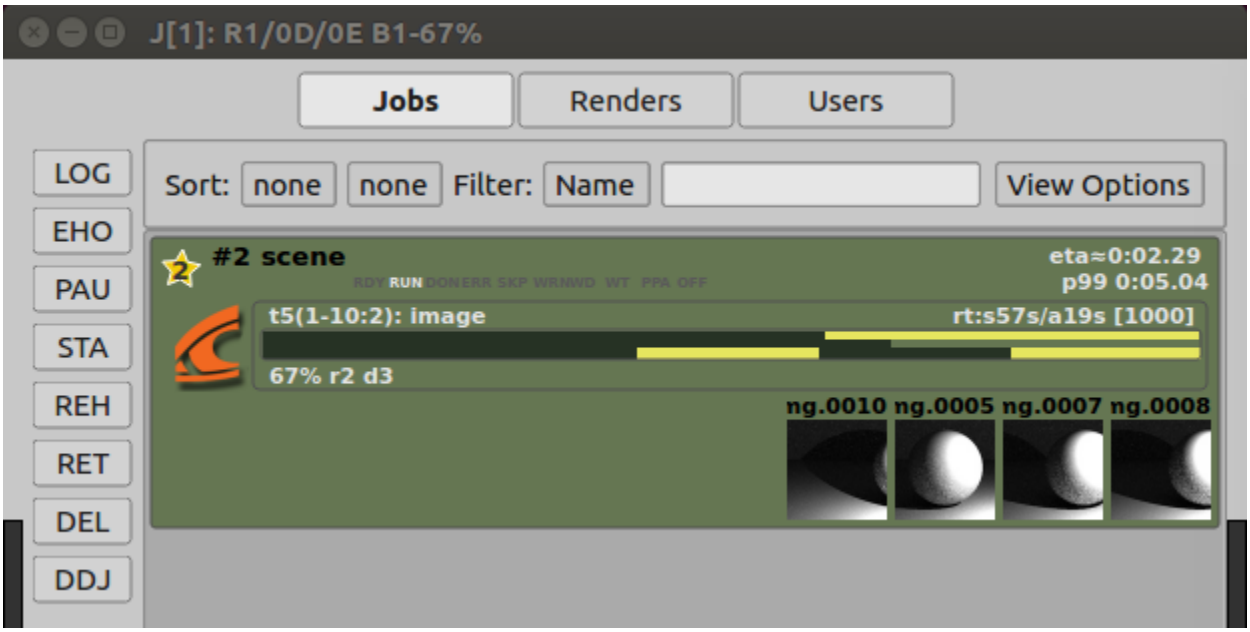
Negative value means use defaults.

5.5.1.3 Conditions Tab

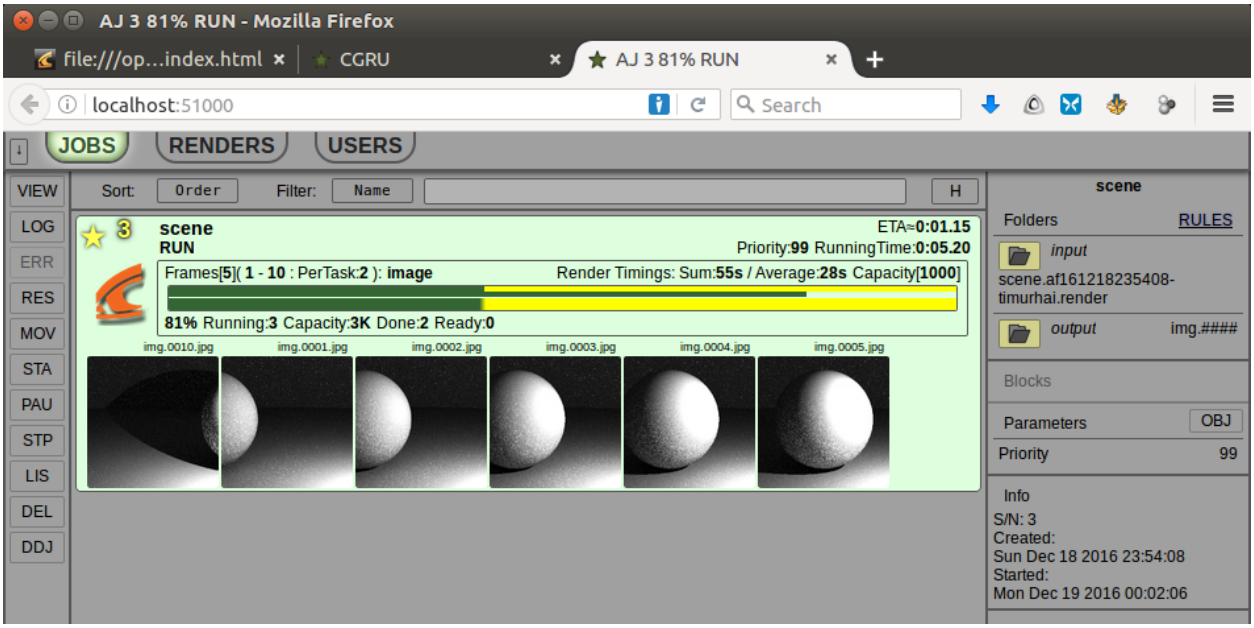


Empty field disables condition.

5.5.2 AfWatch

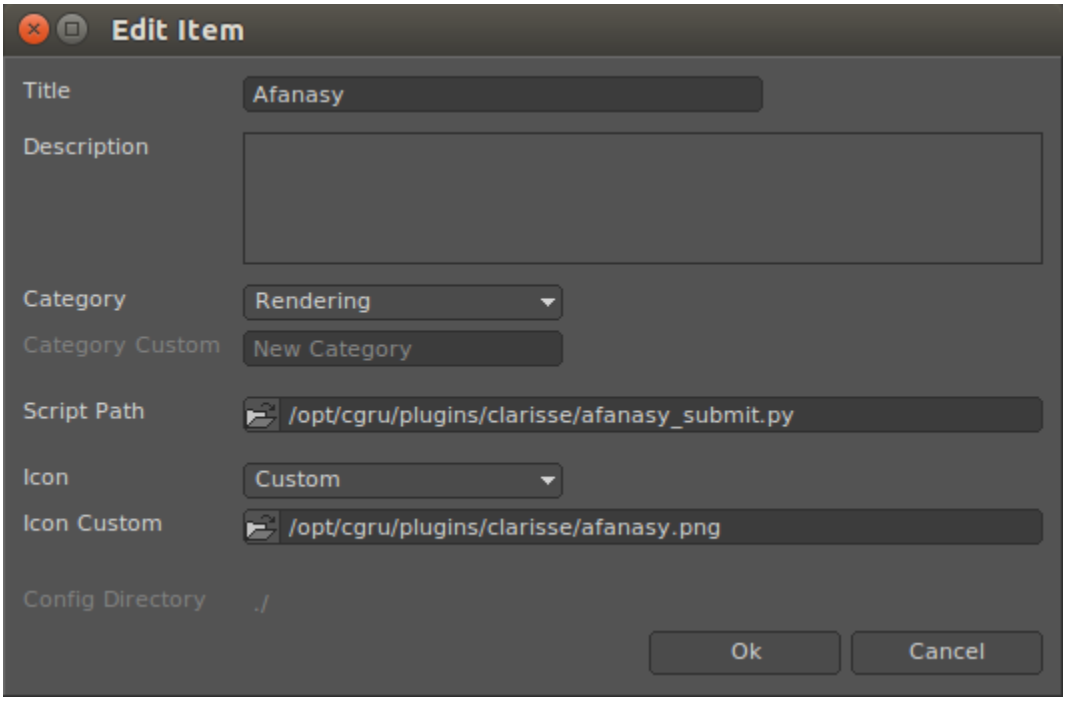


5.5.3 WebGUI



5.5.4 Setup

5.5.4.1 Shelf Item



5.5.5 AfStarter

You can also send Clarisse render archive to Afanasy with a stand-alone dialog AfStarter. You do not need to open main Clarisse application (GUI) for it.

5.5.6 Developers

In-app submission dialog GUI is created with PyQt (PySide). Qt binding in Python is represented by a Qt. It chooses existing Qt binding automatically.

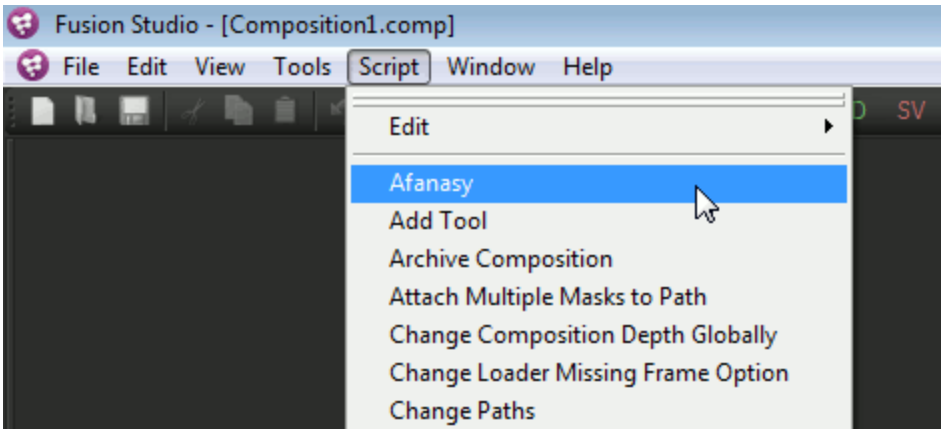
Submission script:

https://github.com/CGRU/cgru/blob/master/plugins/clarisse/afanasy_submit.py

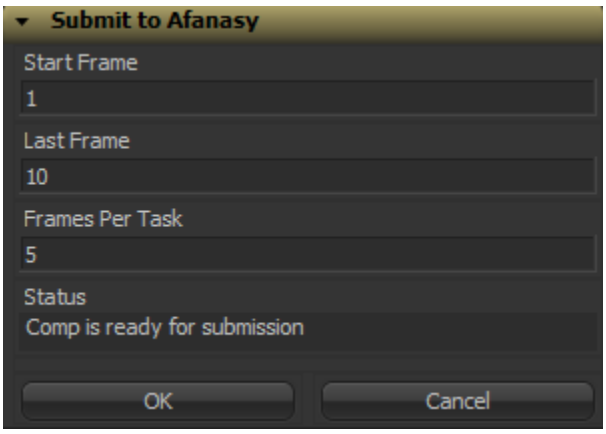
5.6 Fusion

5.6.1 Menu

There is a script that raises a dialog that constructs and send jobs to server.

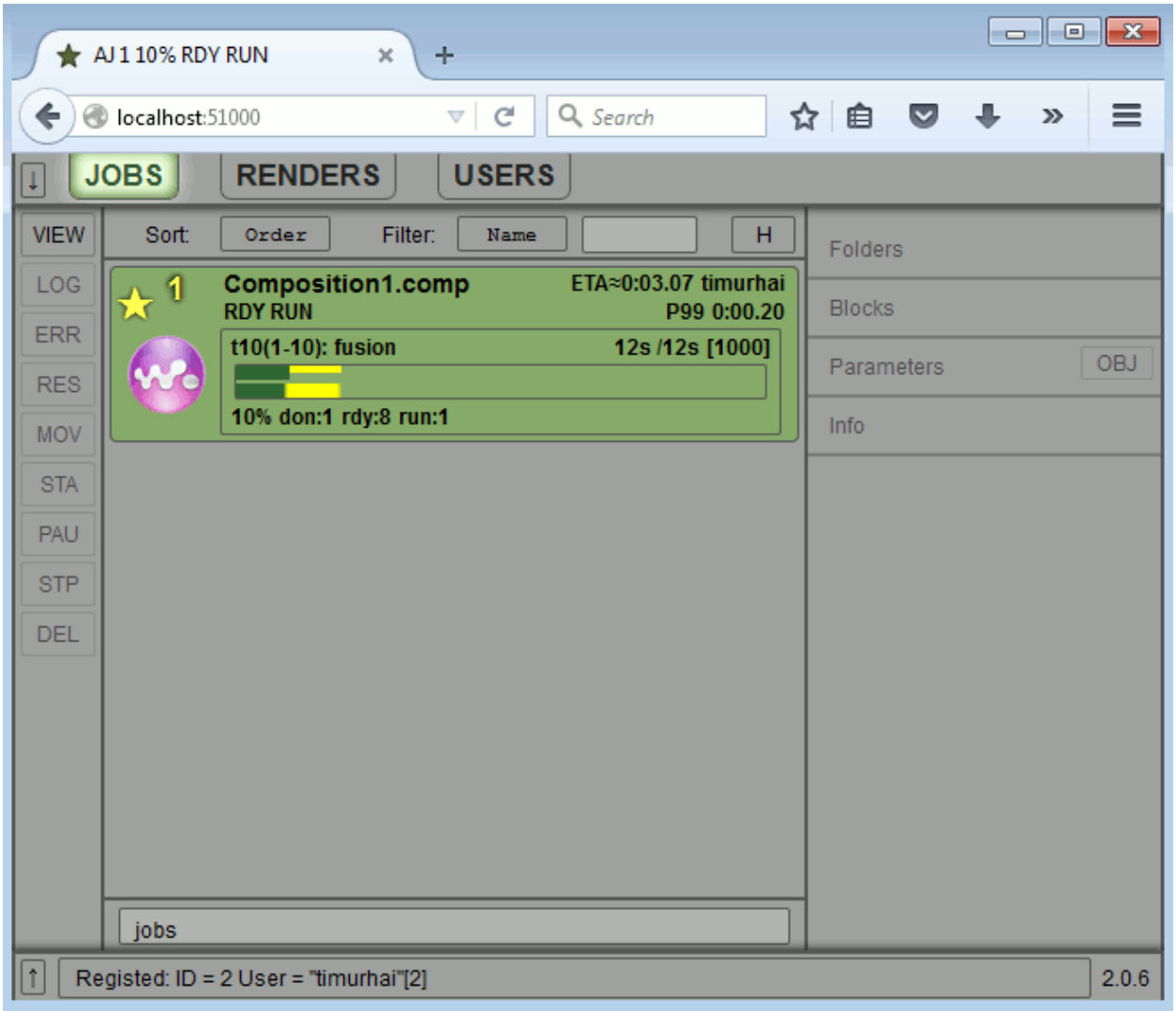


5.6.2 Dialog



- Start Frame
- First frame to render.
- Last Frame
- Last frame to render.
- Frames Per Task
- Number of frames in task.

5.6.3 Job GUI



5.6.4 Setup

You can just run Fusion from Keeper. This way menu item (script) will be added automatically by preferences manipulation.

To setup manually, you can copy script

```
cgru\plugins\fusion\Comp\Afanasy.eyeonscript
```

to Fusion scripts folder

```
C:\Program Files\Blackmagic Design\Fusion\Scripts\Comp
```

5.7 Houdini

Afanasy is represented by a special multi-functional ROP. You can connect several other ROP-s to Afanasy ROP to render. You can connect several Afanasy ROP-s to Afanasy ROP for a job with a complex dependencies.

5.7.1 Afanasy ROP

- **Submit** Generate a job and send it to server.
- **Start Paused** Send a job in off-line state.
- **Preview Approval** Set job *preview approval* flag. For example, if sequential is 10, it will render every 10 frame and wait for approve.

5.7.1.1 General

- **Job Name** Afanasy job name.
- **Output Driver** You can not (don't want) to connect Afanasy ROP to render ROP, you can specify it.
- **Valid Frame Range:**
  - **Render Any Frame** Use frame range form downstream node. Or render current frame if no range in network defined.
  - **Render Frame Range** Render this specified frame range.
  - **Render Frame Range Only (Strict)** Render this specified frame range. Other ROP-s will wait this whole frame range rendered.
- **Single Task**
  - Generate single task for whole frame range, useful for simulations.
- **Local Render** Render on the local render client. Job host mask will be automatically set to the local host name.
- **Frames Per Task** Number of frames in each task.
- **Sequential**

1	Render frames one by one from the first to the last
10	Render every 10 frame at first, than render last other frames
-1	Render frames backwards from the last to the first
-10	Render every 10 frame at first backwards, than render last other frames backwards
0	Render the first, the last, the middle, the middle of the middle and so on

- **Wait Time** Set job *time_wait* parameter to the current day *Hours* and *Minutes*. If current time will be greater than specified, the next day will be used.
- **Render With Take** Specify take to render.
- **Connected Nodes Are Independent** Allow run the same frames of all connected nodes at the same time.
- **Allow Sub-Task Dependence** Tasks can wait other tasks to be done partially. Useful for simulations. Frames render can start w/o waiting the whole simulation is finished.
- **Ignore ROP Inputs** Do not execute input ROP-s.

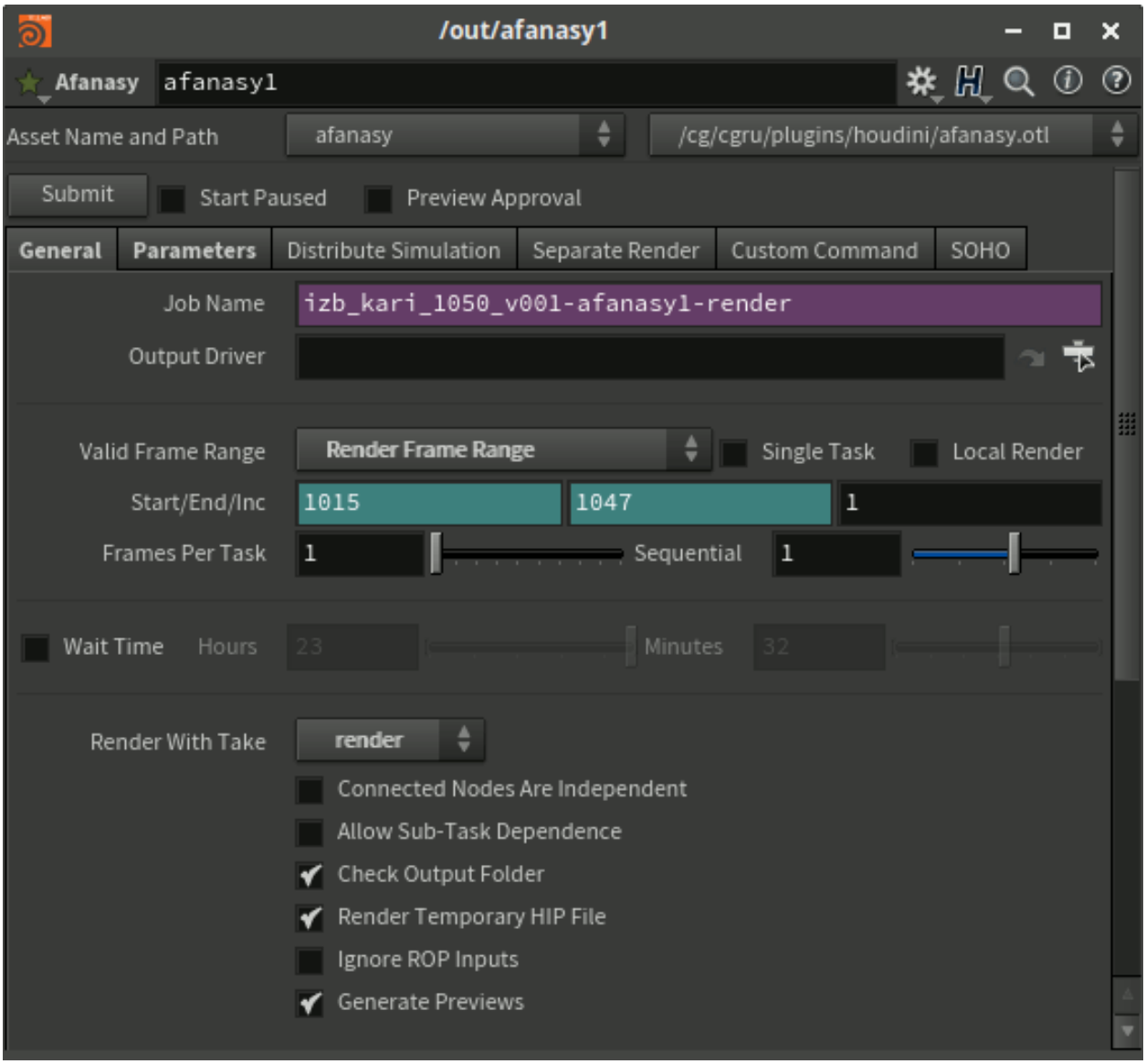


Fig. 5: Afanasy ROP General tab

5.7.1.2 Parameters

- **Platform**

  OS type the job can launch tasks on:

  - **Any**: any OS.
  - **Native**: the same as this OTL was launched on.

- **Tickets**

  Use tickets:

  - *Auto*: Submission script will try to set tickets automatically, depending on the ROP to render.
  - *Memory*: If not zero, this amount of *MEM* tickets will be set.
  - *Aux*: Any other tickets string as `TICKET:COUNT` comma separated list.

- **Pools** Use pools that are specified as `pool:priority` comma separated list.
- **Enable Extended Parameters** To switch ON/OFF it fast.
- **Job Branch** `$HIP` should be used in most cases. No matter how deep you placed *hip* file in file-system. It just help to find server an existing parent branch (department, project, scene).
- **Hosts Mask** Hosts names pattern where job can run on (empty value means that job can run on host with any name).
- **Exclude** Hosts Mask Exclude: Hosts names pattern where job can not to run on.
- **Depend Mask** Same user jobs names pattern to wait to be done to start
- **Global** Depend Mask Global: Same as Depend Mask, but waits for a jobs from any user.
- **Priority** Job order in user jobs list (`-1` means to use default value).
- **Maximum Running Tasks** Maximum tasks job can run at the same time (`-1` means no limit).
- **Per Host** Maximum Running Tasks Per Host: Maximum running tasks on the same host. (`-1` means no limit).
- **Capacity** Tasks capacity value (`-1` means use default value). Render must have enough free capacity to run it.
- **Render Time Min** Minimum time task should run (seconds). Sometimes tasks finishes with a good exit status too early.
- **Max** Task maximum running time (in hours). If task will not finish after this time, it will considered as an error and will be restarted.
- **Progress Timeout** If a task will not produce any output for this time (in hours), it will be considered as an error.
- **Min RAM** Minimum free memory (Gigabytes) should have render client to be able to start a task.
- **Override Service** This will be any custom service name for a job block tasks.
- **Parser** Override Parser: This will be any custom parser name for a job block tasks.
- **Life Time** *DONE* job will be automatically deleted after this time (in hours). Useful for some auxiliary jobs.
- **Files Check** Service (task instanced Python class) can check rendered files for existence. Submitter (script) should know file names that task should produce. Can not work on expressions/*takes*/overrides.
  - **Skip Existing** Render can check files for existence before run task command.

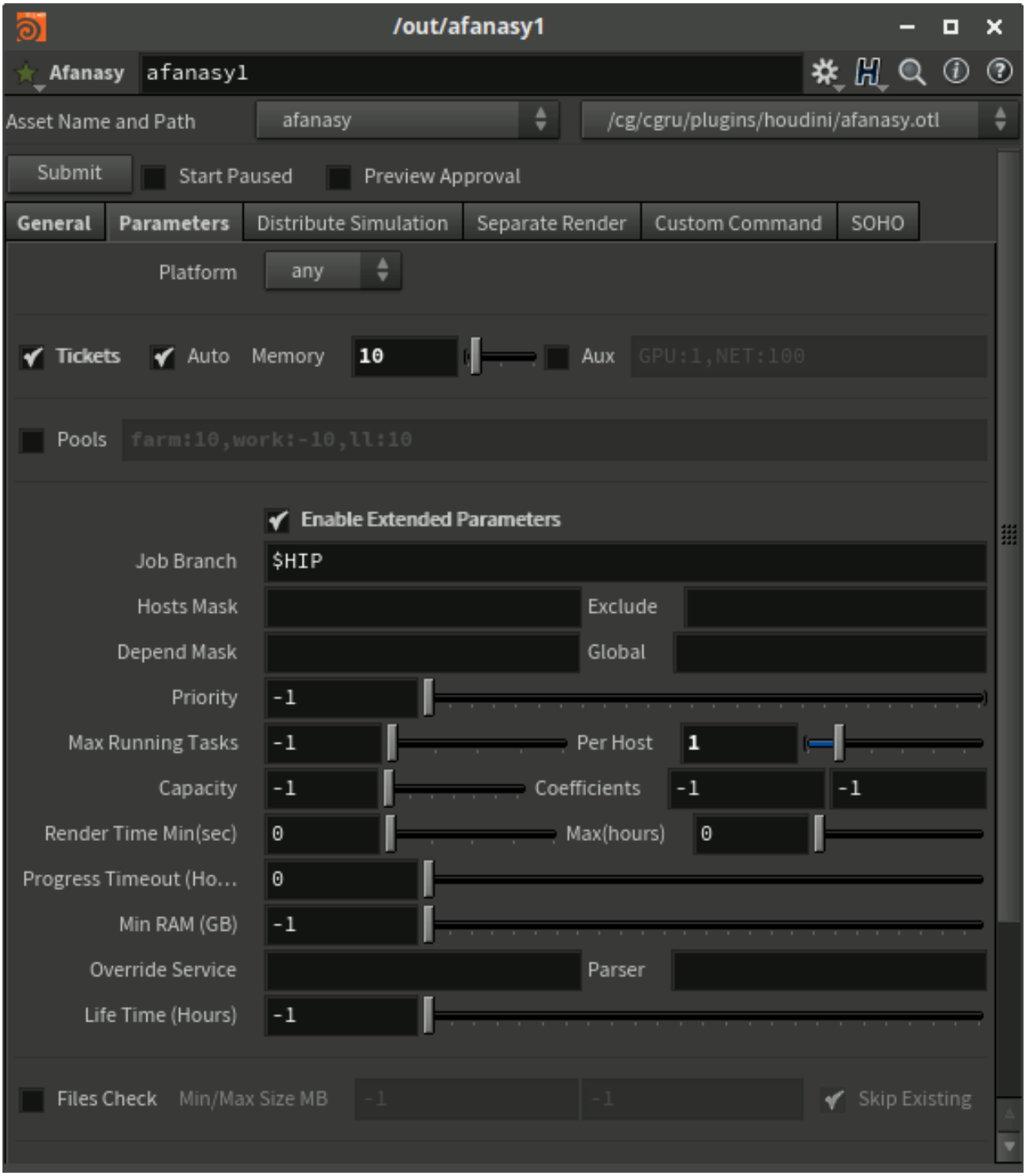


Fig. 6: Afanasy ROP Parameters tab

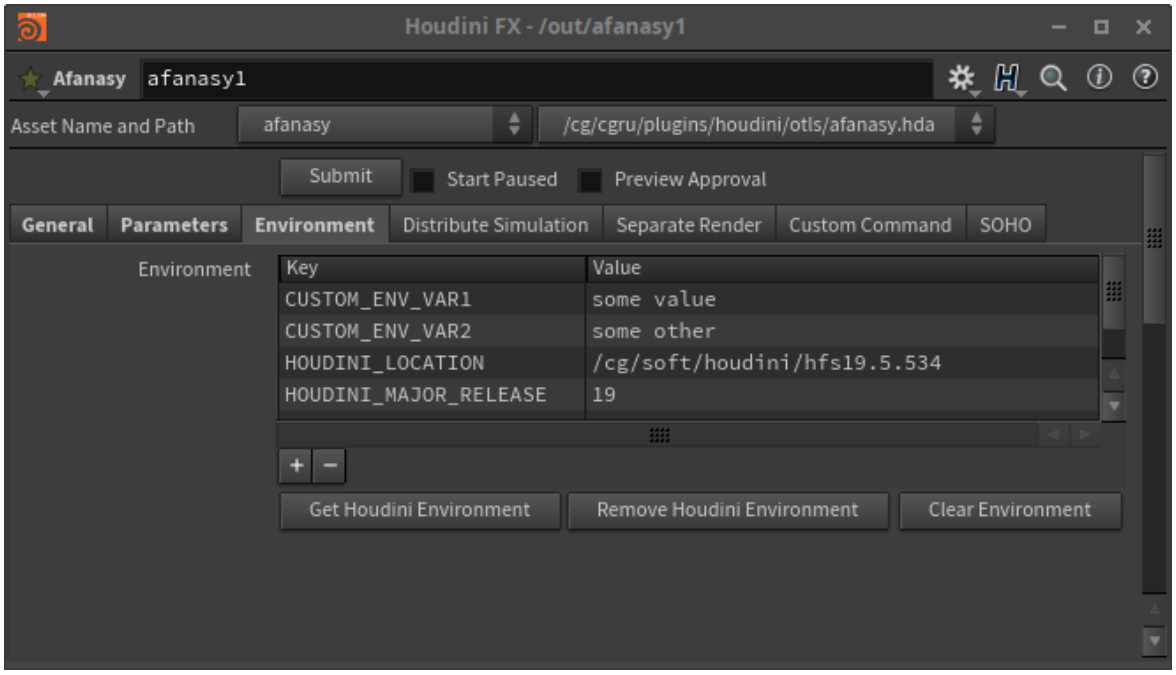


Fig. 7: Afanasy ROP Environment tab

5.7.1.3 Environment

- **Environment** It is key-value dictionary. Key used as an environment variable name. This environment variables will be *added* to task process environment.
- **Get Houdini Environment** Get and store in variables Houdini location and version. It can be used in setup scripts to launch the same version, initialize proper plug-ins.
- **Remove Houdini Environment** Remove Houdini related variables (starting with *"HOUDINI_"*).
- **Clear Environment** Remove all variables.

5.7.1.4 Distribute Simulation

- **Controls Node** Distributed simulation control node.
- **Number Of Slices** Distributed simulation slices number.
- Tracker Parameters

  Distributed simulation slices tasks should communicate via tracker service.

  - **Capacity** Tracker task capacity.
  - **Host Mask** Tracker will run only hosts that names match this regular expression.
  - **Service** Tracker task service.
  - **Parser** Tracker task parser.
  - **Manual Tracker** Use manual launched tracker service at specified **Address** and **Port**

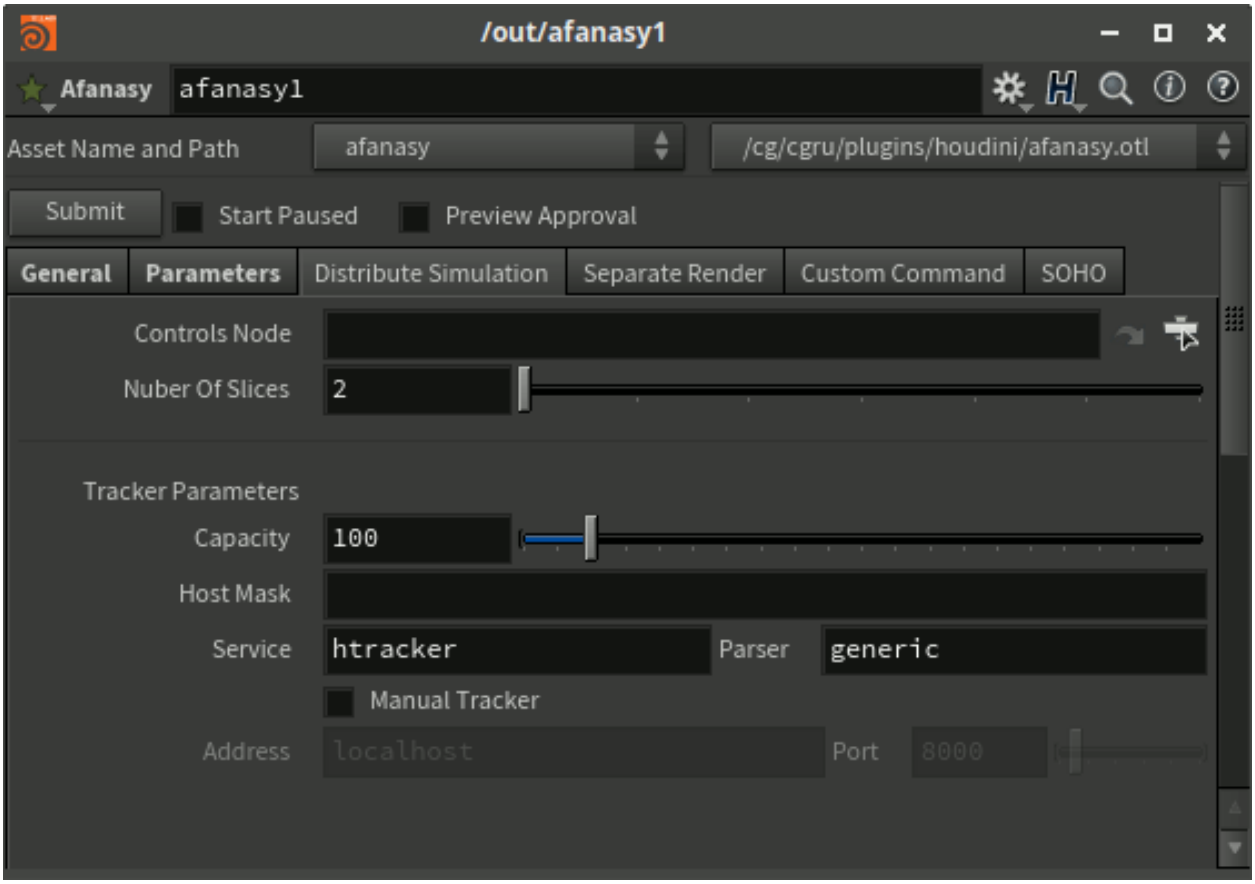


Fig. 8: Afanasy ROP Separate Render tab

5.7.1.5 Separate Render

Separate Render allows to separate render process on IFD files generation and render it by `mantra`. It can give several advantages on some *heavy* scenes.

Separate render generates a job that can:

- Render images locally in temporary folder and copy whole image after successful render. It can save your network traffic as render do not send small portions of an image during render process.
- Generate IFD file locally and then render it in separate process but in the same task. It can save render memory.
- Split one frame on tiles to render them simultaneously. So you can increase speed of one frame render. And also it can reduce memory needed to render a frame.
- Cleanup rendered IFD files and joined tiles images.

