Real-ESSI Simulator

Domain Specific Language

José Abell, Yuan Feng, Han Yang, Hexiang Wang

and

Boris Jeremić

University of California, Davis, CA

Version: 9Jul2025, 13:38

http://real-essi.us

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

please use google-chrome to view this PDF so that hyperlinks work

Contents

Chapter 1

Input, Domain Specific Language (DSL)

1991-2005-2010-2011-2012-2015-2016-2017-2020-2021-

1.1 Chapter Summary and Highlights

1.2 Introduction

This chapter presents the domain specific language developed for the Real-ESSI. The language was designed with a primary goal of developing FEA models and interfacing them with various Real-ESSI functionalities. In addition to that, syntax is used to self-document models, provide physical-unit safety, provide common flow control structures, provide modularity to scripting via user functions and "include" files, and provide an interactive environment within which models can be created, validated and verified.

The development of Real-ESSI Domain specific language (DSL) (the Finite Element Interpreter, FEI) is based on LEX (Lesk and Schmidt, 1975) and YACC (Johnson, 1975).

Self-documenting ensures that the resulting model script is readable and understandable with little or no reference to the users manual. This is accomplished by providing a command grammar structure and wording similar to what would be used in a natural language description of the problem.

FEA analysis is unitless, that is, all calculations are carried out without referencing a particular unit system. This leaves the task of unit correctness up to the user of FEA analysis. This represents a recurring source of error in FEA analysis. Physical unit safety is enforced in Real-ESSI by implementing all base variables as physical quantities, that is, all variables have a unit associated with it. The adimensional unit is the base unit for those variables which have no relevant unit (like node numbers). Command calls are sensitive to units. For example, the node creation command call expects the node coordinates to be input with the corresponding units (length in this case). Additionally, the programming/command language naturally supports operation with units like arithmetic operations (quantities with different unit types will not add or subtract but may be multiplied). This approach to FEA with unit awareness provides an additional layer of security to FEA calculations, and forces the user to carefully think about units. This can help catch some common mistakes.

The Real-ESSI language provides modularity through the `include` directive/command, and user functions. This allows complex analysis cases to be parameterized into modules and functions which can be reused in other models.

Finally, an emphasis is placed on model verification and validation. To this end, Real-ESSI provides an interactive programming environment with all the ESSI syntax available. By using this environment. the user can develop tests to detect errors in the model that are not programming errors. For example, the user can query nodes and elements to see if they are set to appropriate states. Also, several standard tools are provided to check element validity (Jacobian, etc.).

The ESSI language provides reduced model development time by providing the aforementioned features along with meaningful error reporting (of syntax and grammatical errors), a help system, command completion and highlighting for several open source and commercial text editors.

Some additional ideas are given by Dmitriev (2004), Stroustrup (2005), Niebler (2005), Mernik et al. (2005), Ward (2003), etc.

1.3 Domain Specific Language (DSL), English Language Binding

Overview of the language syntax.

- Each command line has to end with a semicolon ";"
- Comment on a line begins with either "//" or "!" and last until the end of current line.
- Units are required (see more below) for all quantities and variables.
- Include statements allow splitting source into several files
- All variables are double precision (i.e. floats) with a unit attached.
- All standard arithmetic operations are implemented, and are unit sensitive.
- Internally, all units are represented in the base SI units (m - s - kg).
- The syntax ignores extra white spaces, tabulations and newlines. Wherever they appear, they are there for code readability only. (This is why all commands need to end with a semicolon).
- The user should be familiar with the list of the reserved keywords from Section 1.7 on page 339.

1.3.1 Running Real-ESSI

At the command line type "essi", to get to the ESSI prompt and start Real-ESSI in interactive mode.

Command line output

```
  The Finite Element Interpreter Endeavor

  The Real-ESSI Simulator
  Modeling and Simulation of Earthquakes, and Soils, and Structures ↩
   and their Interaction

  Sequential processing mode.

Version Name       : Real-ESSI Global Release, June2018. Release date: ↩
   Jun 13 2018 at 11:02:19. Tag: adc085ae70
Version Branch     : GLOBAL_RELEASE
Compile Date       : Jun 13 2018 at 14:36:56
Compile User       : jeremic
Compile Sysinfo    : sokocalo 4.13.0-43-generic x86_64 GNU/Linux
```

```
Runtime User      : jeremic
Runtime Sysinfo   : sokocalo 4.13.0-43-generic x86_64 GNU/Linux
Time Now          : Jun 13 2018 at 15:32:52
Days From Release : 0
PostProcessing Compatible Version: ParaView 5.1.2
PostProcessing Compatible Version: ESSI-pvESSI Date: Feb 15 2018 at ↩
   11:00:28. Tag: 58fe430a19

Static startup tips:
 * Remember: Every command ends with a semicolon ';'.
 * Type 'quit;' or 'exit;' to finish.
 * Run 'essi -h' to see available command line options.

ESSI>
```

A number of useful information about Real-ESSI is printed on the screen. From here, commands can be input manually or a file may be included via the `include` command which is as follows.

```
include "foobar.fei";
```

to include the file `foobar.fei`.

A more efficient way to start Real-ESSI and analyze an example is to pass input file name to the command line. Real-ESSI command to execute an input file immediately is done by issuing the following command: `essi -f foobar.fei`. This will execute essi directly on input file `foobar.fei`. After executing the file, the essi interpreter will continue in interactive mode unless the command line flag `-n` or `--no-interactive` is set. A list of command line options is available by calling essi from the command line as `essi -h`.

