# OCTA

## INSTALLATION GUIDE for Linux

**V8.4**

**May 18 2021**

**Authors of the Manual**

JSOL Corporation

**Acknowledgment**

This work was primarily supported by the national project, which was entrusted to the Japan Chemical Innovation Institute (JCII) by the New Energy and Industrial Technology Development Organization (NEDO) under METI's Program for the Scientific Technology Development for Industries that Creates New Industries.

This work was also supported by CREST-JST (Japan Science and Technology Agency) from 2003FY, and has been supported by NEDO since 2016, grant JPNP16010.

# Contents

**Introduction** ..... 1
1.1 Supported Systems ..... 1
1.2 Required external software ..... 2
**Installation on Linux** ..... 4
2.1 Installation ..... 4
2.2 On Red Hat (Red Hat, CentOS, CentOS Stream) distributions ..... 4
2.3 On Ubuntu (version 18.04, 20.04) distribution and others ..... 5
2.4 Supplemental Information ..... 7
2.5 Launch ..... 8
2.5 Uninstallation ..... 9
**About gourmetterm** ..... 10
3.1 gourmetterm ..... 10
3.2 An example of use ..... 11
**Details on how to compile GOURMET software** ..... 12
1. Required compiler or tools ..... 12
2. Compilation ..... 12

# Chapter 1

# Introduction

This manual describes how to install GOURMET version 4.3 and the simulators (calculation engines) on Linux. GOURMET (Graphical Open UseR interface for Material design EnvironmenT) is developed as the graphical user interface and the UDF I/O interface library part of OCTA computer aided material designing system.

This installer includes the precompiled executable simulation engine modules for Linux x86_64, and includes the pre-compiled binary executable image about GUI part of GOURMET. On Linux for Red Hat distributions, it has been confirmed that OCTA is available without compilation if the external software shown in Chapter 2 Section 2 is installed. On other Linux distributions, you need to compile Python and GOURMET as described in Chapter 2 Section 3.

## 1.1 Supported Systems

### 1.1.1 GOURMET

Currently available hardware on which GOURMET's graphical user interface (GUI) runs are:

・computers with x86_64 series CPU of Intel corp. or compatible CPU's of them.

Standard hardware specification requirements are:

・Memories with 2GByte or more.

・Free hard disk space of 10GByte or more.

GOURMET GUI is known to work on the following Linux distributions.

Red Hat Enterprise Linux release 7/8 (x86_64)

CentOS-7/8(x86_64), CentOS 8 Stream(x86_64)

Ubuntu 18.04(amd64), Ubuntu 20.04(amd64) (Requires compilation of Python and GOURMET)

### 1.1.2 Simulation engines

Simulation engines of the current OCTA distribution are known to work on the following OS.

・ Linux x86_64

Note that not all possible combinations of the engines and the above environment are guaranteed.

## 1.2 Required external software

GOURMET needs several kinds of software which are not developed in our project. They are the following "external software".

If you want to use GOURMET, install the following external libraries

java-1.8.0-openjdk

gnuplot

■ Java Development Kit (jdk 1.8)

To compile and run GOURMET's GUI, JDK is necessary. GOURMET GUI is confirmed to work on:

OpenJDK 1.8

Install OpenJDK to compile GOURMET’s Java programs. gcj (GNU Java compiler) cannot compile GOURMET’s Java programs.

■ JOGL v2

JOGL is a library for binding Java and OpenGL. OCTA installer includes JOGL Library for Linux 64-bit version. JOGL v2.2.4 library is located in OCTA84/GOURMET/lib/linux_64 directory.

■ OpenGL

GOURMET GUI must have OpenGL library (libGL, libGLU) in order to work on Linux system. Usually, the OpenGL library on Linux system is used.

■ Python (version 3.7. Python 2.7 or former is not supported.)

GOURMET and simulation engines use Python version 3. The tools require the extension packages such as numpy and scipy etc. To use the extension packages easily, the installer includes Python library, which includes the necessary additional extension packages. (If you use a Linux distribution other than Red Hat (CentOS), you need to build Python from the source.)

■ gnuplot

Gnuplot helps displaying graphs from GOURMET. We confirmed that gnuplot3.7.3 or later works on GOURMET.

■ Image and file compression library

GOURMET uses JPEG and PNG libraries to output image files, which are both standard features of Linux system. PNG library needs ZLIB. These two libraries and header files are necessary for construction and execution of GOURMET GUI modules. If the library is not installed, the GUI module cannot output a corresponding image file.

