# Real-ESSI Simulator

# Output Formats

---

José Abell, Yuan Feng, Sumeet Kumar Sinha, Han Yang

and

Boris Jeremić

University of California, Davis, CA

Version: 3Jul2025, 10:20

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

# Output Formats 2012-2014-

(In collaboration with Mr. José Abell and Mr. Sumeet Kumar Sinha)

## 1.1 Chapter Summary and Highlights

## 1.2 Introduction

All output from ESSI simulator is stored inside a database format, specifically designed for handling scientific array-oriented data, called HDF5 (Group, 2020). HDF stands for 'Hierarchical Data Format' and is a self-describing data format suitable for portable sharing of scientific data. The format was created and is maintained by the HDF group (http://www.hdfgroup.org/)

Data is stored within the file using a hierarchy similar to a unix filesystem, with groups to store related data and the actual data stored in so-called 'datasets' within each group.

HDF5 was chosen because it meets our design goals of provides:

- A simplified output format. Output is a single HDF5 file per analysis stage.
- An efficient binary (possibly compressed) file format that optimizes random access to data.
- A data format that is amenable to store output from parallel computations.
- Has a reasonable API exposed in several languages so that users can easily and customizably access simulation data.

One very convenient tool for the basic exploration of HDF5 files is the viewer 'hdfview' (http://www.hdfgroup.org/products/java/hdfview/index.html).

## 1.3 Output Filename and Format

On running any simulation on Real-ESSI simulator output files are produced for each analysis stage. The number of outputs and the filename is slightly different for sequential and parallel runs. Each output file, contains the information about the model mesh, nodal displacements, elements output, boundary conditions, material tags.. etc. The output files are designed as completely independent files containing all the data for the loading stage. In parallel each of the follower compute process outputs contains all the data corresponding to only the follower compute process. This is done to make the visualization and output process efficient.

### 1.3.1 Sequential

For sequential runs a single output file is produced per analysis stage. The files are named according to `model` and `stage names`, *not* by the filename that runs the analysis. The extension is set to be '`.h5.feioutput`', to distinguish from future possible alternative output formats.

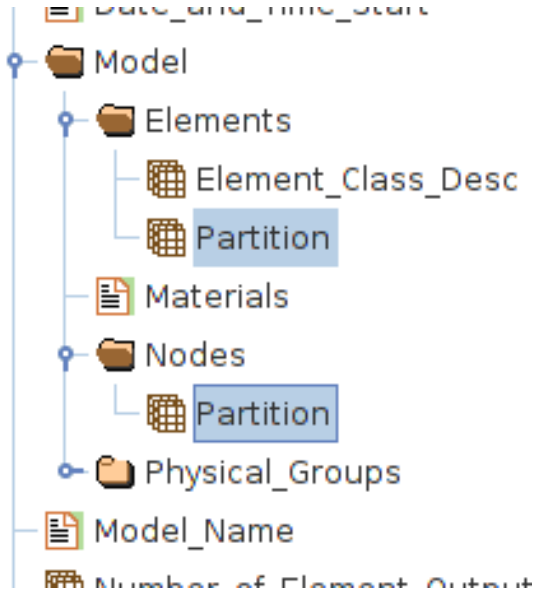


Figure 1.1: Partition info in output file produced by main compute node in parallel run

For example, if the model name is '`site_response`' and the stage name is '`earthquake_shaking`' the corresponding output filename will be '`site_response_earthquake_shaking.h5.feioutput`'.

### 1.3.2 Parallel

In parallel, for each stage, output files produced are equal to the number of CPU's used. For example, a simulation run of 8 CPU's will produce 8 output files per stage for each corresponding CPU's (cores).The filename remains the same as sequential output each CPU (process id) used, but the extension is set to be as '`.h5.pid.feioutput`', where *pid* refers to the process id of the CPU. However, the main compute process having *pid* equal to 0 follows the extension '`.h5.feioutput`'.

For example, In parallel, if the model name is '`site_response`' and the stage name is '`earthquake_shaking`' and the analysis is run on $n$ CPU's, the corresponding output filename for the main compute process ($pid = 0$) would be '`site_response_earthquake_shaking.h5.feioutput`'. All the follower compute process having $pid > 0$ would have output filename '`site_response_earthquake_shaking.h5.pid.feioutput`'.

The main compute process usually does not contain any nodes and element once the partition is achieved and nodes and elements are transfered to their respective CPU's or cores. Thus, the output produced by main compute process does not contain any mesh or output results. However, it contains the partition data as shown in Figure 1.1 describing the process id on which any node or element is assigned. This information is quite useful, during post processing when the result of a particular node or element needs to be extracted. The main compute output can be read to find out the follower compute process id on which the data is located and then the output of that process id can be read to get the data of interest.

## 1.4 Output Units

Real-ESSI `.feioutput` file stores all the results in standard units. The table below shows the units of all the data stored in HDF5 file.

| Quantity | Unit |
|---|---|
| Force $F_x, F_y, F_z$ | $N$ |
| Moments $M_x, M_y, M_z$ | $N-m$ |
| Pressure $p$ | $Pa$ |
| Displacement $u_x, u_y, u_z$ | $m$ |
| Rotation $r_x, r_y, r_z$ | $radian$ |
| Stress $\sigma$ | $Pa$ |
| Strain $\epsilon$ | unit less |
| Acceleration $a_x, a_y, a_z$ | $m/s^2$ |
| Time $t$ | $s$ |

## 1.5 Data organization

In HDF5 jargon a multidimensional array is called a *dataset*. Datasets are indexed arrays (up to 32 dimensions) that can contain different types of data. Supported data types are: integers (various sizes), floating point numbers (`float`, `float`, `long double`, etc.), strings of text (fixed and variable size `char *`), and arbitrary structures of data (similar to C language `struct`). A file can contain as many independent datasets as needed. Datasets can be organized into 'groups', which are like folders in a file system. HDF5 provides additionally convenience data-types such as 'references', which provide views (slices) into diferent datasets or portions of them.

