Real-ESSI Simulator

Model Development and Mesh Generation Manual

Sumeet Kumar Sinha, Yuan Feng,

Hexiang Wang, Han Yang

and

Boris Jeremić

University of California, Davis, CA

Version: March 24, 2021, 23:13

http://real-essi.us/

This document is an excerpt from: http://sokocalo.engr.ucdavis.edu/~jeremic/LectureNotes/

Contents

Chapter 1

Pre Processing for Real-ESSI Simulator

(2015-2016-2017...)

1.1 Introduction

1.2 Model Development Using gmsh

1.2.1 Introduction to gmESSI

The **gmESSI**, pronounced as *[gm-ESSI]*, is a translator that converts mesh file from gmsh (a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities) to Real-ESSI DSL format. The primary aim of this program is to provide an efficient pre-processing tool to develop Finite Element (FE) models in gmsh and make them interface with various Real-ESSI functionalities. The gmESSI translator package contains the translator, sublime plugin and the manual.

The gmESSI package is available at http://sokocalo.engr.ucdavis.edu/~jeremic/lecture_notes_online_material/_Chapter_SoftwareHardware_Pre_Processing/Real-ESSI_gmESSI.tgz.

The text editor sublime plugin *[gmESSI-Tools]* can be downloaded here: http://sokocalo.engr.ucdavis.edu/~jeremic/lecture_notes_online_material/_Chapter_SoftwareHardware_Pre_Processing/fei-syntax-n-snippets.tar.gz.

Getting Started

The translator utilizes the physical and entity group concept of *Gmsh* (http://geuz.org/gmsh/doc/texinfo/gmsh.html) (Geuzaine and Remacle, 2009), which gets imprinted in the mesh ".msh" file. The translator then manipulates these groups to convert the whole mesh to ESSI commands. Thus, making physical groups is the essential, key for conversion. The Translator basically provides some strict syntax for naming these Physical Groups which provides gmESSI information about the elements or (nodes) on which the translation operates. The translator is made so general that any other FEM program can use it with little tweaks to have their own conversion tool. A quick look at some important features of the program are:

- It has a lot of predefined commands which do the conversion at the blink of an eye. These commands make it easier to define elements, boundary conditions, contacts/interfaces, fixities, loads ....
- It provides a python module "gmessi". The users can import this modulue and can extend the functional capability of gmESSI.
- The *[gmESSI-Tools]* sublime plugin makes it easy by providing syntax coloring and auto-text-completion for gmESSI commands. *[gmsh-Tools]* sublime plugin can also be installed for gmsh syntax coloring and auto-completion.
- The translator uses a *mapping.fei* file to check for its command syntax and conversion. A user can easily add a command in *mapping.fei* and it would get reflected automatically in the translation.

- It automatically optimizes the Real-ESSI tags (node, element, load) for space and time efficiency while running simulation.

Installation Process: The Translator have its dependencies on Octave (3.2 or higher), Boost(1.58 or higher), (Python 2.7 or higher). One should make sure to have them before compiling it. On Linux Ubuntu distros the dependencies can be installed as

```
sudo apt-get install liboctave-dev
# Boost version should be higher than 1.48
sudo apt-get install libboost-all-dev
sudo apt-get install python-dev
```

Installation of the gmESSI translator is easy, just follow steps below.

```
## go to folder where you want to store and build gmESSI application

## download the package from main Real-ESSI repository
## this line below should be all one line
## HOWEVER it had to be broken in two lines to be readable
## so please make a single command out of two lines below
##
#
# using curly brackets to help in checking scripts, that rely on these
# brackets being available around URL
#
wget ↩
   {http://sokocalo.engr.ucdavis.edu/~jeremic/lecture_notes_online_material/\
_Chapter_SoftwareHardware_Pre_Processing/Real-ESSI_gmESSI.tgz}
##
# make directory, move files, expand archive
mkdir Real-ESSI-gmESSI
mv _all_files_gmESSI_.tgz Real-ESSI-gmESSI
cd Real-ESSI-gmESSI
tar -xvzf _all_files_gmESSI_.tgz

## build the package
make         # builds the application in curr_dir/build

## install the package
# -- by default the pacakage is installed in /usr/local
make install # installs the package in /usr/local
# -- to change the install directory
make install INSTALL_DIR=install_dir_specified_by_user

############## For installation of gmESSI plugin in sublime ↩
   ##############
# open sublime-text
# make sure you have installed package control
# if not then install it first from
# {https://packagecontrol.io/installation}
```

```
# go to Preferences->PackageControl->InstallPackage
# search for [gmESSI-Tools] and install it
# also install [gmsh-Tools] following the same steps
# restart sublime
##########################################################################
```

Running gmESSI

gmESSI can be invoked from the bash terminal by typing *gmessy*. It can take one or multiple *xyz.gmessi* files as an argument and convert them to Real-ESSI files in their respective simulation directory defined by the user. By default, the 'gmessi' python module is automatically imported and available as 'gmESSI' in '.gmeesi' input file.

gmessy is a top level python script that parses the .gmessi file and categories commands in the following order as

- **gmESSI Command:** gmESSI Commands are one line commands. They start and end with '[' and ']' respectively. Section 1.2.3 describes the syntax.
- **gmESSI Comments:** The lines that start with '//' are considered as gmESSI comments. It gets translated and copied to the main file (See Section 1.2.4).
- **Singular Commands:** The lines that start with '!' are directly copied to the main file (See Section 1.2.4 and Section 1.2.5). Real-ESSI domain specific language (DSL) are written following the exclamation mark '!' sign.
- **Python Comments:** The lines that start with '#' are considered as python comments.
- **Python Commands:** Whatever lines left are considered as python commands. This option is only for the advanced user and is not documented to make the manual simple. Only some useful commands required are explained in the manual.

The categorized commands then generates an equivalent python (.py) script, which gets finally run in python interpretor. The generated equivalent python script can be seen by adding '-l' or '–logfile= LOG_FILE' option during execution. It is important to note that nodes, coordinates, element no etc generated from the translator have a precision associated with them. By default the precision is up-to '6' significant digits. The user can change the precision anywhere in the .gmessi file as

```
gmESSI.setPrecision(10);
```

This will set the precision to '10' significant digits. Lowering precision can be helpful in generating same coordinates for contact/interface node pairs. See [Example_4.gmessi] for its usage.

The full description of *gmessy* can be invoked from the terminal as

```
$gmessy --help

usage: gmessy [-h] [-l] [-nm] [-em] [-ne] [--logfile= LOG_FILE]
[--nodemap= NM_FILE] [--elemap= ELM_FILE]
gmessi_filename

positional arguments:
gmessi_filename      filename containing semantics of conversion

optional arguments:
-h, --help           show this help message and exit
-l                   generate the log file at the current location
-nm                  generate the node map file at the current location
-em                  generate the element map file at the current location
-ne                  don't carry out the conversion
--logfile= LOG_FILE  generate the log file at specified location
--nodemap= NM_FILE   generate the node-map (gmsh-to-Real_ESSI) file at specified location
--elemap= ELM_FILE   generate the element-map (gmsh-to-Real_ESSI) file at specified location
```

Since gmESSI optimizes the 'node' and 'element' tag for Real-ESSI, it provides an interface to retrieve the node map and element map containing mapping from gmsh_tag to Real_ESSI_Tag.

Running gmESSI requires, the *.gmessi* input file and the gmsh mesh (.msh) mesh file containing physical groups. Let's go and run an example to see how gmESSI works. [Example_1] can be obtained here.

Alternatively in the gmESSI directory, navigate to the Examples directory and then to Example_1 directory.

```
$cd ./Examples/Example_1
$ls
Example_1.geo    # geometry file [gmsh]
Example_1.msh    # mesh file [gmsh]
Example_1.gmessi # gmessi input file [gmESSI]
```

Contents of Example_1.gmessi input file: As described above, the .gmessi input file contains gmESSI commands, singular commands and python commands. Also, it can contain comments followed by // or #. At the beginning of the input file, the simulation directory, main, node, element, load filenames must be specified. Also, before adding any gmESSI command, mesh must be loaded using 'gmESSI.loadGmshFile' command.

```
$ cat Example_1.gmessi

### loading the msh file
gmESSI.loadGmshFile("Example_1.msh")

```

```
### Physical Groups defined in the msh file.
#2 2 "Base_Surface"
#2 3 "Top_Surface"
#3 1 "Soil"

### Defining the Simulation Directory
gmESSI.setSimulationDir("./Example_1_ESSI_Simulation")
gmESSI.setMainFile(gmESSI.SimulationDir+ "main.fei")
gmESSI.setNodeFile(gmESSI.SimulationDir+ "node.fei")
gmESSI.setElementFile(gmESSI.SimulationDir+ "element.fei")
gmESSI.setLoadFile(gmESSI.SimulationDir+ "load.fei")


// My new model
! model name "Soil_Block";

[Add_All_Node{ unit:= m, nof_dofs:= 3}]

// Adding Material layer wise and also assigning it to elements
[Vary_Linear_Elastic_Isotropic_3D{Physical_Group#Soil, ↩
   ElementCommand:= [Add_8NodeBrick{}], Density:= 1600+10*(10-z)\ 0 ↩
   \kg/m^3, ElasticModulus:= 20e9+10e8*(10-z)\-8\Pa, PoissonRatio:= ↩
   0.3}]

! include "node.fei";
! include "element.fei";
! new loading stage "Stage1_Surface_Loading";

# Applying Fixities
[Fix_Dofs{Physical_Group#Base_Surface, all}]

#### For applying Surface load on the Top Surface of the Soil Block
#[Add_8NodeBrick_SurfaceLoad{Physical_Group#1,Physical_Group#3,10*Pa}]

#### For applying Nodal loads to all the nodes of the top surface
[Add_Node_Load_Linear{Physical_Group#Top_Surface, ForceType:= Fx, ↩
   Mag:= 10*kN}]

#### For applying Self-Weight Load to the soil elements
! add acceleration field # 1 ax =  0*g ay = 0*g  az = -1*g ;
! add load #18 to all elements type self_weight use acceleration ↩
   field # 1;

# Updating the tag inside gmESSI as user entered by himself load tag
gmESSI.setESSITag("load",19)

! include "load.fei";
! NumStep = 10;
!
!  define algorithm With_no_convergence_check;
!  define solver UMFPack;
!  define load factor increment 1/NumStep;
```

```
!  simulate NumStep steps using static algorithm;
! bye;
```

Running Example_1 in Terminal:

```
$ gmessy Example_1.gmessi
Message:: newDirectory created as ./Example_1_ESSI_Simulation


Add_All_Node{ unit:= m, nof_dofs:= 3}
   Found!!
   Successfully Converted

Vary_Linear_Elastic_Isotropic_3D{Physical_Group#Soil, ↩
   ElementCommand:= [Add_8NodeBrick{}], Density:= 1600+10*(10-z)\ 0 ↩
   \kg/m^3, ElasticModulus:= 20e9+10e8*(10-z)\-8\Pa, PoissonRatio:= 0.3}
   Found!!
   Successfully Converted

Fix_Dofs{Physical_Group#Base_Surface, all}
   Found!!
   Successfully Converted

Add_Node_Load_Linear{Physical_Group#Top_Surface, ForceType:= Fx, ↩
   Mag:= 10*kN}
   Found!!
   Successfully Converted

************************ Updated New Tag Numbering ↩
   **********************
damping         = 1
displacement    = 1
element         = 28
field           = 1
load            = 17
material        = 4
motion          = 19
node            = 65
nodes           = 65
Gmsh_Elements   = 46
Gmsh_Nodes      = 65
```

It must be noted that the terminal only displays information about gmESSI commands. The singular commands are directly copied to the main file. The translator creates a user defined directory *Example_1_ESSI_Simulation* and places

1. node.fei
2. element.fei
3. load.fei

4. main.fei

5. Example_1.msh

The terminal displays the WARNING, ERROR messages and log of command conversions as shown above. At the end, it displays the *Available ESSITag's* numbering, which can be refereed and used for further conversion.

ESSITags are explained later in this manual in Section 1.2.4.

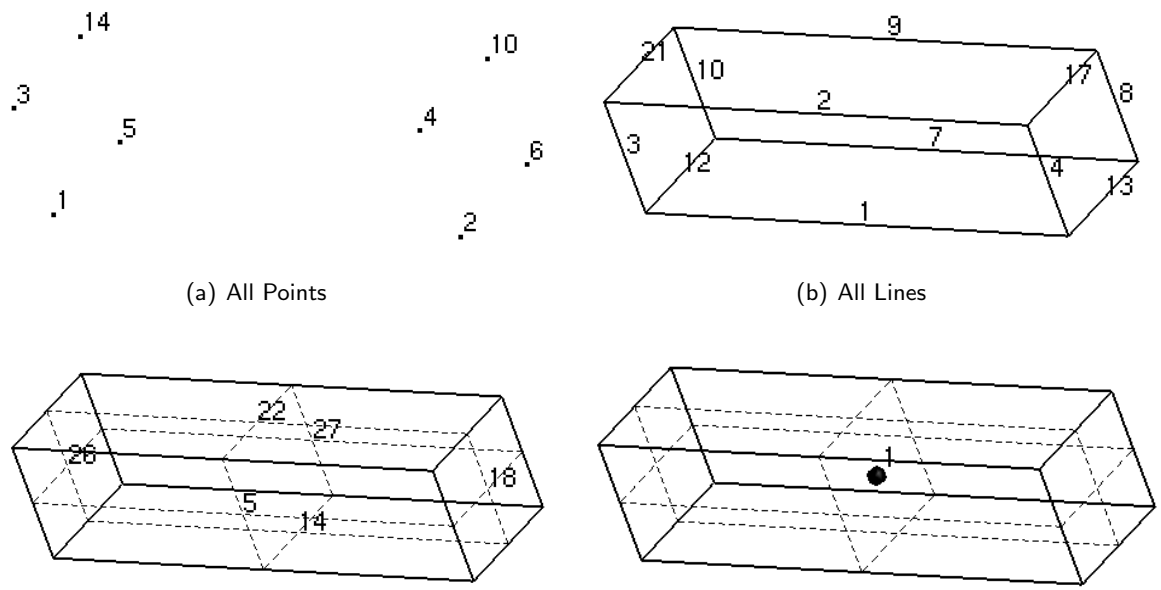
The Real-ESSI input files produced can be tweaked a little if required. Once all is set, the model can be run through Real-ESSI Simulator

```
cd Example_1_ESSI_Simulation

### To run ESSI in sequential
# -- assuming sequential executable name is 'essi'
essi -f main.fei

### To run ESSI in parallel
# --  assuming parallel executable name is 'pessi'
mpirun -np 4 pessi -f main.fei
```

1.2.2 Gmsh Physical Groups and Geometrical Entities

Geometrical Entities are the most elementary group in Gmsh. Each point, line, surface and volume is a geometrical entity and possess a unique identification number. Elementary geometrical entities can then be manipulated in various ways, for example using the *Translate, Rotate, Scale or Symmetry* commands. They can be deleted with the *Delete* command, provided that no higher-dimension entity references them. Example_2.geo shows description of a geometry (.geo) file in gmsh for creating a *cantilever beam*. The files can be downloaded here. Alternatively, it can be located in the gmESSI directory by navigating to the Examples/Example_2 directory.

```
$ cat Example_2.geo
// Creating a point
Point(1) = {0,0,0};

// Dividing the beam length in 5 parts
Extrude (4,0,0) {Point{1}; Layers{5};}

// Dividing the beam width in 2 parts
Extrude (0,1,0) {Line{1}; Layers{2};Recombine;}

// Dividing the beam depth in 2 parts
Extrude (0,0,1) {Surface{5}; Layers{2};Recombine;}
```



(a) All Points (b) All Lines

(c) All Surface (d) All Volume

Figure 1.1: Showing Geometrical Entities. Every point, line, surface and volume has an unique identification number assigned to it.