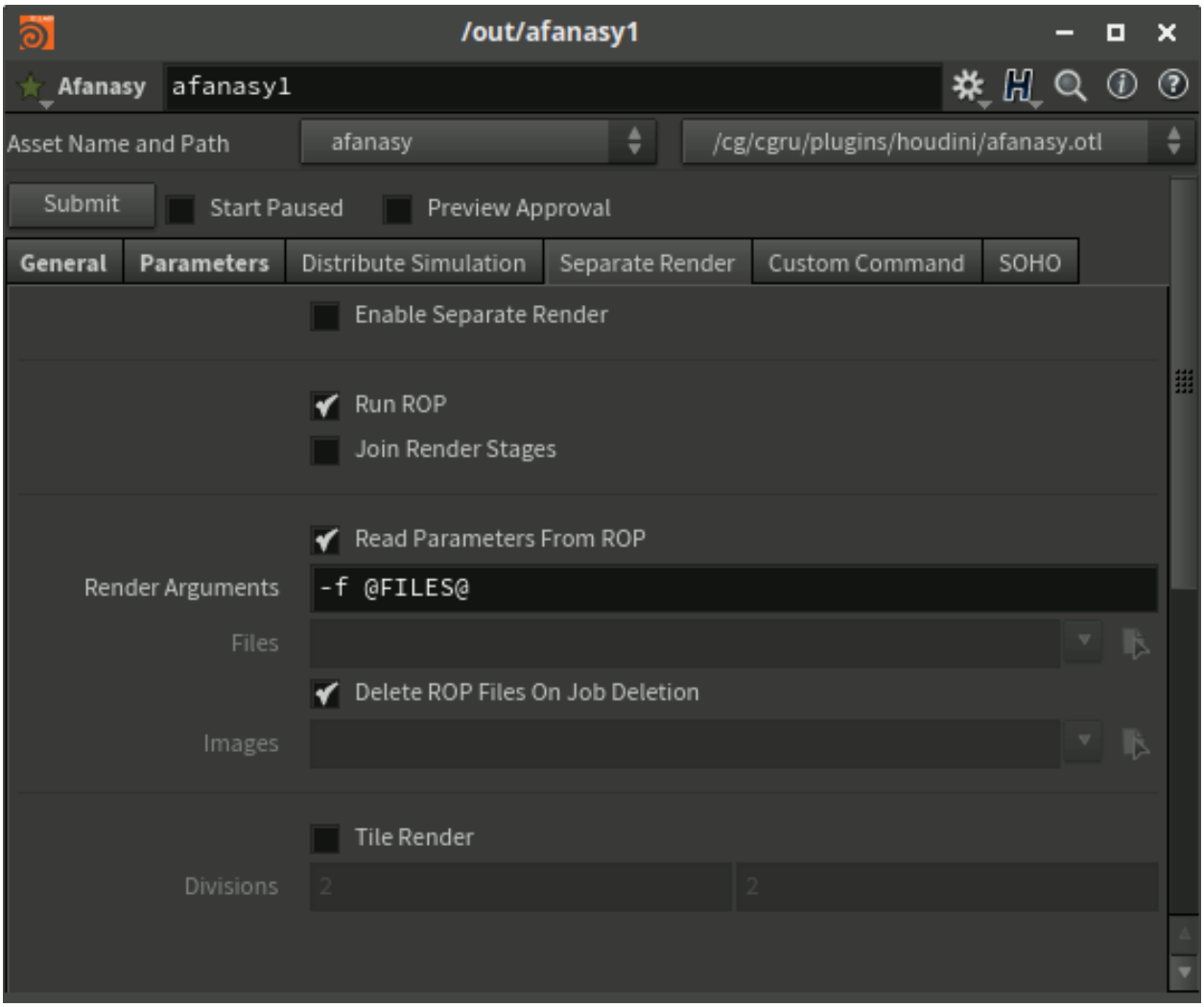


Fig. 9: Afanasy ROP Separate Render tab

- **Enable Separate Render** Turn this feature on.
- **Run ROP** Run ROP to generate files to render. Houdini will generate IFD files for mantra.

- **Join Render Stages** Generate IFD files and render in the same task. In this case IFD files will be generated to local temporary folder. It can save and memory usage and network traffic.
- **Read Parameters from ROP** Read files to generate and images to render parameters from specified ROP.
- **Render Arguments** Arguments for render command. Usually files and may be some other options.
- **Files** Files to generate.
- **Delete ROP Files On Job Deletion** ROP files (IFD-s) can be deleted when user will delete the job.
- **Images** Images which render will produce. Needed for tile render, AfWatch preview/thumbnails.
- **Tile Render** Enable rendering tiles and then combine them.
- **Divisions** Tiles divisions.

5.7.1.6 Custom Command

Run any custom command. For example you can render IFD files using `mantra` command, generate a preview movie with `ffmpeg`.

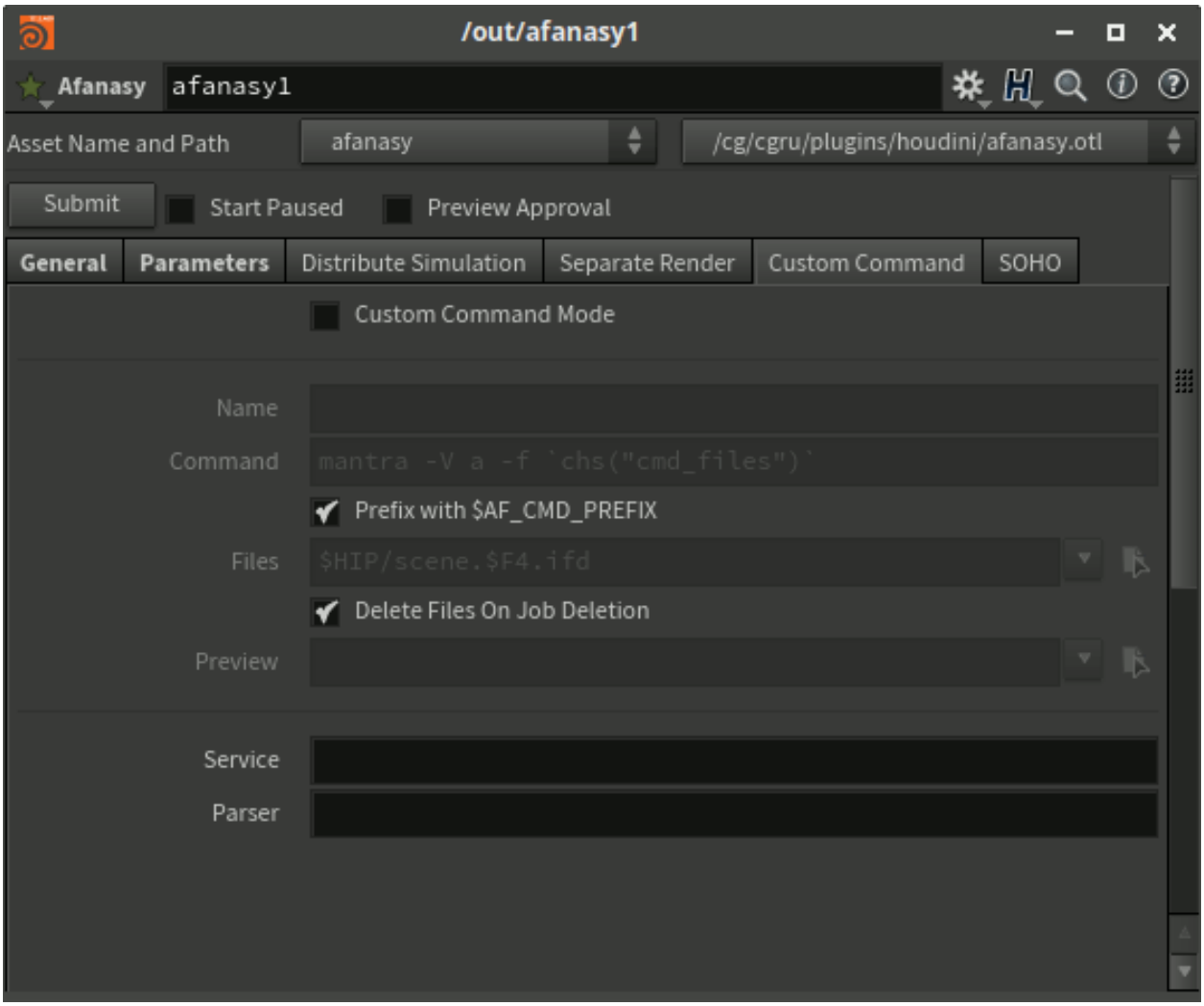


Fig. 10: Afanasy ROP Custom Command tab

- **Custom Command Mode** Add custom command tasks block to a job.
- **Name** Tasks block name. If empty the first word of the command will be used.
- **Command** The command.
- **Prefix with $AF_CMD_PREFIX** Add `$AF_CMD_PREFIX` environment variable value to the beginning of the command. This may be needed for some software (environment) setup.
- **Files** Some files you can point to use in command.
- **Delete Files On Job Deletion** Delete this files when user will delete job.
- **Preview** Specify result picture here to enable tasks preview.
- **Service** Tasks block service. If empty the first word of the command will be used.
- **Parser** Tasks block parser.

5.7.1.7 SOHO

This can be used to explain other ROP network what to do with Afanasy node.

Fig. 11: Afanasy ROP SOHO tab

- **Afanasy ROP** Specify Afanasy ROP to execute by SOHO.
- **Program** Script that will be executed on SOHO demand. That default script will execute *Submit* button on a specified Afanasy ROP.

5.7.1.8 ROP Examples

Simple

Just connect `afanasy` ROP to your render ROP.

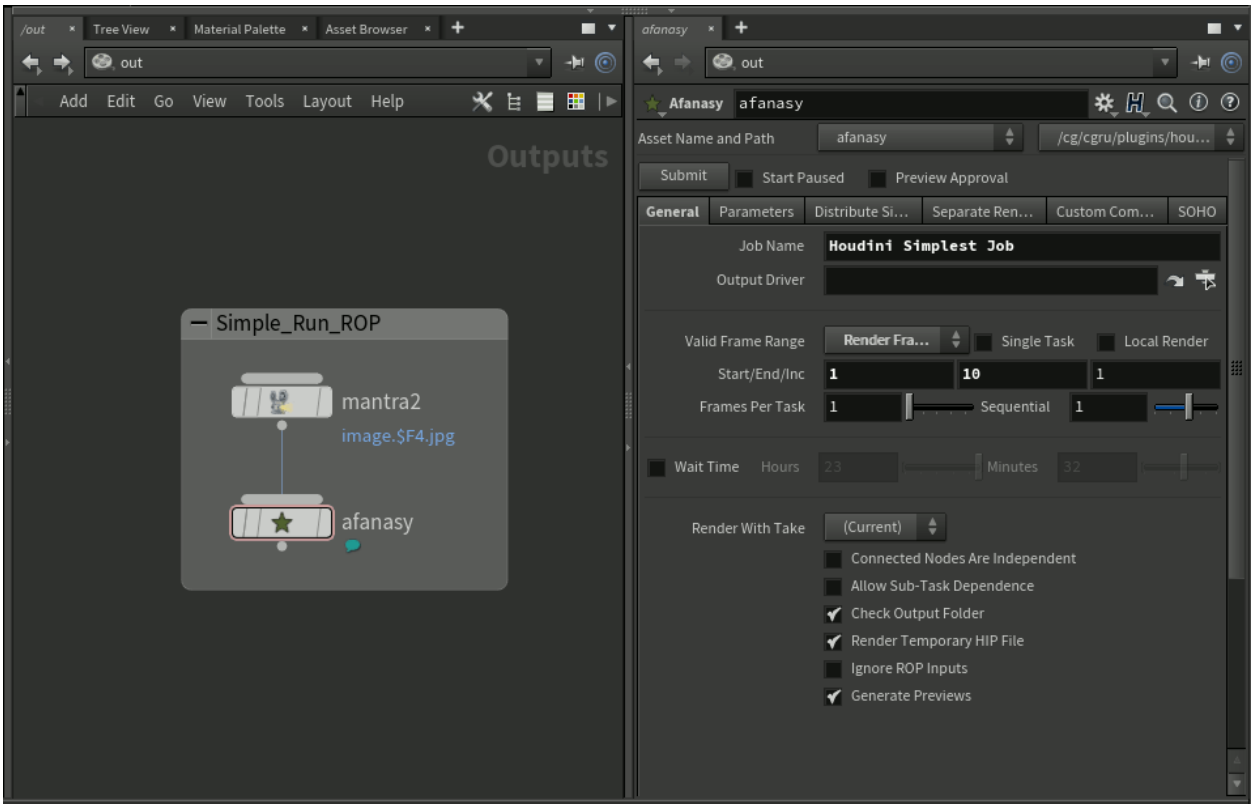


Fig. 12: Simple Network

The job consists of single tasks block. Each task represents a frame or several number of frames, specified in *Frames Per Task* parameter.

Command Render

You can send any custom command to your farm. Usually you need separate IFD files generation and run `mantra` as a standalone process to render.

This job consists of two blocks of tasks. The first block produced by `mantra_ifd` node, with `Disk File` parameter turned on. Next block runs `mantra` with `files` parameter pointing to the generated files.

Tile Render

You can split single image to render on several hosts. Each host (task) will produce a *tile* - some part of an image. Tiles will be combined in a single image.

Tile job consists of three blocks:

- **Generate** Generate IFD files.
- **Render** Render tiles with `mantra` standalone process.
- **Join** Join tiles to assemble an image. If tiles were successfully joined they will be removed. At the end of this stage, IFD will be removed, if it was asked.

Houdini native `itilestitch` tool is used to join tiles.

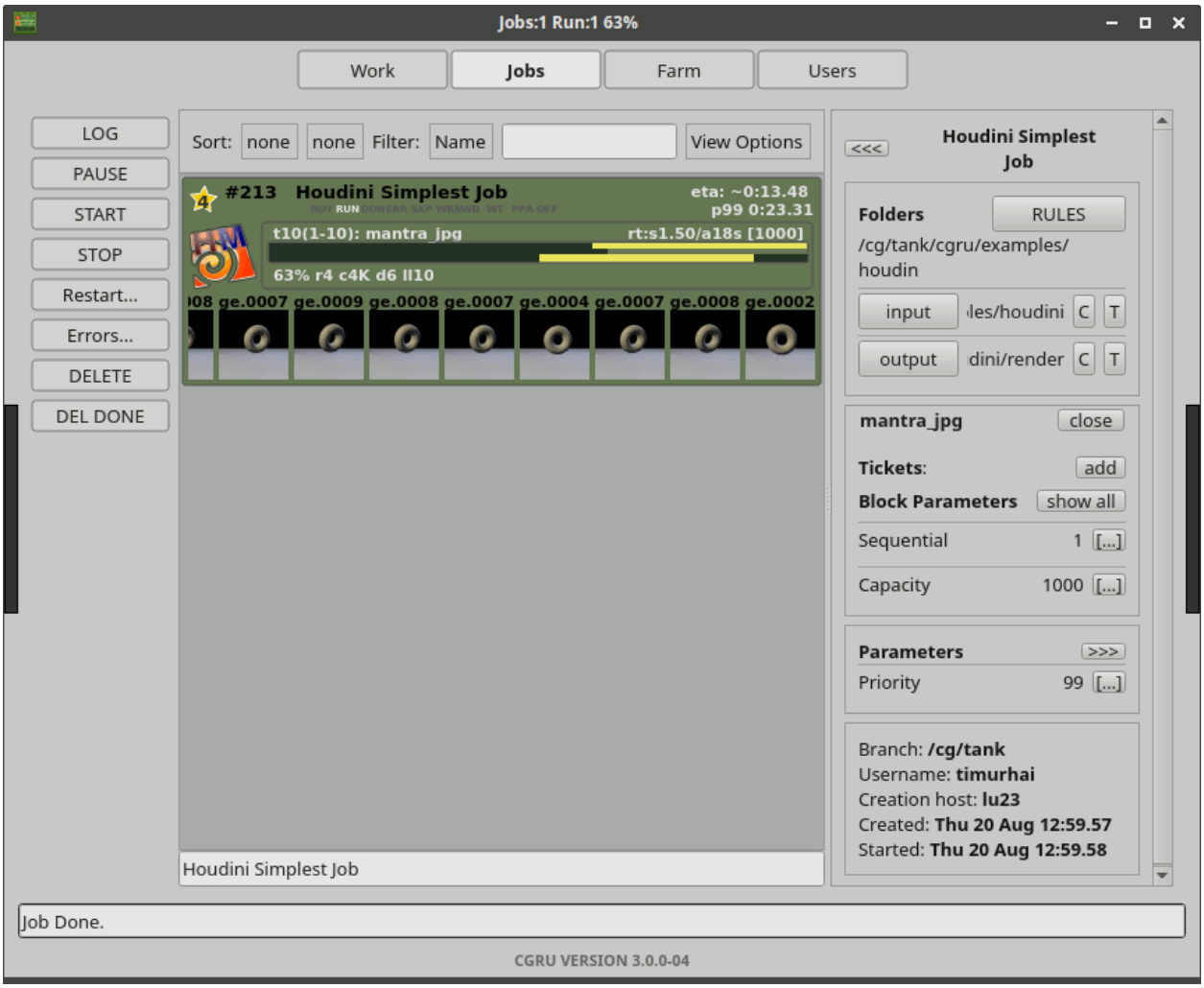


Fig. 13: Simple Job

Fig. 14: Simple Job Tasks

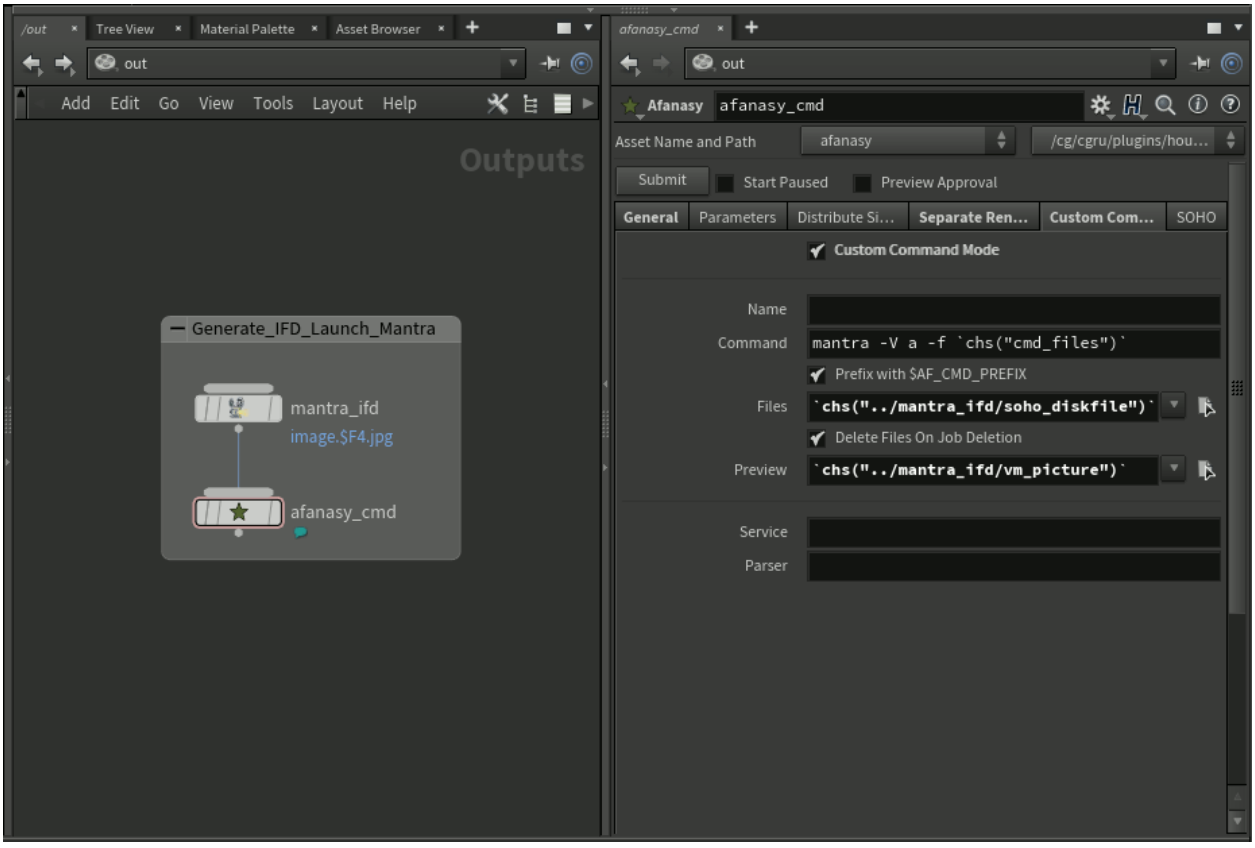


Fig. 15: Command Network

Fig. 16: Command Job

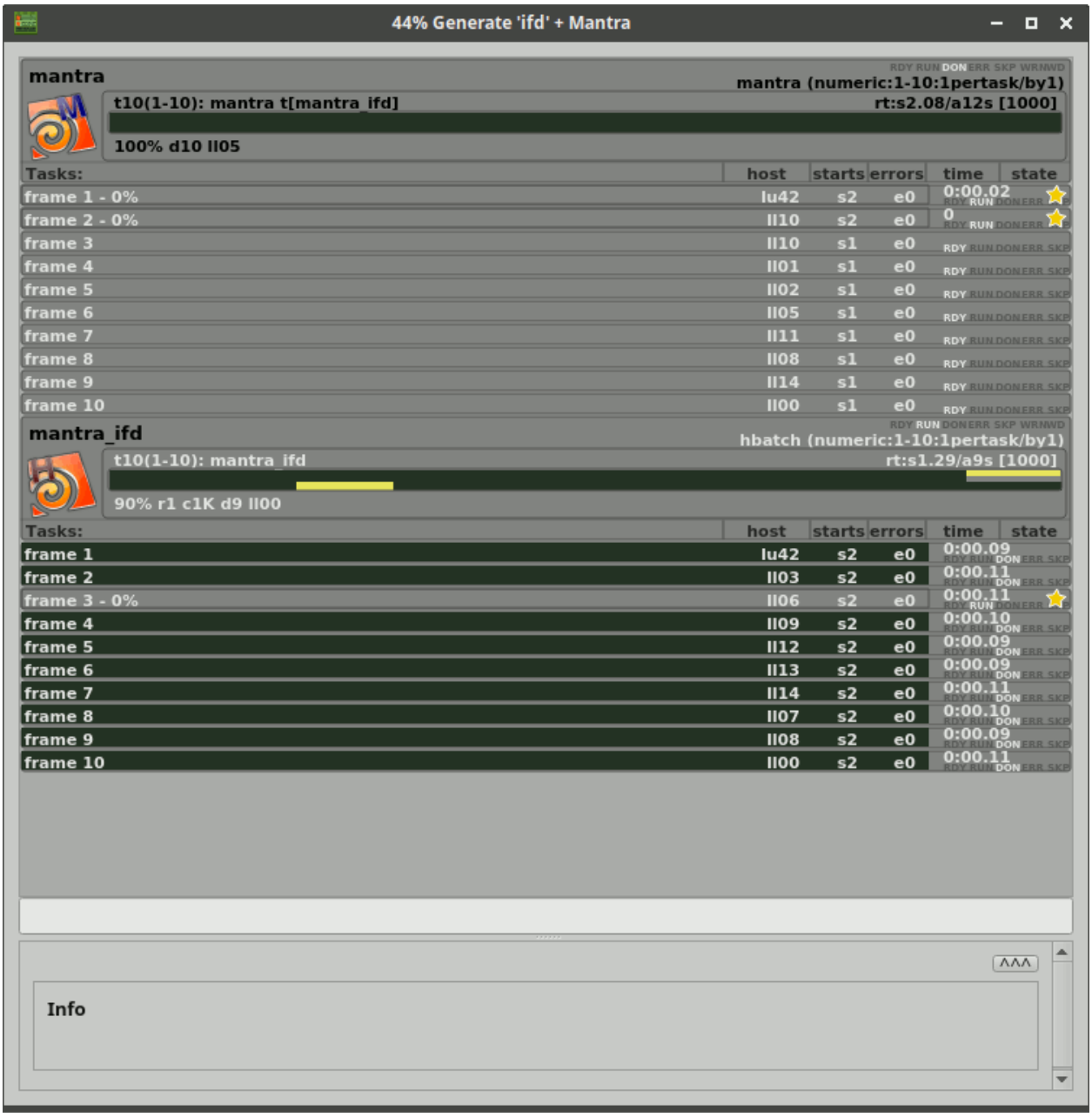


Fig. 17: Command Job Tasks

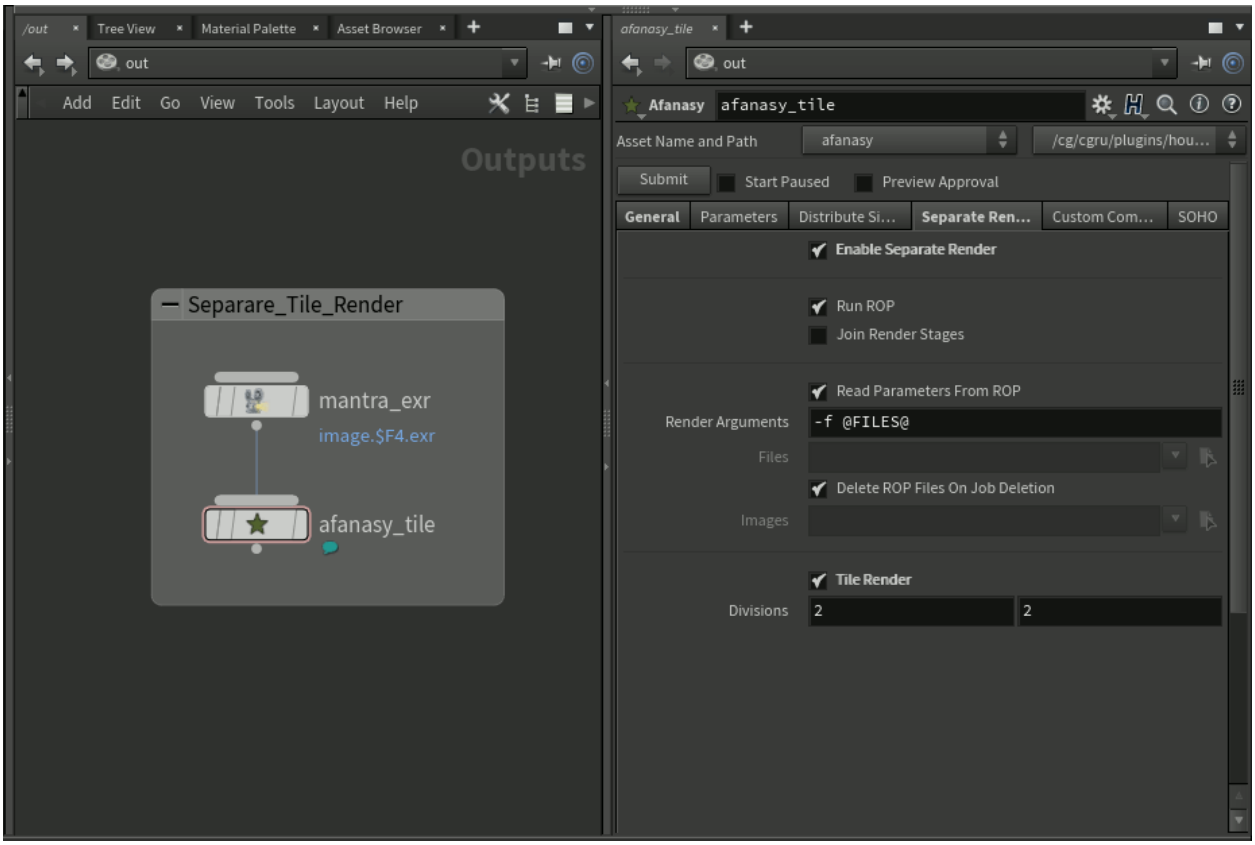


Fig. 18: Tile Render Network

Fig. 19: Tile Render Job

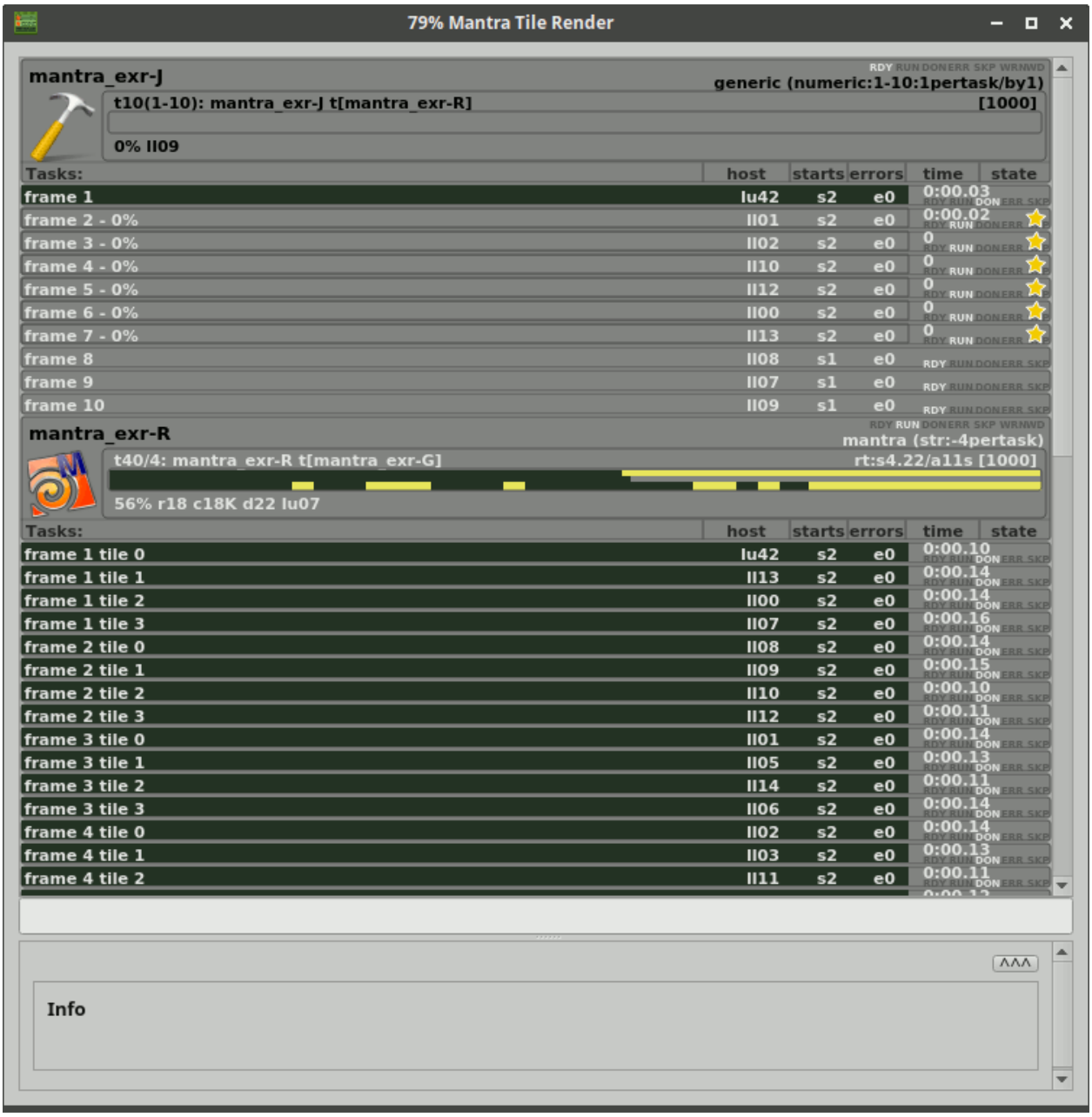


Fig. 20: Tile Render Job Tasks

Sub Task Dependence

This option is designed to start to render simulation without waiting the whole simulation is finished.

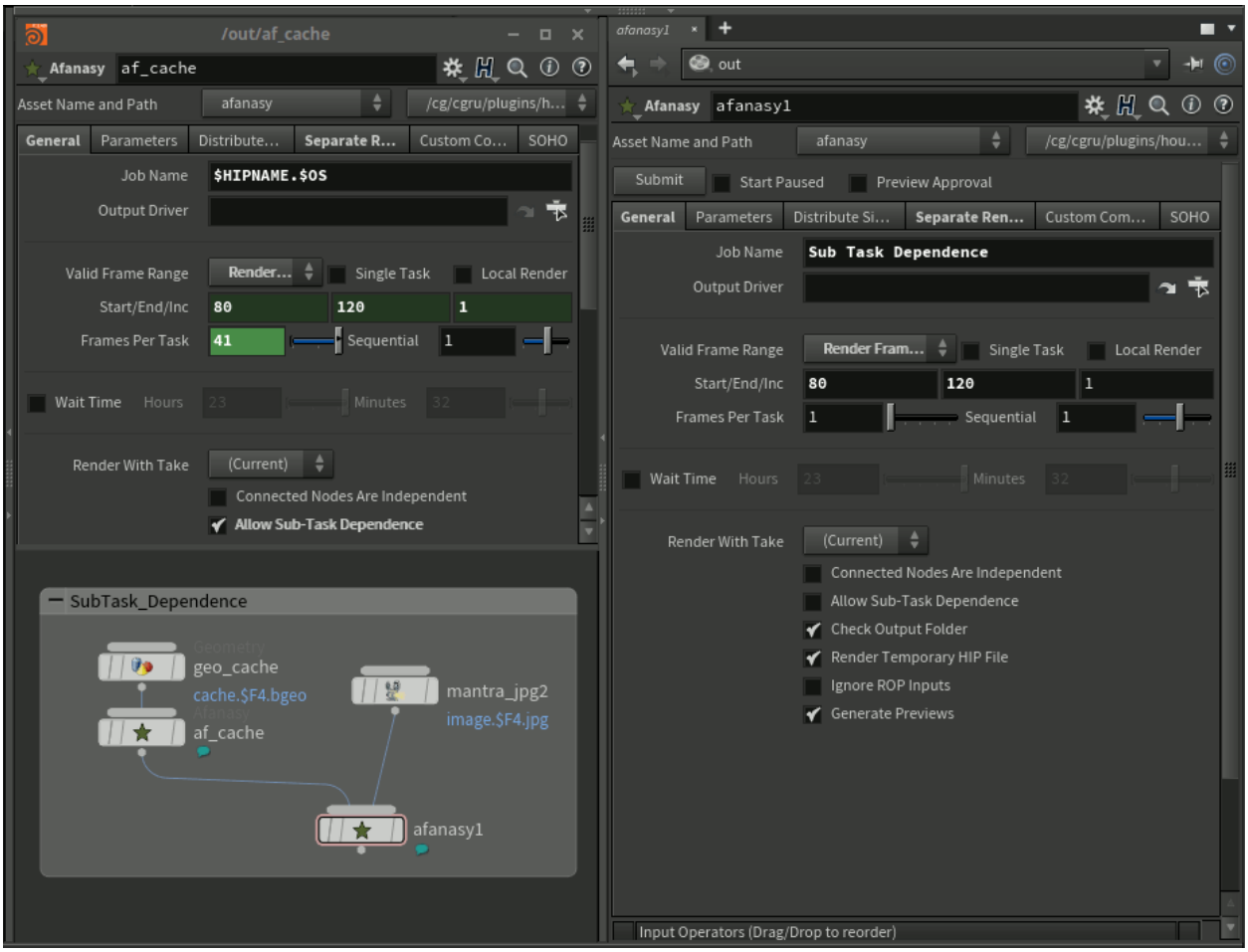


Fig. 21: Sub-Task Dependence Network

The first block of a job is a simulation. It consists of a single task (*Frames Per Task* parameter is set to the whole frame range). The second block set to wait the first one with sub-task dependence. So it begins to render as first frames of a simulation completed, while the simulation task is still running.

