# OCTA

Linux 版インストールマニュアル

V8.4

May 18 2021

## 執筆者

株式会社JSOL

## 謝辞

本プログラム開発の大半は、経済産業省の出資・補助を受け、新エネルギー・産業技術総合開発機構(NEDO)が(財)化学技術戦略推進機構に委託した、大学連携型産業科学技術研究開発プロジェクト「高機能材料設計プラットフォーム」通称「土井プロジェクト」の下で行われたものである。

本プログラムの改良は、独立行政法人 科学技術振興機構（旧称　特殊法人 科学技術振興事業団）の補助を受け、「多階層的バイオレオシミュレータの研究開発」における「バイオレオシミュレータ用プラットフォーム」機能改良ソフト開発の下で行われたもの、および、国立研究開発法人新エネルギー・産業技術総合開発機構(NEDO)の委託業務「超先端材料超高速開発基盤技術プロジェクト」の下で行われたものである。

# 目次

# 第１章はじめに

本マニュアルは、高機能材料設計プラットフォーム OCTA を構成するソフトウェアである物質設計支援グラフィカル・ユーザー・インタフェース GOURMET (Graphical Open UseR interface for Material design EnvironmenT) version 4.4 およびシミュレータ(計算エンジン) 群を Linux オペレーティングシステムへインストールする手順を解説しています。

この配付版には、シミュレーションエンジンの Linux x86_64 用コンパイル済みバイナリ実行モジュールが含まれています。

また、GOURMET のコンパイル済みバイナリ実行イメージが含まれています。 Red Hat 系ディストリビューションの Linux において、２章２節に示す外部ソフトウェアをインストールすればコンパイルすることなく利用可能であることを確認しています。その他の Linux ディストリビューションでは２章３節に従って Python および GOURMET をコンパイルする必要があります。

## 1.1 動作環境

### 1.1.1 Linux 版 GOURMET 動作環境

GOURMET のグラフィカル・ユーザー・インタフェース(GUI)が動作するハードウェアは現状では以下のとおりです。

・Intel 社製 x86_64 系 CPU もしくはその互換 CPU を搭載するコンピュータ

GOURMET が動作するために要求する機能の目安は以下の通りです。

・ メモリ 2 GByte 以上 (OS 等が必要とする分は除く)

・ ハードディスク空き容量 10 GByte 以上

下記ディストリビューションの Linux で動作することを確認しています。

・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) (Python および GOURMET のコンパイルが必要)

### 1.1.2 エンジン

シミュレーションエンジンが動作する環境(OS)は以下のとおりです。

・ Linux/x86_64

ただし、すべてのエンジンと上記の環境すべての組合せで動作が保証されているわけではありません。

### 1.2 必要とする外部ソフトウェア

OCTA システムは GOURMET とエンジンを含みますが、GOURMET の動作のためには OCTA システムの一部として開発されたものではない外部ソフトウェアを必要とします。

GOURMET を実行するだけならば、以下のライブラリをインストールしてください。

java-1.8.0-openjdk

gnuplot

OCTA が使用する外部ソフトウェアは以下のとおりです。

■ Java 開発環境(jdk)

GOURMET のモジュール等をコンパイル・実行するためには Java 開発環境(JDK)を必要とします。下記の Java 環境で動作を確認しています。

OpenJDK 1.8

gcj (GNU Java compiler)では GOURMET の Java プログラムをコンパイルできませんので OpenJDK をインストールしてください。

■ JOGL v2

Java から OpenGL を利用するためのライブラリです。64 ビット版 Linux 用 JOGL ライブラリは OCTA インストーラに同梱されています。(OCTA84/GOURMET/lib/linux_64 ディレクトリに JOGL v2.2.4 を置いています。)

■ OpenGL

GOURMET をLinux システム上で動作させる場合にOpenGL 実行ライブラリ(libGL,libGLU)を必要とします。通常はLinux システムにある実行ライブラリを使用します。必要に応じて、Mesa などをインストールしてください。

■ Python

オブジェクト指向スクリプト言語 Python のバージョン 3 が GOURMET およびエンジンで使用されています。また、シミュレーションエンジン COGNAC に付属する Python スクリプトによる描画機能および結果解析機能を利用するためには、Python 拡張モジュールライブラリ numpy が必要です。

numpy や scipy などの拡張パッケージを手軽に利用するために、インストーラには必要な追加パッケージを含む Python が同梱されています。(RedHat 系以外のディストリビューションを使用する場合