Command line output

```


  The Finite Element Interpreter Endeavor

  The Real-ESSI Simulator
  Modeling and Simulation of Earthquakes, and Soils, and Structures ↩
   and their Interaction

  Sequential processing mode.

Version Name      : Real-ESSI Global Release, June2018. Release date: ↩
   Jun 13 2018 at 11:02:19. Tag: adc085ae70
Version Branch    : GLOBAL_RELEASE
Compile Date      : Jun 13 2018 at 14:36:56
Compile User      : jeremic
Compile Sysinfo   : sokocalo 4.13.0-43-generic x86_64 GNU/Linux
Runtime User      : jeremic
Runtime Sysinfo   : sokocalo 4.13.0-43-generic x86_64 GNU/Linux
Time Now          : Jun 13 2018 at 16:22:08
Days From Release : 0
PostProcessing Compatible Version: ParaView 5.1.2
```

```
PostProcessing Compatible Version: ESSI-pvESSI Date: Feb 15 2018 at ↩
   11:00:28. Tag: 58fe430a19

Static startup tips:
 * Remember: Every command ends with a semicolon ';'.
 * Type 'quit;' or 'exit;' to finish.
 * Run 'essi -h' to see available command line options.

The Real-ESSI Simulator
Modeling and Simulation of Earthquakes, and Soils, and Structures and ↩
   their Interaction

Usage: essi [-cfhnsmbe FILENAME]
  -c --cpp-output                : Output cpp version of the model.
  -f --filename [FILENAME]       : run ESSI on a FILENAME.
  -h --help                      : Print this message.
  -n --no-interactive            : Disable interactive mode.
  -s --set-variable              : Set a variable from the command line.
  -d --dry-run                   : Do not execute ESSI API calls. ↩
   Just parse.
  -m --model-name [NAME]         : Set the model name from the ↩
   command line.
  -p --profile-report [FILENAME] : Set the filename for the profiler ↩
   report (and activate lightweight profiling)


Example to set a variable name from command line:
    essi -s a=10,b=20,c=30
Runs ESSI with variables a, b, and c set to 10, 20 and 30 respectively.
At this time, only ESSIunits::unitless variables can be set.
```

1.3.2 Finishing Real-ESSI Program Run

To properly finish Real-ESSI program run, and save and close all the output files, user has to use final, closure command:

```
bye;
```

Command `bye;` has to be included at the end of input file script, or at the end of each interactive/interpretative session. Command `bye;` ensures that Real-ESSI program gracefully exits simulation, and that all the output files are properly saved and closed. Proper finishing of simulation using Real-ESSI Simulator is very much necessary, while the choice of command `bye;` is done as an homage to Professor Knuth and his Literate Programming endeavor (Knuth, 1984), that is driving much of the Real-ESSI DSL development.

There are a number of alternative final commands, for example:

```
exit;
quit;
zdravo;
```

```
vozdra;
dvojka;
voljno;
zaijian;
tschuess;
geia-sou;
tchau;
sair;
khoda-hafez;
doei;
nasvidenje;
ajde-bok;
izhod;
konec;
czesc;
ciao;
hoscakal;
```

These additional, alternative final commands can all be written using original scripts:

zdravo ↔ здраво

vozdra ↔ воздра

dvojka ↔ двојка

voljno ↔ вољно

zaijian ↔ 再见

tschuess ↔ tschüss

geia-sou ↔ γεια σου

khoda-hafez ↔ خداحافظ

hoscakal ↔ hoşçakal

1.3.3 Real-ESSI Variables, Basic Units and Flow Control

Variables are defined using the assignment (=) operator. For example,

```
var_x   =  7;       //Results in the variable x be set to 7 (unitless)
var_y = 3.972e+2;   //Scientific notation is available.
```

The language contains a list of reserved keywords. Throughout this documentation, reserved keywords are highlighted in blue or red.

All standard arithmetic operations are available between variables. These operations can be combined arbitrarily and grouped together with parentheses.

```
var_a = var_x + var_y;       // Addition
var_b = var_x - var_y;       // Subtraction
var_c = var_x * var_y;         // Product
var_d = var_x / var_y;         // Quotient
var_e = var_y % var_x;         // Modulus (how many times x fits in y)
```

The 'print' command can be used to display the current value of a variable.

```
print var_x;
print var_y;
print var_a;
print var_b;
print var_c;
print var_d;
print var_e;
```

Command line output

```
var_x = 7 []
var_y = 397.2 []
var_a = 404.2 []
var_b = -390.2 []
var_c = 2780.4 []
var_d = 0.0176234 []
var_e = 5.2 []
```

Here the "unit" (sign) [] means that the quantities are unitless.

The command 'whos' is used to see all the currently defined variables and their values. After a fresh start of essi, needed to clear up all the previously defined variables, command whos;' produces a list of predefined variables:

Command line output

```
ESSI> whos;

Declared variables:
  *       Day =           86400 [s]
  *       GPa =               1 [GPa]
  *      Hour =            3600 [s]
  *        Hz =               1 [Hz]
  *       MPa =               1 [MPa]
  *    Minute =              60 [s]
  *         N =               1 [N]
  *        Pa =               1 [Pa]
  *      Week =          604800 [s]
  *        cm =               1 [cm]
  *      feet =          0.3048 [m]
  *        ft =          0.3048 [m]
```

```
  *         g =              9.81  [m*s^-2]
  *      inch =            0.0254  [m]
  *        kN =                 1  [kN]
  *       kPa =                 1  [kPa]
  *        kg =                 1  [kg]
  *       kip =           4448.22  [N]
  *        km =                 1  [km]
  *       ksi =       6.89476e+06  [Pa]
  *       lbf =           4.44822  [N]
  *       lbm =          0.453592  [kg]
  *         m =                 1  [m]
  *      mile =           1609.35  [m]
  *        mm =                 1  [mm]
  *        pi =           3.14159  []
  *       psi =           6894.76  [Pa]
  *         s =                 1  [s]
  *      yard =            0.9144  [m]

    * = locked variable
ESSI>
```

Predefined variables shown above have a preceding asterisk to show they are locked variables which cannot be modified. The purpose of these locked variables are to provide names for units. Imperial units are also supported as shown above.

The units for variable are shown between the brackets. Note that unit variables have the same name as their unit, which is not the case for user defined variables. Variables preceded by a star (*) are locked variables which can't be modified.

For example, the variable 'm' defines 'meter'. So to define a new variable L1 which has meter units we do:

```
L1 = 1*m;      //  Defines L1 to 1 m.
L2 = 40*mm;    //  Defines L2 to be 40 millimeters.
```

Even though L2 was created with millimeter units, it is stored in base units.

`print L2;` displays

Command line output

```
 L2 = 0.04 [m]
```

As additional examples, let us define few forces:

```
F1 = 10*kN;
F2 = 300*N;
F3 = 4*kg*g;
```

Here g is the predefined acceleration due to gravity.