We also can notice here, that the render block got *HYTHON* and *MANTRA* tickets, while the simulation block got only *HYTHON* ticket

Complex

You can construct a complex Afanasy ROP network to construct a complex job.

This job consists of a simulation with sub-task dependence. Two caches waiting the simulation, but can run independently from each other. Mantra tile render which produces three blocks which wait all the cache. Two blocks for preview which can run independently but wait tile render tasks. One to convert EXR files to JPEG-s and one to generate a preview movie form EXR-s.

Fig. 22: Sub-Task Dependence Job

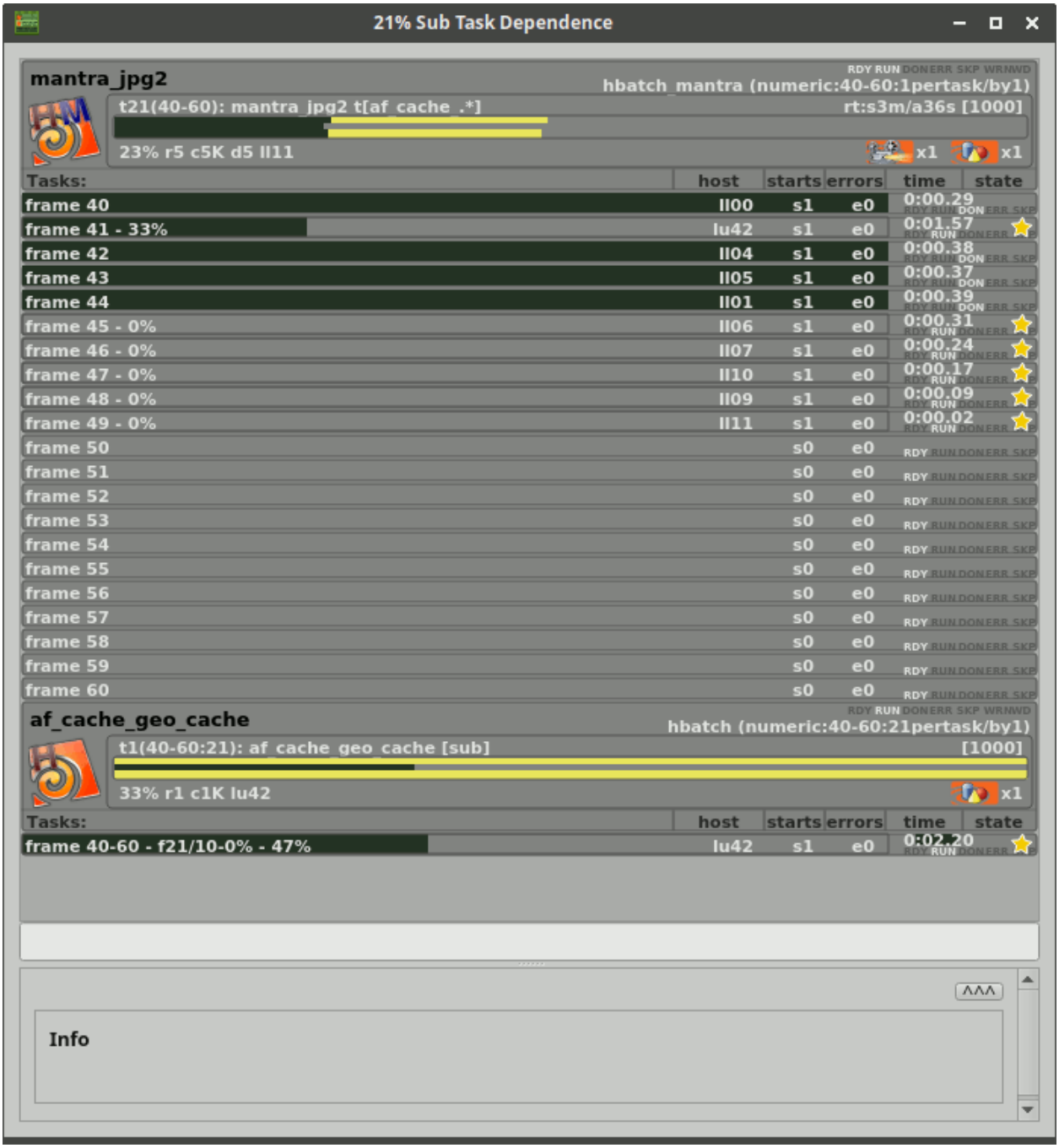


Fig. 23: Sub-Task Dependence Job Tasks

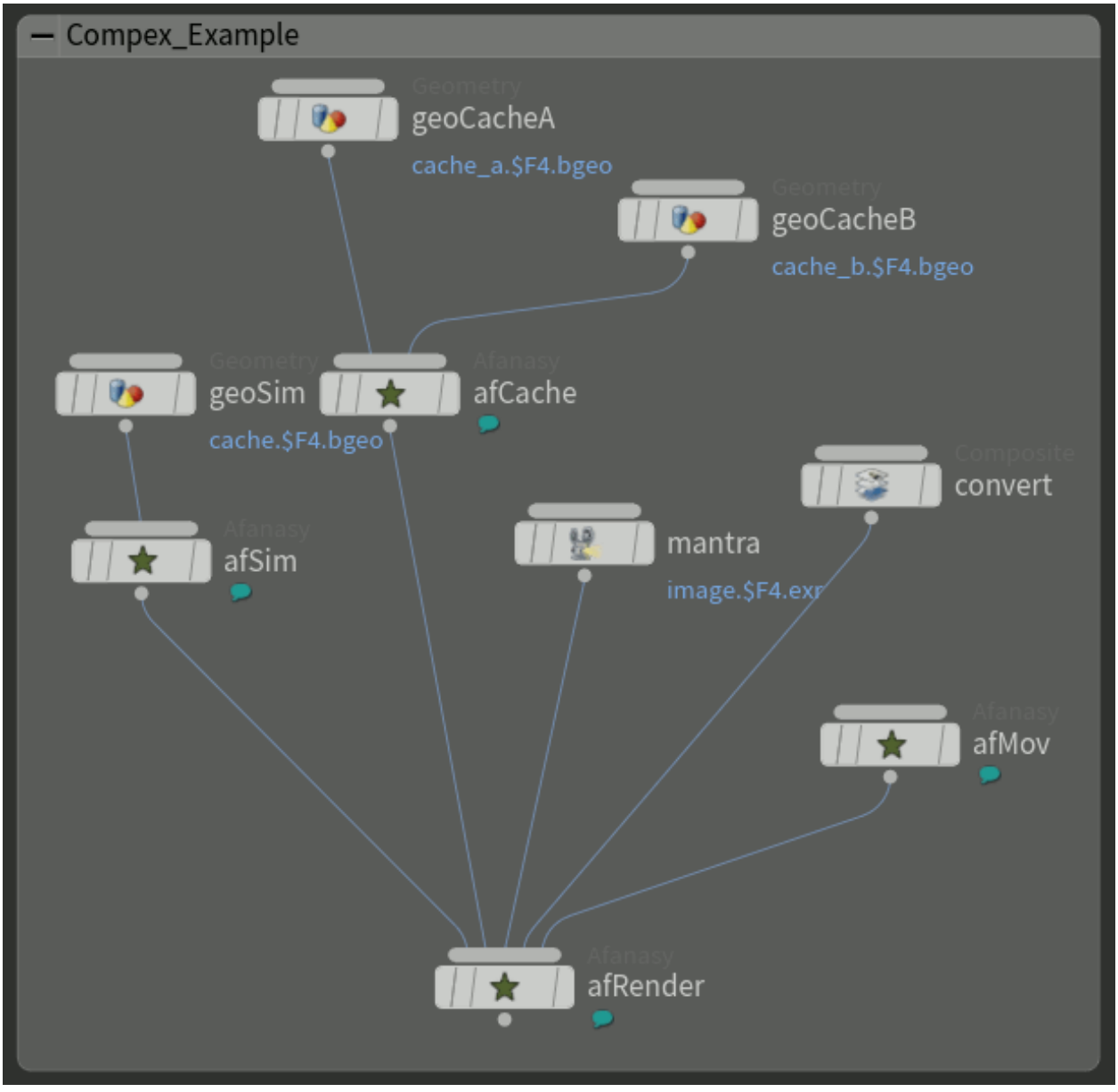


Fig. 24: Complex Network

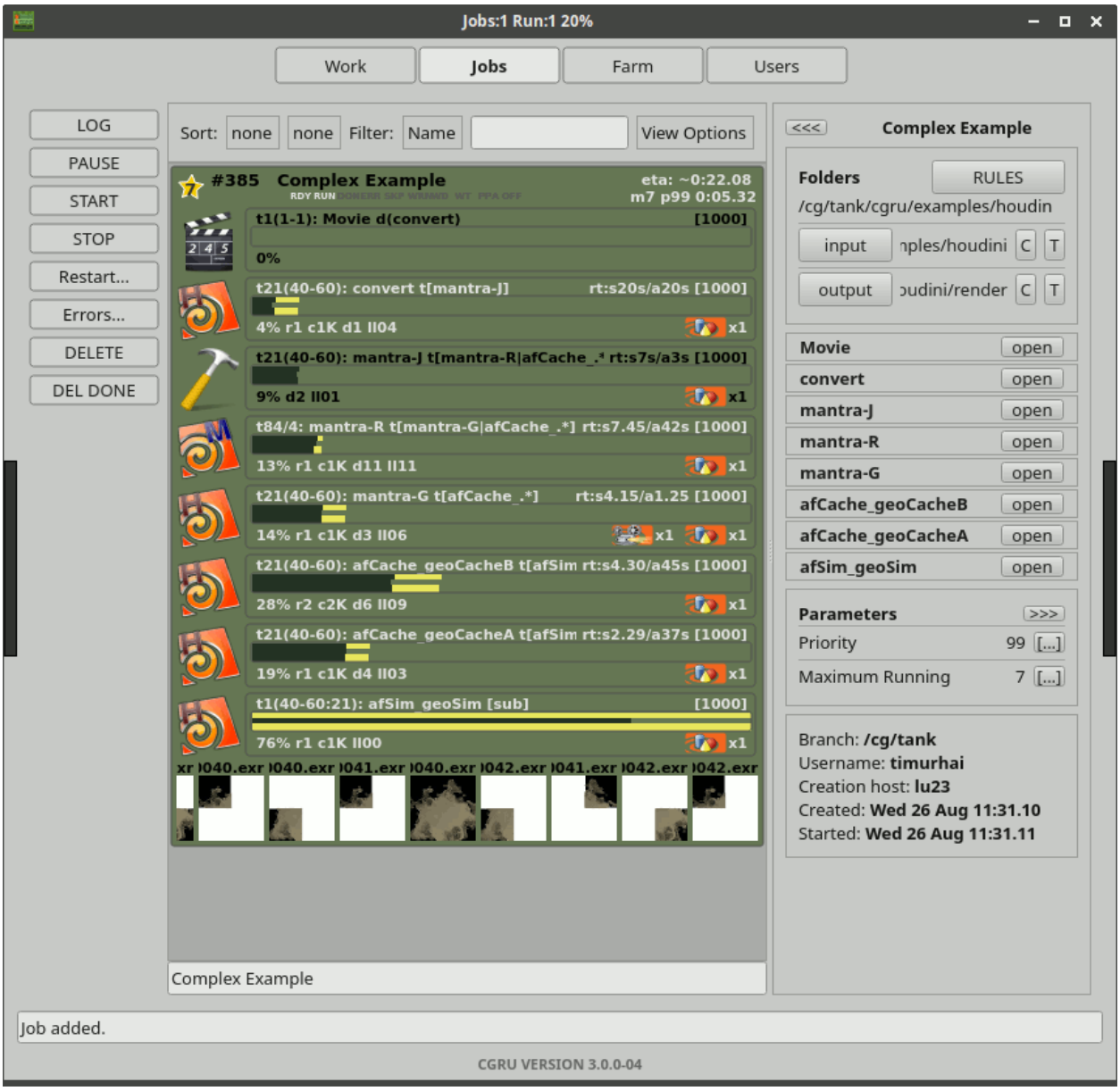


Fig. 25: Complex Job

5.7.2 Distributed Simulations

Houdini can calculate the same simulation on several machines.

5.7.2.1 How It Works

Simulation can be split on slices, so each machine calculates own slice. But different slices simulations should exchange information to pass data from slice to slice. Houdini has a special Python script *simtracker.py* for it. It needs to launch a server that simulations will connect to. So each slice simulation should know tracker address and port. Also tracker has a simple web interface to see logs.

5.7.2.2 What We Should Do

- Prepare distributed simulation, setup slices.
- Launch tracker server and get its address and port.
- Open several Houdini applications with simulation scene (on different machines or not).
- Specify tracker and port.
- Start each Houdini instance to simulate own slice.
- Stop tracker.

So, you can distribute Houdini simulation without any render farm manager.

5.7.2.3 Step-By-Step

1. Create a sphere.
2. Create simulation via Wispy Smoke shelf tool.
3. Apply Distribute Container shelf tool.
4. You will be moved to */out/* network.
5. Create Afanasy ROP node.
6. Set *Output Driver* to */obj/distribute_pyro/save_slices* and in the *Distributed Simulation* tab set *Controls Node* to */obj/AutoDopNetwork/DISRIBUTE_pyro_CONTROLS*. You can copy this values from *HQueue Simulation* ROP that was automatically created.

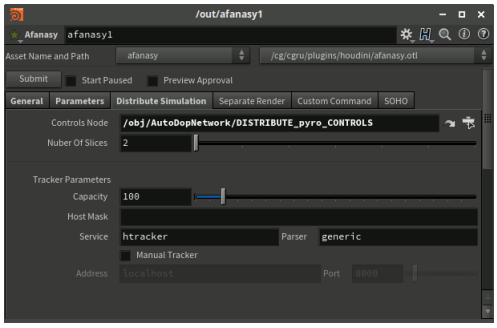


Fig. 27: Distributed Simulation Tab

7. Uncheck *Render Temporary HIP File* option on Afanasy ROP. By default, Afanasy renders a temporary scene to allow user to continue working with original file. But in this case *$HIPNAME* variable will change, and it widely used in shelf tools and examples.
8. Go to */obj/AutoDopNetwork/*.
9. Remove resize_container node.
10. **Disconnect *distribute_pyro* node from *merge* node (do not merge it with source). And connect it to the solver *Velocity Upda***

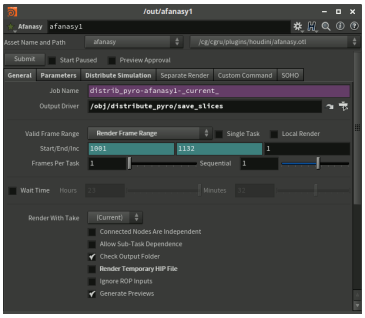


Fig. 26: Genetal Tab

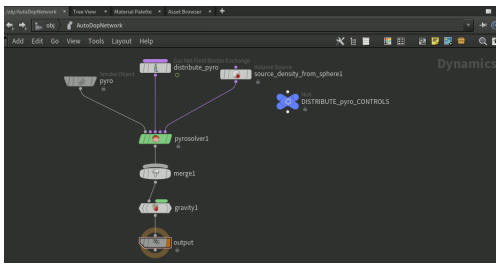


Fig. 29: Adjusted network

11. Set slices divisions 1 x 2 x 1.
12. Now you can submit simulation by Afanasy ROP in */out/* network.

5.7.2.4 Afanasy Job

Afanasy will create a job that consists of four blocks each contains just one task. First block task to start tracker. A block (task) for each slice that waits tracker start. And the last block task to stop the tracker.

1. **tracker**

   The first task block has a special service *htracker*. This service just adds job ID to the task command. Job ID is needed to manipulate job using JSON protocol. The command calls a special CGRU Python script `plugins/houdini/htracker.py`.

   ```
   htracker --start --envblocks "save_slices.*|tracker-stop" --depblocks "save_
   ↪slices.*"
   ```

   - It starts Houdini *simtracker* in a separate thread and gets its address and port.
   - Set other job blocks environment variables `TRACKER_ADDRESS` and `TRACKER_PORT` to blocks specified by *–envblocks* argument.

Fig. 30: Distributed Simulation Job Running

- Set slices job blocks depend masks to an empty string to blocks specified by *–depblocks* argument, So that blocks will wait nothing and can to start.
- Waits *simtracker* for completion.

2. **save_slices-s0**

   The first slice simulation. Slices are simulated by CGRU multi-functional Hython script `cgru/plugins/houdini/hrendef_af.py` that Afanasy uses for almost everything.

```
hrender_af -s 1001 -e 1133 --by 1 -t "_current_" --ds_node "/obj/AutoDopNetwork/
↪DISTRIBUTE_pyro_CONTROLS" --ds_address "localhost" --ds_port 8000 --ds_slice 0
↪"/opt/cgru/examples/houdini/distrib_pyro.hip" "/obj/distribute_pyro/save_slices"
```

   Control node, tracker address and tracker port, that was specified in Afanasy ROP and passed by command line argument, will be overridden by environment variables.

   Script will open HIP file, set control node tracker address and port parameters. Set *SLICE* variable to the specified slice number.

   Run simulation ROP.

3. **save_slices-s1**

   The second slice simulation. It is the same as the first, but with one key difference. Slice will be equal to 1.

```
hrender_af -s 1001 -e 1133 --by 1 -t "_current_" --ds_node "/obj/AutoDopNetwork/
↪DISTRIBUTE_pyro_CONTROLS" --ds_address "localhost" --ds_port 8000 --ds_slice 1
↪"/opt/cgru/examples/houdini/distrib_pyro.hip" "/obj/distribute_pyro/save_slices"
```

4. **tracker-stop**

   Stop tracker. It will be performed by the same script that starts tracker.

```
htracker --stop
```

   It just sends `quit` string to tracker_address:tracker_port socket.

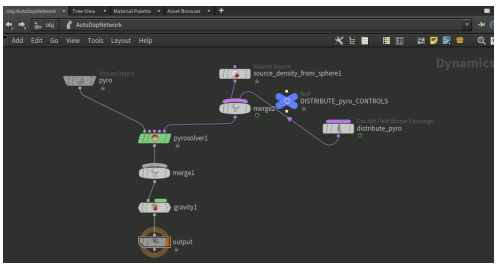


Fig. 28: Original network

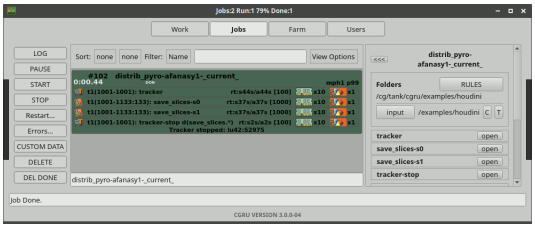


Fig. 31: Distributed Job Done

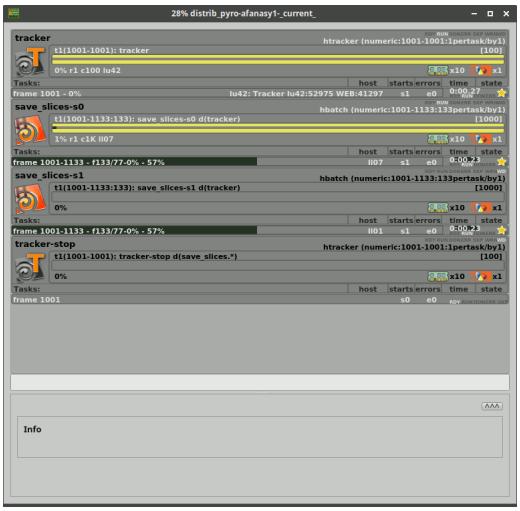
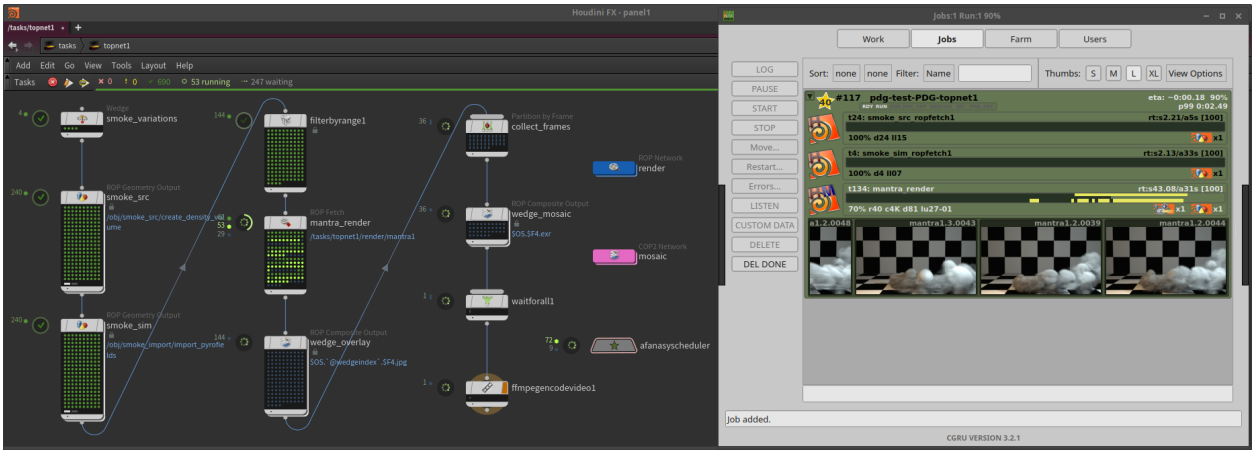
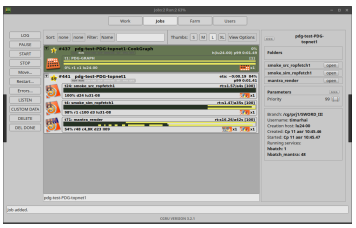


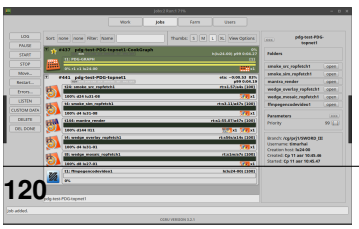
Fig. 33: Distributed Job Tasks

5.7.3 Afanasy TOP Scheduler

This node executes work items on farm using Afanasy render manager. It can schedule work items from TOP UI, and as standalone job. Using a standalone job you can close Houdini session and watch progress via Afanasy GUI.







Here is the scheduler nodes parameters description. Almost anywhere in Afanasy `-1` means that the value is not set and defaults will be used.

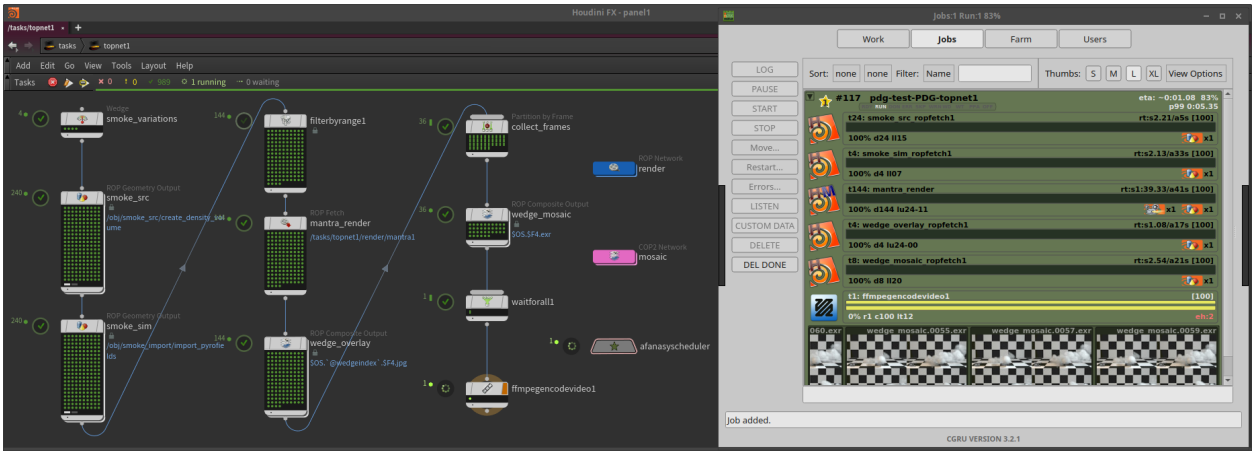


Fig. 34: Scheduling from Houdini TOP UI

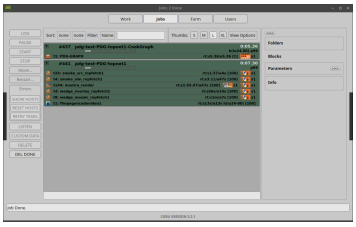


Fig. 35: Scheduling using a standalone job

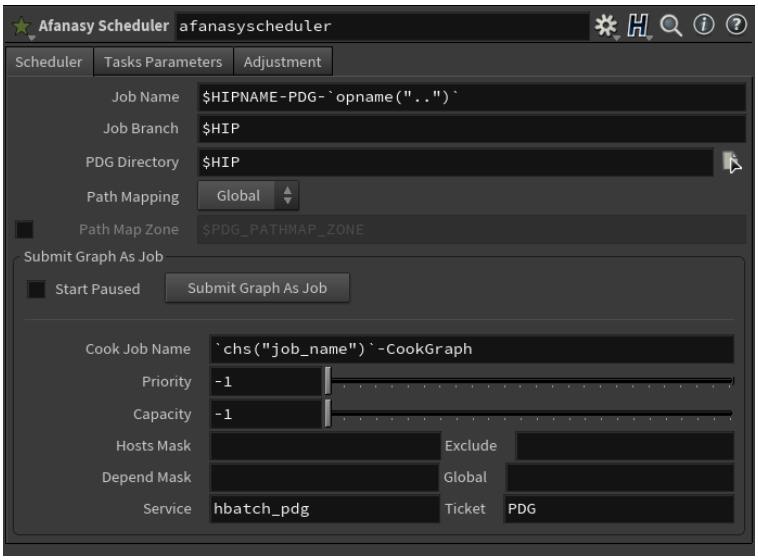


Fig. 36: Afanasy TOP Shecduler tab

5.7.3.1 Scheduling Parameters

- **Job Name** The name of the job where work items tasks will be appended to.
- **Job Branch** The branch of the job. The same value will be used if you submit graph as job.
- **PDG Directory** Specifies the directory where the cook generates intermediate files. The intermediate files are placed in a subdirectory named `pdgtemp`.

- **Path Mapping**
  - **Global** If the PDG Path Map exists, then it is applied to file paths.
  - **None** Delocalizes paths using the PDG_DIR token.
- **Path Map Zone** When on, specifies a custom mapping zone to apply to all jobs executed by this scheduler. Otherwise, the local platform is `LINUX`, `MAC` or `WIN`.

5.7.3.2 Submit Graph As Job

- **Submit Graph As Job** Cooks the entire TOP network as a standalone job. Displays the status URI for the submitted job. The submitting Houdini session is detached from the cooking of the TOP network.
- **Start Paused** Start graph cooking job paused.
- **Priority** Graph cooking job priority value.
- **Capacity** Cooking task capacity.
- **Hosts Mask** Hosts names regular expression, where graph job can run.
- **Exclude** Hosts names regular expression, where graph job can not run.
- **Depend Mask** Current user jobs names expression, that job will wait to start for.
- **Global** Any user jobs names expression, that job will wait to start for.
- **Service** Cooking job task block service name.
- **Ticket** Cooking job task block will need and take one ticket with this name. See tickets documentation for details.

5.7.3.3 Tasks Parameters

You can override this parameters on each TOP node, except *Job Priority* which will be set to an entire job.

- **Job Priority** Priority value of a job were working items tasks will be executed.
- **Capacity** Work items tasks block capacity.
- **Hosts Mask** Hosts names regular expression, where tasks can run.
- **Exclude** Hosts names regular expression, where tasks can not run.
- **Max Running Tasks** Running tasks count at the same time limit.
- **Per Host** Running tasks count at the same time on the same host limit.
- **Render Time Min (Sec)** Minimum task running time limit. If task will finish for seconds below this value, task finish will be considered as with an error.

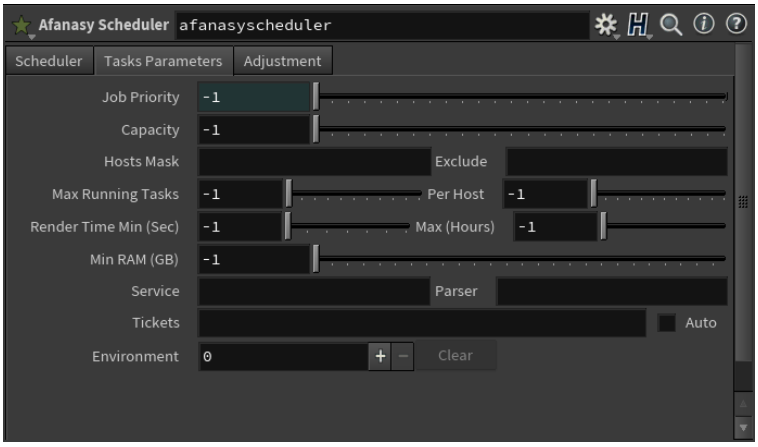


Fig. 37: Afanasy TOP Tasks Parameters tab

- **Max (Hours)** Maximum task running time limit. If task will run for hours above this value, it will be forced to stop with an error.
- **Min RAM (GB)** Host should have this count of Gigabytes of a free RAM to be able to run tasks.
- **Service** Tasks block service. If empty it will try to detect automatically. If node fetches *ifd* ROP, service will be *hbatch_mantra*, *ffmpegencodevideo* will be *ffmpeg*.
- **Tickets** asks block tickets. A comma separated list of key:count. Example: `MEM:64,GPU:1`. See tickets documentation for details.
- **Auto** Automatically add common tickets. Almost all tasks launch *hython*, so *HYTHON* ticket will be added. If node fetches *ifd* ROP, *MANTRA* ticket will be added.
- **Environment** Adds custom key-value environment variables to tasks block.

To override task parameter on TOP node add it via Edit Parameter Interface window:

5.7.3.4 Adjustment Parameters

- **Report Item Fail On Error** If task gets error state, scheduler will report PDG that work item failed. You can turn it off and try to solve errors via Afanasy only. Read output for an error cause, try to fix it, restart task. And PDG will know nothing about it.
- **Block On Failed Work Items** When this option is enabled the scheduler will block the cook from completing if there are any failed work items in that scheduler. This makes it possible to manually retry those work items, by preventing the PDG graph cook from ending before failed items can be retried. A cook that is blocked on failed work items can still be canceled using the ESC key, the cancel button in the TOP task bar, or the cancel API method.
- **Validate Outputs When Recooking** When enabled, PDG will check the output files of work items when the graph recooks, to see if the files still exist on disk. Work items that are missing output files will be dirtied and cook again.
- **Check Expected Outputs On Disk** When enabled, PDG will look on disk for any expected work items outputs that were not explicitly reported when the work item cooked. Expected outputs for a work item are checked immediately after the scheduler marks the work item as cooked. Output files that were reported by the work item normally while cooking will not be checked.
- **Use IP Address** Use IP address instead of host name as work item result server address. Some times render farm can't solve workstations by name. Also it can save DNS load.

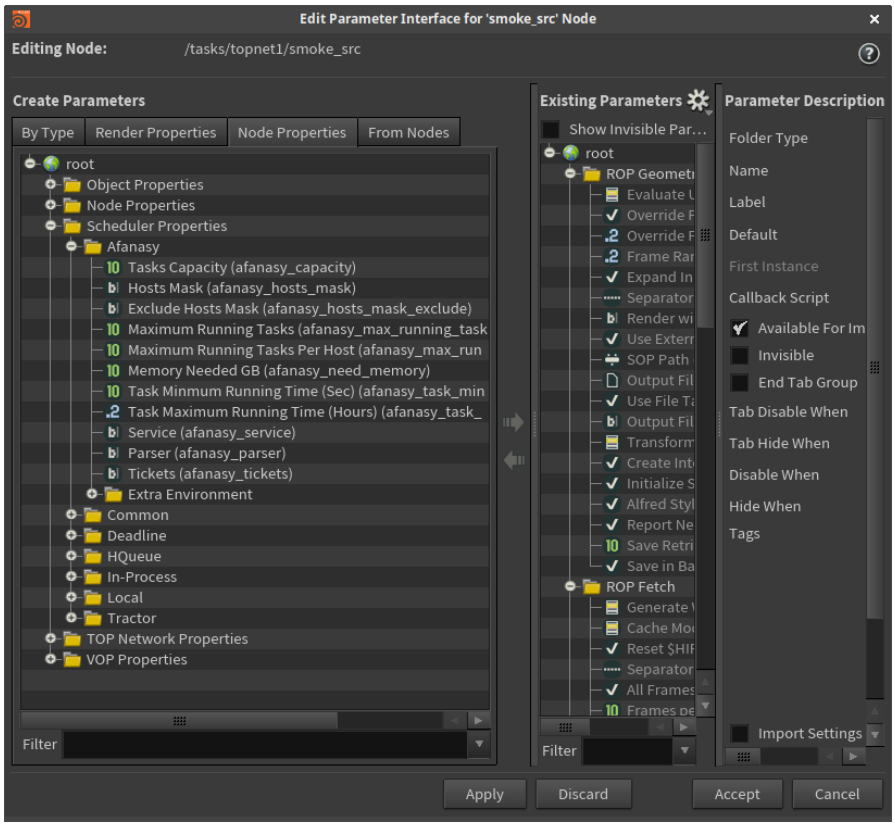


Fig. 38: Edit Parameter Interface window

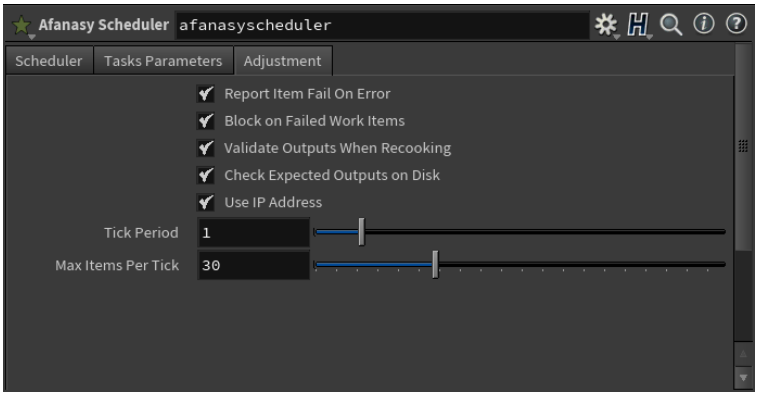


Fig. 39: Afanasy TOP Adjustment tab

Work item task can send progress to PDG itself. It is used by batch work item to notify that a specific frame (item in the batch) is done. This way PDG can start to render images when just first frames of a simulation rendered, and there is no need to wait the entire simulation finish. So work item should know address and port to send progress to.