In the particular case of the ESSI HDF5 output, the files are designed with the contents and structure explained hereafter and depicted in Figure 1.3.

### 1.5.1 The `Root` group

The root of the HDF5 file contains information about each stage of loading. In parallel simulations, the information corresponds to the process Id (follower compute node) involved in that stage. The objects under this group are shown in Figure 1.4 and described in List 1.5.1

- `time` : (`float`) A floating point array named which contains the available time steps for this analysis.
- `Number_of_Time_Steps`: (`int`) A single scalar integer array with the number of times steps.

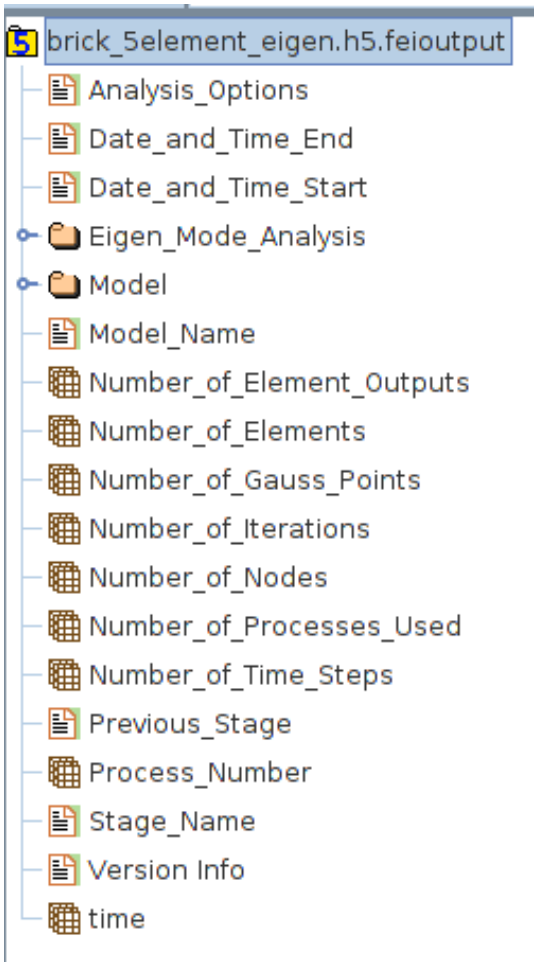


Figure 1.2: Output from a typical analysis.

- `Model_Name`: (`string`) A single string with the model name.
- `Stage_Name`: (`string`) A single string with the stage name.
- `Previous_Stage`: (`string`) A single string containing previous stage name.
- `Process_Number`: (`int`) An integer representing the *process id* by which output was generated. For sequential runs, *process id* would be zero. In parallel runs, *process id* corresponds to the *mpi rank* or *follower compute id* of processor involved in computation.
- `Number_of_Processes_Used`: (`int`) An integer representing total number of processors/CPU/nodes used in the simulation. For sequential runs, it is equal to one whereas for parallel runs, it is equal to the number of CPU's/Cores used.
- `Number_of_Nodes`: (`int`) A single scalar integer array with the number of nodes defined in that domain.