Figure 1.1 shows, the different unique identification number attached to each of the nodes, lines, surface and volume of the geometry of *cantilever beam*. Physical groups can now be created of type {nodes, lines, surface or volume} containing one or more geometrical entities of their respective type.

Physical Groups are groups of same type {nodes, lines, surface, volume} of elementary geometrical entities. These Physical Groups cannot be modified by geometry commands. Their only purpose is to assemble elementary entities into larger groups, possibly modifying their orientation, so that they can be referred to by the mesh module as single entities. As is the case with elementary entities, each physical point, physical line, physical surface or physical volume are also assigned a unique identification number.

NOTE:- A geometrical entity has only one elementary entity number but can be a part of many physical groups by sharing their unique identification number.

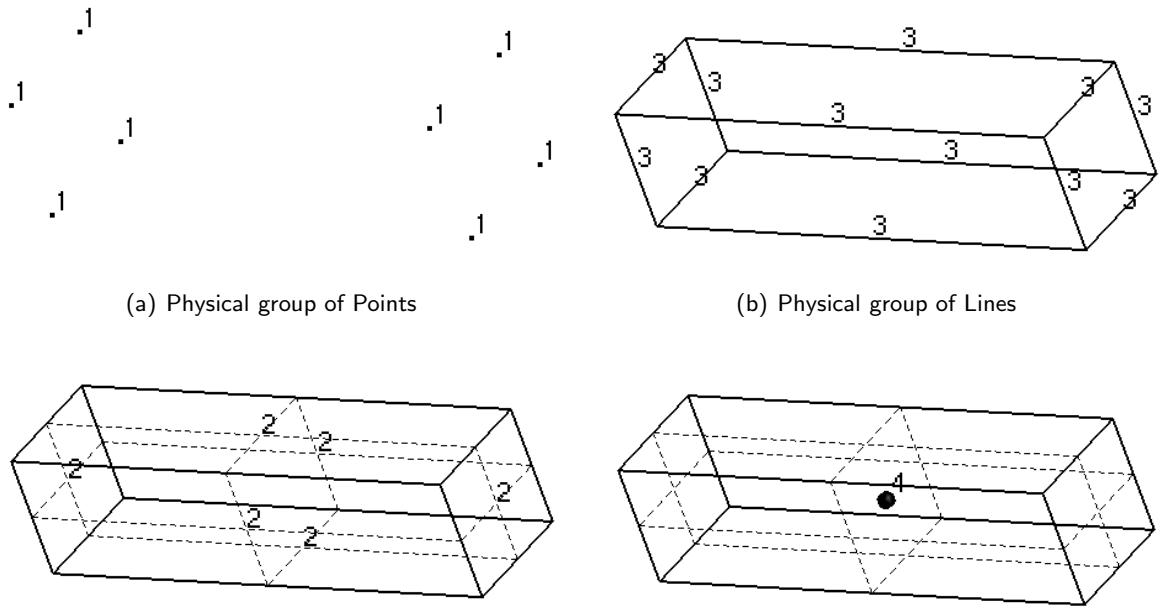
Below is the continuation of Example_2.geo in Gmsh for creating physical Groups of *cantilever beam*. Just for the sake of example, 4 physical groups are created which consist of all points, lines, surface and volume respectively of the *cantilever beam model*. Also physical groups of the surface where fixities and load is applied is created.

```
$ cat Example_2.geo
.....
Physical Point ("All_Points") ={1,2,3,4,5,6,10,14};
Physical Surface("All_Surfaces") = {5,14,22,27,18,26};
Physical Line("All_Lines") ={1,2,3,4,12,13,21,17,7,8,9,10};
Physical Volume("All_Volumes") ={1};
Physical Surface("ApplySurfaceLoad") ={27};
Physical Surface("SurfaceToBeFixed") ={26};
```

In generated mesh (.msh) file, all the geometrical entities have a tag list which contains the ids of the physical groups to which it belongs or is associated. In the above example shown in Figure 1.2, every point, line, surface, volume belongs to only one physical group and thus are showing only one associative number against themselves. Figure 1.3 shows geometrical entities which are part of many physical groups. For example:- the volume shown in Figure 1.3 shows physical group of volumes having id 1 and 7.

The whole idea of creating a Physical Group of points, lines, surfaces and volumes and giving it a unique string name is to allow quick identification and manipulation during gmESSI commands. In Gmsh the name of these Physical Group along with their corresponding elements and nodes gets transferred to the mesh .msh file as shown below. Figure 1.4 shows how Gmsh interprets these Physical groups in .msh file.

```
$cat Example_1.msh
.....
$PhysicalNames
```



(a) Physical group of Points

(b) Physical group of Lines

(c) Physical group of Surface

(d) Physical group of Volume

Figure 1.2: Showing all 4 Physical Groups with entities numbered by their physical group id's.

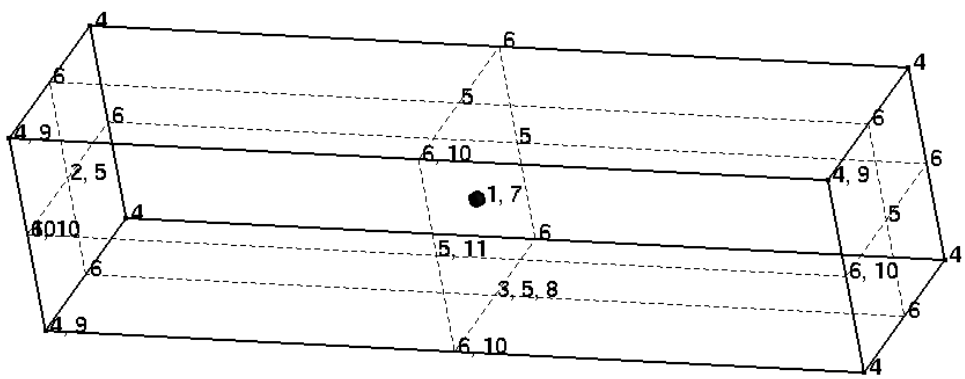


Figure 1.3: Showing geometrical entities associated with more than one physical group.

```
6
0 1 "All_Points"
1 3 "All_Lines"
2 2 "All_Surfaces"
2 5 "ApplySurfaceLoad"
2 6 "SurfaceToBeFixed"
3 4 "All_Volumes"
$EndPhysicalNames
......
```

NOTE:- While creating a physical group in Gmsh, only the information (nodes and elements) of that physical

group gets written in the .msh file and rest are not written. So one must be careful to create physical groups of all entities which is needed during post-processing or conversion. More information about Gmsh syntax, physical groups, commands, .msh file, save options, is available at the main online documentation web site: http://geuz.org/gmsh/doc/texinfo/gmsh.html

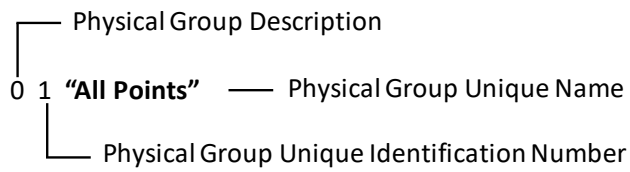


Figure 1.4: Description showing how gmsh interprets the Physical Groups.

- **Physical Group Description ::** Gmsh uses it to identify the type of physical group. $0, 1, 2 \quad and \quad 3$ represents the physical group of geometric points, lines, surface and volume respectively.
- **Physical Group Unique Identification Number ::** It is an unique identification number automatically assigned to each physical group by gmsh.
- **Physical Group Unique Name ::** It is also the same as *Physical Group Unique Identification Number* but the difference is that it is not automatic but defined by the user and that too in the form of string.

The gmESSI Translator utilizes the property of naming the physical group as "string" to get gmESSI commands from the user along with specific physical group on which it is operated. Below in shown [Example_2.gmessi] input file for a *Cantilever analysis*. It shows how to write gmESSI commands with physical group information on which it is operated. gmESSI utilizes the mesh (.msh) file to get the respective physical group and translated it to ESSI input (.fei) files.

```
$ cat Example_2.gmessi

gmESSI.loadGmshFile("Example_2.msh")

####### Physical Groups Available in Example_2.msh file
#0 1 "All_Points"
#1 3 "All_Lines"
#2 2 "All_Surfaces"
#2 5 "ApplySurfaceLoad"
#2 6 "SurfaceToBeFixed"
#3 4 "All_Volumes"

############ Important!! to set the file names ###################
gmESSI.setSimulationDir("./Example_2_ESSI_Simulation")
```

```
gmESSI.setMainFile(gmESSI.SimulationDir+ "main.fei")
gmESSI.setNodeFile(gmESSI.SimulationDir+ "node.fei")
gmESSI.setElementFile(gmESSI.SimulationDir+ "element.fei")
gmESSI.setLoadFile(gmESSI.SimulationDir+ "load.fei")

// My new model
! model name "Cantilever_Analysis";

[Add_All_Node{Unit:= m, NumDofs:= 3}]

// Adding Material
! add material 1 type linear_elastic_isotropic_3d mass_density = 2000*kg/m^3 elastic_modulus = 200*MPa poisson_ratio = 0.2;

[Add_8NodeBrick{Physical_Group#All_Volumes, material_no:= 1}]
[Fix_Dofs{Physical_Group#SurfaceToBeFixed, all}]

! include "node.fei";
! include "element.fei";

! new loading stage "Stage1_Uniform_Surface_Load";

# Adding Surface Load
#[Add_8NodeBrick_SurfaceLoad{Physical_Group#All_Volumes, Physical_Group#ApplySurfaceLoad, -10*Pa}]
[Add_Node_Load_Linear{Physical_Group#ApplySurfaceLoad, ForceType:= Fz, Mag:= -10*kN}]

! include "load.fei";
!  define algorithm With_no_convergence_check;
!  define solver UMFPack;
!  define load factor increment 1;
!  simulate 10 steps using static algorithm;
! bye;
```

NOTE:- The first command in [.gmessi] file should be to load the mesh (.msh) file. The syntax to load the gmsh generated mesh file is

```
gmESSI.LoadGmshFile("meshfile.msh")
```

The gmESSI translator reads the command *[Add_All_Node{ Unit:= m, NumDofs:= 3}]* and adds all the nodes from mesh file to ESSI input files. Similarly it translates all the other commands as well.

1.2.3 gmESSI Command Description

gmESSI Translator as said above utilizes the naming of the physical groups to get commands from the user and then carry out the conversion by acting on the defined physical group.

gmESSI Syntax

gmESSI follows strict syntax. gmESSI parses the physical group name string in mesh (.msh) file. Let us have a quick look at the syntax of physical group name.

Physical Group Names : Physical group names are created inside gmsh geometry file. gmESSI follows special syntax as described below.

1. Physical group names used in gmsh should be unique for gmESSI to identify them during post processing.
2. Physical group names should not contain any space
3. Physical group tags can be any alphanumeric sequence but should not contain any of these **[ ]$** literals in their names. Example "Physical_Group_1"

gmESSI Command Syntax : gmESSI translator commands are always enclosed between opening/closing square brackets [and] respectively. A typical gmESSI command syntax is shown in Figure 1.5

Figure 1.5: gmESSI command description.

- **Command Name :** Just as regular function gmESSI Commands have a name and take arguments. The names are usually self explanatory of its function like *Add_8NodeBrick*{...}, *Free_Dofs*{...} .. *etc*
- **Physical Group Argument :** Usually the gmESSI commands have first argument as physical group . For Example:- *Add_8NodeBrick*{*PhysicalGroup#5,...*}, *Add_8NodeBrick*{*PhysicalGroup#All_Volumes,...*}, *Free_Dofs*{*PhysicalGroup#4,...*},.. *etc*.

  Physical Group Id can be the gmsh unique string or number representing that physical group (as shown in .msh file).
- **Arguments :** Arguments as always are separated by comma **','**.

- **Argument Tag** The arguments of gmESSI commands can also have tags associated with them so that it becomes easy for the user to interpret the argument and make changes in future. The tag and the argument is separated by :=. Tag itself has no meaning but it serves as an important information center for user. An example is shown below to show how tags are applied.

  - gmESSI command having arguments without tags
    1. [Add_Node_Load_Linear{Physical_Group#ApplySurfaceLoad, Fz, -10*kN}]
  - gmESSI command having arguments with tags
    1. [Add_Node_Load_Linear{Physical_Group#ApplySurfaceLoad, ForceType:= Fz, Mag:= -10*kN}]
    2. [Add_Node_Load_Linear{Physical_Group#ApplySurfaceLoad, ForceType:= Fz, -10*kN}]
    3. [Add_Node_Load_Linear{Physical_Group#ApplySurfaceLoad, Force_Direction:= Fz, Strength:= -10*kN}]

It can be seen from above examples that the tags are optional and also the user can put their own tag names. The sublime plugin [*gmESSI-Tools*] comes with elaborative tags for the parameters and a lot more with syntax coloring and text-completion for gmESSI commands. It is encouraged to use the plugin and take its advantage.