- **Tick Period** Sets the minimum time (in seconds) between calls to the *onTick* callback. This callback is called periodically when the graph is cooking. The callback is generally used to check the state of running work items.

  Afanasy server heart beat is 1 second, so there is no sense to set this parameter less than a second.

- **Max Items Per Tick** Sets the maximum number of ready item *onSchedule* callbacks between ticks.

  For example by default the tick period is 1s and the max items per tick is 30. This means that scheduler can send a maximum of 30 work items per second to farm. Adjusting these values can be useful to control the load on the farm scheduler.

5.7.4 Setup

CGRU setup should be sourced before. To do this you can source *setup.sh* script in CGRU root folder. Afanasy Houdini operator library and Python module are located in:

`cgru/plugins/houdini`

You should add this folder `HOUDINI_PATH` and `PYTHONPATH` environment variables.

Houdini setup example (*bash*):

```
# Setup CGRU
cd /opt/cgru
source ./setup.sh

# Setup CGRU houdini plugins location:
export HOUDINI_CGRU_PATH="${CGRU_LOCATION}/plugins/houdini"

# Append HOUDINI_PATH with CGRU plugins:
export HOUDINI_PATH="${HOUDINI_CGRU_PATH}:&"

# Append Python path with afanasy submission script:
export PYTHONPATH="${HOUDINI_CGRU_PATH}:${PYTHONPATH}"
```

If you avoid sourcing `cgru/setup.sh` see Manual Environment Setup.

5.8 Maya

Warning: Documentation is outdated.

5.8.1 meTools for Afanasy

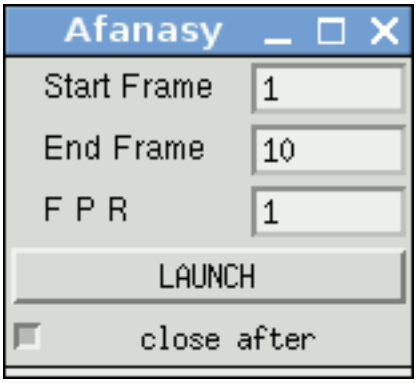
http://meshstudio.blogspot.ru/2013/01/metools-for-afanasy.html

Maya VRay, Arnold and MentalRay standalone rendering.

5.8.2 Stand-Alone Dialog

Use AfStarter

5.8.3 The Simplest MEL Dialog



Afanasy menuitem in CGRU raises a window. Dialog is minimalistic but enough to do anything that needed in most cases. It can render any engine type: Software, VRay, Arnold, MentalRay or whatever. It simple runs Maya Render command and ask it to open scene and render special frame(s). All render settings Maya takes from render globals in this case.

5.8.4 CGRU Maya

A set of MEL scripts and a plug-in.

Documentation:

https://cgru.info/maya

5.9 Natron

CGRU in Natron consists of Afanasy node (group) and menu items in main CGRU menu.

Afanasy nodes need to render connected Write nodes and to store render settings. You can connect one Afanasy node to other Afanasy node to render other Write node with different settings at the same time. Each connected node will produce a block - an array of tasks (frames) to render. You can specify dependence between connected nodes.

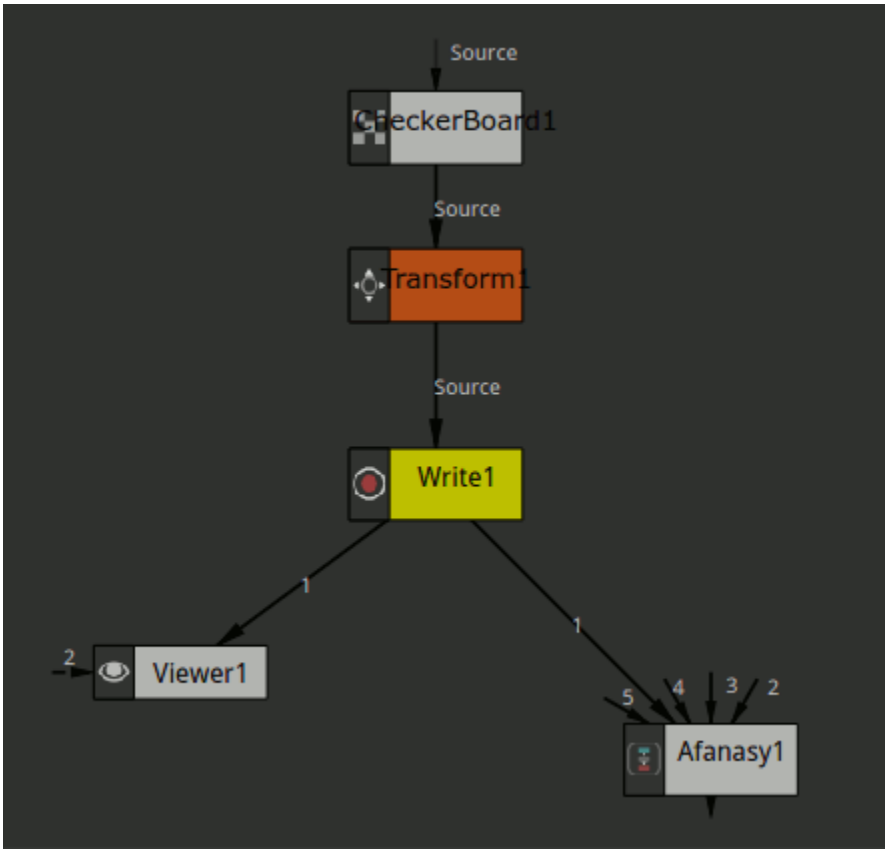


Fig. 40: Node Graph

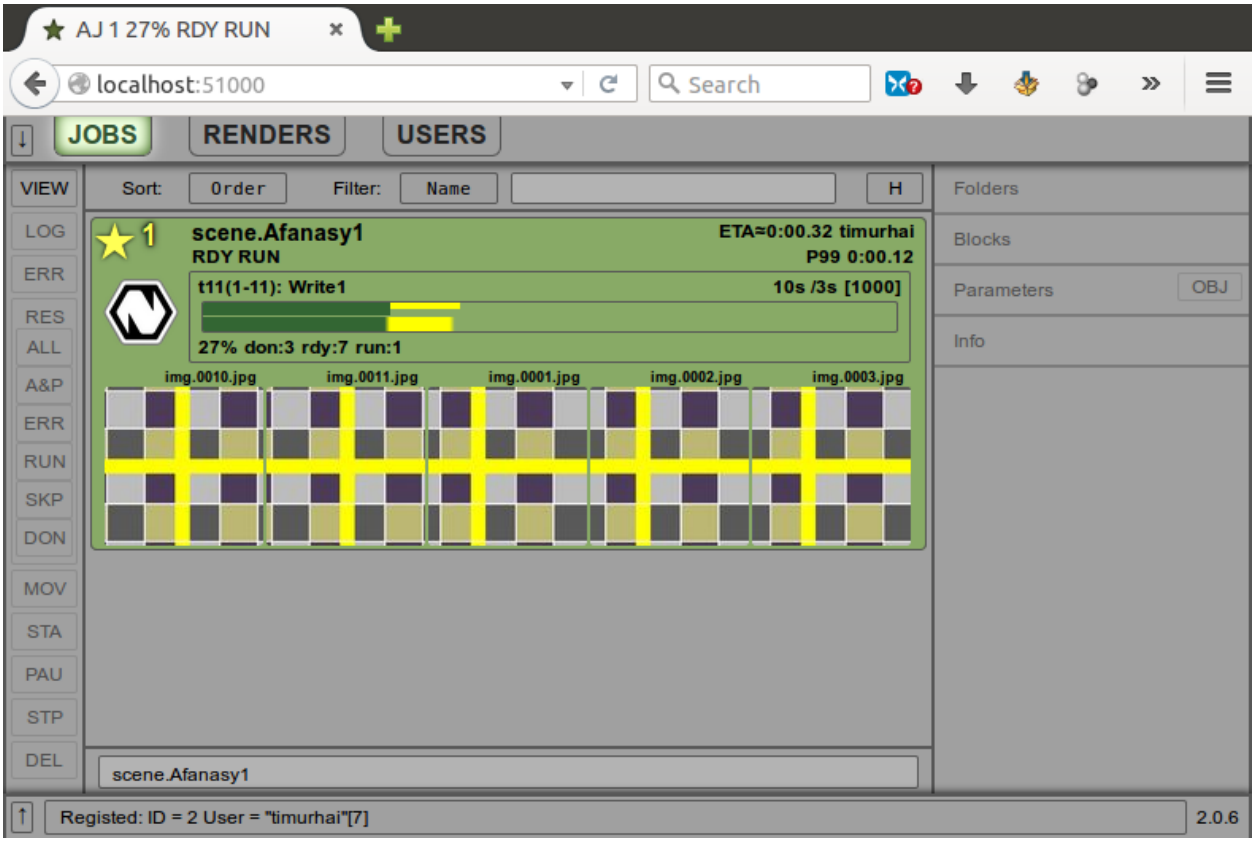
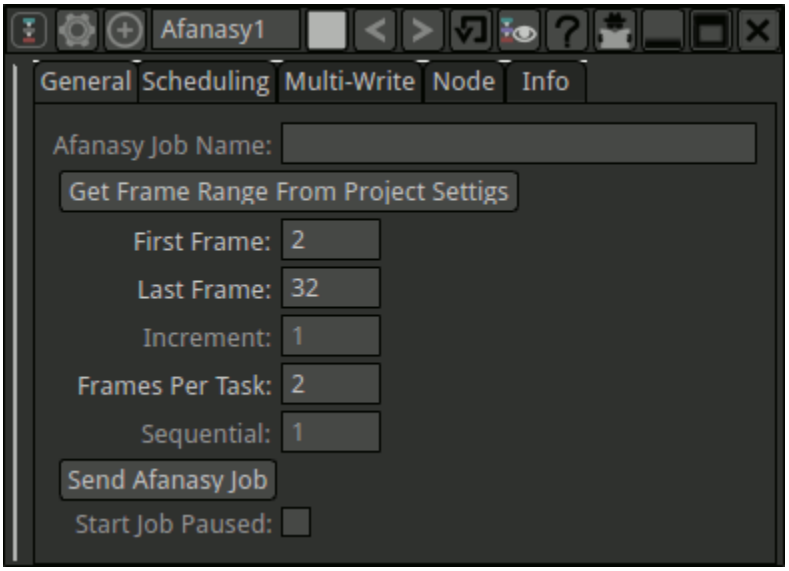


Fig. 41: Job (Web GUI)

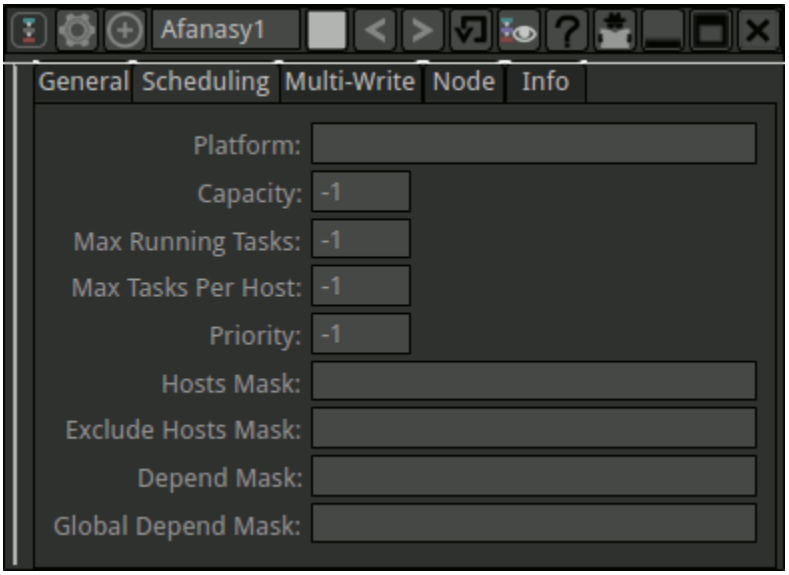
5.9.1 Afanasy Node

5.9.1.1 General



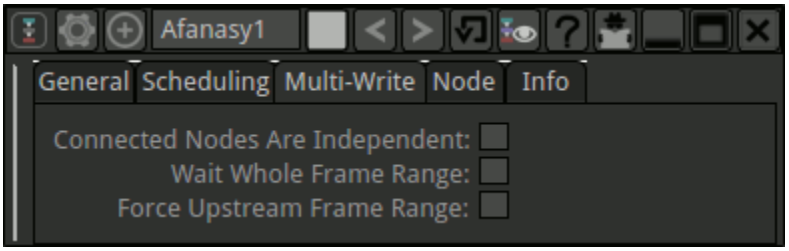
- **Job Name** Name to add to job or blocks names. If empty, Afanasy node label will be used.
- **Get Frame Range From Project Settings** Set first and last frames to project settings.
- **First Frame** First frame to render.
- **Last Frame** Last frame to render.
- **Increment** Frames increment step.
- **Frames Per Task** Number of frames in task.
- **Sequential** Frames solving method.
- **Send Afanasy Job** Construct and send job to Afanasy server.
- **Start Job Paused** Job will be send paused.

5.9.1.2 Scheduling



- **Platform** OS type the job can launch tasks on: 'Any' - any OS, 'Native' - the same as the script was launched on.
- **Capacity** Tasks capacity. '-1' - use default value.
- **Max Running Tasks** Maximum number of running at the same time tasks. '-1' means no limit.
- **Max Tasks Per Host** Maximum number of running at the same time at the same host tasks. '-1' means no limit.
- **Priority** Job priority. '-1' - set default priority value.
- **Hosts Mask** Job will only on hosts which name matches this pattern.
- **Exclude Hosts Mask** Job will not run on hosts which name matches pattern.
- **Depend mask** Job will wait job(s) to be done, which name(s) matches pattern.
- **Global Depend mask** The same, but will wait for jobs from any user.

5.9.1.3 MultiWrite



- **Connected Nodes Are Independent** Nodes can run at the same time, they will not wait each other.
- **Wait Whole Frame Range** Down stream connected node(s) will wait until whole specified frame range will be rendered. If not checked, each frame will be wait only corresponding frame(s) from this node.
- **Force upstream frame settings** All upstream connected Afanasy nodes will use this node frame range.

5.9.2 Complex Situation

In this example Final Write waits Back and Front precomps to be rendered.

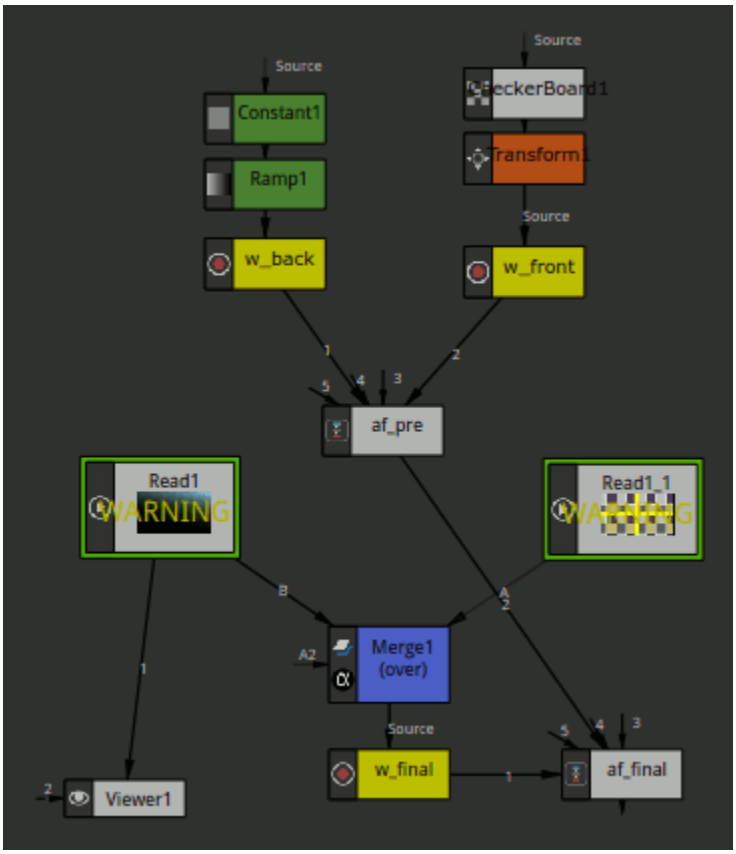
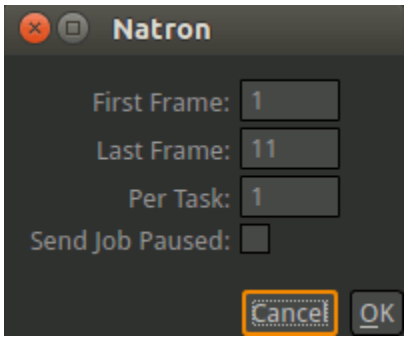


Fig. 42: Complex Network

First job block from **w_final** Write node has `af_pre.*` tasks depend mask. This means that it will wait all Write nodes that connected to **af_pre** Afanasy group.

Examples are located in `cgru/examples/natron`.

5.9.3 Render Selected



- **First Frame** First frame to render.

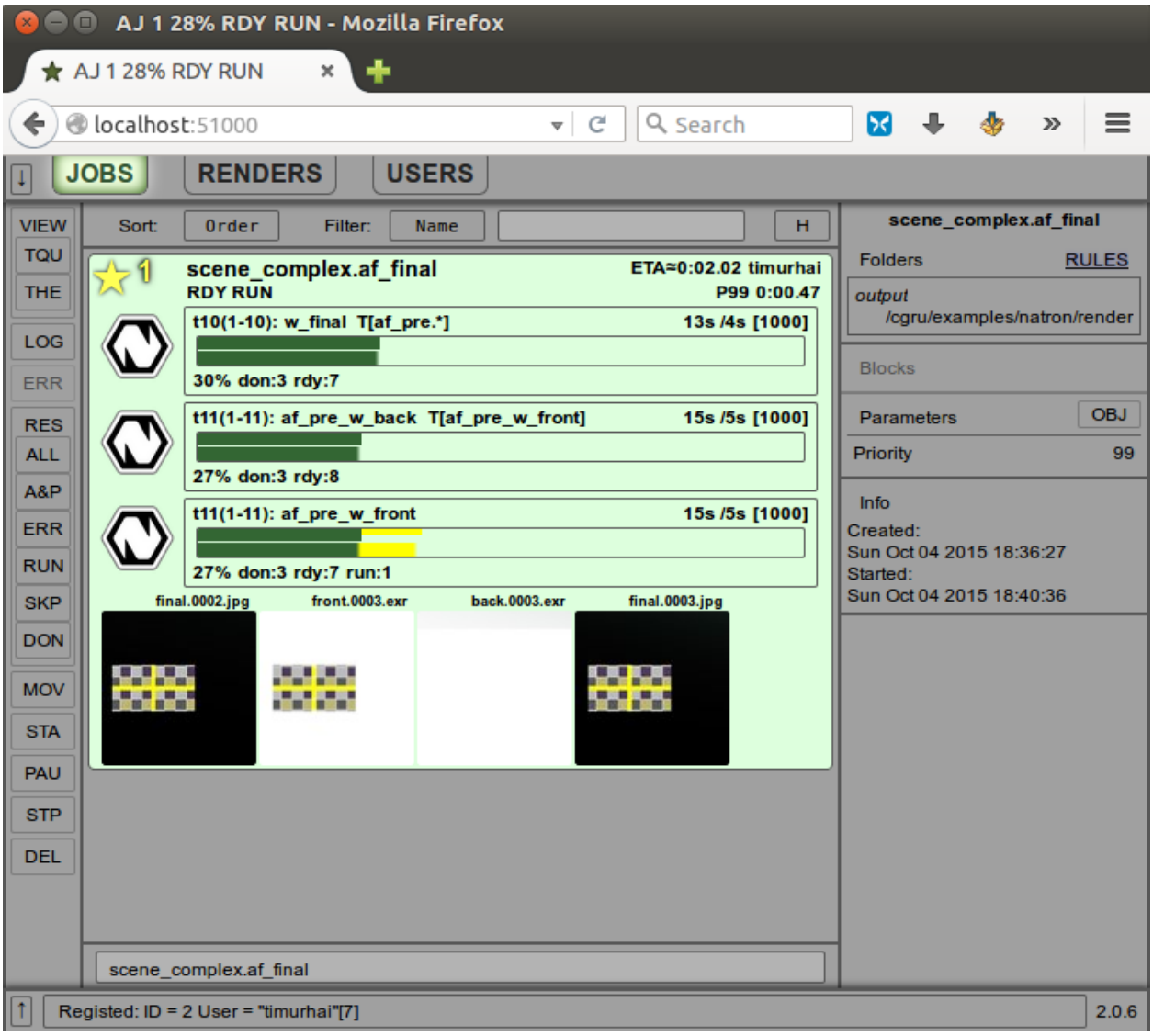


Fig. 43: Complex Job (Web GUI)

- **Last Frame** Last frame to render.
- **Per Task** Number of frames in task.
- **Send Job Paused** Job will be paused.

5.9.4 Setup

If you start Natron from CGRU Keeper all should work automatically.

5.9.4.1 Manual Setup

CGRU Natron plugins are located in

`cgru/plugins/natron`

Add this path to `NATRON_PLUGIN_PATH` environment variable.

5.10 Nuke

5.10.1 CGRU Menu

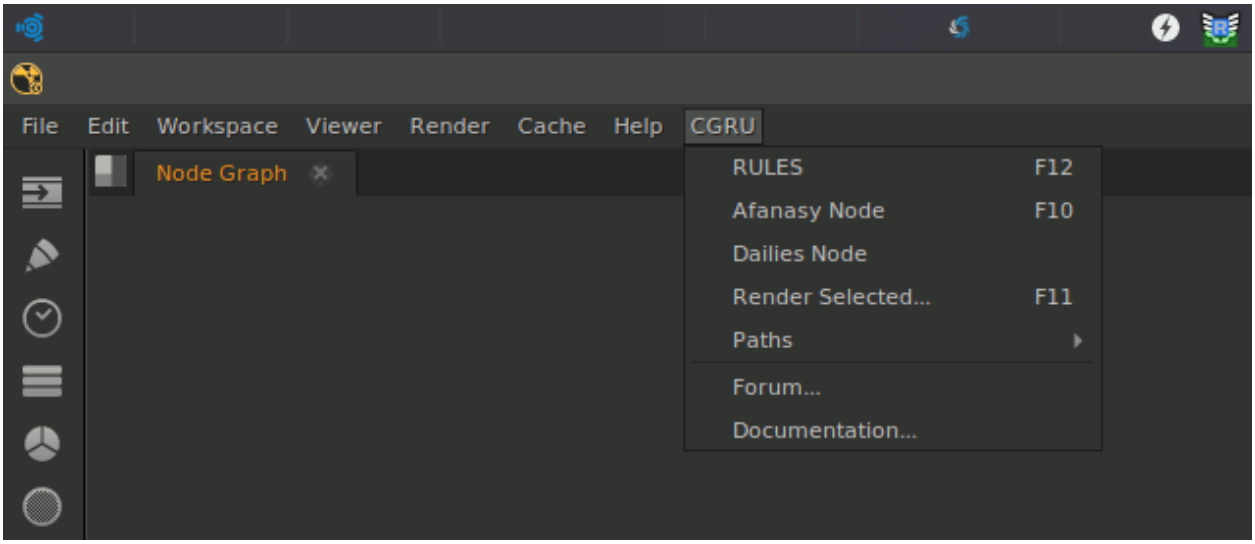


Fig. 44: Nuke CGRU Menu

Nuke Afanasy interface consists of **afanasy** nodes (gizmos) and menu items in main CGRU menu.

Just create **afanasy** gizmo *(F10)* and connect it to **Write** node to render.

5.10.2 Afanasy Gizmo

5.10.2.1 General

- **Job Name** Name to add to job or blocks names. If empty, 'afanasy' node name will be used.

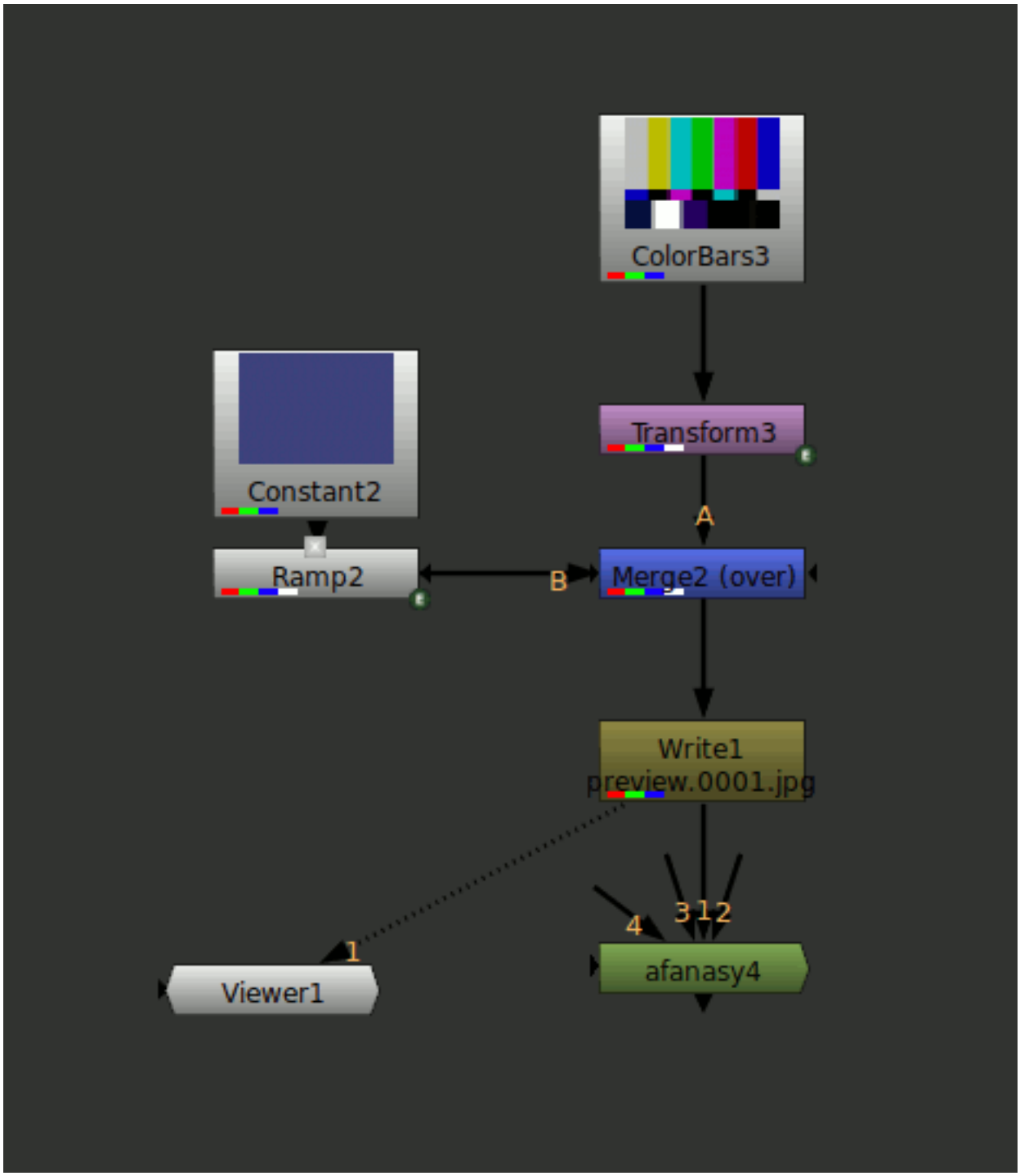


Fig. 45: Complex Node Network

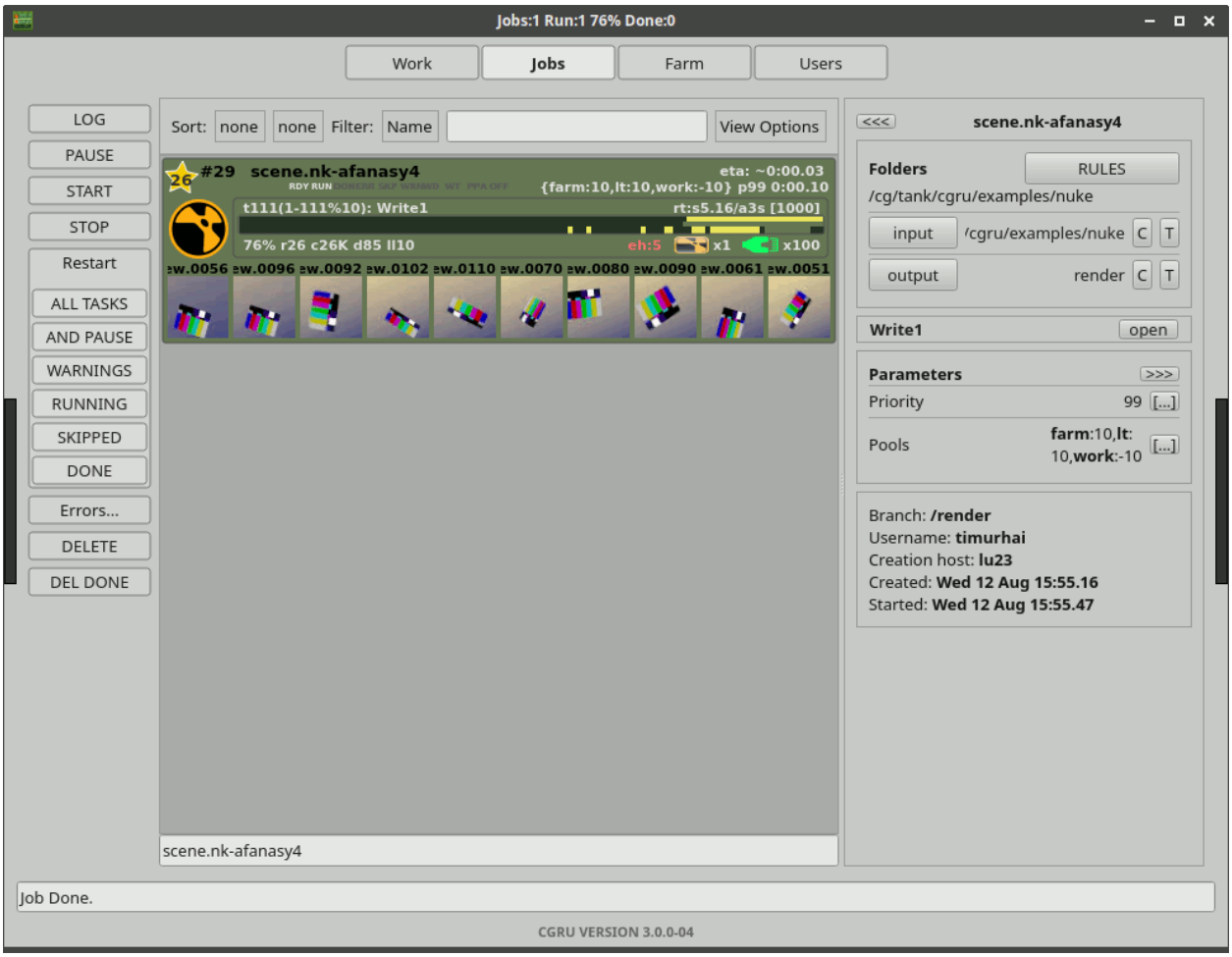


Fig. 46: Complex Job (AfWatch)

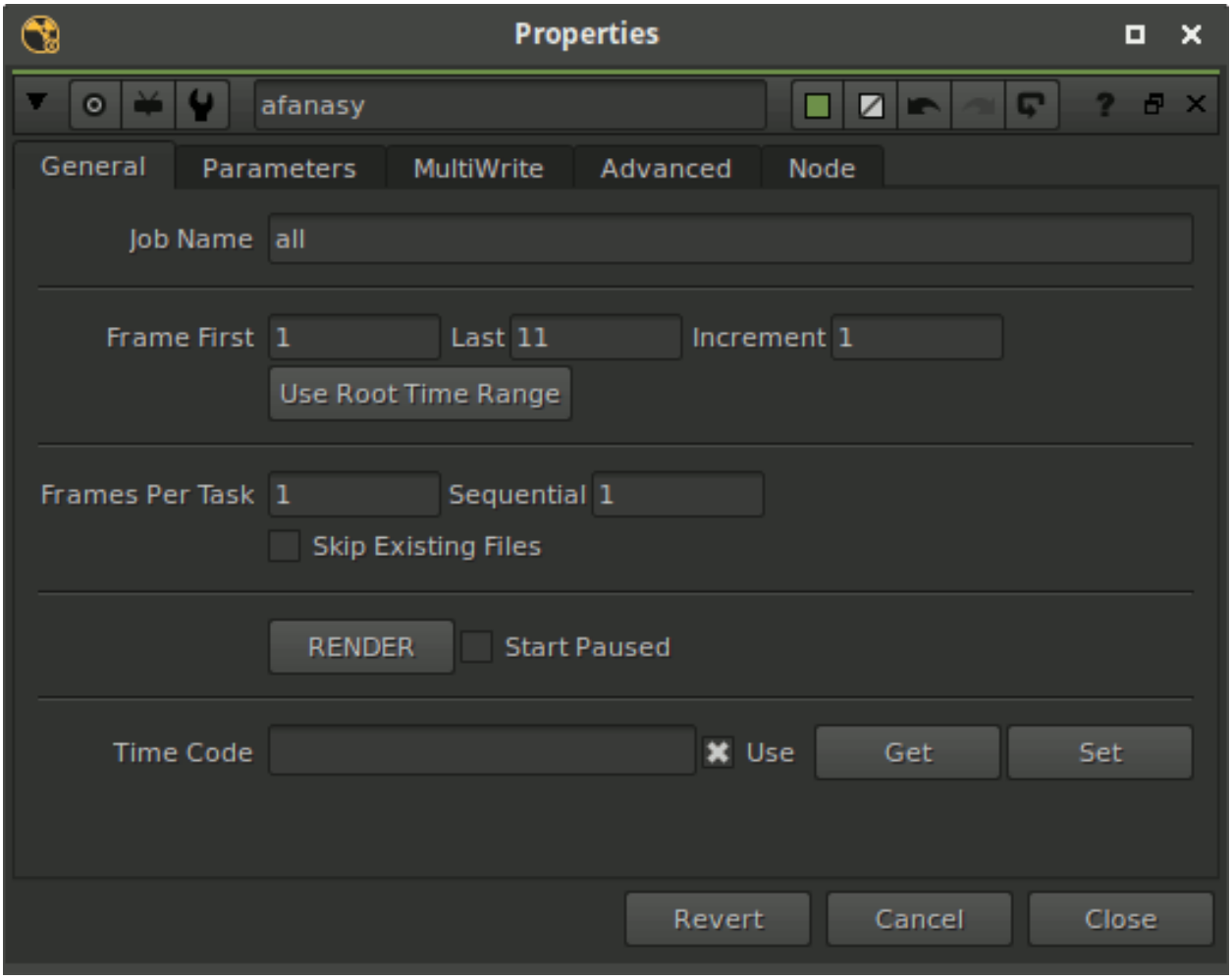


Fig. 47: Afanasy Gizmo General Tab

- **Use Root Time Range** Set 'First Frame' and 'Last Frame' fields to project settings.
- **First Frame** First frame to render.
- **Last** Last frame to render.
- **Increment** Frame increment.
- **Frames Per Task** Number of frames in task.
- **Sequential**

1	Render frames one by one from the first to the last
10	Render every 10 frame at first, than render last other frames
-1	Render frames backwards from the last to the first
-10	Render every 10 frame at first backwards, than render last other frames backwards
0	Render the first, the last, the middle, the middle of the middle and so on

- **Skip Existing Files** Skip existing files works fine, when *Frames Per Task* is 1
- **Render** Send job to Afanasy server.
- **Start Paused** Job will be sent in offline state.
- **Time Code** Two Time Codes from - to.
  - **Use** Use Time Code instead of frame range. If Time Code is empty, frame range will be used.
  - **Get** Get Time Code from frame range.
  - **Set** Set frame range from Time Code.

