OCTA

Integrated simulation system for soft materials

Multi-Phase Dynamics Program

Muffin

version 5.1

User's Manual

- Volume I -

General Functions, I/O and Tools

OCTA User's Group

January 01 2016

Authors of the Manual

Introduction	Tatsuya Yamaue
Muffin	Tatsuya Yamaue and Takashi Taniguchi
Milk and Tools	Tatsuya Yamaue and Makoto Sasaki
Appendix	Tatsuya Yamaue

Programers

Muffin	Tatsuya Yamaue and Takashi Taniguchi
Milk	Tatsuya Yamaue and Akiyoshi Kuroda
Python Tools	Tatsuya Yamaue and Makoto Sasaki

Version 3.3 release

Programer, Authors of the Manual Tatsuya Yamaue

Version 4.0 release

Programer, Authors of the Manual Tatsuya Yamaue

Version 4.1 release

Programer, Authors of the Manual Tatsuya Yamaue

Version 5.0 release

Programer, Authors of the Manual Tatsuya Yamaue, Taku Ozawa

Version 5.1 release

Programer, Authors of the Manual Taku Ozawa

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

Introduction

This manual explains about the function and input-and-output data dealing with

Multi-Phase Dynamics

of a simulator

Muffin
(MUltiFarious Field Simulator for Non-equilibrium system)

(with the following ***Muffin***).

Muffin is a general solver for the continuum models for the multi-phase dynamics of soft materials on finite difference method (FDM) or finite element method (FEM). Muffin includes the following six packages (three fluid system packages, two solid system packages and one optics package).

1. **PhaseSeparation**. . . . Multi-Fluid Phase Dynamics Simulator (FDM and FEM).
2. **Electrolyte**. . . . Electrolyte Fluid Dynamics Simulator (FDM and FEM).
3. **MEMFluid**. . . . Micro Electro Chemical Fluidics Chip Simulator (FEM).
4. **Elastica**. . . . Linear Elasticity Simulator.
5. **Geldyn**. . . . Gel and Elasticity Dynamics Simulator.
6. **Turban**. . . . Light Transmittance Simulator.

They can deal with various problems in soft materials, such as the elasticity of multi-phase materials, swelling and deswelling of gels, ion transport in charged colloids, reaction and diffusion in narrow channels, and phase separation and droplets deformation in shear and electric field. Muffin can take the multi-phase structure obtained by SUSHI and can calculate various properties such as the effective elastic modules of the system.

Moreover, to provide a flexible environment for combining various functions and for future development, Muffin is designed by the object oriented modeling, which enables **DynamicsManager** to combine, cooperate and analyze the various fields with various solvers.

The structure of this manual is as follows.

In volume I , we explain basic functions, such as the starting method of Muffin, input and output UDF, and the dynamics manager.

In volume II-VII, we explain the function of each package, the usage, and the example of application in detail. In volume II, Multi-Fluid Phase Dynamics Simulator package named "PhaseSeparation", in volume III, Electrolyte Fluid Dynamics Simulator package named "Electrolyte", in volume IV, Micro Electro Chemical Fluidics Chip Simulator package named "MEMFluid", in volume V, Linear Elasticity Simulator package named "Elastica", in volume VI, Gel and Elasticity Dynamics Simulator package named "Geldyn", and in volume VII, Light Transmittance Simulator package named "Turban".

In volume I, we explain the general pre, post and analysis tools (support tools) for Muffin. The support tools consist of two parts, a pre processor (a mesh generation program "Milk", Modeler for "Elastica" and "Milk", the conversion program of the finite element data of NASTRAN form into that of UDF form and initial value generation program) and a post processor (an analysis program, a drawing program, Plot program).

Appendix A describes the compile method for reconstructing the module of Muffin. In appendix B, the source codes of C++ class libraries for the meshing, the dynamics manager and so on are described. It also explains how a user can add a new function to Muffin.

If you use the Muffin base class libraries, various simulations (the finite difference method or the finite element method using an unstructured or a structured mesh) can be performed. You can also deal with an input and output data on GOURMET automatically. Moreover, the users can also use the common analysis tools for all the output data in Muffin.

A tutorial manual for adding your own simulator to Muffin, which is given in the Appendix B, allows you to develop your simulator using the C++ class libraries of Muffin.

We wish that Muffin finally becomes the tool which produces a meaningful, completely new results in research of polymer science and soft condensed matter physics. Furthermore, we hope that Muffin will be developed continuously in the future, and become the essentially innovative simulation software in the near future.

22 Feb. 2002, Muffin development team.

Comment for version 3.3 release

Merry Christmas !!
Please enjoy your simulation using Muffin version 3.3.

25 Dec. 2002, Muffin development team.

Comment for version 4.0 release

In Muffin4, we add the new action programs, "import_picture" and "porous_generator", in which we wish the advance of the collaboration of the OCTA simulation and experiment and the expansion of users and application of Muffin. Please enjoy your work using Muffin version4.0.

25 Dec. 2003, Muffin development team.

Comment for version 4.1 release

In Muffin4.1, we add new functions to "Elastica" and "Milk", new action programs related to "Elastica" and "Milk" modeler and engine run and lots of new samples of "Elastica" and "Milk", which make easy to build FEM model, run and analyze FEM solvers of Muffin. Please enjoy your work using Muffin version4.1.

03 Mar. 2003, Muffin development team.

Comment for version 5.0 release

In Muffin5.0, we unified "PhaseSeparationFDM" and "ElectrolyteFDM" into one package (Sep.2006). and added new functions to "PhaseSeparation(FDM)". Manuals of "PhaseSeparation" and "Elastica" were revised. Please enjoy your work using Muffin version5.0.

01 Feb. 2015, Muffin development team.

Comment for version 5.1 release

In Muffin5.1, we added new functions to "PhaseSeparation(FDM)". Especially, diffusion (e.g.heat conduction) simulation on the phase separated structure is available. Action tools of "PhaseSeparation(FDM)" and "Elastica" were revised. Please enjoy your work using Muffin version5.1.

01 Jan. 2016, Muffin development team.

Chapter 1

What is Muffin?

Muffin is a general solver for the continuum models for the multi-phase dynamics of soft materials on finite difference method (FDM) or finite element method (FEM). Muffin includes the following six packages, **PhaseSeparation**, **Electrolyte**, **MEMFluid**, **Elastica**, **Geldyn** and **Turban** , and can deal with various problems in soft materials, such as the elasticity of multi-phase materials, swelling and deswelling of gels, ion transport in charged colloids, reaction and diffusion in narrow channels, and phase separation and droplets deformation in shear and electric field. Muffin can take the multi-phase structure obtained by SUSHI and can calculate various properties such as the effective elastic modules of the system.

Moreover, in order to provide a flexible environment for combining various functions and for future development, the interface between functions needs to be unified. By the object oriented design, functional separation of fields, solvers, and the dynamics is clearly made for Muffin, and the function (***DynamicsManager***) to freely combine, to cooperate and to analyze the object of fields with various solvers is provided. The list of the analysis function with which Muffin of this release is equipped, and the calculation techniques is shown below.

1. The basic analysis functional list of Muffin
    - Analysis of the flow under various external fields, such as the shear and the electric field.
    - Analysis of multi-phase behavior, such as the coalescence and breakup process of droplets.
    - The chemical reaction of fluid, a diffusion flow, and separation and extraction of reaction components under pressure and electric fields in MEMS (Micro Electro Mechanical Systems), such as micro reactor and micro-TAS (Total Analysis System).
    - Deformation analysis of an elastic body with multi-phase structure in a stress seal of approval and a distortion seal of approval.
    - Deformation dynamics of a gel and an elastic body, with multi-phase structure under the stress, pressure and electric field impression, and temperature and solvent change.
    - Simulation of the spherulites growth and the light transmittance.
2. Basic calculation technique list of Muffin
    - The flow action analysis by the structured lattice, the finite difference method (FDM), and Euler picture.
    - The flow action analysis in arbitrary form by the non-structure lattice, the finite element method (FEM), and Euler picture.
    - The deformation analysis of gels and elastic bodies in arbitrary form by the non-structure lattice, the finite element method (FEM), and Lagrange picture.
    - Dynamics Manager for cooperating and dispelling various fields, solvers, and dynamics.
    - Generation of the non-structure lattice by the Delaunay automatic division.
    - Use of the common UDF of the mesh, field, and boundary conditions, which makes a zooming analysis with SUSHI easy.

In this release, these functions are divided into the following packages. FDM simulators are unified into one package.

1. Multi-Fluid Phase Dynamics Simulator : PhaseSeparation Package
   (a) FDM PhaseSeparation Simulator (muffin5)
   (b) FEM PhaseSeparation Simulator (muffin5e_phaseseparation)
2. Electrolyte Fluid Dynamics Simulator : Electrolyte Package
   (a) FDM Electrolyte Simulator (muffin5)
   (b) FEM Electrolyte Simulator (muffin5e_electrolyte)
3. Micro Electro Chemical Fluidics Chip Simulator : MEMFluid Package (muffin5e_memfluid)
4. Linear Elasticity Simulator : Elastica Package (muffin5e_elastica)
5. Gel and Elasticity Dynamics Simulator : Geldyn Package (muffin5e_geldyn)
6. Light Transmittance Simulator : Turban Package (turban)

These modules are built on common functions of Muffin, such as a field, a mesh, boundary conditions, a dynamics manager, and an UDF IO interface, the operation method, the input-and-output UDF file, and the analysis program.

Chapter 2

Starting Muffin

2.1 Starting guide

2.1.1 The UDF files needed by Muffin

In order to use Muffin, the following UDF files are needed.

- UDF definition file (muffin5.udf or muffin5e.udf) ... The definition of the data structure of input and output UDF.
  It is necessary to put this file on the directory specified by environment variable UDF_DEF_PATH. Although it is already set up when engine is installed according to an installation manual, if you set up it manually, it is necessary to set environment variable UDF_DEF_PATH as the `udf/` directory under the directory (environment variable PF_ENGINE) where engines are installed. When UDF_DEF_PATH was not defined, UDF definition file is necessary to exist in the directory where the execution module exists.
- Input UDF file (inputUDF) ... input data of UDF structure.
- Output UDF file (outputUDF) ... output data of UDF structure.

As options, the following files are also necessary for the real-time display of analysis results and for controlling the engine.

- Analysis-result output UDF definition file (muffin5res.udf) ... The data structure of the analysis-result output UDF is defined.
  It is necessary to put this file on the directory specified by environmental-variable UDF_DEF_PATH.
- Control parameter input UDF file (parameterUDF:muffin5par.udf) ... The UDF file where the changeable parameters and control parameters for summary data during simulation are described.
- Analysis-result output UDF file (summaryUDF) ... The UDF file where analysis-result output data are described by real time.

2.1.2 Starting command

Names of modules are listed below.

- FDM Multi-Fluid Phase Dynamics Simulator ... muffin5
- FEM Multi-Fluid Phase Dynamics Simulator ... muffin5e_phaseseparation
- FDM Electrolyte Fluid Dynamics Simulator ... muffin5
- FEM Electrolyte Fluid Dynamics Simulator ... muffin5e_electrolyte
- Micro Electro Chemical Fluidics Chip Simulator ... muffin5e_memfluid
- Multi-Phase Linear Elasticity Simulator ... muffin5e_elastica

- Multi-Phase Gel Dynamics Simulator . . . muffin5e_geldyn

The staring commands and the data structures of input and output UDFs are common to all the above modules, as the following sections.

How to start using the command line

User can perform the execution module of MUFFIN with some arguments from a command prompt in the windows system or from a shell prompt in the Unix System.

Muffin can be performed from a command line. An argument surrounded by {} is an option. Usually, -I and -O are designated.

% muffin(module name) -I *inputUDF* {-O *outputUDF*} {-r *or* -i *input_record_No.*} {-e *end_record_No.*} {-P *parameterUDF*} {-S *summaryUDF*}

- inputUDF . . . The path of an input UDF file
- outputUDF . . . The path of an output UDF file
- input_record_No. . . . The record number of the UDF data to input. -r is for a FDM engine and -i is for FEM engines.
- end_record_No. . . . The maximum record number of the UDF data.
- parameterUDF . . . The path of the input UDF file of an interactive parameter change command
- summaryUDF . . . The path of the analysis result output UDF file for an interactive surveillance of calculation result
- In the execution of an engine on GOURMET, the control parameter file specified by "-P" option is surely read once in the initial processing. When "RESUME" of the engine (a parameter changed) is carried out, it is read again. The analysis-result data specified by the control parameter file are outputted to the analysis-result output file specified by "-S" option. An analysis result is outputted at intervals of the step of input-parameter "INTERVAL_OF_MONITORING". When the analysis-result data to output are not specified by the control parameter file, all analysis-result data are outputted to an analysis-result output file.
- Batch processing which performs an engine directly from a console can also output analysis-result data using "-S" option. It is only the once in the case of initialization that the control parameter specified by "-P" option is read in the case of batch processing. The analysis-result data specified by the control parameter file are outputted to the analysis-result output file specified by "-S" option at intervals of the step of input-parameter "INTERVAL_OF_MONITORING". When a control parameter file ("-P") was not specified, or when the analysis-result data to output are not specified by the control parameter file, all analysis-result data are outputted to an analysis-result output file.

Chapter 3

Operation guide of Muffin

3.1 Input and output UDF

3.1.1 The outline of input and output UDF

The structure of an input UDF file and an output UDF file is the same in Muffin. The outline of input and output UDF files is explained below.

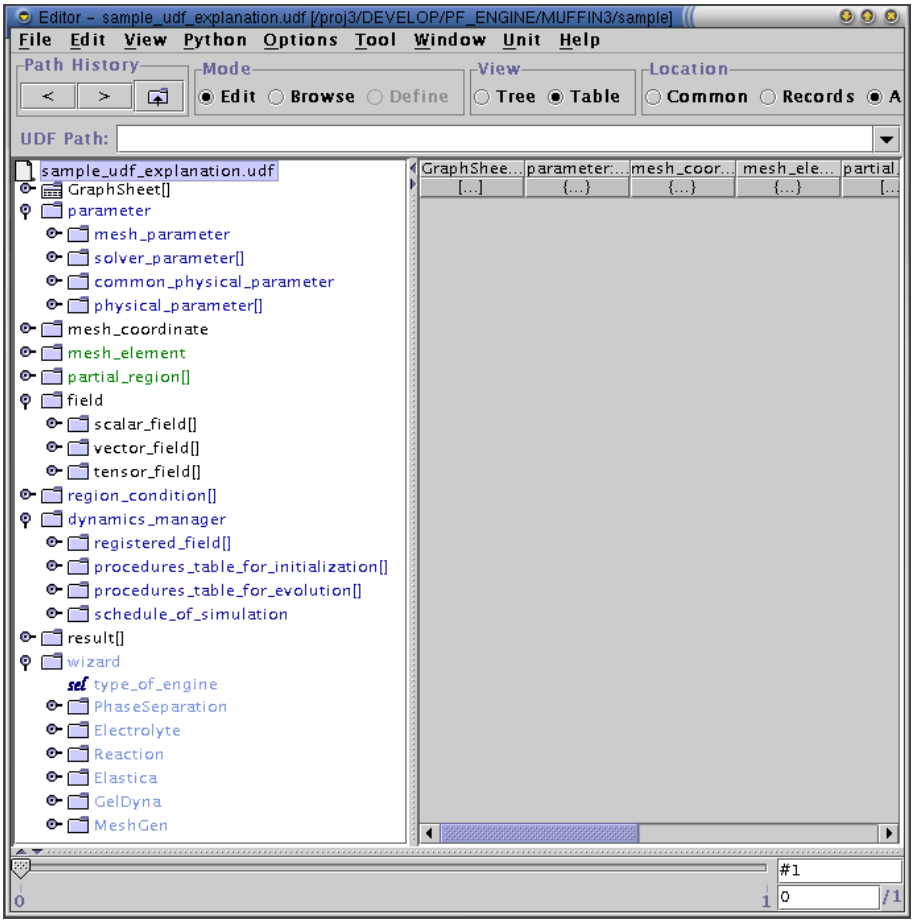

Figure 3.1: The top objects of the input and output UDF of Muffin

As shown in Fig.3.1, the input and output data of **Muffin** have the following structure.

1. Parameter part(parameter)

- Mesh structure parameter (mesh_parameter)
- Solver control parameter (solver_parameter)
- Common physics parameter (common_physical_parameter)
- Physical parameter (physical_parameter)

2. Mesh node coordinates data part (mesh_coordinate)
3. Mesh element connection data part (mesh_element)
4. Partial region data part (partial_region[])
5. Fields data part
   - Scalar field data (scalar_field)
   - Vector field data (vector_field)
   - Tensor field data (tensor_field)
6. Partial region (boundary) conditions data part (region_condition[])
7. A dynamics manager part (dynamics_manager)
   - The list of fields to be used (registered_field)
   - The list of initialization procedures (procedures_table_for_initialization)
   - The list of 1 step execution procedures (procedures_table_for_evolution)
   - Data of the simulation schedule to perform (schedule_of_simulation)
8. Analysis result data part (result[])
9. Modeler part (modeler)

The input and output data for Muffin consist of parameters, mesh node coordinates data, mesh element connection data, partial region data, the data of fields, partial region (boundary) condition data, a dynamics manager, analysis result data, and modeler part.

1. Parameter part (parameter) . . . Initial data
2. (Only FEM) Mesh node coordinates data part (mesh_coordinate)
   - Fluid system simulator (Euler picture) . . . Initial data
   - Solid system simulator (Lagrange picture) . . . Record data
3. (Only FEM) Mesh element connection data part (mesh_element) . . . Initial data
4. (Only FEM) Partial region data part (partial_region[]) . . . Initial data
5. Fields data part(field) . . . Record data
   (When the field data is inputted into the initial data, it will be read as an initial value distribution of a field)
6. Partial region (boundary) conditions data part (region_condition[]) . . . Initial data
7. A dynamics manager part (dynamics_manager) . . . Initial data
8. Analysis result data part (result[]) . . . Record data
9. Modeler part (modeler) . . . Initial data

Below, each data of parameters, mesh node coordinates data, mesh element connection data, partial region data, the data of fields, partial region (boundary) condition data, a dynamics manager, and analysis result data are explained.

3.1.2 Parameters (parameter)

The outline of a parameter part (parameter)

All parameters are inputted from the common data of Input UDF, and are outputted to the common data of Output UDF as it is. The parameter is mainly divided into four parts as follows.

1. Mesh structure parameter (mesh_parameter)
   The value of the parameter group for a mesh creation is given.

2. Solver control parameter (solver_parameter)
   The parameters for tuning up a solver are given.
   (An acceleration factor for the SOR method, a maximum number of the iteration in CG method, etc.)

3. Common physical parameter (common_physical_parameter)
   The parameters used commonly in all the modules of Muffin are given here. For example, they are the time unit "DT", a number of initial steps "INITIAL_STEP", an initial time "INITIAL_TIME", and a number of end steps "FINAL_STEP".

4. Physical parameter (physical_parameter)
   The physical constants used in a simulation or non-dimensional parameters are given.

Mesh structure parameter (parameter.mesh_parameter)

Write all the information about generation of Mesh here.