Chapter 2 Section 2 - 3 shows command examples for installing the above libraries.

# Chapter 2

# Installation on Linux

## 2.1 Installation

Decompress the following installation files and select a location for installation.

OCTA84_LINUX64_INSTALLER.tar.gz

This section assumes that a user named “octa” installs OCTA into the /home/octa directory.

```
$ tar -zxvf OCTA84_LINUX64_INSTALLER.tar.gz -C /home/octa
```

The structure of the directories under OCTA84 is as shown below.
The directory OCTA84 contains ENGINES, GOURMET and Python37.

```
OCTA84
  + DOCUMENTS
  + ENGINES
  + GOURMET
  + Python37
```

Normal users need to set the following environment variables in the .bashrc file, which is located in the home directory.

```
export OCTA84_HOME=$HOME/OCTA84
export PATH=$OCTA84_HOME/GOURMET:$PATH
```

The first line sets the location of the OCTA installation as the environment variable OCTA84_HOME. Setting PATH as shown in the second line above allows you to invoke gourmet, gourmetterm and other OCTA tools at any location.

## 2.2 On Red Hat (Red Hat, CentOS, CentOS Stream) distributions

On Red Hat and CentOS version 7 or 8 and CentOS 8 Stream distributions, when you execute the

following commands to install the OS libraries, you do not need to compile Python and GOURET. You need have the root privilege and be connected to the Internet to execute the following commands.

```
yum install gcc gcc-c++
yum install libGL-devel libGLU-devel
yum install java-1.8.0-openjdk java-1.8.0-openjdk-devel
yum install zlib-devel libjpeg-devel libpng-devel
yum install gnuplot
yum install libffi-devel
yum install openssl-devel
yum install xz-devel
yum install bzip2-devel
```

The script below executes the above commands at once.
OCTA84/setup0-lib.sh

After CentOS 8 Stream Linux, you will need to recompile Python and GOURMET. In the case, install according to the method described in the next section.

## 2.3 On Ubuntu (version 18.04, 20.04) distribution and others

For other than Red Hat Linux distributions, follow the steps below to install.
(1) Install the required libraries (setup0-lib.sh).
(2) Build Python 3.7 (setup1-python.sh).
(3) Install Python packages (numpy, scipy, numba) (setup2-pypkg.sh).
(4) Build GOURMET (setup3-gourmet.sh).
(5) (Cleaning (setup4-clean.sh)).

Execute the following commands in order.

```
$ cd OCTA84
$ sudo ./setup0-lib.sh
$ ./setup1-python.sh
$ ./setup2-pypkg.sh
$ sudo update-alternatives --config javac
```

The result of the above command is as shown in the example below. Specify the full path of the

javac compiler of the java-8 as an argument to the following setup3-gourmet.sh command. When the java-8 compiler is the current compiler, you do not need to specify the argument to the setup3-gourmet.sh command.

```
  Selection    Path                                        Priority   Status
------------------------------------------------------------
* 0            /usr/lib/jvm/java-11-openjdk-amd64/bin/javac   1111       auto mode
  1            /usr/lib/jvm/java-11-openjdk-amd64/bin/javac   1111       manual mode
  2            /usr/lib/jvm/java-8-openjdk-amd64/bin/javac    1081       manual mode
```

```
$ ./setup3-gourmet.sh  /usr/lib/jvm/java-8-openjdk-amd64/bin/javac
$ ./setup4-clean.sh
```

Script setup0-lib.sh executes the following commands.

```
apt install gcc g++
apt install libgl1-mesa-dev libglu1-mesa-dev
apt install zlib1g-dev libjpeg-dev libpng-dev
apt install openjdk-8-jdk
apt install gnuplot
apt install libffi-dev
apt install libssl-dev
apt install libbz2-dev
apt install liblzma-dev
```

When you run setup1-python.sh, you will see the following examples.

```
>>> /home/octa/OCTA84/Python37 already exists.
      Rename or remove the directory.
      Do you want to delete the directory ?
>>> OK ? (y: yes, other: quit) ==> y
>>> Install python to /home/octa/OCTA84/Python37
>>> OK ? (y: yes (default), other: quit) ==>
```

After the above, Python compilation begins. The compiled Python is installed to OCTA84/Python37 so as not to affect the Linux OS Python.

The running of setup2-pypkg.sh script installs the Python extension packages with the pip

command. At this time as well, the PC needs to be connected to the Internet.[1]

See the contents of the each script for details.