5.10.2.2 Parameters

- **Platform** OS type the job can launch tasks on: *Any* - any OS, *Native* - the same as the script was launched on.
- **Max Running Tasks** Maximum number of running tasks at the same time. *-1* means no limit.
- **Priority** Job priority. *-1* - use default priority value.
- **Hosts Mask** If not empty, job can run only on hosts which name matches this pattern.
- **Exclude Hosts Mask** If not empty, job can not run on hosts which name matches pattern.
- **Depend mask** If not empty, job will wait job(s) to be done, which name(s) matches pattern.
- **Global Depend mask** The same, but will wait for jobs from any user.
- **Capacity** Tasks capacity. *-1* - use default value.
- **Max Tasks Pet Host** Maximum running tasks on the same host at the same time.
- **Max Task Run Time** Maximum task running time in seconds. After this time, running task will be restarted. Useful if script can hang.

5.10.2.3 MultiWrite

- **Connected nodes are independent** nodes can run at the same time, they will not wait each other.
- **Reverse dependences on connected nodes** First block will wait second block. Most depended "Write" node usually produces more final result, and it will be executed as soon as possible.

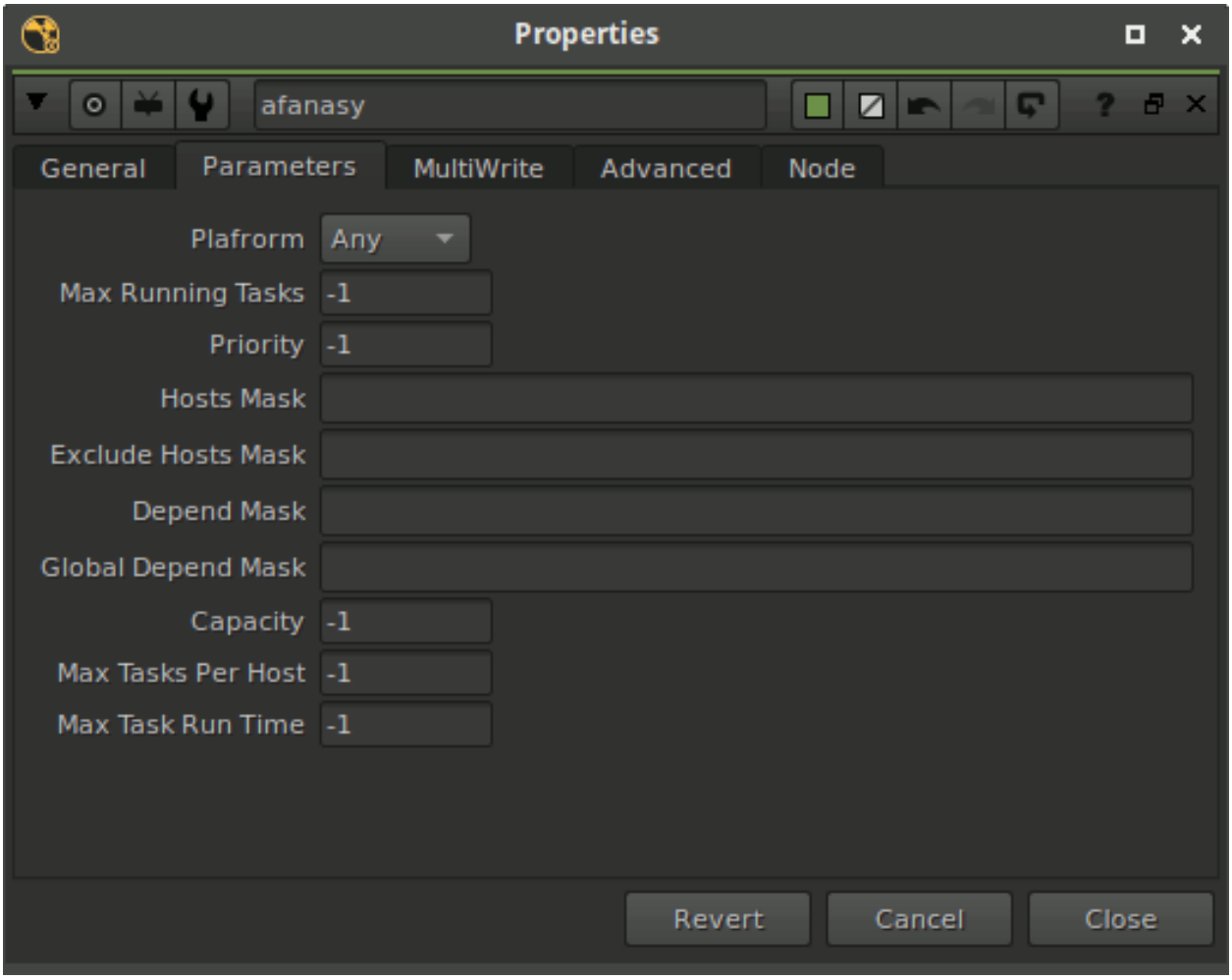


Fig. 48: Afanasy Gizmo Parameters Tab

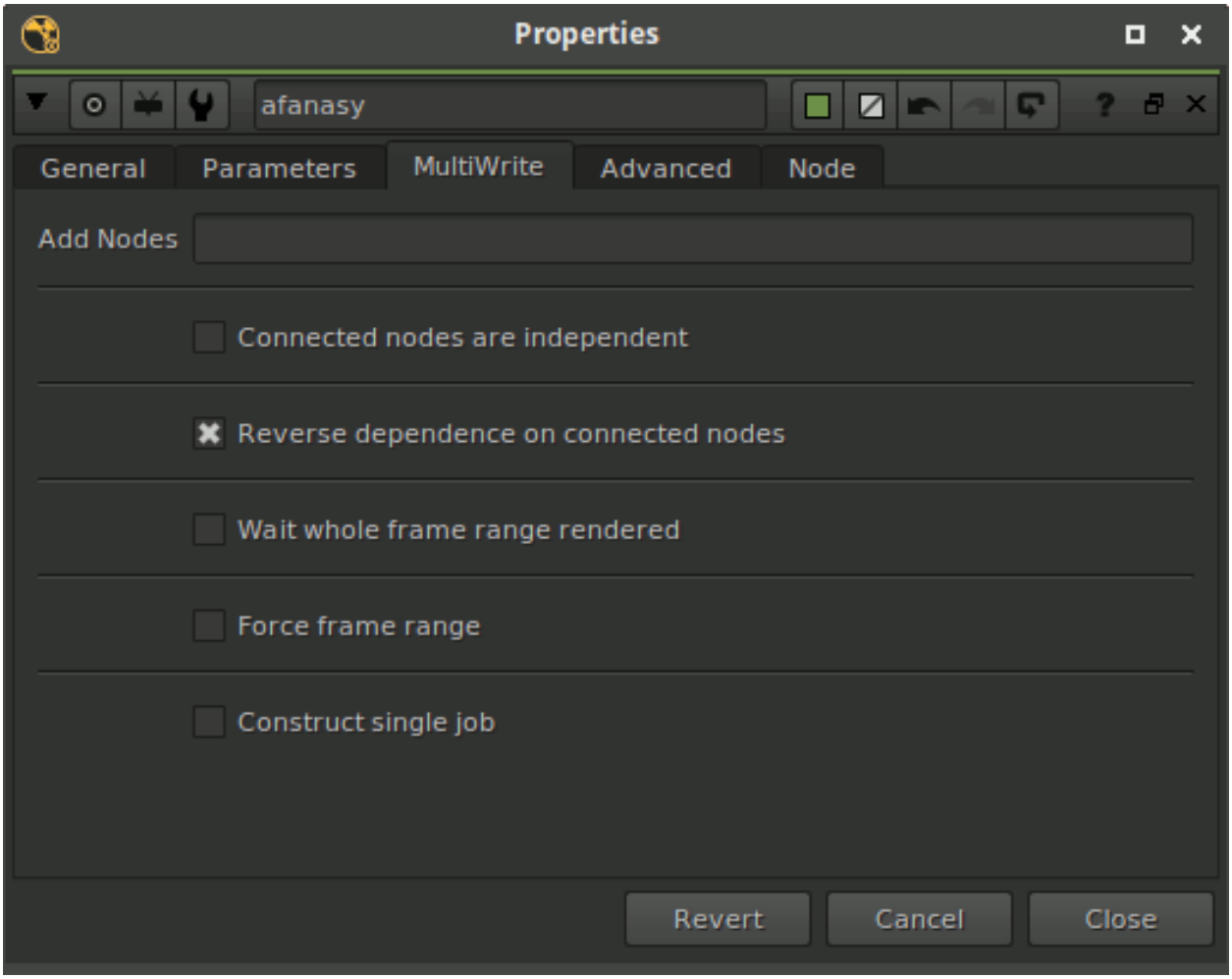


Fig. 49: Afanasy Gizmo MuitiWrite Tab

- **Down stream will wait for whole frame range rendered** Down stream connected node(s) will wait until whole specified frame range will be rendered. If not checked, each frame will be wait only corresponding frame(s) from this node.
- **Force upstream frame settings** All upstream connected nodes will use this node frame range. Connected upstream node can re-force it, if this parameter is checked too.
- **Construct single job from all connected write nodes** Construct a block from each connected 'Write' node and put them into one job. If not checked, each connected 'Write' node will produce a job.

5.10.2.4 Advanced

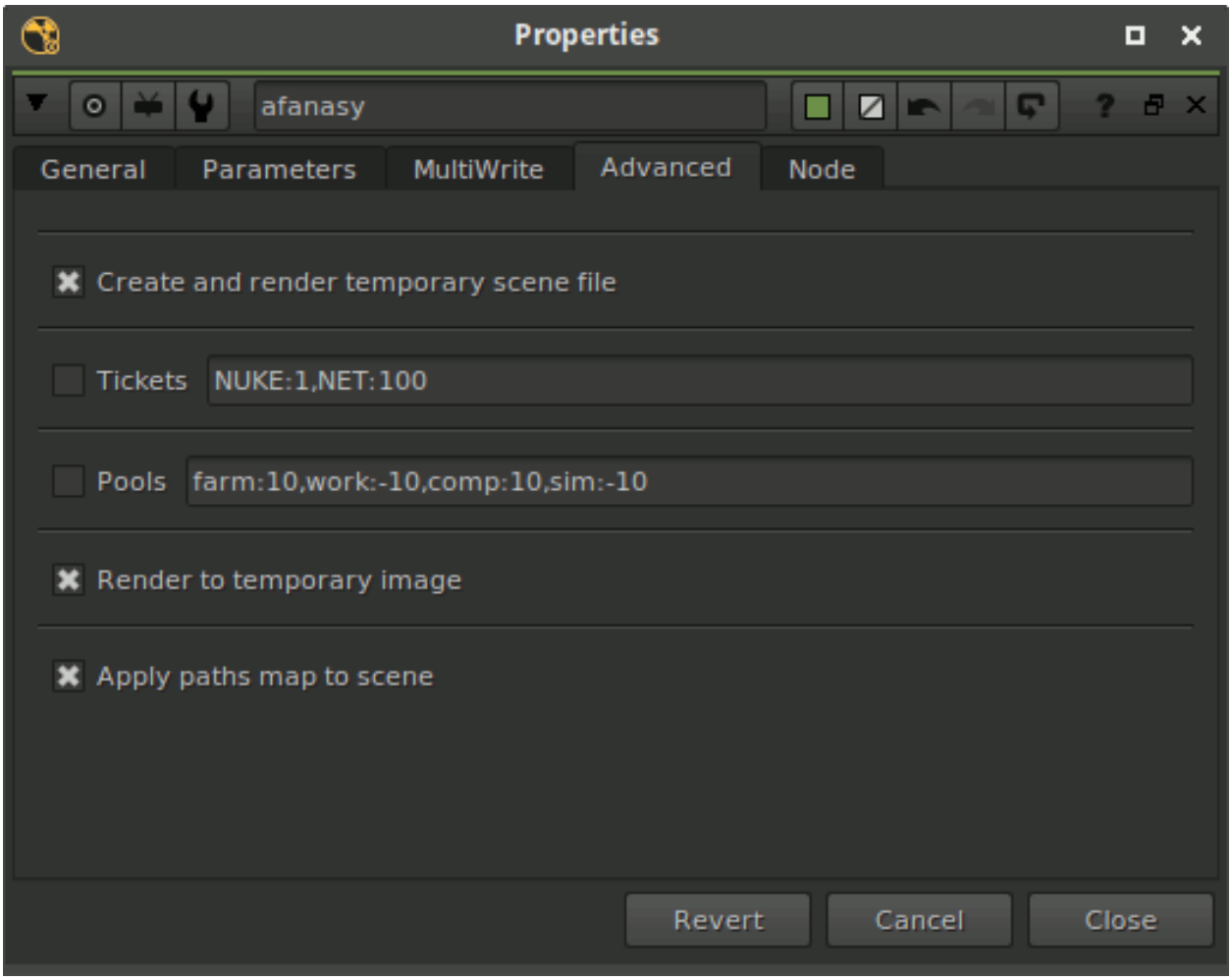


Fig. 50: Afanasy Gizmo Advanced Tab

- **Create and render temporary scene** On job creation, nuke submission script saves scene to temporary name. That temporary scene will be rendered and deleted on a job deletion. This way artist can continue to modify and save working scene. And all frame will be rendered from the same modified scene.
- **Tickets** Job Block tickets counts. Syntax like: `NAME1:count1,NAME2:count2`. Tasks will run only on pools that has enough free tickets. See *Tickets* documentation for details.
- **Pools** Pools that job will run on with priorities. Syntax like: `name1:priority,name2:priority2`. Tasks will prefer pools with a greater priority. See *Pools* documentation for details.

- **Render to temporary image** This can save network traffic, as the entire image will be saved at once. By default Nuke writes a portions of rendered frame.
- **Apply paths map to scene** Transfer all scene files paths from client to server. Using CGRU Path Map you can work and render on different platforms.

5.10.2.5 Environment

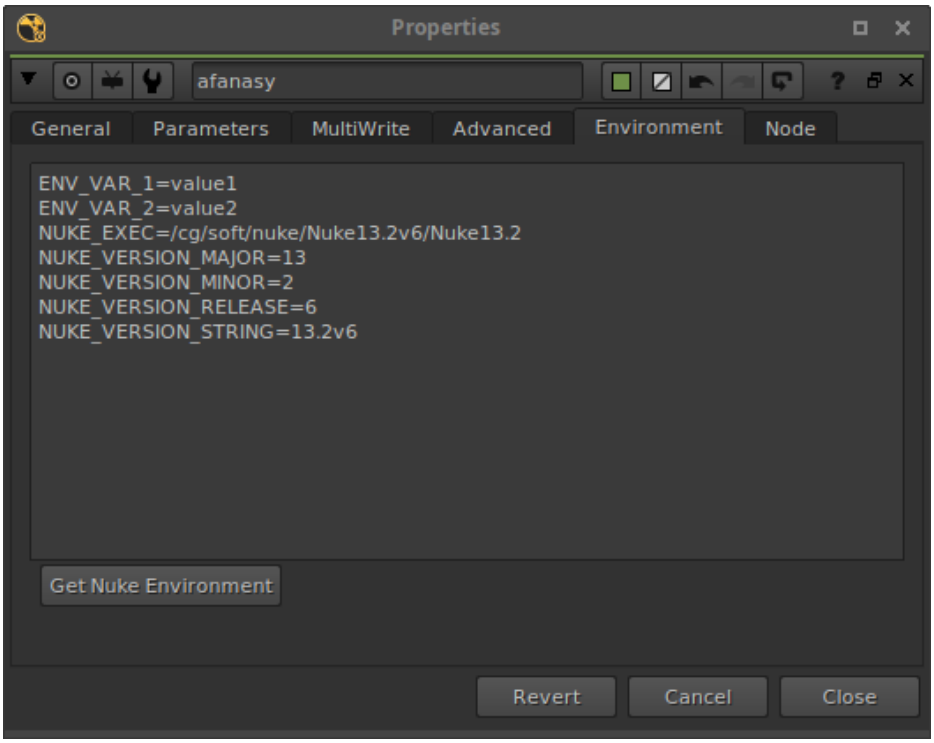


Fig. 51: Afanasy Gizmo Environment Tab

- **Environment variables.** This environment variables will be *added* to task process environment.
- **Get Nuke Environment** Get and store in variables Nuke location and version. It can be used in setup scripts to launch the same version, initialize proper plug-ins.

5.10.3 Complex Job (Precomps)

You can connect one **afanasy** node to several **Write** and **afanasy** nodes. Each connected node will produce a block - an array of tasks (frames) to render. You can specify dependence between connected nodes. This is useful to re-render precomps and the final result in a single job.

5.10.4 Render Selected

You can send to farm selected node(s) using a simple dialog *(F11)*.

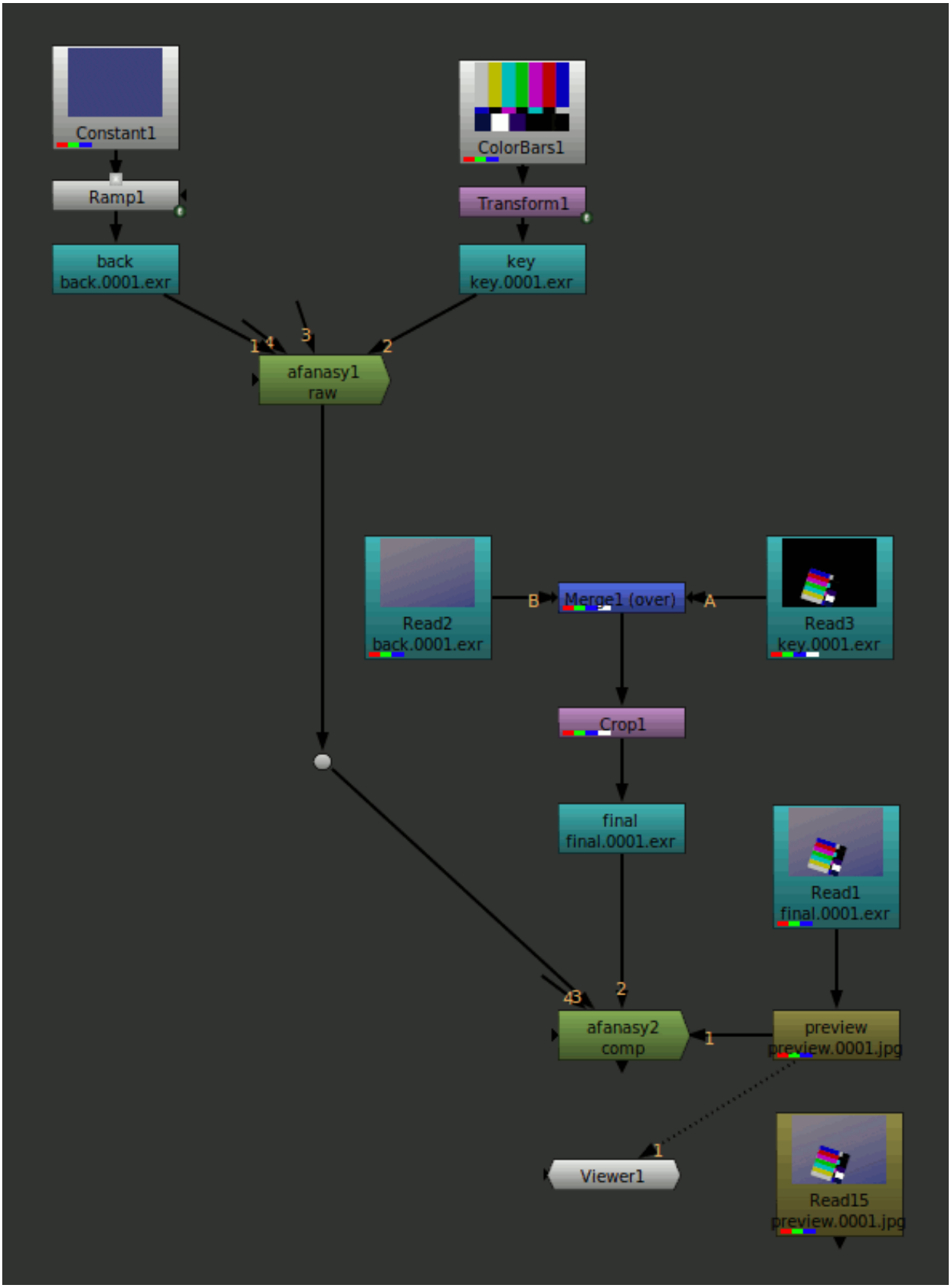


Fig. 52: Complex Node Network

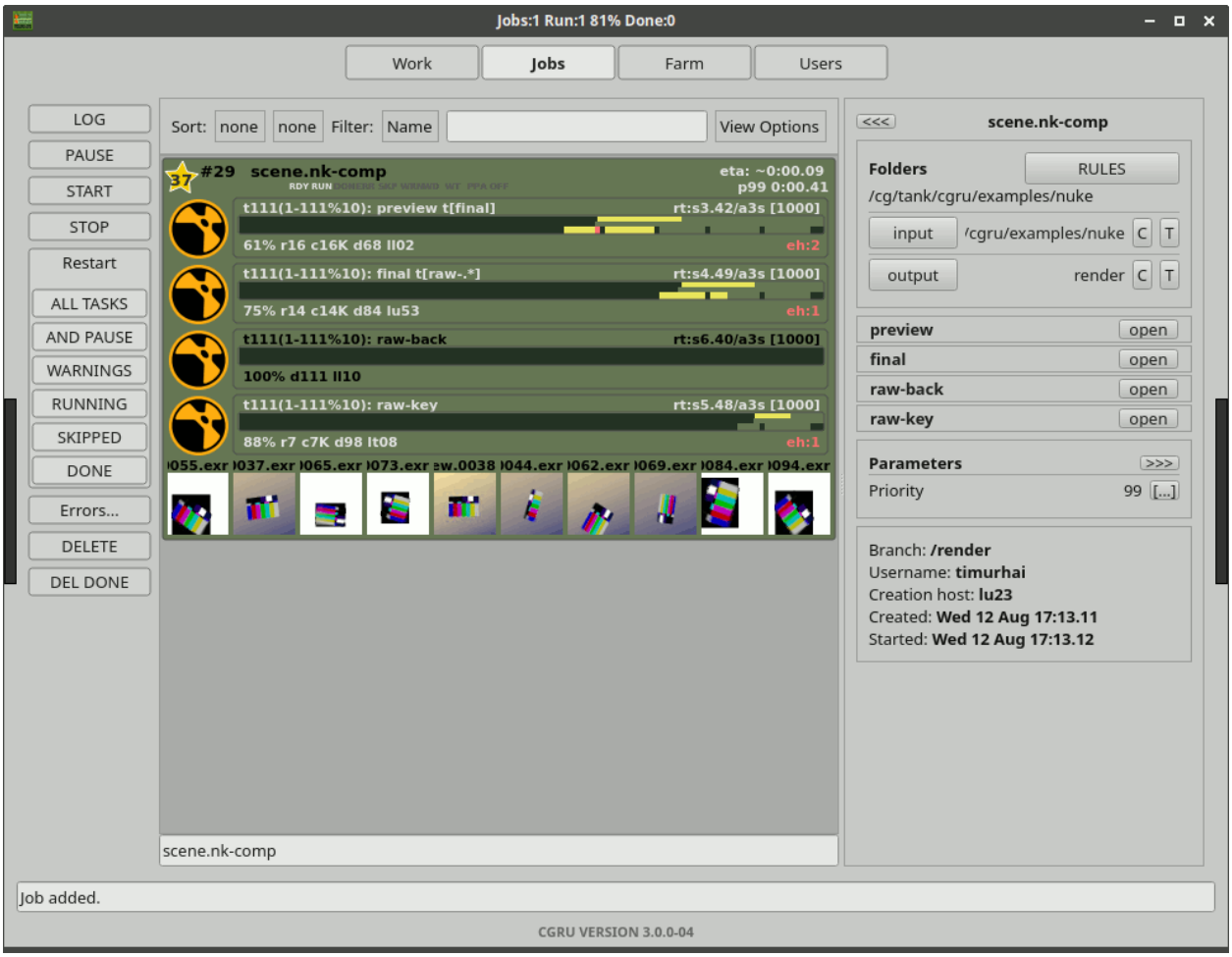
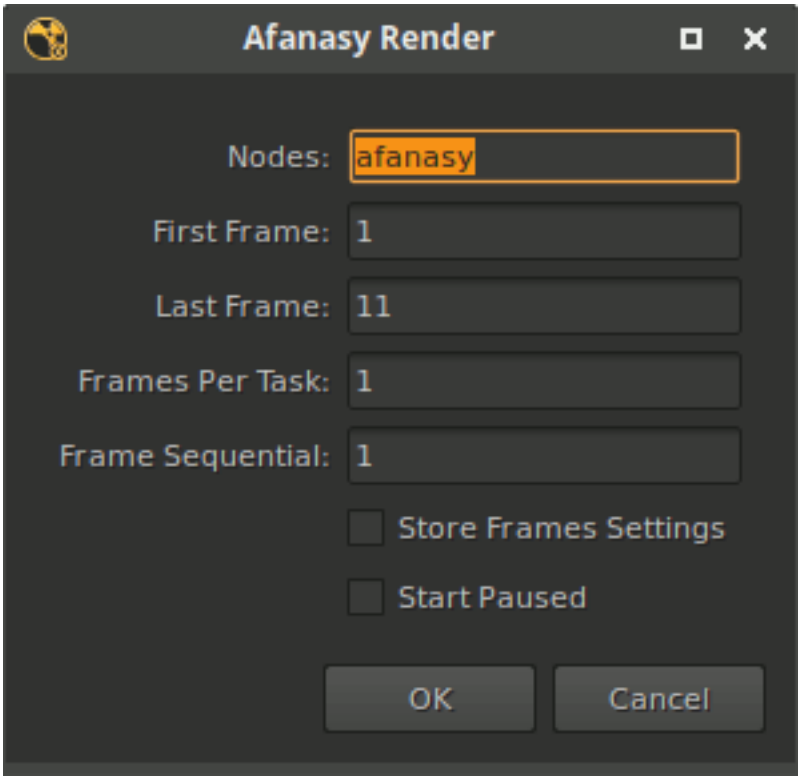
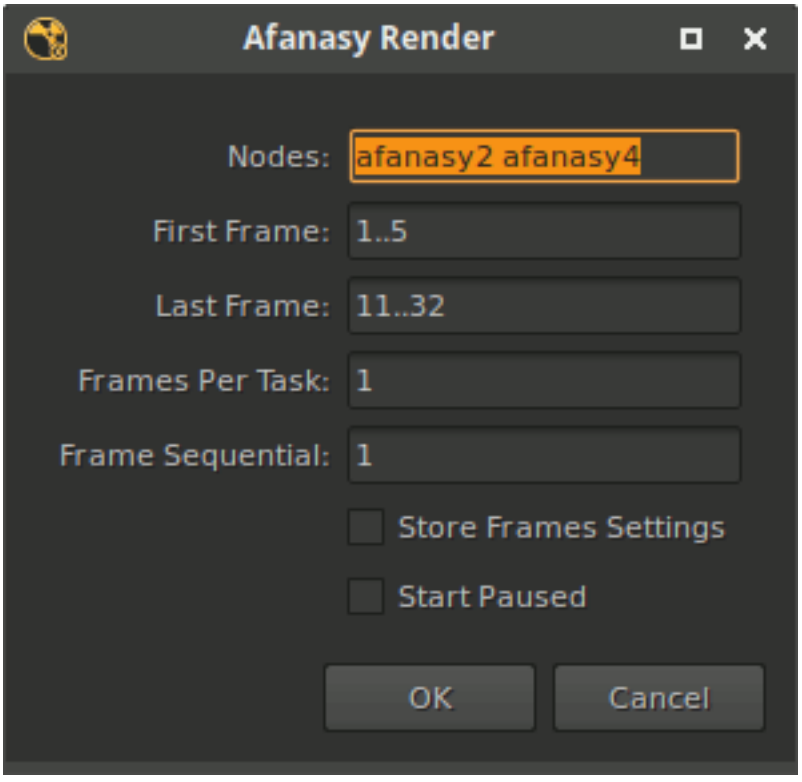


Fig. 53: Complex Job (AfWatch)





`1..5` - two numbers, separated with two points means the lowest and highest value from all selected nodes. Type one number in input field to override frame settings on selected nodes.

- **Nodes** Selected nodes names. You can check and edit selection.

- **First Frame** First frame to render.
- **Last Frame** Last frame to render.
- **Frames Per Task** Number of frames in task.
- **Store Frames Settings** Store frame settings on selected nodes.
- **Start Paused** Job will be created in offline state.

5.10.5 Setup

You can launch nuke from CGRU Keeper and it set all needed environment.

Or you can setup CGRU manually. Setup CGRU and append its `cgru/nuke/plugnis` to `NUKE_PATH`:

```
cd /opt/cgru

source ./setup.sh

export NUKE_PATH="${NUKE_PATH}:${CGRU_LOCATION}/plugins/nuke"
```

5.11 Softimage XSI

5.11.1 Afanasy Window

5.11.1.1 Submission

- **Job Name** Job name. Scene name by default. And pass name if render several passes at once.
- **Pass** Pass(es) to render. You can choose to render current, all or selected pass(es).
- **Take Frame Range From Pass** Render pass with its own frame range, if it is set.
- **Force Pass Frame Range** Render specified below frame range in any case.
- **Start Frame** First frame to render.
- **End Frame** Last frame to render.
- **By Frame** Frames increment.
- **Frames Per Task** Number of frames to render by one task.
- **Simulate** Force not to simulate by setting Play Control to Frame Range.
- **Start Job Paused** Send job in Off-line state.

5.11.1.2 Scheduling

- **Priority** Job Priority parameter, `-1` use default priority.
- **Capacity** Job tasks Capacity, `-1` use default capacity.
- **Max Hosts** Maximum Hosts the job can run on, `-1` means no limit.
- **Task Max Run Time** Maximum Time, `0` means no limit.
- **Hosts Mask** Hostname Mask pattern, job can run on.