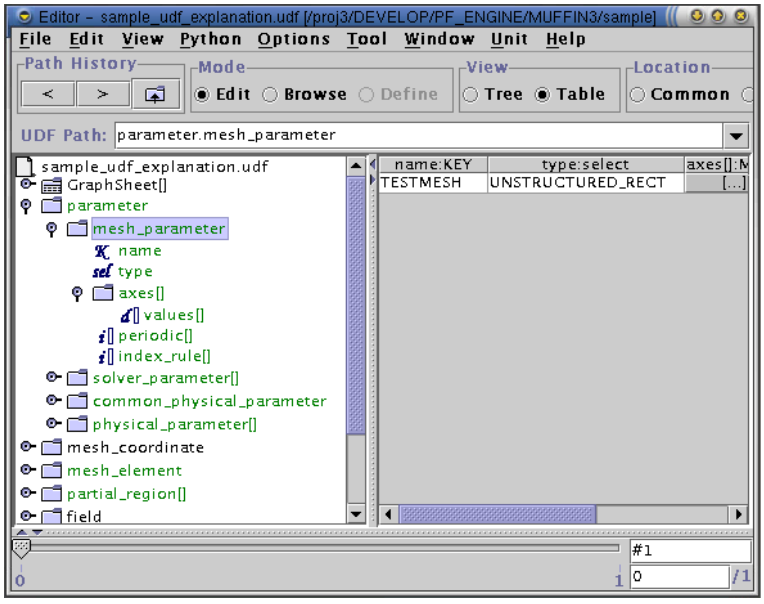

Figure 3.2: Mesh parameter: A mesh type input screen

Fig.3.2 shows the example of an opened mesh_parameter in Editor.

- name : . . . The name of a mesh
  It is useful in using two or more mesh data in a simulation.

- type : . . . The type of a mesh
  As a type of a mesh, the structure lattices of the following four types and the unstructured lattices of four types are supported.

  – REGULAR . . . Regular mesh (for FDM)
  – UNSTRUCTURED_RECT . . . Unstructured mesh (for FEM) (square form and rectangular parallelpiped form)

– UNSTRUCTURED_SPHERE . . . Unstructured mesh (for FEM) (the shape of circle and sphere)
– UNSTRUCTURED_INPUT . . . Unstructured mesh-data input (for FEM)

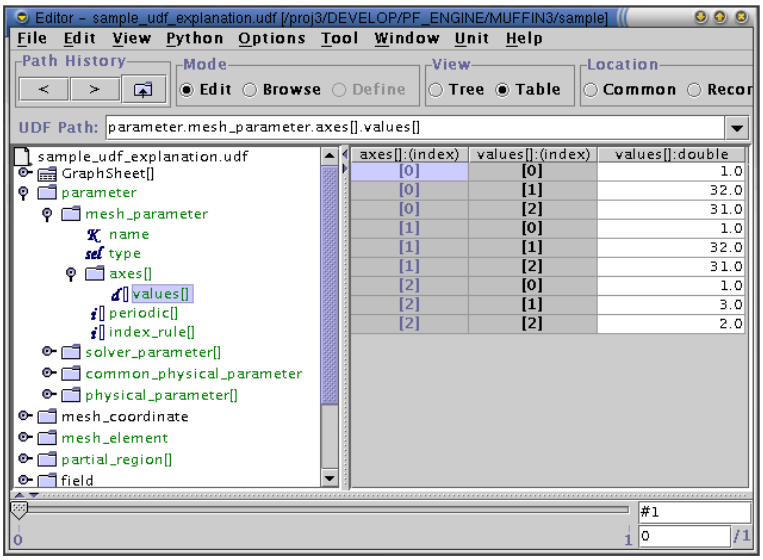

Figure 3.3: Mesh parameter: The input screen of axial data

Fig.3.3 shows the example of axes[].values[] in Editor.

- axes[] : . . . The array of axial data
  The number of elements of the array of axes[] expresses a space dimension. Fig.3.3 is the case of three dimensions and axes [0], axes [1], and axes [2] express X, Y, and the Z-axis, respectively.
- axes[].values[] : . . . The coordinates and the number of division of each axis

  – axes[].values[0] . . . The minimum value of the region of an axis
  – axes[].values[1] . . . Maximum of the region of an axis
  – axes[].values[2] . . . The number of division of an axis
    However, in the case of unstructured lattice data input (UNSTRUCTURED_INPUT), the number of division of an axis is disregarded.
  – unit . . . Unit of coordinates.

  When a mesh type is RECTANGULAR, the array of the coordinates of a mesh point must be inputted.

  Fig.3.3 shows the case of the minimum value 1.0, maximum 2.0, and 15 division for each of X, Y, and Z-axis.

Fig.3.4 shows the example of the periodic[] and the index_rule[] in Editor.

- periodic[] : . . . The array of the periodic boundary condition data of an axis
  1 . . . periodic, 0 . . . non-periodic
  periodic[0], periodic[1], and periodic[2] correspond to the periodicity of the axis of axes[0], axes[1], and axes[2], respectively.

  Fig.3.4 figure shows the case where all of X, Y, and the Z-direction are non-periodic, in the three dimension system.
- index_rule[] : . . . In the case of a structured lattice, the relation between the order of the data array of a field and the surroundings order of an axis is described.

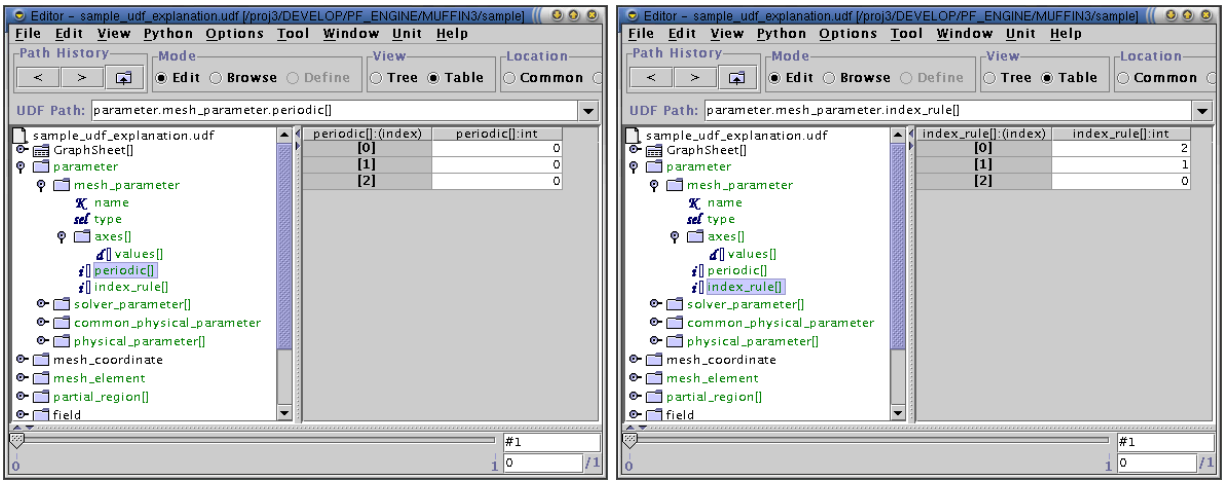

Figure 3.4: Mesh parameter: The input screen of periodic boundary conditions and INDEX_RULE

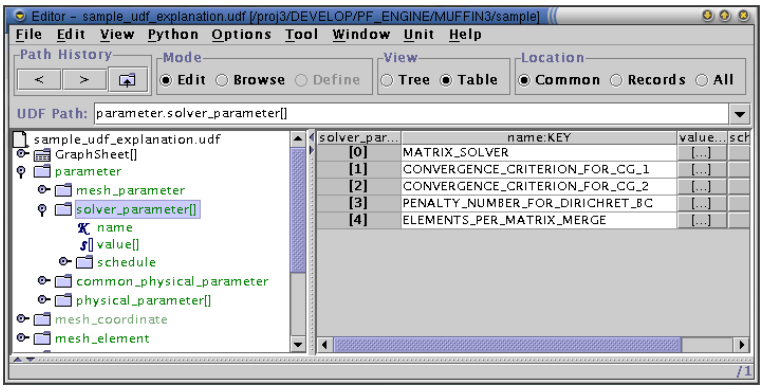

Figure 3.5: Solver control parameter (UDF description)

Solver control parameter (parameter.solver_parameter)

Parameters related to a numerical algorithm such as the value of criterion of convergence, the maximum number of iteration of a using-as release-repeating method case, and the acceleration factor of a SOR solver etc. should be written here.

The solver control parameter in Elastica is shown as an example in Fig.3.5.

Common physical parameter (parameter.common_physical_parameter)

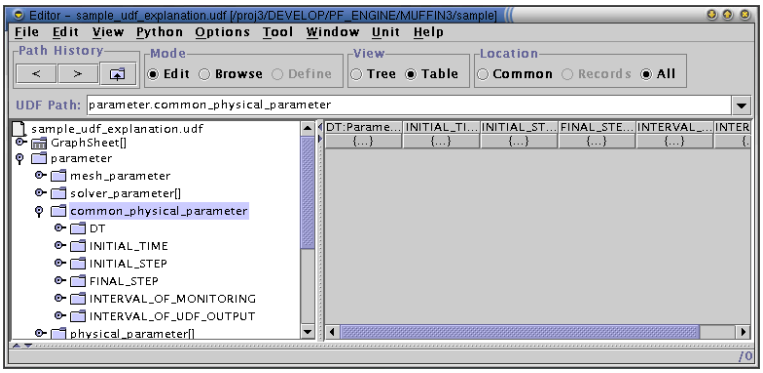

Figure 3.6: Common physical parameter

A parameter common to all the simulators of Muffin is described here. The following six parameters exist as common physical parameters that we have at this moment.

- DT: . . . The time unit for the integral in the direction of time
  If this value is set up, a dynamics manager's time counter will be automatically added only for DT in every 1STEP execution.

- INITIAL_STEP: . . . The record number of restart data (the default value is 0)
  It becomes the initial value of a dynamics manager's time step counter.
  (cautions: Positive integer in which INITIAL_STEP contains 0)

- INITIAL_TIME: . . . The initial value of a time counter (the default value is 0)
  It becomes the initial value of a dynamics manager's time counter.

- FINAL_STEP: . . . The number of the maximum calculation steps.
  The maximum in this simulation of a time step counter is specified.
  (Cautions: the FINAL_STEP must be a positive integer.)

- INTERVAL_OF_MONITORING: . . . Every this step interval, the status of simulation is outputted.

- INTERVAL_OF_UDF_OUTPUT: . . . Every this step interval, the record data of simulation is outputted.

A common physical parameters are shown in Fig.3.6 as an example.

Physical parameter (parameter.physical_parameter)

The physical constant which describes a system is inputted. Since the explanation on each simulator is described in the following parts I to IV, please refer to those chapters about the parameters and their meanings.

The physical parameters in Elastica are shown in Fig.3.7.

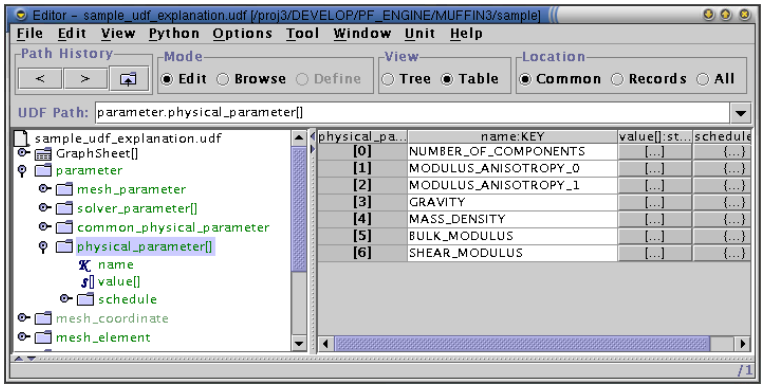

Figure 3.7: Physical parameter

Input of a parameter

The parameter data of

- Solver control parameter (parameter.solver_parameter)
- Common physics parameter (parameter.common_physical_parameter)
- Physical parameter (parameter.physical_parameter)

have the same structure, so shall we explain how to input these parameters together.

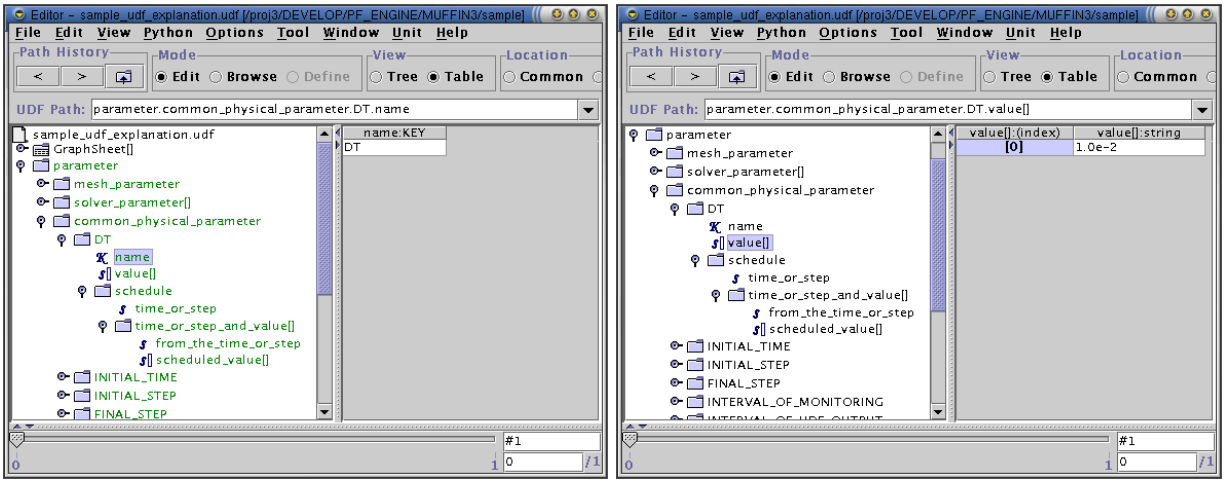

Figure 3.8: Input of a parameter

The structure of parameter data is as follows.

- name : . . . Parameter name
  The left figure of fig.3.8 shows the case of parameter name'DT'.
- value[] : . . . The array of the value (a character sequence or numerical value) of a parameter
  The right figure of fig.3.8 shows the case where the value of 'DT' is set to be $1.0e-2$.
- schedule : . . . The schedule of a parameter can be inputted here.

[Agreement01]

You can refer another parameter by writing as $(NameOfParameter).
Description of $(NameOfParameter) can be used in the following parts.

- Solver control parameter (parameter.solver_parameter)
- Common physics parameter (parameter.common_physical_parameter)
- Physical parameter (parameter.physical_parameter)
  The value of all these parameters.
- The value of boundary conditions
  region_condition.condition[].value[]
- Registration of the number of components a dynamics manager's field
  dynamics_manager.registered_field[].num_of_component
- The various steps and step width of a schedule of a dynamics manager
  dynamics_manager.schedule_of_simulation.evolution_plan.schedule[].from_this_time_step
  dynamics_manager.schedule_of_simulation.bifurcation_plan.interval_for_evaluation
  dynamics_manager.schedule_of_simulation.analysis_plan.interval_for_evaluation

[Agreement02]

About the operation of $(NameOfParameter) variable
For example, when writing like $(NumberOfComponent) $(*) $ (2), it was made to carry out calculation deployment automatically. Now, $ (*), $ (/), $ (+), and $ (-) can be used.
* Explanation * about an operation order
Every one operation is performed from before. Therefore, A+B*C is interpreted as (A+B) *C.

The input of parameter scheduling

The scheduling of a parameter is the function to change the value of a parameter automatically at two or more time (or step) which the user should set up beforehand. For example, you can give a parameter (for example, χ-parameter) as a function of time in the simulation. It is also possible to set the common physical parameter DT as 1.0e-2 at the initial time, to also change it into 5.0e-3 at the time step of 10000, and to change it again into 2.5e-3 at the time step of 20000. This will be useful when you want to make DT smaller gradually to stabilize a numerical computation scheme. The figure of Editor at the time of carrying out the schedule of the DT using STEP is shown in Fig.3.9.

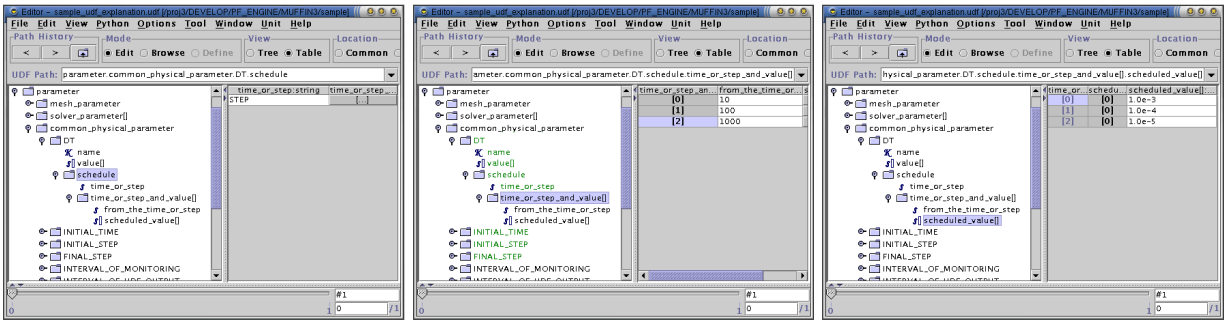

Figure 3.9: The input of the scheduling of a parameter

The structure of the schedule data of a parameter is as follows.

- time_or_step : . . . Either "TIME" or "STEP" should be described.
  It specifies whether a schedule is controlled by time or step.
  In the case of the left figure of Fig.3.9, the schedule of parameter'DT' is controlled by STEP.

- time_or_step_and_value[] : ... The array of schedules

- time_or_step_and_value[].from_the_time_or_step : ... The time or the step, at which a parameter is changed is inputted.
  The example in which parameter'DT' is changed at 10, 100, and 1000 STEP is shown in the center of Fig.3.9.

- time_or_step_and_value[].scheduled_value[] : ... The value of a parameter you want to change.
  The right of Fig.3.9 shows the example where a value is changed into $1.0e-3$ at the 10 STEP, into $1.0e-4$ at the 100 STEP and into $1.0e-5$ at 1000 STEP.

3.1.3 Mesh node coordinates data (mesh_coordinate)

The coordinates data of all node elements (Vertex) and mesh node coordinates data are outputted only in the case of an unstructured lattice.

When the mesh type of a mesh parameter is set to UNSTRUCTURED_INPUT, the data is read into the engine. What is necessary is just to put the node coordinates data generated by the pre-post processor into this mesh node coordinates data part, in using another meshing software as a pre processor and generating a mesh.

In the fluid system simulator (Euler picture) it is outputted to "initial data", while in the solid system simulator (Lagrange picture) it is outputted to "record data".

Below, a data structure is explained.

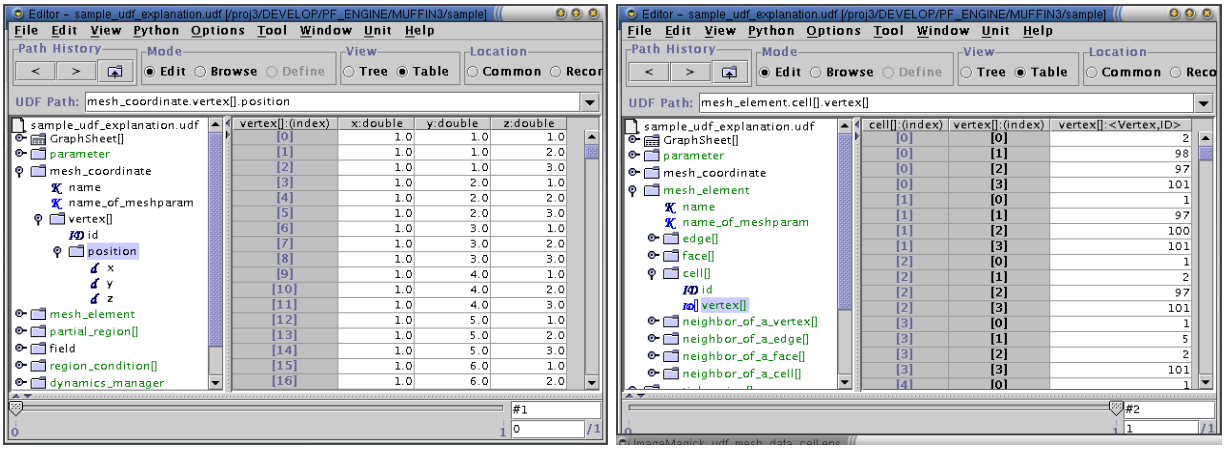

Figure 3.10: Mesh structure data

- vertex[] : ... Coordinates data of all the node elements that constitute a system
  The aggregate of these elements is treated by the name "**ALL_VERTEX**", from the data of a field, a dynamics manager, a python analysis program, etc.