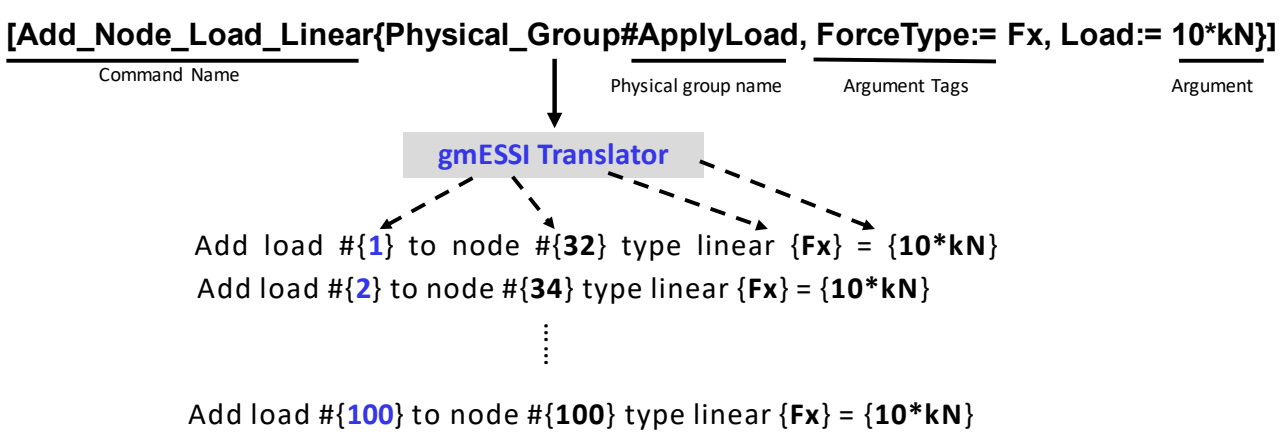


Figure 1.6: gmESSI conversion description.

Figure 1.6 shows the illustration how gmESSI works. Load gets added to all the nodes of the physical group 'ApplyLoad'. gmESSI translator automatically assigns the unique load tag sequentially. It retrieves the node tag from the physical group. Rest of the information like 'ForceType' and 'Magnitude' is obtained from the arguments.

Most of the time these arguments are dummy which means that they just get copied to their equivalent ESSI command at their respective places. These arguments thus have a "string" data-type. For example: the command Add_Node_Load_Linear{Physical_Group#ApplySurfaceLoad, Fz, -10*kN} is equivalent to the Real-ESSI command add load #{} to node #{} type linear {} = {}. Fz and -10*kN goes to their respective position directly through the translator as shown in the Figure 1.6 load number 1 and node number 32 are computed by the translator and then inserted in the ESSI command.

NOTE:- The gmESSI Translator does not provide syntax checking for those dummy arguments. It means that, whatever is written gets copied at the respective position in the equivalent ESSI command, so the one must be careful with what they are writing in these arguments. For Example the command Add_Node_Load_Linear{Physical_Group#Id, ForceDirection, Magnitude} based on the arguments can get converted as

1. [Add_Node_Load_Linear{Physical_Group#ApplySurfaceLoad, ForceType:= Fz, Mag:= -10*kN}]
   −− > add load #1 to node #32 type linear Fz = -10*kN
   −− > add load #2 to node #33 type linear Fz = -10*kN
   .....
   −− > add load #100 to node #100 type linear Fz = -10*kN
2. [Add_Node_Load_Linear{Physical_Group#ApplySurfaceLoad, ForceType:= Fz, Mag:= -10}]
   −− > add load #1 to node #32 type linear Fz = -10
   −− > add load #2 to node #33 type linear Fz = -10
   .....
   −− > add load #100 to node #100 type linear Fz = -10
3. [Add_Node_Load_Linear{Physical_Group#ApplySurfaceLoad, ForceType:= Ft, Mag:= -10*kN}]
   −− > add load #1 to node #32 type linear Ft = -10*kN
   −− > add load #2 to node #33 type linear Ft = -10*kN
   .....
   −− > add load #100 to node #100 type linear Ft = -10*kN

All the above conversions are correct. But only conversion (1.) is correct as an input for Real-ESSI Simulator because force direction is one of *Fx,Fy,Fz* and magnitude *10*kN* has *proper units*. So one must be very careful while writing the arguments.

Note: Some of the arguments are not string but represents numerical quantities, which are manipulated by the translator during conversion. Thus, the one must supply only numbers without any alphabets else it would

lead an unexpected termination of program. These arguments corresponds to *Special Commands* such as *Connect Command* and *Variational Commands*. The manual talks about them later in Section 1.2.5.

gmESSI Command's Physical Group

As iterated earlier, gmESSI commands operates on physical groups. The gmESSI command usually have their first argument as physical on which it operates. The gmESSI syntax allows the users to operates it's command on specific physical groups. The user specifies the group by including an argument Physical_Group#Tag in front of the gmESSI commands describing the command. The *tag* can be either *Physical_Group_Id, Physical_Group_Name*. Let's look at some of them

- [Add_Node_Load_Linear{Physical_Group#5,Fz,-10*kN}] operates on physical group 5
- [Add_8NodeBrick{Physical_Group#All_Volumes, 1}] operates on physical group which has string_tag as All_Volumes

For example in reference to *[Example_2.gmessi]* Physical_Group#All_Volumes or Physical_Group#4 refers the same physical group.

A physical group is a group of point, line, surface or volume defined by the user which contains all the geometrical entities that falls under that domain/group. Figure 1.2 shows physical groups.

1.2.4 gmESSI Output

gmESSI Translator translates the gmESSI commands operated on mesh (.msh) file to different ESSI input (node, element, load and main) (.fei) files and put them in user-defined directory. It also updates the mesh (.msh) file and puts it in the same directory. The *log of translation, errors and warnings* are displayed on the terminal. Below is the demonstration of log messages one by one using *[Example_2.gmessi]* with mesh-file name *Example_2.msh*. The folders and Reall ESSI input (.fei) files that are created by the translator for Example_2.gmessi input file are.

Directory Example_2_ESSI_Simulation

gmESSI Translator creates simulation directory as specified by the user. The user is expected to create the necessary node (Section 1.2.4), element (Section 1.2.4) , load (Section 1.2.4) and main (Section 1.2.4) file to that directory. The user is expected to provide the directory and filenames before executing any gmESSI command. In case the directory already exists a warning messages is shown on the terminal and a new directory following the original name with '_n' (n is number) is created. A new Real-ESSI simulation directory is assigned by the following command

```
gmESSI.setSimulationDir("./Example_2_ESSI_Simulation", overwrite\_mode)
```

where, 'overwrite_mode=0' means that in case of already existing folder, a new directory following the original name with '_n' (n is number) is created. 'overwrite_mode=1' would not check for any conflicts and use the same directory as specified by user. For example:- running [Example_.gmessi] file would produce the following message.

```
$ gmessy Example_2.gmessi
Files converted to Examples/Example_2_ESSI_Simulation
```

Again, running the same example would produce the following message as shown below. In [Example_2.gmessi] overwrite is turned off and thats why it creates new-non conflicting directory by appending $_1$ to end.

```
$ gmessy Example_2.gmessi
Message:: newDirectory created as ./Example_2_ESSI_Simulation_1
```

The execution of *gmessy XYZ.gmessi* produces warnings/errors in the following situations.

- **ERROR:: Please Enter the gmessi File ::** It occurs if the user does not give a filename. The possible situation for getting this error is

```
$ gmessy
```

- **ERROR:: The program failed to open the file XYZ.msh** It occurs if the given file or one of the files in the argument does not exist or fails to open because of some reason.
- **WARNING::Directory Already Present.The contents of the Folder may get changed ::** It occurs when users translates the mesh file file XYZ.msh in overwrite mode and the corresponding folder XYZ_ESSI_Simulation already exists at the execution location.
- **Files converted to Examples/Example_2_ESSI_Simulation ::** The message refers to the location of the folder where the translations have been saved.

Translation Log Terminal

gmESSI Translator displays the log of translation of gmESSI commands to corresponding *Real-ESSI commands* on the terminal. Proper *Errors Messages* and *Warnings* are echoed to the user. The execution of the commands are sequential which means the commands written first are executed first and similarly their success and failure is also echoed first. Let us look at this aspect with Example_2.gmessi.

```
$cat Example_2.gmessi
......
! add material 1 type linear_elastic_isotropic_3d mass_density = ↩
   2000*kg/m^3 elastic_modulus = 200*MPa poisson_ratio = 0.2;

[Add_8NodeBrick{Physical_Group#All_Volumes, material_no:= 1}]

[Fix_Dofs{Physical_Group#SurfaceToBeFixed, all}]

! include "node.fei";
! include "element.fei";
......
```

Here, the sequence of execution of commands is '! add material # 1 type linear_elastic_isotropic_3d mass_density $= 2000 * kg/m^3$ elastic_modulus $= 200 * MPa$ poisson_ratio $= 0.2$; ', [Add_8NodeBrick{ Physical_Group#All_Volumes, material_no:= 1}], [Fix_Dofs{ Physical_Group#SurfaceToBeFixed, all}] and '! include "node.fei";'. Notice that the same order gets reflected in the translation log on the terminal as shown below. Also, it must be noted that the commands followed by '!' or '//' or '#' or python commands do not have any log messages corresponding to them.

It must be noted that the lines following '!' are directly copied to the main (Section 1.2.4). Usually Real-ESSI domain specific language that does not operate/require any physical group should be written following exclamation '!' sign.

```
$ gmessy ./Example_2.gmessi
......

```

```
Add_8NodeBrick{Physical_Group#All_Volumes, material_no:= 1}
Found!!
Successfully Converted

Fix_Dofs{Physical_Group#SurfaceToBeFixed, all}
Found!!
Successfully Converted
```

Apart from displaying the log details on the terminal, similar log is added for each translation of gmESSI commands in their respective files in which they are translated. In these files, each successful translation is enclosed between corresponding *RespectiveGmESSICommand Begins* and *RespectiveGmESSICommand Ends*. The same is shown below through the contents of node.fei. Notice that all the translations are enclosed between Begins and Ends Tag.

```
$ cat Examples/Example_2_ESSI_Simulation/node.fei

//*************************************************************
//              Add_All_Node{Unit:= m, NumDofs:= 3}Starts
//*************************************************************

add node # 1 at (0.000000*m,0.000000*m,0.000000*m) with 3 dofs;
add node # 2 at (4.000000*m,0.000000*m,0.000000*m) with 3 dofs;
add node # 3 at (0.000000*m,1.000000*m,0.000000*m) with 3 dofs;
.............................................................

//*************************************************************
//              Add_All_Node{Unit:= m, NumDofs:= 3}Ends
//*************************************************************
```

```
$ cat Examples/Example_2_ESSI_Simulation/element.fei

//*********************************************************************
//              Add_8NodeBrick{Physical_Group#All_Volumes, ↩
   material_no:= 1}Starts
//*********************************************************************

add element #1 type 8NodeBrick with nodes (51,46,29,37,33,17,1,9)  ↩
   use material #1;
add element #2 type 8NodeBrick with nodes (47,28,5,19,51,46,29,37) ↩
   use material #1;
add element #3 type 8NodeBrick with nodes (42,32,46,51,13,3,17,33) ↩
   use material #1;
.............................................................

//*********************************************************************
//              Add_8NodeBrick{Physical_Group#All_Volumes, ↩
   material_no:= 1}Ends
//*********************************************************************
```

NOTE:- The ordering/sequence of commands in ESSI analysis file is important and so the user must make sure that the translations are made in the same order or if not the user should change it manually by (cut/copy/paste) in *node.fei, load.fei and main.fei* files before execution.

Having given a short description of the other translation log/error messages. Let us look more closely one by one and understand the messages,errors and warnings prompted on the terminal.

- **Found!! :** This message in front of the gmESSI command as shown above on translation log in the terminal means that, the corresponding command was found in the gmESSI Command Library.
- **Successfully Converted :** As the message itself describes, it occurs if the command has been successfully translated.
- **Not Found!! :** It occurs if the gmESSI Translator could not find the arbitrary command XYZ in the gmESSI Command library. Example:- *Loading{Fx,10*kN} NotFound!!*
- **WARNING:: Execution of the command escaped. The Gmessi command XYZ could not be found :** The gmESSI Translator does not terminate the translation if a command is not found, instead gives this warning message following the *Not Found!! Error*.
- **Error:: The command XYZ has a syntax error in Physical_Group# tag :** It occurs if there is a syntax error in Physical_Group# argument. The correct represent ion for Physical group Tags is Physical_Group#n, where n is the group id as 1,2,3.. etc. Examples of improper representation are *Phy#2, Physical#Node, ..*
- **Warning:: The command XYZ failed to convert as there is no such Physical Group ::** It occurs if one of the arguments in the command is Physical_Group# and the specified physical group by the user does not exists in the .msh file.
- **Warning:: The command XYZ could not find any nodes/elements on which it operates :** It occurs if for a specified command, the required element types for translation could not be found in the specified Physical group. For Examples:- [Add_8NodeBrick{Physical_Group#1,1}] would give this warning as the Physical_Group#1 being a Physical line group does not contain any 8-Noded Brick elements on which this command operates.
- **ERROR:: Gmsh File has invalid symbols in Node Section. Unable to convert string to integer in Gmsh File :** It occurs if there is perhaps a string inside the Nodes section of .msh file.
- **ERROR:: The command XYZ has a syntax errors ::** It occurs if the specified command by the user contain any syntax errors caught while parsing the command.

- **ERROR:: Gmsh File has invalid symbols in Element Section. Unable to convert string to integer in Gmsh File :** It occurs if there is perhaps a string inside the Element section of .msh file.

Element File (element.fei)

Element file *element.fei* is one of four parts of Real-ESSI input file that contains the translation of commands related to only initialization of elements of the FEM mesh. Generally, all the conversions from Elemental Command (Section 1.2.5) are written to element file.