You can build with the other version of Python source file when you run the setup1-python.sh command. The supported versions are:

Python 3.8 and 3.9

You must prepare the Python source file. The file extension supports only tgz. You need to edit the configuration file setup-config.sh as follows. For example, for Python version 3.8:

The exaple of setup-config.sh

```
## Python3.7 (default)
#PYTHON_INST_SRC=Python-3.7.10.tgz
#PYTHON_INST_DIR=Python37
#PYTHON_INST_EXE=python3.7

## Python3.8
PYTHON_INST_SRC=Python-3.8.10.tgz
PYTHON_INST_DIR=Python38
PYTHON_INST_EXE=python3.8
```

The meanings of shell variables are as follows:

PYTHON_INST_SRC : Python source file name.

PYTHON_INST_DIR : Installation directory name.

PYTHON_INST_EXE : Python executable file name (This is symbolically linked to "python").

## 2.4 Supplemental Information

(1) On Ubuntu Linux, set the following environment

export MESA_GL_VERSION_OVERRIDE=3.0

(2) When using Ubuntu with a Japanese environment

When using Ubuntu with a Japanese environment, add the environment variable shown below.

(This prevents issues occurring when opening the file selection dialog box in GOURMET.)

export LC_CTYPE=ja_JP.UTF-8

[1] You may need to set a proxy for pip. Set in the configuration file ($HOME/.pip/pip.conf) as shown in the example below.
[global]
proxy = [user:passwd@]proxy.server:port
trusted-host = pypi.python.org pypi.org files.pythonhosted.org

(3) How to use the pip command

The pip command is executed using a Python command as follows.

python -m pip ***command options***

Make sure python is under OCTA84 directory, as shown in the example below.

```
$ . gourmetterm -
$ which python
/home/octa/OCTA84/Python37/bin/python
$ python -m pip list
```

If you set python under your OCTA84 directory to the environment variable PATH, some OS commands cannot be used (yum/apt etc.). Therefore, do not set python under your OCTA84 directory to the environment variable PATH.

### 2.5 Launch

If the environment variable PATH includes the path for launching GOURMET, you can launch GOURMET from any location. You will need to execute it from a directory with write permissions because temporary files may be created in the directory. The OCTA tools are launched with the following commands.

| Tools | Launch command |
|---|---|
| GOURMET | gourmet |
| ImageLoader | imageloader |

The each tools work directory are followings.

ImageLoader OCTA84/GOURMET/tool/ImageLoader

## When executing engines in OCTA

Executing the following command once from the command terminal will set the environment variables required to execute engines. The environment for using the Python extension library (UDFManager) for handling UDF files from the command terminal will also be set.

```
$ . gourmetterm -
```

## 2.5 Uninstallation

To uninstall, delete all the content in the installation directory (OCTA84).

In addition, delete any environment variables that you added to the .bashrc file during installation.

# Chapter 3

# About gourmetterm

## 3.1 gourmetterm

There is another script 'gourmetterm' created by the installer. It starts xterm with the same environment where GOURMET works. Below are the environmental variables for the xterm[2] started.

PF_FILES : This is the GOURMET installed directory.

PF_ENGINE : This is the OCTA engines installed directory.

PF_ARCH : Current OS information. It is one of win32/cygwin/linux/alpha.

PYTHONHOME : This is Python directory installed as an external application.

UDF_ACTION_PATH : This is $PF_FILES/action.

UDF_DEF_PATH : $PF_FILES/udf and $PF_ENGINE/udf directory are set.

PATH : The execution paths for external applications (Python and Java) are inserted to PATH.

PF_ENGINEARCH : Same as the above PF_ARCH.

PYTHONPATH : The script search paths of Python. The followings are set up:

$PF_FILES/python, $PF_FILES/lib/$PF_ARCH, $PF_ENGINE/python and $PF_ENGINE/lib/$PF_ARCH are set to this.

If you have specified the above environmental variables, your specifications may have priority.

To execute the following command sets the above environment variables to your terminal window:

$ **. gourmetterm -**

These environment variables are set up by the following two scripts. (See the start-up script "gourmet" and "gourmetterm".)

. $OCTA_HOME/GOURMET/bin/gourmet_profile.sh

. $PF_FILES/bin/platform_env.sh

"gourmet_profile.sh" sets up the environment variables about the installation location of OCTA and the

[2] On CentOS-7 or some other Linux distributions, install xterm through package manager (ex. yum install xterm), prior to running gourmetterm. This is because gourmetterm depends on xterm.
Without having xterm, gourmetterm command is still able to specify the environmental variables necessary for running and compiling OCTA engine on your current terminal window (ex. ". gourmetterm -").

external applications (Python and Java) are set. The others are set in "platform_env.sh".