は、Python をソースからビルドして拡張パッケージをインストールする必要があります。）

■ gnuplot

GOURMET からのグラフ表示に gnuplot を使用しています。

■ 画像およびファイル圧縮ライブラリ(JPEG, PNG, ZLIB)

GOURMET から画像イメージファイルを出力するために JPEG および PNG ライブラリを使用しています。また、PNG ライブラリは ZLIB を必要とします。Linux システムに標準インストールされているライブラリを利用します。GOURMET の GUI モジュール構築時および実行時にこれらのライブラリが必要です。

上記ライブラリをインストールするためのコマンド例を２章２節から３節に示しています。

# 第２章 インストール方法

## 2.1 インストール作業

下記インストールファイルを解凍します。
OCTA84_LINUX64_INSTALLER.tar.gz

ここでは octa というユーザーが/home/octa　にインストールすると仮定して説明します。

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

ここで、解凍された OCTA84 ディレクトリの構成は以下のようになっています.
ENGINES, GOURMET および Python37 を含みます.

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

次に環境変数 OCTA84_HOME をホームディレクトリの.bashrc などに設定します。

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

１行目は環境変数 OCTA84_HOME に OCTA のインストール先を設定します。２行目は、上記のように PATH を設定しておくと、どこからでも gourmet、gourmetterm および各種ツールが呼び出せるようになります。 複数の OCTA をインストールしている場合は、上記 PATH の設定をしないで、それぞれの OCTA**/GOURMET ディレクトリの gourmet 等のコマンドを実行します。

## 2.2 Red Hat 系(Red Hat, CentOS, CentOS Stream) ディストリビューション

Red Hat 系ディストリビューションのバージョン 8 (Red Hat および Cent OS のバージョン 7 または 8, Cent OS 8 Stream )までは、下記の OS ライブラリをインストールするコマンドを実行しておけば、Python および GOURMET のコンパイルは必要ありません。以下のコマンドを実行するためには root 権限が必要で、インターネットに接続されている必要があります。

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

下記スクリプトは、上記コマンドを一度に実行します。

OCTA84/setup0-lib.sh

Cent OS 8 Stream 以降になると、Python および GOURMET の再コンパイルが必要になると考えられます。その場合は、次節 Ubuntu ディストリビューションに記述した方法に従ってインストールして下さい。

## 2.3 Ubuntu(18.04, 20.04) ディストリビューション その他

Red Hat 系ディストリビューション以外では、以下の手順でインストール作業を実施します。

(1) ライブラリインストール (setup0-lib.sh)

(2) Python 3.7 ビルド (setup1-python.sh)

(3) Python パッケージ(numpy, scipy, numba)インストール(setup2-pypkg.sh)

(4) GOURMET ビルド (setup3-gourmet.sh)

(5) (ビルドのクリーニング (setup4-clean.sh))

インストール作業は、以下のコマンドを順に実行します。

$ cd OCTA84

$ sudo ./setup0-lib.sh

$ ./setup1-python.sh

$ ./setup2-pypkg.sh

$ sudo update-alternatives --config javac

上記コマンドの結果が以下の例のように表示されたとすると、java-8 に対応する javac のフルパスを次の setup3-gourmet.sh の引数に指定します。java-8 の javac コンパイラが現在使用するコンパイラになっている時は、setup3-gourmet.sh に引数を指定する必要はありません。

```
  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

setup0-lib.sh では以下のライブラリインストールコマンドが実行されます。

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

setup1-python.sh を実行すると以下の確認事項(例)が表示されます。

```
>>> /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) ==>
```

確認事項の後、Python のコンパイルが始まります。OS の Python に影響を与えないように OCTA84/Python37 にインストールされます。

setup2-pypkg.sh は pip コマンドによって Python パッケージをインストールします。この時も、インターネットに接続されている必要があります。[①]

[①] pip install コマンド実行のために、proxy の設定が必要な場合があります。
設定ファイル($HOME/.pip/pip.conf)に以下の３行を記述します。