A new analysis element file is assigned by the following python command

```
gmESSI.setElementFile(gmESSI.SimulationDir+ "element.fei")
```

where, 'gmESSI.SimulationDir' returns the Real-ESSI Simulation directory specified by the user (see section 1.2.4).

Node File (node.fei)

Node file *node.fei* is one of four parts of Real-ESSI input file that contains the translation of commands related to only initialization of nodes of the FEM mesh. All the conversions from Add Node Command (Section 1.2.5) are written to node file.

A new analysis node file is assigned by the following python command

```
gmESSI.setNodeFile(gmESSI.SimulationDir+ "node.fei")
```

where, 'gmESSI.SimulationDir' returns the Real-ESSI Simulation directory specified by the user (see section 1.2.4).

Load File (load.fei)

Load file *load.fei* contains the translation of commands related to the load and boundary conditions on the structure, like declaration of fixities, boundary conditions, master slave, nodal loads, surface loads etc....

A new load file is assigned by the following python command

```
gmESSI.setLoadFile(gmESSI.SimulationDir+ "load.fei")
```

where, 'gmESSI.SimulationDir' returns the Real-ESSI Simulation directory specified by the user (see section 1.2.4).

Analysis File (main.fei)

Analysis file *main.fei* is the main file which is run on Real-ESSI Simulator. The main file must include load, node and element file through *include 'filename.fei'* command.

A new analysis main file is assigned by the following python command

```
gmESSI.setMainFile(gmESSI.SimulationDir+ "main.fei")
```

where, 'gmESSI.SimulationDir' returns the Real-ESSI Simulation directory specified by the user (see section 1.2.4). A typical analysis file after conversion looks like the following.

```
$ cat Examples/Example_2_ESSI_Simulation/Example_2_analysis.fei

// My new model
 model name "Cantilever_Analysis";

// Adding Material
add material 1 type linear_elastic_isotropic_3d mass_density = 2000*kg/m^3 elastic_modulus = 200*MPa poisson_ratio = 0.2;

include "node.fei";
include "element.fei";

new loading stage "Stage1_Uniform_Surface_Load";

include "load.fei";
 define algorithm With_no_convergence_check;
 define solver UMFPack;
 define load factor increment 1;
 simulate 10 steps using static algorithm;
bye;
```

The user can now add solver, time steps and even rearrange the file structure accordingly to Real-ESSI syntax.

NOTE: Real-ESSI Interpreter is sequential and follows certain ordering in commands like materials should be declared before assigning to elements, master-slave can be assigned only when both nodes are declared .. etc.. One should be careful with the order in which conversions are made and if necessary should change it manually by (cut/copy/paste) later in the files geometry.fei, load.fei and analysis.fei or use the python module discussed later before running in ESSI.
Please refer to the Real-ESSI manual for more details on the ordering of the commands.

Mesh File (XYZ.msh)

Mesh file *XYZ.msh* is the input required by the translator. The translator updates the mesh file with users addition. For example:- if Connect-Command (Section 1.2.5) is used, the file contains additional physical group, nodes and 2-noded elements. The Connect Command is discussed in the more detail later in Section 1.2.5.

Updated ESSI Tags Terminal

Updated ESSI Tags refers to the new tag numbering reference associated with ESSI Tags. ESSI has tag numberings associated for damping, displacement, element, field, load, material, and node/nodes. For example in Real-ESSI Command *add node # 1 at (x,y,z) with 3 dofs*, node is a tag and requires a new number like 1 to be associated with that node. The translator displays the new numberings available for each ESSI Tag so that the user is made aware of new numberings for manually specifying an ESSI command after the translation.

gmESSI also provides a python command to set the ESSI Tag. The command is

```
gmESSI.setESSITag(ESSI_Tag_Name,Tag)
```

where,

- **ESSI_Tag_Name :** It refers to a string representing to the Real-ESSI tag such as 'node', 'element', 'field'...etc
- **Tag :** It refers to an integer representing the next available tag.

NOTE : If user is writing its own Real-ESSI domain specific language (DSL), it is expected that the user will update the corresponding Real-ESSI tag used in that DSL. Otherwise, gmESSI would not be able to know the updated available tags. See Example_1.gmessi for its usage.

```
$ gmessy Example_2.gmessi
.....................
******* Updated New Tag Numbering ********
Damping           = 1
displacement      = 1
element           = 21
field             = 1
load              = 19
material          = 2
motion            = 1
node              = 55
nodes             = 55
Gmsh_Elements    = 127
Gmsh_Nodes       = 55
```

1.2.5 gmESSI Commands

Having the knowledge about the syntax, output files, errors and warnings, its time to move on to different types of commands that gmESSI offers. it provide commands operated on physical group to allow conversion for to equivalent Real-ESSI commands. There are also some special command that gmESSI supports. For simplicity, the commands are categorized on the basis of their operation on nodes/elements. As stated earlier, the commands are translated to one of the four files *node.fei, element.fei, load.fei and main.fei*. Let us look at them closely one by one along with all its supported commands.

Singular Commands

Singular Commands does not require any physical group to operate. All the text following exclamation mark '!' are copied directly to the *main.fei* (Section 1.2.4). For Example:- ' ! include 'load.fei'; ' is translated as 'include "load.fei" ' in main.fei analysis file. See [Example_1.gmessi] for its usage.

Note:- Real-ESSI DSL/commands must be followed by the exclamation mark '!'.

Add Node Commands

Add Node Commands have only two commands. *[Add_All_Node{unit,nof_dofs}]* adds all the nodes generated in mesh (.msh) file to 'node.fei' file. Whereas, *[Add_Node{Unit,NumDofs}]* add all the nodes of only specified physical group by the user. These commands operates on all the nodes of the physical group and generate an equivalent Real-ESSI DSL for each of them.

NOTE:- Every Add Node commands get translated into the *node.fei* (Section 1.2.4).

- **gmESSI** : [Add_Node{PhysicalGroup , Unit , NumDofs}]
  translates to series of
  Real-ESSI DSL : add node $\# < . >$ at $(< L >,< L >,< L >)$ with $< . >$ dofs;
  operated over all the nodes defined in the gmsh '.msh' file.

- **gmESSI** : [Add_All_Node{Unit , NumDofs}]
  translates to series of
  Real-ESSI DSL : add node $\# < . >$ at $(< L >,< L >,< L >)$ with $< . >$ dofs;
  operated over all the nodes of the defined physical group

Nodal Commands : Operates On All Nodes of the defined Physical Group

Nodal commands operates on all the nodes of the physical group defined by the user. For example:- [Fix_Dofs{Physical_Group#Lateral_Surface,ux}] would fix ux degree of freedom of all the nodes of physical group 'Lateral_Surface'. It will generate equivalent Real-ESSI DSL 'fix node # < . > dof < . >' and apply to all the nodes of that physical group. Figure 1.6 shows how gmESSI operated on physical groups.

As earlier stated, that the arguments of gmESSI commands are dummy and gets copied directly to the ESSI equivalent command, so one must be very much aware while writing the arguments to the commands. The arguments should be filled with values of the corresponding ESSI command along with required units if any. For more details about the values to the arguments, please refer to ESSI Manual.

NOTE:- Every Nodal command gets translated to the *load.fei* file (Section 1.2.4).

The different commands under this category and their corresponding Real-ESSI commands are listed below

1. **gmESSI** : [Add_Nodes_To_Physical_Group{PhysicalGroup , Physical_Node_Group_String}]
   translates to series of
   Real-ESSI DSL : add nodes (< . >) to [physical_node_group] "$string$";
   operated over all the nodes of the defined physical group

2. **gmESSI** : [Add_Self_Weight_To_Node{PhysicalGroup , field#1}]
   translates to series of
   Real-ESSI DSL : add load # < . > to node # < . > type [self_weight] use acceleration field # < . >;
   operated over all the nodes of the defined physical group

3. **gmESSI** : [Add_Node_Load_Linear{PhysicalGroup , Force_Type , Magnitude}]
   translates to series of
   Real-ESSI DSL : add load # < . > to node # < . > type [linear] [FORCETYPE] = < $forceormoment$ >;
   //[FORCETYPE] = [Fx] [Fy] [Fz] [Mx] [My] [Mz] [F_fluid_x] [F_fluid_y] [F_fluid_z]
   operated over all the nodes of the defined physical group

4. **gmESSI** : [Add_Node_Load_Path_Time_Series{PhysicalGroup , Force_Type , Magnitude , Series_File}]
   translates to series of
   Real-ESSI DSL : add load # < . > to node # < . > type [path_time_series] [FORCETYPE] = < $forceormoment$ > series_file = "$string$";
   operated over all the nodes of the defined physical group

5. **gmESSI** : [Add_Node_Load_Path_Series{PhysicalGroup , Force_Type , Magnitude , Time_Step , Series_File}]

translates to series of

Real-ESSI DSL : add load # < . > to node # < . > type [path_series] [FORCETYPE] = $< forceormoment >$ time_step = $< T >$ series_file = "$string$";

operated over all the nodes of the defined physical group

6. **gmESSI** : [Add_Node_Load_From_Reaction{PhysicalGroup}]

   translates to series of

   Real-ESSI DSL : add load # < . > to node # < . > type [from_reactions];

   operated over all the nodes of the defined physical group

7. **gmESSI** : [Add_Node_Load_Imposed_Motion_Time_Series{PhysicalGroup , Dof_Type , Time_Step , Disp_Scale , Disp_File , Vel_Scale , Vel_File , Acc_Scale , Acc_File}]

   translates to series of

   Real-ESSI DSL : add imposed motion # < . > to node # < . > dof $< DOFTYPE >$ time_step = $< T >$ displacement_scale_unit = $< L >$ displacement_file = "$string$" velocity_scale_unit = $< L/T >$ velocity_file = "$string$" acceleration_scale_unit = $< L/T^2 >$ acceleration_file = "$string$";

   operated over all the nodes of the defined physical group

8. **gmESSI** : [Add_Node_Load_Imposed_Motion_Time_Series{PhysicalGroup , Dof_Type , Time_Step , Disp_Scale , Disp_File , Vel_Scale , Vel_File , Acc_Scale , Acc_File}]

   translates to series of

   Real-ESSI DSL : add load # < . > type imposed motion to node # < . > dof $< DOFTYPE >$ time_step = $< T >$ displacement_scale_unit = $< L >$ displacement_file = "$string$" velocity_scale_unit = $< L/T >$ velocity_file = "$string$" acceleration_scale_unit = $< L/T^2 >$ acceleration_file = "$string$";

   operated over all the nodes of the defined physical group

9. **gmESSI** : [Add_Node_Load_Imposed_Motion_Series{PhysicalGroup , Dof_Type , Disp_Scale , Disp_File , Vel_Scale , Vel_File , Acc_Scale , Acc_File}]

   translates to series of

   Real-ESSI DSL : add imposed motion # < . > to node # < . > dof $< DOFTYPE >$ displacement_scale_unit = $< L >$ displacement_file = "$string$" velocity_scale_unit = $< L/T >$ velocity_file = "$string$" acceleration_scale_unit = $< L/T^2 >$ acceleration_file = "$string$";

   operated over all the nodes of the defined physical group

10. **gmESSI** : [Add_Node_Load_Imposed_Motion_Time_Series{PhysicalGroup , Dof_Type , Time_Step , Disp_Scale , Disp_File , Vel_Scale , Vel_File , Acc_Scl , Acc_File}]

    translates to series of

    Real-ESSI DSL : add load # < . > type imposed motion to node # < . > dof $< DOFTYPE >$

displacement_scale_unit $= < L >$ displacement_file $= "string"$ velocity_scale_unit $= < L/T >$ velocity_file $= "string"$ acceleration_scale_unit $= < L/T^2 >$ acceleration_file $= "string"$;
operated over all the nodes of the defined physical group

11. **gmESSI** : [Add_Damping_To_Node{PhysicalGroup , damping#1}]
    translates to series of
    Real-ESSI DSL : add damping $\# < . >$ to node $\# < . >$;
    operated over all the nodes of the defined physical group

12. **gmESSI** : [Add_Mass_To_Node{PhysicalGroup , MassX , MassY , MassZ}]
    translates to series of
    Real-ESSI DSL : add mass to node $\# < . >$ mx $= < M >$ my $= < M >$ mz $= < M >$;
    operated over all the nodes of the defined physical group

13. **gmESSI** : [Add_Beam_Mass_To_Node{PhysicalGroup , MassX , MassY , MassZ , ImassX , ImassY , ImassZ}]
    translates to series of
    Real-ESSI DSL : add mass to node $\# < . >$ mx $= < M >$ my $= < M >$ mz $= < M >$ Imx $= < ML^2 >$ Imy $= < ML^2 >$ Imz $= < ML^2 >$;
    operated over all the nodes of the defined physical group

14. **gmESSI** : [Fix_Dofs{PhysicalGroup , Dof_Types}]
    translates to series of
    Real-ESSI DSL : fix node $\# < . >$ dofs $< DofTypes >$;
    operated over all the nodes of the defined physical group

15. **gmESSI** : [Free_Dofs{PhysicalGroup , Dof_Types}]
    translates to series of
    Real-ESSI DSL : free node $\# < . >$ dofs $< . >$;
    operated over all the nodes of the defined physical group

16. **gmESSI** : [Remove_Node{PhysicalGroup}]
    translates to series of
    Real-ESSI DSL : remove node $\# < . >$;
    operated over all the nodes of the defined physical group

17. **gmESSI** : [Remove_Equal_Dof_Constrain{PhysicalGroup}]
    translates to series of

Real-ESSI DSL : remove constraint [equal_dof] node # < . >;
operated over all the nodes of the defined physical group

18. **gmESSI** : [Remove_Displacement_From_Node{PhysicalGroup}]
translates to series of
Real-ESSI DSL : remove displacement from node # < . >;
operated over all the nodes of the defined physical group

General Elemental Commands : Operates On All Elements of the defined Physical Group

General Elemental Commands operates on all the elements of a physical group. The translations are written in *load.fei* file. For example:- [Add_SelfWeight_To_Element{Physical_Group#Soil,Field:= 1}] would add self-weight to all the elements of the physical group 'Soil' along the field#1 direction using series of equivalent Real-ESSI DSL 'add load # < . > to element # < . > type [self_weight] use acceleration field # < . >;'

The different commands under this category and their corresponding ESSI commands are listed below

