OCTA

ソフトマテリアルのための統合化シミュレータ

グラフィカルユーザーインターフェース

GOURMET

操作マニュアル

Version 4.1.3

OCTAユーザーズグループ

Aug. 08 2010

執筆者

第１章	土井正男
第２章以降	西尾裕三
改訂	西本正博、小沢拓

プログラム開発者

プログラム設計	芝野真次、西尾裕三
プログラム開発	西本正博、庄野敦子、橋本寛、乾吉治、増井郁久、 稗島潤一郎、奥野幸洋、田子森裕香、奥田俊樹、西康之
ソフトウェアテスト	西谷英輔、西尾裕三、堀川貴子

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である。

目次

図目次

表目次

第１章　ＧＯＵＲＭＥＴとは

GOURMET(Graphical Open User interface foR Multiscale analysys EnvironmenT)は、OCTAプロジェクトのさまざまなシミュレーションエンジンの共通ユーザインタフェースとして開発されました。OCTAの全てのシミュレーションエンジンは、GOURMET を使うことにより便利になります。GOURMETを使うと、

（１）エンジンの入力ファイルを作成し、

（２）エンジンを実行し、

（３）エンジンの出力ファイルをさまざまな形（数値、グラフ、3D アニメーション）で見て、

（４）分析し、独自のプログラムで出力ファイルを作成することができます。

いくつかのエンジンでは、実行中のエンジンの状況を見たり、計算を中止・再開したり実行中のエンジンに対して適切なタイミングでパラメータを変更するような制御を行えます。

GOURMETはスクリプト言語Pythonのインタフェースを持っており、このエレガントな言語に大変お世話になっています。PythonによりGOURMETをあなたの好きなようにカスタマイズできるのです。Pythonプログラムを書くと、入力ファイルを自動作成したり、出力ファイルの中から選択した特定のデータ項目を3Dグラフィックスで表示したり、解析結果をgnuplotアプリケーションでグラフ化したりするような機能を、GOURMETに追加することができます。

GOURMETの入力ファイルはUDF（User Definable Format）と呼ばれるあるフォーマットのテキストファイルで書かれています。UDFは、2 つの部分からできています。最初の部分はデータ構造と単位を定義し、2つ目の部分はデータそれ自身です。UDFファイルの中では、全てのデータ（数値および文字列）はユニークな名前を持っており、その名前でアクセスできます。このように、UDFファイルはいつでもアクセスして変更することのできるデータストレージとみなすことができます。この機能は、Pythonにより強化されたものです。

UDFのどんなデータやデータ集合でもPythonの変数のように扱うことができます。

数値に名前を付けることは、他人に理解できるデータファイルを作るという点からも重要です。GOURMETは他にもあなたのプログラムをユーザフレンドリーなものにする機能を提供しています。それらは、

（１）各データに単位を定義したり、

（２）あなたのプログラムのユーザを助けるためのコメントを表示したり、

（３）どんなデータにでもアクションを定義できる機能などです。

このマニュアルを読めば、このようなことをどうやって実現するかを理解できるでしょう。もっと詳細には、他のマニュアルも見てください。

GOURMETの目的は、シミュレーション・プログラムを書いたり使ったりする様々な人々のコラボレーションを容易にすることです。多くのプログラムが世界中の研究所で作られていますが、プログラムを他人に使用できるようにするためには莫大な時間が必要なため、めったにそのよう

には書かれることがありません。GOURMETはこのような問題の一部を1つのインタフェースを提案することにより解決します。（もちろん、プログラムを正しく頑丈にすることは難しい別の領域として残っていますが）

我々は、GOURMETが手製のプログラムを他人に使えるようにし、世界の多くのプログラムのコラボレーションのために役立つことを望んでいます。

2001.11.30 土井正男

第 2 章　最初にすること

まず、GOURMET を使い始めましょう。

2.1 GOURMETを起動します

Windowsでは、スタートメニューのOCTA2010->StartGOURMETメニューを選択してください。
Linuxでは、シェルウインドウで"OCTA2010/GOURMET_2010/gourmet"を実行してください。
このコマンドは、GOURMETの起動スクリプトです。起動オプションに関する詳細は、第3章をご覧ください。最初に編集用画面（図1）が現れます。

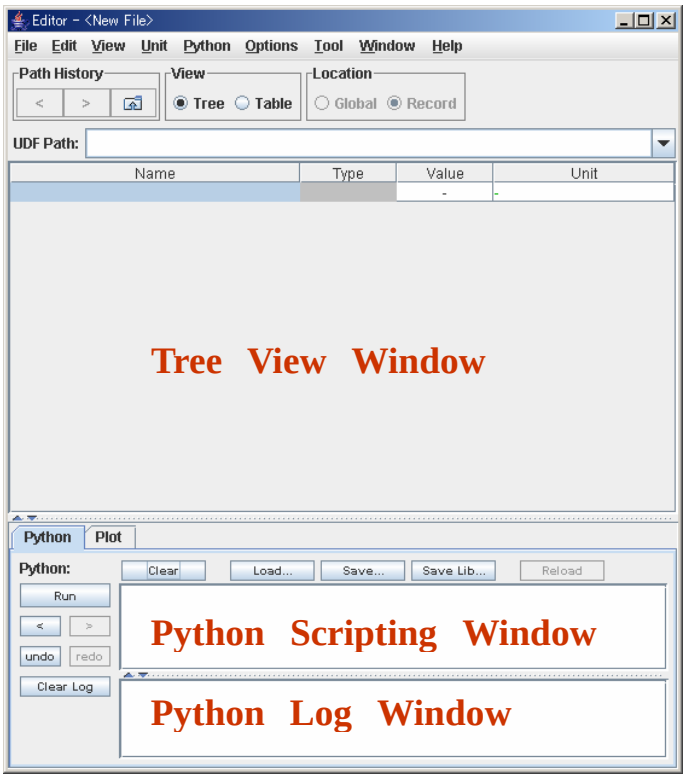


図 1: GOURMET の起動画面

・Tree View Windowでは、データ値を見たり編集することができます。
・Python Scripting Windowでは、Pythonを使ってどんなプログラムでも書くことができます。
例えば、入力データを生成したり、出力データの統計処理を行ったり、出力データを変換したりすることができます。
・Python Log Windowでは、実行したPythonスクリプトの出力結果が表示されます。

2.2 UDFを編集します

UDFファイルを開くには、File/Open... メニューを使います。

ここでは、"GOURMET_XXXX/tutorial/3dball/baseball.udf"という例題ファイルを開いてください。このファイルは、地上で投げられたボールの振る舞いをシミュレートするチュートリアルのサンプルです。

Table View Window の"Ball"をマウスの左ボタンでクリックしてください。これで、"Ball"の属性である値と単位を見ることができます。マウスでデータを選択し、他のウインドウやアプリケーション間でコピー・ペーストを行うことができます。詳しい操作方法は、第4 章をご覧ください。

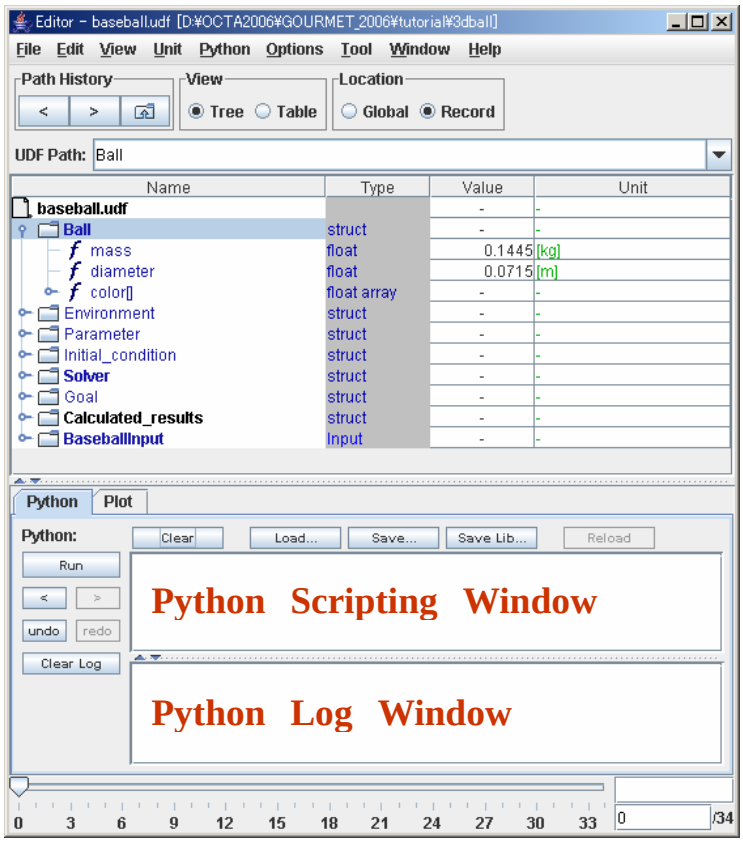


図 2: エディタの Tree ビュー

次に、Table ビューを使うために、Unit menuの下にあるViewグループから"Table"というラジオボタンを選択してください。ビューのスタイルが、Tree View WindowからData Structure Window およびTable View Windowに変わったのが分かると思います。

・Data Structure Windowでは、入出力データの構造と項目を見ることができます。
・Table View Windowでは、データ値をテーブル形式で参照し、編集できます。

Data Structure Windowで、"Goal"をクリックし、さらに"line[]"をクリックしてください。これらのシンボルは（UDFパスと呼ばれます）、UDFファイル中のデータの名前です。これで、"Goal.line[]"の値を、Table View Windowで見ていることになります。これらの値をマウスドラッグで選択し､他のウインドウやアプリケーション間でコピー･ペーストを行うことができます。

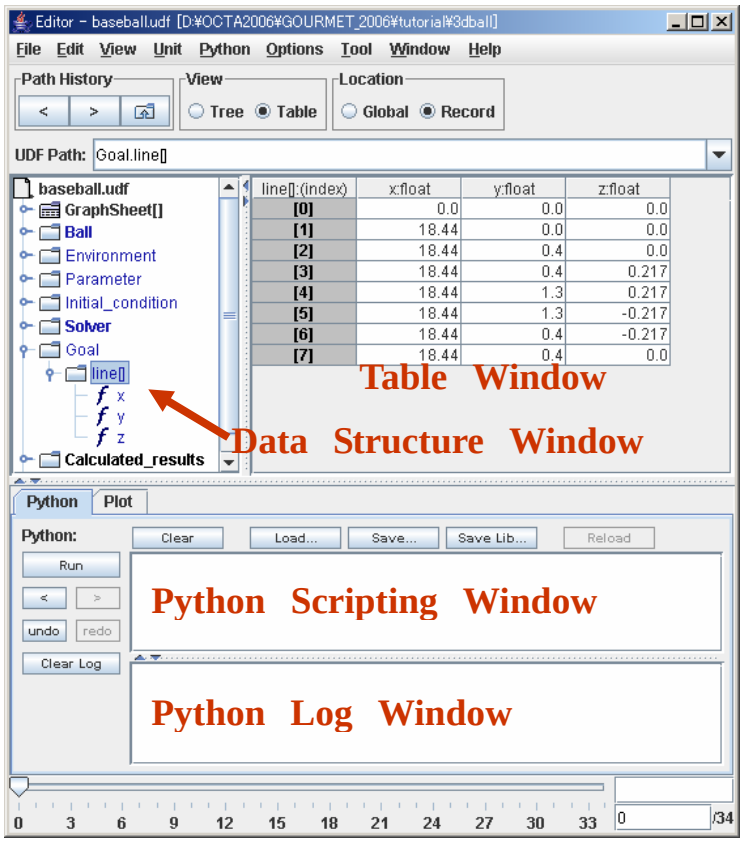


図 3: エディタの Table ビュー

もっと詳しい操作方法は、第4 章をご覧ください。

2.3 Pythonを走らせましょう

さて、開いているUDFファイルに対して、Pythonスクリプトを走らせてみましょう。Python Scripting Window に、次のテキストをタイプして、Runボタンを押してください。

```
print $Ball.mass
```

Python Log Windowに次のような結果が得られるはずです。

```
0.1445
```

これは、Pythonスクリプトを走らせて得られた"Ball"の"mass"（質量）のデータ値です。このようにPythonスクリプトを使えば、開いているUDFのデータにアクセスし、簡単にプログラミングを行うことができます。詳しい操作方法は、第6章をご覧ください。

2.4 エンジンを動かしましょう

GOURMETと交信できるエンジンならば、ネットワーク上で起動し、制御することができます。エンジンを起動したり制御するには､まずエンジンマネージャをエンジンを実行するマシンで起動しておく必要があります。

エンジンマネージャを起動するためには、以下のようにします：

Windowsでは、スタートメニューのOCTA2010->StartEngineManagerメニューを選択してください。

Linuxでは、シェルウインドウで"OCTA2010/GOURMET_2010/eng_man"を実行してください。

エンジンを実行するマシンでエンジンマネージャが稼動していれば、GOURMETのTool/"Engine Run"メニューや"Engine Control"メニューを使えるようになります。

Tool/"Engine Run"メニューを選んで、Engine Run パネルを表示してみましょう。

図4 は、Engine Run パネルです。ここでは、エンジンを動かす条件の設定と解析の開始を行えます。

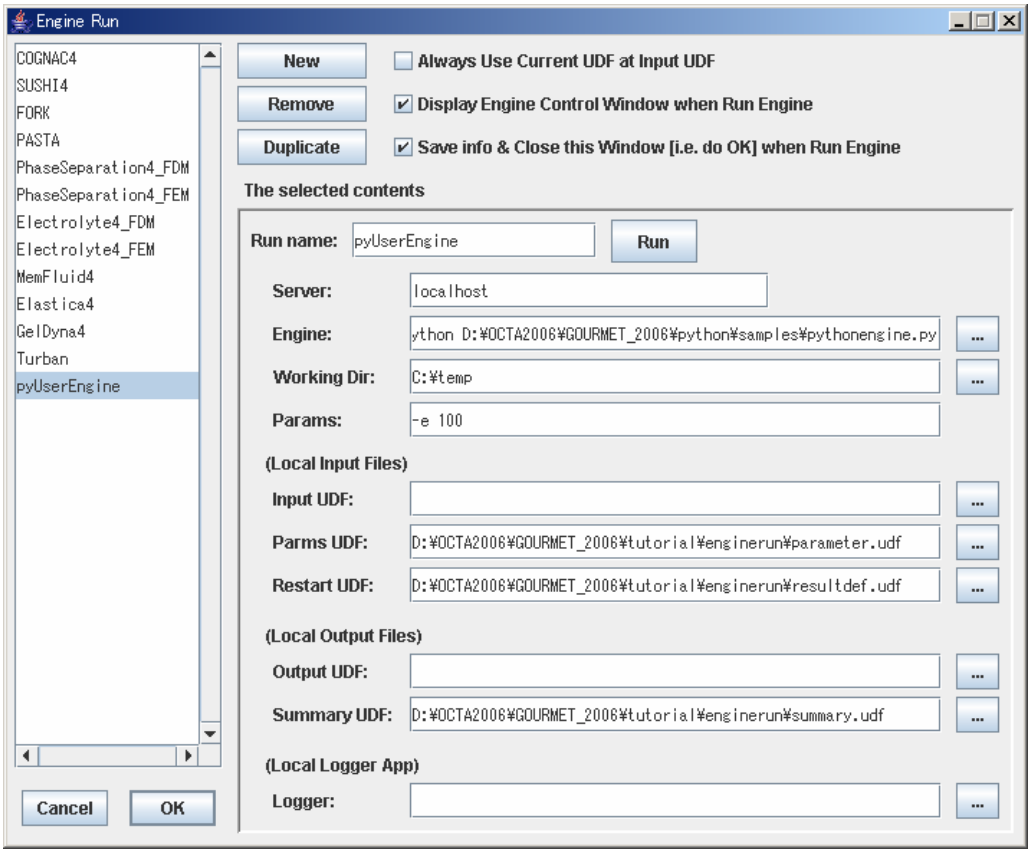


図 4: Engine Run パネル

（補足）エンジン実行デモの設定例

```
Run name: pyUserEngine
Server: localhost
Engine: python C:¥OCTA2010¥GOURMET_2010¥python¥samples¥pythonengine.py
Working Dir: C:¥temp        ← (無ければ作成)
Params: -e 100
Input UDF:
Parms UDF: C:¥OCTA2010¥GOURMET_2010¥tutorial¥enginerun¥parameter.udf
Restart UDF: C:¥OCTA2010¥GOURMET_2010¥tutorial¥enginerun¥resultdef.udf
Output UDF:
Summary UDF: C:¥OCTA2010¥GOURMET_2010¥tutorial¥enginerun¥summary.udf (作成される)
Logger:
```

図5 は、Engine Controlパネルです。Engine Controlパネルは、"Summary UDF"に定義されている項目と、エンジンサーバから受け取った値を表示します。このパネルで、稼動中のエンジンの制御とその結果のサマリーを見ることができます。

エンジン稼動中にできることは以下の通りです。

・パラメータを変更するために解析を中断(Pause)する

・解析を中止するために終了(Stop)させる

・エンジンを強制終了(Kill)する

エンジンを中断（Pause）している場合にできることは以下の通りです。

・パラメータを編集(Edit Parameter)する

・解析を再開（Resume）する

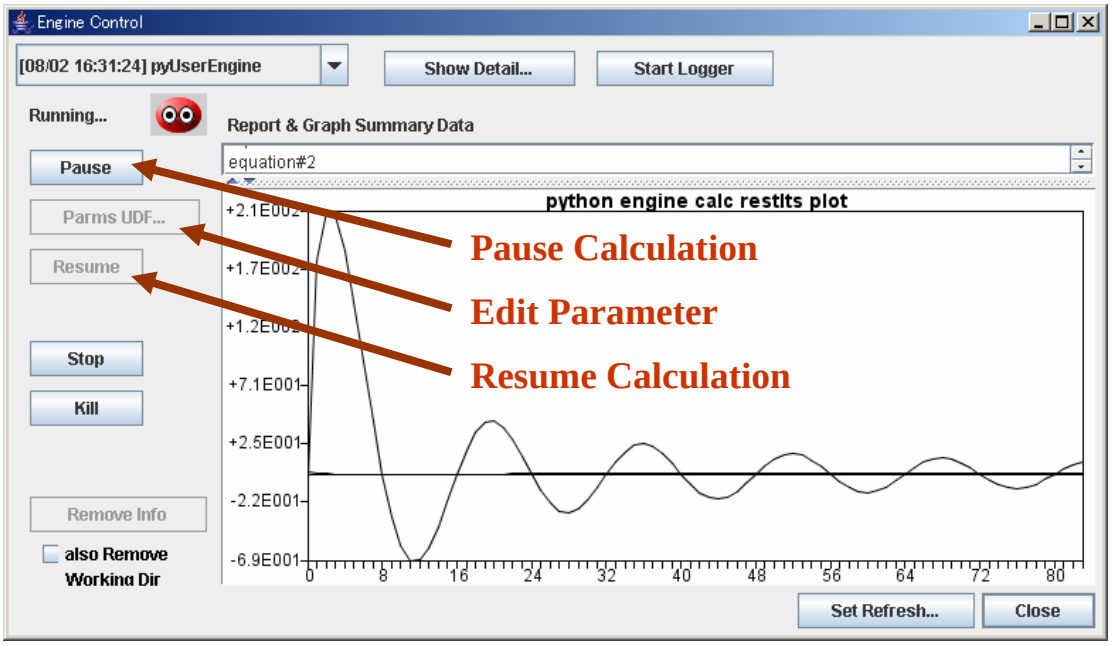


図 5: Engine Control パネル

制御を変更したり結果を見たいときはいつでも、プルダウンメニュー（パネル上部の左端）から制御するエンジン名を選択できます。また、このEngine Controlパネルは、Tool/Engine Controlメニューを選択すればからいつでも表示させることができます。もっと詳しい操作方法は、第7章をご覧ください。

2.5 結果を見ましょう

次に、Window/Viewerメニューを選択して、ビューワを使ってみましょう。開かれているUDFファイルに対応するビューワ画面（図6）が開きます。これは、Pythonスクリプトによって描画される3Dオブジェクト描画用画面です。3D Object Windowの何も描かれていない場所を、コントロールキーを押しながら、マウスの左ボタンでクリックしてください。（これは、アクションスクリプトを実行する操作です）ポップアップメニューから、"show"コマンドを選択してください。3D Object Window にボールとストライクゾーンが表示されます。

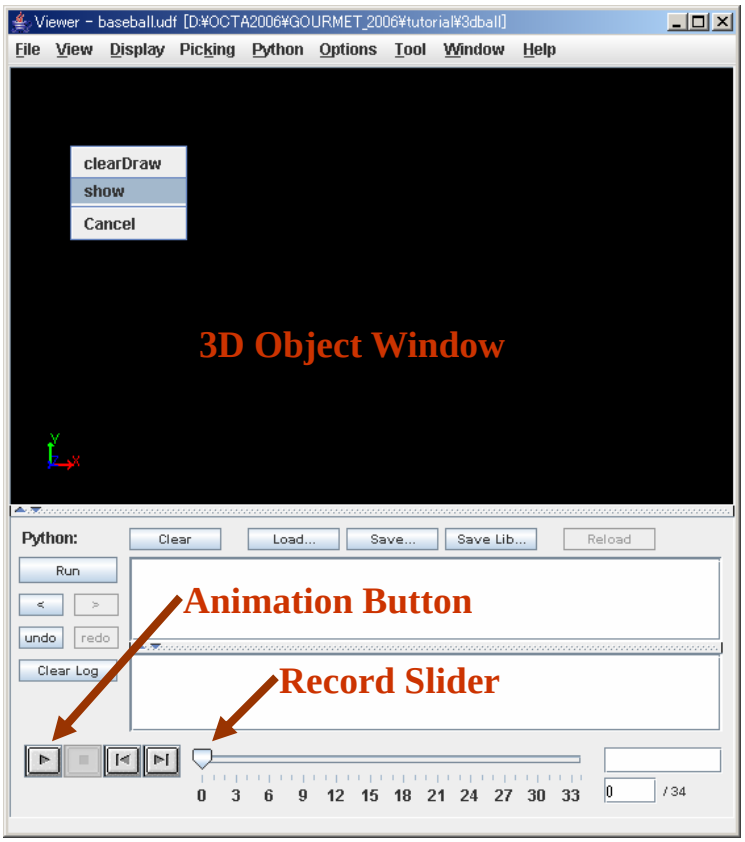


図 6: Viewer 画面

それでは、Python PanelのAnimationボタンを押して、解析結果をアニメーションしてみましょう。

ストライクッ!