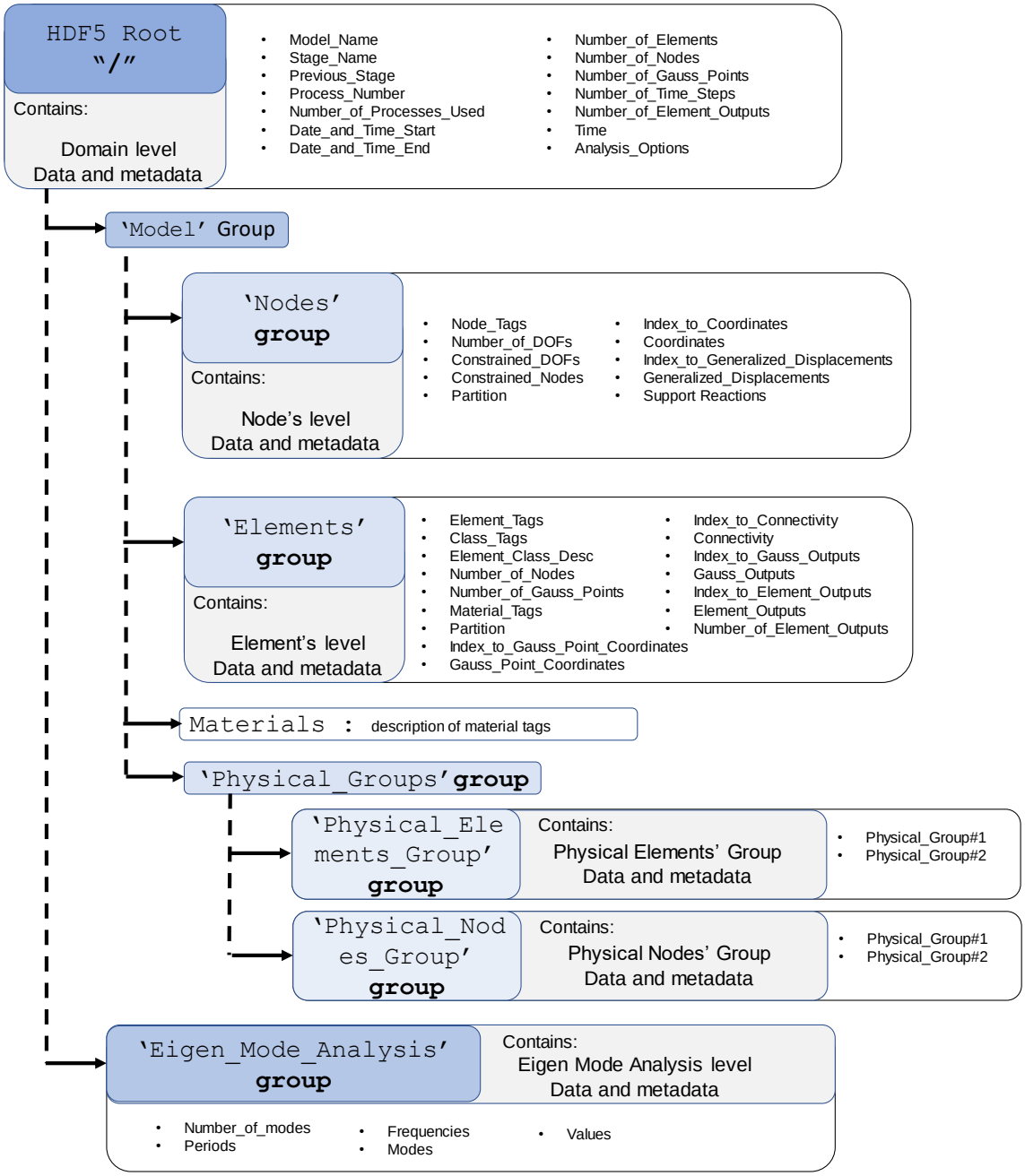


Figure 1.3: Design of the ESSI `.h5.feioutput` data format.

- `Number_of_Elements`: (`int`) A single scalar integer array with the number of elements defined in that domain.
- `Number_of_Gauss_Points`: (`int`) A single scalar integer array with the number of gauss points in that domain.

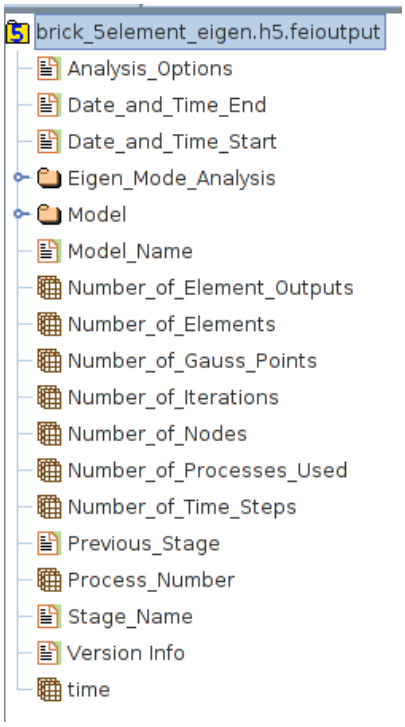


Figure 1.4: Data accessible in the `Root` directory of HDF5 file.

- `Number_of_Element_Outputs`: (`int`) A single scalar integer array with stores the total length of `Element_Output` array in that domain.
- `Analysis_Options`: (`string`) An array of strings with the analysis options selected for the current analysis.
- `Date_and_Time_Start`: (`string`) A single string with the Date and Time of the start of the analysis. (In Coordinated Universal Time, UTC)
- `Date_and_Time_End`: (`string`) A single string with the Date and Time of the end of the analysis. (In Coordinated Universal Time, UTC)

  item `Version_Info`:(`string`) A Long string containing the version information of Real-ESSI simulator.
- `Model`: A group that contains the `Nodes` and `Elements` groups. It contains essential information about the mesh and analysis results for nodes and elements. See Section 1.5.2
- `Eigen_Mode_Analysis`: A group that contains the information about the the eigen mode analysis results of the doamin. See Section 1.5.6

### 1.5.2 The `Model` group

The `Model` group contains information about the mesh and analysis outputs. It contains the following groups as shown in Figure 1.5 and is also described below

- `Nodes`: A group that contains the information about the defined nodes and their output for this analysis. See Section 1.5.3
- `Elements`: A group that contains the information about the defined Elements and their output for this analysis. See Section 1.5.4
- `Physical_Groups`: A group that contains the information physical group of elements and nodes defined in that domain. See Section 1.5.5
- `Material`: (`string`) A string array which contains information about the material tag defined in the analysis for that loading stage. Section 1.5.7

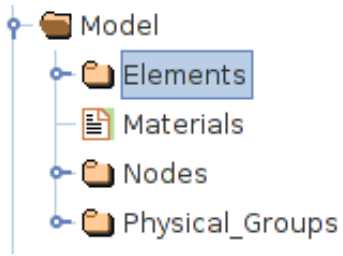


Figure 1.5: `Model` group directory of HDF5 file.

Subgroups `Nodes` and `Elements` store several integer and double precision arrays, that contain all necessary information for post processing.

### 1.5.3 The `Nodes` group

The `Nodes` group contains information about the nodal coordinates of the model, their tags, the number of DOFs defined at each node, and the corresponding solution variables (DOF results or generalized displacements) for each time step.

The format used to store the data is designed to give the fastest possible access time to the data of interest. Stored within the `Nodes` groups (and also in `Elements`) are two types of arrays: *data arrays* and *index arrays*.

- **Data Arrays ::** Data arrays might be floating-point arrays or integer arrays and have names not starting with '`Index_to_`'.

- **Index Arrays ::** All index arrays are integer arrays and have names starting with the word 'Index_to_' followed by the name of the array which this array indexes.