1. **gmESSI** : [Add_Elements_To_Physical_Group{PhysicalGroup , Physical_Element_Group_String}]
   translates to series of
   Real-ESSI DSL : add elements (< . >) to [physical_element_group] "$string$";
   operated over all the nodes of the defined physical group

2. **gmESSI** : [Add_Self_Weight_To_Element{PhysicalGroup , field#1}]
   translates to series of
   Real-ESSI DSL : add load # < . > to element # < . > type [self_weight] use acceleration field # < . >;
   operated over all the nodes of the defined physical group

3. **gmESSI** : [Add_Damping_To_Element{PhysicalGroup , damping#1}]
   translates to series of
   Real-ESSI DSL : add damping # < . > to element # < . >;
   operated over all the nodes of the defined physical group

4. **gmESSI** : [Remove_Element{PhysicalGroup}]
   translates to series of
   Real-ESSI DSL : remove element # < . >;
   operated over all the nodes of the defined physical group

5. **gmESSI** : [Remove_Strain_From_Element{PhysicalGroup}]
   translates to series of
   Real-ESSI DSL : remove strain from element # < . >;
   operated over all the nodes of the defined physical group

Elemental Commands : Operates On All Elements of the defined Physical Group

Elemental Commands operates only to specific elements of a physical group. The translations are written in *element.fei* file. For example:- [Add_8NodeBrick{Physical_Group#Soil,1}] would initialize all the hexahedron elements of physical group 'Soil' to equivalent Real-ESSI commands for defining 8-noded bricks elements 'add element # < . > type [8NodeBrick] with nodes (< . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >) use material # < . >;'. Figure 1.6 shows how gmESSI operated on physical groups. The different commands under this category and their corresponding ESSI commands are listed below

1. **gmESSI** : [Add_20NodeBrick{PhysicalGroup , Num_Gauss_Points , material#1}]
   translates to series of
   Real-ESSI DSL : add element # < . > type [20NodeBrick] with nodes (< . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >) use material # < . >;
   operated over all the elements of the defined physical group

2. **gmESSI** : [Add_20NodeBrick_Variable_GaussPoints{PhysicalGroup , Num_Gauss_Points , material#1}]
   translates to series of
   Real-ESSI DSL : add element # < . > type [20NodeBrick] using < . > Gauss points each direction with nodes (< . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >) use material # < . >;
   operated over all the elements of the defined physical group

3. **gmESSI** : [Add_20NodeBrick_upU{PhysicalGroup , material#1 , Porosity , Alpha , Solid_Density , Fluid_Density , Perm_X , Perm_Y , Perm_Z , Solid_Bulk_Modulus , Fluid_Bulk_Modulus}]
   translates to series of
   Real-ESSI DSL : add element # < . > type [20NodeBrick_upU] with nodes (< . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >) use material # < . > and porosity = < . > alpha = < . > rho_s = $< M/L^3 >$ rho_f = $< M/L^3 >$ k_x = $< L^3T/M >$ k_y = $< L^3T/M >$ k_z = $< L^3T/M >$ K_s = $< stress >$ K_f = $< stress >$;
   operated over all the elements of the defined physical group

4. **gmESSI** : [Add_20NodeBrick_upU_Variable_GaussPoints{PhysicalGroup , Num_Gauss_Points , material#1 , Porosity , Alpha , Solid_Density , Fluid_Density , Perm_X , Perm_Y , Perm_Z , Solid_Bulk_Modulus , Fluid_Bulk_Modulus}]
   translates to series of

Real-ESSI DSL : add element # < . > type [20NodeBrick_upU] using < . > Gauss points each direction with nodes (< . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >) use material # < . > and porosity = < . > alpha = < . > rho_s = $< M/L^3 >$ rho_f = $< M/L^3 >$ k_x = $< L^3T/M >$ k_y = $< L^3T/M >$ k_z = $< L^3T/M >$ K_s = $< stress >$ K_f = $< stress >$;
operated over all the elements of the defined physical group

5. **gmESSI** : [Add_20NodeBrick_up{PhysicalGroup , material#1 , Porosity , Alpha , Solid_Density , Fluid_Density , Perm_X , Perm_Y , Perm_Z , Solid_Bulk_Modulus , Fluid_Bulk_Modulus}]
translates to series of
Real-ESSI DSL : add element # < . > type [20NodeBrick_up] with nodes (< . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >) use material # < . > and porosity = < . > alpha = < . > rho_s = $< M/L^3 >$ rho_f = $< M/L^3 >$ k_x = $< L^3T/M >$ k_y = $< L^3T/M >$ k_z = $< L^3T/M >$ K_s = $< stress >$ K_f = $< stress >$;
operated over all the elements of the defined physical group

6. **gmESSI** : [Add_20NodeBrick_up_Variable_GaussPoints{PhysicalGroup , Num_Gauss_Points , material#1 , Porosity , Alpha , Solid_Density , Fluid_Density , Perm_X , Perm_Y , Perm_Z , Solid_Bulk_Modulus , Fluid_Bulk_Modulus}]
translates to series of
Real-ESSI DSL : add element # < . > type [20NodeBrick_up] using < . > Gauss points each direction with nodes (< . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >) use material # < . > and porosity = < . > alpha = < . > rho_s = $< M/L^3 >$ rho_f = $< M/L^3 >$ k_x = $< L^3T/M >$ k_y = $< L^3T/M >$ k_z = $< L^3T/M >$ K_s = $< stress >$ K_f = $< stress >$;
operated over all the elements of the defined physical group

7. **gmESSI** : [Add_27NodeBrick{PhysicalGroup , material#1}]
translates to series of
Real-ESSI DSL : add element # < . > type [27NodeBrick] with nodes (< . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >) use material # < . >;
operated over all the elements of the defined physical group

8. **gmESSI** : [Add_27NodeBrick_upU{PhysicalGroup , material#1 , Porosity , Alpha , Solid_Density , Fluid_Density , Perm_X , Perm_Y , Perm_Z , Solid_Bulk_Modulus , Fluid_Bulk_Modulus}]

translates to series of

Real-ESSI DSL : add element # < . > type [27NodeBrick_upU] with nodes (< . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >) use material # < . > and porosity = < . > alpha = < . > rho_s = $< M/L^3 >$ rho_f = $< M/L^3 >$ k_x = $< L^3T/M >$ k_y = $< L^3T/M >$ k_z = $< L^3T/M >$ K_s = $< stress >$ K_f = $< stress >$;

operated over all the elements of the defined physical group

9. **gmESSI** : [Add_27NodeBrick_upU_Variable_GaussPoints{PhysicalGroup , NumGaussPoints , material#1 , Porosity , Alpha , Solid_Density , Fluid_Density , Perm_X , Perm_Y , Perm_Z , Solid_Bulk_Modulus , Fluid_Bulk_Modulus}]

   translates to series of

   Real-ESSI DSL : add element # < . > type [27NodeBrick_upU] using < . > Gauss points each direction with nodes (< . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >) use material # < . > and porosity = < . > alpha = < . > rho_s = $< M/L^3 >$ rho_f = $< M/L^3 >$ k_x = $< L^3T/M >$ k_y = $< L^3T/M >$ k_z = $< L^3T/M >$ K_s = $< stress >$ K_f = $< stress >$;

   operated over all the elements of the defined physical group

10. **gmESSI** : [Add_27NodeBrick_up{PhysicalGroup , material#1 , Porosity , Alpha , Solid_Density , Fluid_Density , Perm_X , Perm_Y , Perm_Z , Solid_Bulk_Modulus , Fluid_Bulk_Modulus}]

    translates to series of

    Real-ESSI DSL : add element # < . > type [27NodeBrick_up] with nodes (< . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >) use material # < . > and porosity = < . > alpha = < . > rho_s = $< M/L^3 >$ rho_f = $< M/L^3 >$ k_x = $< L^3T/M >$ k_y = $< L^3T/M >$ k_z = $< L^3T/M >$ K_s = $< stress >$ K_f = $< stress >$;

    operated over all the elements of the defined physical group

11. **gmESSI** : [Add_27NodeBrick_up_Variable_GaussPoints{PhysicalGroup , Num_Gauss_Points , material#1 , Porosity , Alpha , Solid_Density , Fluid_Density , Perm_X , Perm_Y , Perm_Z , Solid_Bulk_Modulus , Fluid_Bulk_Modulus}]

    translates to series of

    Real-ESSI DSL : add element # < . > type [27NodeBrick_up] using < . > Gauss points each direction with nodes (< . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >,

$< . >$, $< . >$, $< . >$, $< . >$, $< . >$, $< . >$, $< . >$, $< . >$, $< . >$, $< . >$, $< . >$, $< . >$, $< . >$, $< . >$, $< . >$) use material # $< . >$ and porosity = $< . >$ alpha = $< . >$ rho_s = $< M/L^3 >$ rho_f = $< M/L^3 >$ k_x = $< L^3T/M >$ k_y = $< L^3T/M >$ k_z = $< L^3T/M >$ K_s = $< stress >$ K_f = $< stress >$;

operated over all the elements of the defined physical group

12. **gmESSI** : [Add_Equal_Dof{PhysicalGroup , Dof_Type}]

    translates to series of

    Real-ESSI DSL : add constraint [equal_dof] with master node # $< . >$ and slave node # $< . >$ dof to constrain $< . >$;

    operated over all the elements of the defined physical group

13. **gmESSI** : [Add_Equal_Dof{PhysicalGroup , Master_Dof ,]

    translates to series of

    Real-ESSI DSL : add constraint [equal_dof] with node # $< . >$ dof $< . >$ master and node # $< . >$ dof $< . >$ slave;

    operated over all the elements of the defined physical group

14. **gmESSI** : [Add_ShearBeam{PhysicalGroup , CrossSection , material#1}]

    translates to series of

    Real-ESSI DSL : add element # $< . >$ type [ShearBeam] with nodes ($< . >$, $< . >$) cross_section = $< l^2 >$ use material # $< . >$;

    operated over all the elements of the defined physical group

15. **gmESSI** : [Add_DispBeamColumn3D{PhysicalGroup , Num_Integr_Points , Section_Number , Density , XZ_Plane_Vect_x , XZ_Plane_Vect_y , XZ_Plane_Vect_z , Joint1_Offset_x , Joint1_Offset_y , J1_z , Joint2_Offset_x , J2_y , J2_Offset_z}]

    translates to series of

    Real-ESSI DSL : add element # $< . >$ type [BeamColumnDispFiber3d] with nodes ($< . >$, $< . >$) number_of_integration_points = $< . >$ section_number = $< . >$ mass_density = $< M/L^3 >$ xz_plane_vector = ($< . >$, $< . >$, $< . >$) joint_1_offset = ($< L >$, $< L >$, $< L >$) joint_2_offset = ($< L >$, $< L >$, $< L >$);

    operated over all the elements of the defined physical group

16. **gmESSI** : [Add_Beam_Elastic{PhysicalGroup , Cross_Section , Elastic_Modulus , Shear_Modulus , Jx , Iy , Iz , Density , XZ_PlaneVect_x , XZ_PlaneVect_y , XZ_Plane_Vect_z , Joint1_Offset_x , Joint1_y , Joint1_Offset_z , Joint2_Offset_x , Joint2_Offset_y , J2_Offset_z}]

    translates to series of

Real-ESSI DSL : add element # < . > type [beam_elastic] with nodes (< . >, < . >) cross_section = < $area$ > elastic_modulus = < F/L^2 > shear_modulus = < F/L^2 > torsion_Jx = < $length^4$ > bending_Iy = < $length^4$ > bending_Iz = < $length^4$ > mass_density = < M/L^3 > xz_plane_vector = (< . >, < . >, < . >) joint_1_offset = (< L >, < L >, < L >) joint_2_offset = (< L >, < L >, < L >);
operated over all the elements of the defined physical group

17. **gmESSI** : [Add_Beam_Elastic_LumpedMass{PhysicalGroup , Cross_Section , Elastic_Modulus , Shear_Modulus , Jx , Iy , Iz , Density , XZ_Plane_Vect_x , XZ_Plane_Vect_y , XZ_Plane_Vect_z , Joint1_Offset_x , Joint1_Offset_y , Joint1_Offset_z , Joint2_Offset_x , Joint2_Offset_y , Joint2_Offset_z}]
translates to series of
Real-ESSI DSL : add element # < . > type [beam_elastic_lumped_mass] with nodes (< . >, < . >) cross_section = < $area$ > elastic_modulus = < F/L^2 > shear_modulus = < F/L^2 > torsion_Jx = < $length^4$ > bending_Iy = < $length^4$ > bending_Iz = < $length^4$ > mass_density = < M/L^3 > xz_plane_vector = (< . >, < . >, < . >) joint_1_offset = (< L >, < L >, < L >) joint_2_offset = (< L >, < L >, < L >);
operated over all the elements of the defined physical group

18. **gmESSI** : [Add_Beam_DisplacementBased{PhysicalGroup , Num_Integration_Points , Section_Number , Density}]
translates to series of
Real-ESSI DSL : add element # < . > type [beam_displacement_based] with nodes (< . >, < . >) with # < . > integration_points use section # < . > mass_density = < M/L^3 > IntegrationRule = "" xz_plane_vector = (< . >, < . >, < . >) joint_1_offset = (< L >, < L >, < L >) joint_2_offset = (< L >, < L >, < L >);
operated over all the elements of the defined physical group

19. **gmESSI** : [Add_HardContact{PhysicalGroup , Normal_Stiffness , Tangential_Stiffness , Normal_Damping , Tangential_Damping , Friction_Ratio , Norm_Vect_x , Norm_Vect_y , Norm_Vect_z}]
translates to series of
Real-ESSI DSL : add element # < . > type [HardContact] with nodes (< . >, < . >) normal_stiffness = < F/L > tangential_stiffness = < F/L > normal_damping = < F/L > tangential_damping = < F/L > friction_ratio = < . > contact_plane_vector = (< . >, < . >, < . >);
operated over all the elements of the defined physical group