  - vertex[].id : ... ID of a node element (Natural-number: 1, 2 and 3, ...)
  - vertex[].position : ... Coordinates of a node element
    - vertex[].position.x : ... X coordinates of a node
    - vertex[].position.y : ... Y coordinates of a node
    - vertex[].position.z : ... Z coordinates of a node

  Refer to the left figure of fig.3.10.

- unit : ... The unit of coordinates.

3.1.4 Mesh element connection data (mesh_element)

The connection data of the mesh element which constitutes the whole system is outputted here.

The information on all edge elements (Edge), all face elements (Face), all cell elements (Cell), and neighbor elements is written. Only in the case of an unstructured lattice, the mesh element connection data is outputted. Mesh type of a mesh parameter When referred to as UNSTRUCTURED_INPUT, the data here is read into engine and a mesh element is built. What is necessary is just to put the mesh element connection data generated by the pre-post processor into this mesh element connection data part, in using another meshing software as a pre processor and generating a mesh. All data are not outputted.

Although all the cell elements (Cell) that constitute a system are indispensable in the case of a 3-dimensional system, the output of an edge element (Edge), a face element (Face), and a neighbor element depends on the necessity for each simulator. Although similarly all the face elements (Face) that constitute a system are indispensable in the case of a 2-dimensional system, the output of an edge element (Edge) and a neighbor element depends on the necessity for each simulator. Moreover, in Muffin, all mesh element connection data is outputted and inputted from "common data."

In the following, a data structure will be explained.

- edge[] : ... Node information on all the edge elements that constitute a system
  The aggregate of these elements is treated by the name "**ALL_EDGE**", from the data of a field, a dynamics manager, a python analysis program, etc.
  - edge[].id : ... ID of an edge element (Natural-number: 1, 2 and 3, ...)
  - edge[].vertex : ... Node information on an edge element
    IDs of two nodes which constitute both ends of an edge.

- face[] : ... Node information on all the face elements that constitute a system
  The aggregate of these elements is specified by the name "**ALL_FACE**", from the data of a field, a dynamics manager, a python analysis program, etc.
  - face[].id : ... ID of a face element (Natural-number: 1, 2 and 3, ...)
  - face[].vertex : ... Node information on a face element
    IDs of three nodes which constitute the vertex of a triangle element are entered.

- cell[] : ... Node information on all the cell elements that constitute a system
  The set of these elements is specified by the name "**ALL_CELL**", from the data of a field, a dynamics manager, a python analysis program, etc.
  - cell[].id : ... ID of a cell element (Natural-number: 1, 2 and 3, ...)
  - cell[].vertex : ... Node information on a cell element
    IDs of four nodes which constitute the vertices of a tetrahedron element are entered.

  Refer to the right figure of Fig.3.10.

3.1.5 Partial region data (partial_region[])

In MUFFIN, partial region data is always outputted and inputted from "common data." When the type of a mesh parameter is UNSTRUCTURED_INPUT, partial region data as well as mesh structure data are read, and a partial region is built inside a simulator. When you generate a mesh inside a simulator without inputting mesh structure data, the set of an indispensable partial regions or a partial boundary is created. However, it describes this partial region data, and if it is registered with a different name from the partial region generated internally, partial region data will be read to add a partial region original in addition to it. All the partial region data generated by the simulator in the input or the inside is outputted to the partial region data of output data.

The following rules are prepared in the method of the name beginning of a partial region (set of an element).

1. If the character sequence of "**ALL**" is included, all the elements that constitute a system, or all boundary elements are expressed.

2. Including the character sequence of "**BOUNDARY**", if a top character is '**B**' or '**b**', it expresses that it is the set of a boundary element.
3. If the character sequence of "**VERTEX**" is included, the set of node elements is expressed.
4. If the character sequence of "**EDGE**" is included, the set of edge elements is expressed.
5. If the character sequence of "**FACE**" is included, the set of face elements is expressed.
6. If the character sequence of "**CELL**" is included, the set of cell elements is expressed.
   Any one character sequence of "**VERTEX**", "**EDGE**", "**FACE**", and the "**CELL**" needs to be contained in the partial region name. On the other hand, two or more character sequences using the four are not allowed to be included.
7. According to the above rule, a **reserved word** cannot re-define the following partial region name.
   (a) "**ALL_VERTEX**" ... The set of all node elements
   (b) "**ALL_EDGE**" ... The set of all edge elements
   (c) "**ALL_FACE**" ... The set of all face elements
   (d) "**ALL_CELL**" ... The set of all cell elements
   (e) "**ALL_BOUNDARY_VERTEX**" ... The set of all boundary node elements
   (f) "**ALL_BOUNDARY_EDGE**" ... The set of all boundary edge elements
   (g) "**ALL_BOUNDARY_FACE**" ... The set of all boundary face elements
   (h) "**ALL_BOUNDARY_CELL**" ... The set of all boundary cell elements

Below, a data structure is explained.

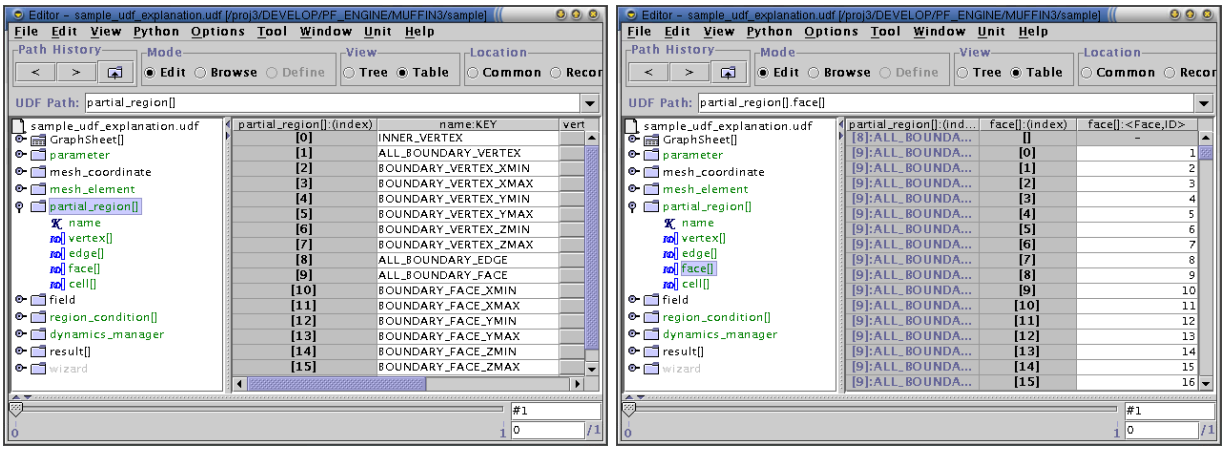

Figure 3.11: Partial region data - partial region name setup -

- name : ... Input the name of a partial region.
  The rule of naming is as mentioned above. Refer to the left figure of Fig.3.11.
- vertex[] : ... ID of the node elements which constitute a partial region is inputted.
  The data here becomes effective when "**VERTEX**" is contained in a partial region name. Refer to fig.3.12.
- edge[] : ... ID of the edge elements which constitute a partial region is inputted.
  The data here becomes effective when "**EDGE**" is contained in a partial region name.
- face[] : ... ID of the face elements which constitute a partial region is inputted.
  The data here becomes effective when "**FACE**" is contained in a partial region name.
- cell[] : ... ID of the cell elements which constitute a partial region is inputted.
  The data here becomes effective when "**CELL**" is contained in a partial region name.

  Refer to the right of Fig.3.11.

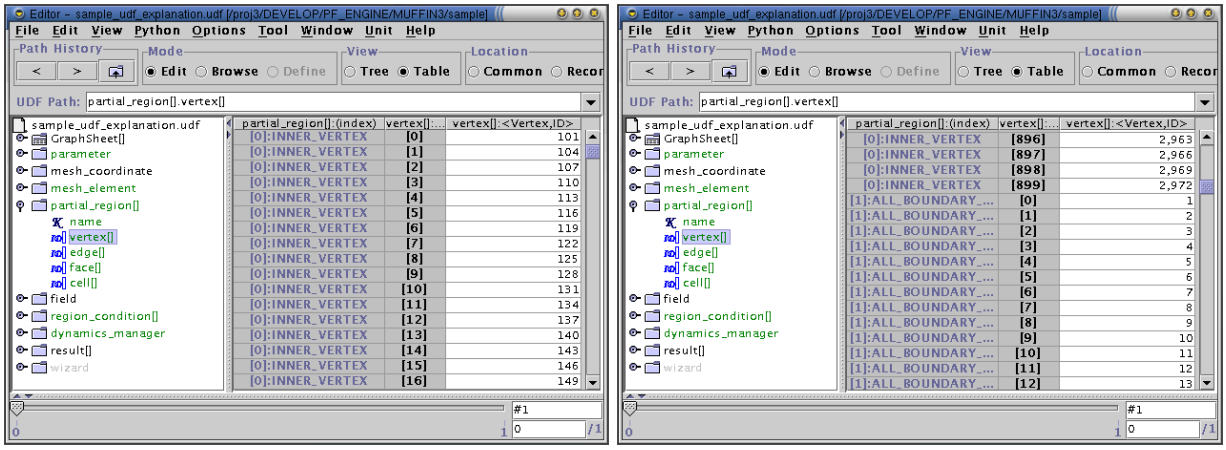

Figure 3.12: Partial region data - partial region element setup -

3.1.6 Field data (field)

The data part of fields consists of the following three arrays.

- The array of scalar fields (field.scalar_field[])
- The array of vector fields (field.vector_field[])
- The array of tensor fields (field.tensor_field[])

The rule of input and output of the value of a field is as follows.

- Input
  In the usual start, if there is data of the field in the initial data region, it will be inputted as an initial value.
  In a restart, if there is data of the field in the record, it will be inputted as an initial value.
- Output
  All the data of the field whose dynamics_manager.registered_field[].io_data_flag is 1 are outputted to record data.

Below, the structure of the data of each field will be explained.

Scalar field data (field.scalar_field[])

- name : ... The name of a scalar field. Refer to fig.3.13.
- name_of_region : ... The name of the region where this field is defined
  The partial region name mentioned above, or either of the reserved name of partial regions.
- num_of_component : ... The number of components of a field
  The number of the values of the field defined on an element (a node, a cell, etc.) in a partial region.
- value[]: ... The array of the value of a field (array about a mesh element)
  The value of the field in each element is outputted in the order of the mesh element array in the partial region data.
  - value[].comp[] : ... The array of the component of the field on each mesh element.

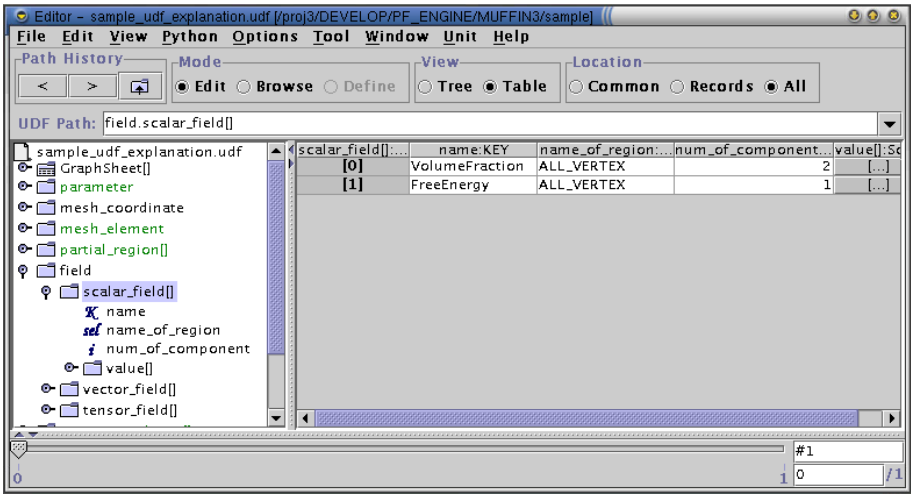

Figure 3.13: Scalar field data - The name of a field and a region -

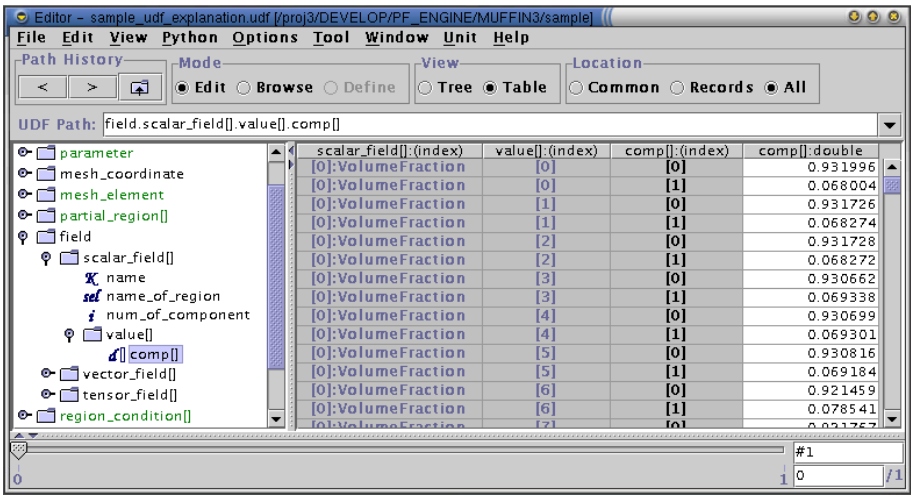

Figure 3.14: Scalar field data - Value of field -

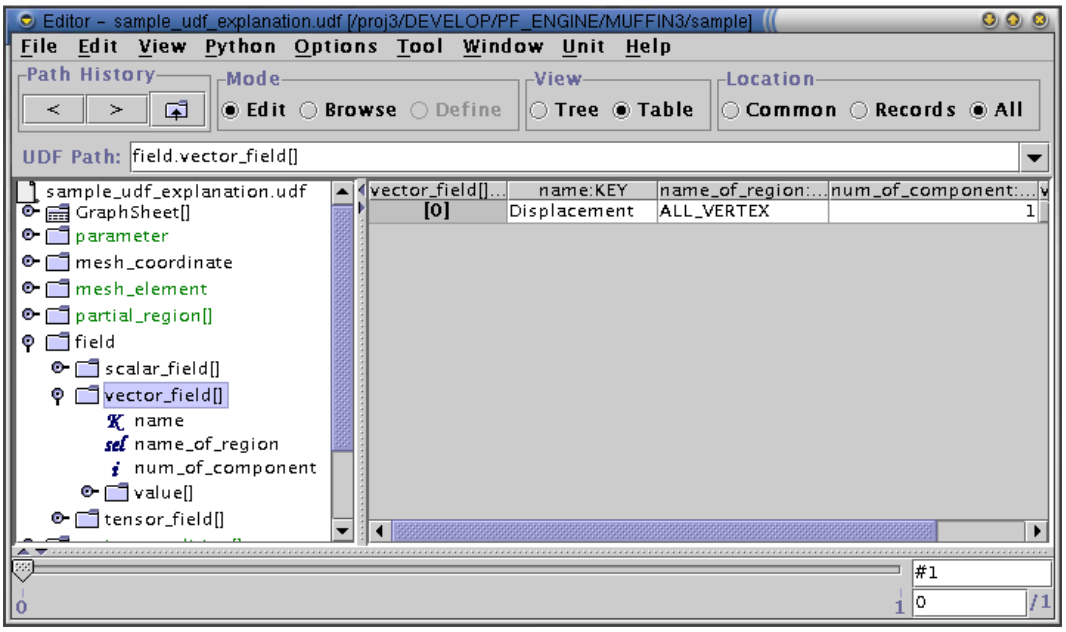

Figure 3.15: Vector field data - The name of a field and a region -

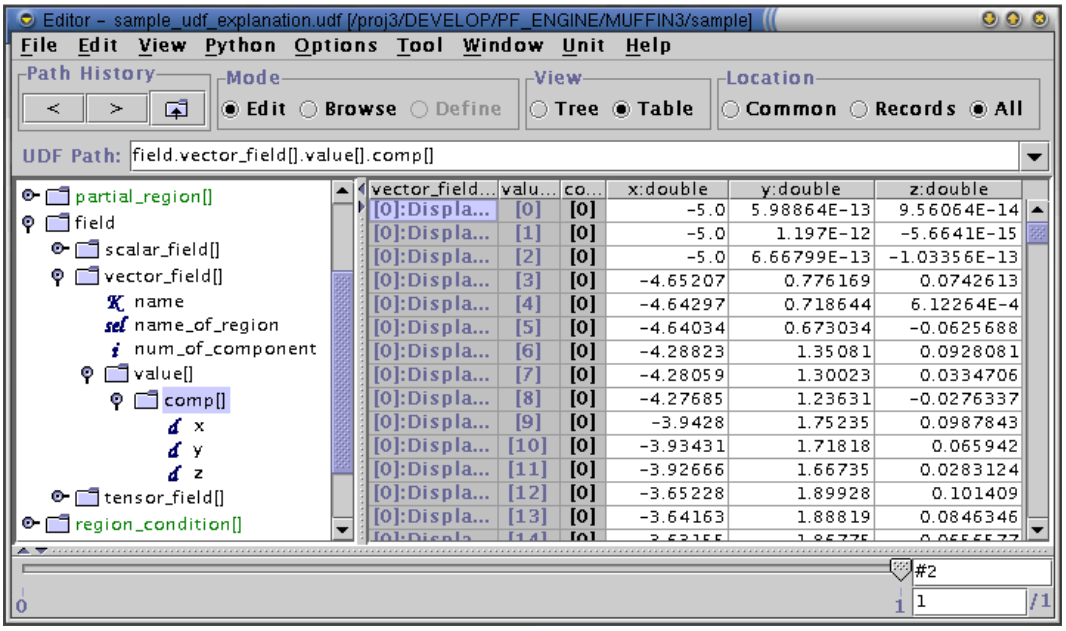

Figure 3.16: Vector field data - Value of field -

Vector field data (field.vector_field[])

- name : ... The name of a vector field. Refer to Fig.3.15.
- name_of_region : ... The name of the region where this field is defined
  The partial region name mentioned above, or either of the reserved name of partial regions.
- num_of_component : ... The number of components of a field
  The number of the values of the field on an element (a node, a cell, etc.) in a partial region.
- value[]: ... The array of the value of a field (array about a mesh element)
  The value of the field in each element is outputted in the order of the mesh element array in the partial region data.
  - value[].comp[] : ... The array of the value of the field in each mesh element (array about a component)
    - value[].comp[].x : ... The value of X component of the vector field on each element and each component
    - value[].comp[].y : ... The value of Y component of the vector field on each element and each component
    - value[].comp[].z : ... The value of Z component of the vector field on each element and each component

Tensor field data (field.tensor_field[])

