OCTA

Integrated simulation system for soft materials

GRAPHICAL USER INTERFACE

GOURMET

OPERATION'S MANUAL

Version 4.1.0

OCTA User's Group

Aug. 08 2007

Authors of the Manual

Chapter 1	Masao Doi, Tokyo University
Rest of all chapters	Yuzo Nishio
Maintenance	Masahiro Nishimoto

Program Developers

Software Design	Shinji Shibano, Yuzo Nishio
Software Development	Masahiro Nishimoto, Astuko Shono, Hiroshi Hashimoto, Yoshiharu Inui, Ikuhisa Masui, Jyunichiro Hiejima, Yukihiro Okuno
Testing	Masahiro Nishimoto, Eisuke Nishitani, Yuzo Nishio

Acknowledgment

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

This work is also partially supported by CREST-JST (Japan Science and Technology Agency) from 2003FY.

Contents

List of Figures

List of Tables

Chapter 1

What is GOURMET

GOURMET (Graphical Open User interface foR Multiscale analysis EnvironmenT) was developed in the OCTA project as a common user interface for various simulation engines developed in the project. All simulation engines in OCTA can be most conveniently used by GOURMET. Using GOURMET, you can

(1) create your input file for the engines,

(2) kick the engines,

(3) see the output file of engines in various forms (numbers, graphs and 3D animations),

(4) analyze and process the output file by your own programs.

In some engines, you can control the running of the engines: you can see the engine status, stop and resume the engine, change the parameters permitted by the engines at proper moments while the engine is running.

GOURMET has a special interface to the script language Python. GOURMET owes greatly to this very elegant language Python. Python allows you to customize GOURMET as you like. By writing python programs, you can add your own function to GOURMET such as preparing the input file automatically, showing the output file in 3D graphics by selecting certain data items, and showing the result of the analysis in a gnuplot graph.

The input file of GOURMET is a text file written in a certain format called UDF (User Definable Format). UDF consists of two parts: the first part defines the data structure and the unit of data, and the second part is the data itself. In the UDF file, all data (numbers and texts) in the file have a unique name, and can be accessed by their name. Thus UDF file can be regarded as a data storage where any data can be accessed and modified in any sequence. This function is powered by Python: any data and data set in UDF can be used like variables in Python.

Naming the numbers is also important for making the data file understandable for others. GOURMET has other functions to make your program user friendly such as (1) defining the unit for each data, (2) putting comments to help users of your program, (3) defining actions for any data set. You will see how you can achieve this by reading this manual. For more detail, see the other manual.

The purpose of GOURMET is to facilitate the collaboration of various people who write or use simulation programs. Many programs are made in many laboratories in the world, but they are rarely written in the form usable for others since making the program usable for others takes enormous amount of time. GOURMET solves a part of the problem, namely to offer an interface (the other part, is of course, to make the program correct and robust, a more difficult part). We hope that GOURMET helps home-made

programs understandable for others, and facilitate the collaboration of many programs in the world.

2001. Nov. 30 Masao Doi

Chapter 2

Getting started

2.1 Start GOURMET

Followings are the command to start Gourmet.

Windows: Go to OCTA2007 of Start menu => StartGOURMET.

Linux: Execute "OCTA2007/GOURMET_2007/gourmet" in shell window.

For more details of startup options, see Chapter 3.

The first screen is Editor. (Figure 1)

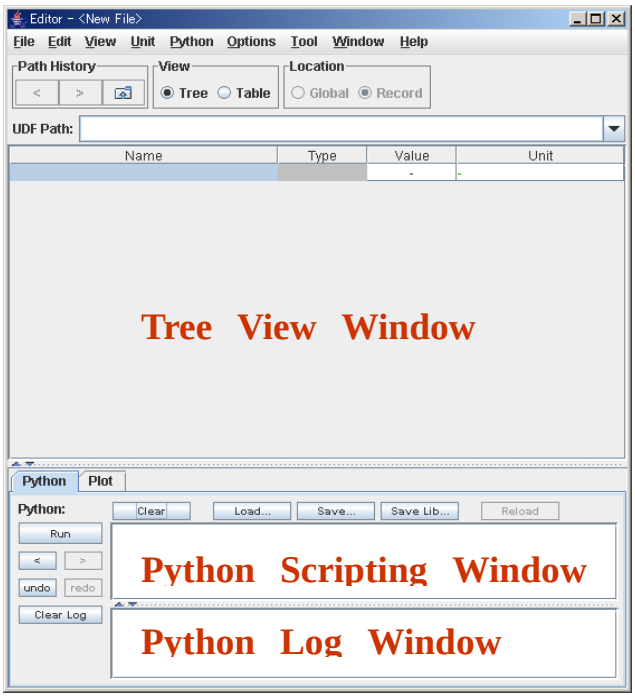


Figure 1: Startup Screen of GOURMET

In Tree View Window, you can see and edit data value.

In Python Scripting Window, you can write variety of programs in Python, such as generating input data, getting computistical output data, converting output data, etc.

In Python Log Window, you get a result of the python script you execute.

2.2 Edit UDF

Click on File/Open... to open a UDF file.

Let's open a tutorial sample "GOURMET_200X/tutorial/3dball/baseball.udf", which simulates the behaviour of a ball that is thrown from the ground.

Click "Ball" on Table View Window, and the attributes (value and unit) of "Ball" is displayed. You can copy/paste on another window or application by selecting the value. See Chapter 4 for more details.

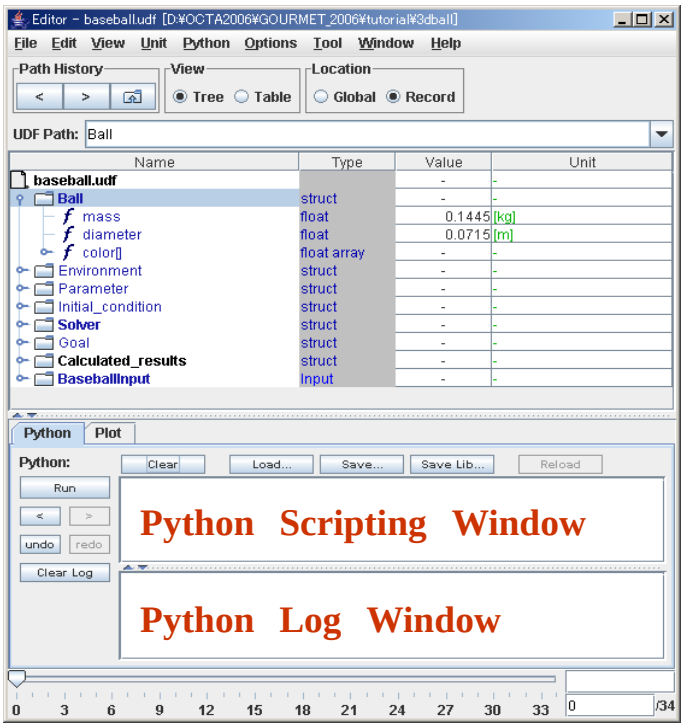


Figure 2: Tree View of Editor

Next, choose the radio button "Table" in View group under Unit menu. Now the View style is changed to Data Structure Window and Table Window.

Data Structure Window shows the hierarchical data structure.

Table View Window shows value table, where you can edit each value.

In Data Structure Window, click "Goal" then "line[]". These symbols are the data names in a UDF file, and called UDF path. What you are looking at is the value of "Goal.line[]" in Table View Window. You can

copy/paste on another window or application by selecting the value.

For more details, see Chapter 4.

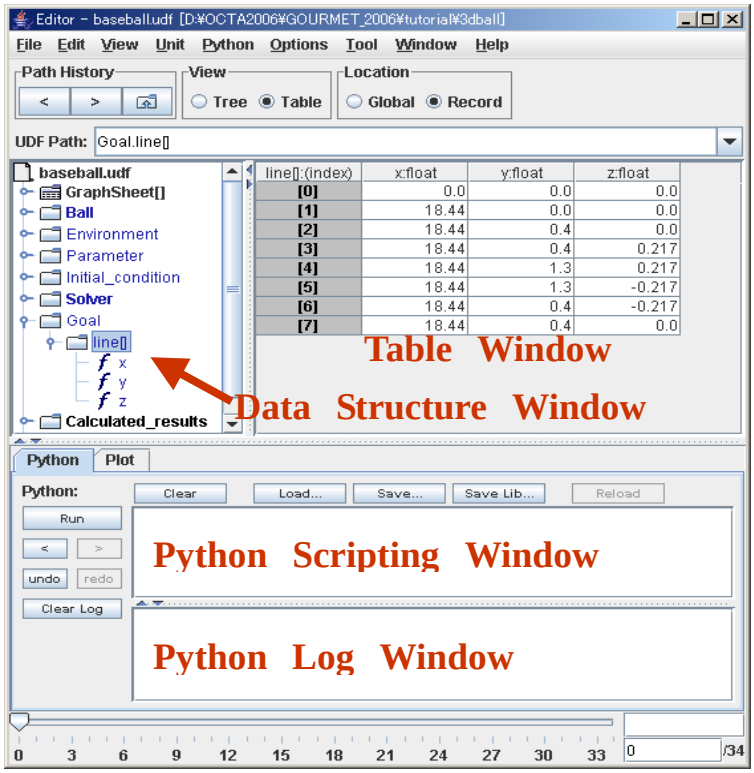


Figure 3: Table view of Editor

2.3 Run Python

At the current UDF file, let's run a command in the python script. Type the following text in Python Scripting Window, and click Run.

print $Ball.mass

You will get the following result in Python Log Window.

0.1445

This is the "mass" value of the "Ball". Use Python script to access UDF data. For more details, see

Chapter 6.

2.4 Run Engine

To control an engine on network, it needs to be able to communicate with GOURMET. Start Engine Manager on the engine server as follows.

Windows: Choose OCTA200X => StartEngineManager from Start menu.

Linux: Execute "OCTA200X/GOURMET_200X/eng_man" in shell window.

Now Tool/"Engine Run" menu and “Engine Control" menu are ready.

Select Tool/"Engine Run" from the menus of the Engine Run panel.

Figure 4 is Engine Run panel. Here the parameters to activate an engine are set up, and calculation starts.

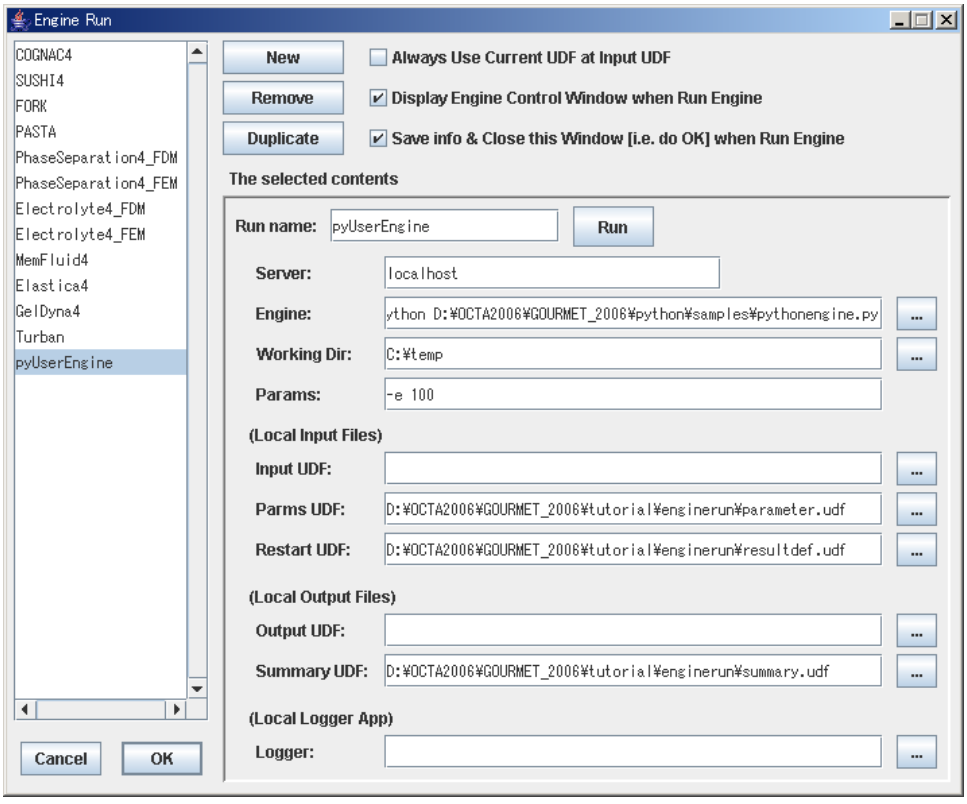


Figure 4: Engine Run Panel

(Note) Example of presetting and executing an engine.
Run name: pyUserEngine
Server: localhost
Engine: python C:\OCTA200X\GOURMET_200X\python\samples\pythonengine.py
Working Dir: C:\temp （You need to create it, if it is not found.）

Params: -e 100
Input UDF:
Parms UDF: C:\OCTA200X\GOURMET_200X\tutorial\enginerun\parameter.udf
Restart UDF: C:\OCTA200X\GOURMET_200X\tutorial\enginerun\resultdef.udf
Output UDF:
Summary UDF: C:\OCTA200X\GOURMET_200X\tutorial\enginerun\summary.udf (this file is created.)
Logger:

Figure 5 is Engine Control panel. It displays the items defined by "Summary UDF", and values that Engine Server receives from the engine. You can control the active engine, and view summary of the result.

The followings are the possible actions while engine is working.

Pause calculation,

Stop calculation,

Shut down (kill) engine.

Edit parameters.

Restart calculation.

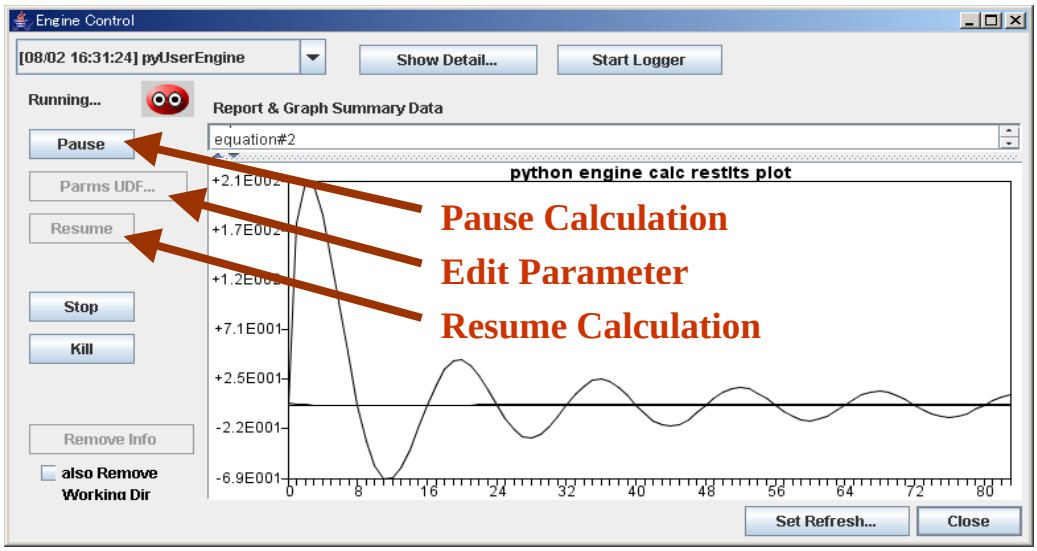


Figure 5: Engine Control Panel

At the upper left side of the screen, choose an executing engine that you want to edit parameters or view results. Tool/Engine Control menu is always available, and you can open Engine Control Panel anytime you need to. For more details, see Chapter 7.

2.5 View Result

Let's go forward to Viewer from Window/Viewer menu. Viewer is a screen where 3D object is drawn by python script. Figure 6 is the Viewer screen for the current UDF file. Ctrl+click on blank area of the 3D Object Window, and an action script gets started. Choose "show" command in pop-up menu, and "Ball" and strike zone are displayed on 3D Object Window.

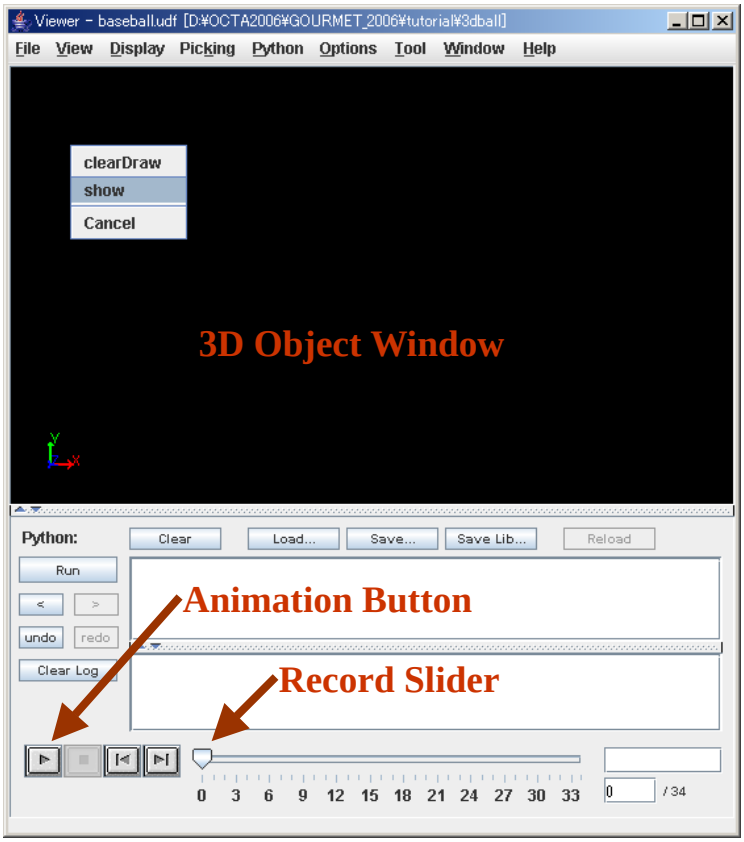


Figure 6: 3D Viewer

Now press Animation on Python Panel, perform animation of the calculation result.

Strike!

You can move forward/backward a record by animation buttons, and move to any record by record slider, because the calculation result is saved as chronological data in UDF record. For more details, see chapter 8.

Let's go back to editor, and use 2D plotting function. Choose Window/Editor from the menus. In Data

Structure Window, click "Calculated results". This is a function to execute action script. Choose "x-y plot" in pop-up menu.

Note: The UDF paths that are executable by right-clicking are in bold font.