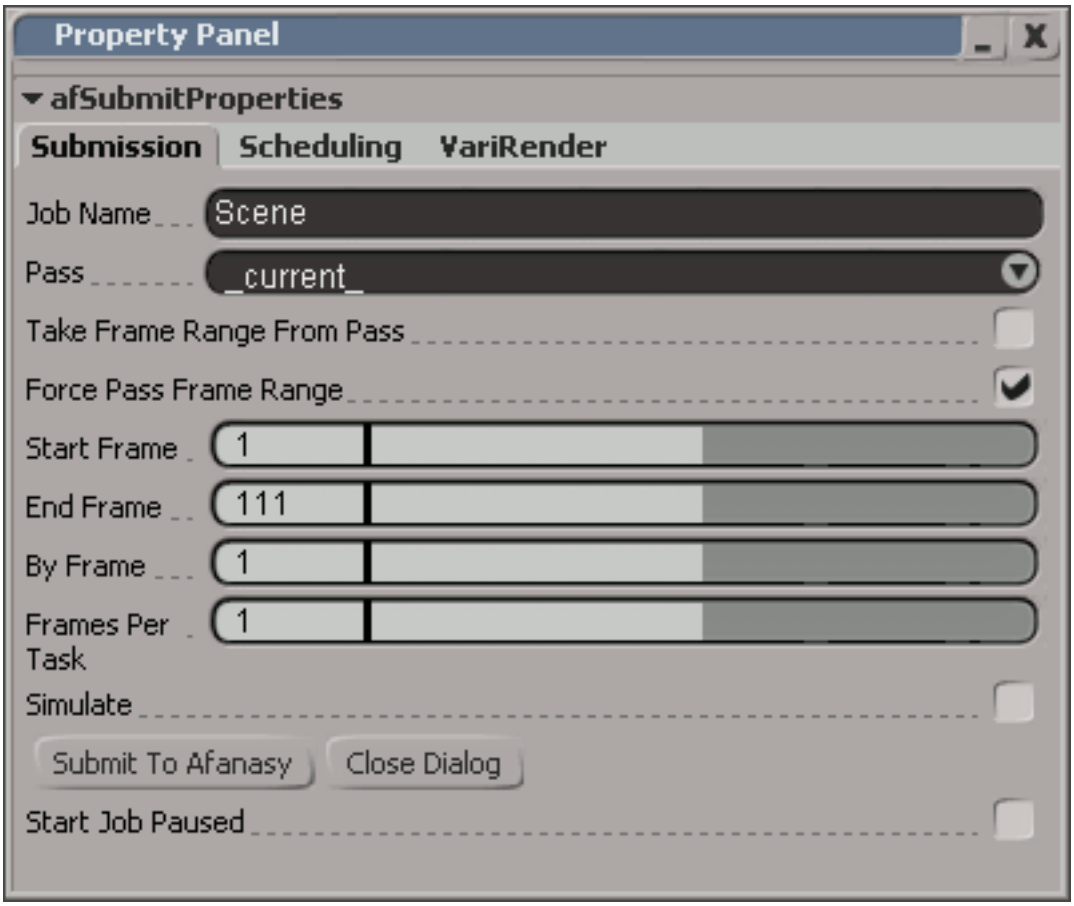


Fig. 54: Submission Tab

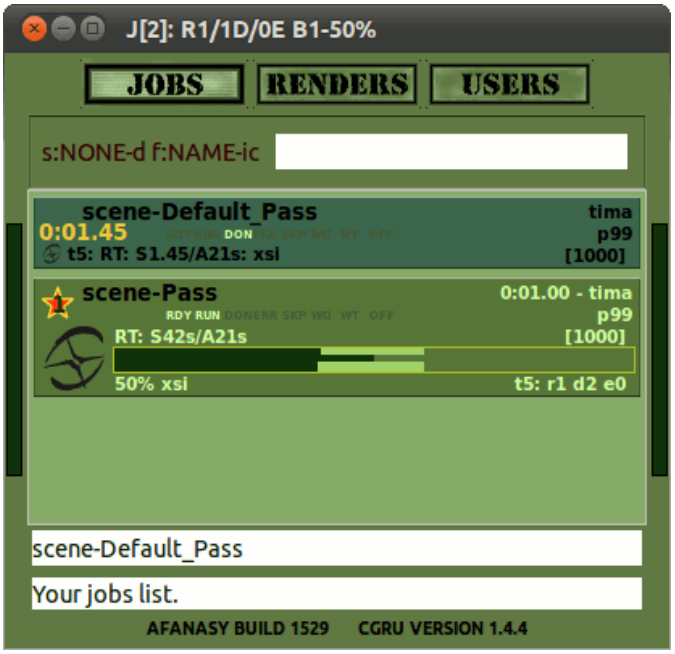


Fig. 55: Job (AfWatch)

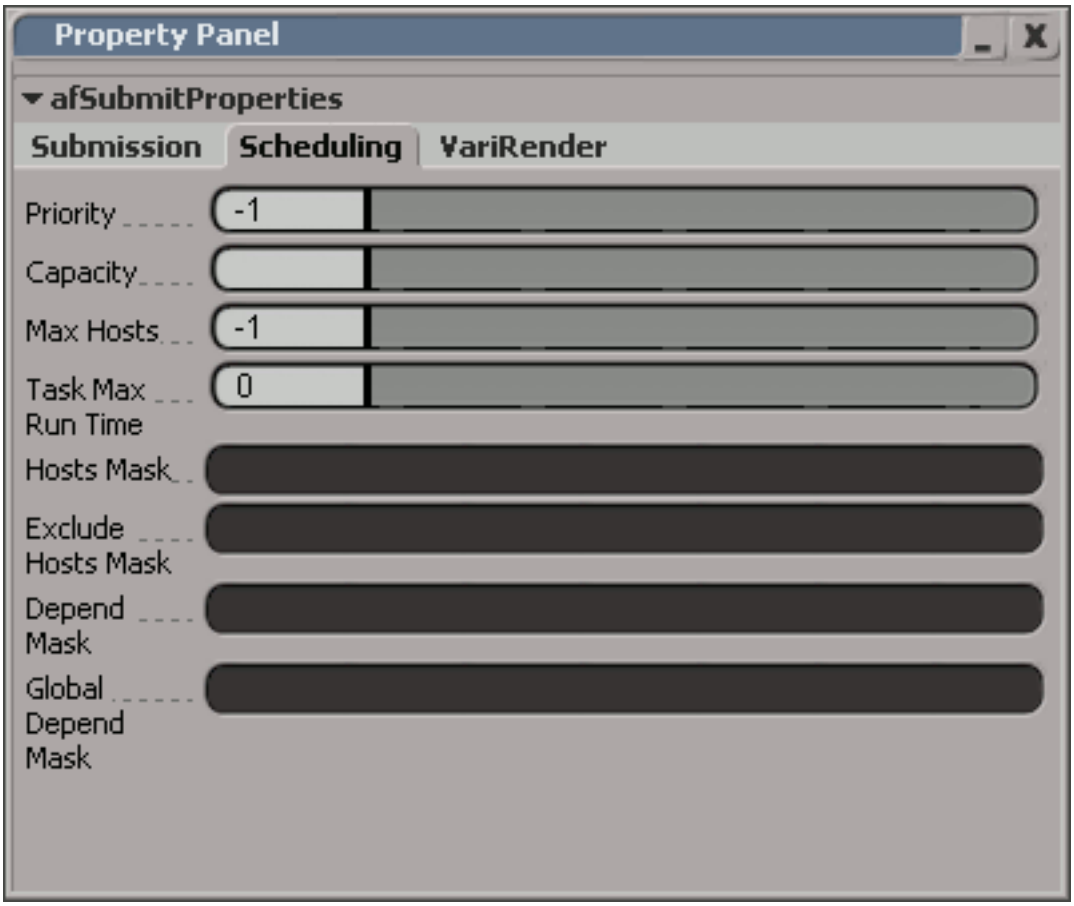


Fig. 56: Scheduling Tab

- **Exclude Hosts Mask** Hostname Mask Exclude pattern, job can not run on.
- **Depend Mask** Job names Depend Mask pattern, to wait other jobs to finish.
- **Global Depend Mask** Job names Depend Mask Global pattern, to wait other jobs from any user to finish.

5.11.1.3 VariRender

VariRender can help you to automatically increment some value and to render scene with different parameters at once. For example particles with different seed, to increase their quantity by compositing.

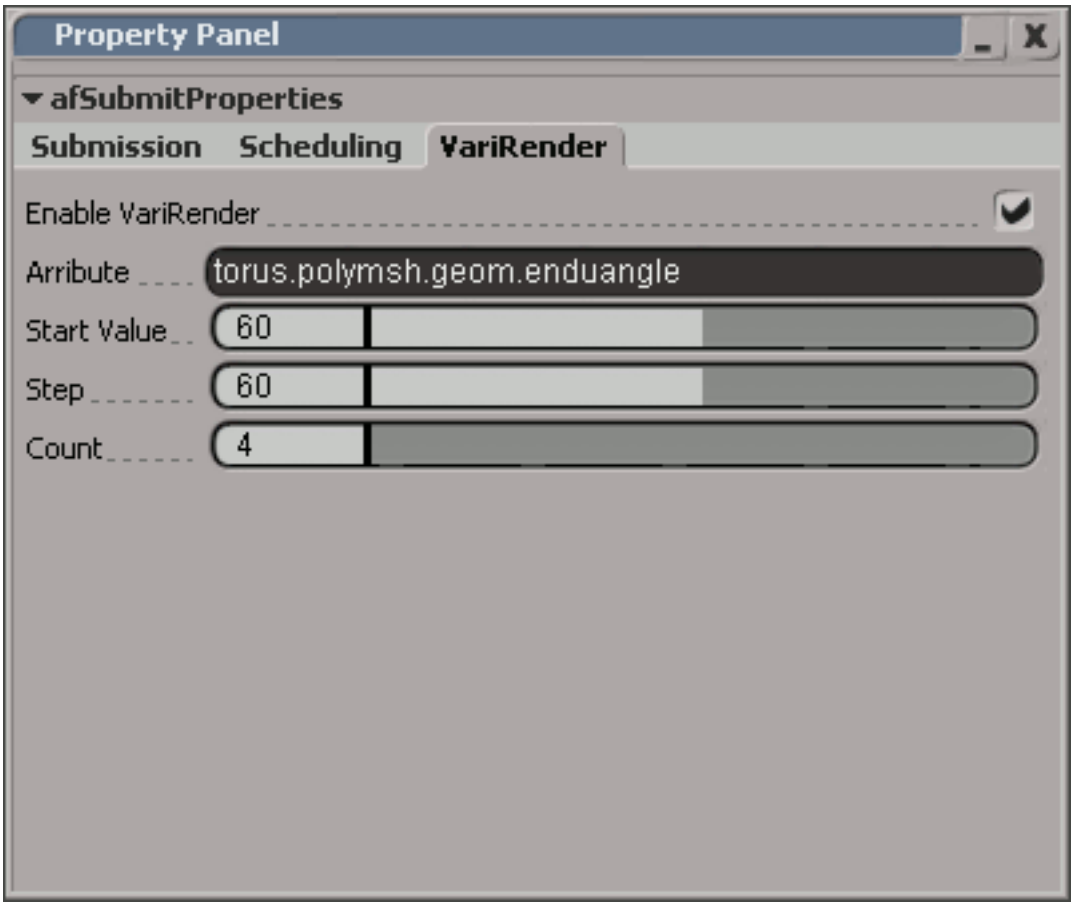


Fig. 57: VariRender Tab

- **Enable VariRender** Turn it on or off.
- **Attribute** Attribute name to variate. Change some object parameter and look script editor, or command line for name.
- **Start Value** First value to start from.
- **Step** Value increment.
- **Count** Quantity of value increments.

This is a job from CGRU examples. Rendering two passes with VariRender.

For each pass it generates a job. For each parameter value it generates a block of tasks.

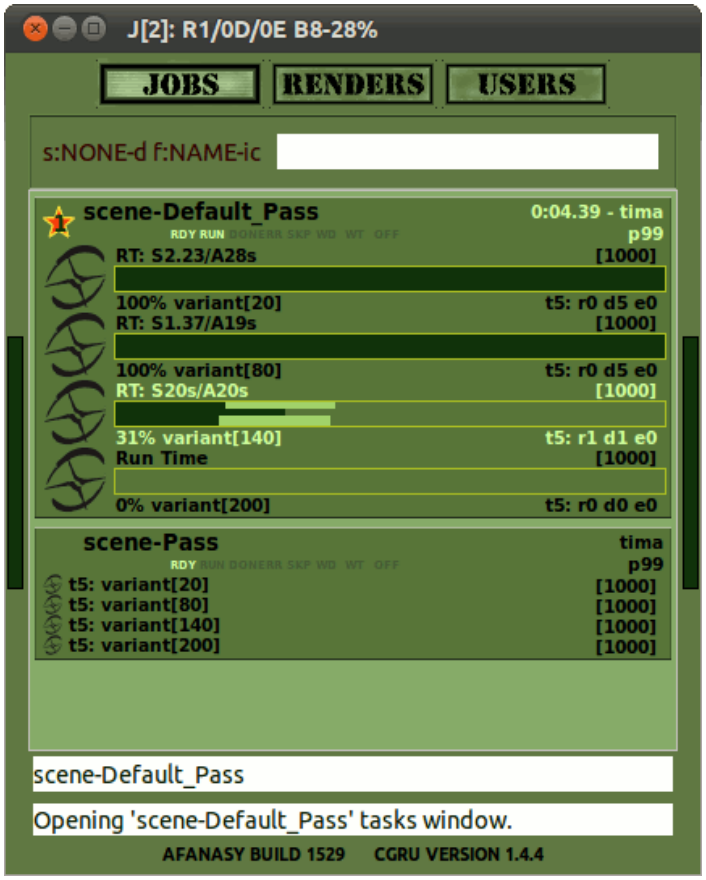


Fig. 58: VariRender Job

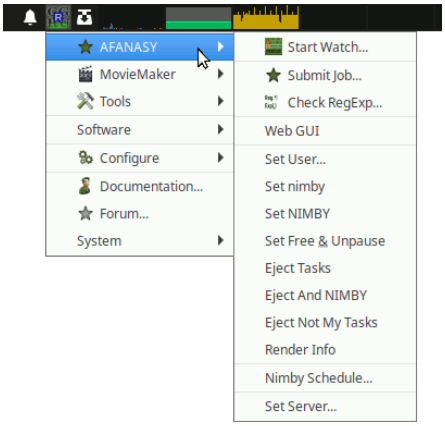
CHAPTER 6

Keeper

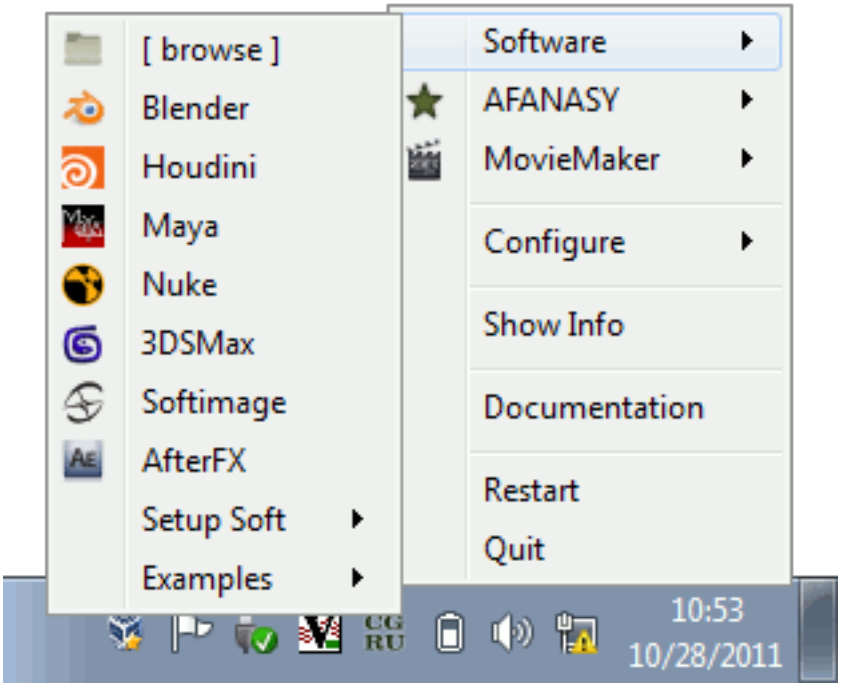
6.1 Description

Keeper is a system tray icon menu to launch and manage CGRU applications, launch and configure other software. It shows local afrender state with R character color, and a local host memory usage, it asks afserver for render info.

Afanasy Menu:



Software Menu:

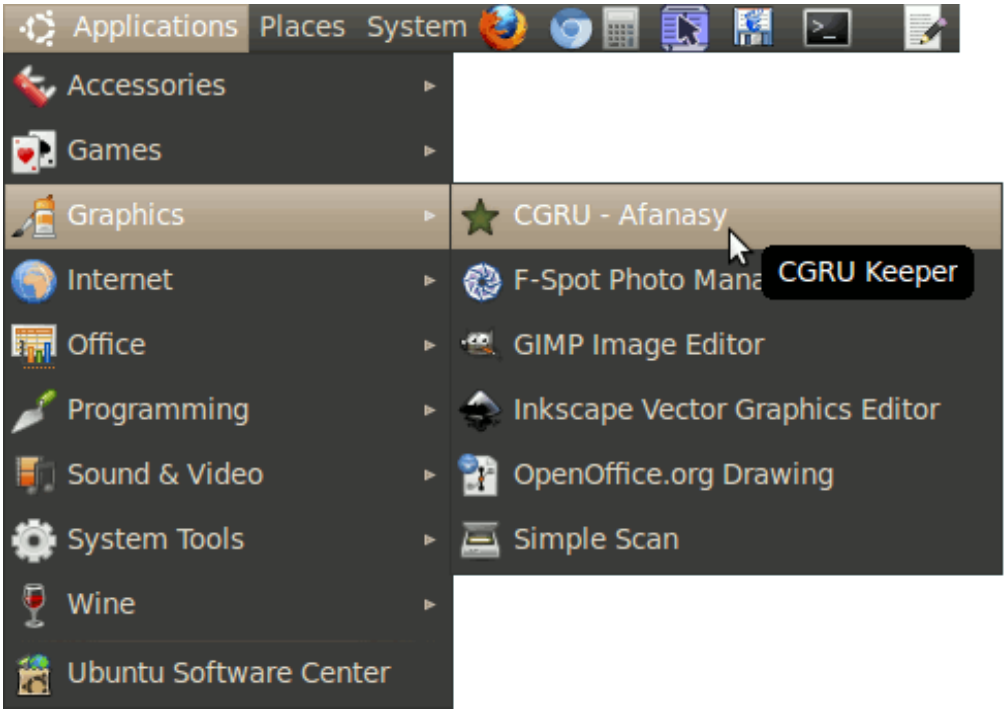


Keeper also can listen port launch commands. Afanasy WebGUI and RULES uses keeper to run something.

6.2 Start Keeper

- **MS Windows**: `cgru\start.cmd`
- **Mac OS X**: `cgru/start.command`
- **Linux**: `cgru/start.sh`

Linux CGRU package creates an item in Applications menu to launch Keeper:



6.3 Launch System commands

Rules and Afanasy Web GUIs launch system commands via keeper. Web browsers can't launch system commands due security reasons. But they can ask Keeper to launch a command.

Rules assumes to use secured HTTPS protocol, as designed to work over internet (external port). Browsers can't mix HTTPS and HTTP protocol for security reasons too. So keeper should use HTTPS protocol for commands. This means that we should setup Keeper HTTPS server.

6.4 HTTPS Server

Keeper uses HTTPServer module and a ssl module to wrap its socket. On start it checks *cgru/utilities/keeper/serverhttps.pem* certificate file for existence. To generate it you can execute:

```
openssl req -new -x509 -keyout serverhttps.pem -out serverhttps.pem -days 3656
-nodes
```

If this file exists, Keeper starts to listen 44443 port.

The next you should create a security exception in a browser for this certificate. For this you should open Keeper test page and confirm a security exception. To open Keeper test page you can use this Keeper tray menu item: *Configure -> HTTPS Server...*

CHAPTER 7

AfStarter

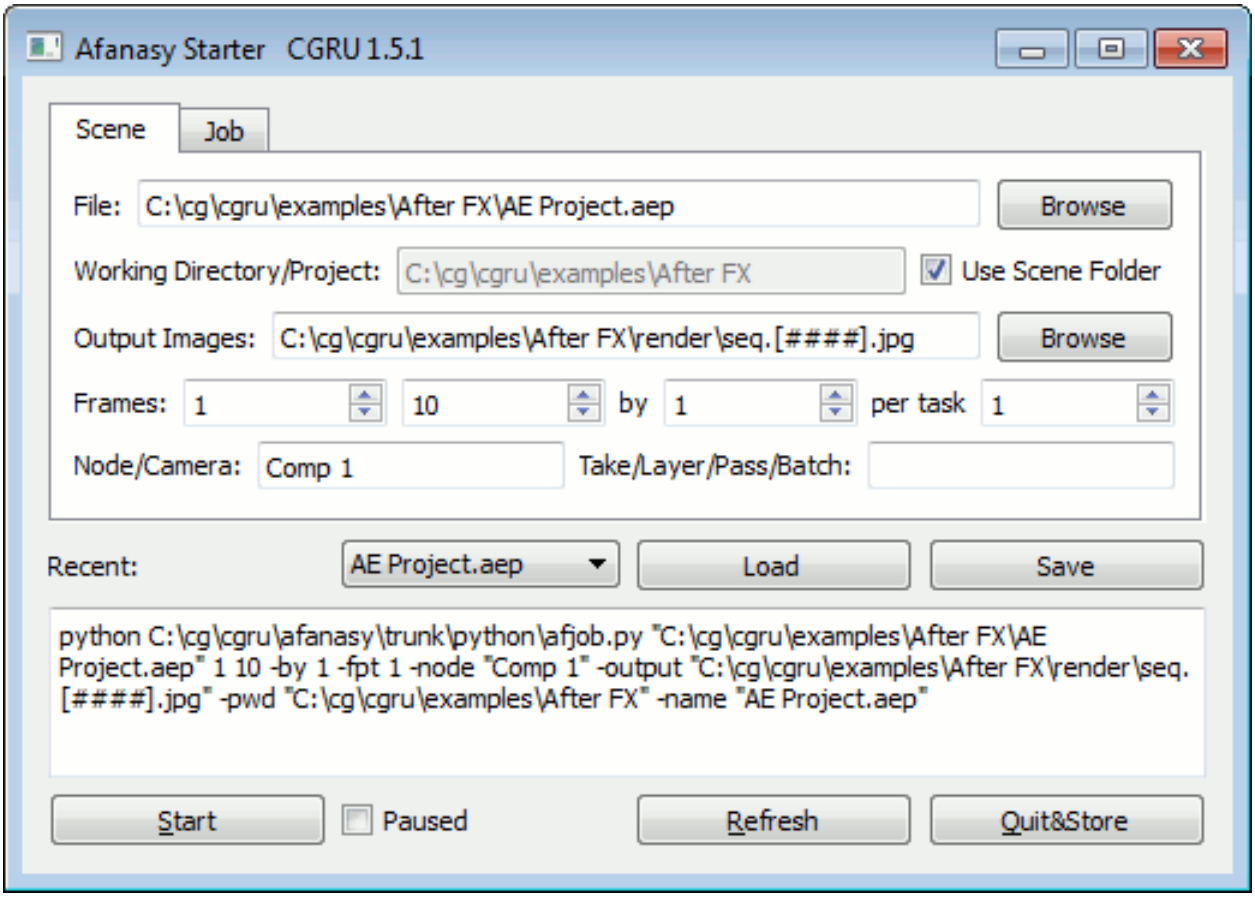
AfStarter is a standalone dialog to submit a job to Afanasy.

7.1 Supported software

- Adobe After Effects
- Autodesk 3D Studio Max
  - Software
  - VRay
- Autodesk Maya
  - Software
  - 3DeLight
  - VRay
  - Arnold
  - Mental Ray
  - Redshift
- Autodesk SoftImage XSI
- Blender
  - Internal
  - Cycles
- Clarisse iFX
- Cinema 4D
- Fusion

- Isotropix Clarisse
- Natron
- SideFX Houdini
  - HBatch *(hython)*
  - Mantra *(standalone)*
- The Foundry Nuke

7.2 Scene Settings

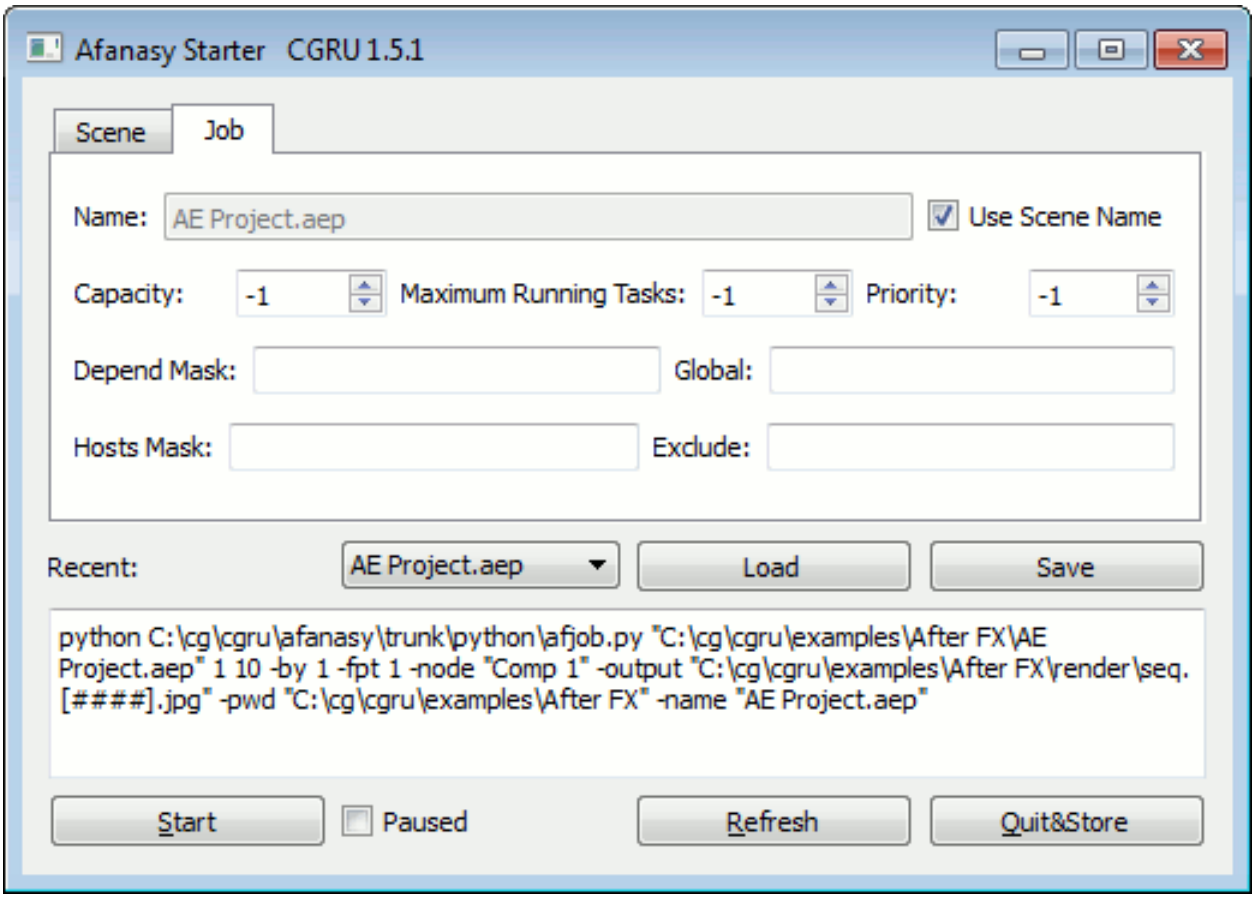


Scene Parameters:

- **File**: Scene file path. IFD files sequence for Mantra.
- **Working Directory/Project**: Tasks process working folder, default is scene folder. Project path for Maya.
- **Output Images**: Some software allows to override output images in command line arguments.
- **Frames**: Frame range to render, 'by' - frames step or increment, 'per task' - number of frames in one task.
- **Node/Camera**: Houdini ROP, Nuke write node, Max, Maya camera to render, After FX composition.
- **Take/Layer/Pass/Batch**: Houdini take, SoftImage pass, Maya layer, Max batch to render, After FX render settings template.

- **Recent**: Store recent sent jobs settings.
- **Start**: Submit job to Afanasy server. It can be started in off-line state ("paused").

7.3 Afanasy Job Settings



Afanasy Parameters:

- **Name**: Job name, scene file name is used by default.
- **Capacity**: Tasks capacity attribute value. '-1' means use the default.
- **Priority**: Job order in user jobs list.
- **Maximum Running Tasks**: Maximum number of tasks can be running at the same time.
- **Depend Mask**: Job(s) name pattern to wait to. Global: wait job(s) of any user.
- **Hosts Mask**: Host(s) name pattern to run on. Exclude: not to run on such hosts.

CHAPTER 8

Regular Expressions

This is a standard Regular Expressions. They are realized in almost all languages like C, C++, Python, JavaScript, bash, Perl, PHP, Ruby and others. Some times such expressions called "Perl-like" as they were popularized within Perl at first.

AFANASY uses this expressions in hosts mask, depend masks and other patterns.

Examples Expression	Some Matched Names	Description
r1|r7|r12	r1 r7 r12	Only specified machines ("|" - means "or")
r1.	r11-r19 r1a-r1z	Names starts with "r" plus one any character ("." - any (one) character)
r.*	r0 r1 rN rnd roman ...	All computers which name starts with "r" ("*" - any number or characters)
r1.*	r1 r10 r11 r102 r1asd ...	All computers which name starts with "r1"
r1[178]	r11 r17 r18	Only specified machines, [1 or 7 or 8]
r1[1-5]	r11 r12 r13 r14 r15	Only specified machines, [from 1 to 5]
r0.|r1[1-5]	r00-r09 r11-r15	All r0# and from r11 to r15

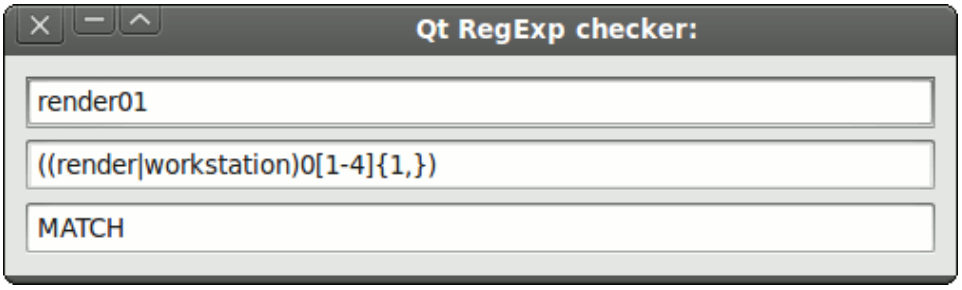
Detailed documentation, examples, constructors and testers:

- http://en.wikipedia.org/wiki/Regular_expression
- http://www.google.com/search?q=Regular+Expressions
- http://www.google.com/search?q=Regular+Expressions+Examples
- http://www.google.com/search?q=Regular+Expressions+Tester

8.1 RegExp Checker

A simple dialog to check regular expressions provided with CGRU.

You can launch it from keeper: *Afanasy* -> *Check RegExp...*



Any render01-render04 or workstation01-workstation04.

CHAPTER 9

Rules

Rules is a Web based CG project tracker.

Note: Documentation is not finished.

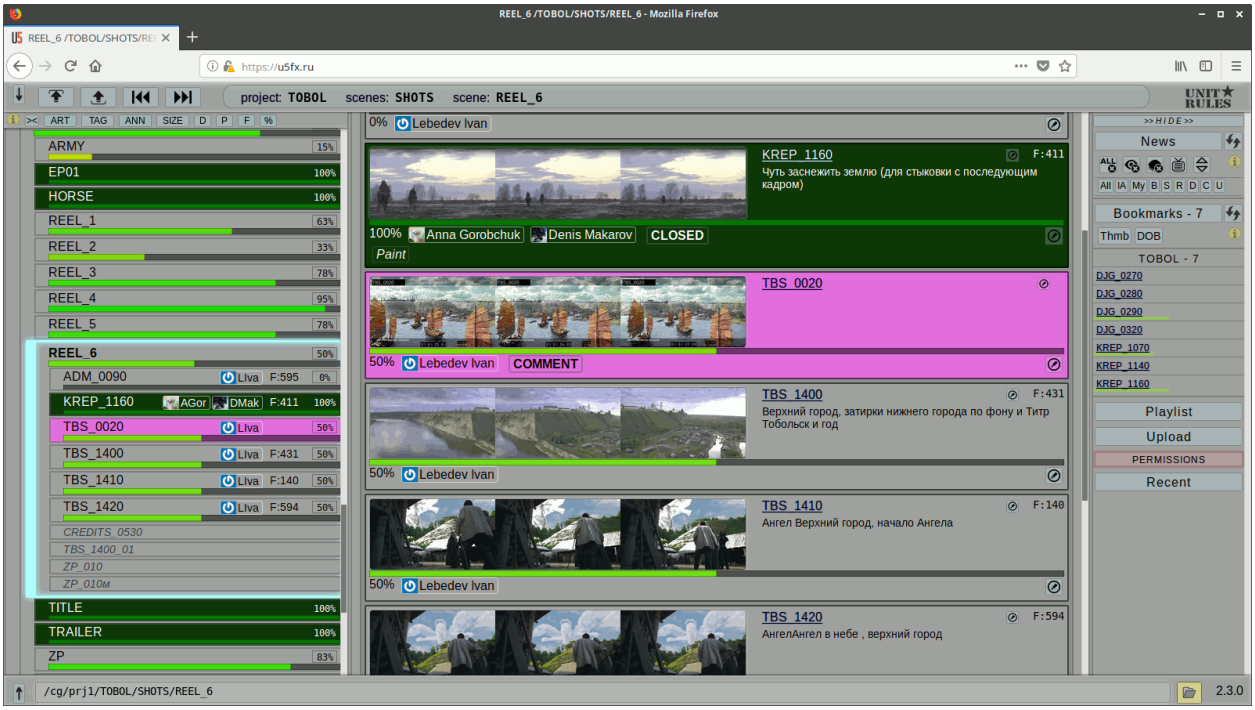


Fig. 1: Scene shots view.

For Rules server hosting you need a Linux server with Apache and PHP.

9.1 Player

You can play and annotate images sequences within Rules Player.

9.1.1 Examples

CHAPTER 10

Examples

CGRU examples are located in `cgru/examples` directory.

General tools for specific soft are located in `cgru/plugins/[software_name]`.

Afanasy Tools for specific soft are located in `cgru/afanasy/plugins/[software_name]`.

Run examples by executing script from it example directory.

Launch Blender example:

```
cd cgru/examples/blender
./start_blender.sh
```

MS Windows You can double click application launch script in `cgru\start` directory, or create link to it, to launch example run its script.

Example for Blender: `cgru\examples\blender\start_blender.cmd`

CHAPTER 11

How To

11.1 Appending new tasks/blocks to an existing job

It is possible to add new tasks or blocks to a job that has already been sent to the farm. One can do it in several manners.

11.1.1 JSON API

The JSON API for actions features the two following operations:

append_tasks which takes a tasks list following the very same spec as when submitting new jobs, e.g.:

```
{
 "action":
 {
  "user_name"   : "elie",
  "host_name"   : "my_pc",
  "type"        : "jobs",
  "ids"         : [3],
  "block_ids"   : [0],
  "operation"   :
  {
   "type"       : "append_tasks",
   "tasks"      :[
    {
        "name"    : "Extra Process A",
        "command" : "python -c \"print('Process EXTRA task A')\""
    },
    {
        "name"    : "Extra Process B",
        "command" : "python -c \"print('Process EXTRA task B')\""
    }
    ]
```

(continues on next page)

(continued from previous page)

```
  }
 }
}
```

append_blocks which takes a blocks list also following jobs submission spec, e.g.:

```
{
 "action":
 {
  "user_name"  : "elie",
  "host_name"  : "my_pc",
  "type"       : "jobs",
  "ids"        : [3],
  "operation"  :
  {
   "type"      : "append_blocks",
   "blocks"    :[
    {
      "name"              : "New numeric block",
      "tasks_name"        : "frames @#@-@#@",
      "service"           : "generic",
      "parser"            : "generic",
      "flags"             : 1,
      "frame_first"       : 1,
      "frame_last"        : 100,
      "frames_per_task"   : 10,
      "frames_inc"        : 2,
      "command"           : "python -c \"print('Process frames @#@-@#@')\"",
      "working_directory" : "E:\\tmp"
    },
    {
      "name"              : "New non numeric block",
      "tasks_name"        : "frames @#@-@#@",
      "service"           : "generic",
      "parser"            : "generic",
      "working_directory" : "E:\\tmp",
      "tasks"             : [
      {
        "command" : "python -c \"print('Process task A')\""
      },
      {
        "command" : "python -c \"print('Process task B')\""
      }
      ]
    }
    ]
  }
 }
}
```

11.1.2 Python af module

Two command methods are available in the `Cmd` object in `af.py`.

appendBlocks(jobId, blocks)

Example:

```
import af

block = af.Block('generic', 'generic')
block.setCommand("python -c \"print('Process frames @#@-@#@')\"")
block.setNumeric(1, 100, 10)

cmd = af.Cmd()
print(cmd.appendBlocks(3, [block]))
```

appendTasks(jobId, blockId, tasks)

Example:

```
import af

task = af.Task('test')
task.setCommand("python -c \"print('Process task A')\"")

cmd = af.Cmd()
print(cmd.appendTasks(3, 0, [task]))
```

11.1.3 Python afcmd module

Job.appendBlocks(blocks)

Example:

```
import afcmd

block = af.Block('generic', 'generic')
block.setCommand("python -c \"print('Process frames @#@-@#@')\"")
block.setNumeric(1, 100, 10)

job = afcmd.getJob(3)
print(job.appendBlocks([block]))

blockCopy = job.blocks[0]
print(job.appendBlocks([blockCopy]))
```

Block.appendTasks(tasks)

Example:

```
import afcmd

task = af.Task('test')
task.setCommand("python -c \"print('Process task A')\"")

job = afcmd.getJob(3)
block = job.blocks[0]
print(block.appendTasks([tasks]))
```

11.1.4 Known limitations

11.1.4.1 Numeric block

It does not makes sense to append tasks to numeric blocks, only non-numeric block can have tasks appended.