"Backward [|<]"および"Forward [>|]"ボタンで、それぞれレコードを前後に移動することができます。この例では、解析結果がUDFレコードに時系列データとして保存されているので、レコードスライダーを使えば、どのレコードにでも移動できます。詳しい操作方法は、第8章をご覧ください。

次に、エディタに戻って2Dプロット機能を使ってみましょう。Window/Editorメニューを選択してください。Data Structure Windowで、"Calculated results"をマウスの右ボタンでクリックしてください。（これも、アクションスクリプトを実行する機能です）ポップアップメニューから、"x-y plot"を選択してください。

Note: マウスの右クリックでアクションを実行できるUDFパスは、ボールドで表示されます

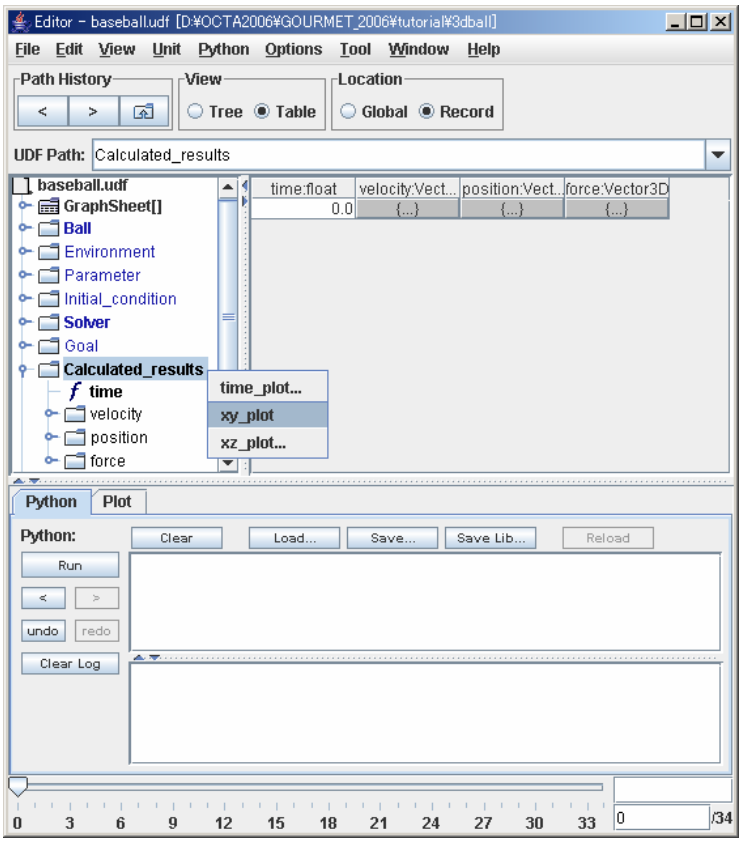


図 7: Plot アクション

x-yプロットされたボール位置がgnuplotウインドウに表示されます。

いくつかの方法でgnuplotツールを使うことができます。もっと詳しい操作方法は、第9章をご覧ください。

2.6 ツールを使いましょう

GOURMETのToolメニューとFileメニューには、いろいろなツールが準備されています。

・UDFファイルを他のマシンに転送するためのFile Transmitterツール

・純粋なPython環境を起動するためのPythonツール

・gnuplotアプリケーションを起動するためのGnuplotツール

・COGNACのUDFファイルにmolfileとPDBフォーマットのデータをインポートするための分子ビルダーツール

・他のフォーマットのテキストデータをUDFファイルにインポートするためのファイル変換ツ

ール

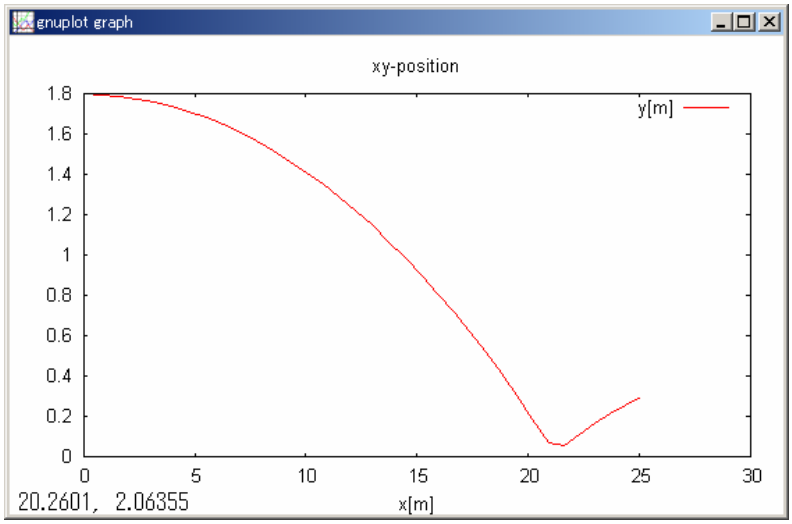


図 8: 2D プロット

詳しい操作方法は、第10 章をご覧ください。

第 3 章　起動時のオプション

3.1 システムの概要

GOURMET (Graphical Open UseR interface for Material design EnvironmenT) はUDFファイルを表示・編集・描画する機能があり、ユーザーフレンドリにUDFファイルを操作することを目的としたGUIシステムです。

GOURMETのGUI部分はJavaで開発されており、UDFファイルの読み書きはC/C++で開発されたライブラリが担当しています。また、３次元グラフィクスはJOGL経由でOpenGLライブラリを利用して描画しています。

UDFファイルはインタープリタ型の言語であるPythonにより操作され、グラフのプロット機能はgnuplotを使用しています。

GOURMETはJava、OpenGLおよびC++で構築されており、Pythonとgnuplotシステムで強化されています。

GOURMETはクライアント・サーバシステムとしても機能します。解析エンジン実行時およびリモートファイル転送時にはGOURMETはクライアントアプリケーションとなり、エンジンマネージャとデータマネージャは、それぞれサーバープログラムになります。エンジンマネージャはリモートおよびローカルのエンジンを制御し、データマネージャはローカル（GOURMET 側）とリモート（エンジン側）の間でUDFファイルを転送します。

そのため、GOURMETからエンジンの制御を行うには、エンジンを稼動させるマシン上でエンジンマネージャを起動する必要があります。

他のマシンへUDFファイルを転送するためには、転送先マシンでデータマネージャを起動する必要があります。

これらのマネージャの起動用スクリプトは以下の節で説明します。

GOURMETシステムの概要を図9に示します。

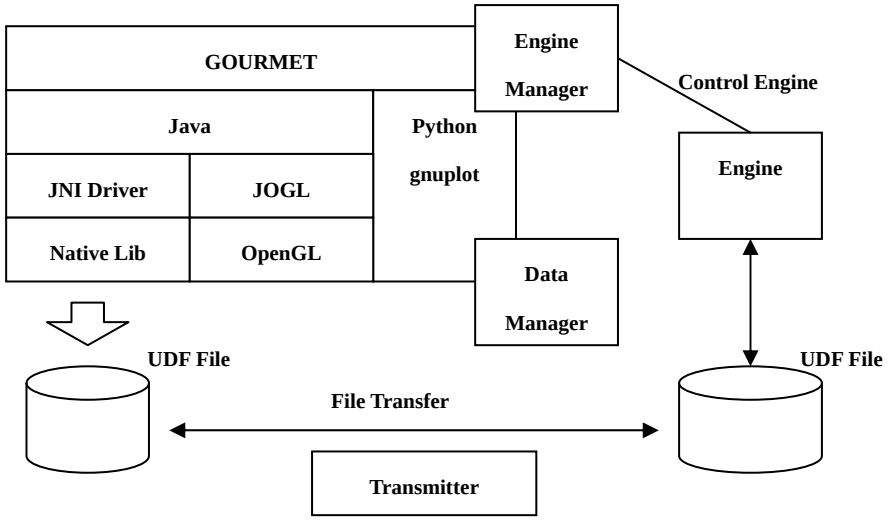


図 9: GOURMET システムの概要

3.2 環境変数

3.2.1 必須の環境変数

少なくとも１つの環境変数がGOURMETの起動に必要です。この環境変数はOCTAのインストーラにより自動設定されるはずです。

・PF_FILES

PF_FILES環境変数は、GOURMETのホームディレクトリのパスです。

3.2.2 オプションの環境変数

他にオプション環境変数がいくつかあります。

・PF_ENGINE

OCTAエンジンのホームディレクトリのパスで、GOURMETがOCTAエンジンの位置を知るために必要

です。

・UDF_DEF_PATH

UDFファイルの定義ファイルのあるパスです。複数のディレクトリパスを、OS依存の環境変数区切り文字で設定できます。（Windowsの場合は";"、Unix/Linuxは":"）

・UDF_ACTION_PATH

アクションファイルがあるディレクトリパスです。（OS依存の環境変数区切り文字で複数設定できます。）

・PYTHON_LIBDIR

GOURMETのPythonライブラリディレクトリです。

インストーラを使ってインストールすれば、PF_FILESおよびPF_ENGINEの環境変数が自動的に設定され、その他の環境変数は起動用のスクリプトファイルによって設定されます。

これらの環境変数の使い方については、次の節も参照してください。

3.3 起動用のシェルスクリプト

Javaで実装されたシステムは使用するメモリーサイズをjava実行環境の引数で指定することができます。デフォルトの使用メモリー量は十分でない場合が多いので、指定をしたほうがよいでしょう。Javaで使われる最大メモリーと初期メモリーサイズを-Xmx と-Xms オプションで指定します。GOURMETはUDFファイルの内容によっては非常に多くのメモリーを使用するため、実装メモリーよりも少ない範囲で十分なサイズを指定すべきです。

例：

マシンが512MBのメモリーを持っている場合、-Xmx256m と-Xms128m を指定。

3.3.1 起動オプション

GOURMETでは次の起動オプションをコマンドラインで指定できます。これらのオプションは起動スクリプトファイルで使用されています（gourmet.bat あるいは gourmet.sh）。

- **-DPF_FILES="GOURMETのホームディレクトリパス"**
  デフォルトとして使うディレクトリおよびファイルをここで指定されたディレクトリ以下から探す。
- **-DUDF_ACTION_PATH="アクションファイルディレクトリパス"**
  ディフォルトとして使うアクションファイルをここで指定されたディレクトリから探す。
- **-DPYTHON_LIBDIR="Python スクリプト保存ディレクトリパス"**
  PythonスクリプトをLoad/Saveするためのディフォルトディレクトリを指定する。
- **-DPF_MODULES="UDFコンバーター実行モジュールディレクトリパス"**
  UDFコンバーター実行モジュールなどが置かれたディレクトリを指定する。
- **-Djava.security.manager**
- **-Djava.security.policy=policy_file_path**
  これらの２つのオプションはセキュリティーのために準備されています。
  詳細は、エンジンマネージャの説明を参照してください。

3.3.2 Microsoft Windows

GOURMETを起動するには、スタートメニューのOCTA2010->Start GOURMET メニューを選択します。または、コマンドプロンプトウインドウで"gourmet"を実行するか、gourmet.bat をダブル

クリックします。また、開きたいUDF ファイルをgourmet.bat もしくはそのショートカットにドラッグ＆ドロップして起動することができます。

3.3.3 Linux

シェルターミナルウインドウで、インストーラによって作成された起動スクリプトを実行します。（OCTA200X/GOURMET_200X/gourmet）

3.4 エンジンマネージャ

エンジンの制御を行う場合、エンジンを稼動させるマシンでエンジンマネージャを起動する必要があります。

3.4.1 起動オプション

- -DSTAND_ALONE_MANAGER=yes
- -Djava.security.manager
- -Djava.security.policy=policy_file_path

これらの3 つのオプションはセキュリティーのために準備されています。これらの設定を行うと、そのマシンをスタンドアローンとして使うことができます。スタンドアローンとは、そのマシンをネットワーク上の他のマシンから使えなくすることを意味しており、エンジンマネージャはpolicy_file_path で定義されたセキュリティーポリシーを使います。

重要:
ローカルマシンでのみエンジンを動かし制御するだけであり（他のマシンのエンジンを使わない）、使用しているネットワーク環境が安全でないと思う場合、OS のシステムツールを使って、セキュリティーポリシーの設定をすべきです。
さらに詳しい説明が必要であれば、付録B エンジンマネージャのセキュリティーポリシーを参照してください。

3.4.2 Microsoft Windows

エンジンマネージャを起動するには、スタートメニューのOCTA2010->Start Engine Manager メニューを選択します。
または、コマンドプロンプトで"eng_man.bat"を実行します。

3.4.3 Linux とUnix

エンジンマネージャを起動するには、シェルターミナルウインドウでインストーラによって作成された"OCTA2010/GOURMET_2010/eng_man"を実行します。

3.5 データマネージャ

他のマシンとUDFファイルを転送するには、データマネージャをそのマシン（転送先マシン）で稼動させておく必要があります。

3.5.1 Microsoft Windows

データマネージャを起動するには、コマンドプロンプトで"dat_man.bat"を実行します。

3.5.2 Linux とUnix

データマネージャを起動するには、シェルターミナルウインドウで
"OCTA2010/GOURMET_2010/dat_man"を実行します。

3.6 GOURMETの終了

GOURMETを終了するには、GOURMETのFile/Exitメニューを選択するか、全てのGOURMETウインドウを閉じます。

第 4 章　UDFを編集するには

UDFファイルを編集するには、File/Open... メニューを選択し、File openダイアログでそのファイルを選択します。図10は、エディタでUDFファイルを開いた状態を示しています。

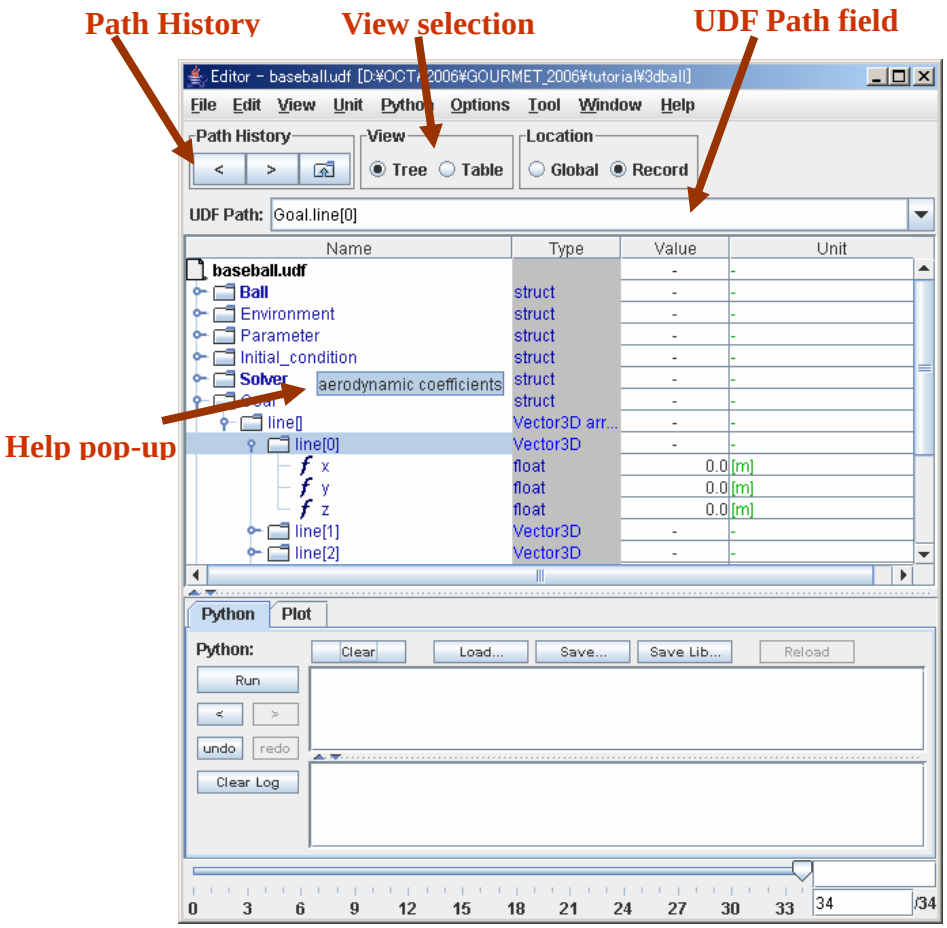


図 10: エディタの Tree ビューで UDF ファイルを開いた状態

・Path Historyを用いて、操作履歴からUDFデータパスを選択したり、履歴上の移動を行うことができます。

・Viewラジオボタンを用いて、TreeまたはTableの表示形式を選択します。

・Locationラジオボタンを用いて、GlobalまたはRecordのUDFのデータロケーションを選択します。特にデータ値を編集するときには、どのレコードにあるデータに対して編集しているかは重要です。

・UDF Pathフィールドに任意のUDFパスを入力してUDFデータを選択することができます。

・ヘルプ定義されたUDFデータ変数の上にマウスカーソルを重ねると、ヘルプ内容がポップア

ップされます。

・ヘルプの内容にインターネットやローカルディスク上のファイルへハイパーリンクが定義されている場合、[Control]+マウス右クリックでポップアップされるメニューで選択したファイル内容がシステムディフォルトブラウザで表示されます。

4.1 編集モード

4.1.1 編集モード

UDFオブジェクトの値を変更するには、編集モードを用います。編集モードでは値の存在しているものと値を作成することのできるもの全てのオブジェクト（構造）がリストされます。（非編集モードおよび定義モードはOCTA2003以降削除されましたので、デフォルトで編集モードになります。）

4.2 ビュー形式の選択

4.2.1 Treeビュー

UDFデータの内容を見たいならば、Treeビューを選択します。Treeビュー形式はデータの構造および値をツリー形式で表示します。

Treeビューで多くの配列数を持つデータを表示しようとするとツリーの展開に時間がかかるため、指定した数だけデータ項目が表示されようになりました。配列の最初だけ表示されている例を図11に示します。例では配列value[]が１０個まで表示されており、以降は「...」で示されています。「...」部分をダブルクリックするか「...」のツリー枝分かれ部分をマウスで左クリックすると次の１０個の配列が表示されます。この例では配列が展開される数は１０個づつであるとしていますが、配列の展開数を変更することができます。後章の「4.6.1 エディタのPreferencesダイアログ」のTreeタブにある"Number of expanding:"で設定します。デフォルトでは配列の展開数は1000になっています。

また、多少時間がかかっても一度に全ての配列が見たい時や配列の一部を見たいときは、「...」部分をマウスで右クリックすると図12に示すポップアップが表示されます。このポップアップには現在表示されているデータに続く次の配列のインデックスが示されます。OKボタンを押すとポップアップに表示されている配列インデックスに対応するデータがTreeビューに表示されます。

データ配列数が設定されているツリーの展開数よりも多い時に、上記の方法で全配列を表示させた場合、図13に示すように配列の最後に「EndOfArray」マークが表示されます。このマークを右クリックすると図12と同様のポップアップが表示されますので、閉じるボタンを押すと最初のツリー展開状態に戻ります。

データ配列数が設定されているツリー展開数よりも少ない時には、「...」や「EndOfArray」マークは表示されません。

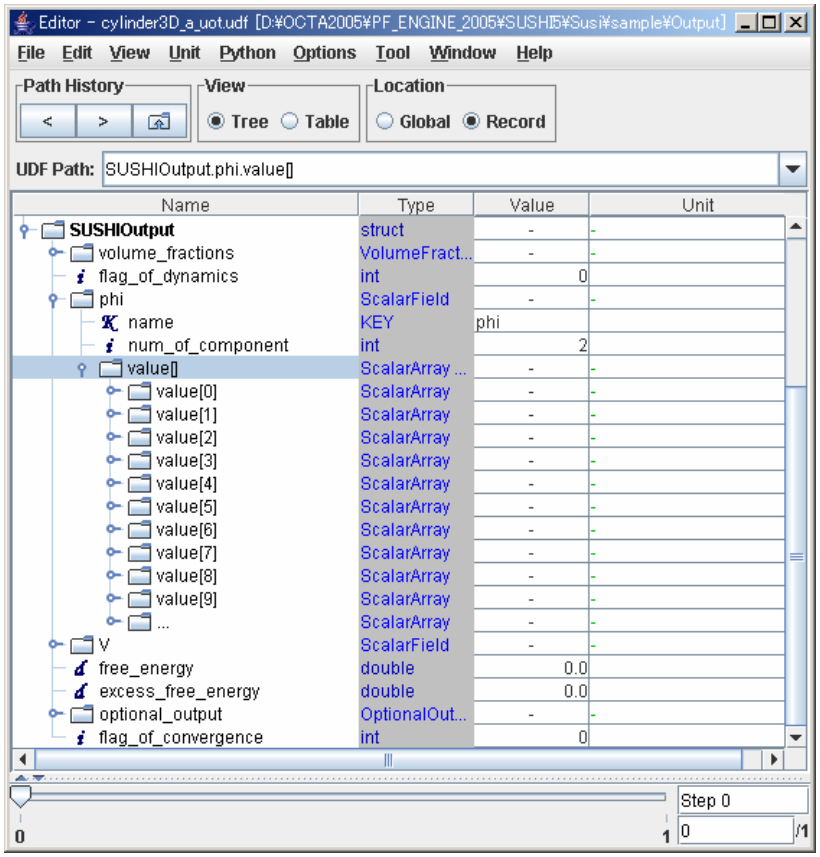


図 11: エディタの Tree ビューで配列を展開した状態

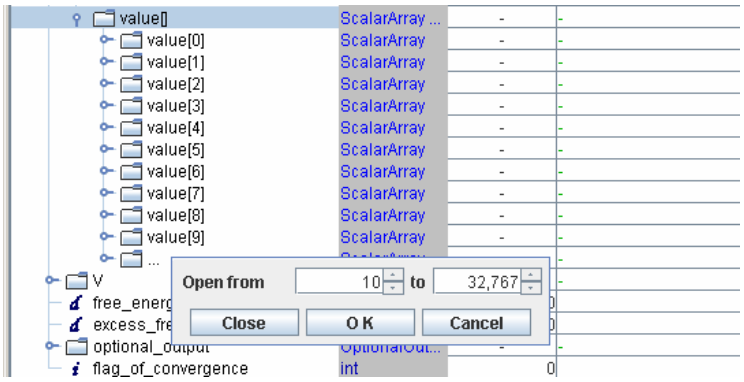


図 12: 「...」を右クリックした時のポップアップ

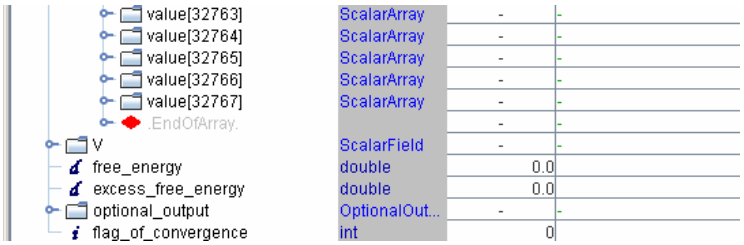


図 13: 「EndOfArray」マーク

4.2.2 Tableビュー

UDFデータの構造を見たいならば、Tableビューを選択します。Table ビュー形式はリレーショナルデータベースの表示形式であり、他のアプリケーションとの間で簡単なコピー・ペースト([Ctrl]+C、[Ctrl]+V) 操作でデータを交換することができます。 (注[1])

図14 は、UDFファイルをTableビューで見た例を示しています。

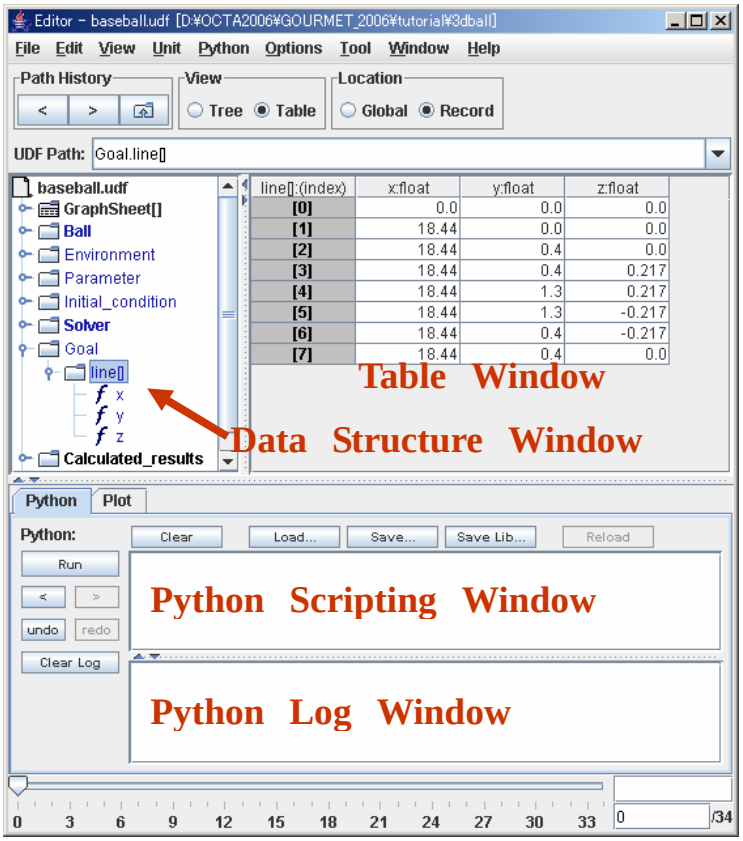


図 14: エディタの Table ビューで UDF ファイルを開いた状態

[1] Table ビューで複数のセルを選択する場合、Java のバージョンによって選択方法が異なります。Java 1.5 以降では違うセルを一旦選択しておいてから、最初の選択から選択終了までの間、Esc キーを押しながらセル範囲をマウスでドラッグします。

4.3 データロケーションの選択

UDFファイルは同じデータ名の異なる値を初期値データとゼロ個以上のレコードデータに持つことができます。ロケーションタイプとは初期値にあるデータを参照するのか、レコードにあるデータを参照するのかの区別です。GOURMETはUDFデータ名の表示色でロケーションタイプとデータ値の有無を区別します。さらにUDFデータの初期値には、2つのタイプが使われることがあります。初期値のみにデータを持つように"global_def"で定義されている場合と初期値およびレコードにデータを持てる"def"で定義されている場合です。"global_def"で定義されている場合、データ名が青で表示されます。"def"定義のUDFデータ名は緑で表示されます。"def"定義のUDFデータがレコードにある場合は黒で表示されます。データ値が存在しなければ、淡色で表示されます。