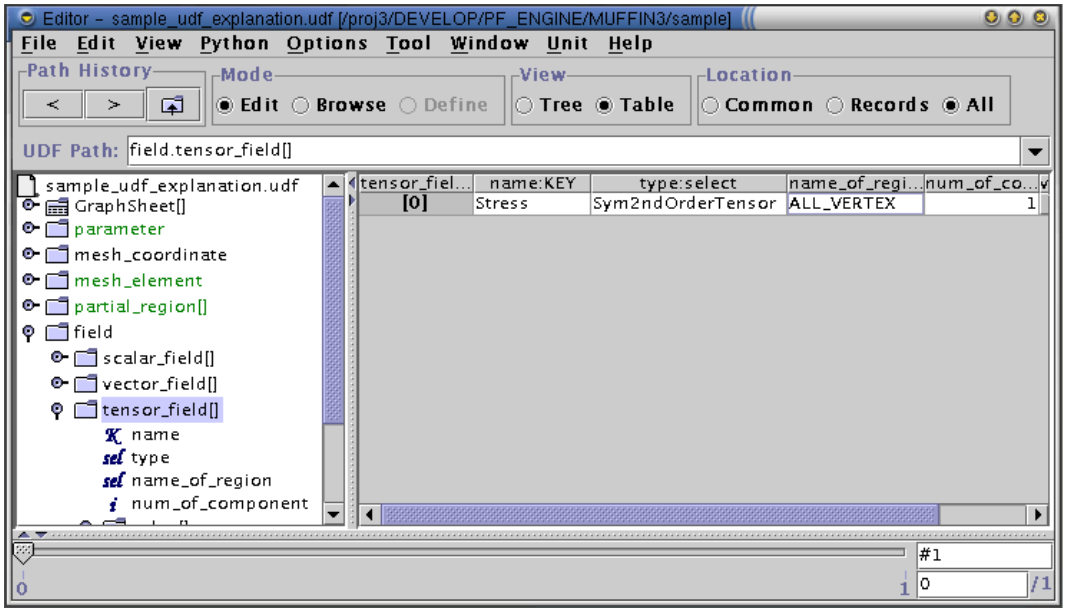

Figure 3.17: Tensor field data - The name of a field and a region, and the type of tensor -

- name : ... The name of a tensor field. In a symmetric second rank tensor field, refer to Fig.3.17.
- type : ... The type of the tensor. Either of the followings.
  1. 2ndRankTensor ... the second rank tensor
  2. Sym2ndRankTensor ... the symmetric second rank tensor
  3. 3rdRankTensor ... the third rank tensor
  4. 4thRankTensor ... the fourth rank tensor

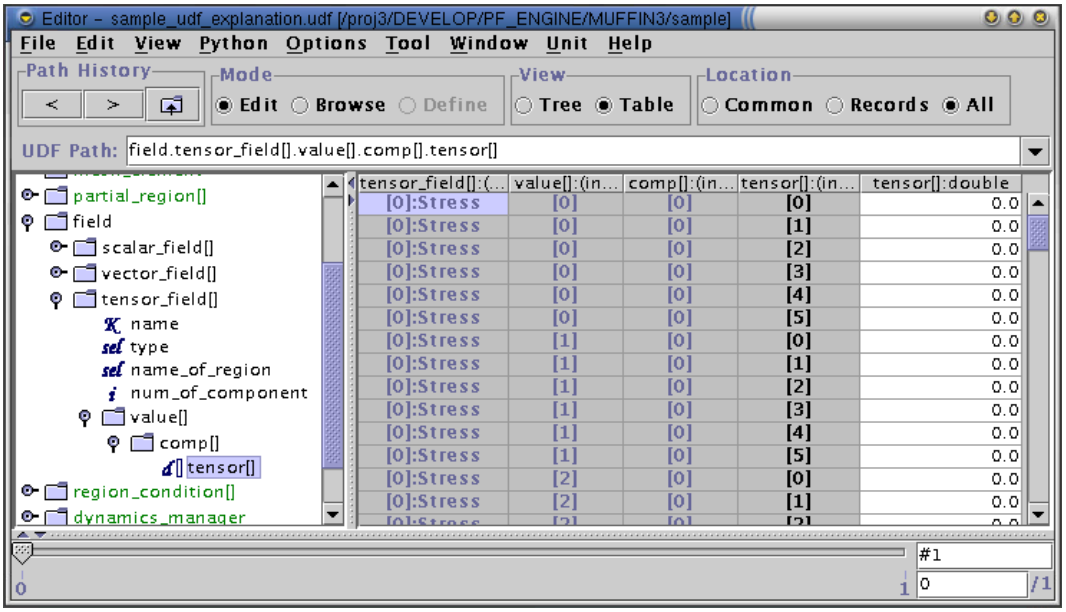

Figure 3.18: Tensor field data - Value of field -

- name_of_region : ... The name of the region where this field is defined
  The partial region name mentioned above, or either of the reserved name of partial regions.
- num_of_component : ... The number of components of a field
  The number of components of the field defined on an element (a node, a cell, etc.) in a partial region.
- value[]: ... The array of the value of a field (array about a mesh element)
  The value of the field on each element is outputted in the order of the mesh element array in the partial region data.
  - value[].comp[] : ... The array of the value of the field in each mesh element (array about a component)
    - value[].comp[].tensor[] : ... The value of the tensor field of each element and each component is held in an array.

3.1.7 Partial region (boundary) condition data (region_condition[])

In MUFFIN, a user can set up the (boundary) conditions of arbitrary fields and arbitrary partial regions (boundary) using partial region (boundary) condition data. When this function and the function in which a user can define a partial region (boundary) freely are put together, the simulation under various conditions is possible without change of a solver.

Partial region conditions are mainly used for the following cases.

- In the case of initializing a certain field with a fixed value in a certain partial region
- A setup of the boundary conditions of Dirichlet, Neumann, and Cauchy

Below, a data structure and the inputting method will be explained.

- name : ... The name of partial region conditions (if there is no duplication, any name can be useful.)
- name_of_region : ... The name of the partial (boundary) region which applies conditions.
  Either the partial region name mentioned above or the reserved name of a partial region.
- name_of_field : ... The name of the field to which conditions are applied
  Either of the names of an effective field (field in dynamics_manager.registered_field[].name) is inputted by the simulator.

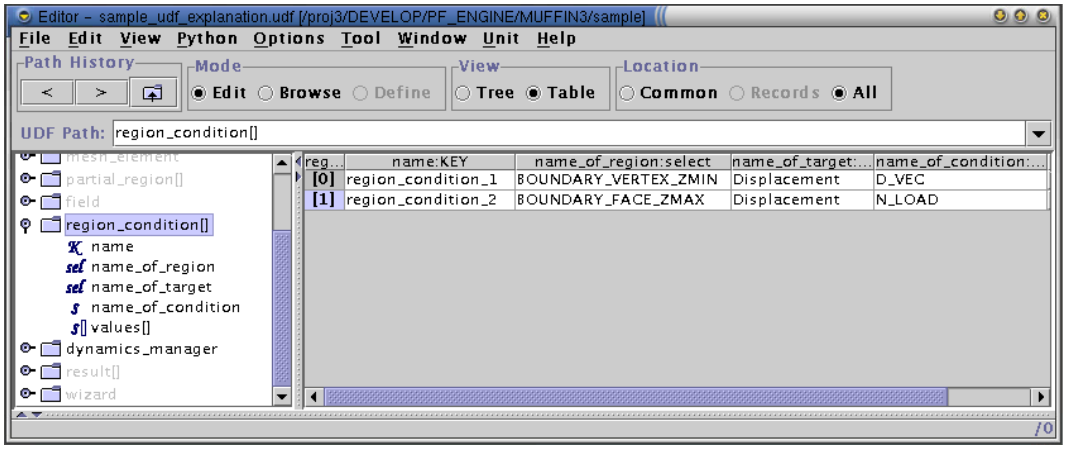

Figure 3.19: Partial (boundary) region condition data - Name of a condition, region and field -

- name_of_condition : . . . The name of partial region (boundary) conditions
  The name of the boundary conditions currently supported by the simulator (it can be used) should refer to the boundary condition description of each simulator. The following prefixes are prepared for naming the boundary conditions as a common rule.

  1. DIRICHLET boundary condition . . . Prefix of "**D_**" is attached like "**D_XXXXX**".
  2. NEUMANN boundary condition . . . Prefix of "**N_**" is attached like "**N_XXXXX**".
  3. CAUCHY boundary condition . . . Prefix of "**C_**" is attached like "**C_XXXXX**".
  4. Initialization conditions . . . Prefix of "**INI_**" is attached like "**INI_XXXXX**".

  An example is shown in Fig.3.19 for a case where the boundary conditions fix the bottom boundary perpendicular to the Z-axis
  "BOUNDARY_VERTEX_ZMIN" (Dirichlet conditions "D_VEC" about displacement field), and apply a force to the upper surface perpendicular to the Z-axis "BOUNDARY_FACE_ZMAX" (Neumann conditions "N_LOAD" about displacement field) are given.

- values[] : . . . The parameter of partial region (boundary) conditions
  The required number and meaning of parameters should refer to description of the boundary conditions of each simulator according to each conditions.

  1. DIRICHLET boundary condition . . . The parameter value of the Dirichlet conditions is inputted.
  2. NEUMANN boundary condition . . . The parameter value of the Neumann conditions is inputted.
  3. CAUCHY boundary condition . . . The parameter value of the Cauchy conditions is inputted.
  4. Initialization conditions . . . The initialization value of a field is inputted.

  An example is shown in Fig.3.20 for a case where the position of the bottom surface perpendicular to the Z-axis "BOUNDARY_VERTEX_ZMIN" is fixed (the Dirichlet condition "D_VEC" about displacement field a value $(0.0, 0.0, 0.0)$) and a force is applied to the upper surface perpendicular to Z-axis "BOUNDARY_FACE_ZMAX" (the Neumann condition "N_LOAD" about displacement field and a value $(5.0, 0.0, 0.0)$).

3.1.8 Analysis result data (result[])

The name and value of data of a result which are analyzed inside the simulator are outputted to analysis result data of each record. Refer to guidance of "dynamic analysis execution" of "3.2 Dynamics Manager", and the analysis command list of each simulator description part about the analysis program included in the simulator.

The structure of analysis result data is as follows.

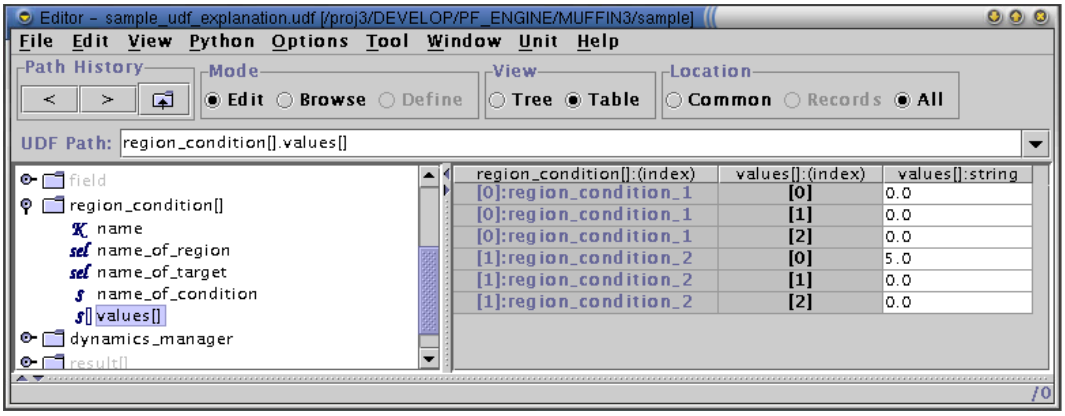

Figure 3.20: Partial region (boundary) condition data - Value of conditions -

- name : . . . The data name of an analysis result
- value[] : . . . Data values of an analysis result. (a character sequence or numerical value)

3.2 Dynamics manager

Here, the dynamics manager which is the important basic function of a MUFFIN simulator is explained.

All the simulators in Muffin are built using the dynamics manager which is the basic function of Muffin. Moreover, all the basic functions of Muffin, such as input and output of UDF files, mesh, partial regions, fields, and boundary conditions, are managed by the dynamics manager function. Therefore, input-and-output UDF form and directions are common to all simulators in Muffin.

A dynamics manager's feature is two of the followings.

- Scheduling function
  The time schedule table of a parameter or a simulation is created and a simulation based on the plan is performed.
- Dynamics construction function
  User can select the field to solve, the equation to be used, boundary conditions, an initial state, etc. and can build up the simulator which a user wants to use.

Before describing the data structure in the input- and output-UDF file of dynamics manager data and the input method, we define terminologies used in the dynamics manager in the next section.

3.2.1 The definition of terms of dynamics manager

Here, the terminologies which are necessary to understand the dynamics manager are explained.

The dynamics manager's feature is the "scheduling function" and the "dynamics construction function." Such functions differ from the use form of the usual simulator fundamentally. The terms required in order to define these functions by the dynamics manager are explained below.

There are seven basic terms of the dynamics manager as:

1. **Command**
2. **Procedure**
3. **ProcedureTable**
4. **DynamicBifurcation**
5. **DynamicAnalyzer**
6. **Schedule**

7. **DynamicsManaging**

The definitions of these are explained in detail below.

1. **Command** ... The minimum basic unit which constitutes a solver.
   In Muffin, one "Command" consists of combination of a field and a message, which are sent to the field.

   Command

   Command = an Operand (a_Field_object) + a Message

   ex.) Command = "Chemical Potential" + "calculation using Flory-Huggins free energy."
   Thus, what combines the operand (a certain field) and the message is called "Command". The Commands is classified into five groups in Muffin.

   - The Command for initialization
   - The Command for 1 step execution
   - The Command for a boundary condition setup
   - The Command for analysis execution
   - The Command for evaluation execution

2. **Procedure** ... The minimum unit of a processing procedure.
   A "Procedure" is the set of one or more Commands, and it is the minimum unit of a processing procedure. The execution of a Procedure performs a series of Commands in order of registration. Moreover, you can give a name to a Procedure and the Procedure is called **NamedProcedure**. **NamedProcedure** is classified into the following two groups.

   - Initialization Procedure
   - (One step) Time development Procedure

   The Procedure may be built using the above-mentioned Commands except for the Command for evaluation execution.

3. **ProcedureTable** ... The table of Procedures built from Commands.
   A **NamedProcedure** of Muffin will be registered into the following two procedure tables as mentioned above.

   - Initialization Procedure table
     A set of **NamedProcedure**s which will be used as an initialization of fields.
   - (One step) Time development Procedure table
     A set of **NamedProcedure**s which will be used in the calculation of the time evolution for fields.

4. **DynamicBifurcation** ... Dynamic change of dynamics.
   DynamicBifurcation means a mechanism which change a Procedure into another Procedure according to the return value of the Bifurcation evaluation command during a simulation. If you input the time schedule of simulation, the schedule is canceled when **DynamicBifurcation** is executed.

5. **DynamicAnalyzer** ... Dynamic execution of an analysis program.
   DynamicAnalyzer means a mechanism which executes the analysis commands of two or more fields according to the return value of the Analysis evaluation commands during a simulation.

6. **Schedule** ... Simulation execution plan (time schedule).
   A building of a schedule is as follows.

   - To select an initialization procedure from an initialization procedure table.
   - To select a time development procedure from a time development procedure table, and the step to start the procedure and to end the procedure. To determine such a plan up to FINAL_STEP.
   - To determine the dynamic bifurcation condition and dynamic analysis execution to perform.

7. **DynamicsManaging** ... To perform a Schedule.

3.2.2 The functions of dynamics manager

Here, the flow of processing of a dynamics manager and the outline of how to use a function are explained using a flow chart.

The flow of the Muffin simulator

First, a flow chart in Fig.3.21 shows the flow of processing common to Muffin. Processing progresses in the order of reading of an input UDF file, construction of a parameter, a mesh, and a field, construction of a dynamics manager, initialization and execution of a simulation, generation of output data, and a simulation ends.

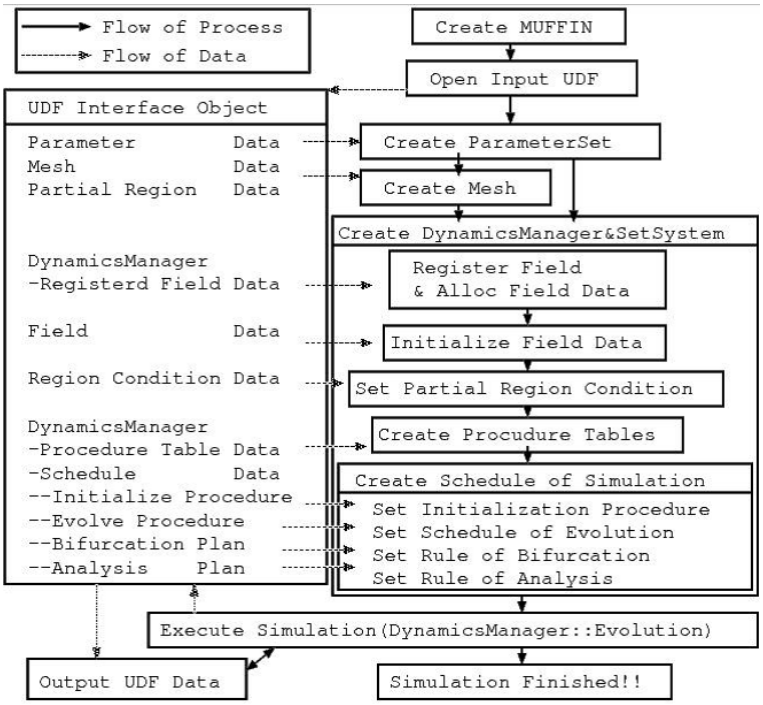

Figure 3.21: The flow of the Muffin simulator

The flow of dynamics manager

After generating a dynamics manager, and initializing it, the simulation is automatically performed with the schedule given to the dynamics manager. For an understanding of a dynamics manager's function, a flow chart in Fig.3.22 shows processing of the contents of simulation execution.

3.2.3 The outline of directions of dynamics manager

A dynamics manager's scheduling function is designed to perform the following three types of simulation:

1. the simulation which changes a procedure at a certain time
2. the simulation which performs condition bifurcation dynamically
3. the simulation which performs analysis processing dynamically

How to perform three kinds of the above simulation plans will be explained briefly in the following sections.