Arithmetic operations do check (and enforce) for unit consistency. For example, `foo = L1 + F1;` produces an error because units are not compatible. However, `bar = L1 + L2;` is acceptable. On the other hand, multiplication, division and modulus, always work because the result produces a quantity with new units (except when the adimensional quantity is involved).

```
A = L1*L2;
Stress_n = F1 / A;
```

Units for all variables are internally converted to SI units ($kg - m - s$) and stored in that unit system. Variables can be *displayed* using different units by using the [] operator. This does not change the variable, it just displays the value of variable with required unit. For example,

```
print Stress_n;            //Print in base SI units.
print Stress_n in Pa;        //Print in Pascal
print Stress_n in kPa;       //Print in kilo Pascal
```

Command line output

```
Stress_n = 250000 [kg*m^-1*s^-2]

Stress_n = 250000 [Pa]

Stress_n = 250 [kPa]
```

The DSL provides functions to test the physical units of variables. For example,

`print isForce(F1);`

Will print an adimensional, Boolean `1` because `F1` has units of force. While,

`print isPressure(F);`

will print an adimensional, Boolean `0`. The language also provides comparison of quantities with same units (remember all values are compared in SI Units).

`print F1 > F2;`

will print an adimensional, Boolean `1` since `F1` is greater than `F2`.

The program flow can be controlled with `if` and `while` statements, i.e.:

```
if (isForce(F1))
{
 print F1;    // This will be executed
};

if (isForce(L1))
{
 print L1;    // This will not.
};
```

Note the necessary semicolon (`;`) at the closing brace. Unlike C/C++, the braces are always necessary. Closing colon is also always necessary.

The "else" statement is also available:

```
if (isForce(L1))
{
 print L1;    // This will not execute
}
else
{
 print L2;    // This will execute instead
};
```

While loops are also available:

```
i = 0;
while( i < 10)
{
 print i;
 i = i +1;
};
```

1.3.4 Modeling

This section details ESSI modeling commands. Angle brackets <> are used for quantity or variable placeholder, that is, they indicate where user input goes. Within the angle brackets, the expected unit type is given as well, i.e.. <L> means the command expects an input with a value and a length unit. The symbol <.> represents the adimensional quantity.

In addition to that, the vertical bar | ("OR" sign)) is used to separate two or more keyword options, i.e. `[a|b|c]` is used indicate keyword options `a` or `b` or `c`. The symbol |...| is used to denote where several long options exist and are explained elsewhere (an example of this is available below in a material model definitions).

All commands require unit consistency. Base units, SI or other can be used as indicated below:

- length, symbol L, units [m, inch, ft]
- mass, symbol M, units [kg, lbm],
- time, symbol T, units [s]

Derived units can also be used:

- angle, symbol rad (radian), unit [$dimensionless, L/L$]
- force, symbol N (Newton), units [$N, kN, MN, M * L/T^2$],
- stress, symbol Pa (Pascal), units [$Pa, kPa, MPa, N/L^2, M/L/T^2$]

- strain, symbol (no symbol), units $[L/L]$
- mass density, symbol (no symbol), units $[M/L^3]$
- force density, symbol (no symbol), units $[M/L^2/T^2]$

All models have to be named: `model name "model_name_string";` This is important as output files are named based on model name.

Each loading stage has to be named as well. A new loading stage[1] is defined like this:

`new loading stage "loading stage name string";`

In addition to model name, loading stage name is used for output file name for given loading stage.

[1]See more in section 101.4.5 on page 97 in Jeremić et al. (1989-2025).

Modeling, Material Model: Adding a Material Model to the Finite Element Model

Adding constitutive material model to the finite element model/domain is done using command:

```
add material  # <.> type |...|
                 mass_density = <M/L^3>
                 (more model dependent parameters) ;
```

- Material number # (or alternatively No) is a distinct integer number used to uniquely identify this material.
- Mass density should be defined for each material (even if only static analysis is performed, for example if self weight is to be used as a loading stage).
- Depending on material model, there will be additional material parameters that are defined for each material model/type below:

Starting with version 03-NOV-2015 all elastic-plastic material models require explicit specification of the constitutive integration algorithm. More information on this can be found in 1.3.5. Only the material linear_elastic_isotropic_3d_LT ignores this option.

Choices for material_type are listed below.