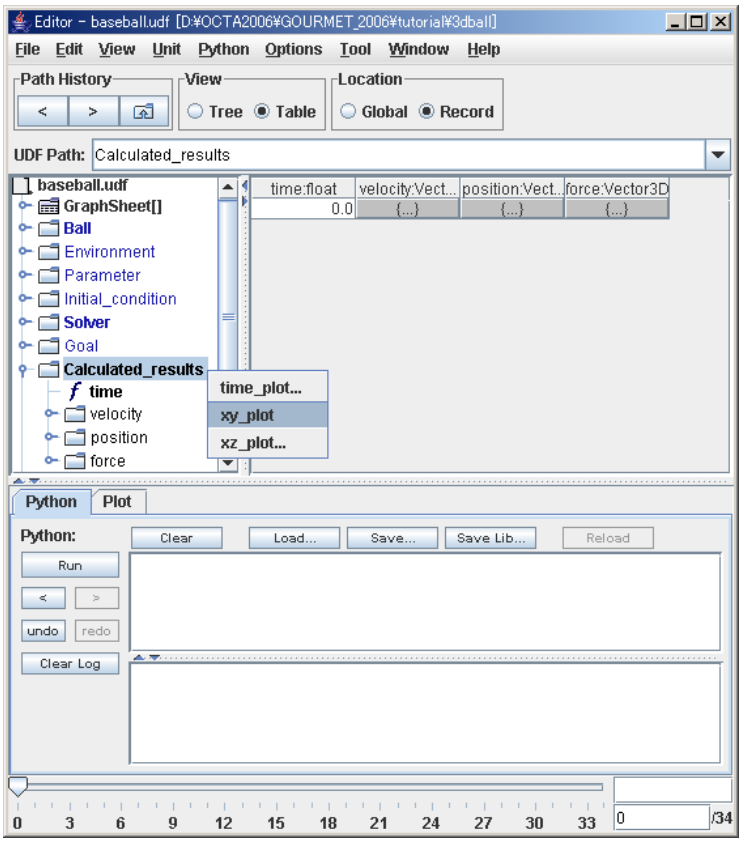


Figure 7: Plot action

Gnuplot window displays x-y plotted ball location.

There are several ways to use gnuplot tools. For more details, see Chapter 9.

2.6 Tools

Below are the available tools in Tool menu and File menu of GOURMET.

File Transmitter to transport UDF file to another PC.

Python tool to start up pure Python environment.

gnuplot tool to start up gnuplot applications.

Molecule builder to import molfile and PDB-formatted data to COGNAC UDF file.

File converter to import non-UDF formatted text data to UDF file.

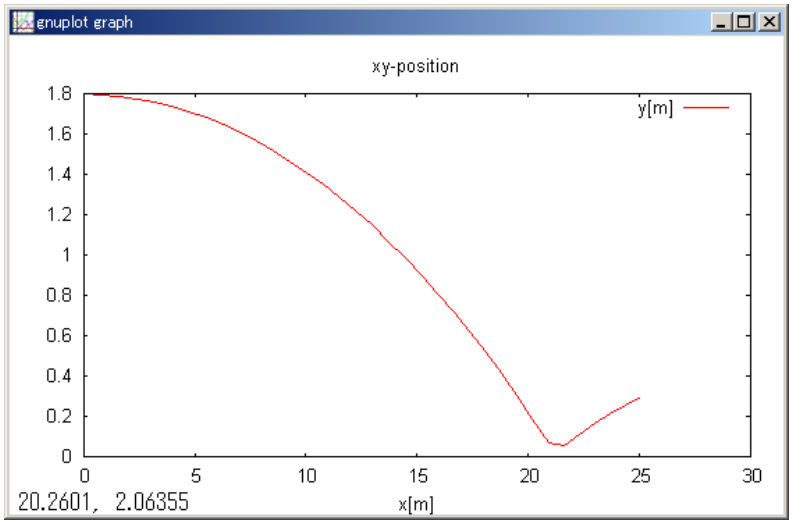


Figure 8: 2D plotting

For more details, see Chapter 10.

Chapter 3

Startup Options

3.1 System Outline

GOURMET（Graphical Open UseR interface for Material design EnvironmenT）is a GUI system whose viewing, editing, and drawing functions enable users to work on UDF file comfortably.

GUI portion of GOURMET is based on JAVA, and UDF files are supported by library coded by C/C+. 3D graphics drawing is supported by OpenGL Library via JOGL.

UDF file is processed by an interpreter language Python, and graph-plotting function is based on gnuplot.

GOURMET serves as client-server system. When it executes calculation engine or transfers a remote file, GOURMET serves as client application, and Engine Manager and Data Manager serve as server program. Engine Manager controls remote and local engine, and Data Manager transfers UDF file between local side (GOURMET) and remote side (engine).

To control engine from GOURMET, start up Engine Manager at the pc you activate an engine.

To transfer UDF file to another pc, activate Data Manager at the receiver pc.

The following chapter explains start-up script of Engine/Data Managers.

Figure 9 is the outline of GOURMET system.

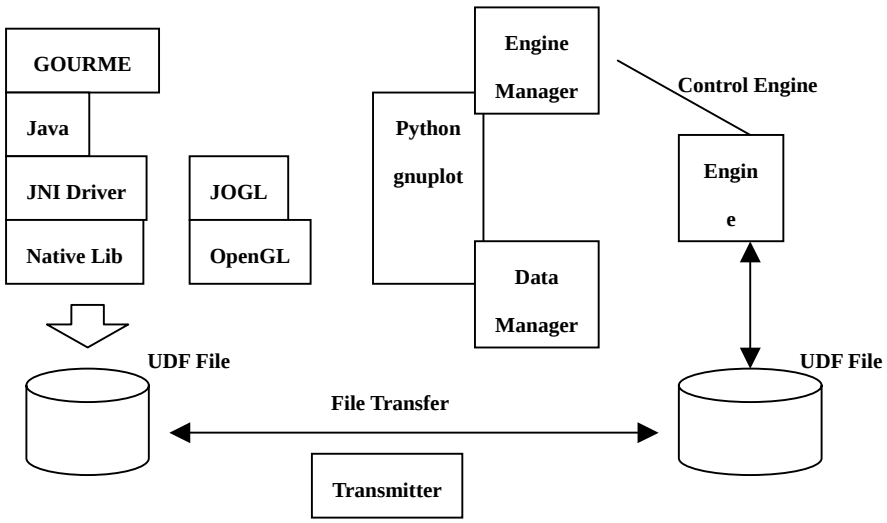


Figure 9: Outline of GOURMET System

3.2 Environment Variables

3.2.1 Needed Environment Variable

GOURMET needs an environment variable “PF_FILES”, which is the home directory path of GOURMET. This variable is automatically set up if GOURMET is installed by OCTA installer.

3.2.2 Optional Environment Variable

There are some optional environment variables.

PF_ENGINE

Home directory path of OCTA engine, which enables GOURMET locate OCTA engine.

UDF_DEF_PATH

A directory path where UDF denifition files are located. Use OS’s own delimiter to set two or more directory paths. (; for Windows, : for Linux).

UDF_ACTION_PATH

A directory path where action files is located. Two or more paths can be set up using OS’s own delimiter.

PYTHON_LIBDIR

Python library directory of GOURMET.

If GOURMET is installed by OCTA installer, environmental variables of PF_FILES and PF_ENGINE are automatically set up. The other variables above are set up by start-up script file.

The next chapter explains how to use these variables.

3.3 Startup Shell

In the Java system, you can specify the memory size to use by Java Virtual Machine Options. It is recommended that you extend using memory size before starting, because the default memory size is not enough. Use –Xmx and –Xms options to specify maximum and minimum memory size to use for Java. Remember that GOURMET may consume more memory if the UDF file has more information.

Example:

Set 256m as -Xmx and 128m as –Xms, if your pc has 512MB memories.

3.3.1 Startup Options

GOURMET can accept the following start-up options. These options are used in the startup script files (gourmet.bat or gourmet.sh).

-DPF_FILES="*home directory path of GOURMET*"

From the directory specified here, it finds directories or files to be used as default.

-DUDF_ACTION_PATH="*action file directory path*"

From the directory specified here, it finds action files to be used as default.

-DPYTHON_LIBDIR="*directory path where Python script is saved*"

It specifies the default directory where Python script is loaded/saved.

-DPF_MODULES="*Runtime library directory path*"

It specifies a directory where UDF converter is located.

-Djava.security.manager and -Djava.security.policy=policy_file_path

These two options are prepared for security.

See Engine Manager chapter for more details.

3.3.2 Microsoft Windows

To start GOURMET, choose OCTA200X => Start GOURMET from start menu, or execute “gourmet” in command prompt window, or double-click on gourmet.bat. Or just drag & drop a UDF file you want to open on “gourmet.bat” or its short-cut.

3.3.3 Linux

In shell terminal window, execute start-up script created by installer.

（OCTA200X/GOURMET_200X/gourmet）

3.4 Engine Manager

To control engine, Engine Manager should be started at the pc you activate engine.

3.4.1 Startup Options

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

These three options are prepared for security. PCs can be used as stand-alone, if these options are set up. Since stand-alone pc is not accessible from any PCs on network, its Engine Manager uses a security policy defined by policy_file_path.

Important:
If you are not confident with the safety of the network, it is recommended that you specify your own security policy by system tools.
For more information, see Engine Manager security policy in Appendix B.

3.4.2 Microsoft Windows

To start Engine Manager, choose OCTA200X => Start Engine Manager form start menu, or execute “eng_man.bat” in command prompt window.

3.4.3 Linux

To start Engine Manager, execute "OCTA200X/GOURMET_200X/eng_man" in shell terminal window. This is automatically created by OCTA installer.

3.5 Data Manager

To transfer UDF file, Data Manager should be activated at the pc receiving the file.

3.5.1 Microsoft Windows

To start up Data Manager, execute “dat_man.bat” in command prompt window.

3.5.2 Linux

To start up Data Manager, execute "OCTA200X/GOURMET_200X/dat_man" in shell terminal window.

3.6 Stopping GOURMET

To close GOURMET, use File/Exit menu on GOURMET, or close all GOURMET windows.

Chapter 4

Editing UDF

To edit UDF file, select File/Open... menu, and choose the file in File open dialog. In Figure 10, UDF file is open in Editor.

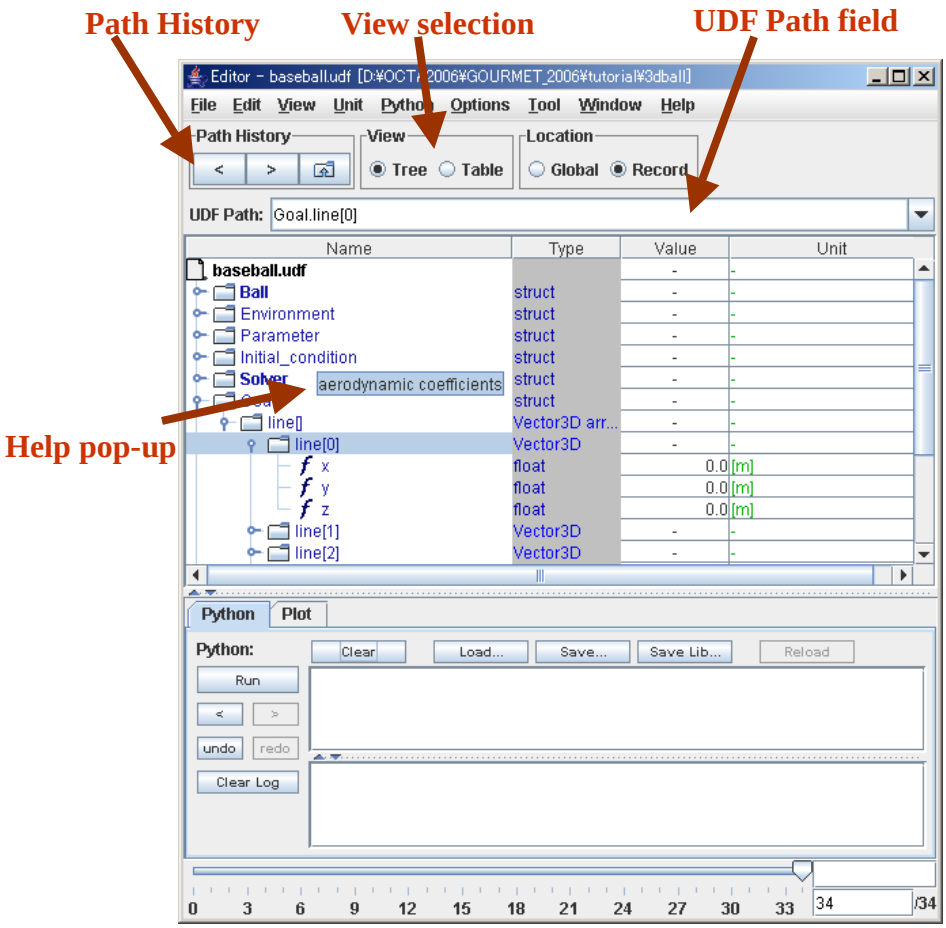


Figure 10: UDF file is open in Editor's tree view.

Use Path History to choose UDF data path from the work history, or go back to the last history.

Use View radio button to choose display type between Tree and Table.

Use Location radio button to choose location UDF data, Global or Record. It is important to know the location of data you are editing.

You can choose UDF data by typing a specific UDF path in UDF Path field.

You can see Help in pop-up for UDF data variables by placing the pointer, if help is defined. Some of the help has a hyperlink to a file on internet or local disk. Ctrl + right-click on the link, and what you choose in the pop-up should be displayed in system default browser.

4.1 Edit Mode

4.1.1 Edit Mode

Use Edit mode to change values in UDF object. In Edit mode, all data structure that could have values are displayed. GOURMET is always in Edit mode, since OCTA 2003.

4.2 Choosing View Format

4.2.1 Tree View

Tree view shows UDF data details. Tree view displays data structure and value as tree format.

From version 4.0, you can specify the number of array items to display. See Figure 11 as an example. Only the top of the arrays are displayed. Ten array value[] are displayed, and the rest is "...". The next 10 arrays are displayed by double-clicking on "...". You can change the number of arrays to expand the tree tab "Number of expanding" of 4.6.1 Editor Preferences Dialog. Default number of expanding is 1000.

To display the whole or part of tree array, right-click on "..." and you will see the index of the next array in pop-up. Click OK, and the tree view shows the data relative to the array index specified in this pop-up.

If the total number of data array is greater than the specified number of expansion, the array ends with "EndOfArray". (See figure 12) To resume the initial tree status, right-click on "EndOfArray", and display pop-up (same as Figure 12)

If the number of data array is lesser than specified number of expansion, you will not see either "..." or " EndOfArray".

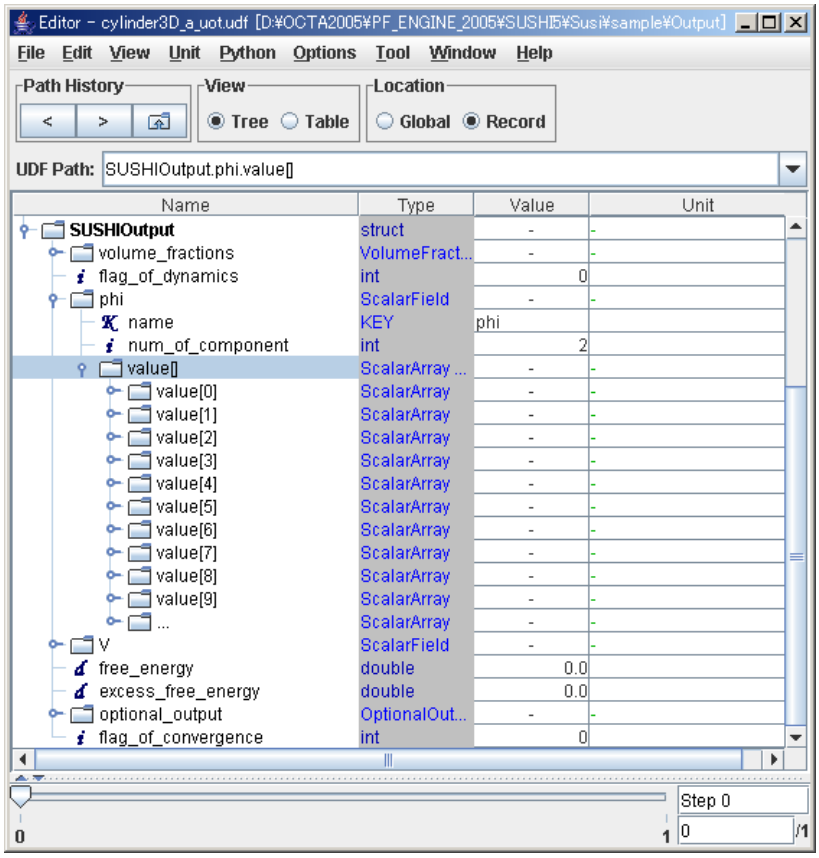


Figure 11: Array expanded on Editor Tree View.

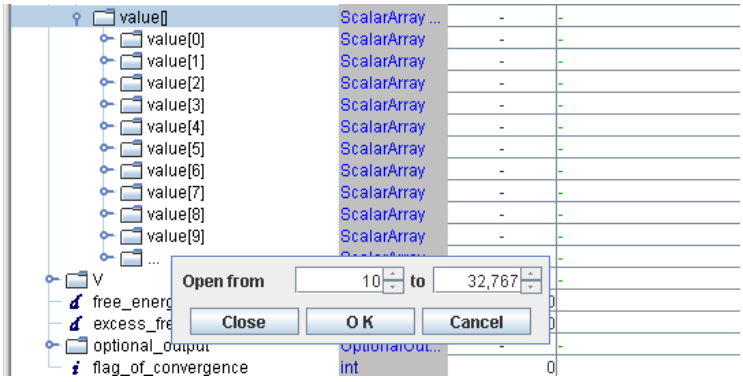


Figure 12: Pop-up after right-clicking "..."

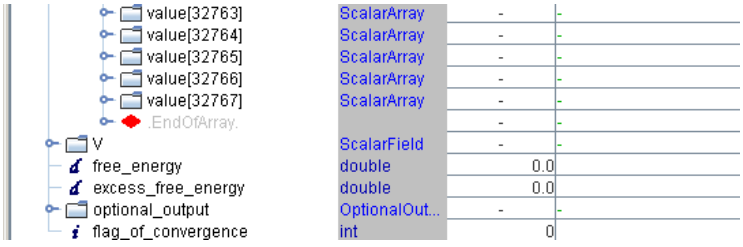


Figure 13: "EndOfArray"

4.2.2 Table View

Table View displays the structure of UDF data. Since the format of Table View is that of relational database, you can exchange data with another application by simply copying/pasting ([Ctrl]+C, [Ctrl]+V).

In Figure 14, you are looking at UDF file in Table View.