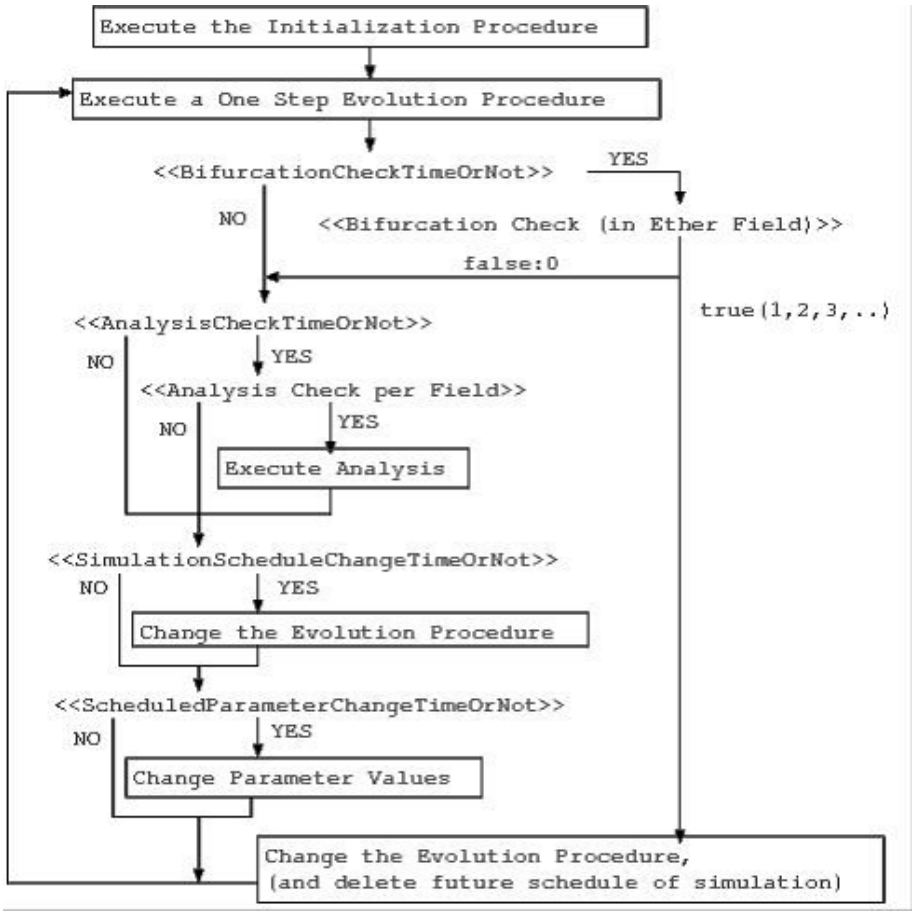

Figure 3.22: The flow of Dynamics Manager

from_this_time_step	this_procedure_will_be_used
INITIALIZE	InitializeA
$(INITIAL_TIME)	EvolveA
TIME1	EvolveB
TIME2	EvolveC

Table 3.1: The simulation which changes a procedure at a certain decided time

return_value	this_procedure_...bifurcation
1	EvolveD
2	EvolveE
3	EvolveF

Table 3.2: The simulation which performs condition bifurcation dynamically

The simulation which changes a procedure at a certain time

For example, if you want to execute the initializing procedure "A"(InitializeA), the procedure "A"(EvolveA) from INITIAL_TIME to TIME1, the procedure "B"(EvolveB) from TIME1 to TIME2, and the procedure "C"(EvolveC) from TIME2 to FINAL_TIME, input "evolution_plan" as shown in table 3.1.

The simulation which performs condition bifurcation dynamically

For example, when you execute the initializing procedure "A"(InitializeA) and the procedure "A"(EvolveA) after initialization, if you want to change the procedure (EvolveD, EvolveE or EvolveF) according to the return value (1, 2 or 3) of evaluation function, input "bifurcation_plan" as shown in table 3.2.

The simulation which performs analysis processing dynamically

Using "analysis_plan", you can make plans as follows.

1. A fixed time interval analysis execution.
   It is enabled by using the function which always returns a true (non-zero) as an evaluation function.

2. The analysis execution plan controlled by condition judgment.
   For example we consider the following dynamic analysis. Two evaluation functions ("nameOfEvalFunc_C" and "nameOfEvalFunc_D") of a field ("nameOfField_Y") and the evaluation function ("nameOfEvalFunc_E") of a field ("nameOfField_Z") are performed every 1000 steps, and if each evaluation value is a true, the corresponding analysis function of a field is performed. Such a dynamic analysis can be performed as follows :

   - Input "STEP" to analysis_plan.unit_of_schedule.
   - Input "1000" is described to analysis_plan.interval_for_evaluation.
   - Input the name of the analysis function which is performed when the name of a field, the name of an analysis evaluation function, and an evaluation value are true, into evaluation_function_name, and analysis_function_name of analysis_plan.analyzer_list[], as shown in table 3.3.

field	evaluation_function_name	analysis_function_name
nameOfField_X	nameOfEvalFunc_A	nameOfAnalyzerFunc00
nameOfField_X	nameOfEvalFunc_B	nameOfAnalyzerFunc01
nameOfField_Y	nameOfEvalFunc_C	nameOfAnalyzerFunc11
nameOfField_Y	nameOfEvalFunc_D	nameOfAnalyzerFunc12
nameOfField_Z	nameOfEvalFunc_E	nameOfAnalyzerFunc23

Table 3.3: The simulation which performs analysis processing dynamically

3.2.4 Dynamics manager data of input and output UDF (dynamics_manager)

The Outline of Dynamics Manager data (dynamics_manager)

In the dynamics manager part of an input UDF file, a simulation is assembled and a simulation schedule is described. The dynamics manager data of an input is outputted to the dynamics manager part of an output UDF file as it is. The dynamics and the schedule which were used in the simulation are recorded on it. Since the dynamics manager part of an output UDF file is read in the case of a restart, it is necessary to edit it before a restart, to change dynamics and a schedule.

An input-and-output UDF dynamics manager data consists of four parts as shown below.

1. The fields to be registered (dynamics_manager.registered_field[]).
   The field used in a simulation is registered from the list of fields which can be used written to the manual.
2. Initialization procedure table (dynamics_manager.procedures_table_for_initialization[]).
   Register the candidates of initialization procedure for each of the registered field.
3. Time development procedure table (dynamics_manager.procedures_table_for_evolution[]).
   In time development, a group of the command executed for the registered field is created with an order, and is named and registered to make it into a (one step) time development procedure. Two or more time development procedures can be registered. The schedule of a simulation, which is explained later, can be created using two or more procedures.
4. Schedule of simulation (dynamics_manager.schedule_of_simulation).
   The schedule of a simulation is described. The following four items are inputted if necessary.
   (a) Selection of an initialization procedure.
   Initialization to be used is chosen from an initialization procedure table.
   (b) Selection of a time development procedure, and a time schedule.
   Select a time development procedure to be used from a time development procedure table, and it is determined to which time steps this procedure is applied using a step number or time. This must be specified till the last time (FINAL_STEP or FINAL_TIME).
   (c) Dynamic condition bifurcation.
   The following information is described in order to perform bifurcation of a simulation.
      i. The name of the evaluation command which gives a condition of bifurcation.
      ii. A combination of the returned value and time development procedure for changing a time development procedure according to the return value of an evaluation command.
      iii. Time (or STEP) interval to execute an evaluation command.
   (d) Dynamic analysis execution.
   Select a field to analyze and a method (command for analysis execution) for analysis, with an evaluation command which decides whether analysis is performed or not. Execution of two or more analyses is possible.

The fields to be registered (dynamics_manager.registered_field[])

The method of describing the field registered into a dynamics manager is explained. Two or more fields required for a simulation are registered here. For each of the field registered, its name and type, the number of components, and the existence of output data are registered.

Below, a data structure and the inputting method are explained.

- name : ... The name of a field
  The name of a field is chosen from "a list of the field which can be used" of the references of each simulator.
- type : ... The type of the variable of a field.
  1. Scalar ... Scalar field
  2. Vector ... Vector field

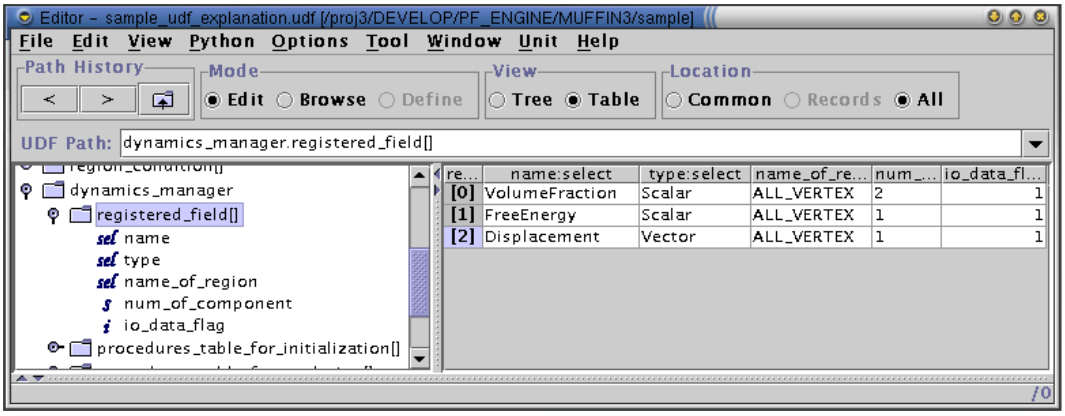

Figure 3.23: Dynamics manager - registered field - (UDF description)

Name of a field	Type of a field	Region	Number of components	Output data
VolumeFraction	Scalar	ALL_VERTEX	2	yes
FreeEnergy	Scalar	ALL_VERTEX	1	yes
Displacement	Vector	ALL_VERTEX	1	yes

Table 3.4: Dynamics manager - registered field - (UDF description)

3. 2ndRankTensor . . . 2nd. rank tensor field
4. Sym2ndRankTensor . . . Symmetric 2nd. rank tensor field
5. 3rdRankTensor . . . 3rd. rank tensor field
6. 4thRankTensor . . . 4th. rank tensor field

- name_of_region : . . . The name of the region where the field is defined
- num_of_component : . . . The number of components of a field
- io_data_flag : . . . Existence of the output of the data of this field to an output UDF file.
  "1" means there is output, while "0" means there is none. In the case of "1", the data of a field is outputted to an output UDF file. The data of a field to re-input in the case of a restart calculation should be outputted.

The above five data must be given to each field to register. The example of an input is shown in Fig.3.23. Fig.3.23 is an example which registers three fields of table3.4.

Usually, however, a user will edit the input template UDF file and the sample UDF file of each simulator and create an input UDF file. In such a case, it is not necessary to consider and put in these data.

The procedure table of initialization and time development

A dynamics manager's initialization and the method of describing a time development procedure table are explained. Here, one or more initialization procedures and time development procedures are described, respectively. It becomes possible to build the simulation schedule using these tables.

Since the structure of initialization procedures table (dynamics_manager.procedures_table_for_initialization[]) is the same as the procedure table of time development procedures table (dynamics_manager.procedures_table_for_evolution[]), a data structure and the inputting method for these are explained together.

A procedure table is the array of a procedure. One procedure is created in inputting a procedure name and a series of command sequences (the names of a field and a message). Here, the case where four procedures of table 3.5 are registered into an initialization procedure table is shown and explained in Fig.3.24 as an initialization procedure.

[1]	Initialization procedure name	INITIALIZE:LINEAR_ELASTICITY: ISOTROPIC:UNIFORM
The order of execution	The name of a field	The name of the message to perform
1	VolumeFraction	INITIALIZE:UNIFORM
2	Displacement	INITIALIZE:TO_ZERO
3	FreeEnergy	SOLVE:LINEAR_ELASTICITY
[2]	Initialization procedure name	INITIALIZE:LINEAR_ELASTICITY: ISOTROPIC:ONE_COMPONENT
The order of execution	The name of a field	The name of the message to perform
1	VolumeFraction	INITIALIZE:ONE_COMPONENT
2	Displacement	INITIALIZE:TO_ZERO
3	FreeEnergy	SOLVE:LINEAR_ELASTICITY
[3]	Initialization procedure name	INITIALIZE:LINEAR_ELASTICITY: ISOTROPIC:TWO_COMPONENT
The order of execution	The name of a field	The name of the message to perform
1	VolumeFraction	INITIALIZE:TWO_COMPONENT
2	Displacement	INITIALIZE:TO_ZERO
3	FreeEnergy	SOLVE:LINEAR_ELASTICITY
[4]	Initialization procedure name	INITIALIZE:LINEAR_ELASTICITY: ISOTROPIC:MULTI_COMPONENT
The order of execution	The name of a field	The name of the message to perform
1	Displacement	INITIALIZE:TO_ZERO
2	FreeEnergy	SOLVE:LINEAR_ELASTICITY

Table 3.5: Dynamics manager - Example of initialization procedure -

Moreover, the case where two procedures of table 3.6 are registered into a time development procedure table is shown in Fig.3.25.

The data structure and the inputting method of a procedure are explained below.

- name : ... The name of a procedure.
  The example of an input of an initialization procedure name is shown in the upper part of Fig.3.24, and the example of an input of a time evolution procedure name is shown in the upper part of Fig.3.25.

- command_list[]: ... A series of commands executed by this procedure.
  Since commands are performed in the order of registration, be careful about the order of an input. One command consists of the following data.

  - command_list[].field : ... The name of a field.

[1]	Time development procedure name	SOLVE:LINEAR_ELASTICITY:ISOTROPIC
The order of execution	The name of a field	The name of the message to perform
1	Displacement	SOLVE:LINEAR_ELASTICITY:ISOTROPIC
2	FreeEnergy	SOLVE:LINEAR_ELASTICITY
3	Displacement	MOVE:POSITION_OF_VERTEX
[2]	Time development procedure name	SOLVE:LINEAR_ELASTICITY:ANISOTROPIC
The order of execution	The name of a field	The name of the message to perform
1	Displacement	SOLVE:LINEAR_ELASTICITY:ANISOTROPIC
2	FreeEnergy	SOLVE:LINEAR_ELASTICITY
3	Displacement	MOVE:POSITION_OF_VERTEX

Table 3.6: Dynamics manager - Example of a time development procedure -

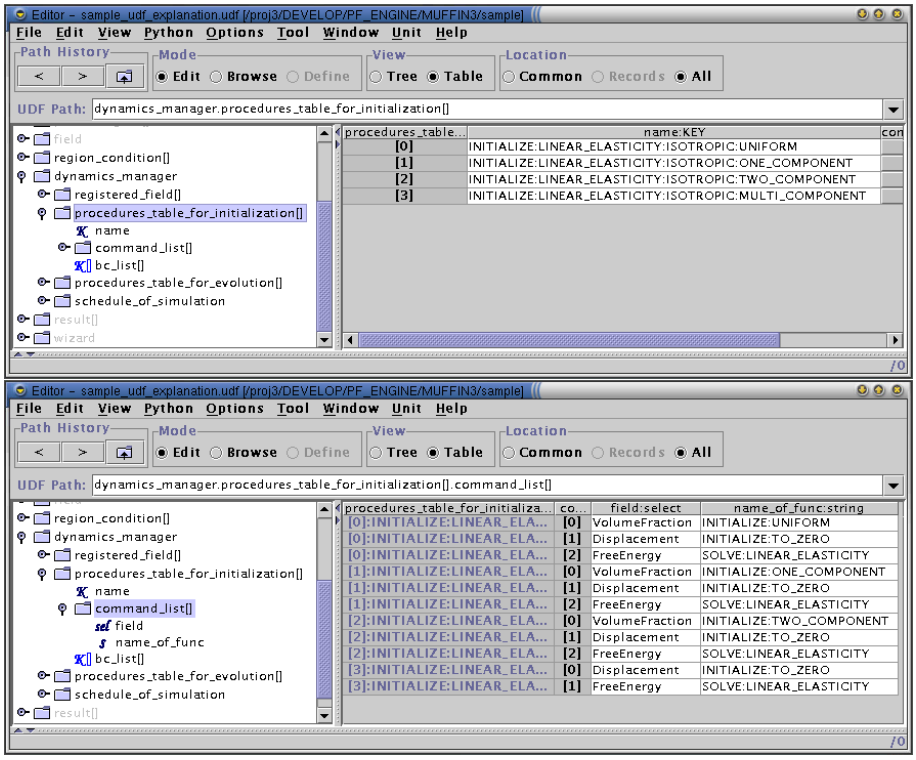

Figure 3.24: Dynamics manager - initialization procedure table -

– command_list[].name_of_func : ... The name of the processing (message) to be performed.
Select a command from the command list of each fields written in the reference manual.

The example of an input of the commands of an initialization procedure is shown in the lower part of Fig.3.24, and the example of an input of the commands of a time development procedure is shown in the lower part of Fig.3.25.

Description of a schedule (dynamics_manager.schedule_of_simulation)

Here, the method of describing a dynamics manager's simulation schedule is explained.

All the information about the schedule of a simulation is described here. Select and input the initialization procedure used in a simulation and of a time development procedure, and, if required, input the time schedule of a time evolution procedure, dynamic condition bifurcation, and dynamic analysis execution.

The outline and the describing method of an input data structure of a schedule are explained below.

- name : ... The name of a schedule (good anything)
The example of an input is shown in Fig.3.26.

- evolution_plan : ... Selection of an initialization procedure, and description of the simulation of a type which changes a procedure at a certain time

  – evolution_plan.unit_of_schedule : ... The unit used for a time schedule
  Either "TIME" or "STEP" is inputted.

    1. "TIME" ... The scheduling is controlled using time.
    2. "STEP" ... The scheduling is controlled using a calculation step.

  An example in the case of performing a schedule controlled using a calculation step is shown in Fig.3.27.

  – evolution_plan.schedule[] : ... Selection of an initialization procedure and a time development procedure.

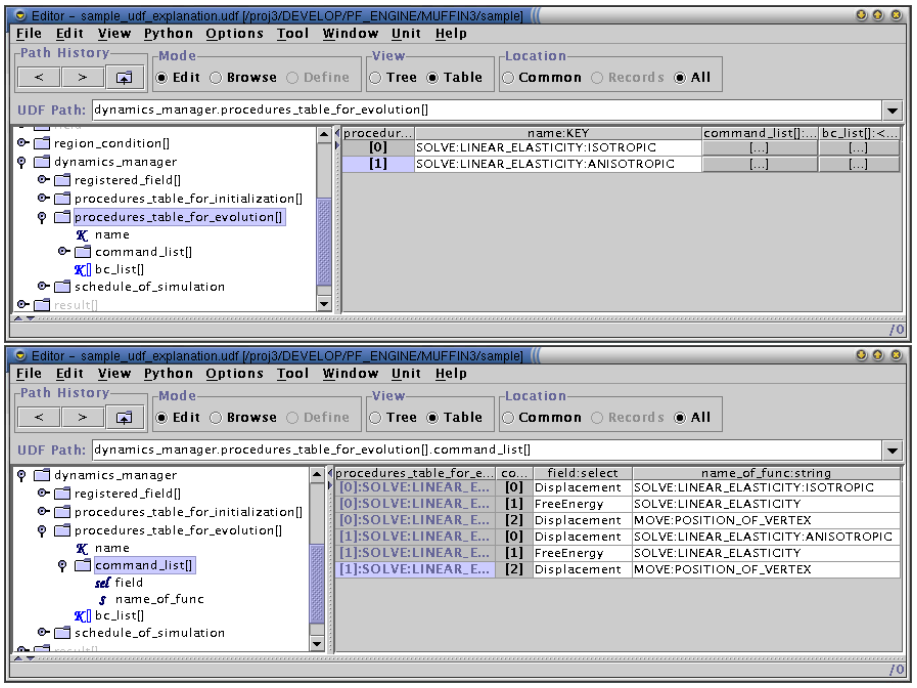

Figure 3.25: Dynamics manager - time development procedure table -

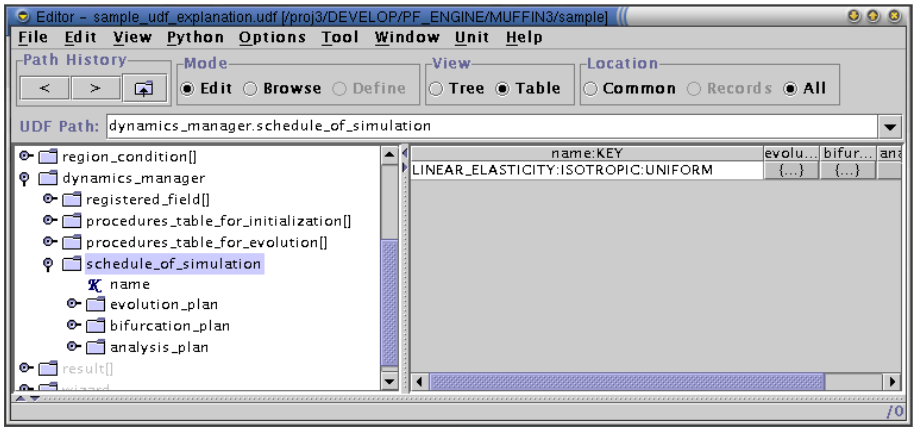

Figure 3.26: Dynamics manager - Name of schedule -

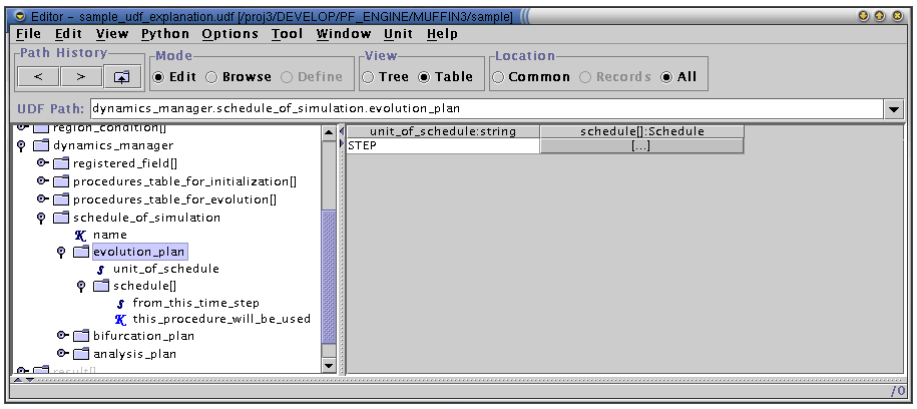

Figure 3.27: Dynamics manager - Unit of time schedule -

An initialization procedure is the first data of an array, and the execution schedule of a time development procedure is entered in the data of the 2nd henceforth. Schedule data is given as a combination of the name of start time or start step and the procedure to perform, and a schedule is registered as an array of these data. As a starting time or step of an initialization procedure, a string "INITIALIZE" must be inputted.