The concept of *index array* is an important one regarding speed of access to data. These arrays map the integer tag number of the nodes (or elements) to the data. This allows fast access to components which minimizes searching within arrays to find the data of interest.

The Nodes group contains the following index arrays.

- Index_to_Coordinates: (int) Indexes the coordinates of nodes. See section 1.5.3
- Index_to_Generalized_Displacements: (int) Indexes the outputs of nodes (generalized displacements). See section 1.5.3

The following are the data arrays available in the Nodes group (shown in Figure **??** along with their respective indexing array:

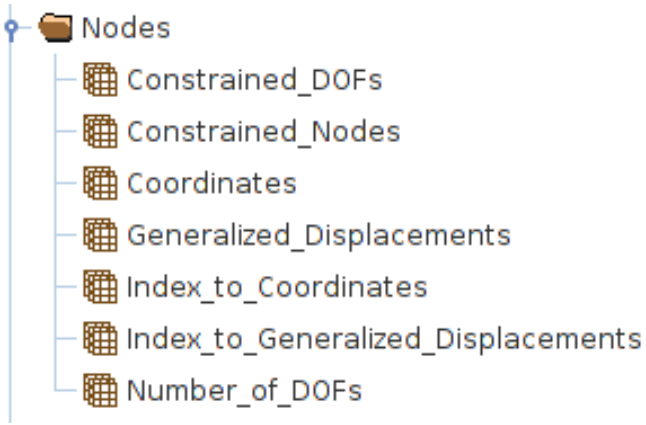


Figure 1.6: Nodes group directory of HDF5 file.

- Coordinates: (float) 1-D array containing nodal coordinates fixed in time. [Indexed by Index_to_Coordinates array]. See section 1.5.3
- Generalized_Displacements: (float) 2-D array containing the DOF values for the solution at each time step. [Indexed by Index_to_Generalized_Displacements array]. See section 1.5.3
- Number_of_DOFs: (int) 1-D array mapping the integer tag of each node to the number of DOFs at that node. See section 1.5.3
- Constrained_Nodes: (int) 1-D array mapping the integer tag of each node to the number of DOFs at that node. See section 1.5.3

- `Constrained_DOFs`: (`int`) 1-D array mapping the integer tag of each node to the number of DOFs at that node. See section 1.5.3
- `Partition`: (`int`) 1-D array mapping the integer tag of each node to the number of DOFs at that node. See section 1.5.3
- `Support_Reactions`: (`float`) 1-D array mapping the integer tag of each node to the number of DOFs at that node. See section 1.5.3

For example, lets imagine that the user has defined 4 nodes and applied the following constrained as shown below :

Listing 1.1: Node_Example

```
1  // defining nodes
2  add node # 2 at (0*m, 0*m, 0*m) with 3 dofs;
3  add node # 4 at (1*m, 1*m, 1*m) with 6 dofs;
4  add node # 5 at (2*m, 2*m, 2*m) with 3 dofs;
5  add node # 6 at (1*m, 0*m, 5*m) with 3 dofs;
6
7  // applying constraints
8  fix node # 2 dofs ux uy;
9  fix node # 4 rx ry rz;
10 fix node # 6 Ux p;
```

The index and the data arrays for the given example would look like the following as shown in the subsections ahead.

#### `Number_of_DOFs`

`Number_of_DOFs` array defines the number of degrees of freedom for each node defined in the model. It is an `integer` array of length equal to the maximum node tag + 1 (including tag 0). If a node tag does not exists, the corresponding dofs is output as -1. Figure 1.7 shows how to read `Number_of_DOFs` array. In the given example Listing 1.1 , node tag 2 has 3 degrees of freedom. Similarly, node tag 4 has 6 degrees of freedom and so on.

#### `Partition`

`Partition` array contains the domain or process id on which nodes tags were defined in case of a parallel simulation. For sequnetial runs, this dataset is not available. If a node tag does not exists, the corresponding partition process id is output as -1. Figure 1.7 shows how to read `Partition` array. In the given example Listing 1.1, node tag 2 is assigned to process id 1. Similarly, node tag 4 is assigned to process id 2 and so on.

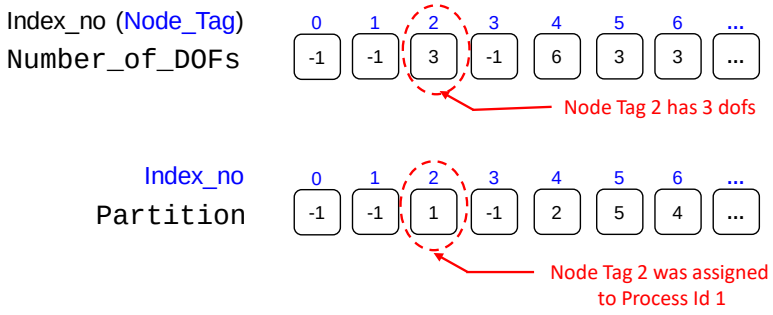


Figure 1.7: Arrays describing node information in `Nodes` group directory of HDF5 file.

### `Constrained_Nodes`

`Constrained_Nodes` array contains a list of node tags for each dof on which fixities were applied. Figure 1.8 shows how to read `Constrained_Nodes`. In the given example Listing 1.1, dof $ux$ and $uy$ of node tag 2 is fixed. Similarly, for node tag 4 dofs $rx$ $ry$ and $rz$ are fixed. Thats why in the `Constrained_Nodes` array contains node tags 2,2,4,4,4 and so on multiple times for each dof of the corresponding node tag fixed.