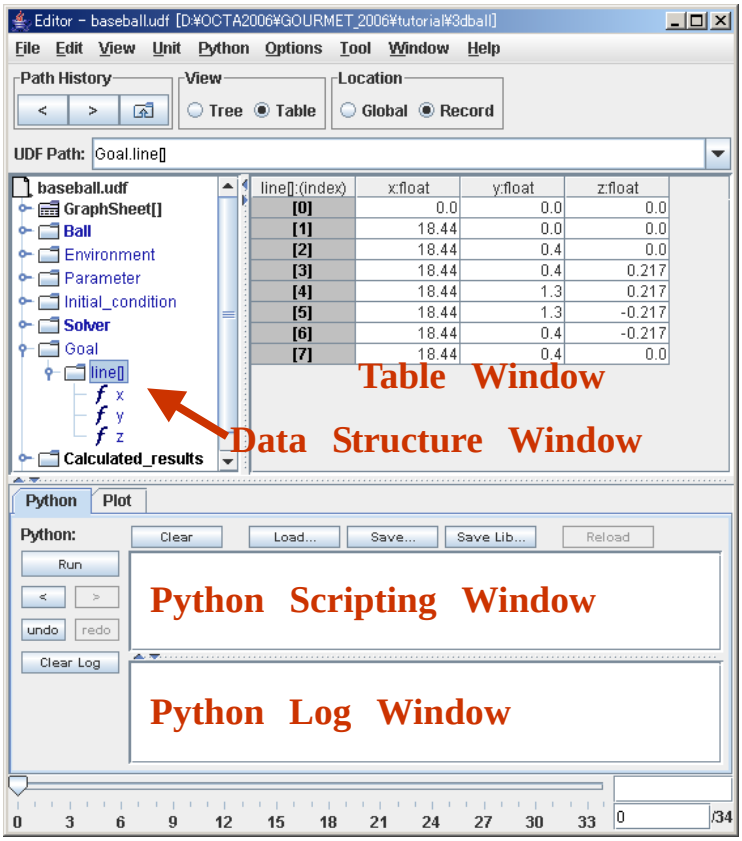


Figure 14: UDF file is open on Editor Table View.

4.3 Choosing Data Location

The initial value and the record values of UDF file have the same data name. So, you need to specify by location type which one to choose. GOURMET distinguishes location types and availability of data by color of the UDF data name. For initial value could be defined as “global_def” which enables initial value only to have data, or as “def” which enables both initial and record values to have data. If defined as "global_def", the UDF data name is blue. If defined as “def”, the UDF data name is green. "def"-defined UDF data located in record is black. UDF data with no data is gray.

Figure 15 is an example of Tree View when “Records” is chosen as data location. “global_value” is blue, because it is defined as “global_def”. 'common_num' and “common_pos” are green, because they have initial value but no value in this record. The other data is black, because these have data in this record.

4.3.1 Global Location

Click on Global radio button to edit Global Data.

Note: Global Data was called Initial Data until OCTA2002.

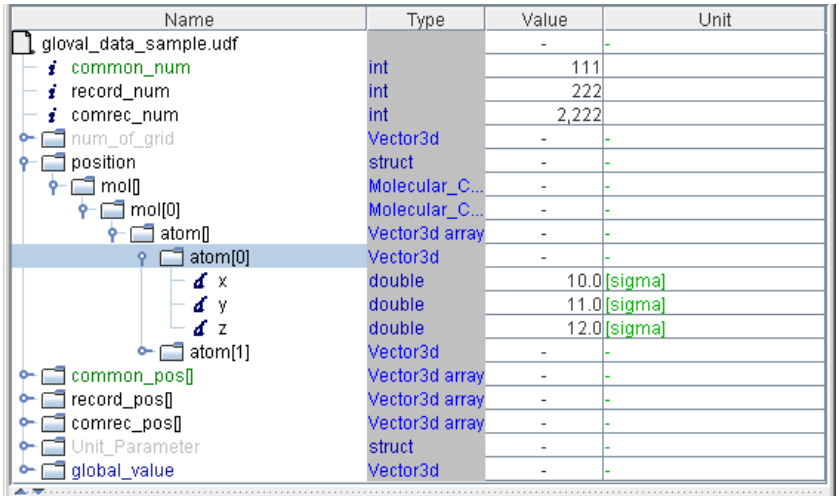


Figure 15: Color variation of UDF objects.

4.3.2 Record Location

Click on Records radio button to edit/add record data.

Record Slider is displayed at the bottom of Editor, if record data exists. You can change the location of record by Record Slider.

4.4 File Menu

Figure16 is the File menu of Editor.

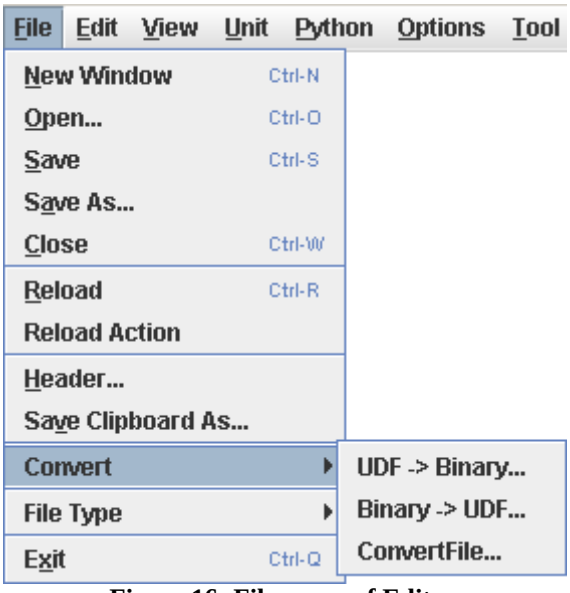


Figure 16: File menu of Editor

The following options are available in File menu.

New Window : Opens a window to see/edit another UDf file.

Open : Opens UDF file. (Autorun action is executed, if defined.)

Save : Overwrites the current UDF file.

Save As... : Saves as another file name.

Close : Closes the current UDF file.

Reload : Reloads all the data of the current UDF file. (Autorun action is executed, if defined.)

Reload Action : Reloads all action files used by the current UDF file. (Autorun action is executed, if defined.)

Header... : Sees/Edits header information of the current UDF file.

Save Clipboard As... : Saves the current clipboard to as text file.

Convert : Converts from a text-formatted UDF file to a binary-formatted UDF file, and vice versa. It also imports an outside file by file filter.

File Type : Shows the type of the current UDF file.

Exit : Exits GOURMET.

4.4.1 Editing UDF Header

To edit UDF header, choose File/Header.... Figure17 is an example of the UDF header dialog.

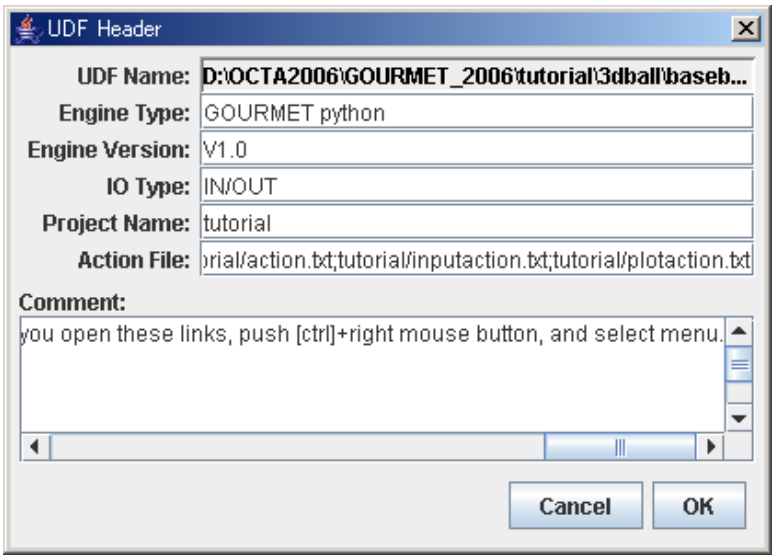


Figure 17: UDF header dialog

4.4.2 File Converter

To import fixed-formatted text data which is not UDF, click on File/Convert/ConvertFile...

Figure 18 is an example of ConvertFile dialog.

Data File:

Data file to convert. Tab-delimited, more complicated NASTRAN bulk file, etc.

Rule File:

Filter rule file to execute conversion.

You can find a couple of examples in $PF_FILES/filter/ directory.

In the following example, 2nd, 3rd and 4th columns values in a data file are substituted for atoms[].Position.x, atoms[].Position.y and atoms[].Position.z respectively.

Example:

simple example for convert molecule data from tab delimited text file

LABEL . atoms[].Position.x atoms[].Position.y atoms[].Position.z

Input UDF:

UDF definition file of conversion.

In this example, the following UDF definition file is used.

Example of Input UDF file:

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

Output UDF

UDF file that you output the conversion result.

You can easily create a new conversion rule. See Appendix D for details of File Filter.

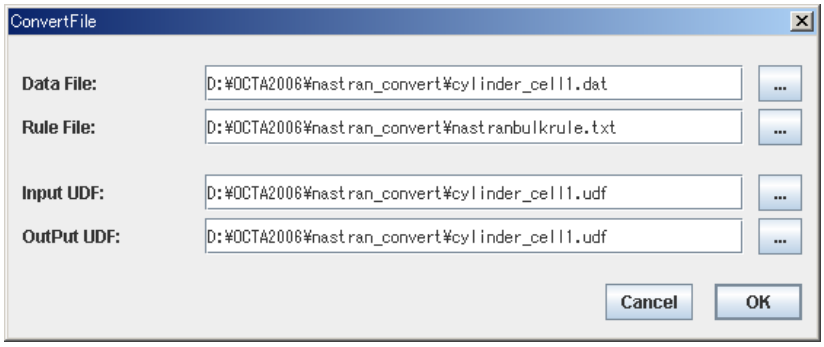


Figure 18: File Converter Dialog

4.4.3 Using Text-Formatted UDF and Binary-Formatted UDF

Binary-formatted UDF helps GOURMET to read UDF file with record data of 100MB or greater. It speeds up GOURMET's reading process.

One characteristic is that a binary file's extension is .bdf. Its record data starts with the total size of all data it has, which speeds up reading UDF file with a lot of records.

To execute conversion of UDF file from text format to binary format, choose File/Convert/UDF =>Binary...

To execute reverse (from binary to text), choose file/Convert/Binary =>UDF....

4.5 Edit Menu

Figure 19 is Edit menu of Editor.

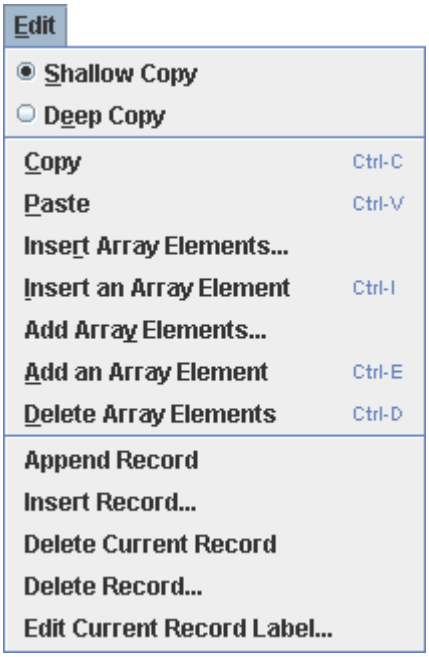


Figure 19: Edit menu of Editor

Below is the available actions in Edit menu.

Shallow Copy mode : Copies data in tabular format.

Deep Copy mode : Copies data in object format.

Copy : Copies the current data on clipboard in the current copy mode.

Paste : Pastes data on clipboard to the specified location.

Insert Array Elements... : Inserts more than one element to the specified array data.

Insert an Array Element : Inserts a new element in front of the specified array element.

Add Array Elements... : Adds one or more elements after the specifies array element.

Add an Element : Adds an element after the specified array element.

Delete Array Elements : Deletes one or more array elements that you specify.

Append Record : Adds a new record at the end of the existing record.

Insert Record... : Inserts one or more new records at the specified location.

Delete Current Record : Deletes the current record.

Delete Record... : Deletes one or more records located at the specified location.

Edit Current Record Label : Edit the current name of the record.

4.5.1 Copy/Paste Mode

Copying ([Ctrl]+C) and pasting([Ctrl]+V) operations are the same as the other applications. GOURMET has two kinds of mode as follows.

Shallow copying/pasting mode

This is to copy/paste tabular format data as it looks. GOURMET can convert data on clipboard with other application.

Deep copying /pasting mode

Since UDF file has a lot of structured data, GOURMET is able to copy/paste the whole data as an object, which we call “deep” copying/pasting mode. In this mode, data on clip board is in python list format. In deep copy/paste mode, clipboard data is in Python’s list formatted.

4.6 View Menu

Figure 20 is View menu of Editor.

Show Global : Shows data of Global location.

Show Record : Shows data of Record location.

Preferences... : Shows Editor preferences dialog .

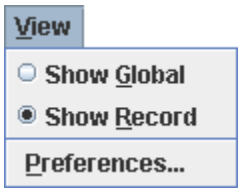


Figure 20: Editor View menu

4.6.1 Editor Preferences Dialog

You can edit the preferences of display form for Editor.

TreeView Tab and TableView Tab

Figure 21 and Figure 22 shows preferences dialog for TreeView tab and TableView tab.

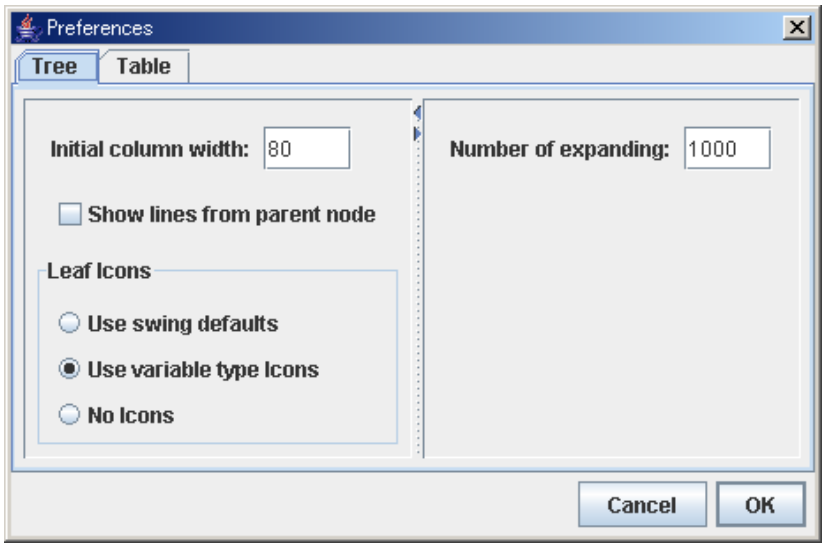


Figure 21: Preference/TreeView tab

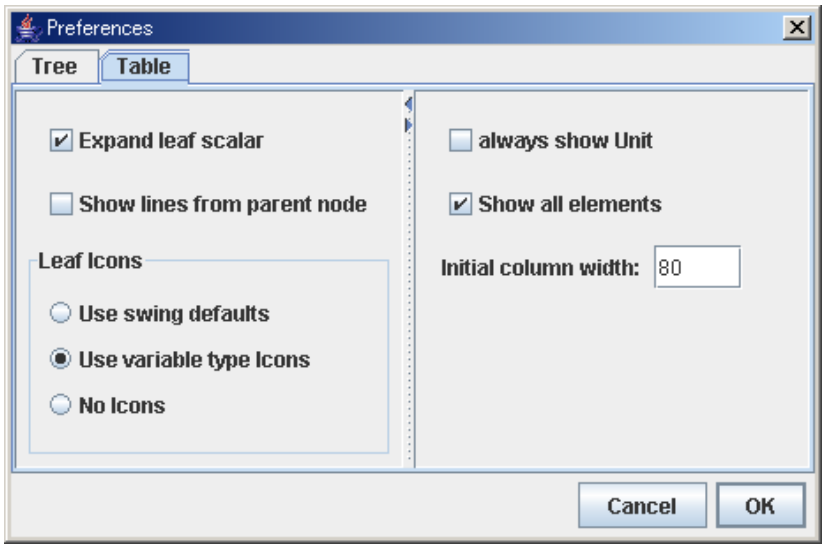


Figure 22: Preference/TableView tab

4.7 Unit Conversion

If the UDF file has unit definition, you can convert its unit in both Tree view and Table view.

In Table view, right-click on the data name that you want to convert its unit. In Tree view, right-click on unit field of the data that you want to convert its unit. Figure 23 is an example of the unit conversion dialog.

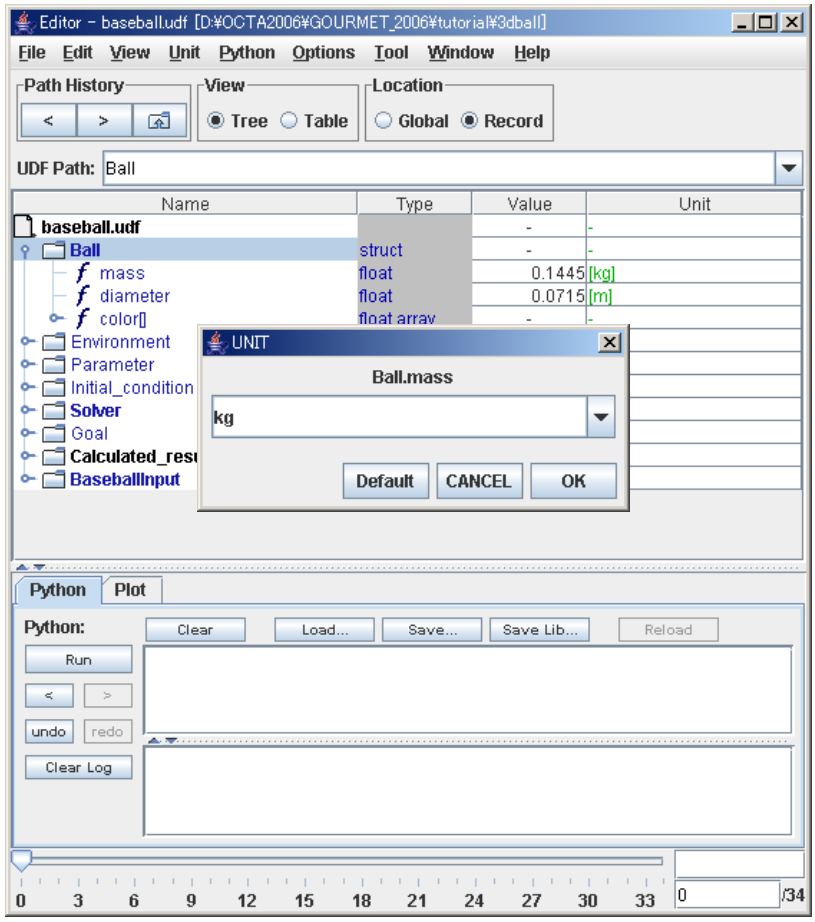


Figure 23: Unit Conversion dialog

4.8 Tips for Using Editor

Filtering by UDF Path field

UDF Path field has a filtering function. Type UDF data name that has array index in Table view, and you get a result shown in Figure 24.