* evolution_plan.schedule[].from_this_time_step : ... Start time or start step.
* evolution_plan.schedule[].this_procedure_will_be_used : ... Execution procedure name.

An example of input is shown in Fig.3.28.

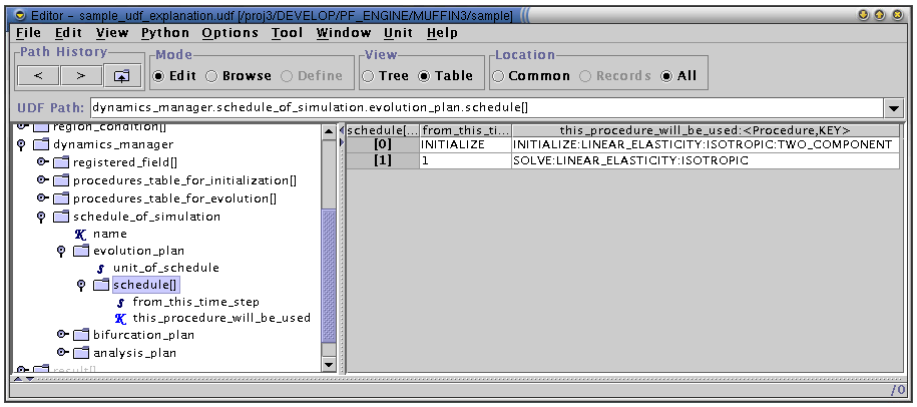

Figure 3.28: Dynamics manager - time development schedule -

- bifurcation_plan : ... Description of a dynamic condition bifurcation type simulation.
  - bifurcation_plan.unit_of_schedule : ... The unit used for a time schedule. Input either "TIME" or "STEP".
  - bifurcation_plan.interval_for_evaluation : ... Evaluation command execution interval.
  - bifurcation_plan.evaluation_function_name : ... Condition bifurcation evaluation command name.

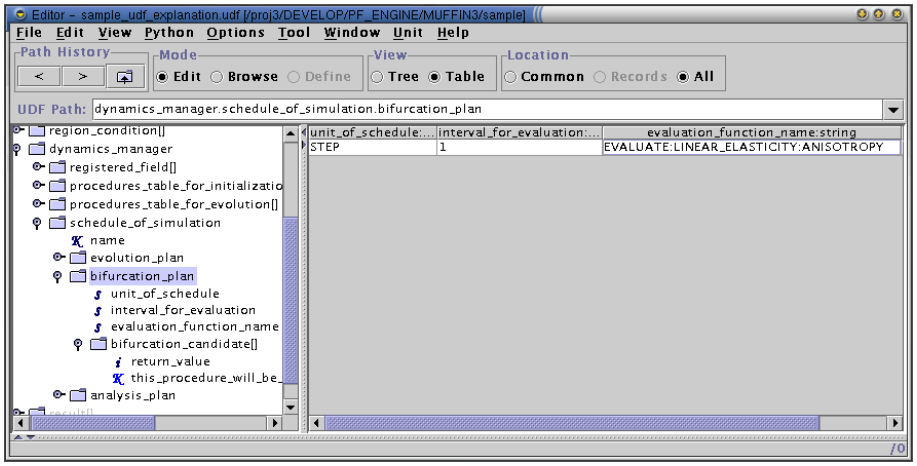

Figure 3.29: Dynamics manager - dynamic condition bifurcation / evaluation command -

  - bifurcation_plan.bifurcation_candidate[] : ... Description of the candidate procedure of a bifurcation.
    When the return value of the condition bifurcation evaluation command inputted in the above is 0 (an evaluation result is false), bifurcation does not take place. Only in the case of a natural number $(1, 2, 3, ...)$ is returned, the value changes a time development procedure. The relation of the returned value and the name of a time development procedure to change is described below.

* bifurcation_plan.bifurcation_candidate[].return_value : ... The return value of an evaluation function.
  The return value from an evaluation function of the bifurcation condition is inputted. A return value must be a natural number $(1, 2, 3, ...)$ and in order from a candidate's head.
* bifurcation_plan.bifurcation_candidate[]. this_procedure_will_be_used_for_evolution_after_bifurcation : ... The name of a time development procedure to perform after bifurcation.
  Corresponding to each return value, the name of a time development procedure is chosen from a time development procedure table, and is filled in.

An example is shown in Fig.3.30. In this example, when a return value is 1, a time development procedure is changed into
"SOLVE:LINEAR_ELASTICITY:ISOTROPIC", and when a return value is 2, the case where it changes into
"SOLVE:LINEAR_ELASTICITY:ANISOTROPIC".

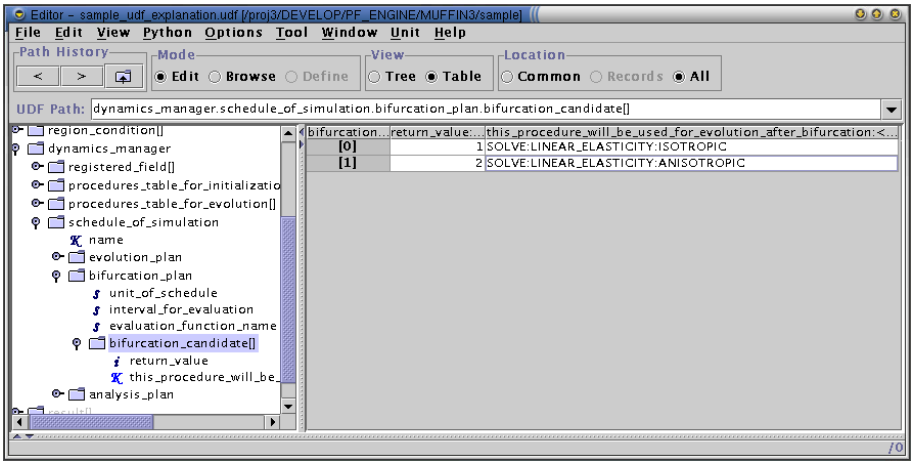

Figure 3.30: Dynamics manager - dynamic condition bifurcation and candidate procedure -

- analysis_plan : ... Description of dynamic analysis execution.
  The dynamic analysis execution is described. It is a mechanism which executes the registered analysis execution evaluation command for the registered field for every fixed time of calculation, or fixed step, and will execute the registered analysis execution command if the return value is a true (non-zero). The field for analysis, an analysis evaluation command, and an analysis execution command are described.
  - analysis_plan.unit_of_schedule : ... The unit used for a time schedule.
    Input either "TIME" or "STEP".
  - analysis_plan.interval_for_evaluation : ... Evaluation command execution interval.
    An example of an input in the case of executing an analysis execution evaluation command at a calculation step at the time of each step end is shown in Fig.3.31.
  - analysis_plan.analyzer_list[] : ... The list of the field to analyze, an analysis evaluation command, and analysis execution commands.
    * analysis_plan.analyzer_list[].field : ... The name of the field for analysis.
    * analysis_plan.analyzer_list[].evaluation_function_name : ... The name of the analysis evaluation command of a field.
    * analysis_plan.analyzer_list[].analysis_function_name : ... The name of the analysis execution command of a field.

    Fig.3.32 shows an example of an input to perform analysis execution evaluation by "EVALUATE:TRUE" (evaluation command which always returns a true), and performs "OUTPUT:AVS" (command which carries out the file output of the data of a field in AVS form) as an analysis function of the both "a free energy field (FreeEnergy)" and "a volume-fraction field (VolumeFraction)".

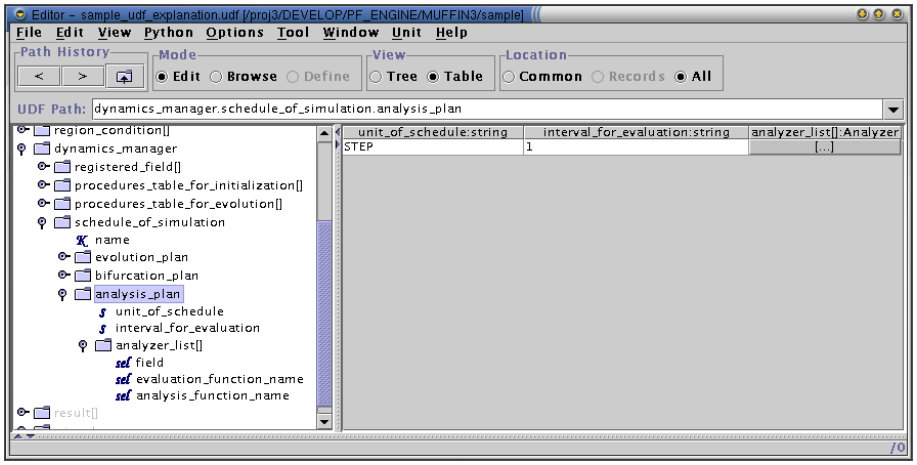

Figure 3.31: Dynamics manager - dynamic analysis execution / evaluation command -

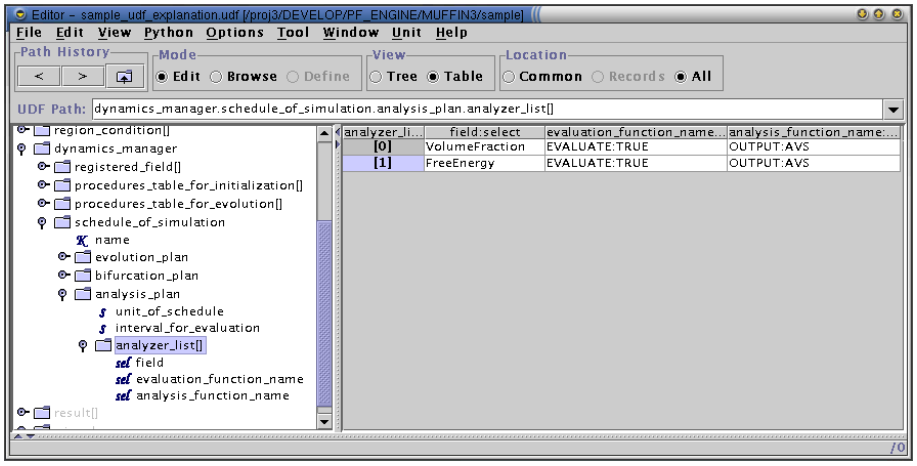

Figure 3.32: Dynamics manager - dynamic analysis execution / analysis execution command -

3.3 Control parameter input UDF

Here, the data structure of the control parameter input UDF (it specifies by {-P parameterUDF} option) for making a parameter change on real time during calculation of Muffin engine is explained.

3.3.1 The outline of control parameter input UDF

The control parameter input UDF consists of two parts as follows.

1. Control parameter section (parameter[]).
   Describe the name and value of a parameter to be changed during calculation.

2. Analysis-result output-control section (summary_control).
   Specify a data outputted to an analysis-result output file (summaryUDF) in a report or in a graph.

   - Control of report output data (report_result[])
   - Control of graph output data (graph_result[])

3.3.2 Control parameter data (parameter[])

The list of parameters to be changed during a calculation is inputted into the control parameter section. Each control parameter is inputted by the following data structures.

- name : ... The name of a parameter to be changed.
  The value is updated if it is the existing parameter. The parameter is registered when a new parameter is inputted.

- value[] : ... The value to be changed (array).
  Muffin can give two or more values in an array to the parameter of one name.

3.3.3 Analysis result UDF control data (summary_control)

The control data for the analysis-result output file (summaryUDF) is inputted.

- report_result[] : ... The list of analysis-result data names which carries out report output.

  - name : ... The analysis-result data name which carries out report output.
    The data to be used as the candidate for report output are analysis-result data (data outputted to the analysis-result data (result[]) of Output UDF), or an input parameter (the solver parameter, a common physical parameter, physical parameter of Input UDF). The report of an input parameter is useful for confirming whether the parameter is normally changed when the scheduling of a parameter is being performed.

- graph_result[] : ... The list of analysis-result data which carries out graph output.

  - name : ... The analysis-result data name which carries out graph output.
  - mode : ... The mode of the graphical representation range ("auto_range" or "manual_range").
    When "auto_range" is chosen, a range is determined automatically from the maximum and the minimum value of a data sequence, and when "manual_range" is chosen, it is necessary to specifying the range manually. When nothing is chosen, it is regarded as "auto_range."
  - manual_range : ... Range specification when the mode of the graphical representation range is "manual_range."

    * min : ... The minimum value of the axis in graphical representation.
    * max : ... The maximum value of the axis in graphical representation.

3.4 Analysis result output UDF

Here, the data structure of an analysis-result output file (it is specified by "-S *summaryUDF*" option) is explained. Since it corresponds to a display in the engine control of GOURMET and is fundamentally common to every engine, this data structure is explained briefly.

In Muffin, basically, if there are report data and graph data which were specified in the control parameter input UDF, this will be outputted.

When the applicable data of the control parameter input UDF are empty, or when the control parameter input UDF itself is not specified by -P option argument, all analysis-result data are outputted. In this case, an input parameter does not serve as a candidate for output.

- report_attribute : ... The attribute of report data.
  - location[] : ... The array of the UDF location of report data.
  - label[] : ... The array of the label of report data.
- graph_attribute : ... The attribute of graph data.
  - title : ... The title of a graph.
  - location[] : ... The array of the UDF location of graph data.
  - label[] : ... The array of the label of graph data.
  - scale[] : ... The range of the axis in graphical representation.
    - min : ... The minimum value of the axis in graphical representation.
    - max : ... The maximum value of the axis in graphical representation.
- report_data : ... Report data.
  - value[] : ... The array of the value of report data.
- graph_data : ... Graph data.
  - item[] : ... The array of the item of graph data.
    - value[] : ... The time series array of the graph data of a certain item.

Chapter 4

MILK – a generator of unstructured mesh for MUFFIN

The MILK generates the unstructured lattice mesh in UDF form. A three-dimensional Delaunay mesh and a two-dimensional Delaunay mesh are used for the formation of a unstructured lattice.

4.1 Generation external shape

The mesh type currently supported by MILK is shown below. The type of the mesh is specified by the mesh parameter (parameter.mesh_parameter.type_of_mesh) as one of the following.

- **UNSTRUCTURED_RECT** ... Unstructured mesh (the shape is square or rectangular parallelepiped)
- **UNSTRUCTURED_SPHERE** ... Unstructured mesh (the shape is a circle or a sphere)

4.1.1 Description of mesh parameters

For each mesh type, parameters axes[] (the axes of a mesh), periodic[] (periodicity) have the following meanings.

- **UNSTRUCTURED_RECT**
  - axes[0] ... X-axis, axes[1] ... Y-axis, axes[2] ... Z-axis (3D case)
  - periodic[0] ... periodicity of X-axis, periodic[1] ... periodicity of Y-axis, periodic[2] ... periodicity of Z-axis (3D case)
- **UNSTRUCTURED_SPHERE**
  - axes[0] ... r-axis, axes[1] ... θ-axis, axes[2] ... ϕ-axis (3D case)
  - periodic[0],periodic[1],periodic[2] ... always 0 (false)

4.1.2 Partial regions generated by MILK

The partial regions (aggregate of mesh elements) which are generated for each mesh type are as follows.

- The commonly generated partial regions for **UNSTRUCTURED_RECT**, and **UNSTRUCTURED_SPHERE**.
  - "ALL_[VERTEX,EDGE,FACE,CELL]" ... All [vertices, edges, faces and cells].
  - "ALL_BOUNDARY_[VERTEX,EDGE,FACE,CELL]" ... All boundary [vertices, edges, faces and cells].
- In the case of **UNSTRUCTURED_RECT**.

  - "BOUNDARY_VERTEX_[XMIN,XMAX,YMIN,YMAX,ZMIN,ZMAX]"
    ... [Vertices on the base perpendicular to X-axis (X-axis non-periodic), Vertices on the top face perpendicular to X-axis (X-axis non-periodic),
    Vertices on the base perpendicular to Y-axis (Y-axis non-periodic), Vertices on the top face perpendicular to Y-axis (Y-axis non-periodic),
    Vertices on the base perpendicular to Z-axis (Z-axis non-periodic), Vertices on the top face perpendicular to Z-axis (Z-axis non-periodic)]
  - "BOUNDARY_FACE_[XMIN,XMAX,YMIN,YMAX,ZMIN,ZMAX]"
    ... [Faces on the base perpendicular to X-axis (X-axis non-periodic), Faces on the top face perpendicular to X-axis (X-axis non-periodic),
    Faces on the base perpendicular to Y-axis (Y-axis non-periodic), Faces on the top face perpendicular to Y-axis (Y-axis non-periodic),
    Faces on the base perpendicular to Z-axis (Z-axis non-periodic), Faces on the top face perpendicular to Z-axis (Z-axis non-periodic)]

- In the case of **UNSTRUCTURED_SPHERE**
  - "CENTER_VERTEX" ... A main node (when there is no inner spherical surface)
  - "BOUNDARY_VERTEX_[TOP,BOTTOM]_CENTER" ... Center points on [top and bottom] surface.
  - "BOUNDARY_VERTEX_[OUTER,INNER]_SPHERE" ... Vertices on the [outside and inside] spherical surface
  - "BOUNDARY_VERTEX_[TOP,BOTTOM]" ... Vertices on [top and bottom] surface.
  - "BOUNDARY_VERTEX_[TOP,BOTTOM]_[OUTER,INNER]_SPHERE" ... Vertices on cross line of [outside and inside] spherical surface and [top and bottom] surface.
  - "BOUNDARY_FACE_[OUTER,INNER]_SPHERE" ... Faces on the [outside and inside] spherical surface
  - "BOUNDARY_FACE_[TOP,BOTTOM]" ... Faces on [top and bottom] surface.

4.2 Generation of internal structures

MILK can generate two or more internal structures to the interior of the possible external shapes described previously.

- **SPHERE** ... Gives the shape of a sphere for three dimensions.
- **CIRCLE** ... Gives the shape of a circle for two dimensions.
- **RECT** ... Gives the shape of a rectangular parallelepiped for three dimensions and a rectangle for two dimensions.