## 3.2 An example of use

CognacUtility.so is needed to draw a result file of the COGNAC engine.[3] The environment variables necessary for creation of CognacUtility is set up by gourmetterm.(Using PF_FILES environment variable)

To start compiling, move your current directory to $PF_ENGINE/COGNAC923/python/analysis/src.

CognacUtility is created and installed as follows.

```
$ make
$ make install
```

When this command completes successfully, CognacUtility is created and installed.

If make command does not complete successfully, the setting of Python include file is often inefficient. Open Makefile by editor, and modify "PYTHON_INC = -I /usr/include/python3" in the Makefile using the appropriate Python version number.

Python version number can be obtained by executing the following command.

```
$ python –V
Python 3.6.3
```

When you get “Python 3.6.3” as a result, use 3.6 as your version number.

```
PYTHON_INC = -I /usr/include/python3.6
```

[3] Access to the additional shared library, such as CognacUtility.so, may be rejected by SELinux. In this case, please execute "chcon" command according to "Allowing Access" displayed on "SELinux Troubleshooting Tool".

# Appendix A

# Details on how to compile GOURMET software

This appendix explains how to compile the executable modules and libraries of GOURMET.

Compilation of GOURMET programs start from running the configure script. If you only create the UDF I/O library (libplatform) which required for compiling the engines, please read 3.2 (1). How to create all the modules and libraries of GOURMET is described in 3.2 (2).

If you just use the engines and do not need to compile the engines, you do not need to compile GOURMET software.

OCTA simulation engines support the following UNIX systems. Currently GOURMET GUI is supported on Linux (x86_64) and Mac OS, but the basic library linked in compiling the engines from the source files can be created on Linux/x86_64 systems.

## 1. Required compiler or tools

The basic library can be compiled by the following compilers :

GCC 4.8.5, GCC 7.4.0, GCC 8.3.1

For compilation, you need to use GNU version of a program “make”. Linux has GNU make. But UNIX users need to install GNU make in advance.

## 2. Compilation

### 2-1 Simple compilation for 64 bit Linux

For 64 bit Linux distribution described in Section 1.1, in case the packages described in Section 1.2 have already been installed, GOURMET modules are easily built by executing the following commands.

```
$ cd OCTA84/GOURMET
$ bash Make-All.sh
```

When using Python outside PATH environment or except name "python", build as the example below.

```
$ bash Make-All.sh /usr/bin/python3.7
```

All the modules needed for starting GOURMET and the libplatform library are built by executing this command. CognacUtility is also included in these modules. When the modules are built, GOURMET starts.

If the module-building is failed, you need to check the installation status of the required libraries. See Chapter 2 Section 2 - 3 for how to install the required libraries.

Below shows how to start GOURMET.

```
$ cd OCTA84/GOURMET
$ ./gourmet
```

To build libplatform library for linking with OCTA simulation engine modules, you execute the following command. In this case, the packages shown in Section 1.2 are not needed.

```
$ bash Make-Libplatform.sh
```

To build Python's extended module (UDFManagerPython.so) for operating UDF files I/O, you execute the following command. In this case, Python-related packages are needed.

```
$ bash Make-UdfManager.sh
```

Execute the following command to build CognacUtility.so in the case of building GOURMET without the above Make-All.sh.

```
$ bash Make-Cognacutility.sh
```

## 2-2 Normal compilation

Move to OCTA84/GOURMET/src directory, and find configure script file there. All the options of configure shows by --help option. But available options related compilation of GOURMET programs are as follows.

```
  -h, --help              display this help and exit
  -V, --version           display version information and exit

 Installation directories:
```

| | |
|---|---|
| --prefix=PREFIX | Specify an installation directory to PREFIX.<br>Usually , do not use --prefix.<br>The parent directory of installation working directory, which is GOURMET directory, is automatically set to PREFIX. Please use --prefix, only if your installation directory differs from the automatically set directory. |