図15はデータロケーションとしてRecordsが選択された場合のTree ビューの例を示しています。'global_value'は、global_defとして定義されているため青で表示されます。'common_num'および'common_pos'は、初期値は存在しますが、このレコードには値が存在しないので、緑で表示されています。他のデータはこのレコードに値が存在するため黒で表示されているます。

4.3.1 Globalロケーション

グローバルデータを変更するには、Globalラジオボタンを選択します。

Note:グローバルデータとは、OCTA2002以前には初期値（Initial Data）と呼ばれていました。

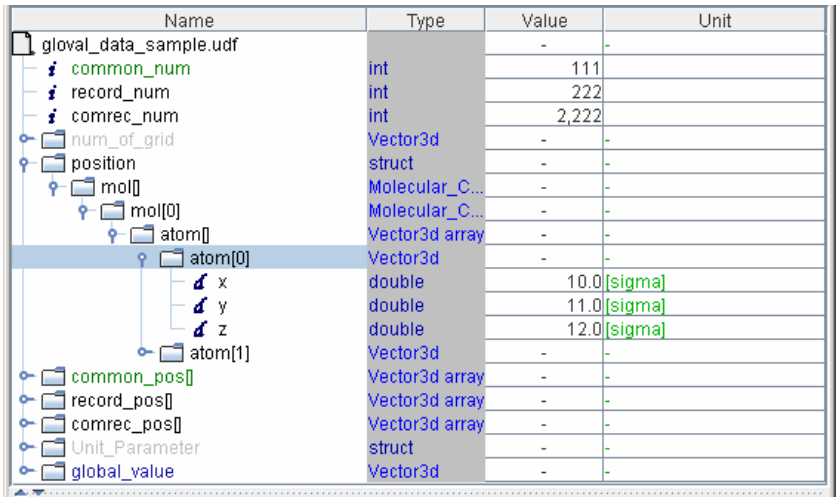


図 15: データロケーションによる UDF オブジェクトの色表示

4.3.2 Record ロケーション

レコード値を変更したり追加するには、Recordsラジオボタンを選択します。

Recordsを選択している場合、レコードデータが存在すれば、Records Sliderがエディターの下部に表示されます。Record Sliderをマウスで動かして、レコード位置を移動することができます。

4.4 File メニュー

図16 にエディターのFile メニューを示します。

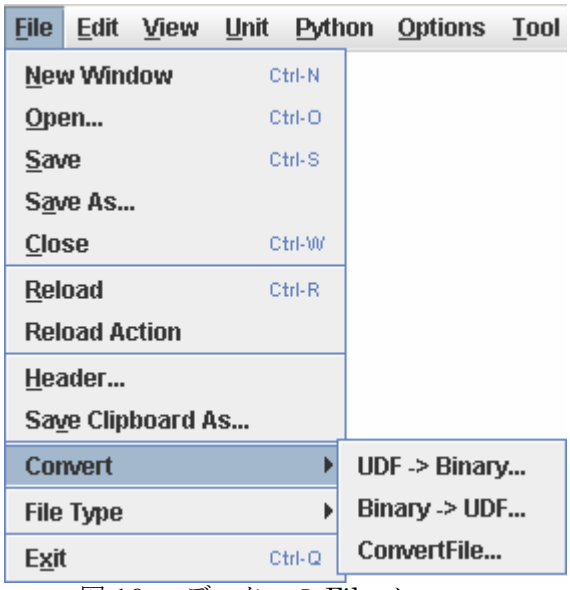


図 16: エディターの File メニュー

以下の作業を行うための操作をFileメニューから選択します。

- **New Window**：他のUDFファイルを編集したり参照するための画面を表示します
- **Open**：UDFファイルを開きます（Autorunアクションが定義されていれば実行します）
- **Save**：現在開いているUDFファイルに対する変更を保管します
- **Save As...**：現在開いているUDF ファイルに対する変更を別名で保管します
- **Close**：現在開いているUDFファイルを閉じます
- **Reload**：現在開いているUDFファイルの全てのデータをリロードします（Autorunアクションが定義されていれば実行します）
- **Reload Action**：現在のUDFファイルに使われているアクションファイルをリロードします（Autorunアクションが定義されていれば実行します）
- **Header...**：現在開いているUDFファイルのUDFヘッダ情報を編集または参照する画面を表示します
- **Save Clipboad As...**：現在のクリップボードの内容をテキストファイルに保管します
- **Convert**：テキスト形式のUDFファイルとバイナリ形式のUDFファイルを変換します。またファイルフィルターを使って外部ファイルをインポートする画面を表示します。
- **File Type**：開いているUDFファイルの形式を表示します。

・**Exit**：GOURMETを終了します

4.4.1 UDFヘッダーを編集するには

File/Header...メニュー選択により、UDF ヘッダーを編集することができます。図 17 に UDF ヘッダーダイアログの例を示します。

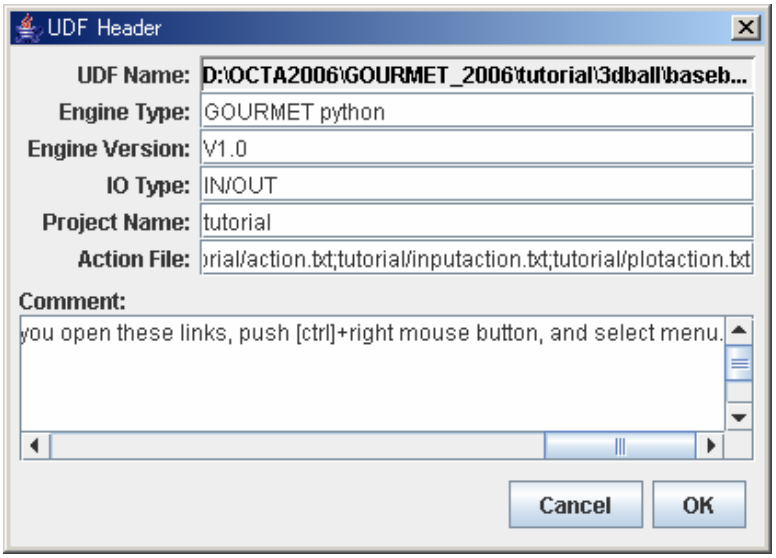


図 17: UDF ヘッダーダイアログ

4.4.2 ファイルコンバータの使い方

File/Convert/ConvertFile... メニュー選択により、外部固定フォーマットのテキストデータを開いている UDF ファイルに取り込むことができます。

図 18 に ConvertFile ダイアログの例を示します。

・**Data File**

変換元のデータファイル

タブ区切りデータやもっと複雑な NASTRAN の bulk ファイルなど

・**Rule File**

変換のためのフィルタルールファイル

いくつかのフィルタの例が$PF_FILES/filter/ディレクトリにあります。

以下の例は、データファイルの第２，３，４列データを、それぞれ atoms[].Position.x atoms[].Position.y atoms[].Position.z に代入します。

フィルタルールファイル例:

```
# simple example for convert molecule data from tab delimitted text file
LABEL . atoms[].Position.x atoms[].Position.y atoms[].Position.z
```

・Input UDF

変換 UDF 定義ファイル

前述の例では、以下に示す定義を持つ入力 UDF を使用します。

入力 UDF ファイルの例:

```
¥begin{def}
class Coodinate:{x:double, y:double, z:double}
atoms[]: {
  Position: Coodinate
}
¥end{def}
```

・Output UDF

変換結果出力 UDF ファイル

変換ルールの記述は単純なので、新たなルールを簡単に作ることができます。ファイルフィルタの詳細は、付録 D を参照してください。

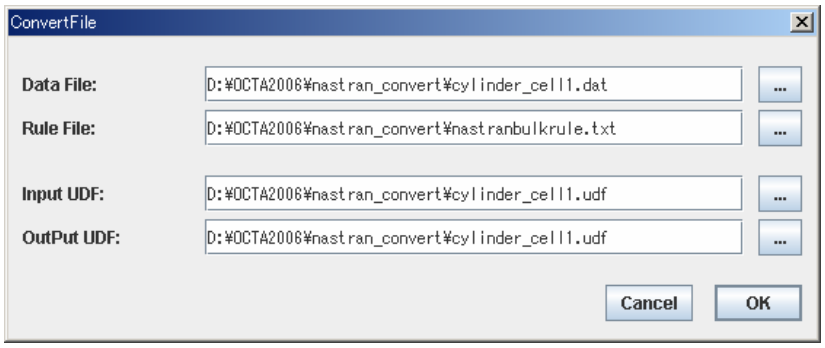


図 18: ファイルコンバータダイアログ

4.4.3 テキスト形式UDFとバイナリ形式UDFの使い方

GOURMET は十分大きなテキスト形式 UDF ファイルを扱えますが、100MB を超えるような多数のレコードデータを持つ UDF ファイルになると、読み込みに時間がかかるようになります。これを解決するのがバイナリ形式 UDF です。

バイナリ形式の UDF ファイルは、拡張子として.bdf を用います。バイナリ UDF は文字通りデータをバイナリ形式で持つ他、各レコードデータの先頭にレコード内の全データサイズ情報を持っているため、多数のレコードを持つ UDF ファイルの読み込みが高速に行えます。

テキスト形式UDFからバイナリ形式UDFへの変換はメニューからFile/Convert/UDF->Binary...を選択することにより実行することができます。バイナリ形式 UDF からテキスト形式 UDF への変換にはFile/Convert/Binary->UDF...から行えます。

4.5 Edit メニュー

図 19 はエディタの Edit メニューを示しています。

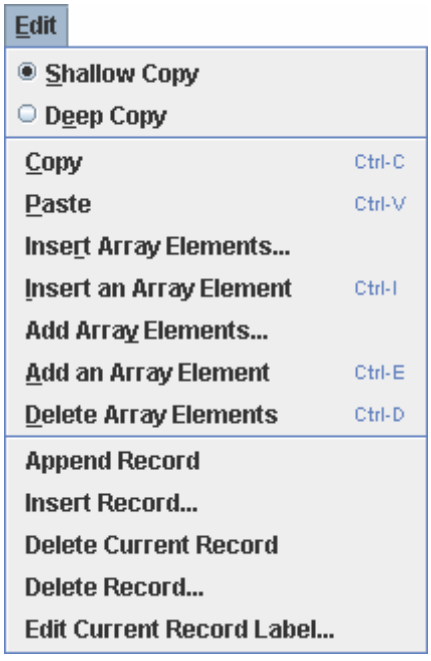


図 19: エディタの Edit メニュー

Edit メニューを使用して、以下の作業を行えます。

・Shallow Copy mode：データを表形式でコピーするモードに設定する

・**Deep Copy mode**：データを構造をもつオブジェクト形式でコピーするモードに設定する
・**Copy**：選択されたデータを現在のコピーモードで、クリップボードにコピーする
・**Paste**：クリップボードから選択された場所にデータを貼り付ける
・**Insert Array Elements...**：選択されている配列データに複数の新しい要素を挿入する
・**Insert an Array Element**：選択されている配列要素の前に１つの新しい要素を挿入する
・**Add Array Elements...**：選択されている配列要素の後に複数の新しい要素を追加する
・**Add an Element**：選択されている配列要素の後に１つの新しい要素を追加する
・**Delete Array Elements**：選択されている複数の配列要素を削除する
・**Append Record**：存在するレコードの最後に新しいレコードを追加する
・**Insert Record...**：任意の位置に複数の新しいレコードを挿入する
・**Delete Current Record**：現在のレコードを削除する
・**Delete Record...**：任意の位置の複数レコードを削除する
・**Edit Current Record Label**：現在のレコードの名称を編集する

4.5.1 Copy/Pasteモードの使い分け

コピー([Ctrl]+C) とペースト([Ctrl]+V) 操作は他のアプリケーションと同様です。さらに、2 つの方式（Shallow または Deep）からコピー・ペーストモードを選択できます。

・Shallow コピー・ペーストモード

Shallow コピー・ペーストモードは他のアプリケーションと同じく、見えている通りに表形式でコピー・ペーストを行います。GOURMET は、このモードの時に、他のアプリケーションとの間でクリップボードにあるデータを交換できます。

・Deep コピー・ペーストモード

UDF ファイルは構造化されたデータを持つことができるため、GOURMET には、選択されたデータ全体を 1 つのオブジェクトとして扱う Deep コピー・ペーストモードが準備されています。Deep コピー・ペーストモードでは、クリップボードデータは Python のリスト形式で表現されます。

4.6 Viewメニュー

図 20 はエディタの View メニューを示しています。
View メニューを選択して、以下の作業を行えます。

・**Show Global**：Global ロケーションのデータを表示します
・**Show Record**： Record ロケーションのデータを表示します
・**Preferences...**：Tree ビューと Table ビューの表示オプションを設定します

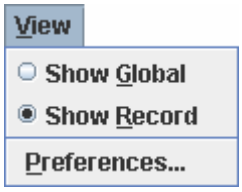


図 20: エディタの View メニュー

4.6.1 エディタのPreferencesダイアログ

エディタの表示形式を Preferences メニューで変更することができます。

・**TreeView Tab および TableView Tab**
図 21 および図 22 は、それぞれ TreeView タブおよび TableView タブを選択した Preferences ダイアログを示しています。

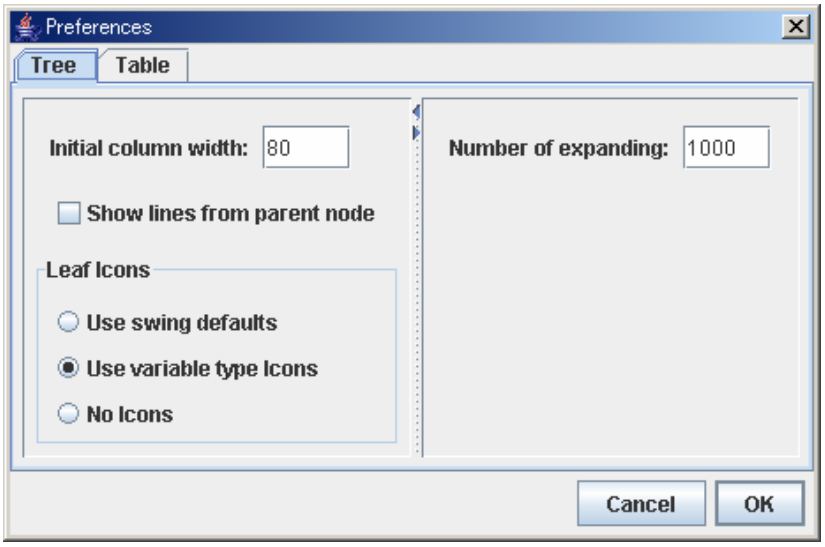


図 21: Preference/TreeView タブ

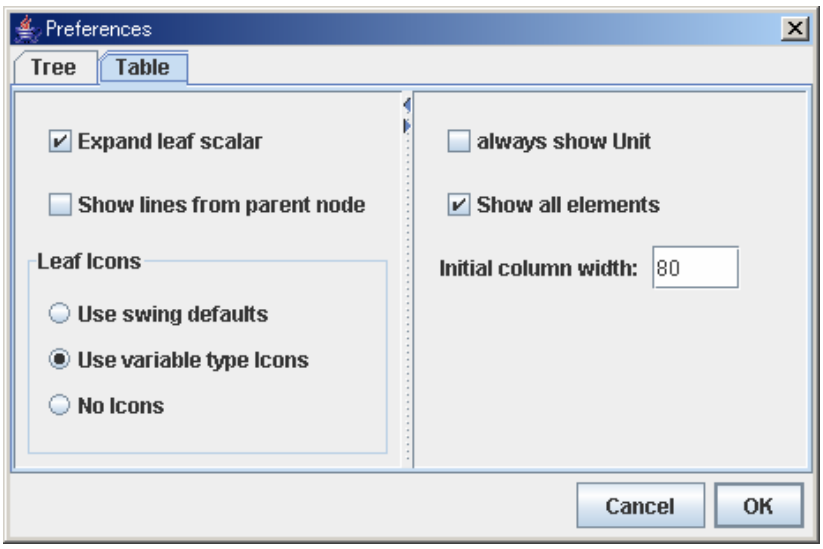


図 22: Preference/TableView タブ

4.7 単位変換

開いている UDF が単位の定義を持つならば、Tree/Table ビューのいずれにおいても、単位変換を行うことができます。

Table ビューでは、変換したいデータフィールドのフィールド名をマウスで右クリックします。Tree ビューでは、変換したいデータフィールドの Unit フィールドをマウスで右クリックします。図 23 に Unit 変換ダイアログの例を示します。

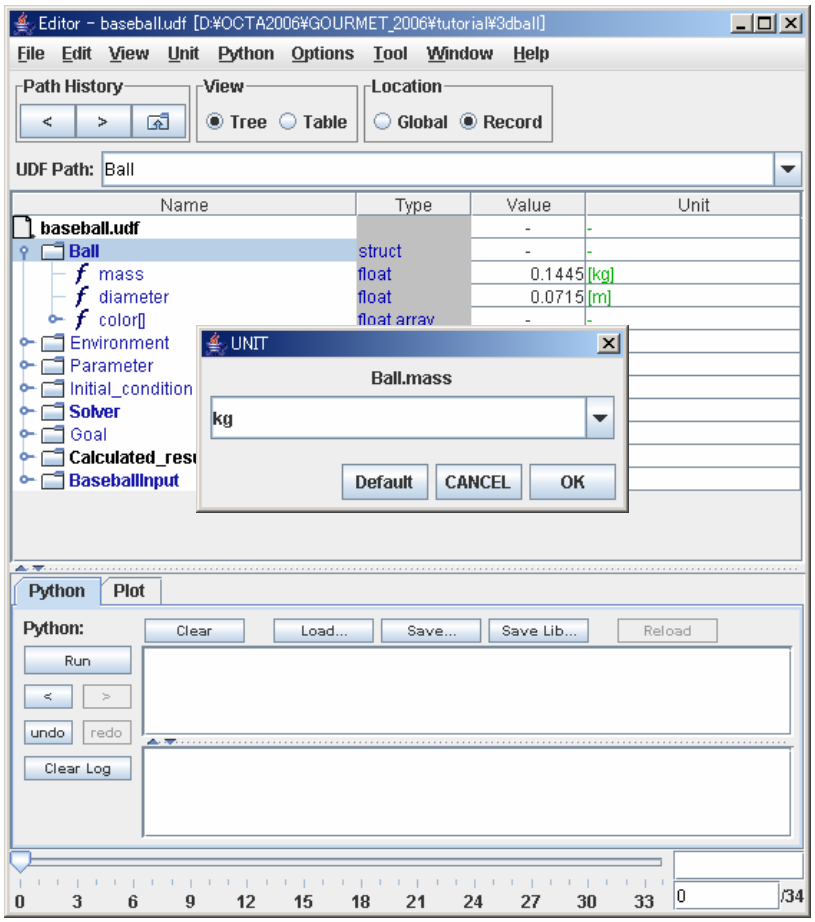


図 23: Unit Conversion ダイアログ

4.8 エディタ使用のTips

・UDF Path フィールドによるフィルタリング

UDF Path フィールドはフィルタリング機能を持っています。Table ビューで配列インデックスを持つ UDF シンボルを入力すれば、図 24 に示すようなフィルター結果が得られます。

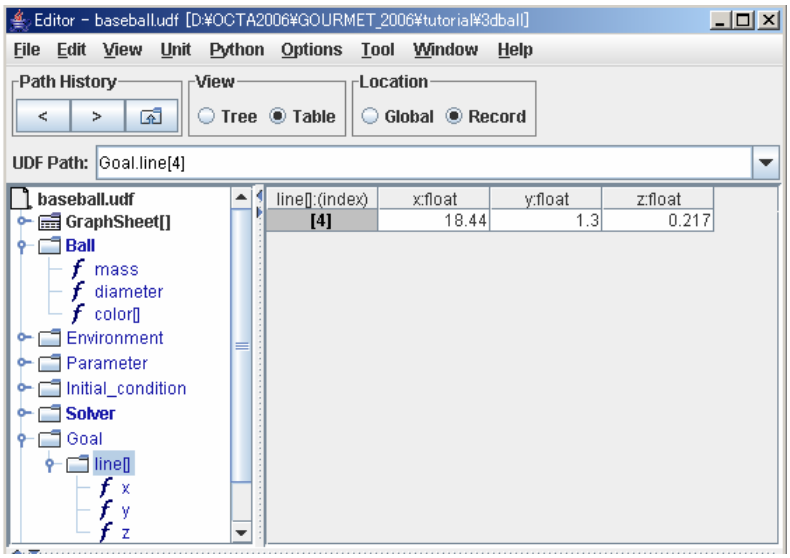


図 24: UDF Path フィールドによるフィルタリング

・Table ビューにおける KEY 値の表示

配列オブジェクトに KEY が定義されている場合、Table ビューでは図 25 のように、配列のインデックスと KEY 値が表示されます。（"[0]"と"AGE"）

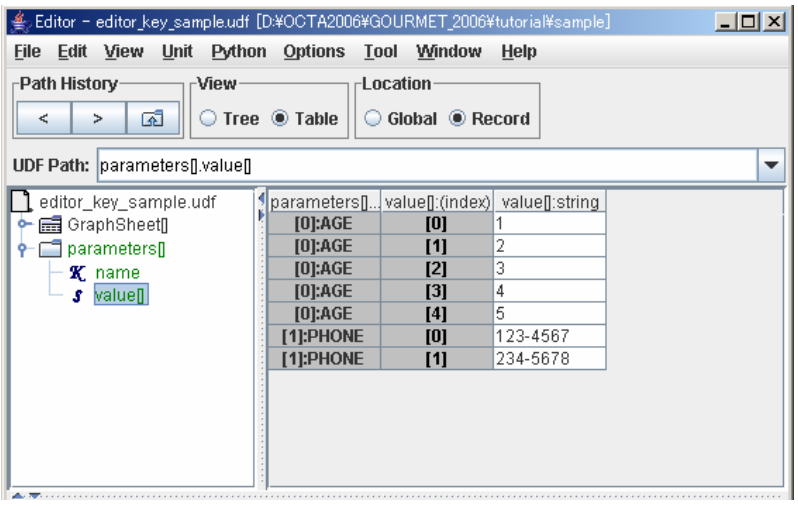


図 25: Table ビューにおける KEY 値の表示

・Select 値によるデータ選択機能

Select 型として定義されているデータは、入力項目リストから選択するだけでデータ入力をすることができます。また入力項目を選択すると、同じ階層に定義されている選択した値と同じ名称のデータ構造だけが表示され、他のデータ構造は隠れます。

・配列要素の挿入と削除

Table ビューでは、同じノード内の配列要素だけを挿入・削除することができます。これらの変更可能な要素は、インデックス番号が太字体で表示されてます。

・アクションの実行

アクションに関連付けられた UDF データ名は、太字で表示されます。すなわち、太字のデータ名部分を右クリックすることにより、関連付けられたアクションが実行できます。

・UDF 変数の説明文表示

UDF 変数名の上にマウスポインタを止めておくと、その内容に関する説明文が UDF ファイルの help として定義されていればポップアップ表示されます。その内容に下線表示された URL リンクが含まれている場合、[Ctrl]+マウス右ボタンでメニューが表示され、選択したリンク項目を開くことができます。

この機能は UDF ファイル名のコメントをポップアップ表示した場合も同様に操作できます。具体例は、Tutorial/3dball/baseball.udf を開き、左上のファイル名（baseball.udf）にマウスポインタを当てて上記の操作を行います。

・多次元配列へのデータの挿入

多次元配列型のデータ要素を追加する場合、上位の要素から挿入していく必要があります。例えば、空の array2[][] に 10x10 の要素を GUI メニューを使って挿入するには、array2[] に 10 個の要素を挿入した後、array2[0][] に 10 個を挿入し、各要素に値を設定していく。。。といった操作を行うことになります。以下のような python スクリプトを使って要素の追加と同時に値を設定した方が簡単です。

```
for i in range(10):
  for j in range(10):
    $array2[i][j] = i*10 + j
```