The inner side of these internal structures can be a cavity, or filled with meshes.

Using to these structures, the (electrolyte) fluid simulation which has two or more structures of the shape of a sphere or a rectangular parallelepiped in the interior region can be performed using PhaseSeparation_FEM or Electrolyte_FEM. And, foaming rubbers and hollow cylinder elastic rubbers or gels with two or more cavities, etc. can be calculated using Elastica or GelDyna.

A mesh which has the spherical structure into the interior of the external shape of a rectangular parallelepiped is shown in fig.4.1.

4.2.1 The description of internal structure parameters

The information of internal structures can be given as solver parameters or physical parameters. Parameters to give internal structures are as follows.

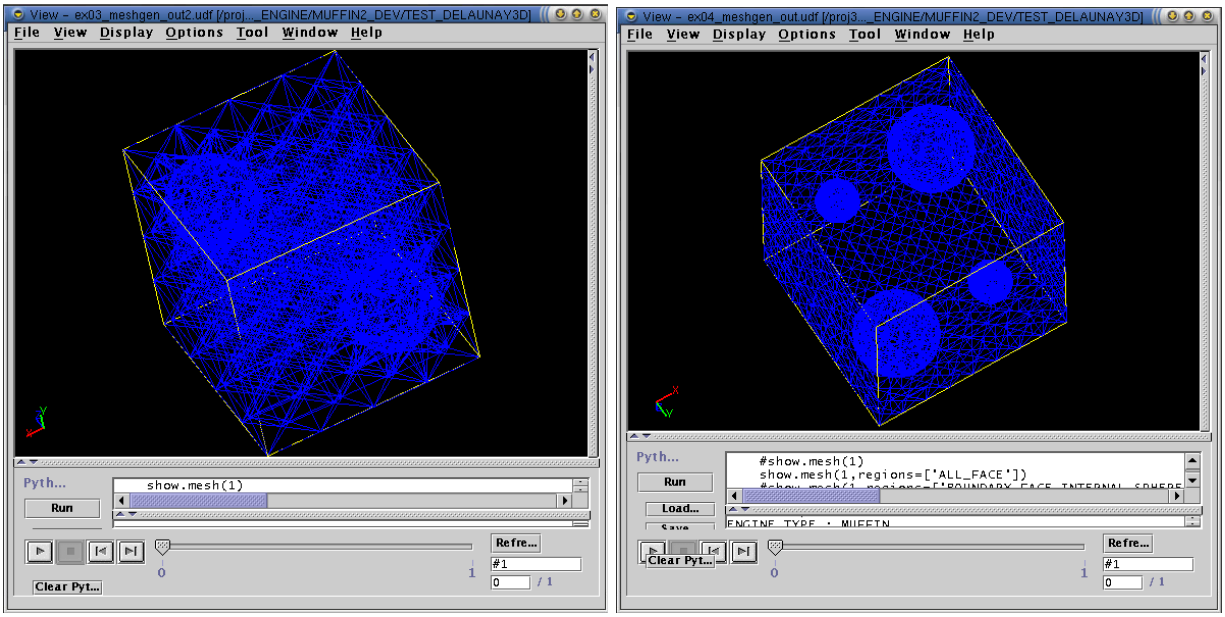

Figure 4.1: Mesh which has a sphere structure in an unstructured lattice of a rectangular parallelepiped external shape. Tetrahedron cells are put in two spheres (left). The boundary triangles of the interior are put in four spheres, and the surface (right).

name of parameter	meaning of parameter
INTERNAL_STRUCTURE	"1"(true) or "0"(false). The flag showing the existence of an internal structure.
NUM_OF_INTERNAL_STRUCTURE	The number of internal structure objects (zero or more integers)
INTERNAL_STRUCTURE_*num*	The parameter array of each internal structure object ($num = 0, 1, 2, ...NUM\_OF\_INTERNAL\_STRUCTURE - 1$)

Give parameter array elements of INTERNAL_STRUCTURE_*num* as follows.

- value[0] ... "RECT" or "SPHERE" or "CIRCLE"
- value[1] ... "1"(fill) or "0"(empty)
  - In the case of "RECT"(3D).
    - value[2] ... minimum value of X-axis. value[3] ... minimum value of Y-axis. value[4] ... minimum value of Z-axis.
    - value[5] ... maximum value of X-axis. value[6] ... maximum value of Y-axis. value[7] ... maximum value of Z-axis.
    - value[8] ... division number of X-axis. value[9] ... division number of Y-axis. value[10] ... division number of Z-axis.
  - In the case of "RECT"(2D).
    - value[2] ... minimum value of X-axis. value[3] ... minimum value of Y-axis.
    - value[5] ... maximum value of X-axis. value[6] ... maximum value of Y-axis.
    - value[8] ... division number of X-axis. value[9] ... division number of Y-axis.
  - In the case of "SPHERE".
    - value[2] ... X-coordinate of the center point. value[3] ... Y-coordinate of the center point. value[4] ... Z-coordinate of the center point.
    - value[5] ... radius.

    - value[6] ... division number of radius.
    - value[7] ... division number of $theta(0 \sim \pi)$ next the center. value[8] ... division number of ϕ next the pole.
  - In the case of "CIRCLE".
    - value[2] ... X-coordinate of the center point. value[3] ... Y-coordinate of the center point.
    - value[5] ... radius.
    - value[6] ... division number of radius (r).
    - value[7] ... division number of $theta(0 \sim 2\pi)$ next the center.

4.2.2 Internal partial regions generated by MILK

A partial region created as an interior region of a mesh structure and the surface of an internal structure have the names which contain the character string of above-mentioned "INTERNAL_STRUCTURE_*num*" to show that it is an internal structure object. The following description explains by replacing the identifier "INTERNAL_STRUCTURE_*num*" with "XXXX".
Notice)

- "ALL_[VERTEX,EDGE,FACE,CELL]_XXXX" ... All the elements contained in an internal structure object are contained in "ALL_[VERTEX, EDGE, FACE, CELL]".
- "ALL_BOUNDARY_[VERTEX,EDGE,FACE]_XXXX" ... All the boundary elements of an internal structure object are contained in "ALL_BOUNDARY_[VERTEX, EDGE, FACE]".
- Common to "RECT","SPHERE" and "CIRCLE".
  - "ALL_VERTEX_XXXX"
    ... All the vertices that are contained in an internal structure object.
- In the case of "RECT".
  - "ALL_CELL_XXXX"
    ... All the cells that are contained in an internal structure object.
  - "BOUNDARY_VERTEX_[XMIN,XMAX,YMIN,YMAX,ZMIN,ZMAX]_XXXX"
    ... Vertices of [the base perpendicular to X-axis, the top surface perpendicular to X-axis, the base perpendicular to Y-axis, the top surface perpendicular to Y-axis, the base perpendicular to Z-axis, the top surface perpendicular to Z-axis] of the internal structure object XXXX.
  - "BOUNDARY_FACE_[XMIN,XMAX,YMIN,YMAX,ZMIN,ZMAX]_XXXX"
    ... Faces of [the base perpendicular to X-axis, the top surface perpendicular to X-axis, the base perpendicular to Y-axis, the top surface perpendicular to Y-axis, the base perpendicular to Z-axis, the top surface perpendicular to Z-axis] of the internal structure object XXXX.
- In the case of "SPHERE".
  - "ALL_CELL_XXXX"
    ... All the cells that are contained in an internal structure object.
  - "CENTER_VERTEX_XXXX" ... The center vertex of the internal structure object.
  - "BOUNDARY_VERTEX_XXXX" ... Vertices on the spherical surface of the internal structure object.
  - "BOUNDARY_VERTEX_XXXX_*num*" ... Vertices of the partial region on the spherical surface of the internal structure object.
    $num = 00, 01, 02, 03, 10, 11, 12, 13$ correspond to the following partial regions respectively, the first, second, third, 4th, 5th, 6th, 7th and 8th quadrant.
  - "BOUNDARY_FACE_XXXX" ... Faces on the spherical surface of the internal structure object.
  - "BOUNDARY_FACE_XXXX_*num*" ... Faces of the partial region on the spherical surface of the internal structure object.
    $num = 00, 01, 02, 03, 10, 11, 12, 13$ correspond to the following partial regions respectively, the first, second, third, 4th, 5th, 6th, 7th and 8th quadrant.

- In the case of “CIRCLE”.
  - “ALL_FACE_XXXX”
    . . . All the faces that are contained in an internal structure object.
  - “CENTER_VERTEX_XXXX” . . . Vertices on a medial axis of the internal structure object.
  - “BOUNDARY_VERTEX_XXXX” . . . Vertices on the surface of the internal structure object.
  - “BOUNDARY_EDGE_XXXX” . . . Edges on the surface of the internal structure object.

4.3 Generation of input UDF by MILK

Command to generate mesh data by using MILK

Usage : % milk3d5(milk2d5) -I *input_for_MILK.udf* -O *output_mesh_data.udf*

Copying generated mesh data into a UDF for a simulation

The mesh generated by MILK can be copied into other UDF files by the action ”export_mesh”. An input UDF file can be created by this script. The type of mesh (parameter.mesh_parameter.type_of_mesh) can be input by an engine as UNSTRUCTURED_INPUT. For details, refer to “5.3 MeshFieldConvertor - the mesh and field UDF converter”.

Chapter 5

Pre-processor

5.1 Modeler – the input UDF generator for Elastica and Milk

The modeler for Elastica and Milk are prepared in "modeler.Elastica" and "modeler.Milk" part of UDF definition. After input modeler UDF, by running the action "run_modeler", "run_elastica_modeler",..., the input UDF file is generated. Furthermore, the action "run_modeler_and_engine", "run_elastica_modeler_and_engine", "run_milk_modeler_and_engine", ..., make the input UDF, save them, run the engine, and open the output UDF automatically.

When using Elastica, if you'd like to import morphology and mesh data from another UDF file, import field and mesh using the action "import_field" and set "import" in "modeler.Elastica.morphology.type_of_input" of modeler UDF. And, if you'd like to import only the mesh data from another UDF file such as output UDF of Milk, import mesh using the action "import_mesh" and set "input" in "modeler.Elastica.mesh.type" of modeler UDF.

5.2 MuffinMujigen – a program to get dimensionless parameters for MUFFIN

The MuffinMujigen.py is a Python script to get dimensionless parameters for MUFFIN engines. The Users should understand the theoretical background of engines such as PhaseSeparation to use this utility. Please refer chapters for theories of MUFFIN engines about the detail of dimensionless parameters.

5.2.1 The description of functions of non-dimensionized program

A non-dimension-ized program is a program which generates an output UDF file which read an input UDF file in which the parameters are converted from, for example, the MKSA system to units to dimensionless ones. Since, as for an output UDF file, an input UDF file is copied and a parameter portion is transformed further, the former data of input are not destroyed. The overview of the flow of processing is as follows.

1. The basic parameters are read in the input UDF file, and PhaseSeparation or Electrolyte simulation is judged.
2. The non-dimensionized unit are calculated from a basic parameter.
3. An input UDF file is copied and an output UDF file is created.
4. All the parameters for the non-dimensionized transformation turn dimensionless, and are outputted.

The interface of the non-dimensionized program

Usage : % python MuffinMujigen.py input_dim.udf output_nondim.udf

5.3 MeshFieldConvertor – a mesh and field UDF convertor

The MeshFieldConvertor is a Python script which performs analysis, a data conversion, etc. between the UDF file of two or more mesh data or field data. In the present version, it has the data conversion function of the mesh and field data from ones of SUSHI to ones for MUFFIN, from ones of MUFFIN to ones for SUSHI, or between ones of SUSHI, and between data of MUFFIN.

5.3.1 The function of MeshFieldConvertor

1. The engine type is automatically judged by the header data of UDF's.
2. The used type of a mesh is judged automatically.
3. All the data of the field in UDF are searched automatically.
4. The correspondence relation of fields between MUFFIN and SUSHI, is automatically recognized. It has a thesaurus of the name of the field currently used in each engine. For example, "phi" of SUSHI corresponds to "VolumeFraction" of MUFFIN.
5. The data of fields which correspond in the thesaurus are automatically converted according to the mesh type.
6. The conversion of the mesh data.

5.3.2 The interface of MeshFieldConvertor

Usage:
% python MeshFieldConvertor.py -i *inputUDF* { -ir *input_record_No.* } -o *outputUDF* { -or *output_record_No.* } { -e *execute_program* }

- *inputUDF* . . . The file name of an input UDF file.
- *input_record_No.* . . . The record number of the input UDF file. (default,-1)
- *outputUDF* . . . The file name of an output UDF file.
- *output_record_No.* . . . The record number of the output UDF file. (default,-1)
- *execute_program* . . . The program to perform. (default,"convert".)
  Currently, `-e convert` and `-e convert_mesh` can be used.

5.3.3 Zoom in/out of the field data between UDF(s) of SUSHI and MUFFIN

The data of a mesh parameter and a field are read from an output UDF of SUSHI. How to build the input UDF of MUFFIN is explained in the following.

1. When you use the Python program directly from the comamnd line, see the usage described previously.
2. On GOURMET:
   Edit the name and record number of input UDF file on MeshFieldConvertor.py and run.

```
....
import muffinlib.MultiMeshFieldAnalysisLib
# The name of input UDF file
import_udf_path="/home/yamaue/...../cylinder3D_b_uot.udf"
# The record number of input UDF file ( -1 means the common data ).
import_udf_record=1
....
```

5.3.4 Zooming in/out of the field data by action operation

Zooming out of the field data from SUSHI

On GOURMET, when you click the right button of a mouse on the UDF path (field) of the data of fields of a UDF data of MUFFIN, a "zoom_out_from_sushi()" menu will appear. By selecting this you can convert the mesh and field data. At the time of run, the dialog opens, which asks an input UDF path and the record number of a SUSHI UDF.

Zooming in of the field data to SUSHI

On GOURMET, when you click the right button of a mouse on the UDF path (field) of the data of fields of a UDF data of MUFFIN, a "zoom_in_to_sushi()" menu will appear. By selecting you can convert the mesh and field data. At the time of run, the dialog opens, which asks an output udf path and the record number of a SUSHI UDF.

5.4 Conversion of mesh data from NASTRAN BULK file to UDF

It is possible to convert mesh data of the NASTRAN BULK file format into the mesh data of UDF data of MUFFIN (a type of common mesh data format for MUFFIN and SUSHI) by using the file converter function of GOURMET and a Python script specially programmed for MUFFIN.

The procedure of the file conversion consists of the following two steps.

1. Create the mesh data of common UDF form by the file converter of GOURMET.
2. Generate partial regions for the converted mesh data by a python program (nastran/MakePartialRegions.py).

5.4.1 Mesh data conversion on GOURMET

The conversion to the common UDF mesh data on GOURMET

1. Select "ConvertFile..." from the "File", and a "ConvertFile" dialog menu opens.
2. Input a mesh data file in the NASTRAN BULK file format on the "Data File:" text field of the menu.
3. Choose a file "$PF_ENGINE/python/nastran/nastranbulkrule.txt", which is the filter to convert NASTRAN BULK format into common UDF form, for "Rule File:" text field of the menu.
4. Choose a file "$PF_ENGINE/def_udf/muffin3.udf", which is the UDF definition part of MUFFIN3, for "Input UDF:" text field of the menu.
5. Input the Output UDF file name in "Output UDF:" text field of the menu.
6. Push "O.K." and the conversion is performed.

The partial region generation program execution on GOURMET

1. Open the mesh data converted into the common UDF form on GOURMET.
2. Open ("Load") the python program "$PF_ENGINE/python/nastran/MakePartialRegions.py" for the partial region generation and excute it ("Run").
3. The mesh data is edited by attaching the suitable name for the partial region name *etc* and saved.

5.4.2 The conversion program execution on console

You can convert a mesh data from of NASTRAN BULK format into MUFFIN common UDF form from a console (MS-DOS window or UNIX terminal) using a Python program for conversion "$PF_ENGINE/python/nastran/filefil
Usage: python filefilter.py *bulkdata.dat outdata.udf* [*def.udf*]

- *bulkdata.dat* ... The file name of a NASTRAN BULK file data.

- *outdata.udf* . . . The file name path of an output UDF file.
- *def.udf* . . . (optional) The definition part of the UDF data.
  It is set to "muffin3.udf" by default.

The mesh data of UDF form are opened by GOURMET after conversion-program run, and it is edited by attaching the suitable name for the partial region name *etc* and saved.

Chapter 6

Post-processor

6.1 MeshFieldPlot - an analysis and plot tool for MUFFIN

A Python script MeshFieldPlot.py includes a plot program of the spatial distribution and an analysis program of the spatial correlation function of a field.

By running MeshFieldPlot.py in the Java Python window of GOURMET, you can analyze the space distribution data of the section of the field of common UDF or the surface and create a GraphSheet, and in the Plot window of GOURMET you can plot the data using gnuplot.

MeshFieldPlot.py also has an interface to the spatial correlation functional-analysis program "spcf" of SUSHI, and you can plot the data by gnuplot in the Plot window of GOURMET.

6.1.1 Function of MeshFieldPlot.py

1. Data extraction and a GraphSheet creation on a cross section of the spatial distribution of a field data.
2. Data extraction and GraphSheet creation on a surface of the spatial distribution of a field data.
3. Data extraction and GraphSheet creation on a line segment of the spatial distribution of a field data.
4. The spatial correlation analysis of a field and a GraphSheet creation using spcf of SUSHI

6.1.2 Interface of MeshFieldPlot.py

1. Interface
   (a) Constructor : MeshFieldPlot()
   Creation of MeshFieldPlot object.
   (b) Method : field(name,component = 0,region = '', cplane=[], cline=[])
   A spatial distribution data analysis is carried out on a cross section, a surface, or a line segment of a field. Generate a GraphSheet.
   Argument description
   - name ... The name of a field (can not omit)
   - component ... The number of the component of the field to be analyzed (the default value is 0)
   - region ... The region name of a section or the surface to be analyzed (the default value is ''). Applicable for structured lattices (REGULAR, RECTANGULAR) and all the unstructured lattices.
   - cplane ... Specification of the cross section and/or the surface to be analyzed (the default value is []).
     Applicable for a structured lattice (REGULAR, RECTANGULAR) and an unstructured lattice (UNSTRUCTURED_RECT). The data is given as cplane=[[point on plane],[normal vector]].
     ex.) cplane=[[0,0,0],[0,0,1]] : A cross section that passes the origin and is a face (XY plane) perpendicular to the Z-axis.

- cline ... Specification of the straight line to analyze (the default value is []).
  Applicable for structured lattices (REGULAR, RECTANGULAR) and the unstructured lattice (UNSTRUCTURED_RECT). The data is given as cline=[[point on line],[direction vector]].
  ex.) cline=[[0,0,0],[0,0,1]] : a line that passes the origin and the direction is parallel to the Z-axis.

(c) Method : spcf(name,component = 0,region = '',cplane=[],direction=[],minmax=[])
An input file of spcf of SUSHI for analyzing the spatial correlation function of a field.
Argument description

- name ... The name of a field (Indispensable)
- component ... The number of the component of the field to be analyzed (the default value is 0)
- region ... The region name of a cross section or a surface to be analyzed (a default value, '').
  Applicable for structured lattices (REGULAR, RECTANGULAR) and all the unstructured lattices.
- cplane ... Specification of the section and the surface to be analyzed (the default value is []).
  Applicable for a structured lattice (REGULAR, RECTANGULAR) and an unstructured lattice (UNSTRUCTURED_RECT). The data is given as cplane=[[point on plane],[normal vector]].
  ex.) cplane=[[0,0,0],[0,0,1]] : A cross section that passes the origin and is a face (XY plane) perpendicular to the Z-axis.
- direction ... The direction vector in the case of calculating the correlation function about the specific direction.
  A default value is [] and an isotropic correlation function is calculated in this case.
  ex.) direction=[0,0,1] : Correlation of a direction parallel to the Z-axis is calculated.
- minmax ... Data clipping specification. Give a minimum value and a maximum value.
  A default value is [] and no clipping.