```
[global]
```

詳細はそれぞれのシェルスクリプトの中身をご覧ください。

なお、Python をビルドする setup1-python.sh コマンド実行時に、他の Python バージョンのソースファイルを使ってビルドすることができます。対応するバージョンは以下のとおりです。

Python 3.8 および 3.9

Python のソースファイルはユーザーが用意する必要があります。ファイル拡張子は、tgz のみ対応します。Python バージョンに対応して、setup1-python.sh スクリプトと同じディレクトリにある setup-config.sh を以下のように編集する必要があります。例えば、Python 3.8 の時は以下のようにします。

**setup-config.sh の修正例**

```
## 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
```

シェル変数の意味は以下のとおりです。

PYTHON_INST_SRC : Python のソースファイル名

PYTHON_INST_DIR : インストール先ディレクトリ名

PYTHON_INST_EXE : 実行ファイル名(この名前を python にシンボリックリンクします)

## 2.4 追加事項

(1) Ubuntu Linux の場合、次の環境変数を設定してください

```
export MESA_GL_VERSION_OVERRIDE=3.0
```

(2) Ubuntu を日本語環境で使う場合

Ubuntu を日本語環境で使うときは、下記の環境変数を追加してください.

---

```
proxy = [user:passwd@]proxy.server:port
trusted-host = pypi.python.org pypi.org files.pythonhosted.org

パスワードが不要な場合の proxy 行の例
proxy = http://proxy.server.co.jp:8080
パスワードが必要な場合の proxy 行の例
proxy = user:pass@http://proxy.server.co.jp:8080
```

(GOURMET でファイル選択ダイアログを開いた時に異常が出ることに対する対応)

```
export LC_CTYPE=ja_JP.UTF-8
```

(3) pip コマンドを利用する方法

pip コマンドは Python コマンドから下記のように実行します。

```
python -m pip コマンド オプション
```

以下の例のように、python モジュールが OCTA84 下のものであることを確認してください。

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

OCTA84 下の python を環境変数 PATH に設定すると一部の OS コマンドが使用できなくなります。そのため、常に OCTA84 下の python を環境変数 PATH に設定することはしないでください。

## 2.5 起動方法

環境変数 PATH に GOURMET 等を起動するためのパスが設定されていれば、以下のコマンドでどこからでも GOURMET およびツールを起動することができます。GOURMET は実行したディレクトリの下に一時ファイルを作ることがあるので、書き込み権のあるディレクトリで実行する必要があります。ツールはそれぞれの実行ディレクトリに移動して実行されます。

| ツール名 | 起動コマンド |
|---|---|
| GOURMET | gourmet |
| ImageLoader | imageloader |

ツールの実行ディレクトリは以下のとおりです。

ImageLoader　　OCTA84/GOURMET/tool/ImageLoader

・OCTA のエンジンを実行する場合

下記コマンドをコマンドターミナルで1回実行すれば、エンジン実行に必要な環境変数が設定されます。また、コマンドターミナルから UDF を操作する Python 拡張ライブラリ(UDFManager)を利用するための環境も設定されます。(OCTA84 下の python も環境変数 PATH に追加されます。)

$ . gourmetterm -

## 2.6 アンインストール

アンインストールはインストールしたディレクトリ(OCTA84)以下を削除します。
また、 .bashrc ファイルに書き加えた環境変数を削除してください。

# 第３章 gourmetterm について

## 3.1 環境設定スクリプト gourmetterm

インストーラによって作られるスクリプトGOURMET/gourmettermはGOURMET及びOCTAエンジンを実行・コンパイルする環境を持つxterm[②]を起動します。". gourmetterm -"を実行すると、現在のコマンド端末に環境変数が設定されます。設定される環境変数は以下のとおりです。

PF_FILES：GOURMETがインストールされたディレクトリが設定されます。

PF_ENGINE：OCTAエンジンがインストールされたディレクトリが。

PF_ARCH：OS情報です。linux等が設定されます。

PYTHONHOME：外部アプリケーションとしてインストールしたPythonディレクトリがセットされます。

UDF_ACTION_PATH：$PF_FILES/actionディレクトリになります。

UDF_DEF_PATH：$PF_FILES/udfおよび$PF_ENGINE/udfディレクトリが設定されます。

PATH：PATHにエンジンおよび外部アプリケーション（PythonおよびJava）の実行パスを追加します。

PF_ENGINEARCH：PF_ARCHと同じ

PYTHONPATH：Pythonがスクリプトを探すディレクトリです。
$PF_FILES/python, $PF_FILES/lib/$PF_ARCH, $PF_ENGINE/pythonおよび
$PF_ENGINE/lib/$PF_ARCH が設定されます。

ただし上記環境変数をユーザーが既に設定済の場合、ユーザー設定の方が優先される場合があります。

以下のようにすると現在のコマンド端末に上記環境変数が設定されます：