・２次元配列の表形式表示

Table ビューを使用している時、表示対象データが array[][] のような２次元配列型なら

ば、その全(i,j)要素が縦横に配列された表形式で表示することができます。UDF Path フィールドで表形式表示を指定するには、例えば b[1].a[][] のように２次元配列の UDF Path がフィールドに表示された状態で Enter/Return キーを押します。Table ウインドウでは、構造体の中に2 次元配列型のデータがある場合にマウスで[...]をクリックした時点で表形式表示されます。表形式表示を元の１次元表示に戻すには Tree ウインドウで、そのデータをクリックします。Tree ウインドウでは表形式表示はできません。

第 5 章　単位系の使い方

5.1 Unit メニュー

図 26 は Unit メニューを示しています。

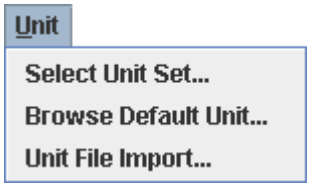


図 26: Unit メニュー

開いた UDF ファイルに単位系が定義されている場合、エディタ画面で Unit メニューを使うことができます。

- **Select Unit Set...**：変換可能な単位系を選択し、その単位系で値を表示する
- **Browse Default Unit...**：定義されているデフォルト単位リストを表示する
- **Unit File Import...**：Unit ファイルに定義された別の単位系をインポートする

重要:
Unit 変換は GOURMET の中だけで、定義済み単位系の間の変換を行うものであり、UDF ファイルのデータ値自体は常にエンジンで定義されている単位系のままです。

5.2 単位系選択の使い方

図 27 は、Select Unit Set ダイアログを示しています。変換先単位系を選択し、全てのデータ値をその単位系で表示することができます。デフォルト単位系は、エンジンが定義している単位系です。

図 27: Select Unit Set ダイアログ

5.3 単位系のインポート方法

Unit File import... メニューを使って、単位系定義を GOURMET にインポートし、変換先単位系として使用できます。

単位系ファイルの定義方法については、UDF 文法リファレンスを参照してください。

単位系定義ファイル例を以下に示します。

```
¥begin{unit_system}{"MYSI"}
CONSTANT=9.9999
[kg]
[myLength]=[10*m]
[myTime]=[ms]
[A]
[myTemp] = [mK]
[mol]
[cd]
[rad]
[sr]
[Hz]=[1/s] // frequency
[N]=[kg*m/s^2] // force
[Pa]=[N/m^2] // pressure
[J]=[N*m] // energy
[W]=[J/s] // power
[V]=[W/A] // voltage
[Wb]=[V*s] // magnetic flux
[T]=[Wb/m^2] // magnetic flux density
¥end{unit_system}
```

5.4 エンジン単位系の参照

″Browse Default Unit...″メニューで、図 28 に示すように、解析エンジン用 UDF ファイルで定義している単位系を参照することができます。

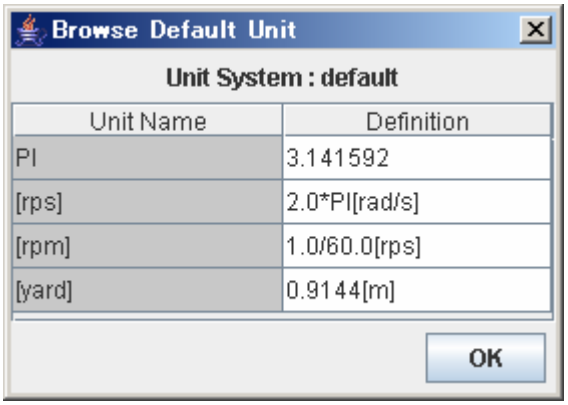


図 28: Browse Default Unit ダイアログ

第６章　Pythonスクリプトを使うには

GOURMET は Python スクリプト言語を用いてＵＤＦに記述されているデータに様々な操作をすることができます。データの値を参照、編集、単位変更、レコード追加、３次元図形描画およびグラフ表示などができます。GOURMET の全てのスクリプト処理は Python スクリプト処理システムにより実行されます。そのため、エディタとビューワはそれぞれ Python Scripting ウインドウを備えています。

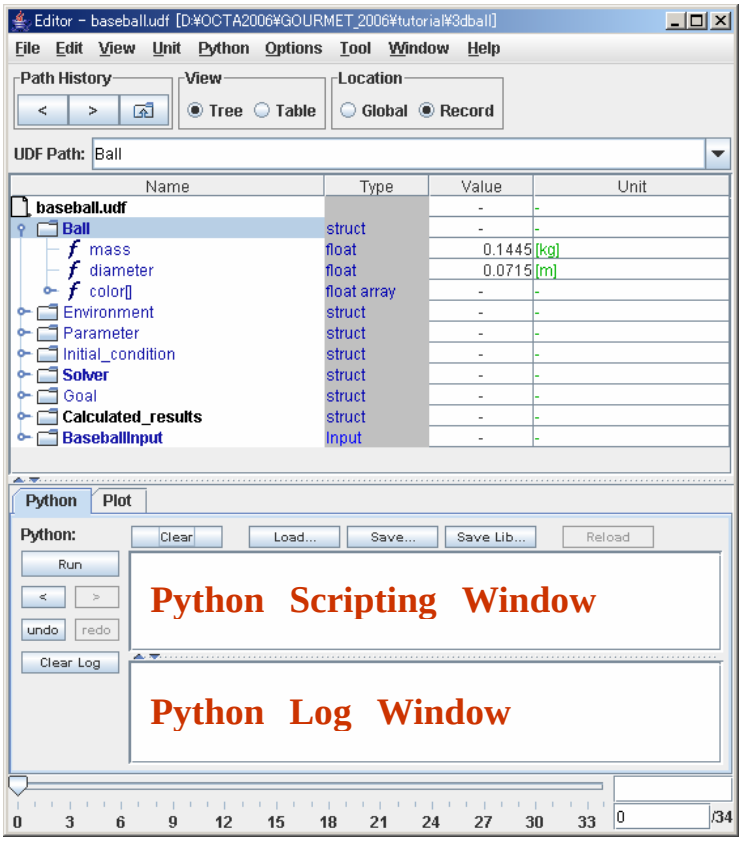


図 29: エディタの Python Scripting ウインドウ

以下の節では、GOURMET における次のような 4 つのタイプのスクリプトについて説明します。

・エディタの Python Scripting ウインドウで直接スクリプトを実行する
・ビューワの Python Scripting ウインドウで直接スクリプトを実行する
・エディタでアクションを実行する
・ビューワでピッキングアクションを実行する

6.1 Python PanelのツールとPythonメニュー

エディタ画面のPythonメニューまたはエディタ画面の下にあるPythonタブでは以下のスクリプトに関する操作を実行できます。

- **Run**：Python Scriptingウインドウのスクリプトを実行する。実行結果は（エラーメッセージを含め)Python Logウインドウに出力される。スクリプトが正常に実行できた場合、GOURMETはそのスクリプトを履歴に記憶する。記憶されているスクリプト履歴を再利用することができる。エディタ画面の下にあるPythonタブの"Run"ボタンを押すことによりスクリプトを実行することができる。
- **Clear**：Python Scriptingウインドウの内容をクリアする。
- **Load...**：Python Scriptingウインドウに外部ファイルの内容を取込む。
- **Save...**：Python Scriptingウインドウの内容を外部ファイルに保存する。
- **Save Lib...**：Python Scriptingウインドウの内容をPython preparserが変換した結果を表示し、編集結果を外部ファイルに保存する。この機能で保存されたPythonスクリプトは、純粋なPython 環境でも実行できる。
- **Save History...**：Python Scriptingウインドウの履歴を外部ファイルに保存する。保存したスクリプト履歴は、将来のセッションで再利用できる。
- **Load History...**：Python Scriptingウインドウの履歴を外部ファイルから取り込む。過去に保存されたスクリプト履歴を再使用することができる。
- **Previous History (<)**：Python Scriptingウインドウに履歴の直前の内容を表示する。
- **Next History (>)**：Python Scriptingウインドウに履歴の次の内容を表示する。
- **Clear Log**：Python Logウインドウをクリアする。
- **AutoRun mode**：onにすると、画面下部のスライダーで表示レコードを変更時には、自動的に「Run」が実行される。ビューワはスクリプト実行により描画されているので、常にAutoRunモードとなっている。

6.2 エディタにおけるPythonスクリプト実行

Python Scriptingウインドウでは、Pythonシステムに用意されている全てのスクリプト処理を実行できます。最も簡単な例では、次のスクリプトをタイプするかロードして、実行します。

例: print "hello GOURMET"

同じ方法で GOURMET Python スクリプト拡張機能も実行できます。ここで、スクリプト拡張機能とは GOURMET で UDF データ階層にアクセスするための Python 拡張機能を意味します。

Python 拡張機能には UDF ファイルおよびデータを扱う便利な機能が豊富に準備されています。1 つの例を示すと、"$"が前置された UDF 名を使用して UDF データにアクセスすることができます。

例:

```
print $Ball.mass
$Environment.gravity = 9.8
```

6.3 ビューワにおけるPythonスクリプト処理

ビューワにおいても、エディタと同じスクリプトを実行できます。これに加えて、ビューワでは３次元描画機能を使用することができます。３次元描画機能には、sphere のような基本的な形状描画関数および molecule や mesh のような特殊な構造を描画する関数が準備されています。

例:

```
pos = $Initial_condition.position
attr = [1.0, 0.0, 0.0, 1.0, $Ball.diameter]
sphere(pos, attr)
```

表 1 に GOURMET で使用できる基本描画関数の一覧を示します。より詳細には、Python スクリプトマニュアルを参照してください。

関数名	備考
line(coordinate1,coordinate2,[r,g,b,a]))	指定された RBG と透明度で描画する
line(coordinate1,coordinate2,attribute_id)	指定された描画属性 ID で描画する
point(coordinate,[r,g,b,a])	以下同様
polygon(coordinate list,[r,g,b,a])	
polyline(coordinate list,[r,g,b,a])	
disk(coordinate1,[r,g,b,t,radius,vx,vy,vz])	
ellipse1(coordinate1,[r,g,b,t,a,b,vx,vy,vz])	
ellipse2(coordinate1,coordinate2,[r,g,b,t,a,vx,vy,vz])	
cylinder(coordinate1,coordinate2,[r,g,b,t,radius])	
sphere(coordinate1,[r,g,b,t,radius])	

ellipsoid1(coordinate1,[r,g,b,t,a,b,c,vx,vy,vz])	
tetra(coordinate1,coordinate2,coordinate3, coordinate4,[r,g,b,t])	
cube(coordinate1,length,[r,g,b,t])	
cone(coordinate1,coordinate2,[r,g,b,a,radius])	
arrow(coordinate1,coordinate2, [r,g,b,t,radius,height])	
text(coordinate,message,[r,g,b,t,size])	
clearDraw()	

表 1: 描画関数

6.4 エディタにおけるアクション

エディタで太字表示されたUDFオブジェクトをマウスで右クリックすると表示されるポップアップメニューでアクションを選択できます。アクションの内容はアクションファイルに記述します。GOURMET は、UDF ヘッダ情報に記述されたアクションファイルを、UDF ファイルのあるディレクトリおよび環境変数 UDF_ACTION_PATH で設定されたディレクトリから探してロードします。

アクションファイルの例: (例題 : GOURMET_XXXX/tutorial/3dball/baseball.udf)

```
action Ball: setColor(BallColor="0|1|2|3|4|5") : color = eval(BallColor)
action Ball: show() : ¥begin
color=[1.0, 0.0, 0.0, 1.0, $Ball.diameter]
def drawGoal():
  lines = $Goal.line[]
  polyline(lines,0)
drawGoal()
if $Calculated_results.position.y > 0:
  pos = $Calculated_results.position
  sphere(pos, color)
¥end
```

この例では、エディタで'Ball'をマウスで右クリックすると、(図 30 のような) ポップアップメニューが表示されます。ここで setColor アクションを選択すると、ボールの色情報が設定されます。

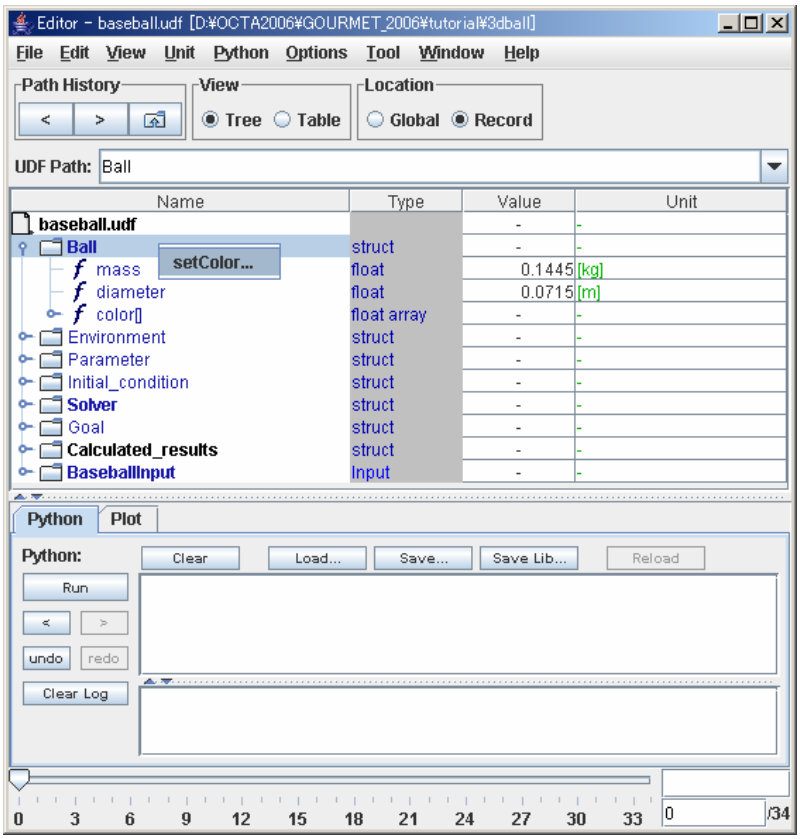


図 30: エディタにおけるアクション例

アクションファイル定義の詳細については、付録 C を参照してください。

6.5 ビューワにおけるアクション

ビューワでは、特別な描画関数(CognacAtom & CognacBond あるいは setDrawRelation & resetDrawRelation)で描画された UDF オブジェクトに関連付けられたアクションのリストを表示・選択するためのポップアップメニューを表示することができます。

例えば、図 31 に示すように、[Ctrl]キー を押しながら、マウスの左クリックで分子構造をピックすると、アクション選択のためのポップアップメニューが表示され、選択したアクションを実行することができます。

アクションファイル定義の詳細については、付録 C を参照してください。

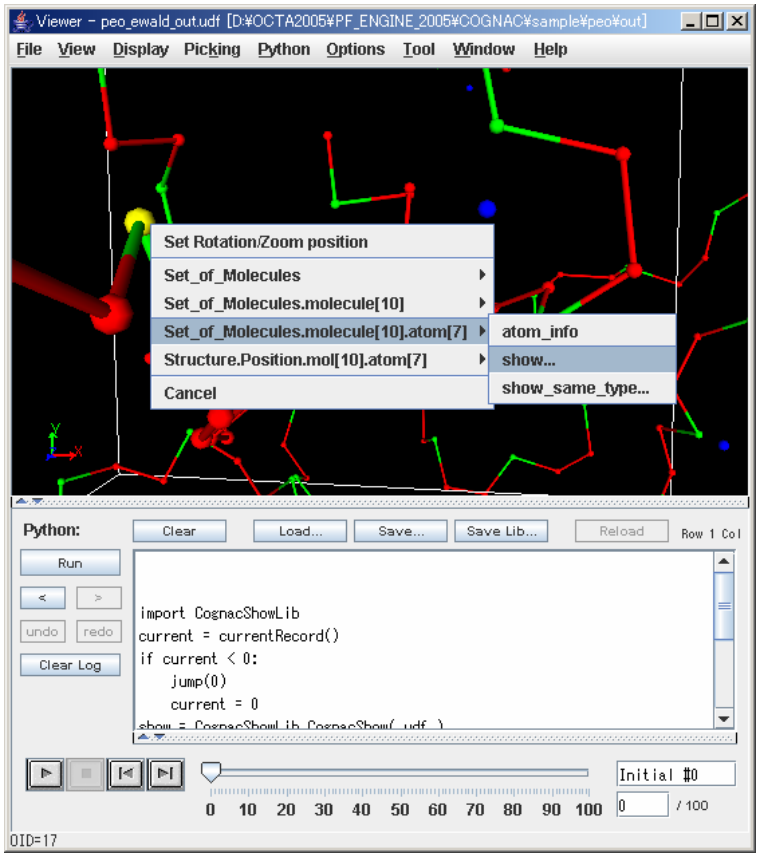


図 31: ビューワにおけるアクション

6.6 Pythonパネルの利用に関するTips

・キー操作[Ctrl+C]/[Ctrl+V] でテキストのコピー・ペーストを行うことができます。DeepCopy モードではクリップボードに Python のリスト形式でデータが保管されているため、これをペーストして Python 変数に簡単に代入できます。

・Python script ウインドウ内でのテキスト編集は、undo ボタンおよび redo ボタンにより、元に戻したり繰り返したりすることができます。ただし、スクリプトの履歴機能によりスクリプト全体を入れ替えると、undo および redo はクリアーされます。

・Python Log ウインドウの Python のエラーメッセージは、エラー行情報を含んでいます。エラーメッセージが出た時、"line"を含む語をダブルクリックすると、Python script ウインドウのスクリプトのエラー行がハイライトします。

・Python script ウインドウと Python Log ウインドウの間のディバイダーによって、各ウインドウの大きさをマウスで変更できます。

・Python 実行の中断

Python スクリプトの実行中は、Python パネルの Run ボタンが Stop ボタンに変わり、この STOP ボタンを押すとスクリプトの実行を強制的に中断することができます。この操作により、アクションによる Python スクリプト実行も中断できます。スクリプトの実行を中断した場合、データ処理の途中で処理が中断しているため、一旦 UDF ファイルを閉じる必要があります。

第7章　エンジンを実行するには

エンジンがGOURMETをサポートしている場合、GOURMETからネットワークを通してエンジンを起動したり、制御することができます。ここで「GOURMETをサポートする」という意味は、最低限UDFファイルを入出力できるということです。もしも、制御（中止や再開など）を行いたければ、そのエンジンはOCTAエンジンのようにプラットフォームインタフェースライブラリを使ってプログラムされている必要があります。

また、エンジンの実行と制御を行いたければエンジンサーバ側でエンジンマネージャを起動しておく必要があります。エンジンサーバ側でエンジンマネージャが起動している場合に、Tool/"Engine Run"や"Engine Control"メニューを使用することができます。

7.1 エンジンマネージャ

GOURMETからエンジンの実行と制御を行う場合、エンジンを実行するマシンでエンジンマネージャを起動しておく必要があります。ローカルマシン（GOURMETを使うマシン）上でエンジンを稼動させる場合、同マシンでエンジンマネージャを起動します。

重要: ローカルマシンでのみエンジンを動かし制御するだけで（他のマシンのエンジンを使わない）、使用しているネットワーク環境が安全でないと思う場合、OS のシステムツールを使って、セキュリティーポリシーの設定をすべきです。詳細は、付録B「エンジンマネージャのセキュリティーポリシーについて」を参照してください。

7.2 Engine Runパネル

Tool/Engine Run メニューから Engine Run パネルを開いて、起動するエンジンに関する条件を設定し、解析を開始することができます。図32 に、Engine Run パネルを示します。
このパネルを用いて以下のような条件を設定することができます。

- **Run name**：エンジン実行名称の設定
- **Server**：ローカルでエンジンを実行する場合は、"localhost"、サーバーでエンジンを実行する場合はサーバーのホスト名かIPアドレスを設定
- **Engine**：エンジン実行スクリプトまたはモジュールの設定
- **Working Dir**：エンジン実行時の作業用Directory の設定

・Params：エンジンに対応する起動時引数の設定

　各エンジンのコマンドライン引数を設定します。

・Input UDF：入力 UDF ファイルの設定

・Params UDF：各エンジンの実行中に変更可能なパラメータファイルの設定

・Restart UDF：リスタートファイルを設定

・Output UDF：出力 UDF ファイルの設定

・Summary UDF：サマリーUDF ファイルの設定

・Logger：任意の実行ステップにおける解析監視スクリプトまたはモジュールを設定

・Always Use... ：

　ON にすると Input UDF には現在 Open している UDF のファイル名とディレクトリ名が常に画面にセットされます。

・Display Engine Control Window... ：

　ON にすると Run 時に Engine Control Window を自動的に表示します。

・Save info & close... ：

　ON にすると Run 時に設定内容を保存し、この Engine Run パネルを自動的に閉じます。

　実行制御の設定内容を複数設定することができます。

・New：エンジン設定項目の新規作成

　ノート：エンジン実行名称がリストに表示されます

・Remove：エンジン設定項目の削除

・Duplicate：エンジン設定項目の複製

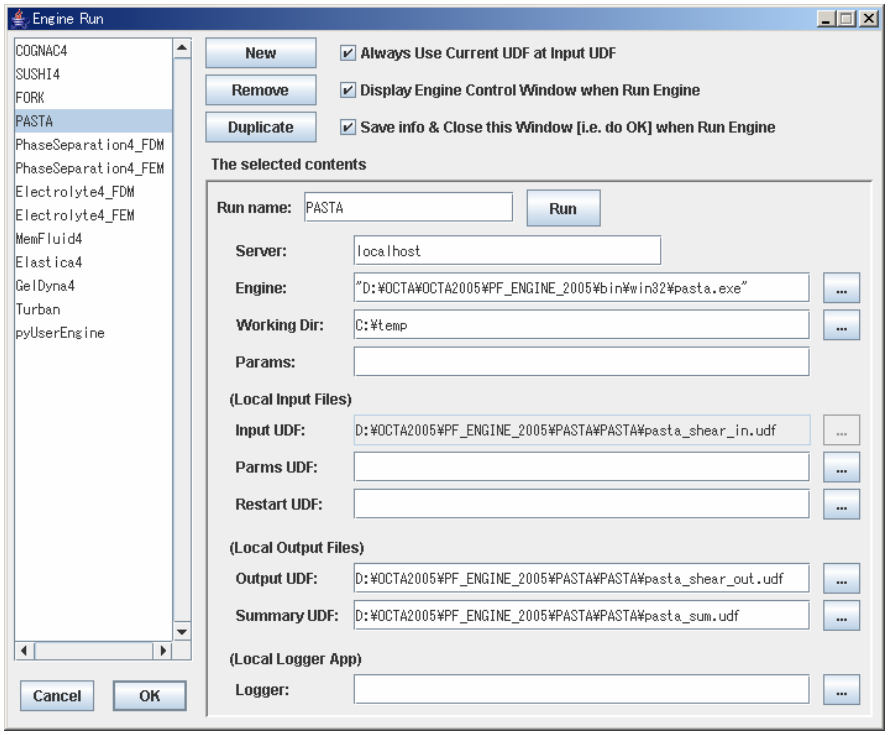


図 32: Engine Run パネル

7.3 Engine Controlパネル

Engine Control パネルを用いて、実行中のエンジンを制御し、結果のサマリーを見ることができます。図 33 に Engine Control パネルを示します。