### `Constrained_DOFs`

`Constrained_DOFs` array contains a list of dofs of the corresponding node tag on which fixities were applied. Figure 1.8 shows how to read `Constrained_DOFs`. In the given example Listing 1.1, dof $ux$ (0) and $uy$ (1) of node tag 2 is fixed. Similarly, for node tag 4 dofs $rx$ (3) $ry$ (4) and $rz$ (5) are fixed. Figure 1.8 also show the DOF if numbering for different dof types i.e. for 3dof, 4dof, 6dof and 7dof nodes.

### `Support_Reactions`

`Support_Reactions` contains a (`float`) array of reaction forces for the constrained degree of freedoms (DOFs). Figure 1.8 shows how to read `Constrained_DOFs`. In the given example Listing 1.1, dof $ux$ (0) and $uy$ (1) of node tag 2 has support reactions 1N and -5N respectively. Similarly, for node tag 4 dofs $rx$ (3) $ry$ (4) and $rz$ (5) have reactions 45, 3 and -5 $N-m$ respectively. The reaction forces for displacement dofs $(u_x, u_y, u_z)$ are forces in units of $N$, for rotational dofs $(r_x, r_y, r_z)$ are moments in units of $N-m$ and for pressure dof $(p)$ is pascal $(Pa)$. Section 1.6 describes the output definitions for nodes with different dof-types.

### `Coordinates`

The `Coordinates` array is a vertical stack of the nodal coordinate values. it is indexed by the `Index_to_Coordinates`, which relates the integer tag of each nodes (defined at the moment of creation of every node) with the position on this array of the 3 nodal coordinates. If the node with a given tag is not defined (if a tag number or several

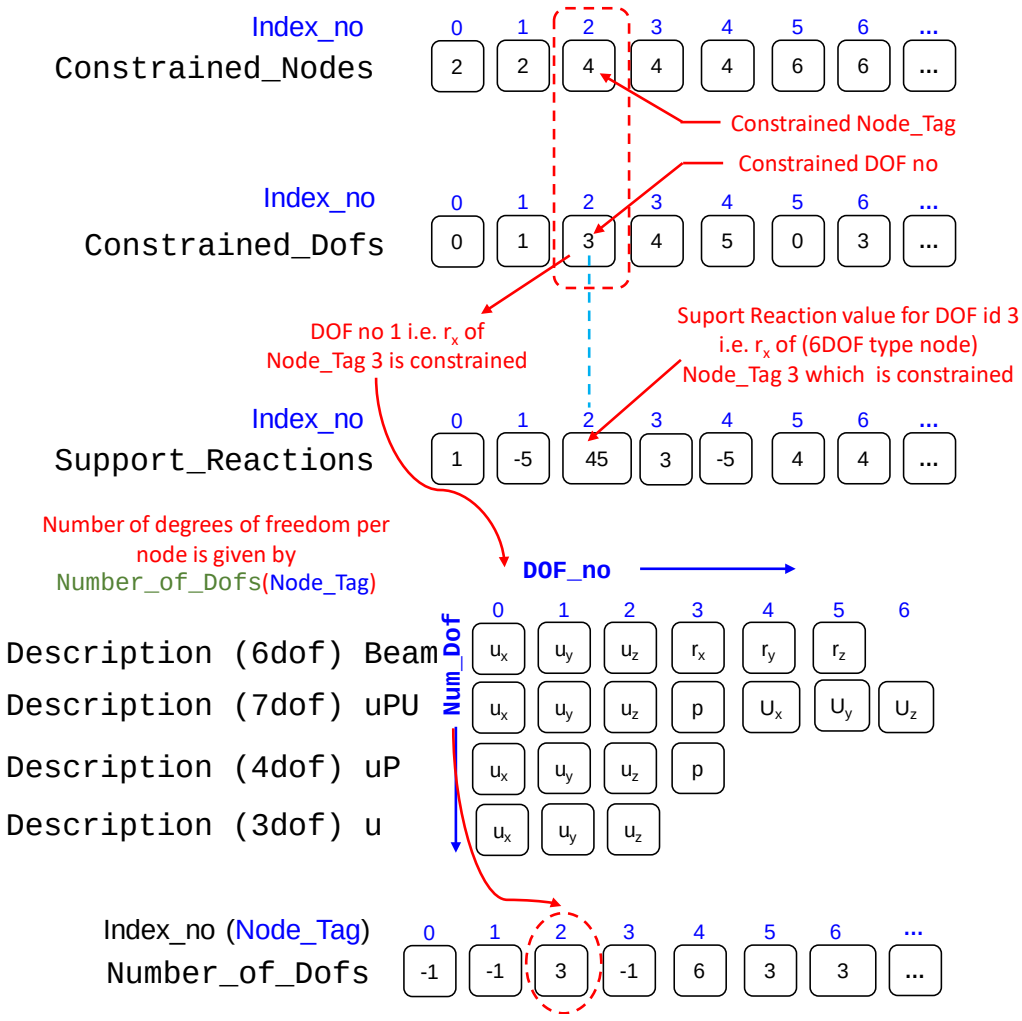


Figure 1.8: Arrays describing constrained nodes and reaction information in `Nodes` group directoory of HDF5 file.

are skipped) this array will contain a negative number (-1) for that tag value. Figure 1.9 shows how to read `Coordinates` and `index_to_Coordinates` of nodes.

The size of `Index_to_Coordinates` is always the maximum tag defined plus one (zero can be a tag too). The size of the `Coordinates` array is three times the number of nodes defined. In the given example Listing 1.1, the coordinates of node tag 2 is (0*m,0*m,0*m). The coordinate of node tag 4 is (1*m,1*m,1*m) and so on. The coordinates have unit of meter ($m$).