11.1.4.2 afwatch

When appending new blocks to a job that is opened in afwatch, one must reopen the job window to see the update.

This is a set of guides for specific tasks.

CHAPTER 12

Coding Rules

- Indentation: TAB. You can vary tab length, default 4 spaces length is normal. Except Python.
- Python indentation: 4 SPACES.
- Code alignment: SPACES, it should not break on various tab length.
- Variable names: `variable_name`.
- Function names: `functionName`.
- Class names: `ClassName`.
- Use prefixes to make code more readable:
  - `i_`: Input variables.
  - `o_`: Output variables.
  - `m_`: Class members.
  - `ms_`: Static class members.
  - `g_`: External variables.
  - `v_`: Virtual functions.
- Do not use `!` as NOT, it is not noticeable while code passing view, much more easy to notice `false ==`
- Use `const &` to pass complex types as function parameters to not to copy class instance.

Example:

```
void someFunction(const SomeClass & i_some_class, int i_some_type)
{
    ...
}

// Function with a long parameters list:
bool someOtherFunction(
        std::string & o_status,
```

(continues on next page)

(continued from previous page)

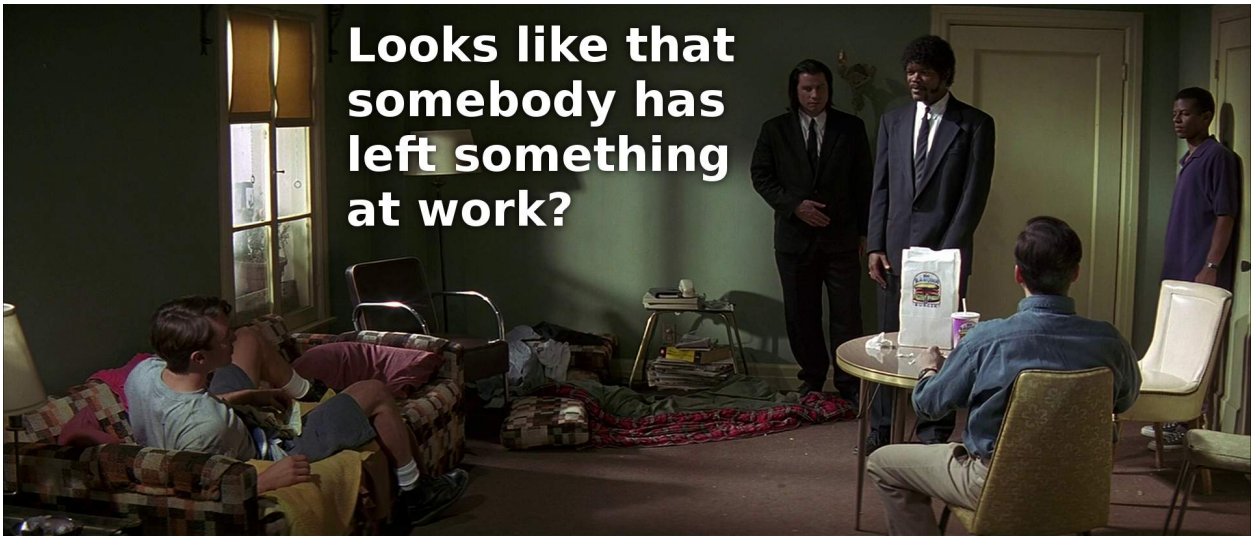
```
        const std::string & i_param1,
        const std::string & i_param2,
        const std::string & i_param3,
        const std::string & i_param4
    )
{
    ...

    if (false == variable_name)
    {
        o_status = "error";
        return false;
    }

    return true;
}
```

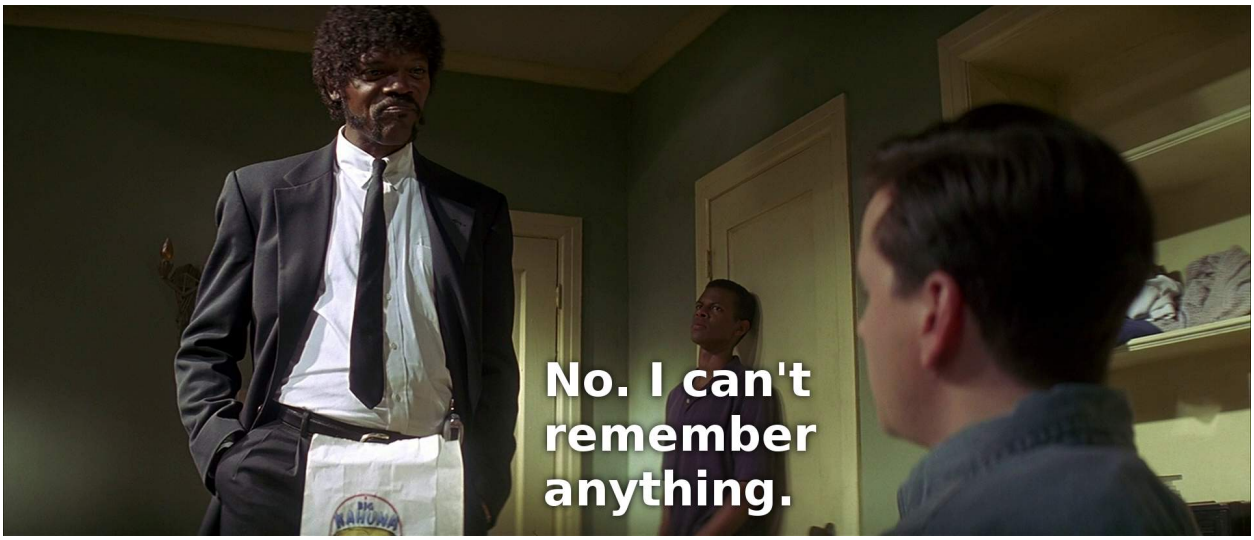
Note: Afanasy code is not all written by this rules. But better to write new code by this rules, and may be replace near parts.

Warning: Old code or new code written not by rules can be re-factored any time.

CHAPTER 13

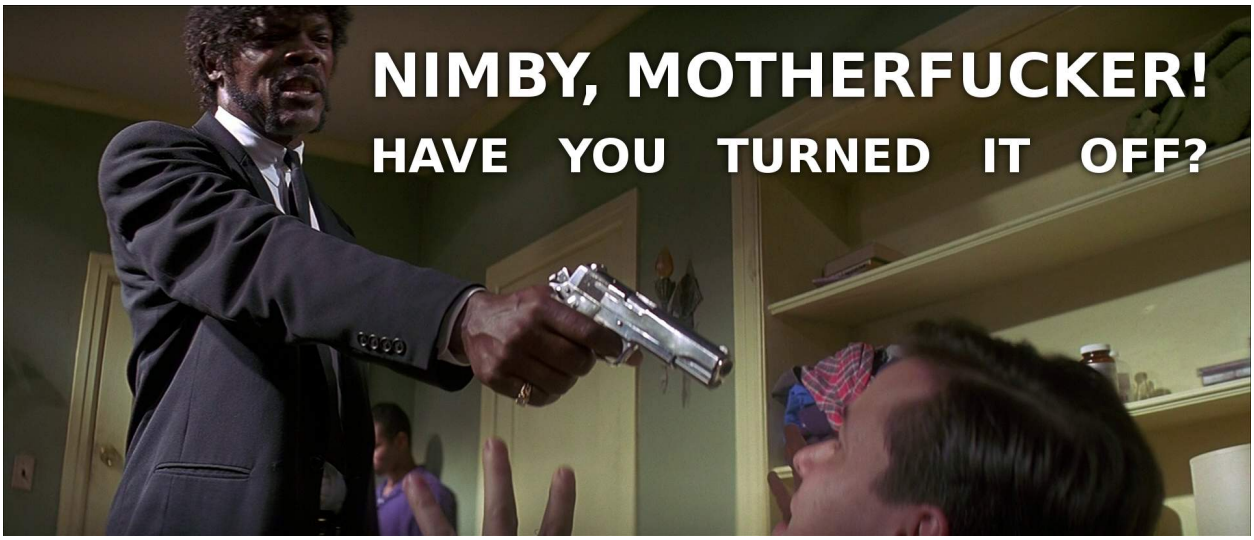
The Nimby Situation




No. I can't
remember
anything.


May be,
nimby?


What?



13.1 Spending much time with machines do not forget that we are humans

13.2 Do not let producers to torture you much

CHAPTER 14

License

CGRU is licensed under the GNU Lesser General Public License (LGPL) license.

It includes some tools and libraries:

	License
ImageMagick	*no limitations*
FFMpeg	LGPL
PostgreSQL	A liberal Open Source license, similar to the BSD or MIT licenses.
PySide	LGPL
PyQt	GPL or Commercial. Will be used only if PySide is not installed.
Python	*no limitations*
Qt	LGPL

CHAPTER 15

About

CGRU is a CG tools pack includes AFANASY - Free Open Source Render Farm Manager.

Name “CGRU” - came from CG - Rules. It has two meanings: computer graphics principles and computer graphics is a cool thing.

Name “Afanasy” - came from a Greek name meaning “immortal”.

http://en.wikipedia.org/wiki/Afanasy

15.1 Companies

- **Algous** | http://www.algousstudio.ru
- **Artjail** | https://www.artjail.com/
- **Asymmetric VFX** | http://www.a-vfx.ru/
- **Cinnamon VFX** | http://cinnamon.com.ua
- **EMG** | http://emg.fm
- **Ghost A/S** | http://ghost.dk
- **Istudios Visuals** | http://www.istudios.se
- **Kiev Postproduction** | http://www.kievpostproduction.com
- **Main Road Post** | https://mrpost.ru
- **Overmind Studios** | http://www.overmind-studios.de
- **Platige Image** | https://www.platige.com
- **Post Kino** | https://postkino.ru/
- **Postmodern** | http://postmodern.com.ua/
- **Redchillies VFX** | http://redchilliesvfx.com/

- **R-Studios** | http://www.r-studios.ru
- **Rise FX** | https://risefx.com
- **Terminal FX** | http://terminalfx.com
- **TurboRender** | https://turborender.com
- **Unit Five** | https://u5fx.ru
- **Virtual Republic** | https://virtualrepublic.org/

15.2 Projects

- **Wanted** (2008) http://www.imdb.com/title/tt0493464/
- **Obitaemyy ostrov** (2008) http://www.imdb.com/title/tt0972558/
- **Chernaya Molniya** (2009) http://www.imdb.com/title/tt1569364/
- **High Security Vacation** (2009) http://www.imdb.com/title/tt1438463/
- **Death Race 2** (2010) http://www.imdb.com/title/tt1500491/
- **Blue Crush 2** (2011) http://www.imdb.com/title/tt1630626/
- **Elena** (2011) http://www.imdb.com/title/tt1925421/
- **Vykrutasy** (2011) http://www.imdb.com/title/tt1682187/
- **Scorpion King: Battle for Redemption** (2011) http://www.imdb.com/title/tt1781896/
- **The Darkest Hour** (2011) http://www.imdb.com/title/tt1093357/
- **Visotsky** (2011) http://www.imdb.com/title/tt2116974/
- **Project: S.E.R.A.** (2012) http://www.imdb.com/title/tt2083304/
- **An Enemy to Die For** (2012) http://www.imdb.com/title/tt1904887/
- **Shpion** (2012) http://www.imdb.com/title/tt2321517/
- **Step Up Revolution** (2012) http://www.imdb.com/title/tt1800741/
- **Universal Soldier: Day of Reckoning** (2012) http://www.imdb.com/title/tt1659343/
- **The Lords of Salem** (2012) http://www.imdb.com/title/tt1731697/
- **Dead in Tombstone** (Video 2012) http://www.imdb.com/title/tt2268419/
- **Unforgotten Shadows** (2013) http://www.imdb.com/title/tt2245906/
- **Pawn Shop Chronicles** (2013) http://www.imdb.com/title/tt1741243/
- **Metro** (2013) http://www.imdb.com/title/tt2649128/
- **The Zero Theorem** (2013) http://www.imdb.com/title/tt2333804/
- **Project S.E.R.A** (2013) http://www.imdb.com/title/tt2806258/
- **Poddubnyy** (2013) http://www.imdb.com/title/tt3767246/
- **Bistree Chem Kroliki** (2013) http://www.imdb.com/title/tt3451498/
- **Kukhnya v Parizhe** (2014) http://www.imdb.com/title/tt3711466/
- **Le grimoire d'Arkandias** (2014) http://www.imdb.com/title/tt3022526/

- **Nobody from Nowhere** (2014) http://www.imdb.com/title/tt3161960/
- **Papa ou maman** (2015) http://www.imdb.com/title/tt4080672/
- **Robin des Bois, la véritable histoire** (2015) http://www.imdb.com/title/tt4032428/
- **Stavka na lyubov** (2016) https://www.imdb.com/title/tt5179324/
- **Strana chudes** (2016) https://www.imdb.com/title/tt5430020/
- **Super Family** (2016) https://www.imdb.com/title/tt5576334/
- **Posledniy bogatyr** (2017) https://www.imdb.com/title/tt6175394/
- **Super Family 2** (2018) https://www.imdb.com/title/tt9398222/
- **Tobol** (2019) https://www.imdb.com/title/tt9795368/
- **Streltsov** (2020) https://www.imdb.com/title/tt12058242/
- **Chernobyl** (2020) https://www.imdb.com/title/tt10648714/
- **Posledniy bogatyr. Koren' Zla** (2021) https://www.imdb.com/title/tt13606158/

15.3 History

CGRU project was started in 2005 as Maya (3d software name) tools pack. It included 3d modeling tools and rules. And was designed to join Maya MEL scripts and plug-ins from several artists. It still exists but as a part of the project. Later the project started to include tools for other software and a render farm manager. Moved on SourceForge and lost all not open-source tools.

Afanasy was started in 2007. In 2008 a fist working alpha version started to solve jobs on real VFX projects in "Film Direction" (Russia). It was merged with CGRU project on SourceForge and became open-source. Now it is a main part of CGRU project and other tools can be considered as an utilities for it. So we may say that CGRU consist of Afanasy render manager and some ready solutions for it, like submit scripts or movie encode scripts, for example.

Since 2012 project code is on GitHub.

15.4 Paper

There is some paper, but only in Russian.

Paper (rus)

It is about render managers generally, Afanasy idea, why it was written, who is interested in. If somebody going to translate it in English, please contact, it is really needed. Paper has some funny shade, it is done not to bore people that are not vary familiar with the theme, may be with computers at all. As practically 90% of people in auditorium do not really know where they are and why.

CHAPTER 16

Contacts

- **Forum** https://forum.cgru.info (EN)
  https://forus.cgru.info (RU)
- **GitHub Issues** https://github.com/CGRU/cgru/issues
  Designed for code development questions only.
- **RULES Online** The simplest way. No registration needed.
  https://rules.cgru.info/#/Ask_Questions_Here
- Mail box on Google: **cgruafanasy**
- **Russian Telegram channel** https://t.me/cgruafanasy

CHAPTER 17

Changes Log

If Afanasy **network protocol changed**, it became incompatible with old one.

- The first number in the version means some significant changes in the project. Probably you should reconfigure Afanasy.
- The second number means major changes that caused compatibility lost. You should upgrade all clients and server at once.
- The third number versions are fully compatible. You can upgrade only one client or server for changes.

17.1 v3.3.1

2023.03.12

- Afanasy service can be configured to limit task post command running time. Task post commands are used to generate thumbnails. By default `af_task_post_limit_sec` config variable stands for it, and it is 16 seconds. Later (on limit exceeded) task post command will be killed.
- Afanasy can be configured not to cut domain names from user and host names. New parameters `af_render_cut_domain_name` and `af_user_cut_domain_name` added to control this. The default behaviour is the same as before, domain names will be cut.
- Afanasy statistics database tables got new rows. Job blocks got job serial. Tasks run got job serial, block and task ids. See database schema.
- Nuke Afanasy Gizmo can set extra environment variables that will be added to task process. You use it to store Nuke location and version.
- Houdini Afanasy ROP can set extra environment variables that will be added to task process. You use it to store Houdini location and version.
- AfWatch can edit job block environment variables.
- AfWatch now uses `open_folder_cmd` config variable to open folders.
- Afanasy server will not allow to change a pool of a busy render.

- Afanasy server branches action added `delete_done_jobs`. You can delete all done jobs from branch from AfWatch.
- Afanasy jobs branch can be paused.
- Afanasy user can be paused.
- Server annoying error log removed on a running job deletion.
- Rules Python API started.

17.2 v3.3.0

2022.07.25

- AfRender can send to server GPU resources: utilization, temperature, memory total and used.

  Note: This feature is sponsored by SMF Animation Studio, LLC

  For now, just NVIDIA is supported.
- New job block need parameters: GPU Memory, CPU Frequency, CPU Cores and CPU Cores*Frequency.
- AfWatch and WebGUI shows some summary info, when several jobs are selected.
- AfWatch got and "Administration" menu to switch to super user mode with password.
- AfWatch warning and error messages and super user mode label highlighted to attract more attention. When selection is not allowed, nodes list displays a warning message.
- Now parser checks whether the task progress was changed. Later server had to compare previous and new percentage value. And this behaviour was build-in. Now you can override it by parser. A new progress_changed variable for it was added. And by default, progress is changed if task produced any output.
- New task state flag Wait Dependencies appeared. Now tasks that wait some other tasks will be marked with this flag. Tasks with this flag are skipped during job solving. So now task dependencies will not wait the last frame of unbroken sequence.
- **Bugfixes:**
  - AfServer and job block with no task crash fixed. The crash was very rare, another block with tasks should follow a block without tasks.

17.3 v3.2.2

2022.02.09

- AfServer `JOB_CREATED` event added.
- Afanasy config: Parameter of a string array type can be overridden by command arguments or environment by a string contains a comma separated items.
- AfWatch
  - Task window uses mono-space font for task output and log.
  - Scroll step size preferences option. Later scroll size was only by item height.
- Houdini Afanasy ROP:

  - Pre and post submission scripts execution.
  - Separate Mantra rendering: Generate IFD block parameters.
- Houdini Afanasy TOP:
  - "Keep Job On Cancel Cook" option added. You can check tasks outputs, compare different jobs after cooking stop.
  - **Changed to work with Houdini 19 version, will not work with 18.5.**
- Nuke: Submission and render scripts support rendering a movie.
- Afanasy new services and parsers added: *shotgun, ftrack*.
- **Bugfixes:**
  - AfServer:
    - Change running job owner or branch fixed.
    - On start set WARNING tasks to READY.
    - Fixed task progress sending to GUIs mixing task and block numbers.
  - AfWatch: Skipped tasks does not affect job block average running timings.
  - Keeper: Refresh when local render deleted now does not produce an error in server log.
  - Houdini:
    - USD ROP now can works like Alembic ROP. One render session for all frame range.
    - MS Windows launch scripts fixed to work with 19 version.
    - Afanasy ROP:
      - Custom command mode block naming fix.
      - Separate Mantra rendering Auto Tickets fixed.

17.4 v3.2.1

2021.08.19

- Houdini Afanasy TOP Scheduler.
- Houdini setup changed. Now `HOUDINI_PATH=cgru/plugins/houdini` and *afanasy.otl* moved to *otls/afanasy.hda*.
- Each not numeric task can have an own extra environment.
- Paths mapping is applied to Block and Task environment.
- AfWatch now understands appending new blocks/tasks to an existing job.
- AfWatch can restart error tasks of a specified block.
- AfWatch Preferences menu got Reset Windows Geometry item.
- AfWatch can hide branch jobs and pool renders.
- Pools got some operations that will be applied to all renders in it: ejects tasks, launch command, exit renders and delete renders.
- MS Windows release moved on MSVC 2019, Python 3.9.6 and Qt 5.15.2.

> **Warning:** MS Windows 7 is no longer supported, as Python 3.9 dropped it.

- **Bugfixes**
  - AfWatch jobs list right panel folders buttons refresh fix on a new job selection.
  - AfWatch jobs list switching from admin mode, other users jobs appear fixed.

17.5 v3.2.0

2021.04.18

New Afanasy network protocol version.

- Such config parameters as *af_render_heartbeat_sec*, *af_render_up_resources_period*, *af_render_zombietime* and *af_render_exit_no_task_time* moved to pool parameters: heartbeat_sec, resources_update_period, zombie_time, exit_no_task_time. On change they will be dispatched to all pool renders. This way you can tune farm "on-the-fly". Parameter *af_render_connection_lost_time* removed. Now render just uses the same *zombie_time* parameter as server for connection lost.
- Parsers do function takes arguments via dictionary. I hope that is was the last time we needed to change all parsers classes on a function interface change.
- Parser takes resources string and can return resources string. Takes host resources as JSON, that afrender gets for server and GUIs. Can return any custom resources, for example peak memory usage or (and) triangles count. Server stores this string in statistics database and dispatches to GUIs.
- RENDER_NO_TASK event and pool no_task_event_time parameter.
- RENDER_OVERLOAD event and pool overload_event_time parameter.
- AfWatch jobs thumbnails size buttons.
- Server HTTP configuration added. Now it is more easy to make server to serve some custom or even several WebGUIs.
- **Bugfixes**
  - AfWatch Work monitor allows modifications only in admin mode.
  - AfWatch Users and Farm monitors allow selection and current item change only in admin mode.
  - AfWatch Job Block operations fixed to work on MS Windows.
  - AfWatch Job Tasks List window on open task progresses refreshing.
  - AfWatch Jobs List hidden nodes mix on reopen / monitor type or change fixed.

17.6 v3.1.1

2021.01.31

- AfWatch job item can collapse blocks. Useful for UI space economy, especially on jobs with a big blocks count. In a View Options menu you can collapse/expand all jobs, and set an option to collapse new jobs.
- Houdini submission ROP works with Python 3. Now you can choose to download Houdini with internal Python 3.

17.7 v3.1.0

2020.10.05

New Afanasy network protocol version.

- Afanasy pool tickets got maximum hosts property. This is mostly needed for licence hosts limits. There is a common type of licensing where you can run multiple instances of software on same host, occupying only single license.

17.8 v3.0.0

2020.09.13

New Afanasy network protocol version.

- ***Pools*** Now renders are organized in pools hierarchy. All farm (services, capacity, limits, ...) settings are configured by pool properties.

  > **Warning:** You will loose your previous farm setup described in **farm.json**.

- ***Tickets*** Pools Job blocks got Tickets. It is like named capacity.

  You can set root pool NUKE:20 pool tickets to limit Nuke licenses on the entire farm. You can set some pool MEM:64 host tickets to limit RAM. And set corresponding tickets to your job blocks.

- Render node becomes *Sick*, when it produces errors only from different users.
- `RENDER_SICK` and `RENDER_ZOMBIE` events.
- AfWatch got side panel to manipulate nodes.
- AfWatch admin mode let you to mark task as DONE w/o SKIP state.
- Block got a server information string. Now it used to store last started task host name. GUIs show it. Useful for a single task blocks, no need to open job to see what host your simulation running on.
- Try this task next. You can ask server to solve some task(s) as soon as possible. Also you can specify such tasks via Python API on a job submission.
- Each Afanasy node stores running services counts. AfWatch shows user and branch items running services.

17.9 v2.3.1

2019.03.11

- **Afanasy**
  - Linux packages moved on System D.
  - Windows service.
  - AfterFX submission improvements. More setting appeared.
  - NVIDIA nvidia-smi Python custom resource class.
- **RULES**

  - News, Bookmarks and Recent items display status.
  - Incoming fresh News statuses update folders statuses.
  - Scenes shots filtering mutes not found artists, flags and tags.
  - Bookmarks folders.

- **Bugfixes**
  - **Afanasy:**
    * Blender submission fix.
    * Redshift parser fix.
    * Houdini distribute Alembic ROP fix.
    * AfWatch any operation does not affect hidden items.
    * AfWatch setting string parameter JSON value escape added, you can set annotation with quotes.
    * AfRender custom resources meter Python classes fixed to work within Python 3.
    * Multi-host task start and server hung fixed. This bug appeared in 2.3.0.

17.10 v2.3.0

2018.10.17

New Afanasy network protocol version.

- **Afanasy:**
  - Generally new class *Branch* appeared. Now jobs solving is going within branches hierarchy. Branches can represent departments/projects/assets and you can vary their priorities.
  - Job block/task command and files pattern frame replacement is processed in a Python service class. Later it was coded in libafanasy and processed by afserver, and you could not alter it. Now you can use custom frame patterns. AfRender receives a pattern and frame settings (first, last, increment) instead of a ready command/files. This is much more flexible. For example, now in a Python service class we can check all numeric task files for existence and size. And decide to skip task execution if all files are fine.
  - Task progress change timeout job block parameter.
  - Task minimum run time job block parameter.
  - New user max_run_tasks_per_host and jobs_life_time default values are configurable.
  - **Some config parameters renamed:**
    * `af_maxrunningtasks` -> `af_max_running_tasks`
    * `filenamesizemax` -> `file_name_size_max`
  - **Houdini submission:**
    * Job Branch, Wait Time and Task Minimum Run Time parameters added.
- **RULES:**
  - Each playlist item has an own delete button.
  - Shots export to table *frames_num* column added.

- **Bugfixes:**
  - **Afanasy:**

    Server - Python API communication invalid JSON answer fixed:

    * Server JSON answer will never contain extra `A` character after JSON object finish (latest `}`).
    * Python API will never try to read JSON data over message size.
  - Server JSON answer *task_files* fixed, now it escape file names. Later when server was running on windows, \ slashes produced JSON syntax error.
  - Server hung on incoming JSON with invalid syntax fixed.

17.11 v2.2.3

2017.11.02

- **Afanasy:**
  - Server creates all needed threads for network IO at start. Networking threads pool size is configurable. Later it has to create a thread for each incoming connection. Now server can handle bigger amount of clients, consumes less system resources for it. Later on some systems afserver could even hung when clients count is more than about a hundred. As system can fail to create 100 threads per second for a long time (have no time to free all thread resources every second).
  - Linux server can be configured to use not-blocking network IO based on Linux EPOLL facility. By default Linux will use the same blocking (threading) IO, as on other platforms. Non-blocking IO consumes less system resources and can handle more connections at the same time.
  - AfWatch (libafqt) switched to non-blocking network IO based on Qt Signal-Slot mechanism. Now it works better (less hangs) with afserver that has a big total amount on clients.
  - Farm setup allows new machines can be configured to register paused/nimby. This can be useful for a just born machines to not to produce error tasks. When a new afrender registers, but has not enough software installed yet.
  - You can find for some string in the text in task output/log in AfWatch.
  - Double click needed on a button to skip and restart task in AfWatch task window. This can help to prevent unwanted task restart by a single occasional click.
  - Previewcmd options added: Houdini *Mplay* and *DJV* open source sequence player.
  - **Houdini submission:**
    * Shorter names for afanasy temporary .hip files.
    * Camera verification for Mantra submission.
    * Support for *Alembic* submission with progress.
    * Support for *Wedge* submission with block per wedge.
- **Bugfixes:**
  - **Afanasy:**
    * After server restart, reconnected tasks are not ignored by "Max Run Tasks Per Host" limit.
    * Multi-host task start and server hung fixed. This bug appeared in 2.2.1.

        * MS Windows server tasks state storing fixed. Later, after restart, it run done tasks again. This bug appeared in 2.2.1.
        * Server memory leak fixed.
        * Houdini Current frame submission fixed.

17.12 v2.2.2

2017.05.21

- **RULES:**
    - If status progress is 100% all its tasks are considered as done.
- **Afanasy:**
    - Maya Redshift support.
    - Parser updates: Fusion, Redshift, Arnold, Redline, Rsync.
- **Bugfixes:**
    - **Afanasy:**
        * MS Windows AfWatch and jpeg images (thumbnails) reading fixed. Missing Qt5 dll added to the package release archive.
        * AfServer on some Linux distributions can hung when clients number over 100. Fixed - reduced default (configurable) afserver client thread stack size.
        * Parser Error fixed. Later on Parser Error, afrender can ignore its restart from afserver and task update/stop timeouts happen.

17.13 v2.2.1

2017.01.28

- Forum.
- Keeper shows machine memory usage in a system tray icon.
- **Afanasy:**
    - Isotropix *Clarisse iFX* support.
    - AfServer solves jobs by running tasks total capacity (by default), not just by running tasks count.
    - Each new job acquires an unique serial number. You can get jobs from server by serial.
    - Linux distributives that has a native Python 3, Qt 5 switched on these new libs version.
    - Qt 4 and Python 2 are still supported. There is no plan to discontinue this support for the near feature.
- **RULES:**
    - Artists got automatic bookmarks on assigned shots.
- **Bugfixes:**
    - **Afanasy:**
        * MS Windows AfServer WebGUI hosting fixed.

  - Blender parser fixed to work with new versions (previous still supported).
  - VRay parser fixed to work with new versions (previous still supported).

17.14 v2.2.0

2016.11.23

New Afanasy network protocol version.

- **Afanasy:**
  - On server restart it reconnects running renders tasks. New task "WaitForReconnect" state.
  - New job flags designed for "Maintenance" added: *maintenance*, *ignorenimby* and *ignorepaused*. Now you can run some command once on each render (even if it is "Nimby"). For example you can install software this way.
  - GUI got "task" window. One place to view and manipulate job block task.
  - New render "Paused" state. It is like "Nimby" but without "Auto Nimby". Only admins can change this state. Designed to disable render permanently while "Auto Nimby" is enabled.
  - Parsers got "tagHTML" function. It designed to mark task output for AfWatch GUI. For example replace terminal escape sequences, highlight errors.
  - AfWatch GUI nodes list has a second sorting parameter.
  - Houdini submission:
  - Added minimum memory, PPA settings, render temp HIP and wedges support.
  - *Separate Render* deletes ROP files not after render, but on job deletion (you can re-render w/o re-generation).
  - Distributed simulations support.
  - Job Block *environment* parameter added. Render can run task process with some extra environment.
  - You can quickly *override* any config parameter w/o any file changing.
  - You can enable/disable services by a regular expression. It is useful when you have several *houdini_.** types services.
  - JOB_DELETED event added.
- **RULES:**
  - Scenes/Shots asset: Export shots to HTML table. You save/send this table. Print to PDF. Open in Exel/Word.
  - Files view: You can colorize and annotate any item.
  - Walk: Calculates and stores disk usage along with total files size.
- **Bugfixes:**
  - **Afanasy:**
    * AfServer store folders removal fixed on some modern file systems.
    * AfServer on windows thumbnails serving fixed.
    * AfServer now reset depend state on a job block if it depend mask changes on empty string.
    * AfServer does not send job changed event every cycle if a job block has depend mask.

        * Mac OS X: AfRender memory resources detection fixed.
        * AfRender get CPU frequency each time it measures resources, and stores its maximum. Now most machines can change CPU frequency depending on load.

17.15 v2.1.0

2016.04.29

New Afanasy network protocol version

- **Afanasy:**
    - Clients does not listen any port (afrender and afwatch). Server does not connects to clients itself. This means that no local network needed any more. Also it increases maximum clients quantity.
    - MS Windows build now compiled with MSVC 2015. You may needed to install Redistributable 2015 x64 package to work, if you have some old updated Windows OS. It also it has Python 3.4.4 and Qt 5.6.0 versions.
    - WebGUI can listen job and task outputs.
    - Job got *report* report. It is some important info to show in GUI that can be returned from a task process parser.
    - Job Block got *skipexistingfiles* and *checkrenderedfiles* flags.
    - Service and parser can write to task log. This is useful when you decide to skip a task or mark it as an error from service or parser to explain why you did it.
    - You can ask render to execute custom command and exit (or not) after.
    - Multi-host task can ignore slave host missing. To control this, *slavelostignore* job block flag was added.
    - Wake-On-LAN: Sleep and wake commands are constructed in 'wakeonlan' Python service class.
- **RULES:**
    - File buffer to move folders/files.
- **Bugfixes:**
    - **Afanasy:**
        * AfWatch: Turn off listening job/task fixed.

17.16 v2.0.8

2015.12.06

- **Afanasy:**
    - AfWatch desktop notifications.
    - Server waits client have closed network connection first. This way there is no TCP socket TIME_WAIT state on server. It can be needed for a big amount of clients.
- **RULES:**
    - Shot tasks price.

  - Auxiliary folders.
  - User last entries record: IP, URL and time.

- **Bugfixes:**
  - **Afanasy:**
    * Thumbnails double generation fixed.

17.17 v2.0.7

2015.10.23

- **Keeper:**
  - Show and change local render user name.
- **Afanasy:**
  - *Fusion* integration.
  - *Natron* integration.
  - Job has folders string map parameter. It can be user in GUIs to open some location.
  - Statistics folders table and graph.
- **RULES:**
  - Player link to the current frame.

17.18 v2.0.6

2015.07.17

- **RULES:**
  - Dailies with sound.
  - Deploy shot renaming prefix and find/replace regexp.
- **Afanasy:**
  - Auto NIMBY and Auto Free now can depend on MEM, SWAP, HDD and Network usage.
- **Bugfixes:**
  - **Afanasy:**
    * AfWatch can reset (set to an empty string) job block mask (host, depend).
    * Python parser class appendFile function fixed (old style parsers lost thumbnails in 2.0.5).
    * MacOSX compilation fixed (2.0.5 has compilation errors).

17.19 v2.0.5

2015.06.30

- **RULES:**

  - Upload rules. You can describe a rules to upload .mov files in a shot dailies folder and .zip files in results folder. So no artist can upload everything just in a shot, and there is no need to know where shot dailies are located. And a news will be created on upload.
  - Edit body and tasks of a several selected shots.
  - Tasks has prices.
  - You can add scene(s) selected shots to playlist.

- **Afanasy:**
  - Preview Pending Approval `PPA` flag. Now job can render just tasks that described with a sequential parameter (every 10 frame, for example). Then job falls in a PPA state and stops to solve tasks. Artist can check that every 10 frame and unset PPA to continue or delete a job.
  - Sequential behavior slightly changed. Now it renders first and last frames at first, then sequential frames.
  - AfStarter and afjob.py supports Natron.
  - AfRender can generate thumbnails while task process is still running. Was designed and now used in dailies creation.