1.4 List of Available Commands (tentative, not up to date)

```
add acceleration field # <.> ax = <accel> ay = <accel> az = <aaccel> ;
add constraint equal_dof with master node # <.> and slave node # <.> dof to constrain <.>;
add constraint equal_dof with node # <.> dof <.> master and node # <.> dof <.> slave;
add damping # <.> to element # <.>;
add damping # <.> to node # <.>;
add damping # <.> type Caughey3rd with a0 = <1/time> a1 = <time> a2 = <time^3> stiffness_to_use = <Initial_Stiffness|Current_Stiffness|Last_Committed_Stiffness>;
add damping # <.> type Caughey4th with a0 = <1/time> a1 = <time> a2 = <time^3> a3 = <time^5> stiffness_to_use = <Initial_Stiffness|Current_Stiffness|Last_Committed_Stiffness>;
add damping # <.> type Rayleigh with a0 = <1/time> a1 = <time> stiffness_to_use = <Initial_Stiffness|Current_Stiffness|Last_Committed_Stiffness>;
add domain reduction method loading # <.> hdf5_file = <string> scale_factor = <.>;
add domain reduction method loading # <.> hdf5_file = <string>;
add element # <.> type 20NodeBrick using <.> Gauss points each direction with nodes (<.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>) use material # <.>;
add element # <.> type 20NodeBrick with nodes (<.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>) use material # <.>;
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>;
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>;
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>;
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>;
add element # <.> type 27NodeBrick using <.> Gauss points each ↩
   direction with nodes (<.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, ↩
   <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, ↩
   <.>, <.>, <.>, <.>, <.>) use material # <.>;
add element # <.> type 27NodeBrick with nodes (<.>, <.>, <.>, <.>, ↩
   <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, ↩
   <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>) use material # <.>;
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>;
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>;
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>;
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>;
add element # <.> type 3NodeShell_ANDES with nodes (<.>, <.>, <.>) ↩
   use material # <.> thickness = <l> ;
add element # <.> type 4NodeShell_ANDES with nodes (<.>, <.>, <.>, ↩
   <.>) use material # <.> thickness = <l> ;
add element # <.> type 4NodeShell_MITC4 with nodes (<.>, <.>, <.>, ↩
   <.>) use material # <.> thickness = <L>;
add element # <.> type 4NodeShell_NewMITC4 with nodes (<.>, <.>, <.>, ↩
   <.>) use material # <.> thickness = <L>;
add element # <.> type 8_27_NodeBrick using <.> Gauss points each ↩
   direction with nodes (<.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, ↩
```

```
    <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>) use material # <.>;
add element # <.> type 8_27_NodeBrick with nodes (<.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>) use material # <.>;
add element # <.> type 8_27_NodeBrick_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>;
add element # <.> type 8_27_NodeBrick_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>;
add element # <.> type 8_27_NodeBrick_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>;
add element # <.> type 8_27_NodeBrick_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>;
add element # <.> type 8NodeBrick using <.> Gauss points each direction with nodes (<.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>) use material # <.>;
add element # <.> type 8NodeBrick with nodes (<.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>) use material # <.>;
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>;
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>;
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>;
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>;
add element # <.> type beam_9dof_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> );
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> );
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> );
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> );
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> );
add element # <.> type HardContact with nodes (<.>, <.>) ↩
    axial_stiffness = <F/L> shear_stiffness = <F/L> normal_damping = ↩
    <F/L> tangential_damping = <F/L>  friction_ratio = <.>  ↩
    contact_plane_vector = (<.>, <.>, <.> );
add element # <.> type HardWetContact with nodes (<.>, <.>) ↩
    axial_stiffness = <F/L> shear_stiffness = <F/L> normal_damping = ↩
    <F/L> tangential_damping = <F/L>  friction_ratio = <.>  ↩
    contact_plane_vector = (<.>, <.>, <.> );
add element # <.> type ShearBeam with nodes (<.>, <.>) cross_section ↩
    = <l^2> use material # <.>;
add element # <.> type SoftContact with nodes (<.>, <.>) ↩
    initial_axial_stiffness = <F/L> stiffening_rate = <m^-1> ↩
    shear_stiffness = <F/L> normal_damping = <F/L> tangential_damping ↩
    = <F/L>  friction_ratio = <.>  contact_plane_vector = (<.>, <.>, ↩
```

```
<.> );
add element # <.> type SoftWetContact with nodes (<.>, <.>) initial_axial_stiffness = <F/L> stiffening_rate = <m^-1> shear_stiffness = <F/L> normal_damping = <F/L> tangential_damping = <F/L>  friction_ratio = <.>  contact_plane_vector = (<.>, <.>, <.> );
add element # <.> type truss with nodes (<.>, <.>) use material # <.> cross_section = <length^2> mass_density = <M/L^3> ;
add element # <.> type variable_node_brick_8_to_27 using <.> Gauss points each direction with nodes (<.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>, <.>) use material # <.>;
add elements (<.>) to physical_element_group "string";
add fiber # <.> using material # <.> to section # <.>  fiber_cross_section = <area> fiber_location = (<L>,<L>);
add imposed motion # <.> to node # <.> dof DOFTYPE displacement_scale_unit = <displacement> displacement_file = "disp_filename" velocity_scale_unit = <velocity> velocity_file = "vel_filename" acceleration_scale_unit = <acceleration> acceleration_file = "acc_filename";