### `Generalized_Displacements`

The `Generalized_Displacements` array is a 2-D array containing the computed solution at the nodal degrees of freedom for all nodes and all times steps. It is indexed by the `Index_to_Generalized_Displacements`

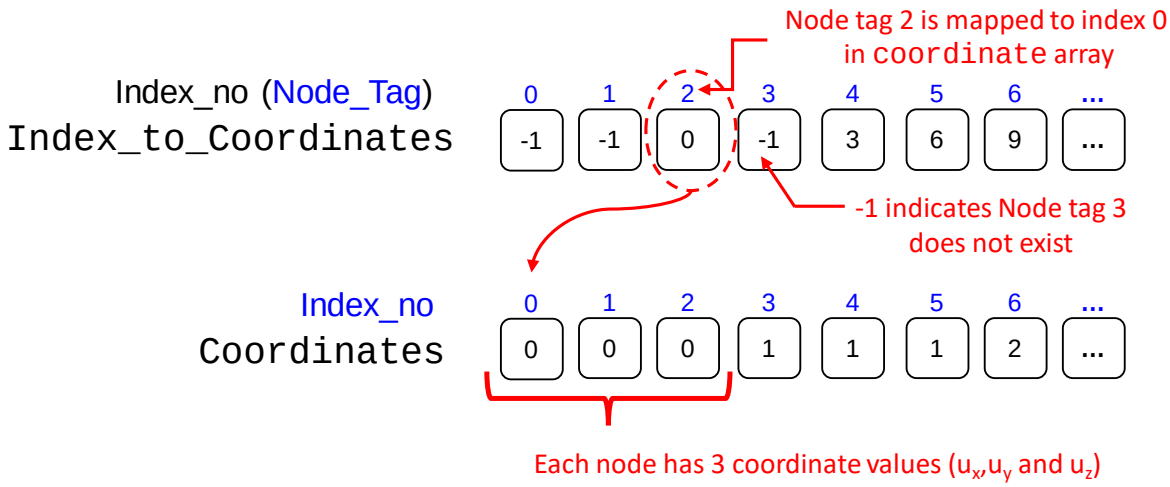


Figure 1.9: `Coordinates` and `Index_to_Coordinates` arrays in `Nodes` group directory of HDF5 file.

array (first index) and time (second index). Figure 1.10 shows how to read `Generalized_Displacements` and `index_to_Generalized_Displacements` of nodes.

Output for displacement dofs $(u_x, u_y, u_z)$ are in units of $m$, for rotational dofs $(r_x, r_y, r_z)$ are in units of $radian$ and for pressure dof $(p)$ is in pascal $(Pa)$.

With every time step, another column is added to the `Generalized_Displacements` array. This means that the time index (starting at 0) is directly related to the `time` array at the root of the HDF5 file.

The rows of the `Generalized_Displacements` array contain the results for each DOF of the current node. For a 3-DOF node these will be displacement in $X$, $Y$, and $Z$ ($u_x$, $u_y$, and $u_z$ ) respectively. For higher number-of-dof nodes the first three components are the same but the remaining ones are going to depend on the connecting elements. For example, 6-DOF nodes might carry information on nodal rotations (beams and shells) or might be fluid displacements in case the elements connecting are u–U coupled fluid-porous-solid elements as shown in Figure 1.8. Section 1.6 describes the output definitions for nodes with different dof-types.

### 1.5.4 The `Elements` group

The `Elements` group contains information on the finite element mesh such as: connectivity array, element types, location of Gauss-point integration coordinates (global), materials defined at each element, and all available output from the elements. At this point it is important to note that the kind and amount of output contained in the element output arrays are dependent on element implementation as it is the elements who are in charge of controlling their output. For information on the specific output of each element, consult the documentation for the respective element.

The idea and organization of the datasets contained herein is analogous to the of the `Nodes` group. This