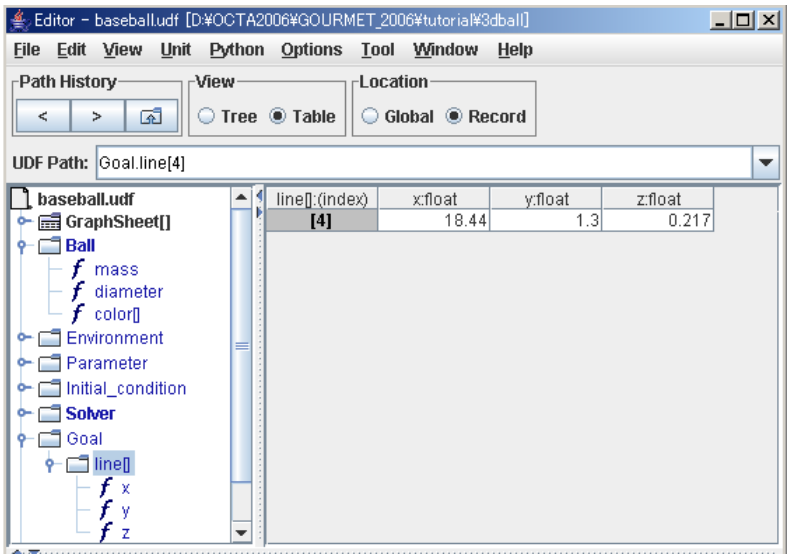


Figure 24: Filtering by UDF Path field

Displaying KEY value in Table view

If an array is KEY type data, the array's index and KEY data are displayed in Table view. See Figure 25. ("[0]" and "AGE")

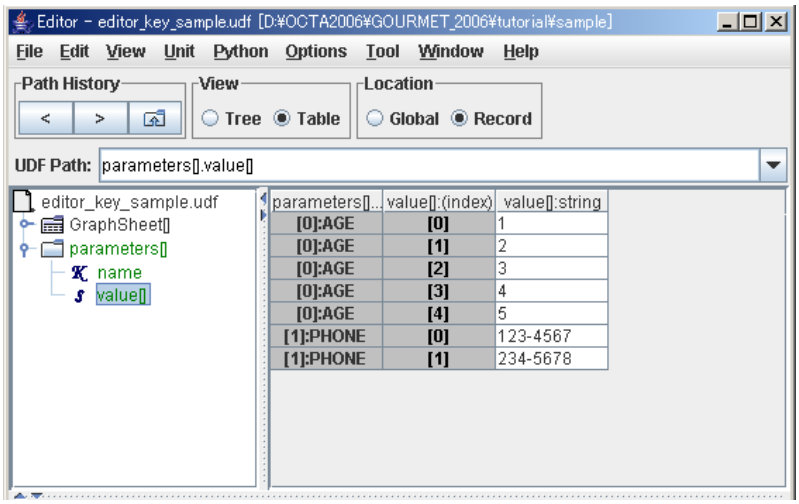


Figure 25: Displaying KEY data in Table view

Data selection by Select value

If the data is Select type, you can input data by selecting from the list. Selected data and its data structure of the same name in the same tree are displayed, and the rests are unshown.

Inserting/deleting array elements

In Table view, you can insert/delete the same array elements of the node that you select.

Execute action

The data names to which some action is related are displayed in bold font. By right clicking the name of bold font, you can execute the action script.

Display help of UDF variables

Place the mouse pointer on a data name, and a pop-up help is displayed if some help string is defined. If the help has underlined URL, [Ctrl]+right-click on it and choose a link item to display.

This works for the comment pop-up of UDF file, too. For example, open Tutorial/3dball/baseball.udf, and place mouse pointer on a file “baseball.udf” at the left top of the screen.

Inserting data into multi-dimensional array

If you insert multi-dimensional array type data, start from the highest array element. For example, in order to insert 10X10 elements to blank array2[][] using GUI menu, insert ten elements to array2[], then next 10 elements to array2[0][], etc. This operation becomes easier by inserting elements and specifying value using python script as follows.

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

Display 2-dimensional array in table format

If the data is 2-dimensional array type, its table is displayed in Table view. To choose table format in UDF Path field, make sure that 2-dimensional UDF path is written on field, and press Enter/Return If the UDF path has three or more [], type an index in [] and press Enter/Return, then its table is displayed. In Table window, click on […] to display table of 2-dimensional array type. To return to 1-dimensional format, click on the data on Tree window. Tree window cannot display tables.

Chapter 5

Using Unit System

5.1 Unit Menu

Figure 26 is Unit menu.

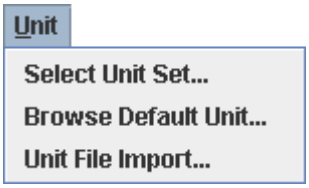


Figure 26: Unit menu

If UDF file has a unit system, unit menu is available in Editor screen.

Select Unit Set : Chooses an available unit system, and display values.

Browse Default Unit : Displays a list of default units.

Unit File Import...: Imports another unit system.

Important:

Unit conversion is effective within GOURMET, but UDF's data value is always in the unit system defined by engine.

5.2 Choosing Unit System

Figure 27 is Select Unit Set dialog. You can choose a unit system into which the data is converted, and display all the data in that unit system. The default unit system is defined by engine.

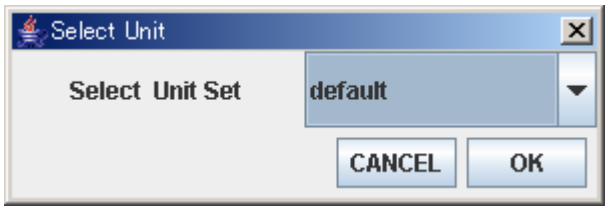


Figure 27: Select Unit Set dialog

5.3 Importing Unit System

You can import a unit system into GOURMET using Unit File import... menu.

For how to define unit system files, see UDF SYNTAX REFERENCE.

Below is an example of a unit system file.

```
\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 Displaying Engine Unit System

In "Browse Default Unit..." menu, the unit system defined by simulation engine UDF file is shown. See Figure 28.

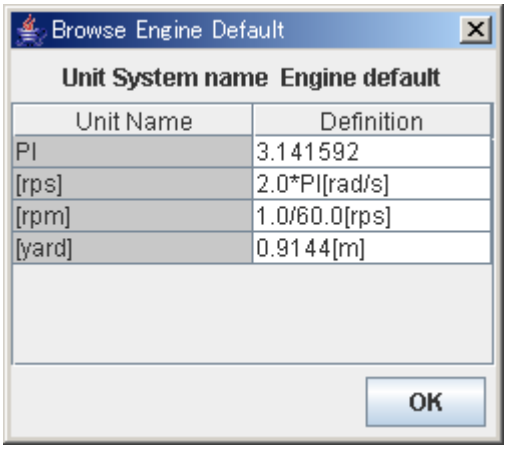


Figure 28: Browse Default Unit dialog

Chapter 6

Scripting with Python

Here, you work on data described on UDF in python script language on GOURMET, such as referring, editing, converting unit, adding records, drawing 3D object and graphs. All the script processing is executed by python interpreter system. So, both Editor and Viewer have its own Python Scripting window.

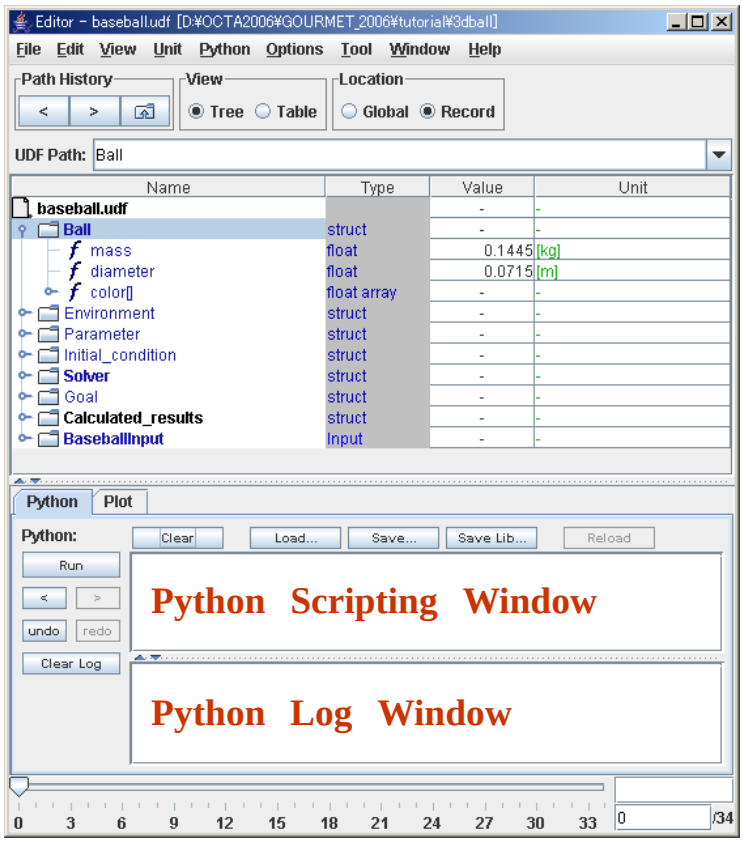


Figure 29: Python Script Window in Editor

GOURMET has four types of executing script as follows.

- Execute script in Python scripting window in Editor.
- Execute script in Python scripting window in Viewer.
- Execute an action in Editor.
- Execute a picking action in Viewer.

6.1 Tools in Python Panel and Python Menu

The following operations are available in Python menu of Editor, and Python tab below Editor.

Run : Executes script in Python Scripting window. Its result and an error message are displayed in Python Log window. If successfully completed, GOURMET memorizes the script as execution history, and the history can be reused. Script can be executed by pressing “Run” button in Python tab in Editor.

Clear : Clears the contents in Python Scripting window.

Load... : Loads a scripting text file to Python Scripting window.

Save... : Saves scripting text in Python Scripting window to a file.

Save Lib... : Python Script is converted by Python pre-parser, and saved in a file. The converted Python script can be executed in pure Python environment.

Save History... : Saves history of Python Scripting window to a file. Saved history can be resued for future sessions.

Load History... : Loads Python Scripting window history from a file. Saved script history can be reused.

Previous History (<) : Displays the previous history in Python Scripting window.

Next History (>) : Displays the next history in Python Scripting window.

Clear Log : Clears Python Log window.

AutoRun mode : “on” to automatically execute “Run” every time you change record by slider at the bottom of the screen. Viewer is always in AutoRun mode, since it is drawn by the executed script.

6.2 Python Scripting in Editor

You can execute any script available in Python system. Below is an easiest example.

```
print "hello GOURMET"
```

The same method works for expansion of GOURMET Python script. Script expansion means Python expansion to access UDF data tree.

Below is an example of useful Python expansion functions. To access UDF data, use a UDF data name

with "$" prefixed.

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

6.3 Python Scripting in Viewer

In Viewer, in addition to the same script as Editor, 3D drawing functions are available, such as basic drawing function “sphere”, structure drawing function “MeshField”.

Example:

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

Table 1 is a list of basic drawing functions of GOURMET. See Python script manual for more details.

Name of function	Remark
line(coordinate1,coordinate2,[r,g,b,a]))	RGB and transparency are specified.
line(coordinate1,coordinate2,attribute_id)	Drawing attribute ID is specified.
point(coordinate,[r,g,b,a])	
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()	

Table 1: Drawing functions

6.4 Action in Editor

In Editor, choose an action from a pop-up menu which is displayed by right-clicking on a UDF data name in bold font. The details of an action is described in action file. GOURMET finds an action file described in UDF header from the directory either where the UDF file is located, or where is specified by environment variable UDF_ACTION_PATH.

Example of action file: (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
```

Click ‘Ball’ in Editor. Figure 30 shows a pop-up, where you choose setColor. setColor specifies color information of ‘Ball’.

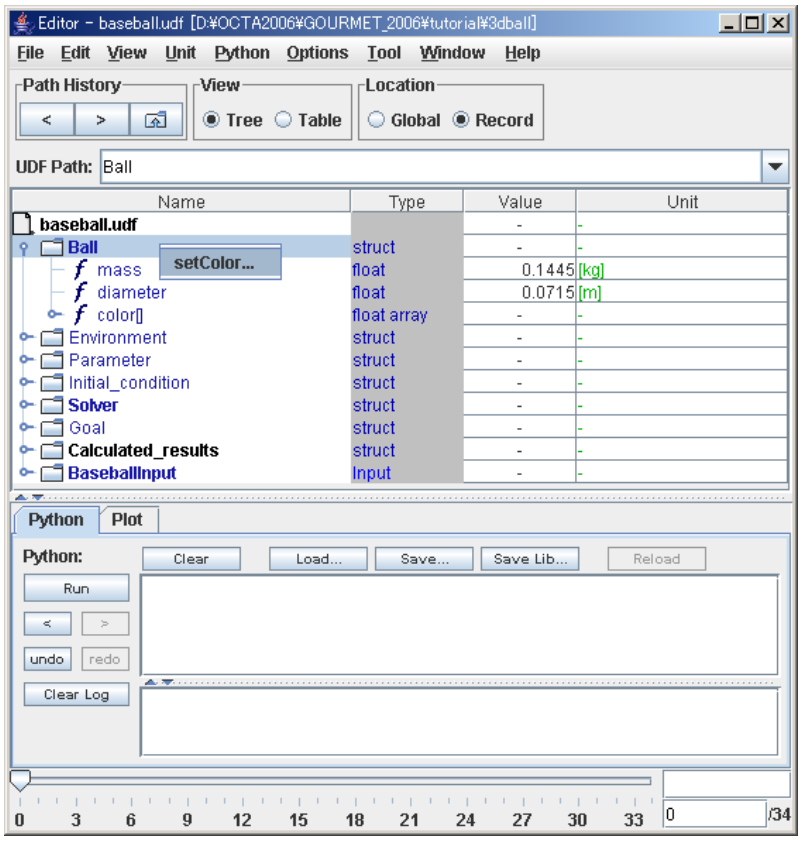


Figure 30: Example of action you take in editor.

For more details of action file definitions, see Appendix C.

6.5 Picking in Viewer

Viewer displays and chooses a list of actions related to a UDF data object that is drawn by a drawing function (CognacAtom and CognacBond, or setDrawRelation and resetDrawRelation).

For example, in Figure 31, if you [ctrl]+click on a molecule structure, you will get a pop-up of related actions. Select one action, and execute it.

For more details of action file definitions, see Appendix C.

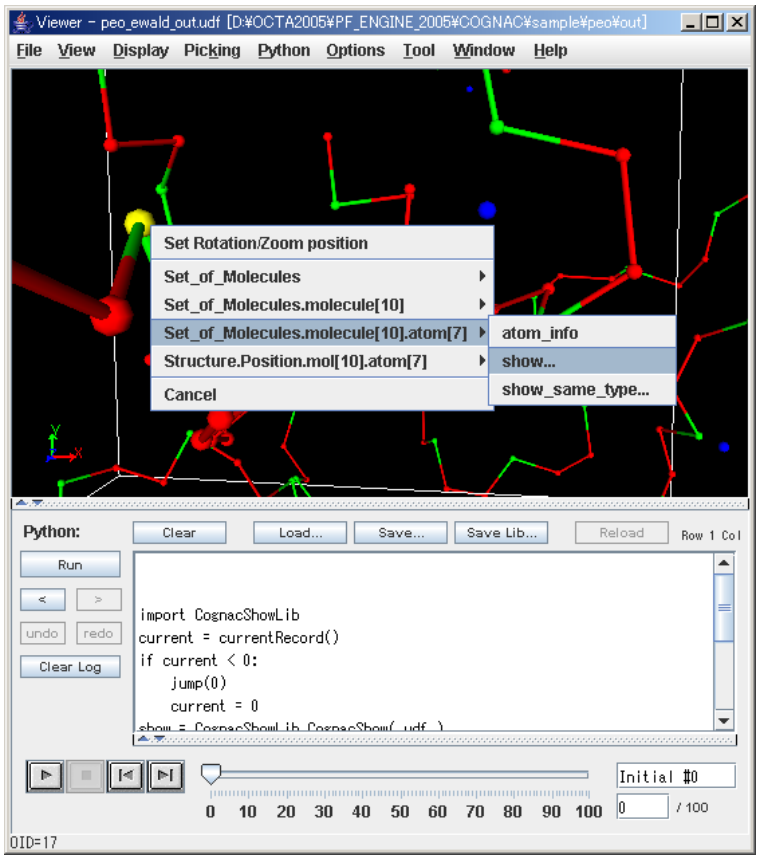


Figure 31: Actions in Viewer

6.6 Tips for Using Python Panel

You can use [Ctrl+C]/[Ctrl+V] for copying/pasting text. Since data is copied on a clipboard in table format, you can DeepCopy/paste and assign it to python variable.

Text edited in Python script window can be restored (undo button) and repeat (redo

button).Remember that undo and redo will not work after switching the whole script by history function.

Python’s error message in Python Log window includes column location of that error. If you double-click on the words including "line", script error row highlights in python script window.

You can change size of each window by a divider between Python script window and Python Log window.

Pausing Python execution. You can stop Python script whenever you want. You can also stop action being executed in Python script. Once script execution is being stopped, data processing is also stopped. To restart it, you need to close the UDF file first.

Chapter 7

Running Engine

If GOURMET is *supported* by some engine, you can start up and control engine from GOURMET via network. At least, an engine has to be able to input/output UDF file, and should be programmed by platform interface library like OCTA engines (COGNAC,SUSHI,...).

If an engine manager has to be started at engine server side, executing/controlling engine can be done by Tool/"Engine Run" and "Engine Control".

7.1 Engine Manager

In order to execute/control an engine from GOURMET, Engine Manager has to be started at the pc where you execute an engine. If the pc is local, activate Engine Manager on that pc.

Important: Engine is controlled in local pc, and independent from an engine in other pc. If you are not confident with the safety of the network, it is recommended that you specify security policy by your own OS system tools. See Appendix “Security policy of Engine Manager” for more details.

7.2 Engine Run Panel

Choose Tool/Engine Run menu, and open Engine Run Panel. See Figure 32. This is where you specify various conditions for starting up an engine. Below is the summary of each contents to specify.

Run name : Label of execution.

Server : “localhost” if an engine runs locally. Server’s host name or its IP address if an engine runs on server.

Engine : Script or module for executing engine.

Working Dir : Working directory while executing an engine.

Params : Engine execution parameters (or command line parameters).

Input UDF : Input UDF file.

Params UDF : Parameter files that is editable while an engine is executed.

Restart UDF : Restart file

Output UDF : Output UDF file

Summary UDF : Summary UDF file

Logger : Script or module for watching calculation.

Always Use... :

If it is ON, a current UDF file and its directory is displayed.

Save info & close... :

If it is checked, all the specifications are saved and Engine Panel is closed.

You can specify multiple options of execution control.

New : Newly specify engine condition.

Note: Name of engine execution is displayed on List.

Remove : Remove engine specification items.

Duplicate : Duplicate engine specification items.