2. Usage

(a) Analysis on a cross section, a surface, and a straight line for spatial distribution data of a field

Python script

```
## Package MeshFieldPlot is imported.
from MeshFieldPlot import *
## Construction of a MeshFieldPlot object.
plot = MeshFieldPlot()
## About scalar field 'Concentration' of a structured lattice,

## cross-sectional data is analyzed by cplane specification.
#plot.field('Concentration',cplane=[ [0.,0.,0.],[0.,0.,1.] ] )
## About scalar field 'Concentration' of a structured lattice,
## surface data is analyzed by region specification.
plot.field('FreeEnergy',region='YMIN' )
## About vector field 'Displacement' of an unstructured lattice,
## surface data is analyzed by region specification.
#plot.field('Displacement',region='ZMIN' )
## About scalar field 'VolumeFraction' of an unstructured lattice,
## data is analyzed by straight line specification.
#plot.field('VolumeFraction',cline=[ [0.,0.,0.],[0.,0.,1.] ] )
```

You can also load the MeshFieldPlot.py on the Python window of GOURMET, edit directly the main function of the end of the script.

(b) The spatial correlation functional analysis of a field

Python script

```
## Package MeshFieldPlot is imported.
from MeshFieldPlot import *
## Construction of a MeshFieldPlot object.
plot = MeshFieldPlot()
## About scalar field 'FreeEnergy' of an unstructured lattice,
## a space correlation function is analyzed.
plot.spcf('FreeEnergy')
```

You can also load the MeshFieldPlot.py on the Python window of GOURMET, edit directly the main function of the end of the script.

6.2 MeshFieldShow - a drawing tool for MUFFIN

A Python script MeshFieldShow.py is a program which draws a mesh and a field of common UDF.

6.2.1 Function of MeshFieldShow.py

1. Drawing of the color contour on a cross section or a surface for a scalar field.
2. Drawing of the vector on the cross section and the surface for a vector field
3. The type of fields, such as a scalar field and a vector field, is judged automatically.
4. An engine type (SUSHI, MUFFIN) is judged from UDF header data, and it operates on UDF of each different engine. It can be used with a common interface.

6.2.2 Interface of MeshFieldShow.py

1. Interface

   (a) Constructor : MeshFieldShow()
   Creation of MeshFieldShow object.

   (b) Method : field(name,component=0,regions=[],cplanes=[],
   minmax=[0,1],color_att=1,arrow_att=[0,1,0,1,0,0], skip_att=0)
   Draw a color contour of a scalar field and the vector of a vector field on a cross section or a region (two or more specification is possible).
   Argument description

   - name ... The name of a field (can not omit)
   - component ... The number of the component of the field to be drawn (the default value is 0)
   - regions ... The region name of a section or the surface to be drawn (the default value is ”). Applicable for structured lattices (REGULAR, RECTANGULAR) and all the unstructured lattices.
     The data are given as regions=['region_name_1', 'region_name_2', ... 'region_name_n']
     Specification of two or more regions is possible.
     ex.) regions=['ZMIN','ZMID','ZMAX'] : draw on three cross sections perpendicular to the Z-axis.
   - cplanes ... Specification of the section and the surface to be drawn (a default value is []). Applicable to a structures lattice (REGULAR, RECTANGULAR) and an unstructured lattice (UNSTRUCTURED_RECT). The data are given as cplanes=[[[point on plane],[normal vector]], ..., [[point on plane],[normal vector]]].
     Specification of two or more sections is possible.

ex.) cplanes=[[[0,0,0],[0,0,1]],[[0,0,0],[0,1,0]]] : draw on XY plane and XZ plane including the origin.

- minmax ... The minimum value and maximum of a scalar field (the default is [0,1]).
- color_att ... Specification of the color attribute of a scalar field (the default is 1).
- arrow_att ... Specification of the vector attribute of a vector field.
  The data are given as arrow_att=[R(ED),G(REEN),B(LUE),T(RANSPARENCY), radius of arrowhead,length of arrowhead]
- skip_att ... The number of skips of the peak in a structured lattice to be drawn (the default is 0).

(c) Method : mesh(color_att=1,regions=[])
Draw mesh lattices of specified regions. Argument description

- color_att ... Specification of the color attribute of the line to draw a lattice (the default is 1).
- regions ... The region name of a cross section or the surface in which a lattice (a default value, ['ALL_CELL']) is drawn.
  Applicable for structures lattices (REGULAR, RECTANGULAR) and all the unstructured lattices.
  The data are given as regions=['region_name_1','region_name_2',...'region_name_n'].
  Specification of two or more regions is possible.
  ex.) regions=['ZMIN','ZMID','ZMAX'] : draw on three cross sections perpendicular to the Z-axis.

(d) Method : frame(color_att=0)
Draw the frame of a spatial region.
Argument description

- color_att ... Specification of the color attribute of the line to draw a frame (the default value is 1).

2. Usage

Python script

```
## Package MeshFieldShow is imported.
from MeshFieldShow import *
## Construction of a MeshFieldShow object.
show = MeshFieldShow()

## Scalar field 'phi' of the structured lattice
## of SUSHI is drawn by cplane specification.
#show.field('phi',cplanes=[[[0.,0.,0.],[0,0,1]],[[0.,0.,0.],[0,1,0]]])
## Scalar field 'Concentration' of the structured lattice
## of MUFFIN is drawn by cplane specification.
#plot.field('Concentration',cplane=[ [0.,0.,0.],[0.,0.,1.] ] )
## Scalar field 'VolumeFraction' of an unstructured lattice is drawn
## by surface region specification.
#show.field('VolumeFraction', regions=[ 'ALL_FACE' ])
## Vector field 'Displacement' of an unstructured lattice is drawn
## by surface region specification.
#show.field('Displacement',regions=[ 'ALL_FACE' ] )
## Scalar field 'FreeEnergy' of an unstructured lattice is drawn
## by cross-sectional specification of the surface and an inside.
#show.field('FreeEnergy',regions=['ZMIN','ZMID','ZMAX'],minmax=[0.0,1.5])
```

You can also load the MeshFieldShow.py on the Python window of GOURMET, edit directly the main function of the end of the script.

6.3 udf2avs - a convertor from UDF to the AVS format

This is a Python program which transforms mesh and field data of the output UDF file from MUFFIN engine into the form which can be displayed by the AVS (Application Visualization System).

Usage

python muffin3_udf2avs.py -d *AVS-file-output-directory* *UDF-file*

The directory to output an AVS file can be specified by the execution-time option "-d *AVS-file-output-directory*". The directory "AVS" is used when this option is omitted.

Generated files

The following files are areated in the directory *AVS-file-output-directory*.

- **In the cases of output UDFs of the finite element method simulators (PhaseSeparation_FEM, Electrolyte_FEM, MEMFluid, Elastica, GelDyna):**

  Files of the UCD (Unstructured Cell Data) form of AVS are created with the following names for every output data of the field included in each record of UDF.

  <*Field-name*><*time-step-number*>`.inp`

  Here, <*Field-name*> is the name of fields, such as Velocity and VolumeFraction, and <*step-number*> is the step number to which data were outputted. The number is an integer in 6 digits like "000100" and "012000".

- **In the cases of output UDFs of the finite difference method simulators (PhaseSeparation_FDM, Electrolyte_FDM):**

  The file of field file form and the main part file of data of AVS are built with the following names for every output data of the field included in each record of UDF.

  <*Field-name*><*step-number*>`.fld`
  <*Field-name*><*step-number*>`.dat`

  `*.fld` are files of the field file form of AVS and `*.dat` is the ASCII format data file which contains data. <*Field-name*> is the name of fields, such as Velocity and VolumeFraction, and <*step-number*> is the step number to which data were outputted, The number is an integer in 6 digits like "000100" and "012000".

Appendix A

Compiling MUFFIN

A.1 Setting of environment variables

1. The value of PF_ENGINE is set as the top directory to which engines are installed.
2. The value of PF_FILES is set as the top directory to which GOURMET is installed. An engine interface library "libplatform" under the directory is linked to engine binaries.

A.2 Compilation of MUFFIN

compiling in unix, linux, and cygwin environment

In the source directory of "MUFFIN5/src" in which sources the engines are placed, compile using Makefile.

% cd $(PF_ENGINE)/MUFFIN5/src/muffin5 or muffin5ebeta
% make

Appendix B

System extension guide

B.1 Basic class libraries of MUFFIN

The source codes of MUFFIN base-class library are in below MUFFIN3/src/common/ directory. The main header files and functional descriptions are as follows.

DYNAMICS/DynamicsManager.h	DynamicsManager
DYNAMICS/PartialRegionCondition.h	Partial Region (Boundary) condition
FIELD/FDM_SuperField.h	Base class for FDM field.
FIELD/SuperField.h	Base class for FEM field.
IO/IOPath.h	Data of Input and Output file path.
IO/SuperStream.h	Output field data with AVS form.
IO/PFIO/Imuffin3.h	UDF Interface object of MUFFIN3.
IO/PFIO/Imuffin3param.h	UDF Interface object of parameter file of MUFFIN3.
IO/PFIO/Imuffin3result.h	UDF Interface object of summary file of MUFFIN3.
IO/PFIO/PFIO.h	Manager of IO between UDF Interface object and MUFFIN.
MESH/CreateMesh.h	Mesh Creator with various shapes.
MESH/ElementManager.h	Manager of creation, delete and collect mesh elements.
MESH/RegisterVertices.h	Creator of vertices in Delaunay triangulation.
MESH/RightTriangulation3D.h	Mesh class (3D unstructured right triangulation).
MESH/SuperElement.h	Base class of mesh elements.
MESH/SuperMesh.h	Base class of mesh classes.
MESH/TriangleElement.h	Mesh element class (triangle)
MESH/delaunaytriangulation.h	Mesh class (2D Delaunay triangulation)
MESH/delaunaytriangulation3d.h	Mesh class (3D Delaunay triangulation)
PARAM/IOParameterSet.h	Manager of Parameter and Result data
SOLVER/FEMTetra.h	Finite element class (tetrahedron).
SOLVER/FEMTriangle.h	Finite element class (triangle).
SOLVER/FiniteElement.h	Base class of finite element classes.
SOLVER/MATRIX/cg.h	CG and ICCG solver.
UTIL/CheckMemorySize.h	memory check function.
UTIL/MotherErr.h	Base class for error classes.
UTIL/ProgramTimer.h	cpu-time check function.
UTIL/StringHelperFunc.h	Utility class for string.
UTIL/Sym2ndRankTensor.h	data type class (symmetric 2nd-rank tensor)
UTIL/Vector3d.h	data type class (vector)
UTIL/RND/rng.h	Base class of random number generator classes.
UTIL/RND/twister.h	random number generator class (MT).
UTIL/DOCMAKER/muffin3docmaker.cpp	Source analyzer and Document maker for MUFFIN3.
UTIL/DOCMAKER/template_euc.tex	Template file of output file for Document maker.
SCRIPT/gxxversion.sh	Version check of compiler.

```
-------------------------------------------------------------------------------
```

It is SuperField.h that is necessary for an engine development by generation of a class of a new field. It will be shown that a generation of the class of a new field can be easily performed by inheriting classes in this file.

It is better to understand SuperMesh.h and SuperElement(ElementManager.h).h further, when complicated operation is desired and formation and analysis of the aggregate of arbitrary mesh elements are carried out.

B.2 Compile options of basic class libraries of MUFFIN

When creating a new simulator, it is useful to reuse the following files, such as a main function of the new simulator, by a little corrections.

```
-------------------------------------------------------------------------------
MUFFIN3/src/Elastica/Makefile
MUFFIN3/src/Elastica/MUFFIN3.cpp       main function
MUFFIN3/src/Elastica/Global_Variable.h header of main function
MUFFIN3/src/Elastica/Fields/Fields.h   header of fields
-------------------------------------------------------------------------------
```

Here, we describe a case that a simulator named MyDynamics with the following fields is created.

- AField
- BField
- CField

A source code is built as follows. For Makefile, MUFFIN3.cpp, Global_Variable.h, and Fields.h, copy the above-mentioned files.

```
-------------------------------------------------------------------------------
MUFFIN3/src/MyDynamics/Makefile
MUFFIN3/src/MyDynamics/MUFFIN3.cpp       main function
MUFFIN3/src/MyDynamics/Global_Variable.h header of main function
MUFFIN3/src/MyDynamics/Fields/Fields.h   header of fields
MUFFIN3/src/MyDynamics/Fields/AField.h (and .cpp)
MUFFIN3/src/MyDynamics/Fields/BField.h (and .cpp)
MUFFIN3/src/MyDynamics/Fields/CField.h (and .cpp)
-------------------------------------------------------------------------------
```

Add the following lines to Makefile.

```
-------------------------------------------------------------------------------
###     For MUFFIN3 MyDynamics  ###
ifeq "$(THIS_TARGET)" "MyDynamics"
  #---------- In the case of FEM.
  #override MACRO += -D__UNSTRUCTURED_MESH__  -D__LAGRANGE_PICTURE__
  override MACRO += -D__UNSTRUCTURED_MESH__  -D__EULER_PICTURE__
  SRC   = $(SRC001) $(SRC004) $(SRC005) $(SRC006) $(SRC007) $(SRC009) $(SRC010)
  #---------- In the case of FDM.
  #override MACRO += -D__STRUCTURED_MESH__  -D__EULER_PICTURE__ -D__FDM__
  #SRC   = $(SRC001) $(SRC005) $(SRC006) $(SRC007) $(SRC010)
  SRCM  = MUFFIN2.cpp
  SRCS  = AField.cpp BField.cpp CField.cpp
  TARGET       = mydyn3
endif
-------------------------------------------------------------------------------
```

The "Field Creation" portion of MUFFIN3.cpp should be changed to the following lines.

```
-------------------------------------------------------------------------------------
    //----------------------<<Field creation>>-----------------------
    // create fields
    AField af ;
    BField bf ;
    CField cf ;
    // Registration of fields in dynamics manager
    a_Universe.RegisterField( &af );
    a_Universe.RegisterField( &bf );
    a_Universe.RegisterField( &cf );
-------------------------------------------------------------------------------------
```

Fields.h should be changed as follows.

```
-------------------------------------------------------------------------------------
#include ‘‘AField.h’’
#include ‘‘BField.h’’
#include ‘‘CField.h’’
-------------------------------------------------------------------------------------
```

The compile option used by Makefile and its effect are described below.

B.2.1 Discretization methods and compile options

The method (the finite difference method (FDM), the finite element method (FEM), etc.) of discretization differs by each simulator, and the methods of reservation of the data of the field in the MUFFIN3 differ according to it. The method (FDM or FEM) of the discretization is specified by the compile option of simulators as follows.

- the finite difference method (FDM) . . . __FDM__
- the finite element method (FEM) . . . When there is no compile option of __FDM__.

B.2.2 Mesh types and compile options

The method of a simulation picture or discretization differs by each simulator, and the mesh type which can be used is determined according to them. The type of a mesh to be used is specified by the compile option as follows.

- Structured mesh (FDM simulator) . . . __STRUCTURED_MESH__
- Unstructured mesh (FEM simulator) . . . __UNSTRUCTURED_MESH__ Furthermore, it is possible to introduce the following meshing algorithms at a compile option.
  - 2D Delaunay triangulation . . . __DELAUNAY2D_MESH__
  - 3D Delaunay triangulation . . . __DELAUNAY3D_MESH__

By specification of these options, only a mesh generator required for each type is compiled. If there is no specification, since the compile link of the mesh generator of all types will be carried out together, module size becomes very (unnecessarily) large and a compile time also becomes long.

And only when there is a unstructured lattice __UNSTRUCTURED_MESH__ macro (there is no structure lattice __STRUCTURED_MESH__macro correctly), mesh structure data are outputted in UDF.

B.2.3 Simulation picture and compile options

Mesh structure data are outputted on UDF only with an unstructured lattice. Distribution of the output to the common or record data of these mesh structure data is dependent on the simulation picture, and is controlled by the compile option.

- Euler picture ... __EULER_PICTURE__
  All data are outputted to common data.
- Lagrange picture ... __LAGRANGE_PICTURE__
  Vertices data are outputted to record data. Edges, faces and cells data are outputted to common data.
- Adaptive mesh ... __ADAPTIVE_MESH__
  All data are outputted to record data.

If it compiles without these compile options, all mesh data will be outputted to both of common data and record data of UDF, and output-data size will become large.

The existence of the output of the neighbor element information on a mesh is controlled by __NEIGHBOR_DATA_IO__ of the compile option of a source file PFIO.cpp. When this compile option is specfied, the neighbor element data are outputted to UDF.

B.3 System Extension Tutorial

The class of a new field includes SuperField.h and is defined by inheriting Class PhysicalField. An interface which is preparing AField.h with how to write the class of a field at an example is explained below. The following is an introductory edition and explains the fundamental function about acquisition of the parameter which SuperField offers, acquisition of the data of other fields, and the conditions of a partial region (boundary) boundary condition.

```
------------------------------------------------------------------------------------------
//--------------------------------------------------------
// AField.h                                  by T.Yamaue
//--------------------------------------------------------
#ifndef _AFIELD_INCLUDED_
#define _AFIELD_INCLUDED_

#include "SuperField.h"

class AField : public PhysicalField<double> // This type of field is double.
{
public:
  // Default constructor.
  AField(void):PhysicalField<double>("AField") // Name of field.
  { SetFunctionName(); }
  virtual int Evolve00(void) // (1Step) evolution command (SOLVE:NONSENSE_SUM)
  {
    // You can get other fields by name.
    SuperField& b_field = Field("BField"); // type of BField -> double (assume)
    SuperField& c_field = Field("CField"); // type of BField -> Vector (assume)

    // You can get parameters by name.
    double param_double = Parameter().value("Double_Parameter");
    string param_string = Parameter().string_value("String_Parameter");
    bool param_bool = Parameter().bool_value("Bool_Parameter");

    // You can get mesh elements by the follow method,
    // vector<SuperElement*> SuperField::ObjList(void).
    for (vector<SuperElement*>::const_iterator i = ObjList().begin() ;
```

```
      i != ObjList().end() ; i++) {
        value(*i) // You can get this field value by value(SuperElement*).
          = param_double + b_field.value(*i)
            // You can get other fields (double type) value
            // by value(SuperElement*) method.
            + (c_field3.value(*i, OVector)).x  ;
            // You can get other fields (vector or tensor type) value
            // by value(SuperElement*, OVector/OSym2ndOrderTensor/,...) method.
    }
    return 1 ;
  }
  virtual int Evolve01(void) // (1Step) evolution command (SOLVE:NONSENSE_EMPTY)
  { return 1 ;}
  virtual int Initialize00(void) // initialization command (INITIAL:NONSENSE_EMPTY)
  { return 1 ;}

protected:
  void SetFunctionName(void)
  {
    // Register name of commands.
    SetSolverName("SOLVE:NONSENSE_SUM") ;    // name of evolve00.
    SetSolverName("SOLVE:NONSENSE_EMPTY") ;  // name of evolve01.
    SetInitializingFuncName("INITIAL:NONSENSE_EMPTY") ; // name of initialize00.
  }
};
#endif
-------------------------------------------------------------------------------
```

About a required field, the class of such a field is built and a simulator is made. Then, registration of the class of a field and registration of procedure are performed in the dynamics manager section of Input UDF. Refer to the input UDF description in detail.