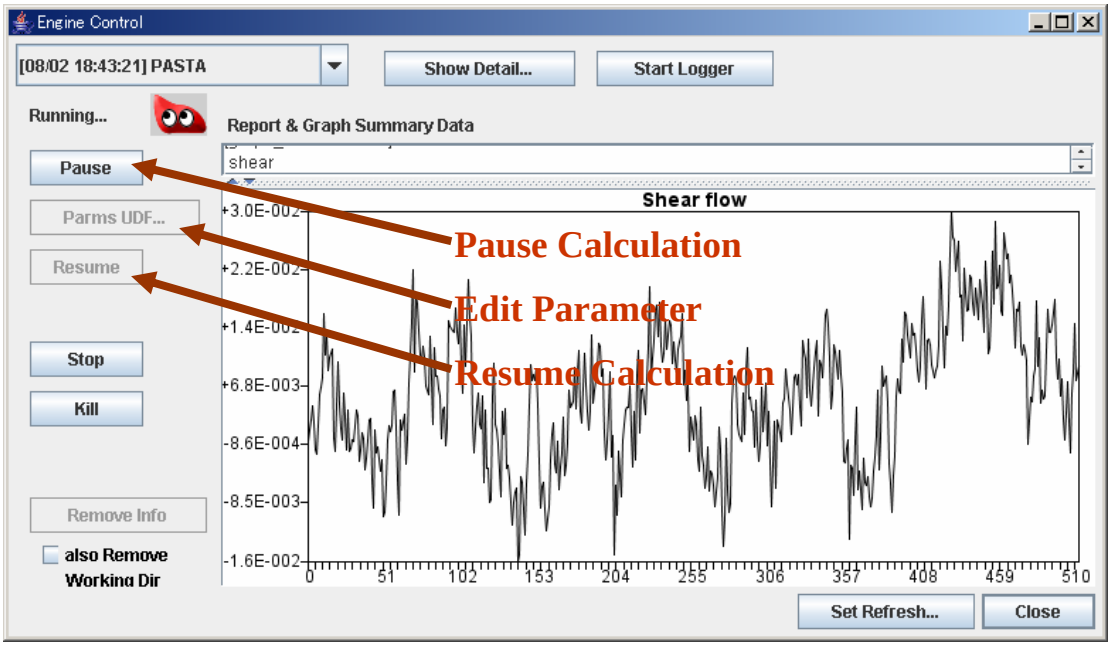


図 33: Engine Control パネル

複数のエンジンプロセスがある場合は画面左上のコンボボックス（[MM/DD HH:MM:SS] Run name の形式表示）より選択して画面を切り替えることができます。

画面内の Report & Graph SummaryData パネルにはサマリーUDF ファイルの内容が一定間隔で簡易表示されます。

- 上部のテキストエリアには UDF データの report_attribute.label[]、report_data.value[] および graph_attribute.label[]のテキスト表示が行われます。
- 下部のグラフエリアには UDF データの graph_attribute.title のタイトル表示および graph_data.item[].value[]のグラフ表示が行われます。

エンジンプロセスの制御は以下のボタンで行います。

- **Pause**：エンジンプロセスの実行を一時停止します。
- **Parms UDF…**：PAUSE 状態のみ使用可能で、パラメータファイルを対象に Editor を起動しま

す。

- **Resume**：パラメータファイルをエンジンサーバに転送して、エンジンプロセスの実行を再開します。
- **Stop**：エンジンプロセスの実行を停止します。
- **Kill**：エンジンプロセスの実行を強制終了します。

  Stop および Kill されたエンジンプロセス情報を破棄するには以下のボタンで行います。

  Remove Info 画面からエンジンプロセス情報を破棄します。このとき"also Remove Working dir"のチェックボックスが ON になっていると、エンジン実行時の作業用 Directory も同時に削除されます。
- **Show Detail...**：Engine Run Window より起動されたエンジンプロセスに関する情報を表示します。
- **Start Logger**：押した時点で、Engine Run パネルで設定した Logger を起動します。
- **Set Refresh**：サマリーファイル内容を Report & Graph SummaryData パネルへ簡易表示する間隔(sec) を設定します。

7.4 エンジン制御に関するTips

・エンジン制御にリコネクトするには

実行時間が非常に長い（例えば 1 日以上）場合、エンジン制御パネルを閉じたり GOURMET を終了してもエンジンは実行しつづけます。次回 GOURMET を起動したとき、Tool/Engine Control... メニューを選択すると、エンジン制御パネルに再接続できます。

・エンジン実行結果をもう一度見るには

保存されているエンジン実行結果をもう一度見るには、Engine Control パネルの左上端のプルダウンメニューから選択します。

第 8 章　3Dオブジェクトを表示するには

8.1 ビューワの起動画面

ビューワは 3D オブジェクトを描画するウィンドウで、Editor 画面の Window メニューで Viewer を選択するか、Editor 画面で描画用スクリプトを実行すれば表示されます。図 34 は、ビューワの起動画面を示しています。

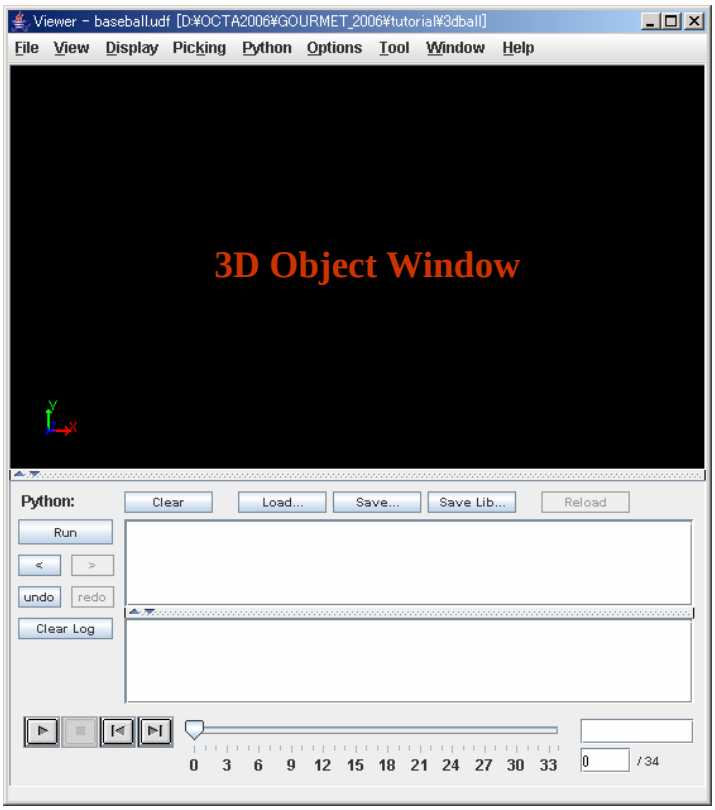


図 34: ビューワの起動画面

ビューワの画面は、メニュー、図形パネルおよび Python パネルから構成されています。図形パネルと Python パネル境界はマウスにより移動することができます。

8.2 3Dオブジェクトウインドウ

3D オブジェクトウインドウ内の以下の操作により、他の 3D アプリケーションと同様な表示変

更を行うことができます。

・回転(マウス左ボタンでドラッグ)
・ズーム(マウス右ボタンでドラッグ)
・ウォークスルー([alt] または[shift] を押しながら、マウス右ボタンでドラッグ)
・ウォークサイド([alt] または[shift] を押しながら、マウス左ボタンでドラッグ)
・視点方向の設定(View メニューで選択)
・背景色、効果、分割数の設定(Display メニューで選択)
・描画イメージのキャプチャ(Options メニューで選択)
・他の描画オプション設定(Options メニューで選択)
・アニメーション(Python パネルのボタン選択)

8.2.1 3Dオブジェクトのピッキング

特別な描画関数(CognacAtom & CognacBond あるいは setDrawRelation & resetDrawRelation)を使って描画され、アクションが準備されている UDF オブジェクトをピックしてアクション実行ポップアップメニューを表示させることができます。

デフォルトアクション(UDF 全体に対するアクション)が定義されている場合、背景を[Ctrl] を押しながら、マウス左ボタンでクリックするとデフォルトアクションメニューが表示されます。

3D オブジェクトが"CognacAtom & CognacBond"関数で描画されており、1 つ以上のアクションが UDF パスに関連付けされている場合、描画対象を[Ctrl]を押しながら、マウス左ボタンでクリックすると関連付けされた UDF のアクションメニューが表示されます。

3D オブジェクトが複数ピッキングの対象であり、1 つ以上のアクションが関連する UDF パスに定義されている場合、Picking/Multi Picking Mode メニューあるいは Ctrl+M で、シングルピッキングモードとマルチピッキングモードを切り替えることができます。

アクションファイルの定義に関する詳細は、付録 C を参照してください。

8.3 ビューワのPythonウインドウ

ビューワの Python ウインドウはエディタと同じ機能を持っています。さらに下部の Start/Stop/Backward/Forward ボタンにより、3D オブジェクトのアニメーションを制御することができます。

8.3.1 3Dオブジェクトのアニメーション

対象となるUDFが複数のRecordを持っている場合は、Record操作ボタン(Start、Stop、Backward、Forward)、あるいは画面下部のスライダーによって、表示するRecordを変更することができます。ビューワ画面ではレコード移動時に、前に実行したPythonスクリプトを再実行します。

・Start:
現Recordから最後のRecordまで連続して前方向に処理する。(アニメーション)

・Stop:
アニメーション処理を中止する。

・Backward:
現Recordから戻り方向に処理する。(ボタンを押している間は処理を続行し、離すと処理を中止する。)

・Forward:
現Recordから進む方向に処理する。(ボタンを押している間は処理を続行し、離すと処理を中止する。)

8.4 ビューワのメニュー

8.4.1 Fileメニュー

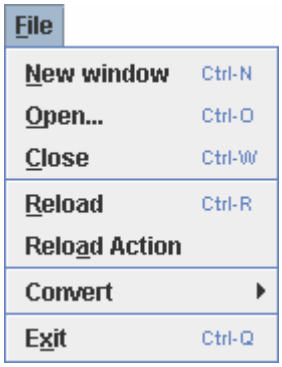


図 35: ビューワの File メニュー

・**Open...**: UDF ファイルを開く。
・**Close**: ビューワを閉じる。
・**Reload**: 開いている UDF ファイルをリロードし、再描画する。
・**Reload Action**: 開いている UDF ファイルに関連したアクションファイルをリロードし、実行する。
・**Convert** ：テキスト形式の UDF ファイルとバイナリ形式の UDF ファイルを変換します。またファイルフィルターを使って外部ファイルをインポートする画面を表示します。
・**Exit**: GOURMET を終了する。

8.4.2 Viewメニュー

View メニューにより、3D オブジェクトウインドウの視点設定を変更できます。

・**Standard**:
3D パネル内の View をすべてのオブジェクトが画面に表示されるような標準的な状態に変更します。Run ボタンから描画スクリプトを実行すると Standard 状態になります。

・**Reset**:
3D パネル内の View を 3D パネル内のマウス操作を行なう前の状態に変更します。
すなわち最後に行われた Standard 状態に戻します。

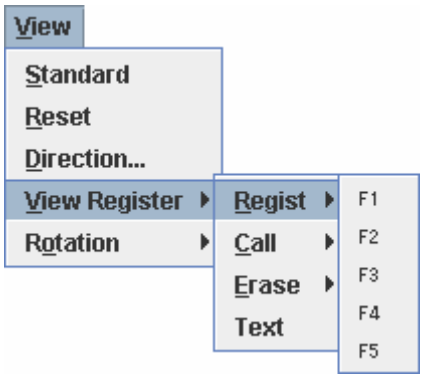


図 36: ビューワの View メニュー

・**Direction**:
3D パネル内の View の視線方向を設定します。標準（Standard）視線方向および現在視線方向を設定することができます。

指定方法は View 設定画面で注視点から視点のベクトルで指定します。
ベクトルの大きさとは関係なく注視点から視点の距離は一定に保たれます。

・View Register:

ビューワでの描画図の見え方を登録（Register）しておき、後で呼び出す（Call）ことができます。また登録内容を消去（Erase）できます。ビューの登録数は、F1～F5 までの５個です。登録および呼び出しは、図のメニューから操作できる他、コントロールキー＋ファンクションキー（１～５）を選択することにより登録し、ファンクションキー（１～５）を押すことにより呼び出すことができます。（注[2]）
また、登録されるとメニューの F1～F5 の文字が太字になり、どのキーにビューが登録されたているかを表示します。登録したビューは Viewer 画面を閉じても記録されています。

View Register の Text を選択して、テキスト画面に現在のビューデータが表示される様子を図 37 に示します。このテキストデータを取っておいて異なる View 画面で利用することができます。

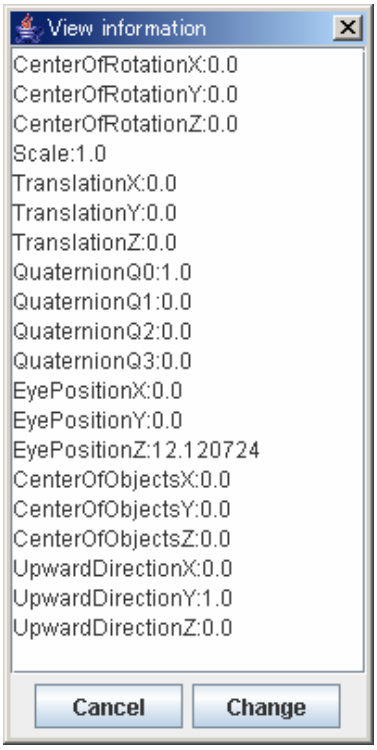


図 37: 現在のビューデータを示すテキスト画面

[2] ファンクションキーを利用するときは、マウスでビューア画面内を一度クリックするなどして、ビューア画面をアクティブにしておく必要があります。Python スクリプト画面がアクティブになっているとビューの登録および呼び出しをキー操作で行うことはできません。

Change ボタンを押すとテキスト画面のビューデータに従って描画図の見え方が変わります。

・Rotation:

Left:左方向に設定角度分回転

Right:右方向に設定角度分回転

Up:上方向に設定角度分回転

Down:下方向に設定角度分回転

Clockwise:時計回りに設定角度分回転

Anticlockwise:反時計回りに設定角度分回転

Angle:キー操作（矢印キーおよび[]キー）または上記メニューによる回転の角度を設定します。

8.4.3 Displayメニュー

3D オブジェクトウインドウの表示方法に関する設定を変更できます。

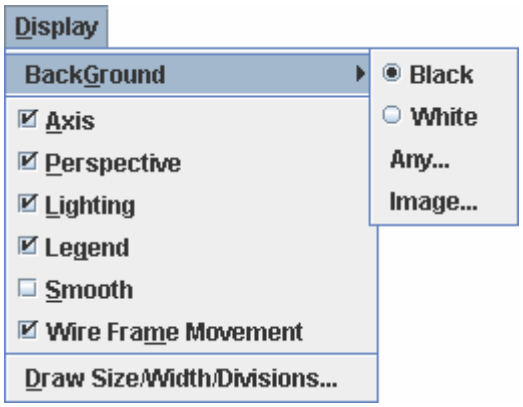


図 38: ビューワの Display メニュー

・BackGround:

Black：図形パネル(3D)の背景を黒色に変更します。

White：図形パネル(3D)の背景を白色に変更します。

Any...：図形パネル(3D)の背景をカラーダイアログで指定した色に変更します。

Image...： 図形パネル(3D)の背景にイメージを貼り付けます。イメージ形式は JPEG または PNG が利用できます。イメージの貼り付け方として以下の方法があります：

Fit max:縦横比を変えずにイメージ全体が画面に入るように貼り付けます。

Fit min:縦横比を変えずに画面全体がイメージで覆われるように貼り付けます。

Raw size:元のイメージサイズで貼り付けます。

Fit canvas:イメージ全体を画面の幅・高さに合わせまて貼り付けます。

・**Axis**:

XYZ 軸の表示の有無を切り替えます。

・**Perspective**:

透視法射影(Perspective projection)あるいは正射影(Orthographic projection)の表示を切り替えます。

・**Wire Frame Movement**:

マウス操作（回転、移動、拡大）中に、操作を早くするために線画で表示するかどうかを切り替えます。

・**Lighting**:

Lighting の効果の有無を設定します。

Lighting の効果を有にすると、立体的に見えます。

Lighting の効果を無にすると、色表示が View の方向に関係なく同じ色に見えます。

ノート: Lighting の効果を無しにすると、SPHERE 等は立体感が無くなります。

・**Smooth**:

画面に表示している線分を滑らかに表示します。

・**Draw Size/Width/Divisions...**:

Point(点)のサイズ、Line(線分)の幅、Sphere（球）、Ellipsoid（楕円体）、Cylinder（円柱）、Cone（円錐）、Disk（円）および Ellipse（楕円）のポリゴン分割数を変更します。

ノート: ポリゴン分割数を増やすと、なめらかに物体が表示されますが、消費メモリーの増加とレスポンスの悪化を招きます。

8.4.4 Pickingメニュー

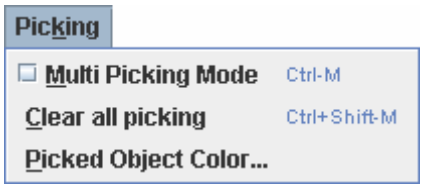


図 39: ビューワの Picking メニュー

・Multi Picking Mode

シングルピッキングモードと複数ピッキングモードの切り替えを行います。

・Clear all picking

ピックされている状態を解除します。

・Picked Object Color...

ピックされたオブジェクトの色をカラーダイアログにから変更します。

8.4.5 Pythonメニュー

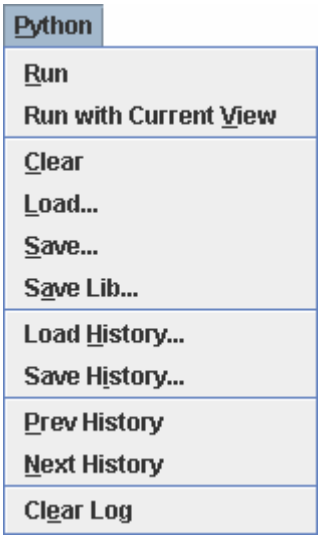


図 40: ビューワの Python メニュー

Python パネルのボタン操作をメニュー形式にしたものです。詳細については 8.3 を参照してください。

8.4.6 Optionsメニュー

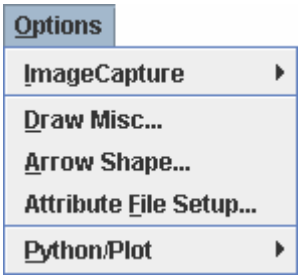


図 41: ビューワの Options メニュー

・ImageCapture:

描画領域のイメージファイル保存（自動保存を含む）についての設定をします。

- Now Capture GraphicPanel

現在の描画領域をイメージファイルに保存します。

- Auto Capture GraphicPanel while animation

Record 操作 animation 中の描画領域の自動保存の ON/OFF を切り替えます。

ノート:デフォルトの保存するディレクトリー・ファイル名は以下の通りです。

(BaseDir)/(UDFFileName)/graphic/(RecordNumber).jpg

例:

ImageCapture/blend_eq_uot.udf/graphic/999.jpg

- Save Option...:以下の設定をします

イメージファイルを保存するディレクトリの設定、イメージファイル形式（JPEG または PNG）の選択および JPEG 形式の場合の Quality(0-100)を設定します。

・Draw Misc...:

描画に関する動作等を設定します。

- Clear Option:

Run 実行時に常に画面の描画を Clear してから描画するか、追加で描画するかを選択します。

- Cache Type for animation:

アニメーション実行時に描画するオブジェクトの取り扱い方法を選択します。

+ USE UDF-Cache

描画スクリプト実行によって生成された描画オブジェクトを描画します。ピッキングからのアクション実行により描画オブジェクトに関連付けられた UDF データを扱う場合は、このモードにしておかないと正しい UDF データ値が得られません。

+ USE Viewer File-Cache

Viewer 内部で各 Record 毎の描画内容を自動的に外部ファイル（File-Cache）に保存しておき、描画スクリプトに変化のないかぎり２回目からの描画に File-Cache を利用して描画の高速化を図ります。

+ USE Viewer Memory-Cache

Viewer 内部で各 Record 毎の描画内容を自動的にメモリ（Memory-Cache）内に保存しておき、描画スクリプトに変化のないかぎり２回目からの描画に Memory-Cache を利用して描画の高速化を図ります。

ノート:Viewer File-Cache および Viewer Memory-Cache が有効なのは以下の操作に限ります。

+ アニメーション操作ボタン(Start, Stop, Prev, Fwd) の操作
+ アニメーションスライダーの操作
+ Record 番号表示テキストエリアの入力
+ Record Label 表示テキストエリアの入力

- Animation Interval[millisecond]:
  アニメーション間の休止時間をミリ秒単位で入力します。
- Animation Slider:
  アニメーションスライダーをマウスでドラッグ操作中に、常にマウス位置に対応するRecordの描画を行うか、最後にマウスボタンを離した時に描画を行うかを選択します。
- Auto Rewind:
  アニメーションを繰り返し実行するかどうかを選択します。

・Arrow Shape...:
矢印の描画形状を設定します。
- User arrow length scale
  描画スクリプト arrow([x1,y1,z1],[x2,y2,z2],[r,g,b,a,h,w,s])で描画された矢印の線分部分の描画長さを調整するためのスケールを設定します。（通常は1.0）
- Mesh arrow length scale
  meshfield によってメッシュ格子点に設定されたベクトル値を描画する場合の矢印の線分部分の描画長さをメッシュサイズの最大値の何倍で描画するかを指定します。(通常は1.0)
- Arrowhead Shape
  矢印の先端部分の形状を Cone(円錐) あるいは複数 Line（線分）にするかを設定します。Line（線分）の時、構成ライン数も設定できます。
- Auto arrowhead size
  "Resize"にチェックを入れると、矢印の最大長さに対する矢じり長さの割合("scale")で矢じりが描画されます。"aspect"は矢じり長さに対する矢じり幅の割合です。

・Attribute File Setup...:
画面に表示する色等の設定ファイルを選択します。この設定ファイルの内容については、Pythonスクリプトマニュアルの描画属性ファイルを参照してください。

・Python/Plot:
Python/Plot スクリプトパネルに表示する Font、History の設定をします。
- Font...
  Python/Plot スクリプトパネルに表示する Font について、Font 名、Style、Size で設

定します。

- **History...**

  Python/Plot 実行スクリプトパネルの内容を記憶する History に関して Window を Open/Close した時に自動的に Load/Save するかを設定します。また、History の記憶最大数の設定も可能です。

第 9 章　プロットを行うには

解析結果の分析のために、gnuplot を使ったプロットを行うことができます。

9.1 Plotツール

Python パネルと Plot パネルは、Python/Plot タブで切り替えることができます。図 42 は Plot タブが選択されたエディタの画面を示しています。

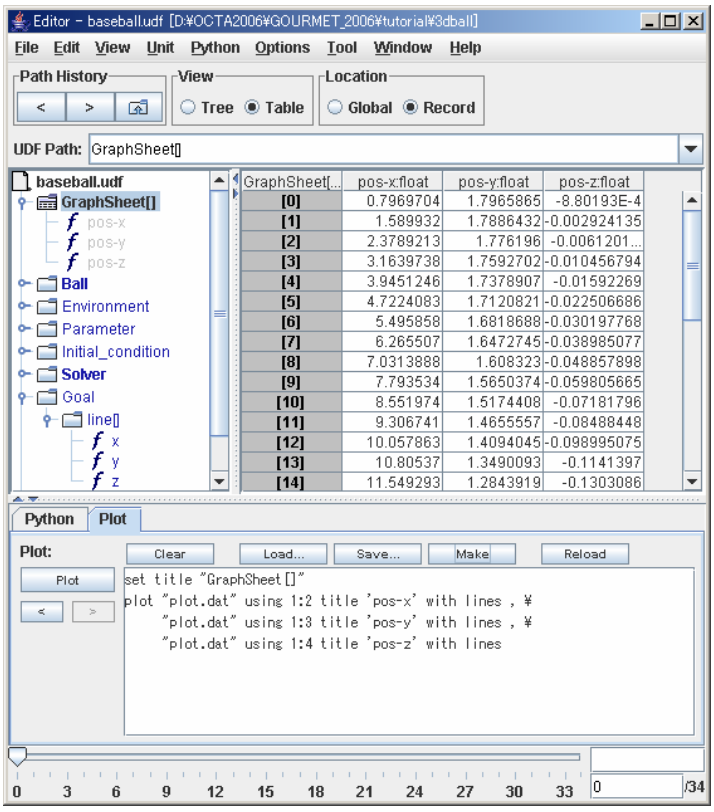


図 42: Plot タブパネル

Plot ツールのグラフ化機能は、データを簡単に可視化することを目的としています。グラフフォーマットを詳細に設定する必要があるならば、Plot ツールで生成されたプロットコマンドを変更するか、gnuplot のインタプリタでパラメータを調整します。

プロットの作成は次の手順で行います。

・プロットしたいデータを Table ビューで表示します。

・Plot タブパネルの Make ボタンを押すと、プロットコマンドとデータが生成されます。この時、プロットデータがカレントディレクトリの plot.dat に生成されます。
・プロットコマンドを編集します。
・Plot ボタンを押すと、プロットコマンドとデータを渡されて gnuplot アプリケーションが起動します。この時、Plot パネルのコマンド内容がカレントディレクトリの cmd.dat に生成されます。図 43 は、gnuplot アプリケーションによるプロット表示例を示しています。