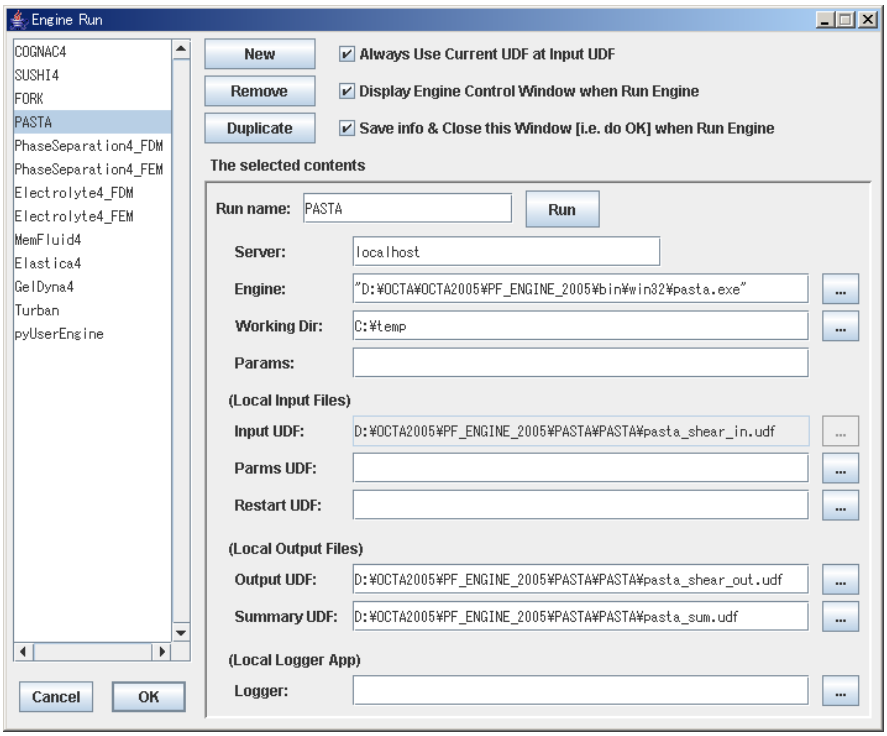


Figure 32: Engine Run Panel

7.3 Engine Control Panel

Engine Control panel is where you control a running engine, and see the summary of result. See Figure 33.

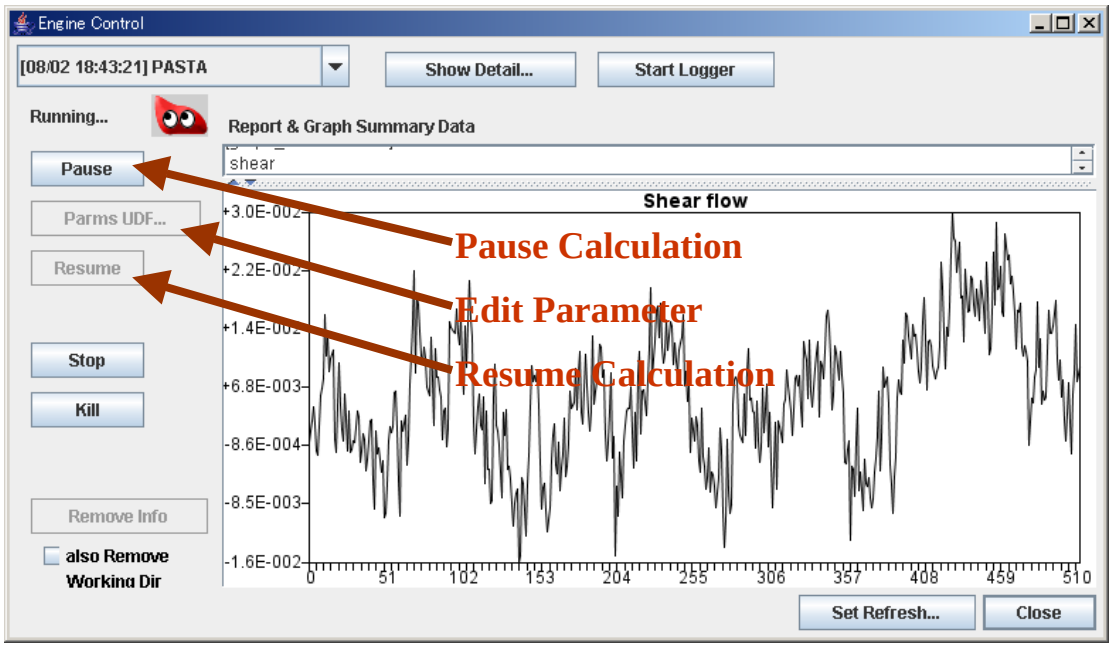


Figure 33: Engine Control Panel

If multiple engine processes are running simultaneously, you can switch from one screen to another by choosing on a combo box (format of [MM/DD HH:MM:SS] run name) at the upper-left of the screen.

In the Report & Graph SummaryData panel, the contents of summary UDF file is displayed at an interval.

In the upper text area, report_attribute.label[], report_data.value[], and graph_attribute.label[] are displayed in text.

In the lower graph area, the title "graph_attribute.title" and graph "graph_data.item[].value[]" are displayed.

The following buttons control engine processes.

Pause: Pauses a part of engine process.

Parms UDF… : While the engine pauses, it starts up Editor.

Resume : Transfers a parameter file to engine server, and resumes executing engine process.

Stop : Stops engine process.

Kill : Shuts down engine process.

Stopped or killed engine process can be deleted on Remove Info screen. If “also Remove Working dir" is ON, working directory is also deleted.

Show Detail... : Displays details of engine process that is started on Engine Run Window.

Start Logger : Starts Logger that is specified on Engine Run panel.

Set Refresh : Sets interval of refreshing Report & Graph Summary Data panel.

7.4 Tips for Using Engine Control

To reconnect to engine control panel

If an engine processes longer than one day, it keeps processing even if engine control panel is closed or GOURMET is closed. When you start GOURMET again later, you can reconnect to Engine Control panel by choosing Tool/Engine Control... menu.

To view the result of engine process

Go to pull-down menu at the upper-left of Engine Control panel.

Chapter 8

Viewing 3D Object

8.1 Viewer Startup Screen

Viewer is a window where you can draw 3D object. Enter Viewer either by choosing Viewer in the Window menu, or by executing drawing script, in Editor. Figure 34 is the start up screen.

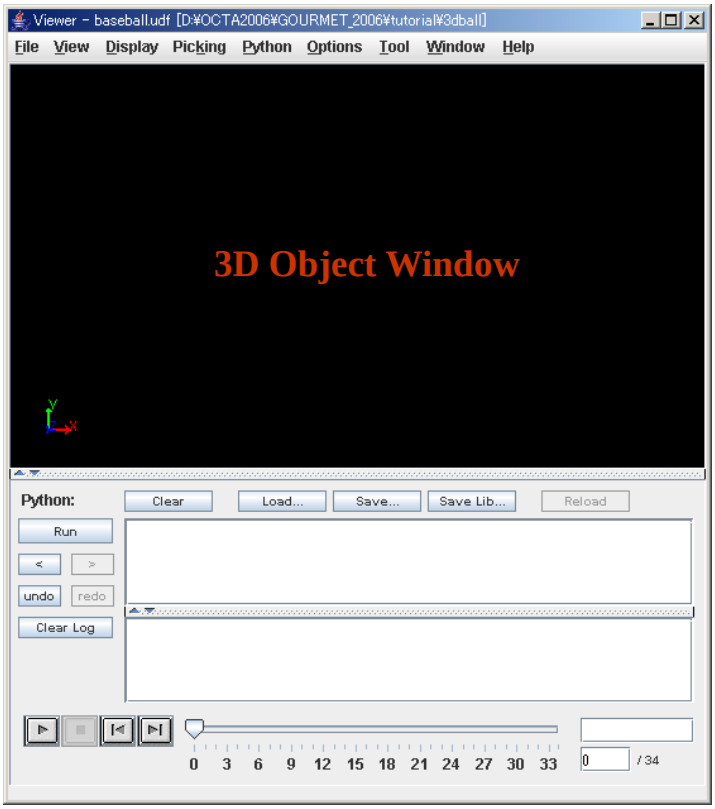


Figure 34: Start up Viewer

Viewer screen consists of the menu, 3D object panel, and python panel. The border between the 3D object panel and the Python panel can be moved by mouse pointer.

8.2 3D Object Window

The following display operations work as the same as other 3D applications.

Round (drag)
Zoom (right-click drag)
Walk through ([alt] or [shift], and right-click drag)
Walk side ([alt] or [shift], and click drag)
View direction (Go to View menu)
Background color, number of division (Go to Display menu)
Capture drawing image (Go to Options menu)
The other drawing options (Go to Options menu)
Animation (Go to Python panel)

8.2.1 Picking 3D Object Operation

You can pick a UDF object that is drawn by a special drawing function (CognacAtom & CognacBond, or set DrawRelation & reset DrawRelation), and dislay a pop-up of available action.

Default action menu is displayed by [Ctrl]+clicking on the background, if any default action (or actions to the whole UDF) is defined.

If the 3D object is drawn by "CognacAtom & CognacBond" function, and at least one action is linked to the UDF path, you can display the UDF action menu by [Ctrl]+clicking on the drawing target.

You can choose multi-picking mode by Picking/Multi Picking Mode menu, or [Ctrl]+M.

See Appendix C for more details of action file definitions.

8.3 Python Window in Viewer

Python window of Viewer has the same functions as Editor. In addition, it controls 3D animation by Start/Stop/Backward/Forward buttons at the bottom.

8.3.1 3D Object Animation

The UDF has two or more records, you can change the record to display by operation button (Start, Stop,

Backward, and Forward), or a slider at the bottom of the screen. In Viewer, the last Python script is re-executed by moving another record you specify.

Start:

Animation starts from the current record to the end record.

Stop:

Stops animation.

Backward:

Animation starts from the current record to backward direction. An operation is kept going while you push this button, and stopped if you leave the button.

Forward:

Animation starts from the current record to forward direction. An operation continues while you push this button, and stops if you leave the button.

8.4 Menu of Viewer

8.4.1 File Menu

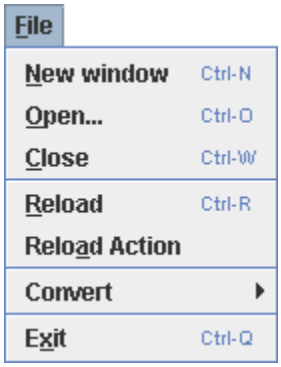


Figure 35: File menu of Viewer

Open...: Opens UDF file

Close: Closes UDF file

Reload: Reloads current UDF file.

Reload Action: Reloads action files linked to the current UDF file, and executes auto-run action if

exists.

Convert : Converts text-formatted UDF file with binary-formatted UDF file. Or, displays a screen where you import any outside file using file filter.

Exit: Exits GOURMET.

8.4.2 View menu

Here, you can change directions of 3D object window.

Standard:

All the objects in 3D panel are displayed. Click on Run button, and execute drawing script, the View goes in Standard state.

Reset:

Resets all the operations you have done in 3D panel, and returns to the last Standard state.

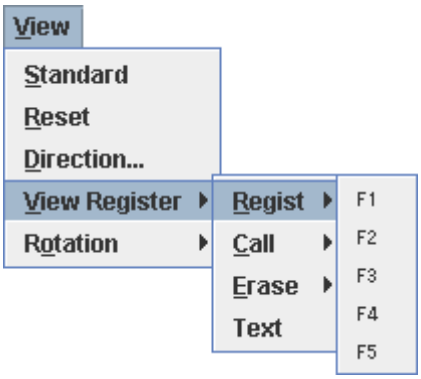


Figure 36: View menu in Viewer

Direction...:

Changes the direction of View in 3D panel. Standard direction and current direction.

Direction is specified by a vector from an eye point to a view direction in View set-up dialog.

The distance between an eye point and a center of all drawing objects are kept constant regardless of the size of the direction vector.

View Register:

Regist/Call a scene of drawings, and erases the registration. Maximum number of view registration is five, from F1 to F5. To register, press [Ctrl]+function key. To call, press function key only.

The font of F1～F5 becomes bold, once it is registered. Registration will not be deleted after Viewer

screen is closed.

Figure 37 is the text screen of current view data which is displayed by choosing Text of View Register. You can keep this text data, and use it another View screen.

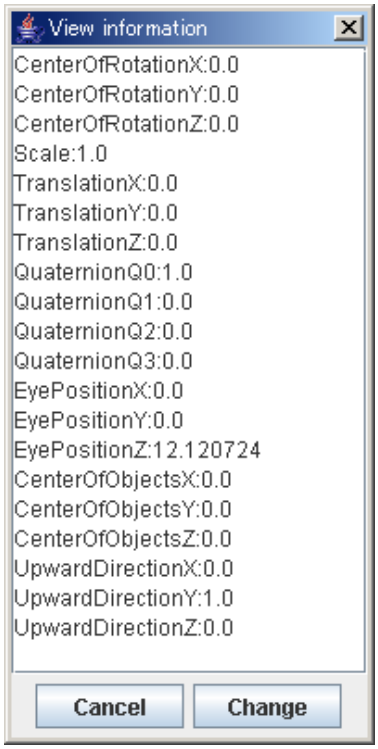


Figure 37: Current view data in Text screen

Press Change button to change drawing scene in accordance with the view data in text screen.

Rotation:

Left: Rotate to the left at a specified angle.

Right: Rotate to the right at a specified angle.

Up: Rotate to the upward at a specified angle.

Down: Rotate to the downward at a specified angle.

Clockwise: Rotate clockwise at a specified angle.

Anti-clockwise: Rotate anti-clockwise at a specified angle.

Angle: Specifies an angle by either key (cursor and [] key) or the above menu.

8.4.3 Display Menu

Figure 38 is the display menus of 3D object window.

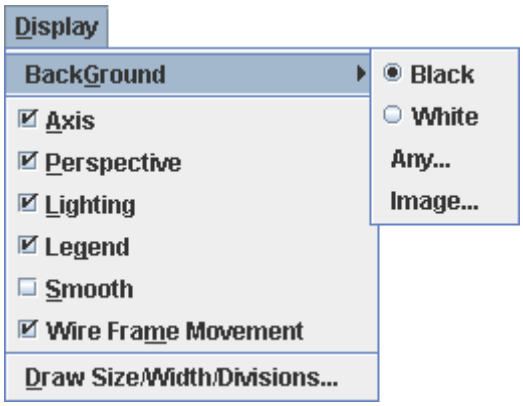


Figure 38: Display menu of Viewer

BackGround:

Black: Blacken the background.

White: Whiten the background.

Any...: Change color to any color specified in color dialog.

Image...: Paste a JPEG or PNG-formatted image data.

Fit max: Maximize the size of the image to fit the screen so that the whole image is displayed, without changing its aspect ratio.

Fit min: Maximize the size of the image to cover the whole screen, without changing its aspect ratio.

Raw size: Paste the image in its original size.

Fit canvas: Fit width and height to the size of the screen.

Axis:

Switch on/off displaying XYZ axis.

Perspective:

Switch displaying between Perspective projection and Orthographic projection.

Wire Frame Movement:

Switch on/off wire frame movement which speeds up pointer operation (round, move and zoom).

Lighting:

Switch on/off lighting effect.

An object looks stereoscopic with lighting effect on.

Colors of object looks single regardless of the direction of View with lighting effect off.

Note: Sphere's 3D appearance is lost with lighting effect off.

Smooth:

Smooth a line displayed.

Draw Size/Width/Divisions...:

Size of Point, width of line, number of polygon division of sphere, ellipsoid, cylinder, cone, disk, and ellipse.

Note: The more you increase the number of polygon division, the smoother an object is displayed, but it consumes more memory, and causes deterioration in response.

8.4.4 Picking menu

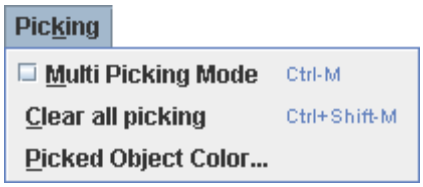


Figure 39: Picking menu in Viewer

Multi Picking Mode

Switch between single picking and multiple picking.

Clear all picking

Clear all picking status.

Picked Object Color...

Change color of picked object in Color Dialog.

8.4.5 Python Menu

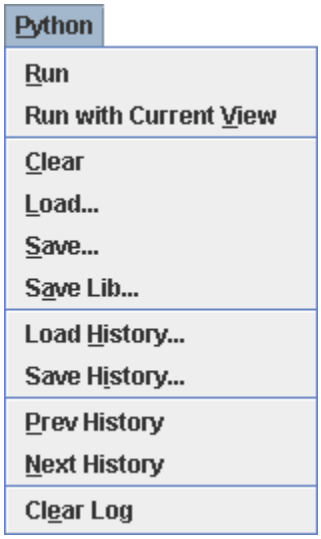


Figure 40: Python menu in Viewer

These are the same as Python panel buttons. See 6.1 for more details.

8.4.6 Options Menu

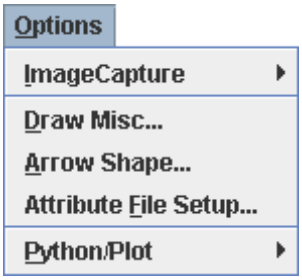


Figure 41: Option menu of Viewer

ImageCapture:

You have a couple of options in regards to saving drawing area as an image file.

- Now Capture GraphicPanel

Save the current drawing area in an image file.

- Auto Capture GraphicPanel while animation

Switch ON/OFF auto–saving drawing area as image files.

Note: Below is the default image file path.

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

例:

ImageCapture/blend_eq_uot.udf/graphic/999.jpg

- **Save Option...**:

Specify the directory where an image file is saved. Choose the format of image file (either JPEG or PNG). Specify Quality(0-100) for JPEG-formatted file.

Draw Misc...:

Specify operations for drawing.

- **Clear Option**:

Choose either running a new object before clearing the former drawing, or keeping the former drawing.

- **Cache Type for animation**:

Specify how to work with drawing object when you execute animation.

+ **USE UDF-Cache**

Execute a drawing script, and draw its result. This mode avoids mis-connecting drawing object and its related UDF data.

+ **USE Viewer File-Cache**

Speed up the second or later drawing by using file-cache, only if the drawing script has no change from the first one. This is possible because drawing contents of each Record are automatically saved in Viewer File-Cache.

+ **USE Viewer Memory-Cache**

Speed up the second or later drawing by using memory-cache, only if the drawing script has no change from the first one. This is possible because drawing contents of each Record are automatically saved in Viewer Memory-Cache.

Note: Viewer File-Cache and Viewer Memory-Cache are not effective other than the following four operations.

+ Animation operation (Start,Stop,Prev,Fwd)
+ Animation slider operation
+ Inputting in Record display text area
+ Inputting in Record Label text area

- **Animation Interval[millisecond]**:

Input interval between animations in millisecond.

- **Animation Slider**:

Choose either drawing every Record that mouse pointer is placed, or drawing only the last Record on which you release the mouse button, while you drag the animation slider.

- **Auto Rewind**:

Specify if you repeat animation or not.

Arrow Shape...:

Specify the shape of the array.

- **User arrow length scale**

  Specify the scale of length for line part of the array drawn by arrow([x1,y1,z1],[x2,y2,z2],[r,g,b,a,h,w,s]). Normally 1.0.

- **Mesh arrow length scale**

  If you draw vector value specified on mesh grid point by meshfield, you specify the length of array line by how many times to multiply the maximum mesh size. (Normally 1.0)

- **Arrowhead Shape**

  Choose the arrow end either Cone or Line. For line, you can specify the number of lines.