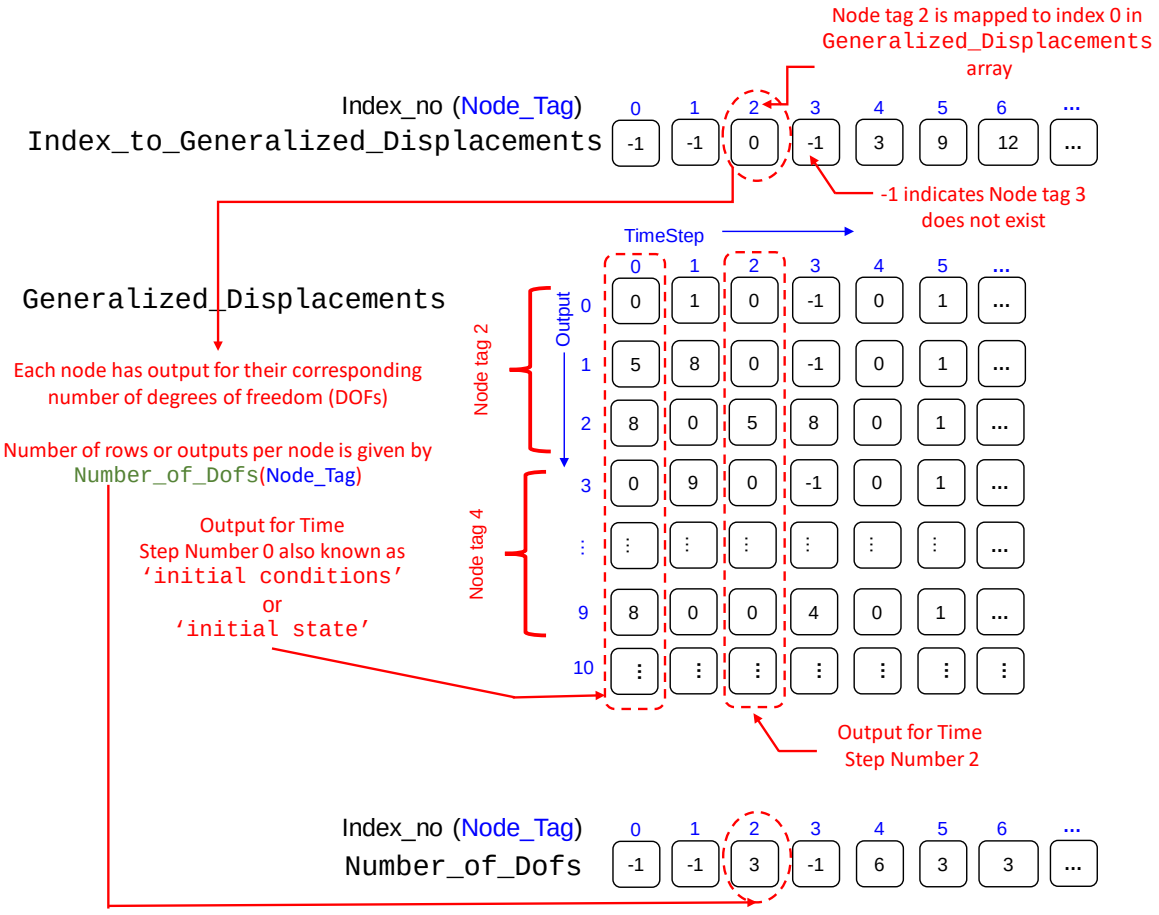


Figure 1.10: The `Index_to_Generalized_Displacements` and `Generalized_Displacements` in `Nodes` group directory of HDF5 file.

group contains the following *index arrays*.

- `Index_to_Connectivity`: (`int`) Maps element tag to location within the `Connectivity` array.
- `Index_to_Gauss_Point_Coordinates`: Maps element tag to location of Gauss-point coordinates in the `Gauss_Point_Coordinates` array.
- `Index_to_Gauss_Outputs`: (`int`) Maps element tag to location of gauss output of element in the `Gauss_Outputs` array.
- `Index_to_Element_Outputs`: (`int`) Maps element tag to location of output in the `Elements_Outputs` array.

The following are the data arrays available in the `Elements` group (shown in Figure **??** along with their respective indexing array:

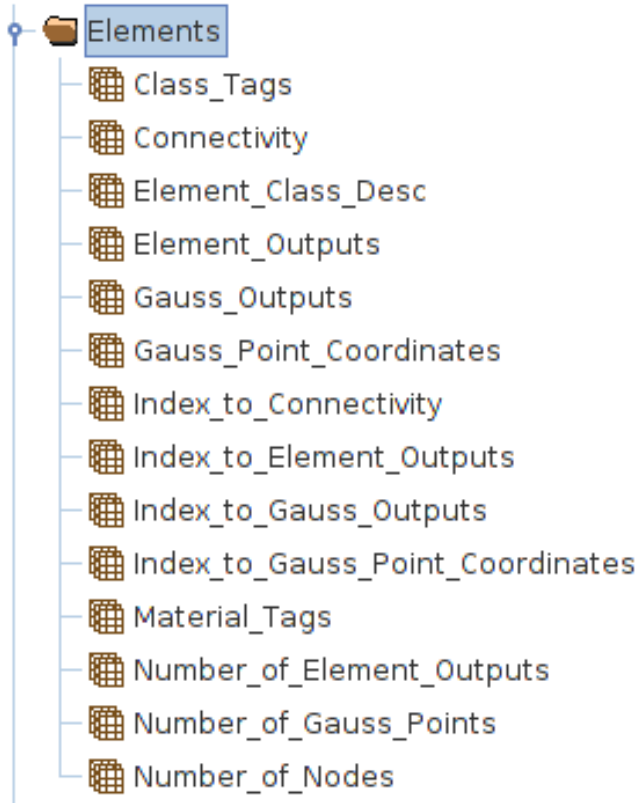


Figure 1.11: `Elements` group directory of HDF5 file.

- `Class_Tags`: (`int`) It is an array that contains the integer ids for each elements tag defined in the model and present in thet domain. See section 1.5.4
- `Number_of_Nodes`: (`int`) Maps element tag number to number of nodes in the element (-1 if element tag is not defined). See section 1.5.4.
- `Number_of_Gauss_Points`: (`int`) Maps tag number to the number of Gauss-integration points in that element (-1 if not element tag is not defined). It stores $0$ in case of no gauss points (mostly for structural elements). See section 1.5.4
- `Material_Tags`: (`int`) Maps tag number to the tag number of the material contained in that element (-1 if element tag is not defined or material for that element is not defined). See section 1.5.4
- `Partition`: (`int`) Maps tag number to the tag number of the processor id on which it is assigned (-1 if element tag is not defined). See section 1.5.4
- `Connectivity`:(`int`) Contains the nodes tags which are connected by this element [indexed by the `Index_to_Connectivity` array] (-1 if element tag is not defined). See section 1.5.4

- Gauss_Point_Coordinates: (float) Contains the coordinates of all the gauss pints of that element tag [indexed by the Index_to_Gauss_Point_Coordinates array] (-1 if element tag is not defined) . See section 1.5.4

- Gauss_Outputs: (float) Contains the stress, strain and plastic strain outputs for each gauss point present in the corresponding element tag[indexed by the Index_to_Gauss_Outputs array]. See section 1.5.4

- Element_Outputs: (float) Contains output other than gauss points by the [indexed by the Index_to_Element_Outputs array]. See section 1.5.4