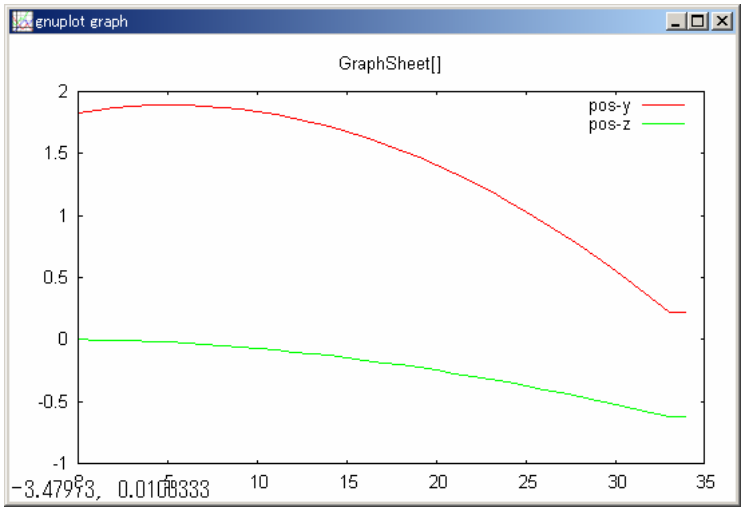


図 43: gnuplot によるプロット表示例

9.2 Graph sheetオブジェクト

GOURMET には、開いている各 UDF 毎に、データの整理領域として GraphSheet というワークシートオブジェクトを準備しています。Table ビューを使用している時に、プロット用データを保管するために GraphSheet オブジェクトを利用することができます。

GraphSheet オブジェクトのデータは通常、Python スクリプトによって生成されますが、他の UDF オブジェクトと同様に直接入力することもできます。

ワークシート作成のために、Python 関数として、行の追加、削除、データカラムの追加が準備されています。

9.2.1 GraphSheet操作を行うPythonスクリプトの例

GraphSheet 操作を行う Python スクリプトの使用例を以下に示します。さらに詳しくは、"GOURMET Python スクリプトマニュアル"を参照してください。

GraphSheet 操作を行う Python スクリプトの使用例:

```
from math import *
createSheetCol(0,'Refference')
createSheetCol(1,'Position')
n = totalRecord()
for i in range(0,n):
  jump(i)
  rsize=getSheetRowSize()
  if rsize <= i:
    insertSheetRow(rsize,1)
    setSheetData(0,i,[-10*sin(pi*i/n)])
    setSheetData(1,i,[get("Structure.Position.mol[0].atom[0].x")])
```

9.3 Plotスクリプティング

GOURMET には、gnuplot.py という gnuplot インタフェースライブラリが準備されており、この Plot ライブラリを、例えばつぎのような Python スクリプトの中で通常の Python モジュールと同様に用いることができます。

・GOURMET アクションファイルの Python スクリプト
・GOURMET の Python スクリプトウインドウ
・GOURMET のエンジン制御におけるロガースクリプト

純粋な Python 環境アクションファイルでの Plot ライブラリ使用例

```
action Calculated_results : xy_plot() : ¥begin
import gnuplot
x = []
y = []
for rec in range(totalRecord()):
  if $Calculated_results.time > $Solver.tmax:
    break
  jump(rec)
  x.append($Calculated_results.position.x)
  y.append($Calculated_results.position.y)
```

```
gnuplot.plot(data=[x,y],labels=['x','y'], title='xy-position')
¥end
```

9.3.1 Plotライブラリの使い方

Python スクリプトから gnuplot 用スクリプトライブラリを使用できます。このライブラリの概要を以下に示します。詳細な仕様については、PF_FILES/python/gnuplot.py を参照してください。

udfdata(udf, datafile='plot.dat', labels, axis='field')

この関数は、開いている UDF (UDFManager のインスタンス) を使って、datafille (プロットデータのファイルパス) を出力します。

labels はプロットする UDF シンボルパスのリストです。axis (field または row) は、プロットするデータ系列を指定します。

rowdata(datafile='plot.dat', datalist, axis='row')

この関数は、指定した datalist を使って、datafile (プロットデータのファイルパス) を出力します。

datalist は、プロットする値のリストです。axis (field または row) は、プロットするデータ系列を指定します。

start(cmdfile='plot.cmd')

この関数は、cmdfile(プロットコマンドのファイルパス)を渡して gnuplot を起動します。

gnuplot(commands, cmdfile='plot.cmd', datafile='plot.dat')

この関数は、cmdfile と datafile を作成した後に gnuplot を起動します。

commands は、gnuplot コマンド行の文字列リストです。cmdfile は、gnuplot に使われるコマンドファイルパスです。datafile は、cmdfile から参照するデータファイルパスです。

plot(data, title='', labels, attrs, udf,cmdfile='plotficmd',
datafile='plot.dat', axis='field')

この関数は、指定したパラメータで cmdfile(プロットコマンドのファイルパス)を作成し、これを渡して gnuplot を起動します。

data は、プロットデータ値のリストです。udf が指定されていれば無視されます。title は、プロットのタイトルです。labels は、プロットラベルもしくは UDF パスです。attrs は、

プロット属性です。
udf は、開いている UDFManager のインスタンスです。cmdfile は、gnuplot に使われるコマンドファイルパスです。datafile は、cmdfile から参照するデータファイルパスです。axis（field または row）は、プロットするデータ系列を指定します。

第 10 章　ツールを使うには

GOURMET には、Tool メニューと File メニューに以下のようなツールが準備されています。

10.1 ファイル転送ツール

ファイル転送ツール（Transmitter）を使って、UDF ファイルを他の UDF サーバ（データマネージャが起動しているマシン）との間で転送することができます。ファイル転送ツールはリモートマシンで起動しているデータマネージャと連携してローカルとリモート間のファイル転送機能と各マシン内での基本的なファイル操作および UDF の Engine Type/Version 等のヘッダ情報の表示を可能にします。ローカルでのみ使用するときは、データマネージャの起動は不要です。

・リモートマシンとの接続初期状態では、左側の LocalHost パネルに表示されるディレクトリは現在 Open してる UDF のディレクトリです。UDF を Open していないときは、GOURMET の実行ディレクトリです。リモートマシンと接続するために Remote Host のテキストエリアに Host 名あるいは IP アドレスを入力して、Connect ボタンを押します。
図 44 は Transmitter パネルの起動時の状態を示し、図 45 はリモートホストと接続した状態を示しています。

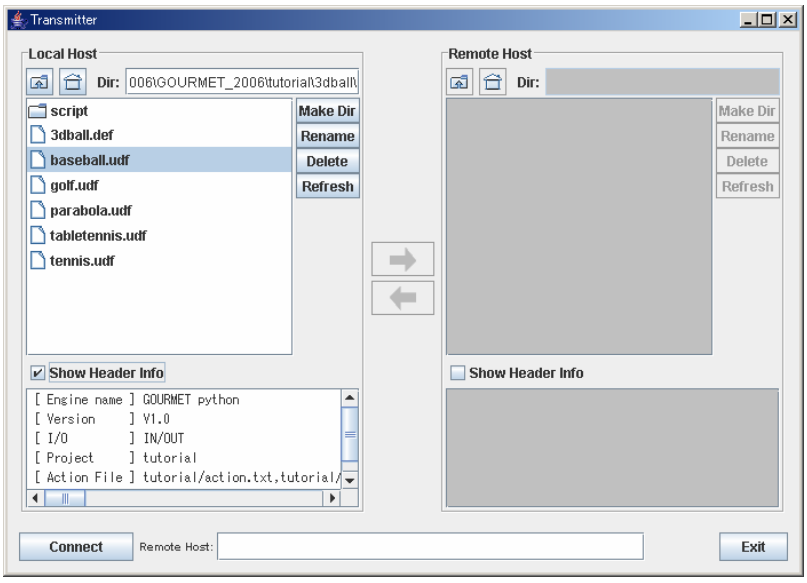


図 44: ファイル転送ツール起動画面

接続後、右側の Remote Host パネルに表示されるディレクトリは、データマネージャの引

数で指定されたディレクトリか起動ディレクトリとなります。但し、Remote Host を Local として Connect した時は、左側の Local Host のディレクトリと同じになります。

・Local と Remote 間のファイル転送機能

Local Host パネルあるいは Remote Host パネル内のファイルを選択して画面中央の矢印ボタン（→: Send,←: Receive）を押下する事によりファイル転送（Send, Receive）が可能です。

・Local、Remote での基本的なファイル操作

Local Host パネルあるいは Remote Host パネル内のファイル操作ボタンを押下する事により基本的なファイル操作が可能です。

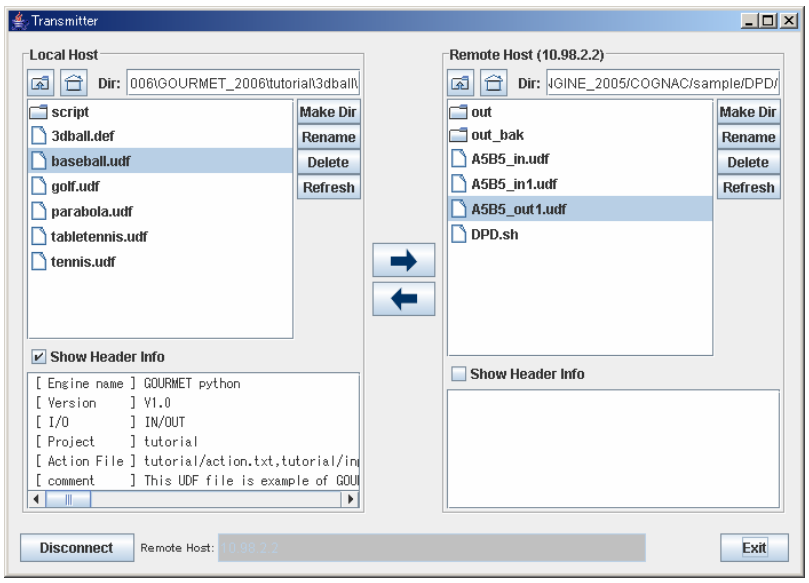


図 45: Transmitter 接続画面

・UDF のヘッダ情報の表示機能

Show Header Info のチェックボックスにより、ファイル選択時の UDF のヘッダ情報の表示が可能です。

10.2 Pythonツール

Tool/Python メニューを使用して、ピュアな（GOURMET 拡張を含まない）Python 環境を起動することができます。起動オプションは、Application Setup ツールで設定します。10.4 を参照

してください。

10.3 Gnuplot ツール

Tool/Gnuplot メニューを使用して、gnuplot アプリケーションを起動することができます。起動オプションは、Application Setup ツールで設定します。10.4 を参照してください。

10.4 Application Setupツール

Python および gnuplot の起動アプリケーションの指定と、起動時の引数の指定を行います。図 46 は Application Setup ツールダイアログを示しています。

重要：
起動時の引数のファイル名にブランクがあれば全体を""でくくる必要があります。
（設定例）
Python Application:"C:¥OCTA2010¥GOURMET_2010¥bin¥win32¥Python¥pythonw.exe"
Python Argument: "C:¥OCTA2010¥GOURMET_2010¥bin¥win32¥Python¥Tools¥idle¥idle.pyw"
Gnuplot Argument: "C:¥OCTA2010¥GOURMET_2010¥bin¥win32¥gnuplot¥wgnuplot.exe"

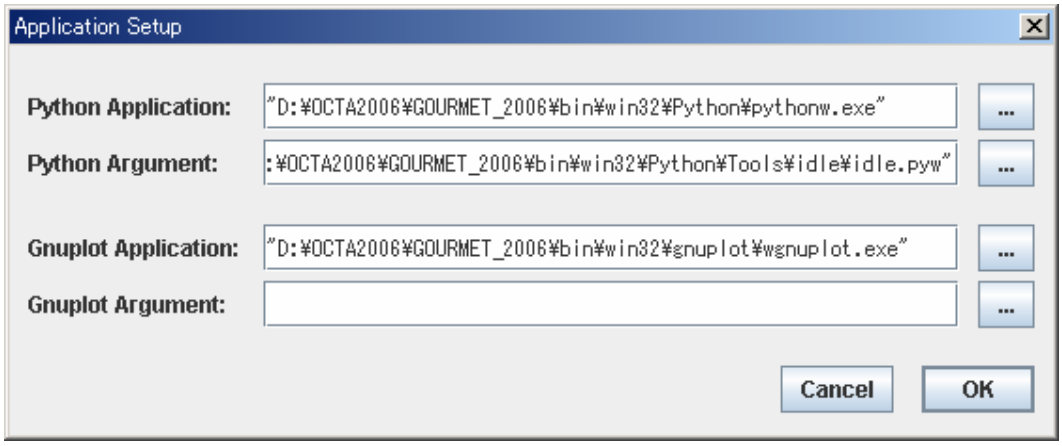


図 46: Application Setup

10.5 分子ビルダーツール

molfile フォーマットおよびPDB ファイルフォーマットのデータを COGNAC などの UDF ファイル

にインポートする機能です。図 47 は分子ビルダーツールダイアログを示しています。

分子ビルダーツールは次の処理を行います。

・molfile か PDB データファイルを、指定されたファイルフィルターで読み込む

・結合角情報、二面角情報を自動生成する

・これらのデータを入力 UDF ファイルの定義に従って変換する

・変換後のデータを入力 UDF ファイルの既存データとマージする

・上記の結果を指定された出力 UDF ファイルに保存する

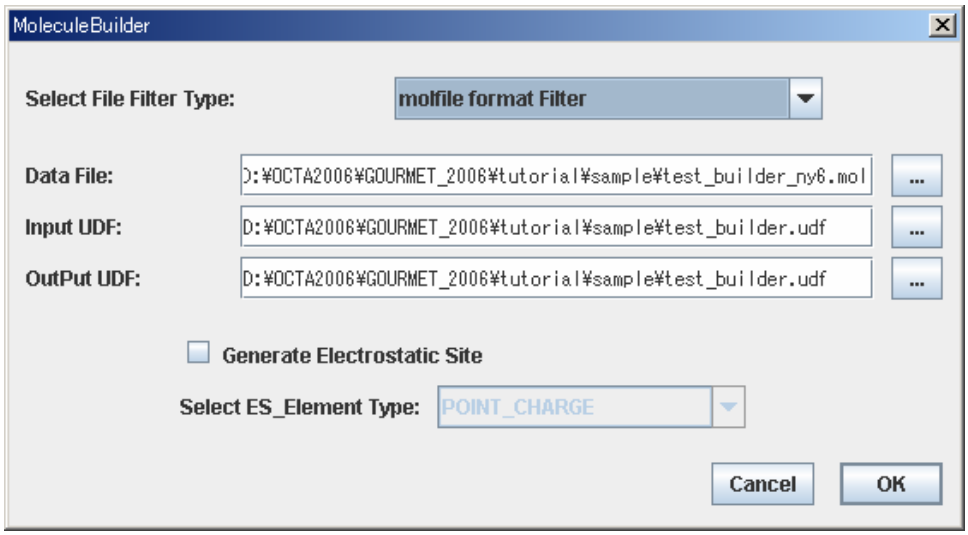


図 47: 分子ビルダー

10.6 起動環境設定ツール

起動環境設定ツールを使って GOURMET を起動するときの環境設定ファイル（platform_win32.ini,platform_linux.ini）の内容を編集することができます。

設定できる環境変数は以下のとおりです：

・Python 本体のインストールディレクトリ（PYTHONHOME）

・Python スクリプトの検索ディレクトリ（PYTHONPATH）

・実行パス（PATH）

・ライブラリパス（Unix/Linux における LD_LIBRARY_PATH）

・エンジン等のグループ単位で以下の設定ができます：

アクションファイルの検索ディレクトリ（UDF_ACTION_PATH）

Python スクリプトの検索ディレクトリ（PYTHONPATH）

インクルード UDF ファイル検索ディレクトリ（UDF_DEF_PATH）
実行パス（PATH）
ライブラリパス（Unix/Linux における LD_LIBRARY_PATH）

グループ単位で環境を設定する場合、グループの先頭ディレクトリ（"Top directory"）を 1 つ指定することにより自動的に以下のディレクトリパスを起動環境に設定することができます：

アクションファイルの検索ディレクトリ（UDF_ACTION_PATH）
"Top directory"+"/action"
Python スクリプトの検索ディレクトリ（PYTHONPATH）
"Top directory"+"/python"
インクルード UDF ファイル検索ディレクトリ（UDF_DEF_PATH）
"Top directory"+"/udf"
実行パス（PATH）
"Top directory"+"/bin"
ライブラリパス（Unix/Linux における LD_LIBRARY_PATH）
"Top directory"+"/lib"

自動設定を使用せずに個別にグループ単位の環境設定をする場合は、"detail"にチェックをつけます。

また、環境変数内の下記キーワードを置き換える機能を持っています：
%ARCH%, ${ARCH}　：アーキテクチャー名置き換え
%PF_FILES%, ${PF_FILES}　：PF_FILES 環境変数に置き換え
%PF_ENGINE%, ${PF_ENGINE}：PF_ENGINE 環境変数に置き換え
%OCTA_DIRECTORY%, ${OCTA_DIRECTORY}：OCTA インストールパスに置き換え

GOURMET 起動時に環境設定ファイルを環境変数生成ツール（gourmet_init）が読み込んで起動に必要な環境変数を生成します。環境変数生成ツールについては付録 G を参照してください。

図 48 および図 49 に起動環境設定ツールの画面例を示します。

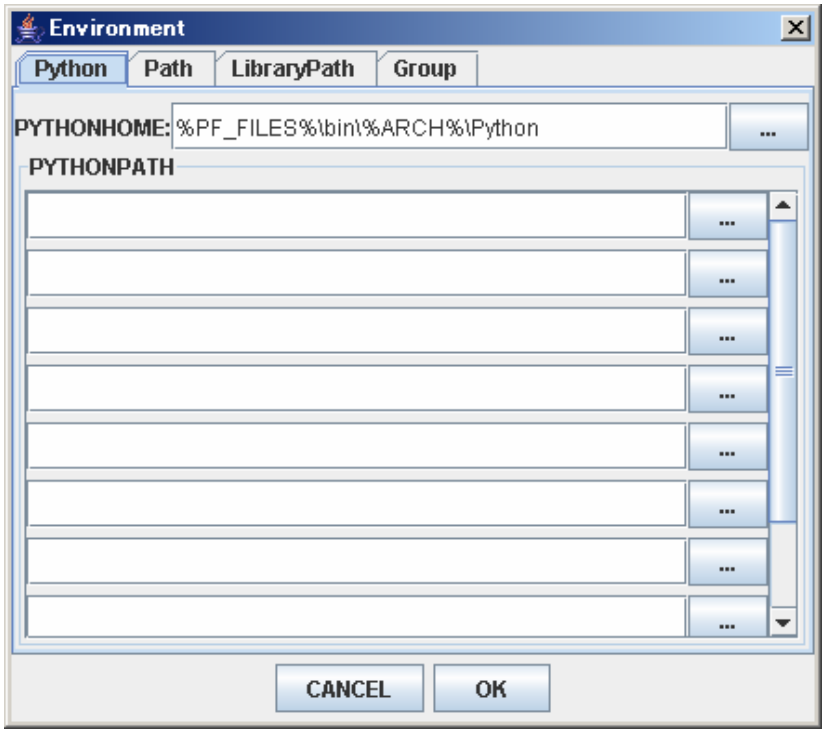


図 48: 起動環境設定ツールの Python 設定タブ

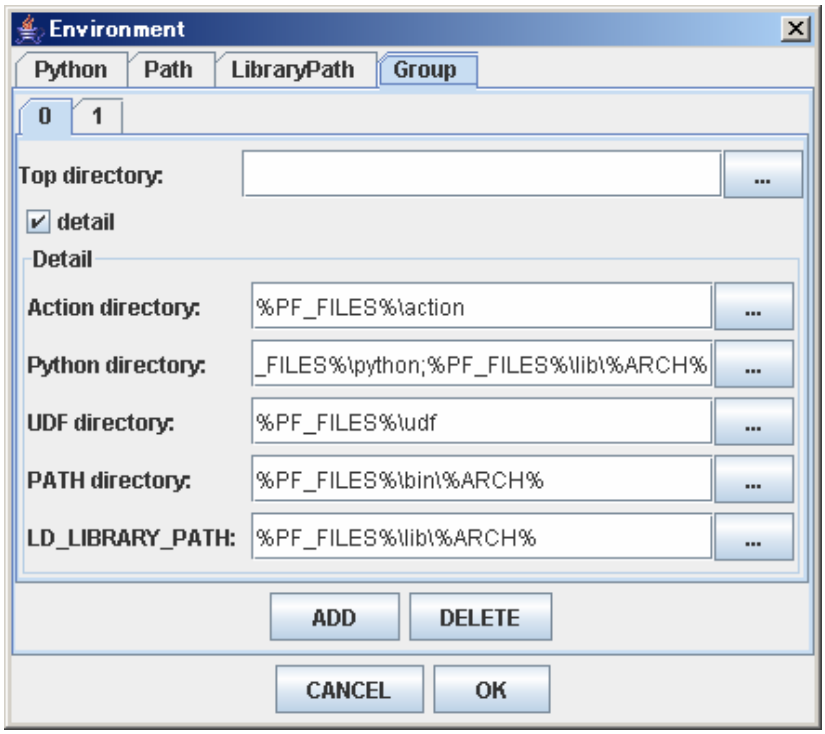


図 49: 起動環境設定ツールの Group 設定タブ

付録A 稼働環境

A.1 Operating System

Operating System	Version	Comments
Microsoft Windows-NT(Intel)	4.0 SP6	動作確認済
Microsoft Windows 2000(Intel)	SP4	動作確認済
Microsoft Windows XP(Intel)	SP1, SP2	動作確認済
Microsoft Windows Vista		動作確認中
RedHat Linux(x86)	9.0J	動作確認済
Red Hat Enterprise Linux (x86)	3	動作確認中
Turbo Linux(x86)	10	動作確認中
Fedora Core	3, 5	動作確認済
Fedora Core x86_64	5	動作確認中(32 ビット動作)
MacOS X Tiger	10.4.7	Intel Core Duo 2.16GHz Power PC G5 Dual 2.5GHz 動作確認済
MacOS X Panther	10.3.9	Power PC G4 1GHz 動作確認済

表 2: Operating System

A.2 Java System

Java System	Version	Comments
SUN Java2 SE/RE	1.4.2	動作確認済
SUN Java2 SE/RE	1.5.0(5.0), 6	動作確認済

表 3: Java System

A.3 Graphics System

Graphics System	Version	Comments
OpenGL	Mesa 1.3以降	Mesa 4.0.4以降
JOGL JSR-231	1.0.0	動作確認済
JOGL JSR-231	1.1.0	Windows 動作確認中 Linux カーネルバージョンに依存

表 4: Graphics System

A.4 Python System

Python System	Version	Comments
Python	1.5.2、2.1.x、2.2.x	動作確認済
Python	2.3.x, 2.4.x	動作確認済
Scientific Python	Pythonに依存	動作確認済
Numerical Python	Pythonに依存	動作確認済

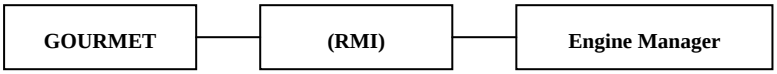
表 5: Python System

Python 2.2.3 : Numerical Python 23.1, Scientific Python 2.4.5

Python 2.4.4 : Numerical Python 23.8.2, Scientific Python 2.4.11

付録B. エンジンマネージャのセキュリティーポリシーについて

GourmetとEngineManagerは、クライアントとサーバの関係にあります。つまり、EngineManagerはGourmetからの要求を待ち続けるサーバとして機能します。



通信経路としては、JavaのRMI(Remote Method Invocation)を使い、RPCライクなサービスとしてエンジンの起動、停止、UDFファイルの転送機能を提供します。ただし、認証機構の実装、Javaからのファイルパーミッション制御については実装していません。

これらのサービスは実行速度を優先して設計されています。そのため、イントラネット内で実行され、かつ、複数ユーザが競合して同じリソース（例：同じサーバの同じディレクトリを同時刻に）を使わないことを想定しています。

そのため、以下のケースでリソースの競合、不具合が発生する可能性があります。

・同一ディレクトリで複数クライアントから実行しようとした場合
・実行中のエンジンの作業ディレクトリ内のファイルを削除、改変した場合
・その他、実行中のエンジンに影響を与える操作をサーバ上で行った場合