- **Auto arrowhead size**

  "Resize" means to draw an arrow head at a specified rate to an head of its maximum line length “scale”. "aspect" is a ratio of arrow head width to arrow head length.

Attribute File Setup...:

Choose set-up files for displaying, such as colors. For more details of set-up files, see Drawing Attributes file of Python Script Manual.

Python/Plot:

Specify a Font and History to display on Python/Plot script panel.

- **Font...**

  Specify font name, style and size, for Python/Plot script panel.

- **History...**

  Choose either save execution history of script or not. Also specify maximum number of history.

Chapter 9

Making Plot

You can use gnuplot to analyze the calculation result.

9.1 Plot Tool

You can switch between Python panel and Plot panel by Python/Plot tab. Figure 42 is the Plot panel.

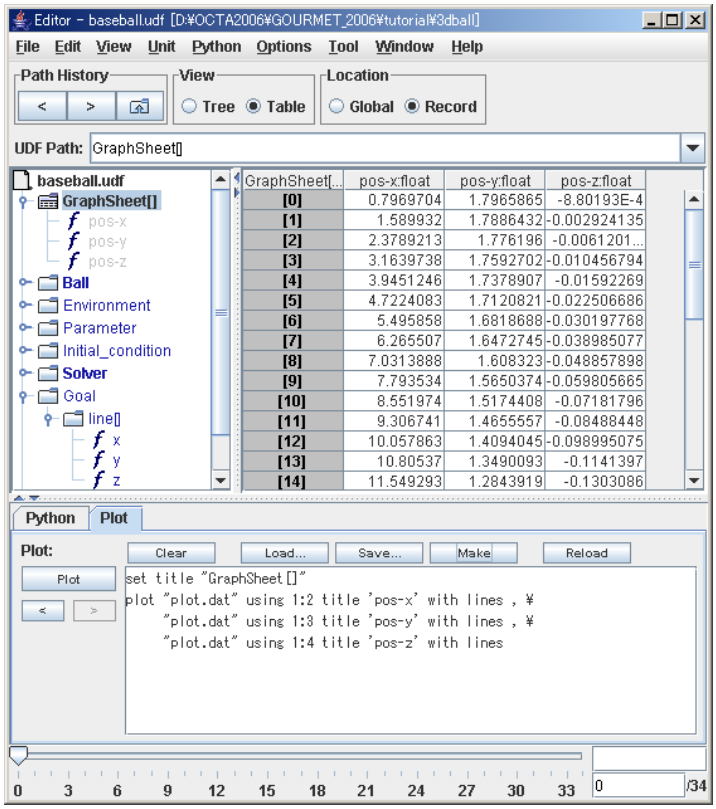


Figure 42: Plot tab panel

The purpose of the graphic function of Plot tool is to visualize data easily. In order to specify the graph format in detail, either edit plot command created by Plot tool, or change parameters by gnuplot interpreter.

Following is the steps to create plot.

Display the data you want to plot in Table view.

Press Make button of Plot tab panel, and plot command and data are created. A plot data file "plot.dat" is created in the current directory.

Edit plot command.

Press Plot button. gnuplot application receives plot command and data, and gets started. A plot command file "cmd.dat" is created in the current directory. Figure 43 is an example of displaying plot by gnuplot application.

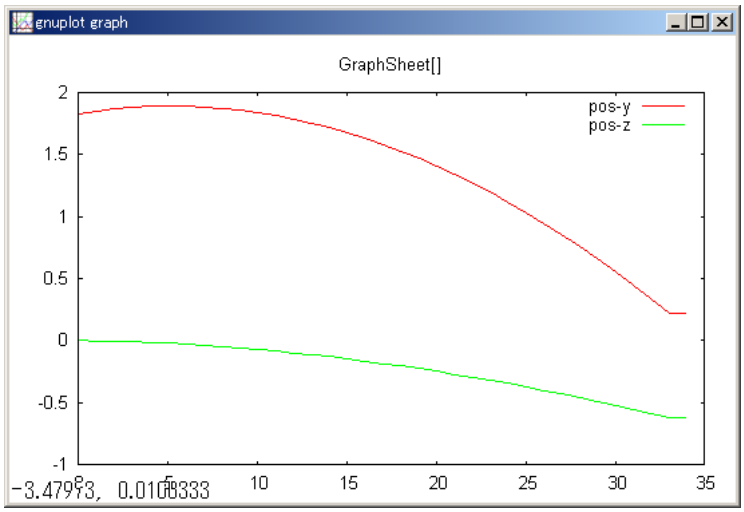


Figure 43: gnuplot graph sample

9.2 Graph Sheet Object

GOURMET has a work sheet object called graphsheet for each active UDF, where you store array data. You can also store plot data.

Its data, normally created by python script, can be directly typed out as other UDF object.

To facilitate building a worksheet, available Python functions are to add/delete a row, and adding a data column.

9.2.1 Python Function for GraphSheet

Below is an example of Python script that operates graphsheet. For more details, see "GOURMET Python Script Manual".

Example:

```
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 Scripting

GOURMET has a gnuplot interface library called gunplot.py. It is used in the following Python script.

Action script
Script window
Logger script

Below is an example of plot script.

```
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 Using Plot Script Library

Followings are the method summaries of each plot script. For the detailed specifications, see GOURMET_2007/python/gnuplot.py.

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

Write plot data.

“udf” is a UDF Manager object. “datafile” is a file name. “labels” is a list of data names to be plotted. ”axis” (either field or row), identifies the data series to be plotted.

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

Write plot data.

“datafile” is a file name. “datalist” is a list of plot data. ”axis” (either field or row), identifies the data series to be plotted.

start(cmdfile='plot.cmd')

Invoke gnuplot. “cmdfile”includes gnuplot command.

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

Write commands to “cmdfile”, and invole gnuplot. “cmdfile” includes gnuplot command.

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

Create “cmdfile, and invoke gnuplot.

“datalist” is a list of plot data, which is ignored if “udf” is specified. “title” is a title of plot. “labels” is either a plot label or UDF path. “attrs” is a graph style of gnuplot, such as ‘lines’ and ‘points’.

“udf” is an instance of current UDFManager. “cmdfile” is a command file path. “datafile” is a data file name referenced by cmdfile. “axis” (field or row) identifies the data series to be plotted.

Chapter 10

Using Tools

10.1 File Transmitter Tool

File Transmitter transfers UDF file between UDF servers whose Data Manager is active. File Transmitter provides the following features.

Transfer files between remote pcs whose Data Manager is active.

Provide basic file operations such as Make Dir, Rename, and Delete.

Display header information of UDF, such as engine type and version.

If you use it on local pc, Data Manager does not have to be active.

The left side of the screen is the directories of the current UDF. If UDF is not opened, it shows those of GOURMET's execution path, instead. To connect your pc with remote pc, type host name or IP address in the text area of Remote Host, and press Connect button.

In Figure 44, Transmitter is started, and connected with a remote host pc.

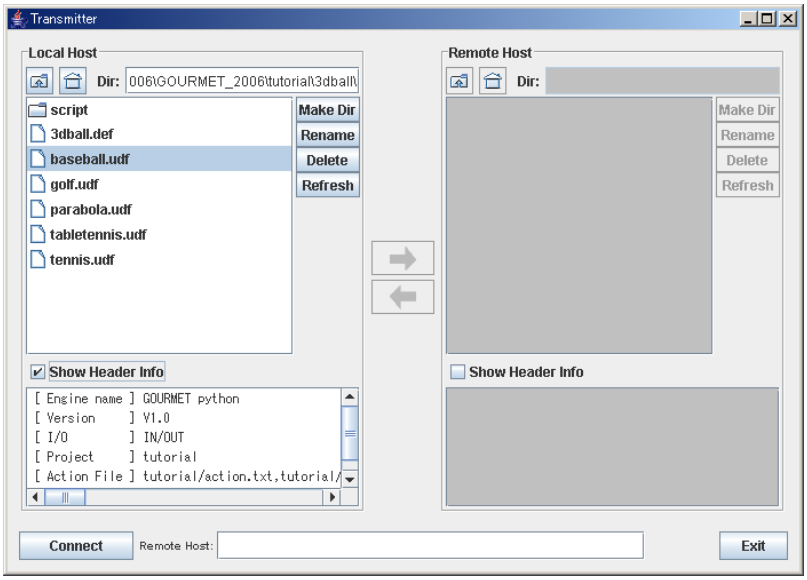


Figure 44: Initial screen of File Transmitter

After being connected, the directories displayed in the Remote Host panel are either those specified by execution parameters, or execution directories. If the Remote Host runs as Local, Remote Host and Local Host display the same directories.

Transfer files between local pc and remote pc.

Choose any file in either Local Host or Remote Host, and press => (Send) or <= (Receive).

Basic file operations in Local and Remote.

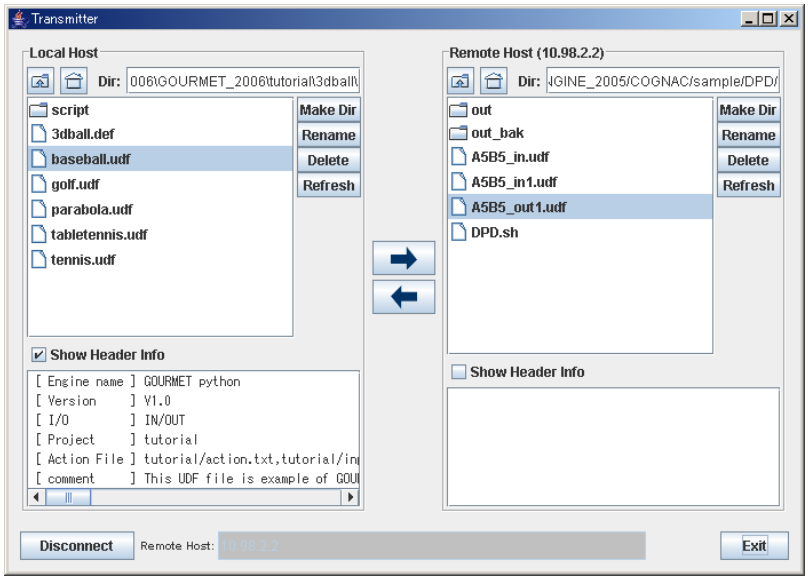


Figure 45: Transmitter Connected

Display UDF header information.

Show Header Info displays the header of the selected file.

10.2 Python Tool

Invoke a pure Python interpreter without GOURMET extension. Start-up parameters can be specified by Application Setup. See 10.4.

10.3 Gnuplot Tool

Invoke gnuplot application. Start-up parameters can be specified by Application Setup. See 10.4.

10.4 Application Setup Tool

Specify Python module in Python Application, Python start-up parameters in Python Argument. Also, specify gnuplot module in Gnuplot Application, Gnuplot start-up parameters in Gnuplot Argument.

Figure 46 is Application Setup tool dialog.

Important:

File names that have one or more space should be enclosed in “ ”.

(Example）

Python Application:"C:\OCTA200X\GOURMET_200X\bin\win32\Python\pythonw.exe"

Python Argument: "C:\OCTA200X\GOURMET_200X\bin\win32\Python\Tools\idle\idle.pyw"

Gnuplot Argument: "C:\OCTA200X\GOURMET_200X\bin\win32\gnuplot\wgnuplot.exe"

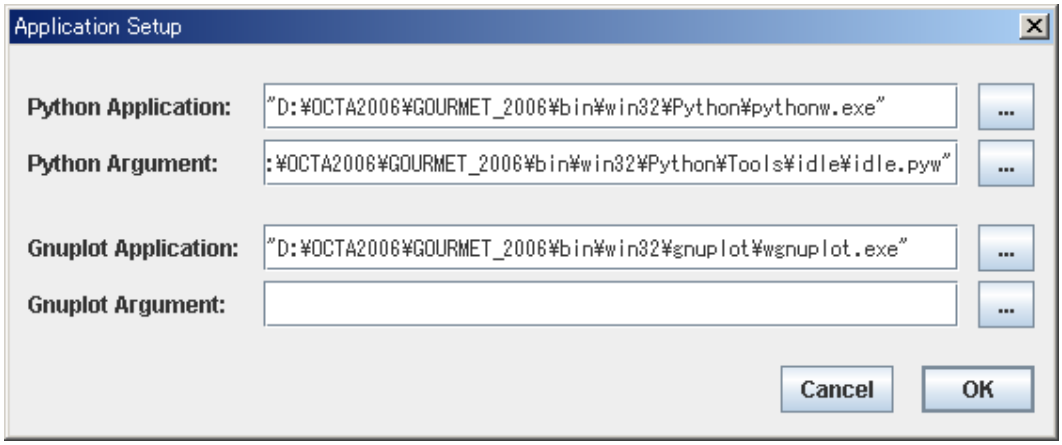


Figure 46: Application Setup

10.5 Molecular Builder Tool

Import molfile-formatted files and PDB-formatted files to UDF file such as COGNAC UDF.

Below are the Molecule Builder’s functions.

Read molfile-formatted or PDB-formatted file by a specified file filter.

Auto-create bond angle and dihedral angle.

Convert the above data following the definitions of Input UDF file.

Save the result to the specified Output UDF file.

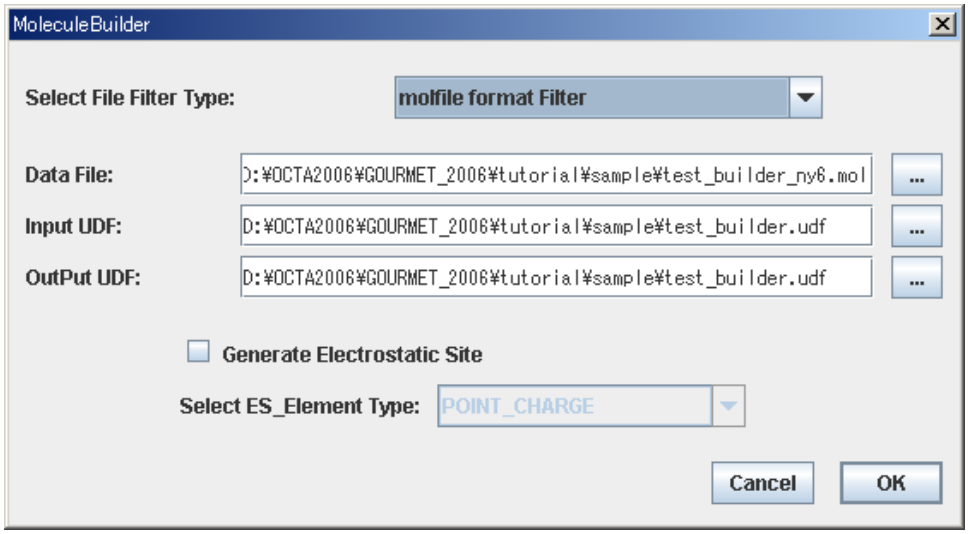


Figure 47: Molecule Builder

10.6 Start-Up Environment Parameter Tool

Edit details of environment set-up file for starting up GOURMET.

（platform_win32.ini,platform_linux.ini）

Followings are the available environment variables.

Installation directory of Python (PYTHONHOME)

Search directory of Python script (PYTHONPATH)

Execution path (PATH)

Library path (LD_LIBRARY_PATH of Unix/Linux)

Followings are the available set-up specifications by group.

Search directory of action file (UDF_ACTION_PATH)

Search directory of Python script (PYTHONPATH)

Search directory of Include UDF file (UDF_DEF_PATH)

Execution path (PATH)

Library path (LD_LIBRARY_PATH of Unix/Linux)

To set up the environment parameters by group, choose the top directory of a group and automataically specify the following directory path as a group's start-up environment.

Search directory of action file (UDF_ACTION_PATH)

"Top directory"+"/action"

Search directory of Python script (PYTHONPATH)

"Top directory"+"/python"

Search directory of INCLUDE UDF file (UDF_DEF_PATH)

"Top directory"+"/udf"

Execution path (PATH)

"Top directory"+"/bin"

Librayr path (LD_LIBRARY_PATH of Unix/Linux)

"Top directory"+"/lib"

To avoid automatic specification, check "detail".

This tool replaces each of the following key words included in an environment set-up file to its right-hand side.

%ARCH%,${ARCH} : architecture name

%PF_FILES%,${PF_FILES} : PF_FILES environment variable

%PF_ENGINE%,${PF_ENGINE} : PF_ENGINE environment variable

%OCTA_DIRECTORY%,${OCTA_DIRECTORY} : OCTA installation path

When GOURMET is started, Environment variables creation tool (gourmet_init) reads environmental parameters, and creates environmental variables necessary for starting up. See Appendix G for the details of environment variable creation tool.

Figure 48&49 are the examples of start-up environmental parameters.