add imposed motion # <.> to node # <.> dof DOFTYPE time_step = <t> displacement_scale_unit = <length> displacement_file = "disp_filename" velocity_scale_unit = <velocity> velocity_file = "vel_filename" acceleration_scale_unit = <acceleration> acceleration_file = "acc_filename";
add load # <.> to all elements type self_weight use acceleration field # <.>;
add load # <.> to element # <.> type self_weight use acceleration field # <.>;
add load # <.> to element # <.> type surface at nodes (<.> , <.> , <.> , <.>) with magnitude <.>;
add load # <.> to element # <.> type surface at nodes (<.> , <.> , <.> , <.>) with magnitudes (<.> , <.> , <.> , <.>);
add load # <.> to element # <.> type surface at nodes (<.> , <.> , <.> , <.>, <.>, <.>, <.>, <.>) with magnitude <.>;
add load # <.> to element # <.> type surface at nodes (<.> , <.> , <.> , <.>, <.>, <.>, <.>, <.>) with magnitudes (<.> , <.> , <.> , <.>, <.>, <.>, <.>, <.>);
add load # <.> to element # <.> type surface at nodes (<.> , <.> , <.> , <.>, <.>, <.>, <.>, <.>, <.>) with magnitude <.>;
add load # <.> to element # <.> type surface at nodes (<.> , <.> , <.> , <.>, <.>, <.>, <.>, <.>, <.>) with magnitudes (<.> , <.> , <.> , <.>, <.>, <.>, <.>, <.>, <.>);
add load # <.> to node # <.> type from_reactions;
```

```
add load # <.> to node # <.> type linear FORCETYPE = <force or moment>; //FORCETYPE = Fx Fy Fz Mx My Mz F_fluid_x F_fluid_y F_fluid_z
add load # <.> to node # <.> type path_series FORCETYPE = <force or moment> time_step = <time> series_file = "filename";
add load # <.> to node # <.> type path_time_series FORCETYPE = <force or moment> series_file = "filename";
add load # <.> to node # <.> type self_weight use acceleration field # <.>;
add mass to node # <.> mx = <mass> my = <mass> mz = <mass> Imx = <mass*length^2> Imy = <mass*length^2> Imz = <mass*length^2>;
add mass to node # <.> mx = <mass> my = <mass> mz = <mass>;
add material # <.> type CamClay mass_density = <M/L^3> M = <.> lambda = <.> kappa = <.> e0 = <.> p0 = <F/L^2> Poisson_ratio = <.> initial_confining_stress = <F/L^2>
add material # <.> type DruckerPrager mass_density = <M/L^3> elastic_modulus = <F/L^2> poisson_ratio = <.> druckerprager_k = <> kinematic_hardening_rate = <F/L^2> isotropic_hardening_rate = <F/L^2> initial_confining_stress = exp;
add material # <.> type DruckerPragerArmstrongFrederickLE mass_density = <M/L^3> elastic_modulus = <F/L^2> poisson_ratio = <.> druckerprager_k = <> armstrong_frederick_ha = <F/L^2> armstrong_frederick_cr = <F/L^2> isotropic_hardening_rate = <F/L^2> initial_confining_stress = <F/L^2>;
add material # <.> type DruckerPragerArmstrongFrederickNE mass_density = <M/L^3> DuncanChang_K = <.> DuncanChang_pa = <F/L^2> DuncanChang_n = <.> DuncanChang_sigma3_max = <F/L^2> DuncanChang_nu = <.> druckerprager_k = <> armstrong_frederick_ha = <F/L^2> armstrong_frederick_cr = <F/L^2> isotropic_hardening_rate = <F/L^2> initial_confining_stress = <F/L^2>;
add material # <.> type DruckerPragerNonAssociateArmstrongFrederick mass_density = <M/L^3> elastic_modulus = <F/L^2> poisson_ratio = <.> druckerprager_k = <> armstrong_frederick_ha = <F/L^2> armstrong_frederick_cr = <F/L^2> isotropic_hardening_rate = <F/L^2> initial_confining_stress = <F/L^2> plastic_flow_xi = <> plastic_flow_kd = <> ;
add material # <.> type DruckerPragerNonAssociateLinearHardening mass_density = <M/L^3> elastic_modulus = <F/L^2> poisson_ratio = <.> druckerprager_k = <> kinematic_hardening_rate = <F/L^2> isotropic_hardening_rate = <F/L^2> initial_confining_stress = <F/L^2> plastic_flow_xi = <> plastic_flow_kd = <> ;
add material # <.> type DruckerPragervonMises mass_density = <M/L^3> elastic_modulus = <F/L^2> poisson_ratio = <.> druckerprager_k = <> kinematic_hardening_rate = <F/L^2> isotropic_hardening_rate = <F/L^2> initial_confining_stress = exp;
```

```
add material # <.> type linear_elastic_crossanisotropic mass_density = <mass_density> elastic_modulus_horizontal = <F/L^2> elastic_modulus_vertical = <F/L^2> poisson_ratio_h_v = <.> poisson_ratio_h_h = <.> shear_modulus_h_v = <F/L^2>;
add material # <.> type linear_elastic_isotropic_3d mass_density = <M/L^3> elastic_modulus = <F/L^2> poisson_ratio = <.>;
add material # <.> type linear_elastic_isotropic_3d_LT mass_density = <M/L^3> elastic_modulus = <F/L^2> poisson_ratio = <.>;
add material # <.> type roundedMohrCoulomb mass_density = <M/L^3> elastic_modulus = <F/L^2> poisson_ratio = <.> RMC_m = <.> RMC_qa = <F/L^2> RMC_pc = <F/L^2> RMC_e = <.> RMC_eta0 = <.> RMC_Heta = <F/L^2> initial_confining_stress = <F/L^2>
add material # <.> type sanisand2004 mass_density = <M/L^3> e0 = <.> sanisand2004_G0 = <.> poisson_ratio = <.> sanisand2004_Pat = <stress>  sanisand2004_p_cut = <.>  sanisand2004_Mc = <.>  sanisand2004_c = <.> sanisand2004_lambda_c = <.> sanisand2004_xi = <.>  sanisand2004_ec_ref = <.>  sanisand2004_m = <.>  sanisand2004_h0 = <.> sanisand2004_ch = <.>  sanisand2004_nb = <.> sanisand2004_A0 = <.> sanisand2004_nd = <.> sanisand2004_z_max = <.>  sanisand2004_cz = <.> initial_confining_stress = <stress> ;
add material # <.> type sanisand2004_legacy mass_density = <M/L^3> e0 = <.> sanisand2004_G0 = <.> poisson_ratio = <.> sanisand2004_Pat = <stress>  sanisand2004_p_cut = <.>  sanisand2004_Mc = <.>  sanisand2004_c = <.> sanisand2004_lambda_c = <.> sanisand2004_xi = <.>  sanisand2004_ec_ref = <.>  sanisand2004_m = <.>  sanisand2004_h0 = <.> sanisand2004_ch = <.>  sanisand2004_nb = <.> sanisand2004_A0 = <.> sanisand2004_nd = <.> sanisand2004_z_max = <.>  sanisand2004_cz = <.> initial_confining_stress = <stress>  algorithm = <explicit|implicit>  number_of_subincrements = <.>  maximum_number_of_iterations = <.>  tolerance_1 = <.>  tolerance_2 = <.>;
add material # <.> type sanisand2008 mass_density = <M/L^3>  e0 = <.>  sanisand2008_G0 = <.>  sanisand2008_K0 = <.> sanisand2008_Pat = <stress> sanisand2008_k_c = <.>  sanisand2008_alpha_cc = <.> sanisand2008_c = <.>  sanisand2008_xi = <.>  sanisand2008_lambda = <.>  sanisand2008_ec_ref = <.>  sanisand2008_m = <.>  sanisand2008_h0 = <.>  sanisand2008_ch = <.>  sanisand2008_nb = <.>  sanisand2008_A0 = <.> sanisand2008_nd = <.>  sanisand2008_p_r = <.>  sanisand2008_rho_c = <.>  sanisand2008_theta_c = <.>  sanisand2008_X = <.>  sanisand2008_z_max = <.>  sanisand2008_cz = <.>  sanisand2008_p0 = <stress>  sanisand2008_p_in = <.>  algorithm = <explicit|implicit>  number_of_subincrements = <.>  maximum_number_of_iterations = <.>  tolerance_1 = <.>  tolerance_2 = <.>;