20. **gmESSI** : [Add_CoupledHardContact{PhysicalGroup , Normal_Stiffness , Tangential_Stiffness , Normal_Damping , Tangential_Damping , Friction_Ratio , Norm_Vect_x , Norm_Vect_y , Norm_Vect_z}]

translates to series of

Real-ESSI DSL : add element # < . > type [CoupledHardContact] with nodes (< . >, < . >) normal_stiffness = $< F/L >$ normal_penalty_stiffness = $< F/L >$ tangential_stiffness = $< F/L >$ normal_damping = $< F/L >$ tangential_damping = $< F/L >$ friction_ratio = < . > contact_plane_vector = (< . >, < . >, < . >);

operated over all the elements of the defined physical group

21. **gmESSI** : [Add_SoftContact{PhysicalGroup , Initial_Normal_Stiffness , Stiffning_Rate , Maximum_Normal_Stiffness , Tangential_Stiffness , Normal_Damping , Tangential_Damping , Friction_Ratio , Norm_Vect_x , Norm_Vect_y , Norm_Vect_z}]

    translates to series of

    Real-ESSI DSL : add element # < . > type [SoftContact] with nodes (< . >, < . >) initial_normal_stiffness = $< F/L >$ stiffening_rate = $< 1/L >$ max_normal_stiffness = $< F/L >$ tangential_stiffness = $< F/L >$ normal_damping = $< F/L >$ tangential_damping = $< F/L >$ friction_ratio = < . > contact_plane_vector = (< . >, < . >, < . >);

    operated over all the elements of the defined physical group

22. **gmESSI** : [Add_CoupledSoftContact{PhysicalGroup , Initial_Normal_Stiffness , Stiffning_rate , Maximum_Normal_Stiffness , Tangential_Stiffness , Normal_Damping , Tangential_Damping , Friction_Ratio , Norm_Vect_x , Norm_Vect_y , Norm_Vect_z}]

    translates to series of

    Real-ESSI DSL : add element # < . > type [CoupledSoftContact] with nodes (< . >, < . >) initial_normal_stiffness = $< F/L >$ stiffening_rate = $< 1/L >$ max_normal_stiffness = $< F/L >$ tangential_stiffness = $< F/L >$ normal_damping = $< F/L >$ tangential_damping = $< F/L >$ friction_ratio = < . > contact_plane_vector = (< . >, < . >, < . >);

    operated over all the elements of the defined physical group

23. **gmESSI** : [Add_Truss{PhysicalGroup , material#1 , Cross_Sectin , Density}]

    translates to series of

    Real-ESSI DSL : add element # < . > type [truss] with nodes (< . >, < . >) use material # < . > cross_section = $< length^2 >$ mass_density = $< M/L^3 >$;

    operated over all the elements of the defined physical group

24. **gmESSI** : [Add_8NodeBrick{PhysicalGroup , material#1}]

    translates to series of

    Real-ESSI DSL : add element # < . > type [8NodeBrick] with nodes (< . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >) use material # < . >;

operated over all the elements of the defined physical group

25. **gmESSI** : [Add_Cosserat8NodeBrick{PhysicalGroup , material#1}]
    translates to series of
    Real-ESSI DSL : add element # < . > type [Cosserat8NodeBrick] with nodes (< . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >) use material # < . >;
    operated over all the elements of the defined physical group

26. **gmESSI** : [Add_8NodeBrick_Variable_GaussPoints{PhysicalGroup , NumGaussPoints , material#1}]
    translates to series of
    Real-ESSI DSL : add element # < . > type [8NodeBrick] using < . > Gauss points each direction with nodes (< . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >) use material # < . >;
    operated over all the elements of the defined physical group

27. **gmESSI** : [Add_8NodeBrick_upU{PhysicalGroup , material#1 , Porosity , Alpha , Solid_Density , Fluid_Density , Perm_X , Perm_Y , Perm_Z , Solid_Bulk_Modulus , Fluid_Bulk_Modulus}]
    translates to series of
    Real-ESSI DSL : add element # < . > type [8NodeBrick_upU] with nodes (< . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >) use material # < . > porosity = < . > alpha = < . > rho_s = $< M/L^3 >$ rho_f = $< M/L^3 >$ k_x = $< L^3T/M >$ k_y = $< L^3T/M >$ k_z = $< L^3T/M >$ K_s = $< stress >$ K_f = $< stress >$;
    operated over all the elements of the defined physical group

28. **gmESSI** : [Add_8NodeBrick_upU_Variable_GaussPoints{PhysicalGroup , Num_Gauss_Points , material#1 , Porosity , Alpha , Solid_Density , Fluid_Density , Perm_X , Perm_Y , Perm_Z , Solid_Bulk_Modulus , Fluid_Bulk_Modulus}]
    translates to series of
    Real-ESSI DSL : add element # < . > type [8NodeBrick_upU] using < . > Gauss points each direction with nodes (< . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >) use material # < . > porosity = < . > alpha = < . > rho_s = $< M/L^3 >$ rho_f = $< M/L^3 >$ k_x = $< L^3T/M >$ k_y = $< L^3T/M >$ k_z = $< L^3T/M >$ K_s = $< stress >$ K_f = $< stress >$;
    operated over all the elements of the defined physical group

29. **gmESSI** : [Add_8NodeBrick_up{PhysicalGroup , material#1 , Porosity , Alpha , Solid_Density , Fluid_Density , Perm_X , Perm_Y , Perm_Z , Solid_Bulk_Modulus , Fluid_Bulk_Modulus}]
    translates to series of
    Real-ESSI DSL : add element # < . > type [8NodeBrick_up] with nodes (< . >, < . >, < . >, < . >, < . >, < . >, < . >, < . >) use material # < . > porosity = < . > alpha = < . > rho_s =

$< M/L^3 >$ rho_f $= < M/L^3 >$ k_x $= < L^3T/M >$ k_y $= < L^3T/M >$ k_z $= < L^3T/M >$ K_s $= < stress >$ K_f $= < stress >$;

operated over all the elements of the defined physical group

30. **gmESSI** : [Add_8NodeBrick_up_Variable_GaussPoints{PhysicalGroup , Num_Gauss_Points , material#1 , Porosity , Alpha , Solid_Density , Fluid_Density , Perm_X , Perm_Y , Perm_Z , Solid_Bulk_Modulus , Fluid_Bulk_Modulus}]

    translates to series of

    Real-ESSI DSL : add element $\# < . >$ type [8NodeBrick_up] using $< . >$ Gauss points each direction with nodes ($< . >$, $< . >$, $< . >$, $< . >$, $< . >$, $< . >$, $< . >$, $< . >$) use material $\# < . >$ porosity $= < . >$ alpha $= < . >$ rho_s $= < M/L^3 >$ rho_f $= < M/L^3 >$ k_x $= < L^3T/M >$ k_y $= < L^3T/M >$ k_z $= < L^3T/M >$ K_s $= < stress >$ K_f $= < stress >$;

    operated over all the elements of the defined physical group

Elemental Compound Commands : Operates On All Surface Elements of the defined Physical Group [Surface Loads]

Elemental Compound Commands operates on two physical groups, one for surface and another for the element on which surface is present. It is used mainly for adding surface loads, which require surface number as well as element no in Real-ESSI DSL. For example:- [Add_8NodeBrick_SurfaceLoad{Physical_Group#Volume, Physical_Group#Surface, 10*Pa}] would initialize surface load of $10Pa$ on surfaces defined by physical_group 'Surface' on elements defined by physical_group 'Volume'.

The different commands under this category and their corresponding ESSI commands are listed below

NOTE:- Every Elemental commands get translated into the *load.fei* (Section 1.2.4).

1. **gmESSI** : [Add_20NodeBrick_SurfaceLoad{PhysicalGroup#Volume , PhysicalGroup#Surface , Pressure}]
   translates to series of
   Real-ESSI DSL : add load # < . > to element # < . > type [surface] at nodes (< . > , < . > , < . > , < . >, < . >, < . >, < . >, < . >) with magnitude $< Pa >$;
   operated over all the elements of the defined physical group

2. **gmESSI** : [Add_20NodeBrick_SurfaceLoad{PhysicalGroup#Volume , PhysicalGroup#Surface , Press1 , Press2 , Press3 , Press4 , Press5 , Press6 , Press7 , Press8}]
   translates to series of
   Real-ESSI DSL : add load # < . > to element # < . > type [surface] at nodes (< . > , < . > , < . > , < . >, < . >, < . >, < . >, < . >) with magnitudes ($< Pa >$, $< Pa >$, $< Pa >$, $< Pa >$, $< Pa >$, $< Pa >$, $< Pa >$, $< Pa >$);
   operated over all the elements of the defined physical group

3. **gmESSI** : [Add_27NodeBrick_SurfaceLoad{PhysicalGroup#Volume , PhysicalGroup#Surface , Pressure}]
   translates to series of
   Real-ESSI DSL : add load # < . > to element # < . > type [surface] at nodes (< . > , < . > , < . > , < . >, < . >, < . >, < . >, < . >, < . >) with magnitude $< Pa >$;
   operated over all the elements of the defined physical group

4. **gmESSI** : [Add_27NodeBrick_SurfaceLoad{PhysicalGroup#Volume , PhysicalGroup#Surface , Press1 , Press2 , Press3 , Press4 , Press5 , Press6 , Press7 , Press8 , Press9}]
   translates to series of

Real-ESSI DSL : add load # < . > to element # < . > type [surface] at nodes (< . > , < . > , < . > , < . >, < . >, < . >, < . >, < . >, < . >) with magnitudes (< Pa > , < Pa > , < Pa > , < Pa >, < Pa >, < Pa >, < Pa >, < Pa >, < Pa >);
operated over all the elements of the defined physical group

5. **gmESSI** : [Add_8NodeBrick_SurfaceLoad{PhysicalGroup#Volume , PhysicalGroup#Surface , Pressure}]
translates to series of
Real-ESSI DSL : add load # < . > to element # < . > type [surface] at nodes (< . > , < . > , < . > , < . >) with magnitude < Pa >;
operated over all the elements of the defined physical group

6. **gmESSI** : [Add_8NodeBrick_SurfaceLoad{PhysicalGroup#Volume , PhysicalGroup#Surface , Press1 , Press2 , Press3 , Press4}]
translates to series of
Real-ESSI DSL : add load # < . > to element # < . > type [surface] at nodes (< . > , < . > , < . > , < . >) with magnitudes (< Pa > , < Pa > , < Pa > , < Pa >);
operated over all the elements of the defined physical group

Special Commands

The translator supports some special commands to perform some special functions that are regularly required in simulations. It supports the Connect Command (Section 1.2.5) allows to join or create nodes between two physical groups.

Connect Command

Connect Commands creates/find layers of 2-noded elements between any two parallel geometrical physical entities like two lines, two surface or two volumes and creates a physical group of those elements and updates this information in the *XYZ.msh* file. Since gmsh does not include the feature of defining or creating 2-noded elements after the mesh creation, this command can be very useful in that case. For example;- defining contacts/interfaces, embedded piles, boundary conditions, connections etc. The command syntax for connect command is
gmESSI :: [Connect{Physical_Group#tag_From , Physical_Group#tag_To, Physical_Group#tag_Between, dir_vect, mag, no_times, $algo\_(find|create)$, tolerance,New_Physical_Group_Name}]

- **Physical_Group#tag_From ::** It defines the starting nodes
- **Physical_Group#tag_To ::** It defines the set of end nodes
- **Physical_Group#tag_Between::** It defines the set of nodes where the intermediary nodes can be found, while searching. While creating nodes, it does not play any role.
- **dir_vect ::** It defines the direction in which the user wants to create or find the nodes. The direction vector argument is given as {x_comp \y_comp \z_comp}. Example:- {0 \0 \-1} , {1 \1 \0} .. etc.
- **mag ::** It defines the length of each 2-noded line elements
- **no_times ::** It defines number of layers of 2-noded elements, the user want to create/find
- **algo_(find/create) ::** It defines the algo which is either 0 or 1 meaning whether to find or create the intermediary node
- **tolerance ::** It defines the tolerance is required to finding the nodes. It should be less than the minimum of the distance of neighboring nodes.
- **New_Physical_Group_Name ::** This argument enables the user to give a name to the 2-noded new-physical group formed

Figure 1.7 graphically describes arguments of connect command.
This command updates and creates additional nodes and 2-noded elements and also assigns a physical group name "$New_Physical_Group_Name$". gmESSI automatically adds the next id available to the new physical group. The user can then manipulate this newly created physical group with any other gmESSI commands.

The working of this command would be more clear through examples. [Example_3] can be downloaded here. [Example_4] can be downloaded here. These examples describes two situation one where new nodes are to be created and the other where already present nodes needs to be found respectively. In both the cases

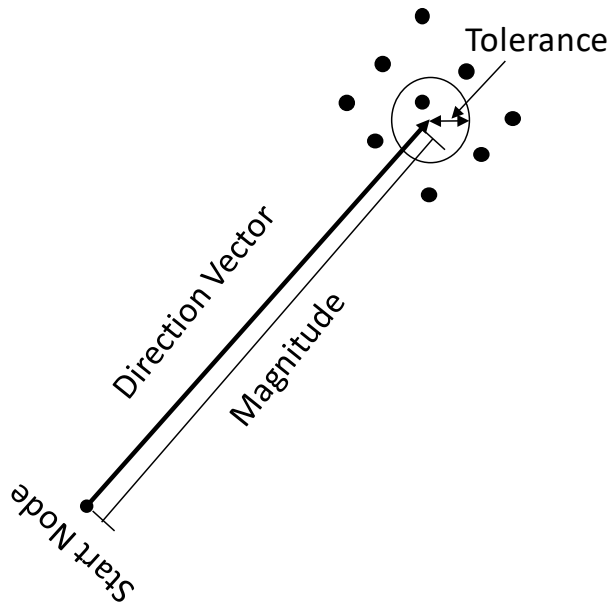


Figure 1.7: Pictorial representation of working of connect command.