- **Bugfixes:**
  - **Afanasy:**
    * User can change his job priority above the default value.
    * Afwatch can show hidden node on some parameter change.
    * Set farm auto nimby parameters to zero (to disable them) and reload farm 'on-the-fly' (afcmd fload) now works.

17.20 v2.0.4

2015.02.26

- **RULES:**
  - Permissions to edit tasks, body, playlist, assign artists.

- **Afanasy:**
  - Job block frame sequential new parameter.
  - AfWatch can edit custom data for job and user.
  - You can get farm setup from afserver via json.
  - Server can save json object. This can be useful to edit config or farm setup.
  - WebGUI major changes, but not finished, in progress. New idea is less RMB menus. Actions are buttons on the left control panel, Parameters manipulation is on the right panel.

- **Bugfixes:**
  - **Afanasy:**
    * Events service fixed (was broken in 2.0.3).
    * Server memory leak fixed. It was small and rare, probably you did not noticed it.

17.21 v2.0.3

2014.11.09

- **Blender:**
  - Blender plugin was completely rewritten. Now there is a CGRU Tools Addon and Afanasy is a part of it. Afanasy now not a Blender render engine.
- **RULES:**
  - Create Nuke scene in a shot using template. Scene will have good project settings, sources and results (Read and Write nodes).
  - News display filter. You can show/hide/delete specific news (dailies, reports, status, ...). Filter news by a project.
  - Results invalid naming highlighting and tool-tip.
  - Status edit: artists are combined by roles.
  - User states: admin can allow user to change his password, make user not-an-artist to hide him from status edit.
  - Player: show images while loading.
- **Afanasy:**
  - Afrender calls parser function on task finished in any case, even if there is no new output. This may be needed if want to perform some finalizing actions in your custom parser.
  - Service python class has a function to check task process exit status value. By default (in service.py) zero is considered as a success, any other as an error. But now you can override this function in you custom service.
  - Web GUI supports afrender custom resources monitor.
- **Bugfixes:**
  - **Afanasy:**
    - Paths map (mixed os farm) and non-ascii character(s) fixed. Now you can have national characters in paths, but it is not recommended in any case.

17.22 v2.0.2

2014.08.19

- **RULES:**
  - Shot tasks and reports statistics.
  - Create and extract archives via Afanasy farm.
  - Put files on FTP via Afanasy farm.
- **Afanasy:**
  - GUI: Job item ETA.
  - **Server:**
    - On a new job, server returns its ID.

        * Tasks solving speed limit configurable parameter.
    - Events: Server sends the entire job JSON object to render. You can use any job parameter in an event Python service class.
- **Movie Maker (RULES Convert and Dailies):**
    - Apple ProRes422 and ProRes444 10-bit codecs presets.
- **Bugfixes:**
    - **Afanasy:**
        * Cmd: Send json job and an error message in any case fixed.
        * **Watch:**
            · Listen task/job output fixed.
            · Zero thumbnails quantity preference and crush fixed.
        * Render: Task output maximum size and output middle truncation invalid characters.
        * Server: Task that reached maximum running time limit takes ERR state.

17.23 v2.0.1

2014.04.10

- RULES: Convert multiply selected movies or sequences to other movies or sequences. You can change codec, fps, resolution and convert DPX-es to JPEG-s with a specified colorspace and quality, for example. Such calculations will be processed on a farm via Afanasy.
- Movie Maker: AV conversion tool is configurable. You can set a custom command or executable path. The default is *ffmpeg*. Some Linux distributions switched from *ffmpeg* to *avconv*. For now, they are fully compatible.
- **Afanasy JSON protocol:**
    - Jobs list can be generated providing to server an array of user names.
    - Server configuration and farm setup can be reloaded via JSON message.
- **Bugfixes:**
    - Movie Maker and RULES thumbnails: EXR and DPX colorspace problem is solved. You need at least ImageMagick >= 6.8.8-8 version for it. EXR bug was in CGRU, DPX bug was in ImageMagick.
- Afanasy Web GUI: Sorting and filtering parameters storing.
- Afanasy Server: Several bugs that can cause hang fixed. You should definitely switch to this version as soon as possible. It is fully compatible with 2.0.0 (you can just replace afserver binary only).

17.24 v2.0.0

2014.03.01

- **No SQL** Afanasy server stores state in *json* files in its temporary folder. Now SQL stands for statistics only. If you does not need statistics you can not to setup SQL at all (or setup it later).

> **Warning:** Server state will not be stored switching to this version. You will loose all jobs, renders and users settings if any.

So now on MS Windows OS, Afanasy server does not needs any installation/configuration procedures to work. Just run (double click) cgru/start/AFANASY/_afserver.cmd. Or drag a link to Startup menu for auto launch at logon.

- **Authentication** Afanasy *json* protocol has an authentication mechanism. It uses Digest Access Authentication method. IP Trust mask allows to skip authentication. By default mask allows any IP, and if you did not configured it, you should not notice authentication at all. Binary protocol does not have authentication mechanism. If IP does not match trust mask and message uses binary protocol (not *json*) - message will be ignored. This was designed to use Web GUI not from a local network. Note, that it is only authentication mechanism and not data encryption. But passwords are not sent in a plain text, and even are not stored in a plain text (see Digest description).
- **Python service class got doPost method.** You can do some post process there. If post command requires enough calculation, you can return a list of commands (strings) from this function. In this case all that commands will be executed in child subprocesses and output will attached to task output.
- **Thumbnails** If task (block) has files parameter or parser finds images thumbnail will be generated. Thumbnails are generated by afrender. Python service *doPost* function returns commands for it. This commands can be configured. Thumbnail files binary data is send by afrender to afserver along with task output. Server stores all files that afrender sends on task finish. AfWatch and Web GUI can show thumbnails. You can get tasks thumbnails from afserver by HTTP GET method. Python parser class can find images in task output. Python service class can ask parsed images for thumbnails generation.
- **Python parser class got mode argument in parse function.** This argument stands for task subprocess status. For example, now parser knows whether the task is running or finished and how it was finished. Now if a task has finished with success you can set an error if output does not contain some required result.
- **All plugins from *cgru/afanasy/plugins/* moved to *cgru/plugins/*.** That old plugins location came from SVN age, when Afanasy has branches, tags, trunk. Within Git it is not needed. So now there is no mess where to put or find files in *cgru/plugins/* or *cgru/afanasy/plugins*.
- You can add new user via JSON. An example is located in *cgru/examples/json/*.
- WindowsMustDie function configures via general configure system (json files). So there is no a special *windowsmustdie.txt* file now.
- Python service class now instance parser class itself. So you can exchange information between service and parser classes.
- Python API Block and Task classes *setFiles* method takes an array of string. And not a single string where several files are separated with ';'. **You should fix your custom submission scripts if any.**
- CGRU Home folder on MS Windows OS moved to *%APPDATA%/cgru/*. It is used to keep user personal configuration. Previously it was in *%HOMEPATH%/.cgru/* where *%HOMEPATH%* is usually user Documents folder.
- Web GUI is not "beta". It is a full functional GUI for Afanasy, that can replace AfWatch (Qt).
- RULES is not "beta". But there is still the lack of documentation and lots of things to do.
- There is no *temporary* users. Any (each) user in stored in its json file. No *af_user_zombietime* variable - time for temporary user to have no jobs to be deleted. (Temporary was a user that was not stored in SQL database.)
- **Maya**
  - No overriding scripts.
  - No auto scripts sourcing.

  - No plugins auto load.
  - No CGRU main menu auto launch.

  Sow now CGRU in Maya is just a set of stand-alone scripts, and it does not modify any native Maya interface and workflow. This means the lost of some features:

  - No autosave manager.
  - No auto project seek.
  - No Outliner and Channels menus custom items.

  Since Maya 2014 CGRU main menu appears on load *cgru.mll* plugin. For auto load, enable it in plugins manager window. Or you can source *cgru.mel* from a shelf or *userSetup.mel*.

- **Movie Maker** *ffmpeg* and *convert* binaries are removed from Linux packages. There are two reasons for it. Modern Linux distributions has various dependences to build and install them, so it begin harder and harder to support them in CGRU. Also modern Linux distributions already has enough high versions of this products to support EXR and H264. If you need to some special version of this binaries, you can to download and build it yourself, there is no problems in Linux to compile them manually.

  Debian based packages will have *ffmpeg* and *imagemagick* (*convert*) dependences. As all such distributions has them in native repositories (they are usually enough big).

  RPM based packages will not have only *imagemagick* dependency, as for *ffmpeg* you need to add some extra repository (native repositories are usually small). The exception is AltLinux.

  MS Windows release will continue to contain this executables.

- AfTalk Afanasy chat client was removed from the project.
- **Bugfixes:**
  - Server hung when a job with no blocks sent.
  - Change job bock (tasks) command (working folder) change from watch GUI.
  - Post command ignore when job json file send with afcmd.
  - Change any user parameter resets jobs solving method to 'order'.

17.25 v1.7.0

2013.06.05

New Afanasy network protocol version.

- WEB GUI (beta).
- **RULES (beta).** It has begun!
- Forum (beta). Based on RULES web engine.
- Parser can return running task *activity* string parameter. For example Nuke can notify which of stereo views is rendering now. Movie Maker notify whether an encoding is started. Activity string is shown by GUI in job tasks list window for each task item.
- Render client Nimby can be set to free if computer is idle for some time. You can configure it in farm setup. Machine considered as idle if CPU busy percentage is less than *idle_cpu* value. It is useful for render on workstations that artists left.
- Render client Nimby can be turned on if computer CPU is busy for some time and has no Afanasy task. You can configure it in farm setup. Machine considered as busy if CPU busy percentage greater than *busy_cpu* value.

- Afanasy server sends to GUI tasks percentage with renders list. GUI renders list items show running tasks percentage.
- New system job block - **events**. New service - **events**. Afanasy server can generate events, on job error, for example. Events are pushed to system job as tasks for events block. Render farm can process events, send email notifications for example.
- Each afnode has a custom data. Afanasy server sends this data to render to service class with a task. In Python service class you can do with this data what you want. For example user email parameter and events settings are stored in custom data via JSON.
- You can restart all job running tasks from GUI menuitem.
- Archived binaries Python version is 3.3.2.
- **Bugfixes:**
  - AfWatch: Several blocks selection for some action works.

17.26 v1.6.12

2013.03.22

- Afanasy configuration now has parameters to control user ability to change priority: *af_perm_user_mod_his_priority* and *af_perm_user_mod_job_priority*. By default user can change his own priority and his jobs priority. Set this parameters to *false*< and only admin will be allowed to change priorities.
- Movie Maker: Apple ProRes codec presets.
- Tested with Nuke 7 - works fine.
- **Bugfixes:**
  - Movie Maker: H264 (ffmpeg-libx264) uses 420 pixel format instead of 444 to work on most players.
  - Nuke Submission: Fixed to render Write-nodes inside group.
  - Nuke Render Script: Fixed to render different views in different folders.

17.27 v1.6.11

2013.02.15

- Maya users should look at meTools for Afanasy. And use it.
- Nuke and Paths Map: Filename filter can be added to always have valid paths on any OS in the same script. You can configure to add or not to add it - not to break you potential in-house filters.
- Tested on Windows 8 - works fine.
- **Bugfixes:**
  - Nuke Submission: Negative frame range fixed.
  - Nuke Render Script: Fixed to render several views in one file (you can write stereo in a singe EXR).
  - AfStarter Blender: Now does not ignore output images parameter.
  - Paths Map: Now works with big files thousands times faster.
  - AfServer: Enable/Disable service fixed (was broken in last versions while json protocol switch).

  - PyQt: Open file dialog fixed to work with old PyQt versions (4.6.2 - CentOS 6).

17.28 v1.6.10

2012.12.21

- **Bugfixes:**
  - AfServer: Creating temp folder it tries to create all parent folders.
  - Keeper: Set Afanasy server fixed.
  - World: No the end, fixed.

17.29 v1.6.9

2012.12.19

- Cinema4D: Submission switched from *afjob.py* command to Afanasy Python API. So there are no issues with *subprocess.Popen* any more. Same code works fine on all platforms.
- **Bugfixes:**
  - AfStarter: Output images browse file button fixed.
  - Cinema4D: Render scene with spaces in path fixed.
  - Keeper: Software setup fixed (select executable dialog).

17.30 v1.6.8

2012.12.10

- Automatic Wake-On-LAN.
- **Bugfixes:**
  - Nuke: It does not really use render script when it should not (when there is no paths map or temporary images).

17.31 v1.6.7

2012.12.03

- All CGRU config files moved to JSON. It refers to any Afanasy configuration, farm setup, paths map. XML is removed from the project at all. Any XML config file will not works. AfWatch GUI turning will be reset.

  Important: You should reconfigure Afanasy.

- Afanasy user 'home' configuration files moved to *HOME/.cgru* from *HOME/.afanasy*.

- One config file can include another file(s). Specify a files to be included in "include" string array. All include files will be included after all file will be read (not like include directive in most common program languages). This is done to override file contents. Any next occurrence of a variable with the same name will override previous value.
- Config file can have OS specific section. So you can setup different OS-es configs in the same file.
- Paths map setup moved to common config files. And you can setup paths map for all OS-es in the same file.
- Some general config parameters, as time format, maximum file name length, command shell, preview commands, moved from Afanasy specific config to global CGRU config. As they can be used later by other CGRU tools.
- Afanasy on start-up reads CGRU config file and does not tries to find some specific config itself. CGRU config file simple includes Afanasy specific config file. All Afanasy specific parameters has *af_* prefix now.

17.32 v1.6.6

2012.09.26

- All Python applications with GUI in CGRU can use and PySide and PyQt. At first PySide will be tried to import and than PyQt. It means that if you have PySide installed it will be used. PySide has LGPL license, PyQt - only GPL. So now all components in CGRU has LGPL license or similar.
- New Linux package *afanasy-qtgui* appeared. Needed only to remove *libqt* dependence from *afanasy-render* and *afanasy-server* packages (to not to install huge Qt on render nodes).
- **Bugfixes:**
  - `afcmd uadd` works fine (is was broken in v1.6.5 - it added users that can't run any tasks)

17.33 v1.6.5

2012.09.04

- Movie Maker can fake dailies date and time.
- Python Parser class can consider that task is already done and ask render to stop a task. AfRender sends to server that it was finished with a success.
- **Bugfixes:**
  - Movie Maker open/save parameters and non ASCII characters bug fixed, all operations uses UTF-8 encoding.
  - AfStarter and negative frame values (actually the bug was in afjob.py).
  - Negative frame values and numeric commands with padding (afserver generates commands, so it should be restarted).

17.34 v1.6.4

2012.06.26

- CGRU now has a domain https://cgru.info. Soon documentation from sourceforge.net will be removed. If you have RSS subscribed, you should resubscribe on http://cgru.info/doc/cgru_rss_feed.xml

- **Bugfixes:**
  - Python API *af.Block.setHostsMask* and *af.Block.setHostsMaskExclude* methods are back after occasional deletion when switching to JSON.
  - AfStarter maya_mental submission set verbose level for task progress parsing, afjob.py changed for it.
  - AfStarter dialog GUI dialog bug fixed: *first_frame* <= *last_frame check* works correctly.
  - Nuke CGRU menu open/save scene through paths map fixed.

17.35 v1.6.3

2012.05.07

- Nuke render and submission scripts options added to skip paths map and render to temporary image stages. Render hosts (farm) should be updated too to recognize such options, as not only submission script changed.
- **Bugfixes:**
  - Nuke render just one frame fixed.
  - Depend sub task and depended block frames per task > 1 fixed.

17.36 v1.6.2

2012.04.23

- **API is based on JSON now.** Python API is the same but no binary module needed, it communicates with server itself by JSON build-in module. (All Python API is written on Python language, not on Python C API.) You can communicate with Afanasy server within any language/script that can create JSON structures. (No libafanasy needed to send and get data, all possible linking problems are in the past.)

  JSON protocol is not finished. Finished only job structure - to remove python binary module dependence to send a job.
- **Bugfixes:**
  - AfWatch shows tasks with no service icon.
  - AfWatch filtering and sorting nodes when new nodes created and old changed fixed.
  - Houdini render script loads scene within try-catch to pass warning exception.
  - Nuke dailies node can handle tcl expressions, it uses *getEvaluatedValue()* instead of *value()*.
  - Nuke render script changes *root.project_directory* according to OS paths map (for mixed OS-es farm rendering).

17.37 v1.6.1

2012.03.28

- Tasks can be solved in a not-sequential manner.

  For example 1-10: 0 9 5 2 7 1 3 6 8 4

  This can be needed to catch some error earlier and to calculate average running time more accurate.

- You can hide jobs or renders in AfWatch by some parameter. Also you can show only hidden nodes. And a new "hidden" parameter was added to every node (job, render) just to hide (and store hidden state).
- **Bugfixes:**
  - AfWatch can preview tasks of a not-numeric blocks in a task information window (by double click). This is a main reason of this release.

17.38 v1.6.0

2012.03.22

New Afanasy network protocol version.

- New parameter added to configuration *cmd_shell*. Render will launch tasks commands with it. Default values are: - UNIX: `/bin/bash -c` - MS Windows: `cmd.exe /c`
- Administrator (super users) can change job owner. It can be performed by AfWatch GUI and afcmd CLI.
- You can enable/disable render service via afcmd (CLI).
- AfWatch GUI styles available. You can change, copy, modify them, create your own. You can set sounds to playback on some events (Job added, finished or got an error).
- Renders list has an ability to change items size.
- No Qt library in Afanasy render client. So Qt is used for GUI only now.
- *Magic Number* to filter connections.
- Afanasy server is available for MS Windows OS.
- GitHub https://github.com/cgru CGRU project started.
- Afanasy branches removed from repository. Use git for branching. As there is no need in branches in project subdirectories structure.
- **Bugfixes:**
  - Server bug fixed. It could hung on job submission. It was a very rare deadlock bug. I never managed to catch it for 4 years.
  - Keeper hung on new network protocol version fixed.

17.39 v1.5.5

2012.02.12

- Cinema 4D support.
- Maya Bins release removed. Use archive for MS Windows for or Linux to get plugins for Maya.
- Nuke dailies gizmo can encode only (skip convert stage).
- Movie Maker allow user to specify container to encode movie to (mov, avi, . . .), through GUI dialog or command line argument.
- Movie Maker can save and load settings, keeps recent jobs options.
- Nuke submit and render scripts can handle write node file expressions.

- Python Class Block - added following functions: setErrorsAvoidHost, setErrorsForgiveTime, setErrorsRetries, setErrorsTaskSameHost.
- Afanasy stand-alone starter has an ability to add some custom arguments to command.
- Server has an acceptable IP Addresses Mask. Connections form addresses not matching specified masks will be ignored by server.
- User can set jobs solving method to parallel.
- Afanasy now supports only PostgreSQL database engine. QtSql library replaced with native PostgreSQL libpq in libafsql module. So there is no Qt in afserver and afcmd applications (as later Qt was removed from libafanasy). Do not forget to update you server database connection settings, if you override defaults.
- **Bugfixes:**
  - Afanasy Starter error message in console fixed, sending a job and with Python 2.x (Fedora Linux raises a warning in system tray in this case).
- Paths map works in lower case mode on windows and only in client -> server direction So you can use paths with uppercase letters with UNIX clients and MS Windows.

17.40 v1.5.4

2011.12.22

- AltLinux RPM packages support.
- "Nimby" schedule improved. Now if *time begin* > *time end* it assumes that *time end* is tomorrow. So now you can set for example for Monday that *time begin* is 14:00 and *time end* is 1:00, and it makes render free at Tuesday 1:00.
- Afanasy stand-alone starter has an ability customize command, preview images and OS needed for render.
- Blender Cycles render engine support.
- Windows version switched on MSVC 10 SP 1.
- Release archives switched on Qt 4.8.0.
- **Bugfixes:**
  - Error messages in standard output fixed opening Movie Maker and Afanasy Starter dialogs.
  - Afanasy render and server Linux packages post install scripts fixed. On some systems they were unable to create *render* user, if it does not exist.

17.41 v1.5.3

2011.12.05

- **Bugfixes:**
  - Home configuration folders and files permissions. Now they writable to all.

17.42 v1.5.2

2011.12.02

- Movie Maker can decode movie to sequence and add sound to movie from an audio or another movie file with audio.
- Keeper tray icon displays Afanasy local render client status.
- **Bugfixes:**
  - Keeper AFANASY client operation local host name bug fixed. Bug was, for example, if you are setting NIMBY on "c1" machine, it will be set to all computers with name starts with "c1": "c10","c11","c19" ...
- Movie Maker fixed to work with a sequence without padding specified ("%d" or single "#" character).
- Houdini submission fixed. Afanasy ROP got a check for a null connection. Full path to ROP is used. You can to submit ROPs placed anywhere in a scene, not only from "/out/".

17.43 v1.5.1

2011.11.14

- **Keeper** - CGRU applications managing program.
- **Afanasy Starter** - Standalone dialog to submit jobs to Afanasy.
- **Adobe After Effects** support.
- Linux packages structure simplified. Some of them removed.
- Server farm setup *clearservces* directive.
- Movie Maker input images and output movie pixel aspect and auto input aspect. Custom aspect cacher.
- Scan Scan input images and output movie pixel aspect and auto input aspect. Search path include and exclude patterns. Search files older than some date option. Place result relative to the sequence.
- Python 3 full support. You can build all Afanasy application with Python 3, construct and submit jobs, write services and parsers for render clients.
- Release for MS Windows uses Python 3. It provided with CGRU. You don't need to install and configure Python and PyQt on MS Windows. On Linux distributions native python version is used and you should to install native PyQt.
- **Bugfixes:**
  - SoftImage submit a scene with a spaces in file path.
  - Scan Scan does not try to create a movie just from one file with digits in a filename like a sequence.
  - When block (job) errors avoid host parameter is zero, block (job) does not avoiding any hosts.
  - Mac OS X Afanasy server with client connection error fixed, render client resources collection improvements.
  - Nuke submission frame increment parameter not ignored any more.

17.44 v1.5.0

2011.08.29

New Afanasy network protocol version.

- Houdini submission improvements. You can connect several Afanasy and other ROP nodes together to describe a complex job with dependencies between ROPs.
- Block tasks can depend on other block sub-task progress.
- New job parameter *Maximum running tasks per host*. The same parameter was added to job block.
- You can override render *Max Tasks* parameter directly from Watch in super user mode.
- New numeric pattern replacement rules.

> **Important:** You should delete all jobs on server as their tasks commands can be invalid.
>
> Jobs created by your custom submission scripts probably will generate invalid numeric tasks too. But it is very simple to fix them.
>
> You should to fix your custom submission scripts.

- Server stores renders IP and MAC addresses in a database. So you can perform some operations with off-line renders after server restart (for example wake-on-lan).
- On start, server checks all database tables, and adds (removes) needed columns.
- All date/time and frame range parameters are 64bit integers.
- All numeric types has BIGINT SQL type.
- Numeric tasks block "frame increment" (or "by frame") parameter plays role in tasks generation. It means that blocks with this parameter grater then one will have less tasks number.

> **Important:** You should delete all jobs on server before upgrade to this version.

- **Bugfixes:**
  - Web Visor statistics average farm usage parameter does not ignore custom dates range.

17.45 v1.4.5

2011.05.26

- Server tries to reconnect to database when connection failed.
- Python 3 supported by Afanasy module. You can construct and send jobs using Python 3.
- Blender 2.5 support.
- Web visor statistics favorite user and favorite service column. You can specify dates to for statistics information tables.
- Autodesk Max, Maya and XSI 2012 support.
- **Bugfixes:**

  - Render "Division by zero" hung fixed. It was very rare bug but you could catch it after machine sleep (was suspended with stored RAM and running afrender process).

    Note: Only Chuck Norris can divide by zero.

17.46 v1.4.4

2011.05.07

- Wake-On-LAN
- Render client sends network interfaces information to server (MAC and IP addresses).
- Watch can request information message from server about render client.
- Watch items tool-tips improved.
- Watch renders custom commands can use selected node(s) IP address ("@IP@" string will be replaced with it).
- Watch can set job block parameter for all selected jobs.
- Web-Visor statistics displays total counters row, first record date, services tasks quantity.
- When render can't import task service Python class, it imports services base class called "service".
- Movie Maker temporary images format and quality settings, option to auto correct color space (Linear and Cineon to sRGB).
- Nuke client-server-client paths map interface in a Nuke CGRU main menu.
- **Bugfixes:**
  - Render busy time calculation corrected (it affects GUI counter only).
  - Watch job tasks list window title - job total percentage fixed.
  - Watch job tasks list - block item tool-tip corrected.
  - Nuke dailies node - job (block) custom capacity not ignored.
  - Nuke afanasy node - "Wait whole frame range rendered" behavior corrected.
  - Client does not try to lookup Afanasy server if direct IP literals specified.
  - MS Windows 7 clients does not try to create Afanasy home folder if it is already exists.
  - Server reload farm setup on-the-fly fixed when new host has less services.
  - Server hung when user tries to restart or skip all job blocks (but not restart entire job menu item) fixed.
  - Listen entire job when some tasks are already running corrected. They begin to sent output too.
  - Fixed ffmpeg presets end-of-line for UNIX. On Linux they cause an error with Windows end-of-line.

17.47 v1.4.3

2011.04.11

- **Bugfixes:**

  - Some server memory leaks fixed.
  - Watch listen just one task bug fixed.

17.48 v1.4.2

2011.04.03

- Added **afcmd** commands to control jobs: start, stop, pause, restart.
- **Bugfixes:**
  - Lots of errors in Afanasy server log if it was launched without database connection fixed.
  - Watch jobs list stores sorting and filtering settings.
  - Nuke parser bug fixed (error could appear in Nuke 6.2).

17.49 v1.4.1

2011.03.30

- Farm Services Limits to describe a number of software licenses.
- Movie Maker can draw a logo on an images sequence.
- Watch renders list can sort and filter renders addresses.
- System job commands queue can be cleared by restarting task.
- **Technical:**
  - Default Python version is 2.7.1. Default Qt version is 4.7.2.
  - XML parser moved from Qt to RapidXML. No library needed, it is implemented by headers only.
  - Regular expressions moved from Qt to POSIX, they are almost the same. No library needed. They are in C standard, already realized in GCC and MSVC>=2008SP1.
  - No Qt needed for *libafanasy* and so for *libafapi* and *libpyaf* too. No errors can happen importing Python module in other software using Qt.
  - Windows version moved to static Qt libraries. No errors can happen with various Qt "dll"s in PATH.
  - If parsing is no needed, parser should have an empty string name. Render do not tries to import parser module with an empty name, no error happen.
- **Bugfixes:**
  - Nuke render script: A try to delete moved temporary image removed.
  - Listen job and task output connection error fixed.
  - Numeric command frame(s) replacement bug fixed. Now it replaces any number of %04d patterns with start and end frame in a cycle. (The bug appears for example on a composite commands: "cmd1 && cmd2" or "cmd1; cmd2". And when one task has several files for preview.)
  - Maya Auto Save Manager history backup filename from date and time construct on MS Windows bug fixed.

17.50 v1.4.0

2011.02.20

New Afanasy network protocol version. New Afanasy database schema.

- Errors forgive time for job tasks block and for user. It is a time form last host error to exclude it from error hosts list.
- **System job** Now job (and block) post commands are executed on a render farm by a special system job. **Your farm hosts must have "system" service to execute job post commands (remove rendered scenes).** `afcmd db_sysjobdel` deletes system job from database. Will be needed if system job will have too much changes with new Afanasy version.
- Nuke *dailies* gizmo can be connected to *Read* node.
- Render views list can be customized.
- Job *Life Time* parameter added, for automatic jobs deletion after some time.
- **WindowsMustDie** windows names list can be defined in several files, matched windowsmustdie*.txt mask.
- User can sort jobs in Watch.
- Server does not store deleted jobs logs and tasks outputs.
- Release **bin_pyaf** removed. Modules for various Python versions are in every release now.
- Release **svn** added. It is an export of a repository.
- **Bugfixes:**
  - Nuke afanasy gizmo: If it creates output folder, it creates recursive all needed folders.
  - Watch job tasks list: Block item error hosts counters corrected.
  - Server stores job order in user list in database, so on server restart user jobs list order restored.
  - When parser on render finds an error, and than rapidly finds a warning, error status may be lost.

17.51 v1.3.1

2010.12.14

- Movie Maker output file naming customizable rules. This rules works for Nuke *dailies* node too.
- Server Farm Setup changed. Now host get setup form every matched pattern. And in each pattern you can precise host settings.
- Render reboot and shutdown commands can be configured.
- **Bugfixes:**
  - Watch job tasks window: Task item: Task host name string may overlap task name strings if this strings are long enough.

17.52 v1.3.0

2010.12.06

New Afanasy network protocol version. New Afanasy database schema.

- AfWatch shows services icons, it is common programs icons for users to recognize jobs type.
- Every Afanasy client has compiled revision number, startup version string and sends them to server. Most dialogues in CGRU show version, Afanasy GUI also shows clients build revision.
- Nuke *dailies* node to generate movies locally or on Afanasy farm.
- *movgen* service added. It will be used for movies generation: annotate frames, encode, make dailies.
- **Bugfixes:**
  - *ScanScan* codecs presets search folder.
  - SoftImage *VariRender* changes output folder name for every Framebuffer if folder is specified.
  - Houdini mantra filter (af_separate_render ROP) does not filter null images now (shadows for example).

17.53 v1.2.4

2010.11.01

- 3D Studio MAX submit to Afanasy scripts. MAX Afanasy service and parser.
- Watch can ask and launch a custom command with render items and has more sort&filter parameters.
- Movie Maker stereo mode, DNxHD codec ffmpeg preset and Utf-8 full support.
- Linear float EXR and logarithm DPX to sRGB conversion bug fixed.
- H264 ffmpeg preset updated: good size&quality and frame navigation on MS Windows QuickTime player.
- Nuke stereo render views in different folders bug fixed.

17.54 v1.2.3

2010.08.18

- Houdini parsers total percentage calculation bug fixed.

17.55 v1.2.2

2010.08.17

- Movie Maker works with folders with spaces.
- **MS Windows:**
  - Afanasy Render prefix commands with *cmd.exe /c*.
  - Afrender kills all child tasks in any case. (There was still some cases when it did not do it. Warning! QtCore4.dll patched, do not use it.)

  - Afrender measures network and disk traffic.
  - Afrender MS Windows version has the same functionality as Linux version.

- Package “afanasy-examples” removed. All examples are in “cgru” package.

17.56 v1.2.1

2010.08.06

- Afanasy server database communication bug fixed.

17.57 v1.2.0

2010.08.02

New Afanasy network protocol version. New Afanasy database schema.

- Afanasy Python *Custom Resources Meter*. You can measure any resource by writing you Python resource meter class.
- Afanasy Python Parsers has a new functionality. A parser can produce *warning* to notify user, *error* to stop task with error, *bad result* to finish task like with bad exit status (with error in any way).
- Afanasy render client *Windows Must Die* function. It finds and kills windows with specified names. When process crashes, MS Windows can raise a window with apologizes. This can hang process until someone closes the window. (AfRender periodically sends WM_CLOSE signal to windows listed in special file.)
- Houdini Separate Render ROP to separate Mantra ROP render process on ‘ifd’ files generation and ‘mantra’ command render. It can also split one frame into tiles and render them simultaneous, clean ‘ifd’ files, clean tiles and render an image in local temporary folder, and after successful render copy it to network location (it can save network traffic, as host do not often write small portions of an image during calculations).
- Block *Frames Per Task* parameter can be negative. Needed for sub-frame dependency.
- Afanasy has an ability to map paths. You can setup farm with various platforms clients. Submit jobs on Windows or Linux (MacOSX) to render and on Windows and on Linux (MacOSX) clients. Every client can have individual a paths map file to translate paths to server and from server.
- Movie Maker works on MS Windows. Linux releases has *ffmpeg* binary compiled with *x264* library to encode ‘H.264’ codec. Windows users need to install ImageMagick, which contains ‘ffmpeg’ with ‘x264’.
- SoftImage XSI submit to Afanasy scripts. XSI Afanasy service and parser.
- Lots of bug-fixes for MS Windows platform. Windows version can be called ‘beta’.

17.58 v1.1.0

2010.05.09

New Afanasy network protocol version.

- Afanasy supports IPv6. Server needs to support new protocol, as it stores client addresses, and do not ask name server at every connect (most managers do, alfred too).
- Nuke render script to render images locally in temporary folder and copy completely rendered image (it can reduce network traffic).

- Nuke render network: ‘afanasy’ nodes can be connected to describe ‘Write’ nodes dependency.
- **Movie Maker** Dialog and command line utility to make movie file from image sequence on Linux.
- RPM build scripts (tested on openSUSE, Fedora, CentOS).
- Windows Afanasy GUI applications does not open terminal.
- Maya 2010 and 2011 support.
- *fbx2clip* utility removed.

17.59 v1.0.0

2009.12.21

- New project structure. Afanasy source code repository contains ‘tags’, ‘branches’ and ‘trunk’.
- CGRU has ‘deb’ packages build scripts (for Debian and Ubuntu Linux).

17.60 v2009.11.12

- Afanasy project building uses **CMake** cross-platform build system.
- CGRU environment initialization is much simplified. You do not need to edit or create scripts. To setup CGRU you need to go in it’s root folder and source setup script (like in Houdini now). Unix and Windows examples corrected to work the same way. (And also total quantity of variables initializing by CGRU setup and needed for correct work is reduced.)

17.61 v2009.10.07

- Python class *Job* has a *blocks* array property. You can manipulate it in your own way it to fill job with blocks.
- Python class *Block* can be constructed without any job and has a *tasks* array property. You can manipulate it in your own way it to fill block with tasks.
- Python class *Task* can be constructed without any block or job.

17.62 v2009.09.16

New network protocol version. New database schema.

- Watch renders colors customization.
- **Multi Host Tasks** - tasks can run on several hosts.
- Python Class *Block* got *setMultiHost* method to describe multi-host tasks.

17.63 v2009.08.24

New network protocol version.

- Afanasy Watch GUI can manipulate job blocks parameters without to open job tasks window.
- *afjob.py* supports tasks capacity and capacity coefficients.

17.64 v2009.08.20

New network protocol version. New database schema.

- Job block capacity can be variable.
- Python Class *Block* got *setVariableCapacity* method to describe variable capacity.
- Job blocks errors solving parameters has '-1' value by default. It means to take this parameters from job user settings. Watch does not show this default values.

17.65 v2009.08.12

New network protocol version. New database schema.

- Job block have a rule for generated tasks names.
- Not numeric block can generated tasks with preview.
- Python Class *Block* got *addTask* method to add tasks.
- Python Class *Task* got *Task* - New interface for not numeric blocks.
- Watch shows block generated task by double click on task in job tasks view.