Optional Packages:

| | |
|---|---|
| --with-win32 | Install libraries into GOURMET/lib/win32 directory. |
| --with-win64 | Install libraries into GOURMET/lib/win64 directory. |
| --with-mno-cygwin | On Cygwin, use -mno-cygwin option in compilation. |
| --with-engine=PF_ENGINE | Path to PF_ENGINE directory. This is automatically set and used in creating gourmet_profile.sh. |
| --with-python[=PYTHON] | Absolute path name of Python executable. This is automatically searched and set up from an environment variable PATH. You can not create GOURMET Java modules without Python module, because Python is used in the configuration process. If non-default Python is used for GOURMET, the following environment setting in GOURMET is necessary. Start GOURMET and select ***Environments Setup ...*** item of ***Tool*** menu so as to open an ***Environment*** dialog. Set the Python directory to **PYTHONHOME** text field in ***Python*** Tab on the ***Environment*** dialog. |
| --with-java-home=DIR | Path to the java development kit home directory. |
| --with-jogl-jar=JAR | Path to the directory with jogl.jar and gluegen-rt.jar. |
| --with-jogl-lib=LIB | Path to the directory with libjogl.so ... |

The following examples describe how to create the library.

(1) Creating library libplatform.a

The UDF input/output library (libplatform.a) and UDF tool modules (makeinterface etc.) are created by only the three steps.[4]

[4] Following is how to create UDF Input/Output library (libplatform.a), which links to native modules for Microsoft Windows. Cygwin is one of the UNIX-like environments that operate on Microsoft Windows operating system.

For creating 32-bit library
./configure --with-win32 --host=i686-w64-mingw32

For creating 64-bit library
./configure --with-win64 --host=x86_64-w64-mingw32

```
$ ./configure
$ make
$ make install
```

If you need the Python extended modules, configure as follows.

```
$ ./configure –with-python
```

When using Python outside PATH environment, configure as the following examples.

```
$ ./configure –with-python=/usr/local/bin/python
```

When using Python except name "python", configure as the example below.

```
$ ./configure --with-python=/usr/bin/python3.7
```

The compiler in PATH environment is used as default. If you use the other compilers, specify C++ and C compiler path to CXX and CC respectively.

(2) Constructing the GOURMET GUI modules and libraries by GCC and JAVA compiler (on Linux)

Following is an example of how you can configure the directory of Java development environment and JOGL library. Note that this is a single line command without line break. In this command, use python executable module in PATH.

```
$ ./configure --with-python --with-java-home=/usr/lib/jvm/java-1.8.0-openjdk
                --with-jogl-jar=/home/octa/JOGL/jogl-v2-linux
                --with-jogl-lib=/home/octa/JOGL/jogl-v2-linux
```

Linux 64-bit version automatically uses JOGL, which is included in the installer. You do not need to specify --with-jogl-jar and -with-jogl-lib. Once Java development environment is installed, with PATH being set Java compiler, configure is as follows:[5]

[5] A warning with java may appear while starting gourmet under the following situation.
- You start installing OCTA8 as root on Ubuntu.

And
- You don’t customize the path specification for –with-java-home option in configure, but use the preset

$ ./configure --with-python --with-java-home

When your Python is not set to the PATH environment variable, the configure command is the following.

$ ./configure --with-python=/home/octa/anaconda3/bin/python --with-java-home

At the end of executing configure, the compile and executable test of a Java program using the JOGL library are done. If JOGL library path is incorrect, configuration message is as follows.

```
configure: failed to jogl library path.          <--(JOGL library test fails)
configure: creating ./config.status
config.status: creating Makefile
 . . .
```

If you continue executing makefile without modification, the libraries need for starting GOURMET will not be created. Please modify JOGL library path to be the following message.

```
configure: jogletest passed.          <--(JOGL library test is successful)
configure: creating ./config.status
config.status: creating Makefile
 . . .
```

If JOGL test is successful, Compilation of the Java module by the Java compiler is also done by the make command.

**Care in changing configuration options**

When you change the configuration option, “make clean” before “make” in order to reflect the changes correctly. If you “make clean” after “make”, you may have to compile all over again. However, just to fix incorrectly generated environment setting script, which is seen quite often, follow the steps below, and redo “make” only in src/build5 directory.

---

default value.
To avoid that warning, specify your path for –with-java-home.
(ex. --with-java-home=/usr/lib/jvm/java-8-openjdk-amd64)

```
$ cd build5
$ make clean
$ make
$ make install
$ cd ..
```

**Change of an Autoconf setup**

When you change configure.ac and Makefile.am and remake configure file, execute src/autoreconf.sh. To execute autoreconf.sh, automake (GNU automake) and autoconf (GNU Autoconf) 2.63 or later have to be installed.