2-noded line elements are always created. The examples can also be alternatively located in Examples folder of gmESSI directory

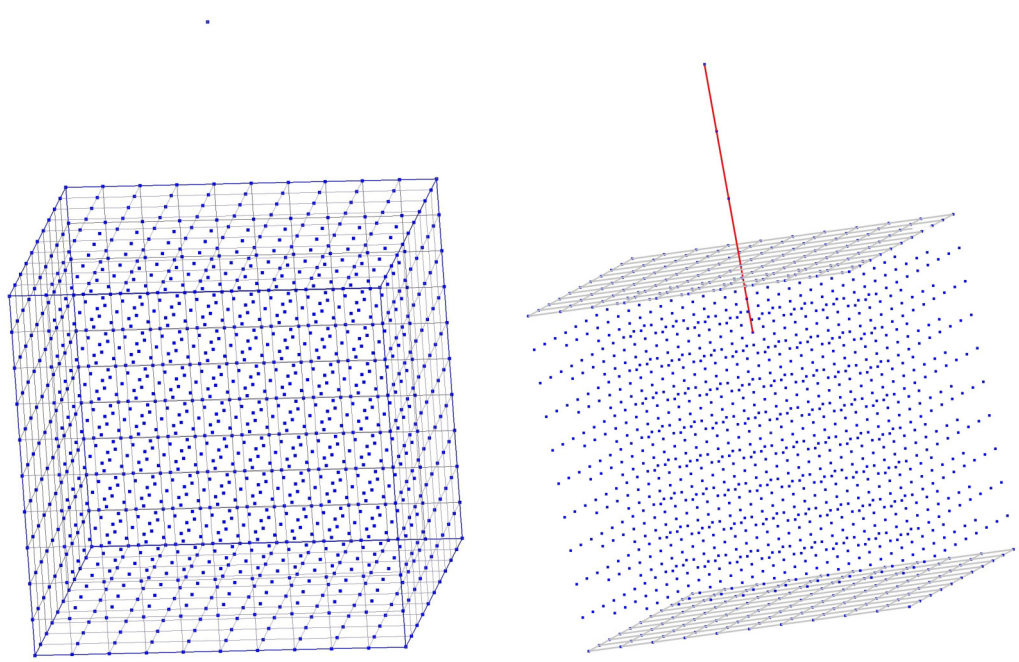
[Example_3] is a simple example where a tower of certain height above ground surface and also its base embedded in soil is modeled. It starts with a mesh file that creates a node for tower at a certain height and then using algorithm -'create' new nodes are created at certain intervals to generate the beam elements. On the other hand, the embedded beam is created by "-find" algorithm. Let us look at the [Example_3.geo] file.

```
$ cat Example_3.geo

// Size of the soil block in meter
Size = 10;

// Height of the Tower in meter
Height = 6;

// Mesh Size of the soil block
Mesh_Size = 1;

// Adding Points and extruding
Point(1)={-Size/2,-Size/2,-Size/2};
Extrude{Size,0,0}{Point{1};Layers{Size/Mesh_Size};Recombine;}
Extrude{0,Size,0}{Line{1};Layers{Size/Mesh_Size};Recombine;}
Extrude{0,0,Size}{Surface{5};Layers{Size/Mesh_Size};Recombine;}

// Make the tower located at height 6 m from the ground surface
Tower = newp;
```

```
Point (Tower) = {0,0,Size/2+Height};

//// Create Physical Groups
Physical Volume ("Soil") = {1};
Physical Surface ("Soil_Base_Surface") ={5};
Physical Surface ("Soil_Top_Surface")  ={27};
Physical Point  ("Tower") = {Tower};


```

Running [Example_3.gmessi] with the .msh output of geometry file would produces additional nodes and elements as shown in Figure 1.8. An excerpt showing use of connect command with create algo in [Example_3.gmessi] is shown below. The effect of the command is shown in Figure 1.8.

```
$ cat Example_3.gmessi
..........
[Connect{Physical_Group#Tower, Physical_Group#Soil_Top_Surface, ↩
  Physical_Group#Soil_Top_Surface, dv1:= 0 \ 0 \ -1, mag:= 2, ↩
  Tolerance:= 0, algo:= create, noT:= 3, PhysicalGroupName:= ↩
  Tower_Beam_Above_Ground}]
..............

```



(a) Initial mesh file Example_3.msh generated by gmsh. The dir vector is in Z axis 0,0,-1

(b) Final mesh after gmESSI

Figure 1.8: Example 3 Contact Problem. (b) shows the nodes and elements generated by gmESSI Translator.

The terminal displays the information about number of elements and nodes created and also displays the information about the new physical group information i.e id and name. The new physical group creation can be seen in the [Example_3.gmsh] in *Example_3_ESSI_Simulation* folder. The terminal message and mesh file is shown below. It also displays error message if more than one node is found in the tolerance provided.

```
$ gmessy Example_3.gmessi
New Physical Group "Tower_Beam_Above_Ground" having id 5 consisting ↩
   of 4 Nodes and 3 2-noded elements created
```

```
$ cat Example_3_ESSI_Simulation\Example_3.msh
............
$PhysicalNames
7
0 4 "Tower"
2 2 "Soil_Base_Surface"
2 3 "Soil_Top_Surface"
3 1 "Soil"
1 5 "Tower_Beam_Above_Ground"
3 6 "TowerBaseNode"
1 7 "Tower_Embedded_Beam"
$EndPhysicalNames
...........
```

[Example_4] describes a foundation on soil problem with contact/interface between them. The contact element is created with the help of comment command using algo "-find". Let us look at the [Example_4.geo] file.

```
 $ cat Example_4.geo

 // Size of the soil block
 Size = 1;

 // Thickness of Foundation
 Thick = 0.1;
 Foundation_Layers = 2;

 //// Mesh Size of the block
 Mesh_Size = 0.2;

 // Adding Points an1 extruding
 Point(1)={-Size/2,-Size/2,-Size/2};
 Extrude{Size,0,0}{Point{1};Layers{Size/Mesh_Size};Recombine;}
 Extrude{0,Size,0}{Line{1};Layers{Size/Mesh_Size};Recombine;}
 Extrude{0,0,Size}{Surface{5};Layers{Size/Mesh_Size};Recombine;}

 // Make sure in Tools -> Geometry -> General
 // Geometry tolerance is set smaller than Epsilon
 // such as Geometry tolerance = 1e-14

 Epsilon = 1e-8;
```

```
Translate {0, 0, Epsilon} {Duplicata{Surface{27};}}
Transfinite Line {29,30,31,32} = Size/Mesh_Size +1;
Transfinite Surface {28};
Recombine Surface {28};

//// Extruding the surface to foundation thickness
Extrude{0,0,Thick}{Surface{28};Layers{Foundation_Layers};Recombine;}

//// Create Physical Groups
Physical Volume ("Soil") = {1};
Physical Surface ("Soil_Base_Surface") ={5};
Physical Surface ("Soil_Top_Surface")  ={27};
Physical Surface ("Foundation_Base_Surface")={28};
Physical Surface ("Foundation_Top_Surface") ={54};
Physical Volume  ("Foundation") = {2};


Physical Surface("Fix_X") = {26, 53, 45, 18};
Physical Surface("Fix_Y") = {22, 49, 14, 41};
Physical Volume("3_Dofs") = {1,2};

```

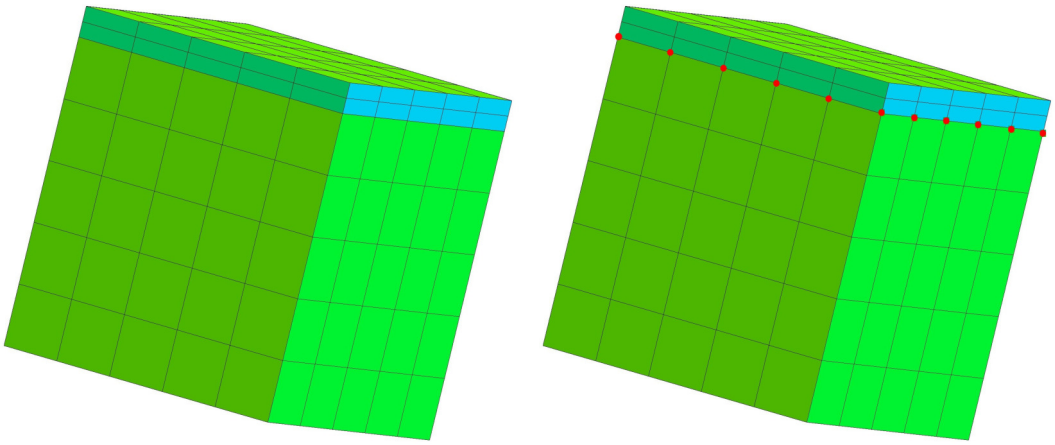
The above geometry file is then meshed with gmsh to get the .msh file. In this file, the connect command is applied between physical group *Foundation_Base_Surface* and *Soil_Top_Surface* to create contact/interface elements. The corresponding connect command would be as

```
$ gmessy Example_4.gmessi
..........
[Connect{Physical_Group#Soil_Top_Surface, ↩
   Physical_Group#Foundation_Base_Surface, ↩
   Physical_Group#Foundation_Base_Surface, dv1:= 0\0\1, mag:= 0, ↩
   Tolerance:= 0.001, algo:= find, noT:= 1, PhysicalGroupName:= ↩
   Contact_Elements}]
..............
```

Similarly the updated [Example_4.msh] contains the new physical group and terminal shows the new physical group of 2-noded elements created. Figure 1.9 shows the new nodes found and creation of 2-noded elements.

```
$ gmessy Example_3.gmessi
New Physical Group "Contact_Elements" having id 10 consisting of 72 ↩
   Nodes and 36 2-noded elements created
```

```
$ cat Example_4_ESSI_Simulation\Example_4.msh
10
2 2 "Soil_Base_Surface"
2 3 "Soil_Top_Surface"
2 4 "Foundation_Base_Surface"
2 5 "Foundation_Top_Surface"
```



(a) Initial mesh file Example4.msh generated by gmsh. The dir vector is in Z axis {0,0,1}

(b) New physical group Contact_Element

Figure 1.9: Example 4 finding nodes problem.(b) shows the nodes and elements generated by gmESSI Translator.

```
2 7 "Fix_X"
2 8 "Fix_Y"
3 1 "Soil"
3 6 "Foundation"
3 9 "3_Dofs"
1 10 "Contact_Elements"
............
```

NOTE : Since the algo is to only find the nodes, so no new nodes are created, but only elements are created. The same message can be seen on the terminal.

Write Command

Write command takes filename as an argument and writes the content of a physical group in two separate files one containing all the nodes info and other containing all the elements info and places in the same XYZ_ESSI_Simulation folder. The command syntax is

gmESSI:: [Write_Data{PhyEntyTag,filename}]

- Creates files *XYZ_filename_Nodes.txt* and *XYZ_filename_Elements.txt*
- **XYZ_filename_Nodes.txt ::** Contains data for all nodes in a physical group. Each node data is represented in one line as

Node_no x_coord y_cord z_cord

with meanings as usual.

- **XYZ_filename_Elements.txt ::** Contains data for all elements in a physical group.Each element data is represented in one line as

  Element_no Element_type node1 node2 node3 ..

  with meanings as usual. Element_type refers to the same as in Gmsh Manual.

[Example_4.gmessi] shows the usage of write command.

Write DRM HDF5 Command

Domain reduction method (DRM) is a very useful method to input 3D seismic excitations into earthquake soil structure interacting system. With a defined physical group as DRM layer, a HDF5 file containing geometric information of the DRM layer, can be generated with the following commands for 1D, 2D and 3D mesh, respectively:

- gmESSI::[Generate_DRM_HDF5_1D{Physical_Group#<PhyEnty Name or Tag>, Surface_Normal:= <X | Y | Z>, Node_Coordinate_Tol:=<tolerance>, FileName:=<HDF5 file name>}]
- gmESSI::[Generate_DRM_HDF5_2D{Physical_Group#<PhyEnty Name or Tag>, Surface_Plane:=<XY|XZ|YZ>, Surface_Normal:= <X | Y | Z>, Node_Coordinate_Tol:=<tolerance>, FileName:=<HDF5 file name>}]
- gmESSI::[Generate_DRM_HDF5_3D{Physical_Group#<PhyEnty Name or Tag>, Surface_Normal:= <X | Y | Z>, Node_Coordinate_Tol:=<tolerance>, FileName:=<HDF5 file name>}]

Where:

- **Physical_Group#** defines the physical group name or tag for the DRM layer.
- **Surface_Normal:=** defines the surface normal direction of the DRM layer. It can be X or Y or Z.
- **Surface_Plane:=** defines the surface plane of the 2D DRM layer. It can be XY or YZ or XZ.
- **Node_Coordinate_Tol:=** defines the tolerance to distinguish two different DRM nodes. The tolerance should be much smaller than the FEM mesh size!
- **FileName:=** defines the file name of the HDF5 file to be generated.

1.2.6 Steps For Using gmESSI tool

Using gmESSI it is very easy to convert a .msh file to ESSI (.fei) file. This section guides the user through a simple [Example_1.geo], to show the steps necessary for generating Real-ESSI files directly from .msh file through gmESSI. Lets define a problem as shown in Figure 1.10. The [Example_1.geo] can be located in the gmESSI ''Examples' directory. Alternatively, it can be downloaded here.

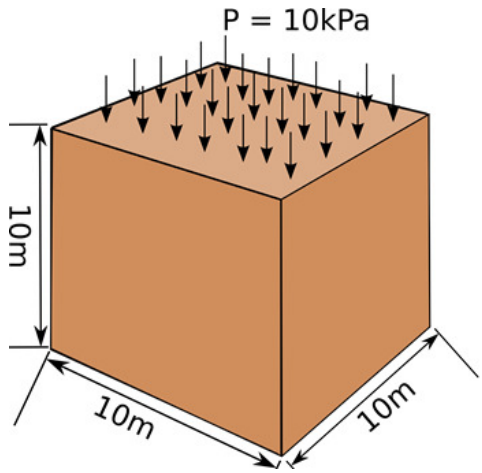


Figure 1.10: Example_1 description of a block of soil with surface load.

It is a block of dimension $10m \times 10m \times 10m$ of soil mass whose all 4 lateral faces are fixed in ux, uy dofs. The bottom face is fixed in ux, uy, uz dofs. A uniform pressure surface load of $10Pa$ is applied. The density and elastic modulus of the soil increases from $2000 * kg/m^3$ and Young's modulus is taken as $200MPa$ as shown in Figure 1.10.