```
$ .  gourmetterm  -
```

ユーザーホームディレクトリの.bashrcに以下の例のように記述することにより、OCTAを利用する環境を設定することができます。

```
. /home/octa/OCTA84/GOURMET/bin/gourmet_profile.sh
. $PF_FILES/bin/platform_env.sh
```

---

② CentOS等、一部のLinuxディストリビューションではxtermがインストールされていません。したがって、xtermに依存するgourmettermは動作しません。パッケージマネージャ等を用いて、xtermをインストールしてください。(ex. yum install xterm)
xtermがインストールされていない場合でも、". gourmetterm -"コマンドで現在のターミナルウインドウにOCTAエンジンを実行およびコンパイルするために必要な環境変数を設定することができます。

gourmet_profile.sh は OCTA および外部アプリケーションのインストール位置に関する環境変数を設定し、他の環境変数は platform_env.sh で設定されます。初期状態の gourmet_profile.sh は、OCTA のインストール位置を OCTA84_HOME 環境変数で指定できるようになっています。GOURMET のコンパイルを行うと現在の位置をインストール位置として再設定します。

### 3.2 gourmetterm の利用例

COGNAC エンジンの結果ファイルを描画するときなどには CognacUtility.so が必要です。[③]

CognacUtility の作成に必要な環境変数が gourmetterm で設定されます。（PF_FILES 環境変数）

gourmetterm で端末を起動するか". gourmetterm -"でコマンド端末に環境変数をセットしてください。$PF_ENGINE/COGNAC1012/python/analysis/src に移動して下記コマンドを実行します。

```
$ make
$ make install
```

上記コマンドが成功すると CognacUtility が作成されてインストールされます。

make コマンドが失敗するときは、Python のインクルードファイルの設定ができていない場合が多いので、エディタで Makefile を開いて、下記の行を適切な Python のインクルードパスに修正してください。

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

例えば以下のようになります。

```
PYTHON_INC = -I/home/octa/anaconda3/include/python3.7m
```

[③] CognacUtility などの追加共有ライブラリを使うとき、SELinux によりアクセスが拒否されることがあります。その場合は、SELinux のトラブルシューティングに表示される解決方法に従って chcon コマンドを実行してください。（" cannot restore segment prot after reloc: Permission denied" などのエラーが表示されます）

# 添付資料　GOURMET 関連ソフトウェアのコンパイル方法の詳細

GPU 版など SUSHI エンジンはコンパイルされた環境で実行できるだけですので、他の環境で実行するためにはコンパイルしなおす必要があります。GOURMET の実行モジュールおよびエンジンのためのライブラリをコンパイルする方法を説明します。

プログラムのコンパイル作業は configure スクリプトの実行による環境テストからはじめます。エンジンをコンパイルする時に必要となる UDF 入出力ライブラリ（libplatform）を作成するだけならば 2-1 をご覧ください。GOURMET の全てのモジュールおよびライブラリを作成する方法も 2-1 に記述しています。

コンパイル済のシミュレーションエンジンを利用するだけで、新たにエンジンをコンパイルする必要が無ければ本章の作業は不要です。

OCTA システムは UNIX 系環境としては以下のものをサポートしています。GOURMET のグラフィカル・ユーザー・インターフェース(GUI)は Linux(i386、x86_64)、Mac OS で動作しますが、エンジンをソースからコンパイルする時にリンクされる基本ライブラリは、Linux/x86_64 環境で作成することができます。Mac OS については今後対応する予定です。

添付資料では、Linux を含む UNIX 環境でのコンパイル方法について解説します。

## 1. コンパイラおよびツール

以下のコンパイラでライブラリがコンパイルできることを確認しています。並列化計算に対応した COGNAC および SUSHI をコンパイルするためには、GCC 4.2 以降でコンパイルする必要があります。

GCC 4.8.5, GCC 7.4.0, GCC 8.3.1

コンパイルのためにプログラム make を使用しますが、必ず GNU 版 make を使用して下さい。Linux 環境の make は GNU make を使用しているので問題はありません。UNIX では GNUmake を導入して使用する必要があります。

## 2. コンパイル方法

### 2-1 簡単コンパイル方法（64 ビット Linux 向け）

1.1 節に示す Linux ディストリビューションの 64 ビット OS については、1.2 節に示すパッケージがインストールされていれば、以下のように GOURMET のビルドを簡単に行うことができます。ビルド方法は、GOURMET ディレクトリに移動して下記コマンドを実行します。