GourmetとEngineManagerとの接続単位でユニークなセッションキーを発行し、他に起動しているGourmet から動作中のエンジンを制御されることは防いでます。しかし、ファイルやディレクトリといったリソースにはアクセス可能であるため、上記のようなケースでは不具合が発生する可能性があります。

重要：
・エンジンマネージャはクライアントに対する認証を行いません。
・実行可能なモジュールに対して制限を設けていません。

このようなセキュリティー上の危険を避けるための方策として、以下が考えられます。
・システム管理者はエンジンマネージャを起動するための特別なユーザIDを準備し、必要な最低限の権限を与えておく。
・もしも、OCTAシステムとGOURMET をローカルマシンでしか稼動させないならば、STAND ALONE MANAGERスタートアップオプションを使う。このオプションを使えば、他の誰もそのマシンを

エンジンサーバとしては使えなくなる。

・エンジンマネージャをMictosoft Windowsで稼動させる場合、エンジンマネージャが使うコミニュケーションポートのフィルタを行い、他のマシンからの接続を行えないようにする。これを行うためには、OS 付属のツールかPersonal Firewallなどのセキュリティー確保のための製品を使用することができる。

付録C. アクションの定義

C.1 アクションファイルの文法

アクションファイルの文法規約について以下に記述します。

・コメント

行の最初の文字が'#'ならば、その行全体がコメントとみなされます。

・import

import 文は、Python の import 文と同じです。

・action

action キーワードはアクション定義のターゲットとなる UDF データに対するアクションの始まりに使用します。UDF データターゲットを指定しなければ、UDF ファイル全体に対するアクションとなります。

・autorun

autorun キーワードは UDF ファイルが開かれた時と、リロードされた時に実行されるアクションを定義するために使用します。autorun は複数定義でき、そのすべてが実行されますがその実行順序は規定されていません。

・ターゲット

アクションのターゲットとは、エディタやビューワでマウスによりポイントされる UDF データオブジェクトです。2 つ以上のターゲットが定義された場合、ビューワにおける複数ピッキングアクションとして呼び出され、実行されます。

・アクション名

アクション名はアクションが選択されたときにポップアップにリスト表示されます。

・アクションパラメータ

アクションパラメータはアクションに渡す引数です。アクションパラメータが定義された場合、ユーザはアクションダイアログでパラメータの値を入力できるようになります。パラメータが文字列型であり'|'で区切られている場合、ユーザーはそれらの選択メニューからひとつを選択できます。パラメータが文字列型であり、その値が [...] である場合、ユーザーはダイアログに表示された[...]位置をマウスで右クリックすることにより現れるファイルオープンダイアログでファイルパスを選択できるようになります。

表 6 はアクションファイルの BNF（厳密な文法）を示しています。BoldFace の語は、キーワードで、UPPERCASE の語はトークンです。

構文規則の名前（非終端記号）：構文規則の構成

```
statement:
        comment_statement
        import_statement
        actions
        ¥n
comment_statement:
        # any_statement ¥n
import_statement:
        import python_module ¥n
actions:
        action target : action_statement ¥n
        autorun : action_statement ¥n
target:
        NOTHING
        UDF_OBJECT
        target, UDF_OBJECT
action_statement:
        ACTION_NAME ( parameters ) : body
parameters:
        NOTHING
        parameters , PARAMETER_NAME
        parameters , PARAMETER_NAME = initial_value
initial_value:
        NUMBER
        "STRING"
        "string_selection"
        "filepath_selection"
string_selection:
        NOTHING
        STRING
        string_selection STRING
filepath_selection:
        [...]
body:
        python_function
        ¥begin python_statements ¥end
```

表 6: Action ファイルの BNF 文法

C.2 Python文の特別語

次の語は、アクション本体の Python 文の中で特別な意味を持ちます。

・parameter 名

Python 文の中で、アクションパラメータ名が使われた場合、アクションパーサはその語をパラメータの値（ユーザが入力した値）に置換します。例 1 では、name が"time" かユー

ザの入力した文字列に置換されます。

・self

python 文の中で self が使用された場合、アクションパーサはその語（self）をアクションのターゲット名に置換します。例１では、self が Calculated_results.time に置換されます。

例１：（ターゲット:Calculated_results.time、アクション名:add_trajectory)

```
action Calculated_results.time : add_trajectory(name="time") : ¥begin
try:
    deleteSheetCol(name)
except RuntimeError: pass
createSheetCol(getSheetColSize(),name)
for rec in range(totalRecord()):
    jump(rec)
    setSheetData('time', rec, 'self')
¥end
```

２つ以上のターゲットが定義されている場合、キーワード"self"＋定義されているターゲット数がこの置換プロセスに使用されます。ビューワの複数ピッキングモードにおいて、ユーザが最初に

Set_of_Molecules.molecule[0].atom[1]

をピックし、次に

Set_of_Molecules.molecule[0].atom[5]

をピックした場合、

例２において、

self1 は Set_of_Molecules.molecule[0].atom[1] に置換され、

self2 は Set_of_Molecules.molecule[0].atom[5] に置換されるます。

例２: 複数ピッキングアクションのサンプル

```
action Set_of_Molecules.molecule[].atom[], Set_of_Molecules.molecule[].atom[]
: distance(option="print|draw") : ¥begin
pos1 = get('Structure.Position.mol[].atom[]', Location('self1').getIndex())
pos2 = get('Structure.Position.mol[].atom[]', Location('self2').getIndex())
ds = 0
for i in [0,1,2] : ds = ds + math.pow(pos1[i]-pos2[i], 2)
  if option == "print":
    print 'Distance', $self1, $self2, ':', math.sqrt(ds)
```

```
  else:
    line(pos1,pos2, 0)
    message = 'Distance:%f' % math.sqrt(ds)
    pos = [(pos1[0]+pos2[0])/2,(pos1[1]+pos2[1])/2,(pos1[2]+pos2[2])/2]
    text(pos, message, 0 )
¥end
```

重要：

アクションパーサは、通常の字句解析を用いて、"self[0-9]*"やパラメータ名にマッチする全ての語を置換します。

したがって、これらの語（"self[0-9]*"）をアクション本体で使う場合には、十分に注意してください。

C.3 アクションファイルのサンプル

例: チュートリアルより

```
# autorun
# This action will be excuted when the UDF is opened or reloaded.
autorun : initialize() : ¥begin
import gnuplot
# initialize GraphSheet
trajsize = totalRecord()
colname = ['time']
$GraphSheet[] = []
for i in range(getSheetColSize()):
  deleteSheetCol(i)
i=0
for name in colname:
  createSheetCol(i,name,trajsize)
  i=i+1
¥end
# Non Target Action
# This action named 'clearDraw' will be executed when the user points UDF file icon in
Editor,
```

```
# or points background with [Ctrl] in Viewer.
action : clearDraw() : ¥begin
clearDraw()
jump(0)
¥end
# Typical Action definition
# This Action named ' setColor' will be executed when the user point ' Ball' in Editor.
# Action Dialog requires to select one of the value for the parameter ' BallColor',
# and the word 'BallColor' in the action body replaced to the selected color (say red).
action Ball: setColor(BallColor="white|blue|green|red") : ¥begin
if BallColor == 'white':
  $Ball.color[] = [1,1,1]
elif BallColor == 'blue':
  $Ball.color[] = [0,0,1]
elif BallColor == 'green':
  $Ball.color[] = [0,1,0]
elif BallColor == 'red':
  $Ball.color[] = [1,0,0]
¥end
```

C.4 UDFファイルとの関連付け

アクションファイルはUDFファイルのヘッダー部のAction項目にアクションファイル名が記述されていれば、UDFファイル読み込み後に読み込まれます。

（例）Action項目記述例

```
¥begin{header}
¥begin{def}
Action:string;
Comment:string;
¥end{def}
¥begin{data}
Action:"cognac_draw.act;cognac_info.act;cognac_plot.act;cognac_anal.act;cognac_edit.act"
Comment:"UDF definition file for COGNAC4.2"
¥end{data}
¥end{header}
```

GOURMETがアクションファイルを探索するディレクトリは、まず、UDFファイルと同じディレクトリであり、次にGOURMET起動時に引数として渡されるUDF_ACTION_PATH変数です。

UDF_ACTION_PATH 変数は、規定では以下のディレクトリになります：

FP_FILES 環境変数ディレクトリ+"action"

PF_ENGINE 環境変数ディレクトリ+"action"

付録D 汎用ファイルコンバーター

D.1 ファイルコンバーターとは

ファイルコンバーターは、専用の言語で書かれたフィルタルールを使って外部のソフトウェアで作成されたデータを UDF に変換するツールです。この章では、このフィルタールールの文法について記述しています。

D.2 フィルタールールの文法

・コメント

行の最初の文字が'#'ならば、その行全体がコメントとみなされます。

・事前処理

行の先頭の語が'BEGIN'の時、フィルタリング処理の前に指定した Python 関数が呼ばれます。AWK スクリプトと同様の機能です。

・事後処理

行の先頭の語が'END'の時、フィルタリング処理の後に、指定した Python 関数が呼ばれます。AWK スクリプトと同様の機能です。

・数値

行の先頭の語が数値の時、フィルターは入力行から指定された数値の行数だけをスキップします。

・制御ルール

行の先頭の語が'CONTROL'の時、その行はあるデータを何行読むかを指定する制御用データです。

FORTRAN で書かれた小さなプログラムがこの形式をよく使用します。制御ルールでは変数（文字列）の並びを指定し、制御用データに記述されている数値を、その変数に読み込む処理を行います。

・変数

行の先頭の語が上記の制御ルールで指定された変数と同じである時、フィルターは入力データから変数の値と同じ行数を読み込む。

・ラベル

行の先頭の語が'LABEL'の時、その行はあるラベルを付けられたデータ行を読み込むためのパターンマッチングデータです。SED や perl 等で書かれた多くの UNIX プログラムや CAD の

pre/post プロセッサがこのデータ形式を使っています。２番目の語がマッチングパターンであり、SED で使われる正規表現を用いています。

・フィールド幅

丸括弧（ ）でくくられた数値は、データフォーマットの読み込みカラム数として使用されます。数値が省略されるかまたは 0 を指定した場合、フィルターはホワイトスペースで区切られたフリーフォーマットとして読み込みを行います。

表 7 はフィルタールールファイルの BNF（厳密な文法）を示しています。BoldFace の語は、キーワードであり、UPPERCASE の語はトークンです。特別な変数（_NDATA）が、その時点で読み込まれているデータ数として使われます。

```
line:
        comment_line
        begin_proc
        skip_line
        control_line
        fixed_line
        maching_line
        end_proc
        ¥n
comment_line:
        # any statement ¥n
begin_proc:
        BEGIN python_func_name ¥n
end_proc:
        END python_func_name ¥n
skip_line: NUMBER ¥n
control_line:
        CONTROL controls ¥n
controls:
        NOTHING
        controls VARIABLE_NAME field_width
field_width:
        NOTHING
        ( )
        ( NUMBER )
```

```
fixed_line:
        VARIABLE_NAME format_rules ¥n
format_rules:
        NOTHING
        format_rules UDF_PATH field_width
        format_rules UDF_PATH = value
        format_rules UDF_PATH = python_func_name
matching_line: LABEL matching_pattern format_rules ¥n
value:
        NUMBER
        "STRING"
        _NDATA
```

表 7: フィルタールールファイルの BNF 文法

D.3 ファイルフィルターの例

molfile ファイルフィルターと PDF ファイルフィルターが分子ビルダーツールで使用されますが、これらは変換元データを、以下の UDF ファイルのデータへ変換します。
以下の例で、(連続行) は次の行へ続いていることを示しています。(実際のデータではない)

```
¥begin{def}
class Vector3d:{x:float, y:float, z:float}
class Atom:{
  Atom_ID:int
  Atom_Type_Name:string
  Position:Vector3d
  Mol_ID:int
}
class Bond:{
  atom1:int
  atom2:int
}
atoms[]:Atom
bonds[]:Bond
```

¥end{def}

例１： molfile フィルター

molfile フィルターは以下のルールにより、molfile 形式のデータを前述の UDF ファイルへ読み込みます。

```
#*
#*  Material Modeling Platform Package
#*
#*  Copyright(c) 2000, The Japan Research Institute, Ltd.
#*  All rights reserved.
#*
#*  $Id: molfilerule.txt,v 1.1.1.1 2004/11/29 12:19:10 nishio Exp $
#*
#*
# example rule for convert molecule data from molfile
3
CONTROL nAtom(3) nConnect(3)
nAtom    atoms[].Position.x(0) atoms[].Position.y() atoms[].Position.z()  (連続行)
                    atoms[].Atom_Type_Name() atoms[].Atom_ID=_NDATA atoms[].Mol_ID=-1
nConnect bonds[].atom1(3) bonds[].atom2(3)
LABEL    ^M..CHG(6)
LABEL    ^M..ISO(6)
END      =END_PROC
```

例２： PDB ファイルフィルター

PDB ファイルフィルターは以下のルールにより、PDB ファイル形式のデータを前述の UDF ファイルへ読み込みます。

```
#*
#* Material Modeling Platform Package
#*
#* Copyright(c) 2000, The Japan Research Institute, Ltd.
#* All rights reserved.
#*
#* $Id: filter.tex,v 1.1 2002/02/21 12:56:12 nishio Exp $
#*
#*
# example rule for convert molecule data from pdb file
LABEL ^ATOM..(6) atoms[].Atom_ID(5) (1) atoms[].Atom_Type_Name(4) (6) (4)  (連続行)
(4) atoms[].Position.x(8) atoms[].Position.y(8) atoms[].Position.z(8) atoms[].Mol_ID=-1
LABEL ^HETATM(6) atoms[].Atom_ID(5) (1) atoms[].Atom_Type_Name(4) (6) (4)  (連続行)
(4) atoms[].Position.x(8) atoms[].Position.y(8) atoms[].Position.z(8) atoms[].Mol_ID=-1
LABEL ^CONECT(6) =PDB_conect
END =END_PROC
```

例３： NASTRAN bulk ファイルフィルター

NASTRAN bulk ファイルフィルターが MUFFIN で使われるますが、このフィルタは、NASTRAN bulk

ファイル形式のメッシュデータをプラットフォームの共通 UDF のメッシュデータに変換するサンプルです。（GRID、CTRIA3、CTETRA ラベルのみに対応する例）

```
#*
#* Material Modeling Platform Package
#*
#* Copyright(c) 2000, The Japan Research Institute, Ltd.
#* All rights reserved.
#*
#* $Id: filter.tex,v 1.1 2002/02/21 12:56:12 nishio Exp $
#*
#*
# example rule for convert mesh data from NASTRAN bulk file
LABEL GRID(8) mesh.data.vertex[].id(8) (8) mesh.data.vertex[].position.x(8)  (連続行)
mesh.data.vertex[].position.y(8) mesh.data.vertex[].position.z(8)
LABEL CTRIA3(8) mesh.data.face[].id(8) mesh.partial_region[0].face[](8)  (連続行)
v0(8) v1(8) v2(8) mesh.data.face[].vertex[]=[v0,v1,v2]
LABEL CTETRA(8) mesh.data.cell[].id(8) mesh.partial_region[0].cell[](8)  (連続行)
v0(8) v1(8) v2(8) v3(8) mesh.data.cell[].vertex[]=[v0,v1,v2,v3]
```

付録E　専用コンバーターツール

E.1 NASTRANデータコンバーターツール

NASTRANバルクデータファイルを有限要素UDFファイルに読み込み、描画および表面抽出の処理を行うユーティリティ群です。有限要素UDFファイル（fem_def.udf）、アクションファイルおよびPythonスクリプトファイルは以下のディレクトリに収められています：

GOURMET_20XX/tool/NASTRAN

使用方法

（１）起動方法

GOURMET を起動して、有限要素 UDF 定義ファイル（fem_def.udf）を読み込みます。データを保存する場合は、必ず別名保存をして UDF 定義ファイルを変更しないように注意してください。

（２）NASTRAN バルクファイルのコンバート読み込み

編集画面の UDF ファイル名部分を右クリックしてアクションリストを表示し、
[ReadBulk]を選択します。

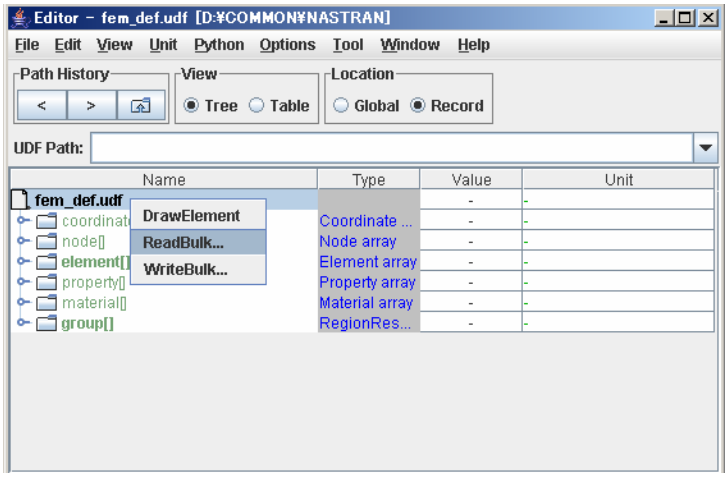


図 50: NASTRAN バルクファイル読み込みアクション選択

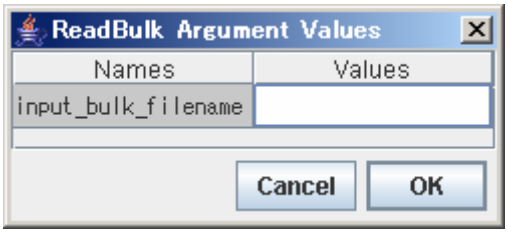


図 51: NASTRAN バルクファイル指定画面

Values の下の入力領域を右クリックするとファイル選択ダイアログが表示されますので、読み込みたい NASTRAN バルクファイルを指定して、「OK」ボタンを押します。すでに NASTRAN バルクファイルを読み込んだ状態でさらに読み込み処理を行うと、以前のデータは全て削除されます。

（3）要素の描画

要素を直接描画するためには、編集画面の"element[]"部分を右クリックしてアクションリストを表示し、[DrawElement]を選択します。

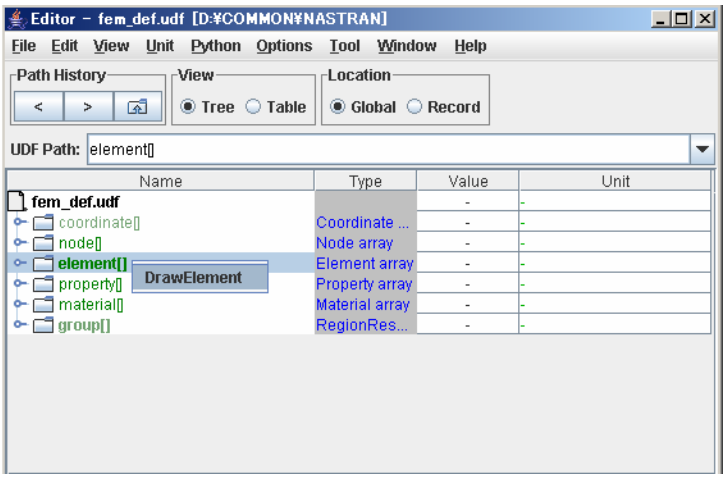


図 52: 要素の描画アクション選択

（4）部分領域の抽出

編集画面の"group[]"部分を右クリックしてアクションリストを表示し、[CreateRegion]を選択します。

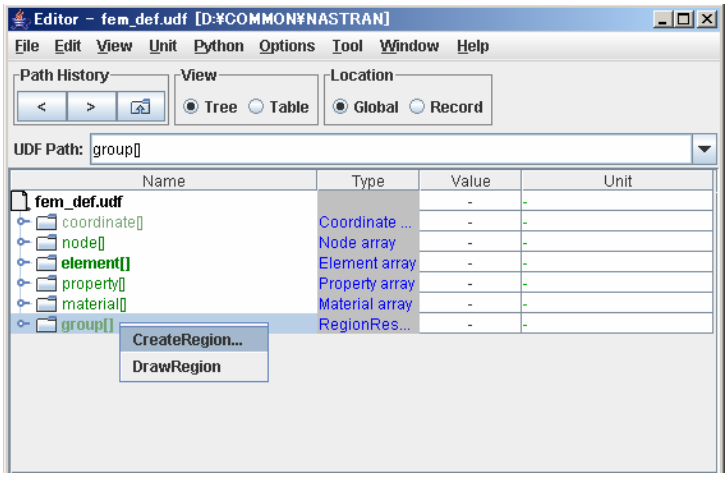


図 53: 部分領域の抽出アクション選択

部分領域を抽出するための条件を"method"から選びます。"angle","property"および"angle_property"から選択することができます。"property"は、要素プロパティの値のみを使って領域分割を行います。"angle_property"の場合は、法線ベクトル角度条件と要素プロパティ両方の条件で領域分割します。法線ベクトル角度条件のみで領域分割する"angle"の場合は法線ベクトル角度（°）を入力して OK ボタンを押します。

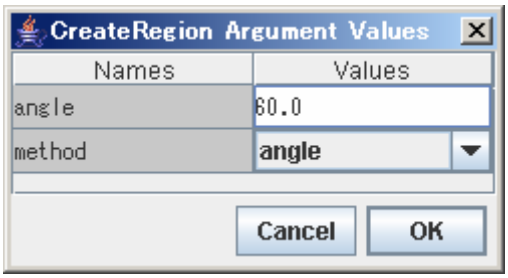


図 54: 部分領域の抽出条件指定画面

編集画面で"group[]"データ以下を展開すると、"group[0]"が作成され、
"group[0].region[]"以下に部分領域が作成されていることが確認できます。

さらに編集画面の"group[]"部分を右クリックしてアクションリストを表示し、[CreateRegion]を選択して部分領域抽出処理を行うと、"group[1]"が作成されて部分領域抽出データが追加されます。

また、例えばインデックス付の"group[0]"部分を右クリックして部分領域抽出処理を行うと、

″group[0]″に今回行った部分領域抽出データが書き直されます。

２次元要素に対する部分領域抽出処理では、境界となる辺の集合を閉多角形となるように出力します。

（５）部分領域の描画

編集画面の″group[0]″部分を右クリックしてアクションリストを表示し、[DrawRegion]を選択します。

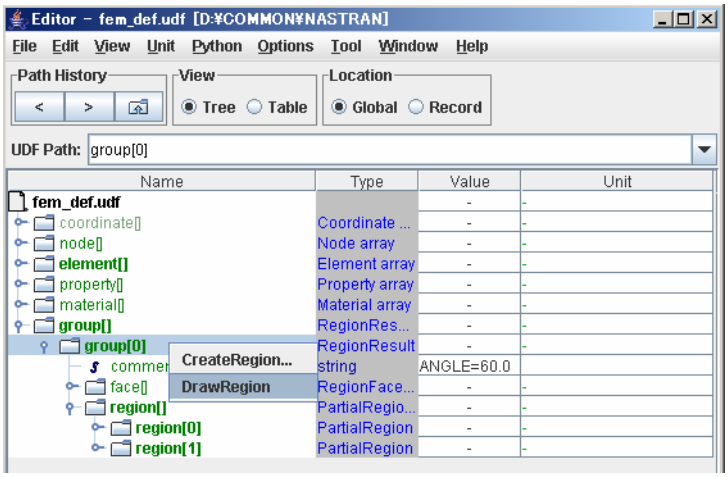


図 55: 部分領域の描画アクション選択

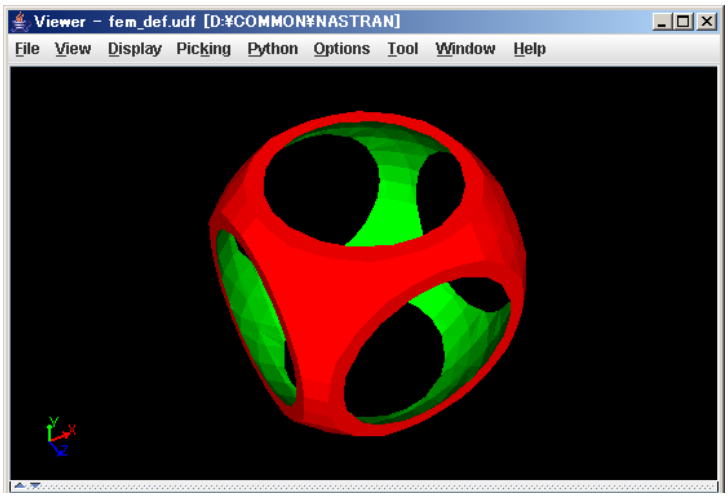


図 56: 部分領域の描画結果（全体）

″group[1]″などを選択して描画すると、その領域分割全体が描画されます。
″group[1].region[5]″など″region[N]″を選択して描画すると、選択した部分領域のみが面で描画され、残りは線で描画されます。