Building geometry (.geo) file in Gmsh

The first step is to make the geometry file in *Gmsh*. While creating the geometry the user should also define all the physical groups on which they intend to either apply boundary condition, define elements, loads etc. In [Example_1.geo], 3 physical groups are needed : one for applying surface load, one for fixities, and one for defining the soil volume and assigning material. The content of [Example_1.geo] file is shown below

```
$cat Example_1.geo

// Size of the block
Size = 10;

//// Mesh Size of the block
Mesh_Size = 2;

// Adding Points and extruding
```

```
Point(1)={-Size/2,-Size/2,-Size/2};
Extrude{Size,0,0}{Point{1};Layers{Size/Mesh_Size};Recombine;}
Extrude{0,Size,0}{Line{1};Layers{Size/Mesh_Size};Recombine;}
Extrude{0,0,Size}{Surface{5};Layers{Size/Mesh_Size};Recombine;}

//// Create Physical Groups
Physical Volume ("Soil") = {1};
Physical Surface ("Base_Surface") = {5};
Physical Surface ("Lateral_Surface") = {18,22,14,26};
Physical Surface ("Top_Surface")  ={27};
```

Generate mesh (.msh) file in Gmsh

Once .geo file is ready with all the physical groups, next step is to mesh the model. The mesh operation will generate the mesh file (.msh) that contains all the mesh information.

The model can be meshed from the terminal directly by running:

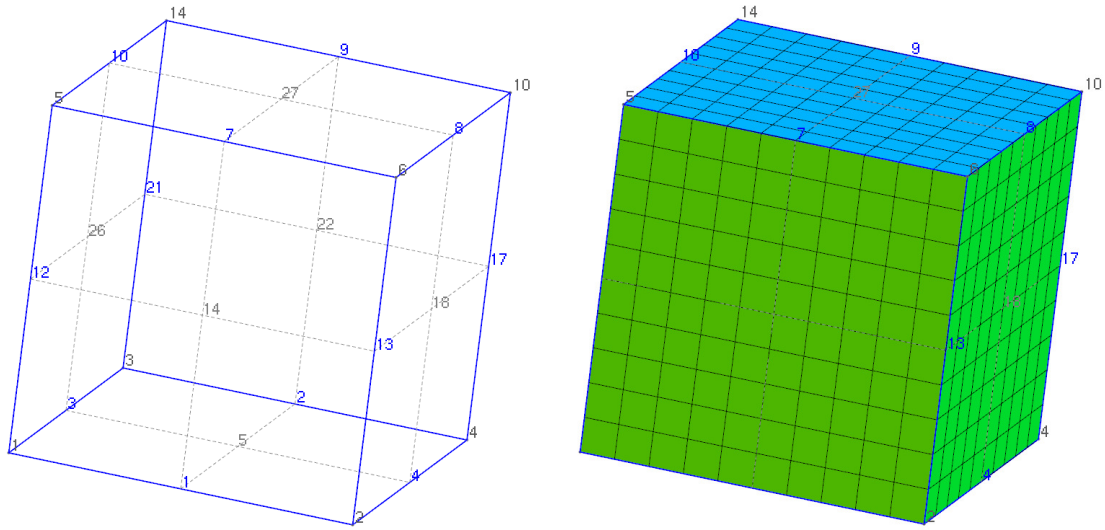
```
gmsh Example_1.geo -3
```

Here **-3** means we are meshing a 3D object, which will automatically mesh all the 3D volumes, 2D surfaces and 1D lines object defined in the geometry model. If there are only 2D surfaces and/or 1D lines object defined in the geometry (.geo) file, use **-2** instead. If there are only 1D lines object defined in the geometry (.geo) file, use **-1** instead.

A quick look at the generated [Example_1.msh] file containing physical groups is shown below:

```
$cat Example_5.geo
..............
$PhysicalNames
4
2 2 "Base_Surface"
2 3 "Lateral_Surface"
2 4 "Top_Surface"
3 1 "Soil"
$EndPhysicalNames
................
```

Figure 1.11 shows the geometry and mesh visualization in Gmsh. It is noted that Gmsh performs meshing for linear interpolation elements by default. In other words, the above cubic block geometry object is meshed into eight-node bricks, that have linear isoparametric interpolation, `8NodeBrick`. For higher order interpolation meshing options, that is for meshing twenty-seven node brick elements mesh, `27NodeBrick` for example, additional **-order int** should be used. **int** here is the integer specifying the order of meshing. For example, the following terminal command with **-order 2**, i.e., 2nd order meshing, generates twenty-seven node brick meshes (`27NodeBrick`):



(a) Geometry File (b) Mesh File

Figure 1.11: Gmsh geometry and mesh file for Example_1

```
gmsh Example_1.geo -3 -order 2
```

Writing all gmESSI Commands for the model

Using gmESSI for mesh conversion is very easy. To achieve this, a [Example_1.gmessi] file is created containing all the required gmESSI commands to be executed sequentially. Let us look at each of them

Since physical group names and ids are required for referring the gmESSI commands, its always best to copy all the physical group data from the .msh file (in this case *[Example_1.msh]* file) in the header of .gmessi file, so that its easier for th user to refer to the physical groups while writing commands in .gmessi file. The contents of the .gmessi file are shown below.

```
$cat Example_1.gmessi

### Physical Groups defined in the msh file.
#2 2 "Base_Surface"
#2 3 "Lateral_Surface"
#2 4 "Top_Surface"
#3 1 "Soil"

### loading the gmsh file
gmESSI.loadGmshFile("Example_1.msh")

### Defining the Simulation Directory and node, element, load and ↩
   main file
```

```
### Its important to define the directory and these files at the beginning of any gmESSI command conversion
### 1 refers as overwrite mode ( will overwrite the directory if present) --- 0 would not overwrite
gmESSI.setSimulationDir("./Example_1_ESSI_Simulation",1)
gmESSI.setMainFile(gmESSI.SimulationDir+ "main.fei")
gmESSI.setNodeFile(gmESSI.SimulationDir+ "node.fei")
gmESSI.setElementFile(gmESSI.SimulationDir+ "element.fei")
gmESSI.setLoadFile(gmESSI.SimulationDir+ "load.fei")

#### // is used to provide commands and gets translated in the main.fei file
#### Also, the commands followed by exclamation '!' get directly copied to the main.fei file
#### Usually, the user would write Real-ESSI DSL against the exclamation mark.

// My new model
! model name "Soil_Block";

[Add_All_Node{ unit:= m, nof_dofs:= 3}]

// Adding Material also assigning it to elements
! add material #1 type linear_elastic_isotropic_3d_LT  mass_density = 2000*kg/m^3  elastic_modulus = 200*MPa  poisson_ratio = 0.3;
[Add_8NodeBrick{Physical_Group#Soil, MaterialNo:= 1}]

! include "node.fei";
! include "element.fei";
! new loading stage "Stage1_Self_Weight";

# Applying Fixities
[Fix_Dofs{Physical_Group#Base_Surface, all}]
[Fix_Dofs{Physical_Group#Lateral_Surface, ux uy}]

#### For applying Self-Weight Load to the soil elements
! add acceleration field # 1 ax =  0*g ay = 0*g  az = -1*g ;
! add load #1 to all elements type self_weight use acceleration field # 1;

 #Updating the tag inside gmESSI as user entered by himself load tag
gmESSI.setESSITag("load",2)


! include "load.fei";
! NumStep = 10;
!
!  define algorithm With_no_convergence_check;
!  define solver UMFPack;
!  define load factor increment 1/NumStep;
!  simulate NumStep steps using static algorithm;

```

```

#### updating the new load file before new loading stage
gmESSI.setLoadFile(gmESSI.SimulationDir+ "Surface_Load.fei")
! new loading stage "Stage2_Surface_Loading";

### For applying Surface load on the Top Surface of the Soil Block
[Add_8NodeBrick_SurfaceLoad{Physical_Group#Soil,Physical_Group#Top_Surface,10*P

##### For applying Nodal loads to all the nodes of the top surface
#[Add_Node_Load_Linear{Physical_Group#Top_Surface, ForceType:= Fx, ↩
   Mag:= 10*kN}]


! include "Surface_Load.fei";
! NumStep = 10;
!
!  define algorithm With_no_convergence_check;
!  define solver UMFPack;
!  define load factor increment 1/NumStep;
!  simulate NumStep steps using static algorithm;

! bye;
```

NOTE: The gmESSI commands are executed and written to the file sequentially, so the user should be careful with the order of translation.

Executing gmESSI on Example_1.gmessi input file

Once .gmessi input file is ready, the next task is to run it using the 'gmessy' command in terminal. Running would carryout the translation to all and produce the log of translation, displayed on the terminal

```
$ gmessy Example_1.gmessi

Message:: newDirectory created as ./Example_1_ESSI_Simulation

Add_All_Node{ unit:= m, nof_dofs:= 3}
   Found!!
   Successfully Converted

Add_8NodeBrick{Physical_Group#Soil, MaterialNo:= 1}
   Found!!
   Successfully Converted

Fix_Dofs{Physical_Group#Base_Surface, all}
   Found!!
   Successfully Converted

Fix_Dofs{Physical_Group#Lateral_Surface, ux uy}
   Found!!
```

```
   Successfully Converted

Add_8NodeBrick_SurfaceLoad{Physical_Group#Soil,Physical_Group#Top_Surface,10*Pa
   Found!!
   Successfully Converted


************************ Updated New Tag Numbering ↩
   **********************
damping          = 1
displacement     = 1
element          = 126
field            = 1
load             = 27
material         = 1
motion           = 126
node             = 217
nodes            = 217
Gmsh_Elements    = 276
Gmsh_Nodes       = 217
```

It would create a folder [Example_1_ESSI_Simulation] and places load.fei, node.fei, element.fei and main.fei files. The user at this point do not need to write anything in the Example_5_analysis.fei file as every command was sequentially written down in .gmessi file and is converted. The content of the main.fei is shown below.

```
$cat Example_5_analysis.fei
// My new model
 model name "Soil_Block";

// Adding Material also assigning it to elements
 add material #1 type linear_elastic_isotropic_3d_LT  mass_density = ↩
   2000*kg/m^3  elastic_modulus = 200*MPa  poisson_ratio = 0.3;

 include "node.fei";
 include "element.fei";
 new loading stage "Stage1_Self_Weight";


 add acceleration field # 1 ax =  0*g ay = 0*g  az = -1*g ;
 add load #1 to all elements type self_weight use acceleration field ↩
   # 1;
 include "load.fei";
 NumStep = 10;

  define algorithm With_no_convergence_check;
  define solver UMFPack;
  define load factor increment 1/NumStep;
  simulate NumStep steps using static algorithm;
 new loading stage "Stage2_Surface_Loading";

 include "Surface_Load.fei";
```

```
NumStep = 10;

 define algorithm With_no_convergence_check;
 define solver UMFPack;
 define load factor increment 1/NumStep;
 simulate NumStep steps using static algorithm;
bye;
```

Running Real-ESSI and visualization in paraview

With all files ready in their place, the next step is to run the main.fei file directly in ESSI.

```
$essi -f main.fei
```

Running ESSI creates .feioutput file which can be visualized in paraview using PVESSIReader plugin. Figure **??** shows the visualization of hdf5 output produced in paraview.

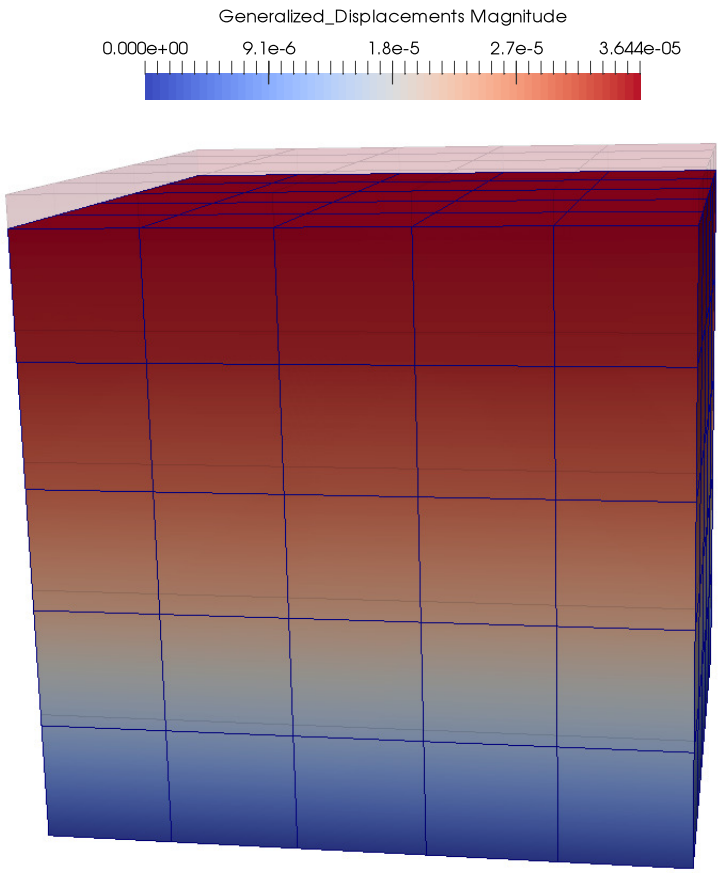


Figure 1.12: Visualizing output in Paraview.

1.2.7 Illustrative Examples

The *Examples* directory of gmESSI folder contains five examples as Example_1,Example_2.... and Example_5. They are summarized as

1. **[Example_1] :** Modeling of Surface load on block of Soil. The geometry (.geo), mesh (.msh) and .gmessi input files can be downloaded HERE.

2. **[Example_2] :** Modeling of Cantilever Beam. The geometry (.geo), mesh (.msh) and .gmessi input files can be downloaded HERE.

3. **[Example_3] :** Modeling of Tower (beam) located above the ground and embedded in soil using contact/interface elements. The geometry (.geo), mesh (.msh) and .gmessi input files can be downloaded HERE.

4. **[Example_4] :** Modeling of a concrete foundation on Soil connected by contact elements. The geometry (.geo), mesh (.msh) and .gmessi input files can be downloaded HERE.

5. **[Example_5] :** Modeling of a embedded shells and beam in Solids. The geometry (.geo), mesh (.msh) and .gmessi input files can be downloaded HERE.

[Example_1] was discussed in the previous section. Examples 1 to 4 are discussed and refereed in the manual at several instances. The user is encouraged to over these examples and learn to create geometry '.geo' and .gmessi input files. Here, two examples Example_2 about cantilever beam analysis and Example_5 about beams and shell is discussed.