```
$ cd OCTA84/GOURMET
```

```
$ bash Make-All.sh
```

PATH が通っている Python 以外の Python を使う場合、または PATH が通っている"python"という名前以外の Python を使う場合は以下のように指定します。

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

上記コマンドにより、GOURMET 起動に必要な全てのモジュールおよびエンジン用 libplatform ライブラリがビルドされます。ビルドモジュールには CognacUtility も含まれます。ビルド終了後に GOURMET が起動します。

途中でモジュールのビルドが失敗する場合、必要なパッケージがインストールされているか確認してください。ビルドに必要なパッケージのインストール方法については、２章２節～３節をご覧ください。

正常にビルドが終了して GOURMET が起動したら、２回目からは下記コマンドで GOURMET を起動します。

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

エンジン用 libplatform ライブラリをビルドする場合は、下記コマンドを実行します。この場合は、1.2 節に示すパッケージは必要ありません。

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

UDF ファイル操作用 Python 拡張モジュール(UDFManagerPython.so)をビルドする場合は、下記コマンドを実行します。この場合、Python 関連以外のパッケージは必要ありません。

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

GOURMET のビルドを Make-All.sh 以外で行ったときは、最後に CognacUtility.so を下記のコマンドで作成しておきましょう。

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

## 2-2 通常コンパイル方法

ディレクトリ GOURMET/src に移動してください。このディレクトリに configure スクリプトが用意されています。configureスクリプトで利用するオプションは、--helpで全て表示されますが、GOURMETプログラムのコンパイルに関係するものは以下のとおりです。

| | |
|---|---|
| -h, --help | 全てのオプションを表示して終了します。 |
| -V, --version | configure スクリプトなどのバージョンを表示して終了します。 |

インストールディレクトリ:

| | |
|---|---|
| --prefix=PREFIX | PREFIX にインストール先ディレクトリを指定します。しかし、通常はPREFIX を指定しないでください。デフォルトで PREFIX にはインストール作業を行うディレクトリの親ディレクトリが設定されます。すなわち、GOURMET ディレクトリが設定されます。もし、GOURMET のソースファイルが GOURMET インストール先の下にない時のみ、--prefix= でインストール先 GOURMET ディレクトリを指定します。 |

外部ソフトウェアの指定およびコンパイル条件:

| | |
|---|---|
| --with-win32 | 作成したライブラリのインストール先をGOURMET/lib/win32にします。 |
| --with-win64 | 作成したライブラリのインストール先をGOURMET/lib/win64にします。 |
| --with-mno-cygwin | コンパイルフラグに -mno-cygwin を付加します。 |
| --with-engine=PF_ENGINE | PF_ENGINE ディレクトリを指定します。デフォルトでプログラムソースディレクトリあるいは指定された GOURMET ディレクトリから自動的に設定されます。環境設定ファイルの生成時に使用されます。 |
| --with-python[=PYTHON] | Python 拡張モジュールを作成するために、Python 実行モジュールパスを指定します。Python 実行モジュールパスが環境変数 PATH で設定されているときは、自動的に認識されます。Python は全てのモジュール作成の元になっているため、Python 実行モジュールパスの指定がないと Java モジュールの作成ができません。デフォルト以外の Python を GOURMET で利用するためには、GOURMET を起動して Tool メニューの Environments Setup...から表示される Environment ダイアログの Python タブの PYTHONHOME に Python のディレクトリを指定する必要があります。 |
| --with-java-home=DIR | Java 開発環境ディレクトリパスを指定します。 |
| --with-jogl-jar=JAR | JOGL ライブラリの JAR ファイル（jogl.jar および gluegen-rt.jar）があるディレクトリを指定します。 |
| --with-jogl-lib=LIB | JOGL ライブラリのライブラリファイル（libjogl.so など）があるディレクトリを指定します。 |

(1) ライブラリ libplatform.a の作成方法

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

上記の操作により、エンジンをコンパイルする時に必要となるUDF入出力ライブラリ(libplatform)およびUDFツール（makeinterfaceなど）が作成されます。④

さらにPython拡張モジュール(UDFManager)を作成するときは、configureスクリプトを以下のようにします。