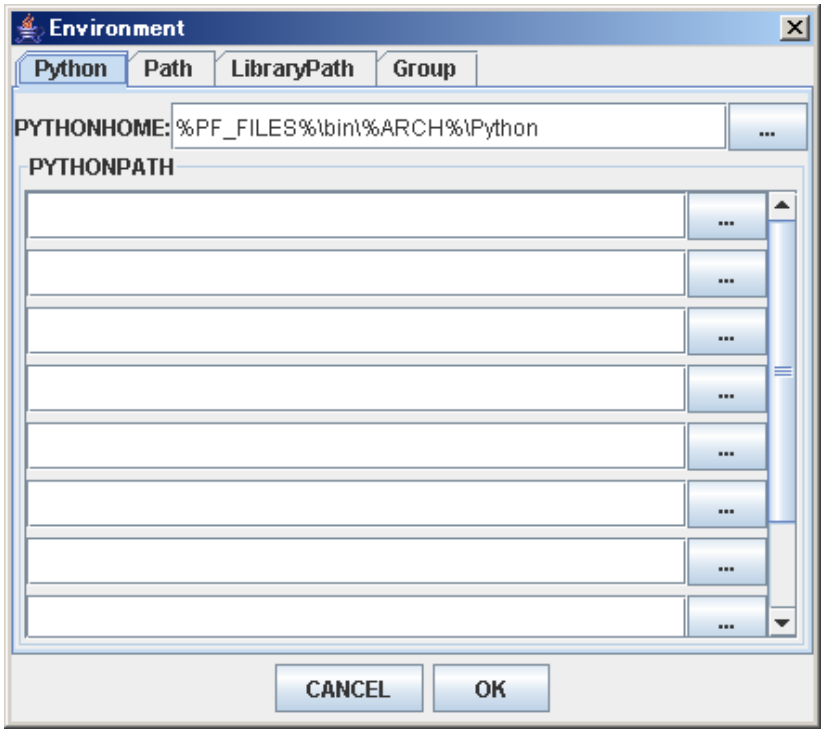


Figure 48: Python tab of Start-up environmental parameters tool

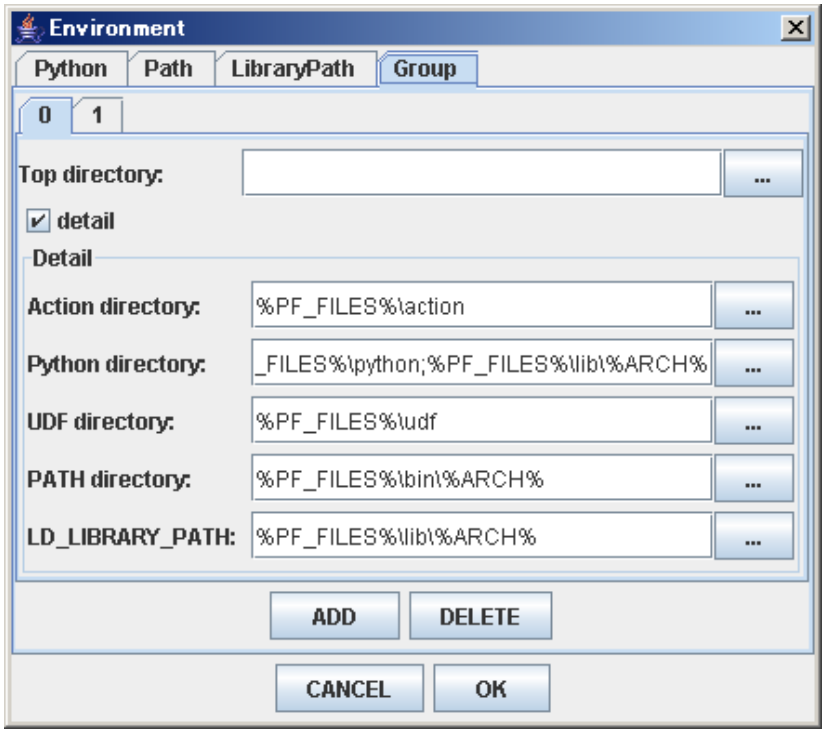


Figure 49: Group tab of Start-up environmental parameters tool

Appendix A Operation environment

A.1 Operating System

Operating System	Version	Comments
Microsoft Windows-NT (Intel)	4.0 SP6	Tested
Microsoft Windows 2000(Intel)	SP4	Tested
Microsoft Windows XP (Intel)	SP1, SP2	Tested
Microsoft Windows Vista		Testing
RedHat Linux (x86)	9.0J	Tested
Red Hat Enterprise Linux (x86)	3	Tested
Turbo Linux (x86)	10	Testing
Fedora Core	3, 5, 7	Tested
Fedora Core x86_64	5	Tested (32bit mode)
MacOSX Tiger	10.4.7	Intel Core Duo 2.16GHz Power PC G5 Dual 2.5GHz Tested
MacOSX Panther	10.3.9	Power PC G4 1GHz Tested

Table 2: Operating System

A.2 Java System

Java System	Version	Comments
SUN Java2 SE/RE	1.4.2	Tested
SUN Java2 SE/RE	1.5.0(5.0), 6	Tested

Table 3: Java System

A.3 Graphics System

Graphics System	Version	Comments
OpenGL	Mesa 1.3 or newer	Mesa 4.0.4 or newer
JOGL JSR-231	1.0.0	Tested
JOGL JSR-231	1.1.0	Windows Tested Linux Depends on Linux kernel version

Table 4: Graphics System

A.4 Python System

Python System	Version	Comments
Python	1.5.2、2.1.x、2.2.x	Tested
Python	2.3.x, 2.4.x	Tested
Scientific Python	Depends on Python version	Tested
Numerical Python	Depends on Python version	Tested

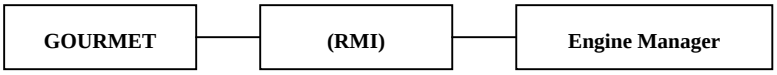
Table 5: Python System

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

Appendix B Security policy of Engine Manager

EngineManager works as a server that responds to a demand from its client application GOURMET.



Its communication path is Java RMI (Remote Method Invocation), which starts or pauses engine, and transfers UDF files. Security checking features (authentication system and file access permission) are not implemented.

These services are designed so that priority is given to execution speed. It expects that two or more users will not use the same resource (the same directory in the same server at the same time) simultaneously.

So the following cases may lead to errors,

Multiple clients take action in the same directory.

Delete or edit any file to which an engine accesses.

Once GOURMET and EngineManager connected, that engine will not accept any demands from the other Gourmet. Yet, an error could happen to its files and directories because they are open to Gourmet.

Important:

Engine manager does not authenticate clients.

Engine manager does not limit executable modules.

To avoid such security risks, the following actions could be effective.

System administrator prepers user ID who can start EngineManager, and who has authority of the necessary minimum.

If engines and Gourmet are used in a local machine, use STAND ALONE MANAGER option. With this option, (stand alone) EngineManager started by eng_man_alone cannot be accessed from other machines.

If EngineManager starts on Microsoft Windows, you can avoid making connection from other machines by using the filter of the communication port that EngineManager uses.

In order to perform this, you use the tool of OS attachment or the product for security.

Appendix C Action definition

C.1 Grammar of Action file

Below is the grammatical rule of the action file.

comments

The line is starts with '#', the whole line is comment line.

import

Import statement is the same as Python import one.

action

Action keyword is placed at the top of action statements, whose next words is a target data name of the action. If the target data name is omitted, the target of the action statement is the UDF file.

autorun

Autorun keyword defines actions executed when UDF file is opened or reloaded. But the orders of executing actions defined with autorun are not defined.

target

Target is a UDF data name in an action statement. It is a trigger of executing action. If the UDF data name displayed in the editor or viewer is clicked, an action is executed or actions selection list is displayed.

action name

Action name is displayed on an actions selection list in the pop-up dialog, when target is clicked.

action parameters

Action parameters are the arguments of the action. With action parameters, input areas are displayed on an action dialog.

If a parameter is string type and divided by '|', a combo box is displayed on an action dialog. If a parameter is string type and '…', a file selection dialog is displayed by right clicking the […] in an action dialog.

Table 6 shows BNF（Backus Naur Form）of action statements. Boldface words are action keywords. UPPERCASE words are tokens.

Name of a syntax rule (non-terminal symbol): Components of a syntax rule

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_DATA_NAME
        target, UDF_DATA_NAME
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
```

Table 6: BNF of action statements

C.2 Special words in python statement

The following words have special meanings of Python statements in an action.

parameter name

A parameter is replaced by an argument value. In example 1, name is replaced by "time" or an input value.

self

self is replaced by a target name. In example 1, ‘self’ is replaced by ‘Calculated_results.time’.

Example 1 (target:Calculated_results.time、action name:add_trajectory)

Example 1:

```
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
```

If there are two or more target data names, keywords "self1", "self2", … are replaced by target names respectively. In viewer in the mode of multiple picking, you first select

Set_of_Molecules.molecule[0].atom[1],

and next

Set_of_Molecules.molecule[0].atom[5],

in following example 2,

self1 is replaced to Set_of_Molecules.molecule[0].atom[1],

self2 is replaced to Set_of_Molecules.molecule[0].atom[5].

Example 2: Sample for multiple picking action

```
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
```

IMPORTANT:

Action parser is replaced all matching words ("self[0-9]*" and parameter) in no conditions. Therefore you carefully used the words （"self[0-9]*"） in the actions.

C.3 Example of Action file

Example: from tutorial,

```
# 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 Connection with UDF file

If action file names are written in the header part, Gourmet will load the action files.

(Sample of action item description in UDF file)

```
\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}
```

Action search directory is the same directory of UDF file at first; the next is the directories specified by the environment variable UDF_ACTION_PATH.

The default directories of UDF_ACTION_PATH are the followings:

the directory specified by environment variable FP_FILES + "action",

the directory specified by environment variable PF_ENGINE + "action".

Appendix D File converter

D.1 What is File converter

File converter is the tool to convert other format data file into UDF file. It needs the special filter rule file in the converting process. This section describes about the grammar of filter rule file.

D.2 Grammar of Filter Rule

COMMENTS

If the first character of a line is '#', the whole line is a comment.

PRE-OPERATION

If the first word is ‘BEGIN’, specified python function will be executed before filtering operation. It is the same function as AWK.

POST-OPERATION

If the first word is ‘END’, specified python function will be executed after filtering operation. It is the same function as AWK.

NUMBER

If the first word is a number, specified number of lines will be skipped from the top.

CONTROL RULE

If the first word is 'CONTROL', the following variables are control data that specify how many lines of data are read. The numerical values described in the data file to be converted are read into the control variables.

VARIABLE

If the first word is the variable name that specified in the control rule, the same number of lines as the value of the variable is read from the data file.

LABEL

If the first word is 'LABEL', the line specifies a search condition and substitution conditions. The second word is the regular expression used by sed, perl, etc. The third word and what follows are the variable names to be substituted.

FIELD LENGTH

The number enclosed by ( ) is a column number of data to read. When a umber is omitted or zero is specified, it reads as a free format divided by white space.

Table 7 shows BNF（Backus Naur Form）of filter rule statements. Boldface words are action keywords. UPPERCASE words are tokens. And “_NDATA” is the counter of reading data at the time.

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

Table 7: BNF expressions of filter rule statements

D.3 Example of File Filter

Molecule builder uses molfile filter and PDB file filter. These filters convert from source data file to the following UDF definition file.
(Note) In the following samples, “(CONTINUE)” shows that a line continues the next line, so it is not a real data.

```
\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}
```

Example 1: molfile filter

Molfile filter converts from a molfile formatted data file to the above UDF definition file by the following rule file.

```
# 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() (CONTINUE)
                                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
```

Example 2: PDB file filter

PDB file filter converts from a PDB formatted data file to the above UDF definition file by the following rule file.

```
# example rule for convert molecule data from pdb file
LABEL ^ATOM..(6) atoms[].Atom_ID(5) (1) atoms[].Atom_Type_Name(4) (6) (4) (CONTINUE)
(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) (CONTINUE)
(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
```

Example 3: NASTRAN bulk file filter

NASTRAN bulkfilefilter converts from a NASTRAN bulk file to the following UDF definition file by the following rule file. (only using GRID, CTRIA3, CTETRA labels)

```
# 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)  (Continues to the next line)
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) (Continues to the next line)
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) (Continues to the next line)
v0(8) v1(8) v2(8) v3(8) mesh.data.cell[].vertex[]=[v0,v1,v2,v3]
```

Appendix E Special converter tools

E.1 NASTRAN Data Converter Tool

NASTRAN data converter tool is a set of utilities, which loads a NASTRAN bulk data file to FEM UDF file, extract surface elements and draw FEM data. It loads NASTRAN bulk data file to FEM UDF file, and processes drawing and surface extraction.

The FEM UDF definition file (fem_def.udf), the action files and the Python script files are located in the following directory:

GOURMET_20XX/tool/NASTRAN

How to use

(1) Start-up

Start GOURMET and load the FEM UDF definition file (fem_def.udf). Save as another file name without changing the UDF definition file. Be careful not to overwrite UDF definition file.

(2) Reading and converting a NASTRAN bulk data file.

Click the UDF file name in the editor view, so action list is displayed, and select [ReadBulk].

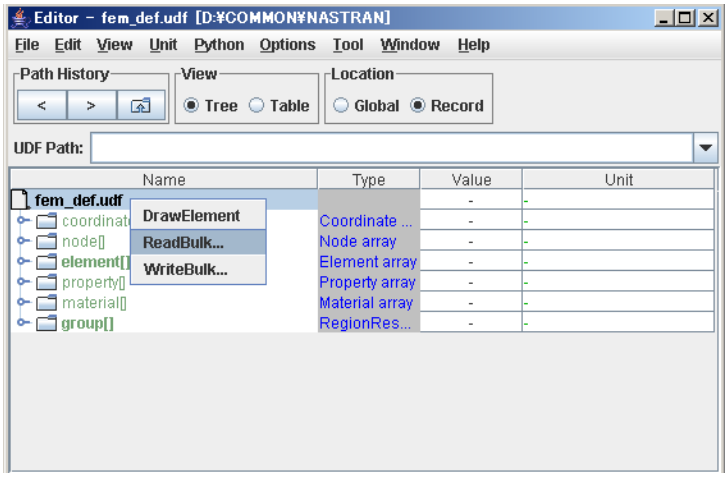


Figure 50: Three actions to read NASTRAN bulk file

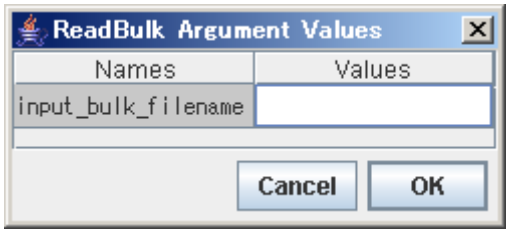


Figure 51: Specifying a NASTRAN bulk file

Click on the column under Values, and choose a NASTRAN bulk file that you want to read and click OK. When you load another file, the prior data is deleted.

(3) Drawing an element

To draw an element directly, click on “element[]” on Editor, and choose [DrawElement] from the action list .

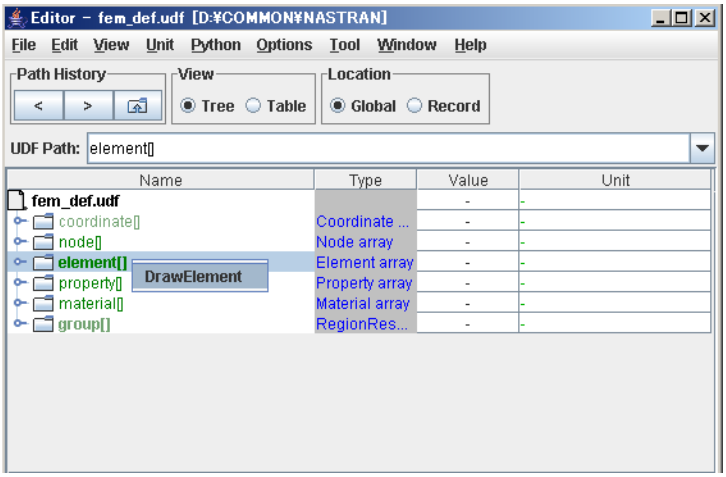


Figure 52: Choosing Actions for drawing an element

(4) Extracting partial region

Click on “group[]”, and choose [CreateRegion] from the action list.

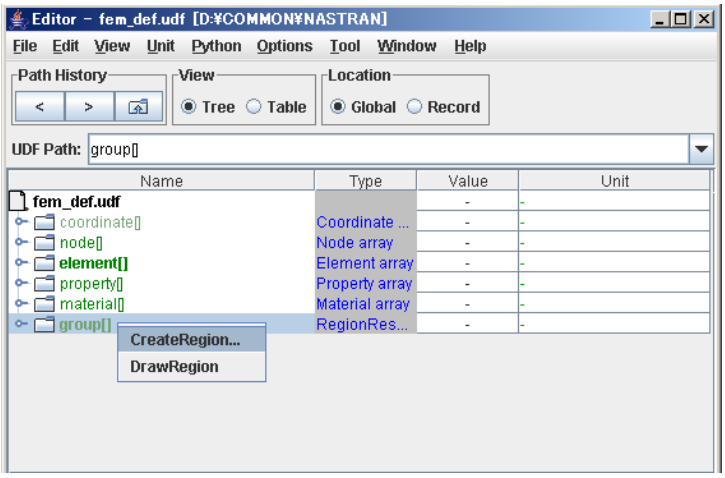


Figure 53: Choosing an action to extracting partial region

You have three alternatives “angle", "property", and "angle_property", as a method of extracting partial region. "property" split region using element property only, while “angle_property” does using both angle of normal vector and element property. To split area by normal vector angle (“angle”), input angle value (°) and press OK.

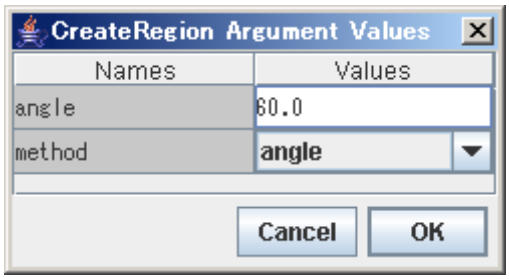


Figure 54: Specifying extraction condition for partial split

If you expand "group[]" in Editor, you will find newly-created "group[0]", and partial region created under group[0].region[].

Now click on “group[]” and choose [CreateRegion]. This action processes partial region extracting. "group[1]" is newly created, and extracted data is added in it.

Click on "group[0]" with an index attached, and process extracting partial region. "group[0]" is revised for the latest extracted data.

By partial region extraction for 2D element, a set of polygons is formed as a closed boundary edges.

（５）Drawing partial region

Click on "group[0]" in Editor, and choose [DrawRegion] from the action list. All partial regions are drawn by polygons. All regions are drawn in minimum different colors.

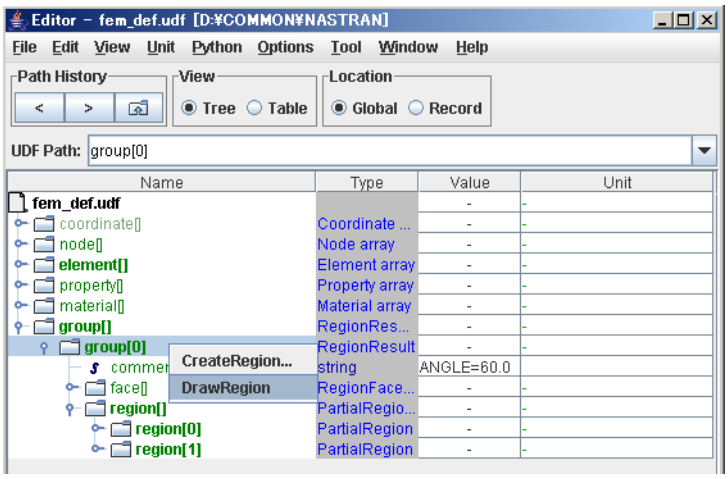


Figure 55: Drawing action for all partial regions

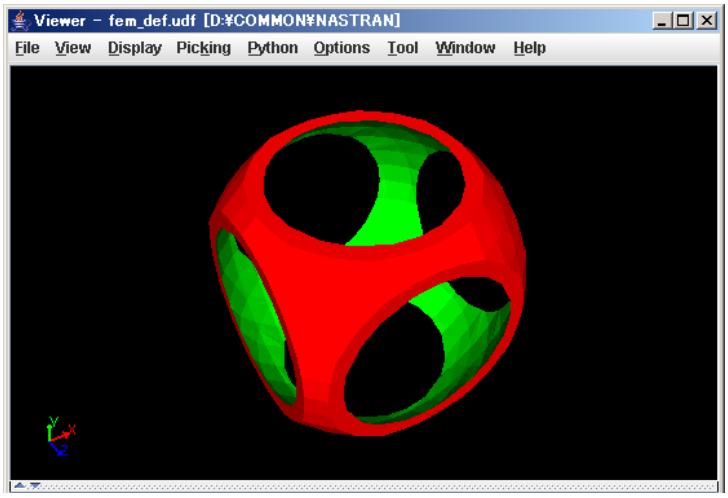


Figure 56: Drawing result for all partial regions (whole image)

If you click on some "group[0].region[N]" in Editor, and choose [DrawRegion] from the action list. If you click on “region[N]”, such as “group[1].region[5]" and draw in Editor, only the selected region[N] is drawn in surface, and the rest is drawn in line.