```

```
add material # <.> type uniaxial_concrete02 compressive_strength = ↩
   <F/L^2> strain_at_compressive_strength = <.> crushing_strength = ↩
   <F/L^2>  strain_at_crushing_strength = <.> lambda = <.> ↩
   tensile_strength = <F/L^2> tension_softening_stiffness = <F/L^2>;
add material # <.> type uniaxial_elastic elastic_modulus = <F/L^2> ↩
   viscoelastic_modulus = <mass / length / time> ;
add material # <.> type uniaxial_steel01 yield_strength = <F/L^2> ↩
   elastic_modulus = <F/L^2> strain_hardening_ratio = <.>  a1 = <.>  ↩
   a2 = <.>  a3 = <>  a4 = <.> ;
add material # <.> type uniaxial_steel02 yield_strength = <F/L^2> ↩
   elastic_modulus = <F/L^2> strain_hardening_ratio = <.> R0 = <.> ↩
   cR1 = <.> cR2 = <.>  a1 = <.>  a2 = <.>  a3 = <>  a4 = <.> ;
add material # <.> type vonMises mass_density = <M/L^3> ↩
   elastic_modulus = <F/L^2> poisson_ratio = <.> von_mises_radius = ↩
   <F/L^2> kinematic_hardening_rate = <F/L^2> ↩
   isotropic_hardening_rate = <F/L^2> ;
add material # <.> type vonMisesArmstrongFrederick mass_density = ↩
   <M/L^3> elastic_modulus = <F/L^2> poisson_ratio = <.> ↩
   von_mises_radius = <> armstrong_frederick_ha = <F/L^2> ↩
   armstrong_frederick_cr = <F/L^2> isotropic_hardening_rate = ↩
   <F/L^2> ;
add node # <.> at (<length>,<length>,<length>)  with <.> dofs;
add nodes (<.>) to physical_node_group "string";
add section # <.> type elastic3d elastic_modulus = <F/L^2> ↩
   cross_section = <L^2> bending_Iz = <L^4> bending_Iy=<L^4> ↩
   torsion_Jx=<L^4> ;
add section # <.> type Elastic_Membrane_Plate elastic_modulus = ↩
   <F/L^2> poisson_ratio = <.> thickness = <length> mass_density = ↩
   <M/L^3>;
add section # <.> type FiberSection TorsionConstant_GJ = <F*L^2>
add section # <.> type Membrane_Plate_Fiber thickness = <length> use ↩
   material # <.>;
add single point constraint to node # <.> dof to constrain <dof_type> ↩
   constraint value of <corresponding unit>;
add uniform acceleration # <.> to all nodes dof <.> time_step = <T> ↩
   scale_factor = <.> initial_velocity = <L/S> acceleration_file = ↩
   <string>;
check mesh filename;
compute reaction forces;
define algorithm With_no_convergence_check / Newton / Modified_Newton;
define convergence test Norm_Displacement_Increment / ↩
   Energy_Increment / Norm_Unbalance / ↩
   Relative_Norm_Displacement_Increment / Relative_Energy_Increment / ↩
   Relative_Norm_Unbalance tolerance = <.> maximum_iterations = <.> ↩
   verbose_level = <0>|<1>|<2>;
```

```
define dynamic integrator Hilber_Hughes_Taylor with alpha = <.>;
define dynamic integrator Newmark with gamma = <.> beta = <.>;
define load factor increment <.>;
define NDMaterial constitutive integration algorithm Forward_Euler;
define NDMaterial constitutive integration algorithm ↩
    Forward_Euler_Subincrement number_of_subincrements =<.>;
define NDMaterial constitutive integration algorithm ↩
    Forward_Euler|Forward_Euler_Subincrement|Backward_Euler|Backward_Euler_Subin
    yield_function_relative_tolerance  = <.> stress_relative_tolerance ↩
    = <.> maximum_iterations = <.>;
define physical_element_group "string";
define physical_node_group "string";
define solver ProfileSPD / UMFPack;
define static integrator displacement_control using node # <.> dof ↩
    DOFTYPE increment <length>;
disable asynchronous output;
disable element output;
disable output;
enable asynchronous output;
enable element output;
enable output;
fix node # <.> dofs <.>;
fix node # <.> dofs all;
free node # <.> dofs <.>;
help;
if (.) { } else {};
if (.) { };
model name "name_string";
new loading stage "name_string";
output every <.> steps;
output non_converged_iterations;
output support reactions;
print <.>;
print element # <.>;
print node # <.>;
print physical_element_group "string";
print physical_node_group "string";
remove constraint equal_dof node # <.>;
remove displacement from  node # <.>;
remove element # <.>;
remove imposed motion # <.>;
remove load # <.>;
remove node # <.>;
remove physical_node_group "string";
remove strain from element # <.>;
```

```
remove physical_element_group "string";
runTest;
set output compression level to <.>;
simulate <.> steps using static algorithm;
simulate <.> steps using transient algorithm time_step = <time>;
simulate <.> steps using variable transient algorithm time_step = <time> minimum_time_step = <time> maximum_time_step = <time> number_of_iterations = <.>;
simulate constitutive testing BARDETMETHOD use material # <.> scale_factor = <.> series_file = <string>  sigma0 = ( <F/L^2> , <F/L^2> , <F/L^2> , <F/L^2> , <F/L^2> , <F/L^2> )  verbose_output = <.>
simulate constitutive testing constant mean pressure triaxial strain control use material # <.> strain_increment_size = <.> maximum_strain = <.> number_of_times_reaching_maximum_strain = <.>;
simulate constitutive testing DIRECT_STRAIN use material # <.> scale_factor = <.> series_file = <string>  sigma0 = ( <F/L^2> , <F/L^2> , <F/L^2> , <F/L^2> , <F/L^2> , <F/L^2> ) verbose_output = <.>
simulate constitutive testing drained triaxial strain control use material # <.> strain_increment_size = <.> maximum_strain = <.> number_of_times_reaching_maximum_strain = <.>;
simulate constitutive testing undrained simple shear use material # <.> strain_increment_size = <.> maximum_strain = <.> number_of_times_reaching_maximum_strain = <.>;
simulate constitutive testing undrained triaxial stress control use material # <.> strain_increment_size = <.> maximum_strain = <.> number_of_times_reaching_maximum_strain = <.>;
simulate constitutive testing undrained triaxial use material # <.> strain_increment_size = <.> maximum_strain = <.> number_of_times_reaching_maximum_strain = <.>;
simulate using eigen algorithm number_of_modes = <.>;
ux uy uz Ux Uy Uz rx ry rz;
while (.) { };
whos;
```