```
$ ./configure --with-python
```

PATHが通っているPython以外のPythonを使う場合は以下の例のようにします。

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

PATHが通っている"python"という名前のモジュールを通常使いますが、違う名前のPythonを使う場合は以下のように名前を指定します。

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

コンパイラとしては、デフォルトでPATHの通っているコンパイラを使用します。他のコンパイラを使用する場合は、CXXおよびCCにそれぞれC++およびCコンパイラを指定します。

(2) GOURMET GUI実行ライブラリの構築

---

④ Microsoft Windowsオペレーティングシステム上で動作するUNIXライクな環境の一つであるCygwinでWindows用のモジュールにリンクするためのUDF入出力ライブラリ(libplatform.a)を作成する方法は以下のようになります。

32ビットライブラリの生成
./configure --with-win32 --host=i686-w64-mingw32

64ビットライブラリの生成
./configure --with-win64 --host=x86_64-w64-mingw32

以下の例のようにconfigureにJava開発環境およびJOGLライブラリのディレクトリを指定します。（見やすくするために改行していますが、続けて１行で指定します。）Python実行モジュールはPATHが通っているものを使用するとします。

```
$ ./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
```

64 ビット版 Linux ではインストーラに同梱されている JOGL を自動的に使用しますので、--with-jogl-jar および--with-jogl-lib の指定は不要です。Java 開発環境がインストールされ Java コンパイラに PATH が通っている場合、configure は以下のようになります。[⑤]

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

PATH に設定していない python モジュールを利用する場合は、以下のようにします。以下の場合、Java は PATH に設定されたものを使っています。

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

configure を実行すると最後のあたりで、JOGL ライブラリを使った Java プログラムのコンパイルおよび実行テストが行われます。JOGL ライブラリの指定が間違っていた場合、以下のように表示されます。

```
configure: failed to jogl library path.  ←JOGL ライブラリテストに失敗
configure: creating ./config.status
config.status: creating Makefile
・・・
```

上記の表示がある時、make を実行すると、GOURMET の実行に必要な実行ライブラリは作成されません。

以下のような表示になるように JOGL ライブラリの指定を修正してください。

```
configure: jogletest passed.    ←JOGL ライブラリテストに成功
configure: creating ./config.status
config.status: creating Makefile
```

[⑤] Ubuntu 上で root 権限にて OCTA8 をインストールしようとして configure の --with-java-home オプションを具体的パス指定なしでデフォルト値を使用しようとした場合、gourmet の起動時に java に関する waring が出ることがあります。これを回避するには--with-java-home に具体的なパスを指定してください。
(ex. --with-java-home=/usr/lib/jvm/java-8-openjdk-amd64)

```
・・・
```

JOGL ライブラリテストに成功した場合、make コマンドで Java コンパイラによる Java モジュールの作成も行われます。

### configure のオプションを変更したときの注意

configure のオプションを変更して make を再実行する時、make の前に make clean を行ってください。make clean を行わないとオプションの変更による実行環境の変更が正しく反映されません。

もし、make clean せずに make した後で make clean をすると、コンパイルを全てやり直さなければなりません。しかし、ほとんどの場合、環境設定スクリプトが正しく生成されていないだけですので、以下のようにすれば、再コンパイルする必要はありません。src/build5 ディレクトリのみで make をやり直します。

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

### Autoconf 設定の変更

configure.ac および各ディレクトリの Makefile.am を変更して configure を作り直すときは、src/autoreconf.sh を実行してください。autoreconf.sh を実行するためには automake (GNU automake) および autoconf (GNU Autoconf) 2.63 以降を使用するため、それらがインストールされている必要があります。

configure 関連スクリプトのタイムスタンプが正しくないためにビルドが失敗するときには、src/touch_config.sh を実行してください。ビルド関連スクリプトのタイムスタンプを修正します。