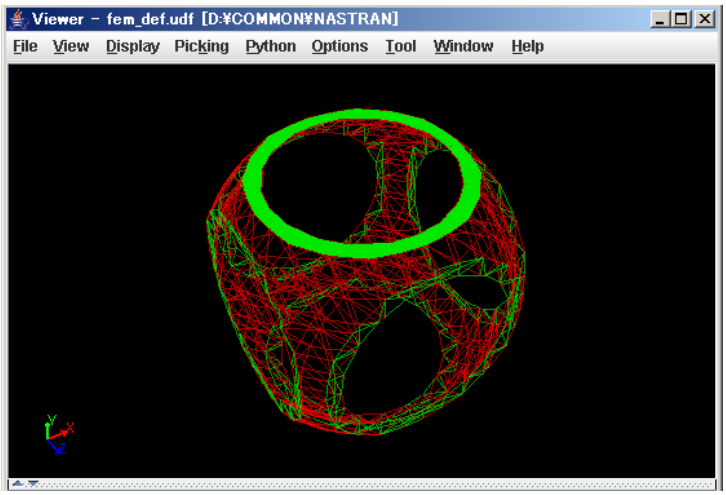


図 57: 部分領域の描画結果（部分選択）

（６）NASTRAN バルクファイルのコンバート書き出し

編集画面の UDF ファイル名部分を右クリックしてアクションリストを表示して、[WriteBulk]を選択します。

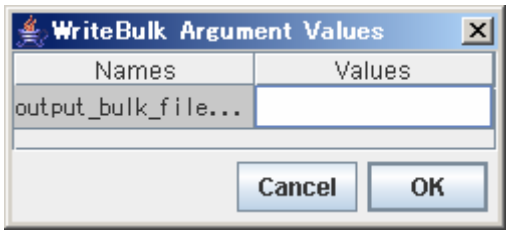


図 58: NASTRAN バルクファイルの書き出し画面

Values の下の入力領域を右クリックするとファイル選択ダイアログが表示されますので、ファイル選択ダイアログで書き出すディレクトリに移動してファイル名を入力してからファイル選択ダイアログを終了し、このダイアログで「OK」ボタンを押します。

UDF解説

トップレベルのデータ構成および内容は以下のとおりです。

coordinate[]:Coordinate	座標系データ
node[]:Node	節点データ
element[]:Element	要素データ
property[]:Property	要素プロパティデータ
material[]:Material	プロパティ物性データ
group[]:RegionResult	部分領域抽出処理データ

各下位データ構成は以下のとおりです。

座標系データ構成

```
class Coordinate:{
 id:ID
 system:{
  type:select{"NodeCartesian","NodeCylindrical","NodeSpherical",
              "PointCartesian","PointCylindrical","PointSpherical"}
  NodeCartesian:{                "節点 ID による直交座標系"
   node1_id:<Node,ID>
   node2_id:<Node,ID>
   node3_id:<Node,ID>
  } "CORD1R"
  NodeCylindrical:{              "節点 ID による円筒座標系"
   node1_id:<Node,ID>
   node2_id:<Node,ID>
   node3_id:<Node,ID>
  } "CORD1C"
  NodeSpherical:{                "節点 ID による球面座標系"
   node1_id:<Node,ID>
   node2_id:<Node,ID>
   node3_id:<Node,ID>
  } "CORD1S"
  PointCartesian:{               "点座標を直接与えることによる直交座標系"
   coordinate_id:int
   a:{ x1:double, x2:double, x3:double }
   b:{ x1:double, x2:double, x3:double }
   c:{ x1:double, x2:double, x3:double }
  } "CORD2R"
  PointCylindrical:{             "点座標を直接与えることによる円筒座標系"
   coordinate_id:int
   a:{ x1:double, x2:double, x3:double }
   b:{ x1:double, x2:double, x3:double }
   c:{ x1:double, x2:double, x3:double }
  } "CORD2C"
  PointSpherical:{               "点座標を直接与えることによる球面座標系"
   coordinate_id:int
   a:{ x1:double, x2:double, x3:double }
   b:{ x1:double, x2:double, x3:double }
   c:{ x1:double, x2:double, x3:double }
  } "CORD2S"
```

```
 }
}
```

座標系タイプの意味は以下のとおり：

"NodeCartesian"：NASTRAN の CORD1R レコードに対応し、３つの節点 ID により直交座標系を指定する。

"NodeCylindrical"：NASTRAN の CORD1C レコードに対応し、３つの節点 ID により円筒座標系を指定する。

"NodeSpherical"：NASTRAN の CORD1S レコードに対応し、３つの節点 ID により球面座標系を指定する。

"PointCartesian"：NASTRAN の CORD2R レコードに対応し、３つの点座標を直接与えることにより直交座標系を指定する。与えた点座標に対する座標系を指定することができる。

"PointCylindrical"：NASTRAN の CORD2C レコードに対応し、３つの点座標を直接与えることにより円筒座標系を指定する。与えた点座標に対する座標系を指定することができる。

"PointSpherical"：NASTRAN の CORD2S レコードに対応し、３つの点座標を直接与えることにより球面座標系を指定する。与えた点座標に対する座標系を指定することができる。

ノードデータ構成

```
class Node:{
 id:ID                                    "節点 ID"
 coordinate_id:int "<Coordinate,ID>"      "座標系 ID"
 position:{                               "節点座標"
  x1:double
  x2:double
  x3:double
 }
}
```

要素データ構成

```
class Element:{
  type:select{"tri","quad","tetra","penta","hexa"}      "要素タイプ"
  id:ID                                                 "要素 ID"
  pid:<Property,ID>                                     "プロパティ ID"
  node[]:<Node,ID>                                      "節点 ID の配列"
}
```

【解説】

要素のノード指定順序は NASTRAN に準拠して、下図のようになります：

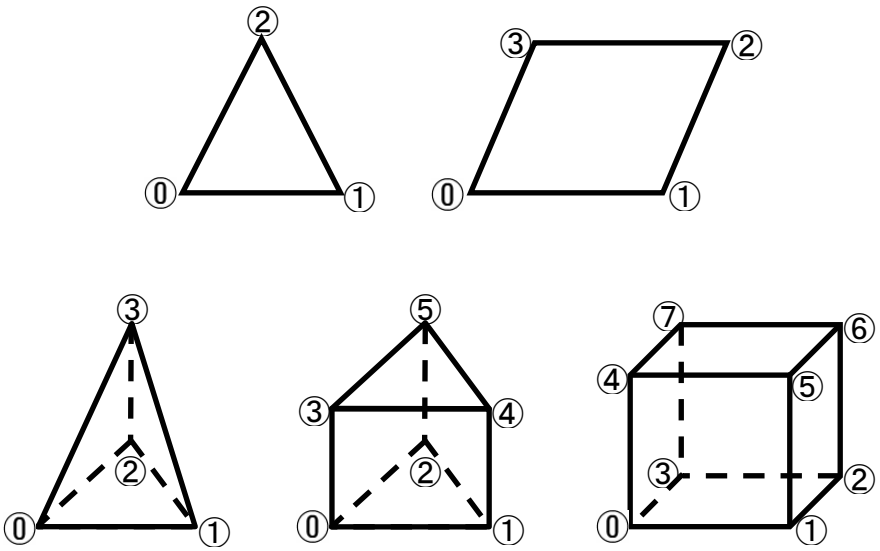


図 59: 要素のノード指定順序

プロパティーデータ構成

```
class Property:{
  id:ID
  mid:<Material,ID>
  dimension:int "2:shell,3:solid"
  values[]:string
}
```

マテリアルデータ構成

```
class MaterialProperty:{
  name:string
  value:double
}
class Material:{
  id:ID
  E:double "Young modulus"
  G:double "Shear momulus"
  NU:double "Poisson ratio"
  RHO:double "Mass devsity"
  A:double "Thermal expation coefficient"
  Other[]:MaterialProperty
}
```

部分領域抽出処理データ構成

```
class RegionSideElement:{
  element_id:<Element,ID>          "面または辺の元要素 ID"
  side_id:int                      "面または辺の元要素に対する位置 ID【解説】参照"
}
class RegionFace:{
  id:ID                            "面または辺の ID"
  node_id[]:<Node,ID>              "面または辺をなす節点 ID の配列"
  element[]:RegionSideElement      "元要素に関するデータ"
}
class PartialRegion:{
  color_id:int                     "部分領域を最小限の色数で塗り分ける色 ID"
  face_id[]:<RegionFace,ID>        "部分領域をなす面または辺の ID の配列"
}
class RegionResult:{
  comment:string
  face[]:RegionFace                "面または辺のデータ配列"
  region[]:PartialRegion           "部分領域のデータ配列"
}
```

【解説】
color_id は全体を最小の色数で塗り分けるための色インデックスです。
side_id は要素の辺または側面を指定するためのインデックスで、要素形状に応じてノードが下記のようになっている部分要素です。
例えば、四面体要素では side_id=0 は、要素内のノード順序{1,2,3}よりなる三角形です。

・三角形要素（サイドは辺）
side_id：[0]={1,2}, [1]={2,0}, [2]={1,2}

・四角形要素（サイドは辺）
side_id：[0]={0,1}, [1]={1,2}, [2]={2,3}, [3]={3,0}

・四面体要素（サイドは三角形面）
side_id：[0]={1,2,3}, [1]={0,3,2}, [2]={0,1,3}, [3]={0,2,1}

・五面体要素（サイドは三角形面または四角形面）
side_id：[0]={0,2,1}, [1]={3,4,5}, [2]={0,1,4,3}, [3]={1,2,5,4}, [4]={0,3,5,2}

・六面体要素（サイドは四角形面）
[0]={ 0,4,7,3 }, [1]={ 1,2,6,5 }, [2]={ 0,1,5,4 },
[3]={ 2,3,7,6 }, [4]={ 0,3,2,1 }, [5]={ 4,5,6,7 }

付録F　描画対象の３次元移動（平行移動・回転移動）機能

３次元描画画面の特定の描画対象（Ver.4.0 から点・線・球・円柱）について、直接描画座標を変換する機能が追加されました。以下にこの機能について説明します。

F.1 準備するもの

３次元移動（平行移動・回転移動）機能を実現するために準備するものは 2 種類のアクションおよび UDF データと関連付けられた描画対象です。

2 種類のアクションとは以下の２つです：
（１）セレクトアクション（$Select_$Translate(), $Select_$Rotate()）
　描画対象に関連付けられた UDF データ名から描画対象をセレクト状態にするアクション
（２）リターンアクション（$Translate(), $Rotate()）
　移動後の座標を UDF データに戻すアクション

上記２つのアクションサンプルが、GOURMET_20XX/action/cognac/cognac_transform.act にあります。サンプルには１アトムの移動（$Translate_ATOM()）、１モレキュールの移動（$Translate_MOL()）および回転（$Rotate()）があります。
COGNAC エンジンの描画アクションファイルにこのアクション文を挿入すれば、次に説明する３次元移動（平行移動・回転移動）機能が操作できます。COGNAC の UDF ファイルを用いて移動アクションを実行する時､境界条件を"NONE"にしておかないと正しい移動座標値が得られないことがあります。

F.2 操作手順

移動操作を行う前に、COGNAC エンジンの描画アクションファイル（cognac_draw.act）の最後に３次元移動サンプルアクション（cognac_transform.act）全体をコピーしておきます。

平行移動

COGNAC エンジンの UDF データを用いて平行移動操作手順を説明します。

（１）COGNAC エンジンの UDF データを開き、"show"アクションを選択して"ball-stick"で描画します。
（２）移動対象となる分子内の１つのアトムを選択します。
（３）アトム選択時のアクション選択メニューで"$Translate_MOL"を選択します。（図 60）
（４）上記「セレクトアクション」が実行され、分子全体が選択された状態になり、平行移動操作ダイアログが表示されます。（図 61）
（５）平行移動操作ダイアログには、２種類の画面があります。１つは"Direction"タブ画面で、画面内の"Up"/"Down"/"Left"/"Right"のボタンを押すと、それぞれ現在見えているビューの上・下・左・右方向に指定長さ（"Length"）分だけ平行移動します。
もう一つは、"Vector"タブ画面で、移動ベクトル量を入力する画面です。
また、平行移動操作ダイアログが表示されている間、コントロールキー＋マウス左ボタンを押したまま描画画面をドラッグすると、選択対象がマウスと同じ方向に移動します。
移動量は描画画面上での１ピクセル移動量が、ボタンによる移動量の１０分の１となります。
（６）"Direction"タブ画面では、各方向ボタンを押すとその分だけ平行移動します。"Vector"タブ画面では Apply ボタンを押すと入力ベクトル量分平行移動します。
（７）OK ボタンを押すと最後の描画状態で平行移動操作が確定し、移動後の座標を UDF データに戻す「リターンアクション」が実行されます。

また、移動操作中でも通常の視点の回転・移動・拡大・縮小を行えます。

回転移動

COGNAC エンジンの UDF データを用いて回転移動操作手順を説明します。
（１）COGNAC エンジンの UDF データを開き、"ball-stick"で描画します。
（２）描画されたビューでマルチピッキングモードにします。
（３）回転の軸となる２つのアトムを選択します。
（４）２つ目のアトム選択時のアクション選択メニューで"$Rotate"を選択します。（図 62）
（５）上記「セレクトアクション」が実行され、２つ目のアトム側のアトムおよびボンドが選択された状態になり、回転操作ダイアログが表示されます。（図 63）
（６）回転操作ダイアログには、回転の中心座標と回転軸ベクトルが示されます。回転の中心座標は、１つ目のアトム座標、回転軸ベクトルは１つ目のアトムから２つめのアトムへのベクトルです。（図 64）
（７）回転の中心座標、回転軸ベクトルおよび回転量（度）を編集することができます。
（８）回転量(度)を入力して回転矢印方向ボタンを押すと回転後の座標で図形が描画されます。

（9）OK ボタンを押すと最後の描画状態で回転操作が確定し、上記「リターンアクション」が実行されます。

回転操作ダイアログで Cancel ボタンを押すと元の座標に戻ります。

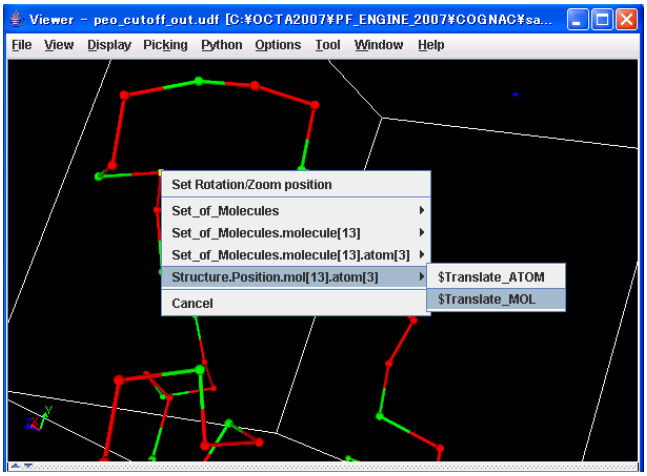


図 60: 平行移動アクション選択

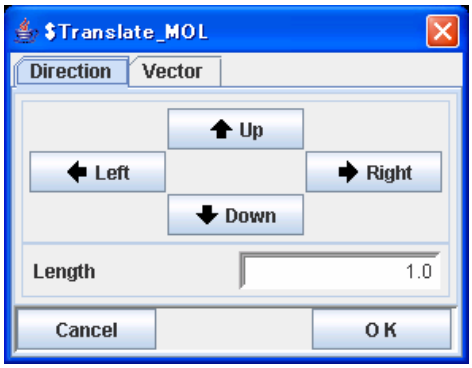


図 61: 平行移動操作ダイアログ

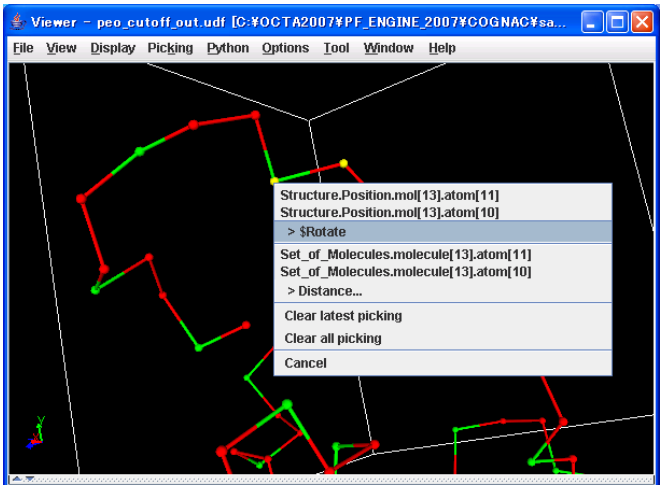


図 62: 回転アクション選択

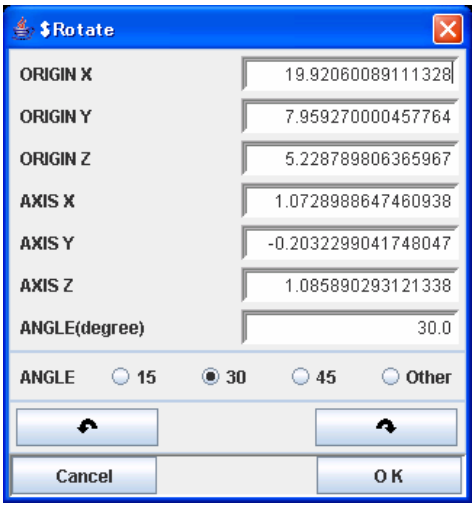


図 63: 回転操作ダイアログ

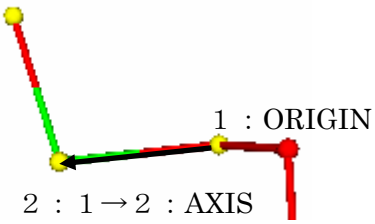


図 64: 回転中心と回転軸ベクトル

付録G　環境変数生成ツール

環境変数生成ツール（gourmet_init）は、C/C++で作成された実行モジュールであり、%PF_FILES%/bin/%ARCH%に置きます。起動シェル（バッチ）ファイルから起動されると、PF_FILES 環境変数が示すディレクトリにある環境設定ファイル（platform_win32.ini または platform_linux.ini)を読み込んでオプション引数に応じた環境変数を生成して起動シェルに返します。

環境変数生成ツールのオプション引数は以下のとおりです：

arch　：マシンアーキテクチャー名を返します。
　　（win32 あるいは、"uname -s"コマンドが返す小文字名）
python.home　：Python ホームディレクトリを返します。
python.path　：Python パスを連結して返します。
action.path　：アクションパスを連結して返します。
udf.path　：UDF パスを連結して返します。
java.home　：Java ホームディレクトリを返します。
java.java　：Java 実行モジュールパスを返します。
user.path　：実行パスを連結して返します。
library.path　：ライブラリパスを連結して返します。
-a：マシンアーキテクチャー名をセットします。
-f：環境設定ファイル名をセットします。（"platform_win32.ini"など）
-j：Java ホームディレクトリまたは Java ホームディレクトリの検索パスをセットします。
　　例えば、C:¥OCTA20XX¥GOURMET_20XX¥bin¥win32¥jre1.5 を渡すと実際に存在する C:¥OCTA20XX¥GOURMET_20XX¥bin¥win32¥jre1.5.0_08 を返します。
-u：パスを渡すとディレクトリ部分を返します。
オプション引数なし：上記の-オプション以外の戻り値一覧を表示します。

環境変数生成ツールを引数なしで実行した場合の出力例を示します：
gourmet_init -f platform_win32.ini

```
arch=win32
python.home=C:¥OCTA20XX¥GOURMET_20XX¥bin¥win32¥Python
python.path=C:¥OCTA20XX¥GOURMET_20XX¥python;C:¥OCTA20XX¥GOURMET_20XX¥lib¥win32;C:¥O
CTA20XX¥PF_ENGINE_20XX¥python;C:¥OCTA20XX¥PF_ENGINE_20XX¥lib¥win32
action.path=C:¥OCTA20XX¥GOURMET_20XX¥action;C:¥OCTA20XX¥PF_ENGINE_20XX¥actionudf.pa
```

```
th=C:¥OCTA20XX¥PF_ENGINE_20XX¥udf
java.home=C:¥OCTA20XX¥GOURMET_20XX¥bin¥win32¥jre1.5.0_08
java.java=C:¥OCTA20XX¥GOURMET_20XX¥bin¥win32¥jre1.5.0_08¥bin¥java.exe
user.path=C:¥OCTA20XX¥GOURMET_20XX¥bin¥win32;C:¥OCTA20XX¥PF_ENGINE_20XX¥bin¥win32
library.path=C:¥OCTA20XX¥GOURMET_20XX¥lib¥win32;C:¥OCTA20XX¥PF_ENGINE_20XX¥lib¥win3
2
```

なお、環境変数生成ツールが返すパスは実際にディレクトリが存在するパスしか返しません。
また、環境設定ファイル内の下記キーワードを置き換えて返すことができます：

%ARCH%,${ARCH}　　：アーキテクチャー名
　　(Windows の場合は win32、Unix/Linux の場合は"uname -s"コマンドが返す小文字名)
%PF_FILES%,${PF_FILES}　：PF_FILES 環境変数 (GOURMET のインストールディレクトリパス)
%PF_ENGINE%,${PF_ENGINE}：PF_ENGINE 環境変数(エンジン群のインストールディレクトリパス)
%OCTA_HOME%,${OCTA_HOME}：OCTA_XXXX_HOME 環境変数
　　(OCTA インストールパスディレクトリパス)

付録H トラブルシューティング

H.1 Pythonスクリプティング

・Python 環境を起動できないのですが...

- インストールが正常に行われていれば、Python パッケージがインストールされ、そのパスが Tool/Application Setup ダイアログに表示されています。これが表示されていなかったり、インストール後に GOURMET の位置を移動して Python パッケージのパスが異なっているなどの場合、手動で正しいパスを設定してください。

H.2 3Dオブジェクトの描画

・3D オブジェクトを描画できません。または、描画しようとすると画面がおかしくなります。

- 比較的新しいマシンの場合は、ディスプレイドライバを更新してみてください。
- Java をご自身でインストールされた場合、JOGL ライブラリがインストールされていません。JOGL JSR-231 (1.0.0)をダウンロードしてインストールしてください。
- インストールマニュアルのトラブルシューティングを参照してください。

・Linux x86_64 上で GOURMET は起動しますが、描画できません。

- GOURMET は x86_64 上でも 32 ビットモードで動作します。
  x86_64 Linux には 32 ビットモード用のライブラリが用意されてるため GOURMET は起動します。しかし 32 ビット用の GL ライブラリは用意されていない場合があるため描画できないことがあります。
  具体的には、/usr/lib64/libGL.so.1 があっても、/usr/lib/libGL.so.1 は無い場合があります。この場合は、32 ビット用の GL ライブラリ (GL および GLU) をインストールしてください。

H.3 プロットの作成

・gnuplot アプリケーションが起動しないのですが。

- インストールが正常に行われていれば、gnuplot パッケージがインストールされ、そのパスが Tool/Application Setup ダイアログに表示されています。これが表示されていなか

ったり、インストール後にGOURMETの位置を移動してgnuplotパスが異なっているなどの場合、手動で正しいパスを設定してください。

H.4 データの編集

・Linuxを使っていますが、GOURMETと他のアプリケーションの間でコピーペーストできません。

- Java1.4からサポートされるようになり、この症状は解決されました。

・Ver.4.0からテーブル編集画面でセルを複数選択できなくなりました。

- Javaのバージョンアップにより選択方法が変わりました。セルを複数選択するときは、一つ目のセルを選択してから、二つ目のセルをシフトキーを押しながら選択します。すると一つ目のセルと二つ目のセルの間の矩形領域にある複数セルが選択状態になります。