1.5 List of reserved keywords

The following keywords are reserved and cannot be used as variables in a script or interactive session. Doing so would result in a syntax error.

First Order (commands)

```
a0
a1
a2
a3
a4
acceleration
acceleration_depth
acceleration_file
acceleration_filename
acceleration_scale_unit
add
algorithm
algorithm
all
all
allowed_subincrement_strain
alpha
alpha1
alpha2
and
angle
armstrong_frederick_cr
armstrong_frederick_ha
asynchronous
at
ax
axial_penalty_stiffness
axial_stiffness
axial_viscous_damping
ay
az
bending_Iy
bending_Iz
beta
Beta
beta_min
case
cases
characteristic_strength
check
chi
cohesion
combine
compression
compressive_strength
```

```
compressive_yield_strength
compute
confinement
confinement_strain
constitutive
constrain
constraint
contact_plane_vector
control
convergence
cR1
cR2
cross_section
crushing_strength
Current_Stiffness
cyclic
damage_parameter_An
damage_parameter_Ap
damage_parameter_Bn
damping
define
depth
dilatancy_angle
dilation_angle_eta
dilation_scale
direction
disable
displacement
displacement_file
displacement_scale_unit
dof
dofs
dofs
domain
druckerprager_k
DuncanChang_K
DuncanChang_n
DuncanChang_nu
DuncanChang_pa
DuncanChang_sigma3_max
DYNAMIC_DOMAIN_PARTITION
e0
each
elastic
elastic_modulus
elastic_modulus_horizontal
elastic_modulus_vertical
element
elements
else
enable
every
```

```
factor
fiber
fiber_cross_section
fiber_location
field
file
fix
fluid
free
friction_angle
friction_ratio
from
gamma
Gamma
Gauss
generate
GoverGmax
h_in
hardening_parameters_of_yield_surfaces
hardening_parameters_scale_unit
hdf5_file
hdf5_filenames_list
help
if
imposed
Imx
Imy
Imz
in
inclined
increment
initial_axial_stiffness
initial_confining_stress
initial_elastic_modulus
initial_shear_modulus
initial_shear_stiffness
initial_velocity
integration
integration_points
IntegrationRule
integrator
interface
isotropic_hardening_rate
joint_1_offset
joint_2_offset
K_f
K_s
k_x
k_y
k_z
kappa
kd_in
```

```
kinematic_hardening_rate
lambda
level
line_search_beta
line_search_eta
line_search_max_iter
liquefaction_Alpha
liquefaction_c_h0
liquefaction_Dir
liquefaction_dre1
liquefaction_Dre2
liquefaction_EXPN
liquefaction_gamar
liquefaction_mdc
liquefaction_mfc
liquefaction_pa
liquefaction_pmin
load
load_factors_list
loading
local_y_vector
local_z_vector
M
M_in
magnitude
magnitudes
mass
mass_density
master
material
max_axial_stiffness
maximum_iterations
maximum_number_of_iterations
maximum_strain
maximum_stress
maximum_time_step
method
minimal
minimum_time_step
model
model
moment_x_stiffness
moment_y_stiffness
monotonic
motion
mu
mx
my
mz
name
NDMaterial
new
```

```
newton_with_subincrement
node
nodes
number_of_cycles
number_of_files
number_of_increment
number_of_integration_points
number_of_iterations
number_of_layers
number_of_modes
number_of_subincrements
number_of_times_reaching_maximum_strain
of
output
output
output_filename
p0
parallel
peak_friction_coefficient_limit
peak_friction_coefficient_rate_of_decrease
penalty_stiffness
pi1
pi2
pi3
plastic_deformation_rate
plastic_flow_kd
plastic_flow_xi
plot
point
points
poisson_ratio
poisson_ratio_h_h
poisson_ratio_h_v
porosity
print
propagation
pure
R0
radiuses_of_yield_surface
radiuses_scale_unit
rate_of_softening
reduction
reference_pressure
remove
rempve
residual_friction_coefficient
restart
restart_files
results
rho_a
rho_f
rho_s
```

```
rho_w
RMC_e
RMC_eta0
RMC_Heta
RMC_m
RMC_pc
RMC_qa
RMC_shape_k
rounded_distance
runTest
sanisand2004_A0
sanisand2004_c
sanisand2004_ch
sanisand2004_cz
sanisand2004_ec_ref
sanisand2004_G0
sanisand2004_h0
sanisand2004_lambda_c
sanisand2004_m
sanisand2004_Mc
sanisand2004_nb
sanisand2004_nd
sanisand2004_p_cut
sanisand2004_Pat
sanisand2004_xi
sanisand2004_z_max
sanisand2008_A0
sanisand2008_alpha_cc
sanisand2008_c
sanisand2008_ch
sanisand2008_cz
sanisand2008_ec_ref
sanisand2008_G0
sanisand2008_h0
sanisand2008_K0
sanisand2008_k_c
sanisand2008_lambda
sanisand2008_m
sanisand2008_nb
sanisand2008_nd
sanisand2008_p0
sanisand2008_p_in
sanisand2008_p_r
sanisand2008_Pat
sanisand2008_rho_c
sanisand2008_theta_c
sanisand2008_X
sanisand2008_xi
sanisand2008_z_max
save
scale_factor
SCOTCHGRAPHPARTITIONER
```

```
section
section_number
sequential
series_file
set
shear
shear_length_ratio
shear_modulus
shear_modulus_h_v
shear_stiffness
shear_viscous_damping
shear_zone_thickness
ShearStrainGamma
sigma0
simulate
single
size_of_peak_plateau
sizes_of_yield_surfaces
slave
slave
soil
soil_profile_filename
soil_surface
solid
solver
stage
steps
steps
stiffening_rate
stiffness_to_use
strain
strain_at_compressive_strength
strain_at_crushing_strength
strain_hardening_ratio
strain_increment_size
stress
stress_increment_size
stress_relative_tolerance
sub-stepping
surface
surface_vector_relative_tolerance
tensile_strength
tensile_yield_strength
tension_softening_stiffness
test
test
testing
thickness
time_step
to
tolerance_1
tolerance_2
```

```
torsion_Jx
torsional_stiffness
TorsionConstant_GJ
total_number_of_shear_modulus
total_number_of_yield_surface
triaxial
type
uniaxial
uniaxial_material
uniform
unit_of_acceleration
unit_of_damping
unit_of_rho
unit_of_vs
use
using
value
velocity_file
velocity_scale_unit
verbose_output
viscoelastic_modulus
von_mises_radius
wave
wave1c
wave3c
while
whos
with
xi_in
xz_plane_vector
yield_function_relative_tolerance
yield_strength
yield_surface_scale_unit
x
y
z
```