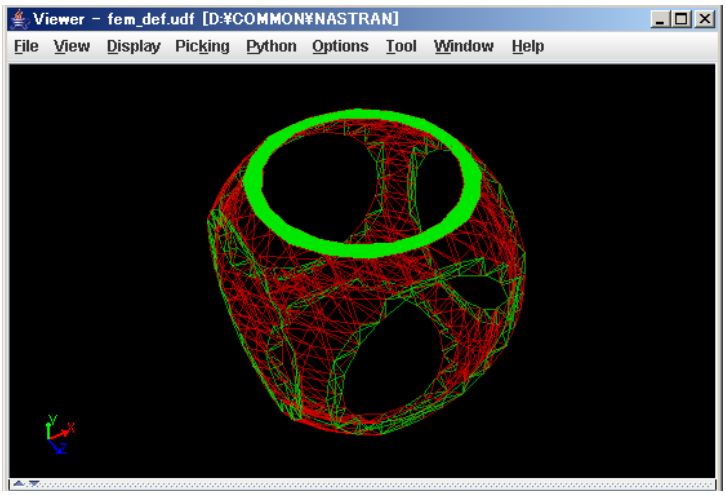


Figure 57: Drawing result for a partial region (partial)

（６）Conversion to NASTRAN bulk file.

Click on UDF file name, and choose [WriteBulk] from the action list.

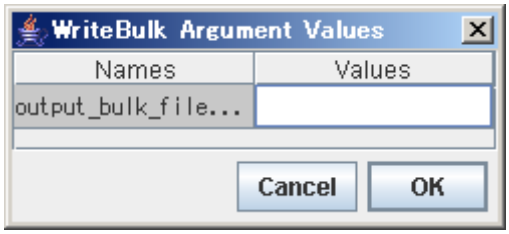


Figure 58: Writing NASTRAN bulk file.

In the above dialog, right-click on the input area below "Values" to open a file selection dialog. Next, go to the directory where you want to save the file, type file name, close file selection dialog, then click OK.

Description of UDF

Followings are the top-levele data structure and its description.

coordinate[]:Coordinate	Coordinate system data
node[]:Node	Node dada
element[]:Element	Element data
property[]:Property	Element property data

material[]:Material Property data

group[]:RegionResult Extracting partially-splitted area.

Followings are the low level data structure.

Corrdinate system data structure

```
class Coordinate:{
  id:ID
  system:{
    type:select{"NodeCartesian","NodeCylindrical","NodeSpherical",
                "PointCartesian","PointCylindrical","PointSpherical"}
    NodeCartesian:{                    "rectangular coordinate system by 3-node_id"
      node1_id:<Node,ID>
      node2_id:<Node,ID>
      node3_id:<Node,ID>
    } "CORD1R"
    NodeCylindrical:{                  "cylindrical coordinate system by 3-node_id"
      node1_id:<Node,ID>
      node2_id:<Node,ID>
      node3_id:<Node,ID>
    } "CORD1C"
    NodeSpherical:{                    "spherical coordinate system by 3-node_id"
      node1_id:<Node,ID>
      node2_id:<Node,ID>
      node3_id:<Node,ID>
    } "CORD1S"
    PointCartesian:{                   "rectangular coordinate system by 3-positions"
      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:{                 "cylindrical coordinate system by 3-positions"
      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:{                   "spherical coordinate system by 3-positions"
      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"
  }
}
```

Followings are the definitions of coordinate system types.

"NodeCartesian" : Defines a rectangular coordinate system using three grid points IDs in correspondence with CORD1R record in the NASTRAN file format.

"NodeCylindrical" : Defines a cylindrical coordinate system using three grid points IDs in correspondence with CORD1C record in the NASTRAN file format.

"NodeSpherical" : Defines a spherical coordinate system by reference to three grid points IDs in correspondence with CORD1S record in the NASTRAN file format.

"PointCartesian" : Defines a rectangular coordinate system using the coordinates of three points in correspondence with CORD2R record in the NASTRAN file format.

"PointCylindrical" : Defines a cylindrical coordinate system using the coordinates of three points in correspondence with CORD2C record in the NASTRAN file format.

"PointSpherical" : Defines a spherical coordinate system using the coordinates of three points in correspondence with CORD2S record in the NASTRAN file format.

Node data structure

```
class Node:{
  id:ID                                          "Node ID"
  coordinate_id:int "<Coordinate,ID>"            "Coordinate system ID"
  position:{                                     "Node coordinates"
    x1:double
    x2:double
    x3:double
  }
}
```

Element data structure

```
class Element:{
  type:select{"tri","quad","tetra","penta","hexa"}   "Element type"
  id:ID                                              "Element ID"
  pid:<Property,ID>                                  "Property ID"
  node[]:<Node,ID>                                   "Array of Node ID"
}
```

【Explanation】

Below is the sequence of nodes specification in accordance with NASTRAN.

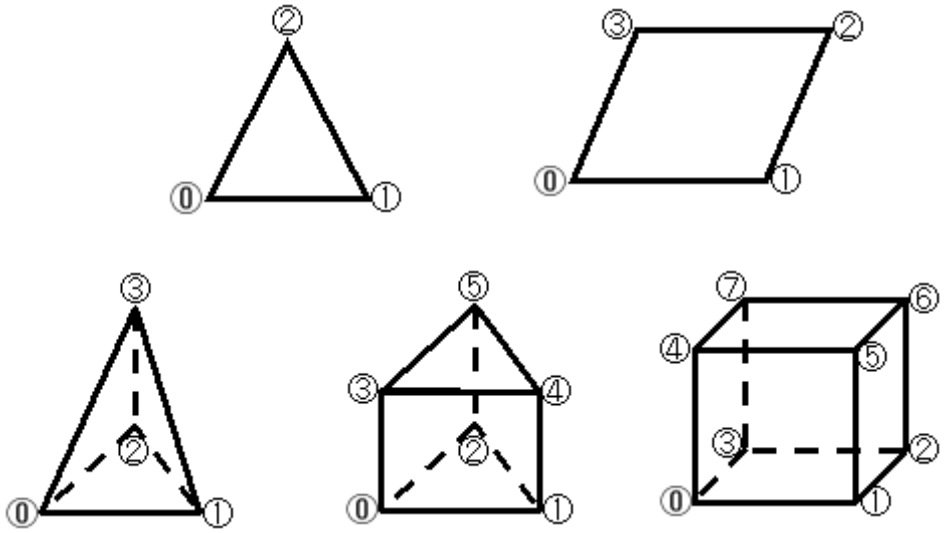


Figure 59: Sequence of nodes specification

Property data structure

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

Material data structure

```
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
}
```

Data structure of partially-extracted area processing

```
class RegionSideElement:{
 element_id:<Element,ID>          "Original element ID"
 side_id:int                      "Side ID for an original element (See [Description] below.)
}
class RegionFace:{
 id:ID                            "ID of an aspect or a side"
 node_id[]:<Node,ID>              "Array of node ID"
 element[]:RegionSideElement      "Original element data"
}
class PartialRegion:{
 color_id:int                     "Color IDs for the minimum number of  colors to color-code the
 partial area"
 face_id[]:<RegionFace,ID>        "Array of IDs for faces or edges forming partial region"
}
class RegionResult:{
 comment:string
 face[]:RegionFace                "Array of data for faces or edges"
 region[]:PartialRegion           "Array of data for partial region"
}
```

[Description]

color_id is a color index that finds the minimum number of colors to color-code the whole image.

side_id is an index that specifies the side of an element. For example, in a tetrahedral element, a triangle for side_id=0 is specified by node {1,2,3}.

Triangle element

side_id：[0]={1,2}, [1]={2,0}, [2]={1,2}

Quadrangular element

side_id：[0]={0,1}, [1]={1,2}, [2]={2,3}, [3]={3,0}

Tetrahedral element

side_id：[0]={1,2,3}, [1]={0,3,2}, [2]={0,1,3}, [3]={0,2,1}

Pentahedron element

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}

Hexahedron element

[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 }

Appendix F

3D Displacement of Drawing Targets (Translation/Rotation)

For a specific drawing target (Ver.4.0 covers point, line, ball and cylinder) in 3D drawing screen, now you have new functions of changing direct drawing coordinates. Below is the explanation.

F.1 Getting Ready

To execute 3D translation or rotation, you need to have two kinds of actions and drawing targets related UDF data.

The two actions are:

（１）Select action ($Select_$Translate(), $Select_$Rotate())

Action script for selecting drawing targets.

（２）Return action ($Translate(), $Rotate())

Return the destination coordinates to UDF data.

These two action examples are in GOURMET_20XX/action/cognac/cognac_transform.act .

The following 3D translation or rotation is possible if you insert this action script to COGNAC engine drawing action file.

F.2 Operating Procedure

Before operating, copy the 3D translation and rotation sample action (cognac_transform.act) at the end of the drawing action file (cognac_draw.act) of COGNAC engine.

Translation

Below is the translation operation procedure using COGNAC engine UDF.

(1) Open a UDF file of COGNAC engine, choose "show" action, and draw by "ball-stick".
(2) Choose an atom within a target molecule.
(3) Choose “$Translate_MOL” from Action selection menu. (Figure 60）
(4) Once the above Select Action is executed, a target molecule is selected, and parallel translation operation dialog is displayed. (Figure 61)
(5) parallel translation operation dialog has two screens. One is "Direction" tab screen, whose "Up"/"Down"/"Left"/"Right" button translates the current view in parallel toward the selected destination for the specified “Length". The other is “Vector" tab screen, where you input translation vector amount.

    While parallel translation operation dialog is open, drag the drawing screen and the target moves in the same direction as a mouse moves.

    1 pixel of mouse movement distance on drawing screen is the tenth of the movement distance by button operation.
(6) In "Direction" tab screen, the target moves in parallel as specified by each direction button. In "Vector" tab screen, it moves in parallel for the vector amount you input.
(7) Click OK, and parallel translation operation is settled in the latest drawing status. Return Action is executed and destination coordinate is returned to UDF data.

General eye sight rotation, movement and zoom are operable during transfer operation.

Rotation

Below is the rotating operation procedure using COGNAC engine UDF file.
(1) Open a UDF file of COGNAC engine, and draw by "ball-stick".
(2) Choose multi-picking mode from Picking menu on the draw view.
(3) Choose two atoms as an axis of rotation.
(4) Choose “$Rotate” from the action menu when you choose the second atom. (Figure 62)
(5) The above “Select Action” is executed, an atom and bond on the second atom side is selected, and rotation operation dialog is displayed. (Figure 63)
(6) In rotating operation dialog, rotation central coordinate and rotation axis vector are indicated. Rotation central coordinate is the first atom axis, and rotation axis vector points at the second atom from the first atom. (Figure 64)
(7) You can edit rotation central coordinate, rotation axis vector, and rotation degrees of angle.
(8) Input rotation degree, press arrow button, and an object is drawn based on the post rotation.
(9) Press OK, and rotating operation is settled as the last drawing status, and the above “Return Action” is executed.

Press Cancel on rotating operation dialog, and the axis returns to the initial status.

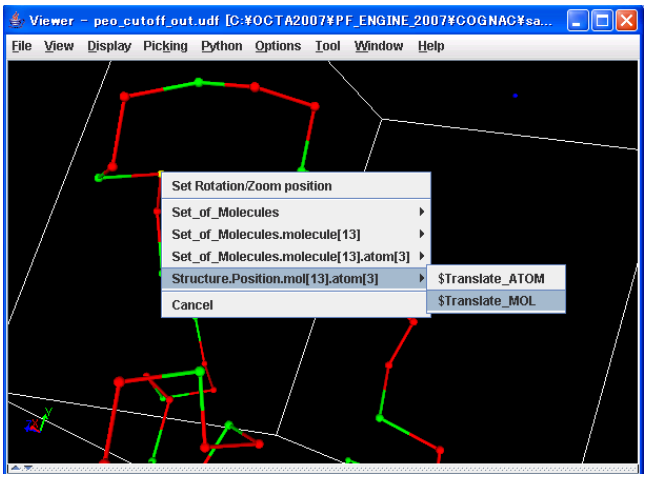


Figure 60: Choosing Parallel Translation Action

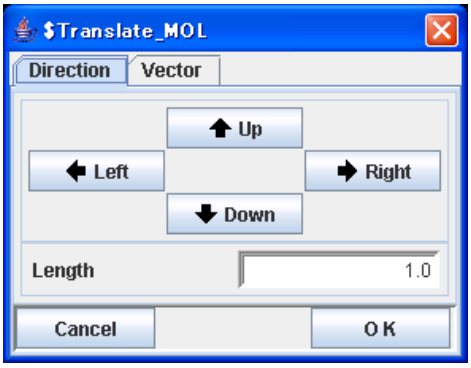


Figure 61: Parallel Translation operation Dialog

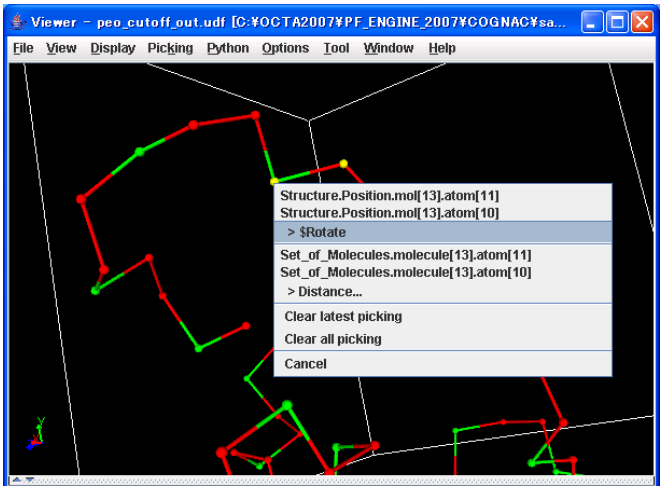


Figure 62: Choosing Rotation Action

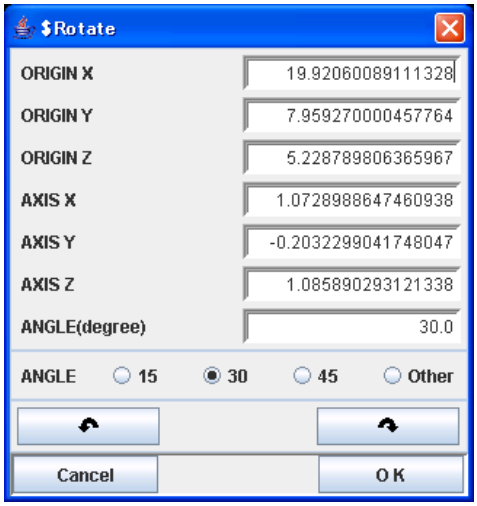


Figure 63: Rotation Operation Dialog

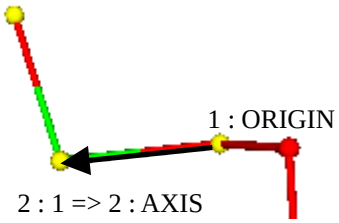


Figure 64: Rotation Center and Rotation Axis Vector

Appendix G

Environment Variables Production Tool

Environment variable production tool (gourmet_init) is an executable module coded by C/C++. The tool is located at %PF_FILES%/bin/%ARCH% directory, and use the environment initial file (platform_win32.ini in Windows and platform_linux.ini in Linux) as user input data.
The tool returns the following directory paths depending on the optional arguments.

Optional arguments of environment variables are as follows.

arch :	Return machine architecture name (Returned win32 or lower-case character of "uname -s" command, or what you specified by -a.)
python.home :	Return Python home directory.
python.path :	Return Python paths.
action.path :	Return action paths.
udf.path :	Return UDF paths.
java.home :	Return a Java home directory.
java.java :	Return a Java execution module path.
user.path :	Return execution paths.
library.path :	Return library paths.
-a :	Specify your machine architecture name.
-f :	Specify your initial environment file name. (ex. "platform_win32.ini")
-j :	Specify your Java home directory, or search paths for Java home directory. For example, if you give C:\OCTA20XX\GOURMET_20XX\bin\win32\jre1.5 to the tool, you will receive a real path C:\OCTA20XX\GOURMET_20XX\bin\win32\jre1.5.0_12 .
-u :	Give a file path, the tool returns the parent directory path.

No optional argument : All the returned values other than the above –options are listed.

Below is an example of executing environment variable tool without argument.

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:\OCTA20XX\PF_ENGINE_20XX\python;C:\OCTA20XX\PF_ENGINE_20XX\lib\win32
action.path=C:\OCTA20XX\GOURMET_20XX\action;C:\OCTA20XX\PF_ENGINE_20XX\actionudf.
path=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\win
32
library.path=C:\OCTA20XX\GOURMET_20XX\lib\win32;C:\OCTA20XX\PF_ENGINE_20XX\lib\wi
n32
```

Environment variable production tool returns real paths only. It can also replace the following key words in the environment initial file.

%ARCH%,${ARCH} : Architecture name

(Returned win32 in Windows or lower-case character name of “uname –s” command in Unix/Linux, or what you specified by -a option.)

%PF_FILES%,${PF_FILES} : PF_FILES environment variable (GOURMET installation directory path)

%PF_ENGINE%,${PF_ENGINE} : PF_ENGINE environment variable (installation directory path of OCTA engines).

%OCTA_HOME%,${OCTA_HOME}: OCTA_XXXX_HOME environment variable (OCTA installation directory path)

Appendix H Trouble Shooting

H.1 Python Tool

I can't start Python Shell from Tool/Python menu.

- Check “Python Application:” and “Python argument:” text fields in the Application Setup Dialog, which is displayed from “Tool/Application Setup…” menu. You need to set the input field of “Python Application:” to the Python module path (pythonw.exe for Windows) and the input field of “Python argument:” to Python IDLE (Tools\idle\idle.pyw for Windows).

H.2 Drawing 3D Objects

I can’t draw 3D objects.

- Make sure that JOGL JSR-231 v. 1.0.0 or later is installed on your PC.
- If your PC is a very new model, update the newest display driver.

In Linux x86_64, GOURMET can start but cannot draw 3D objects.

- Check 32bit GL library in your x86_64 Linux. If there is not /usr/lib/libGL.so.1, you need to install 32bit GL and GLU library in your Linux.

H.3 Plotting Graphs

I can't start gnuplot from Tool/Gnuplot menu.

- Check “Gnuplot Application:” text fields in the Application Setup Dialog, which is displayed from “Tool/Application Setup…” menu. You need to set the input field of “Gnuplot Application:” to the gnuplot module path (wgnuplot.exe for Windows).

H.4 Editing Values

I can’t copy and paste between GOURMET and another application, in Linux.

This is solved after Java 1.4.

I can’t select two or more cells in Table editing view after GOURMET v 4.0.

- The method of selecting multiple cells has been changed by Java upgrade. If you select two or more cells, after selecting the first cell, and select the second cells by pressing [shift] key. This way, you can select all cells within the rectangular region between the first selected cell and the second