Second Order (inside commands)

```
20NodeBrick
20NodeBrick_up
20NodeBrick_upU
27NodeBrick
27NodeBrick_up
27NodeBrick_upU
3NodeShell_ANDES
4NodeShell_ANDES
4NodeShell_MITC4
4NodeShell_NewMITC4
8_27_NodeBrick
8_27_NodeBrick_up
8_27_NodeBrick_upU
```

```
8NodeBrick
8NodeBrick_fluid_incompressible_up
8NodeBrick_up
8NodeBrick_upU
Absolute_Norm_Displacement_Increment
Absolute_Norm_Unbalanced_Force
arclength_control
Average_Norm_Displacement_Increment
Average_Norm_Unbalanced_Force
Backward_Euler
BARDETMETHOD
beam_9dof_elastic
beam_displacement_based
beam_elastic
beam_elastic_lumped_mass
BeamColumnDispFiber3d
BearingElastomericPlasticity3d
BFGS
BondedContact
CamClay
Caughey3rd
Caughey4th
constant mean pressure triaxial strain control
Cosserat8NodeBrick
Cosserat_linear_elastic_isotropic_3d
Cosserat_von_Mises
DIRECT_STRAIN
displacement_control
DOFTYPE
domain reduction method
drained triaxial strain control
DruckerPrager
DruckerPragerArmstrongFrederickLE
DruckerPragerArmstrongFrederickNE
DruckerPragerMultipleYieldSurface
DruckerPragerMultipleYieldSurfaceGoverGmax
DruckerPragerNonAssociateArmstrongFrederick
DruckerPragerNonAssociateLinearHardening
DruckerPragervonMises
dynamic
eigen
elastic3d
Elastic_Membrane_Plate
ElasticFourNodeQuad
Energy_Increment
equal_dof
F_fluid_x
F_fluid_y
F_fluid_z
FiberSection
ForceBasedCoupledHardContact
ForceBasedCoupledSoftContact
```

```
ForceBasedElasticContact
ForceBasedHardContact
ForceBasedSoftContact
FORCETYPE
Forward_Euler
Forward_Euler_Subincrement
from_reactions
Fx
Fy
Fz
Hilber_Hughes_Taylor
HyperbolicDruckerPragerArmstrongFrederick
HyperbolicDruckerPragerLinearHardening
HyperbolicDruckerPragerNonAssociateArmstrongFrederick
HyperbolicDruckerPragerNonAssociateLinearHardening
linear
linear_elastic_crossanisotropic
linear_elastic_isotropic_3d
linear_elastic_isotropic_3d_LT
Membrane_Plate_Fiber
Modified_Newton
Mx
My
Mz
Newmark
Newton
non_converged_iterations
NonlinearFourNodeQuad
Norm_Displacement_Increment
Norm_Unbalance
Parallel
path_series
path_time_series
petsc
petsc_options_string
physical_element_group
physical_node_group
Pisano
PlaneStressLayeredMaterial
PlaneStressRebarMaterial
PlasticDamageConcretePlaneStress
pressure
ProfileSPD
Rayleigh
reaction forces
reactions
Relative_Energy_Increment
Relative_Norm_Displacement_Increment
Relative_Norm_Unbalance
Relative_Norm_Unbalanced_Force
roundedMohrCoulomb
RoundedMohrCoulombMultipleYieldSurface
```

```
sanisand2004
sanisand2004_legacy
sanisand2008
self_weight
ShearBeam
solid fluid interaction transient
static
StressBasedCoupledHardContact_ElPPlShear
StressBasedCoupledHardContact_NonLinHardShear
StressBasedCoupledHardContact_NonLinHardSoftShear
StressBasedCoupledSoftContact
StressBasedCoupledSoftContact_ElPPlShear
StressBasedCoupledSoftContact_NonLinHardShear
StressBasedCoupledSoftContact_NonLinHardSoftShear
StressBasedHardContact_ElPPlShear
StressBasedHardContact_NonLinHardShear
StressBasedHardContact_NonLinHardSoftShear
StressBasedSoftContact_ElPPlShear
StressBasedSoftContact_NonLinHardShear
StressBasedSoftContact_NonLinHardSoftShear
SuperElementLinearElasticImport
support
surface
transient
truss
TsinghuaLiquefactionModelCirclePiPlane
TsinghuaLiquefactionModelNonCirclePiPlane
UMFPack
undrained simple shear
undrained triaxial
undrained triaxial stress control
uniaxial_concrete02
uniaxial_elastic
uniaxial_steel01
uniaxial_steel02
variable transient
variable_node_brick_8_to_27
vonMises
vonMisesArmstrongFrederick
vonMisesMultipleYieldSurface
vonMisesMultipleYieldSurfaceGoverGmax
With_no_convergence_check

beta
gamma
delta

ux
uy
uz
rx
ry
```

```
rz
Ux
Uy
Uz
p
M
m
kg
s
cm
mm
km
Hz
Minute
Hour
Day
Week
ms
ns
N
kN
Pa
kPa
MPa
GPa
pound
lbm
lbf
inch
in
feet
ft
yard
mile
psi
ksi
kip
g
pi

NUMBER_OF_NODES
NUMBER_OF_ELEMENTS
CURRENT_TIME
NUMBER_OF_SP_CONSTRAINTS
NUMBER_OF_MP_CONSTRAINTS
NUMBER_OF_LOADS
IS_PARALLEL
SIMULATE_EXIT_FLAG
then
while
do
let
```

```
vector

cos
sin
tan
cosh
sinh
tanh
acos
asin
atan
atan2
sqrt
exp
log10
ceil
fabs
floor
log
```

1.6 Integrated Development Environment (IDE) for DSL

1.7 Mesh Generation using GiD

1. Download the latest version of GiD from `http://www.gidhome.com/`, and also get a temporary license (or purchase it...).

2. Download essi.gid.tar.gz, unpack it (`tar -xvzf essi.gid.tar.gz`) in `problemtypes` directory that is located in GiD's root directory.

3. When you run GiD, you will see `essi` in "Data > Problem types", and can start using it...

4. A simple movie with instructions for mesh generation is available: (Link to a movie, 11MB).

1.8 Model Development and Mesh Generation using gmesh

1.9 Model Input File Editing using Sublime

http://www.sublimetext.com/