For example, lets imagine that the user has defined 4 elements and some materials as shown below :

Listing 1.2: Element_Example

```
1
2  // defining materials
3
4  add material #1 type uniaxial_elastic elastic_modulus =  1*Pa ↩
       viscoelastic_modulus =  0*Pa*s;
5  add material #2 type linear_elastic_isotropic_3d mass_density = ↩
       2000*kg/m^3 elastic_modulus = 200*MPa poisson_ratio = 0.3;
6
7  // defining elements
8  add element #2  type truss with nodes (1,2) use material # 1 ↩
       cross_section = 1*m^2 mass_density =  0*kg/m^3;
9  add element #4 type 8NodeBrick with nodes (1,8,6, 4, 3, 9, 2, 5) use ↩
       material #2;
10 add element #5  type HardContact with nodes (3,2) normal_stiffness ↩
       =1e10*N/m tangential_stiffness =  1e4*Pa*m normal_damping =  ↩
       0*kN/m*s tangential_damping =   0*N/m*s friction_ratio =  1 ↩
       contact_plane_vector = (0,0,1);
11 add element #6 type 8NodeBrick with nodes (11,18,61,14, 3,19,22,15) ↩
       use material #2;
```

The index and the data arrays for the given example would look like the following as shown in the subsections ahead.

### Number_of_Nodes

Number_of_Nodes array defines the number of nodes for each element defined in the model. It is an integer array of length equal to the maximum element tag + 1 (including tag 0). If a element tag does not exists, the corresponding number of nodes is output as -1. Figure 1.12 shows how to read Number_of_Nodes array. In the given example Listing 1.2 , element tag 2 has 2 nodes. Similarly, element tag 4 has 8 nodes and so on.

### Number_of_Element_Outputs

Number_of_Element_Outputs array defines the number of outputs for each element defined in the model. It is an integer array of length equal to the maximum element tag + 1 (including tag 0). If a element tag does not exists, the corresponding number of outputs is stored as-1. Figure 1.12 shows how to read Number_of_Element_Outputs array. In the given example Listing 1.2 , element tag 2 has 2 outputs. Similarly, element tag 5 has 9 outputs but element tag 4 has 0 outputs and so on.

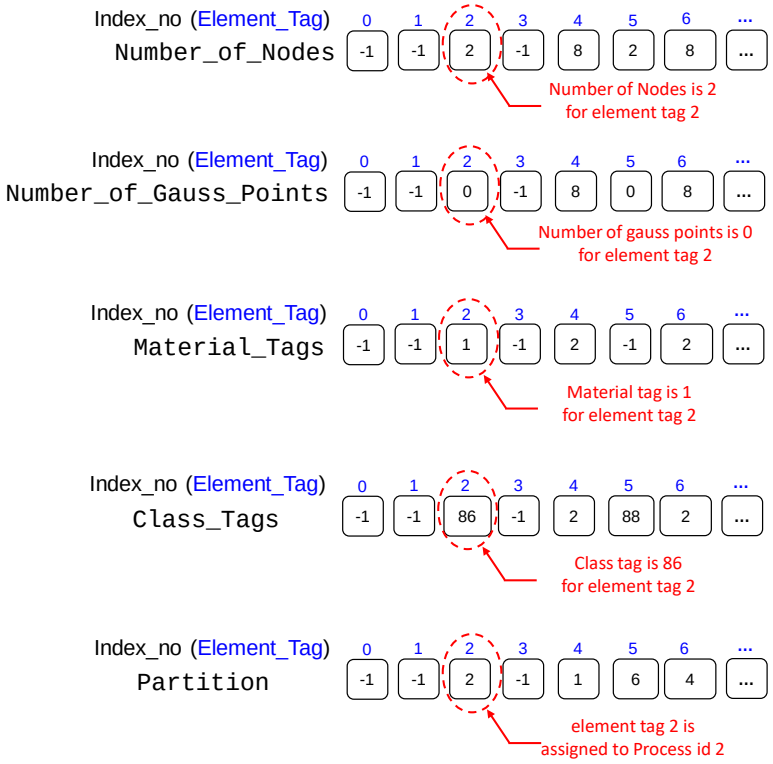


Figure 1.12: Arrays describing element information in Elements group directory of HDF5 file.

### Number_of_Gauss_Points

Number_of_Gauss_Points array defines the number of gauss points for each element defined in the model. It is an integer array of length equal to the maximum element tag + 1 (including tag 0). If a element tag does not exists, the corresponding number of outputs is stored as-1. Figure 1.12 shows how to read Number_of_Gauss_Points array. In the given example Listing 1.2 , element tag 2 and 5 has 0 gauss points. Similarly, element tag 4 has 8 gauss points and so on.

`Class_Tags`

`Class_Tags` array defines an (unique) element type for each of the element defined in the model. It is an `integer` array of length equal to the maximum element tag + 1 (including tag 0). If a element tag does not exists, the corresponding number of outputs is stored as-1. Figure 1.12 shows how to read `Class_Tags` array. In the given example Listing 1.2 , element tag 2 has class tag of 88 (i.e. truss element) . Similarly, element tag 5 has class tag 2 (i.e. 8 node brick element) and so on. Table 1.1 and Table 1.2 shows class tags for different element types.

| Element Type | Class Tag | Element Type | Class Tag |
|---|---|---|---|
| EightNodeBrick | 2 | EightNodeBrick_up | 3 |
| EightNodeBrick_upU | 4 | TwentyNodeBrick | 5 |
| TwentyNodeBrick_up | 6 | TwentyNodeBrick_upU | 7 |
| TwentySevenNodeBrick | 8 | TwentySevenNodeBrick_up | 9 |
| TwentySevenNodeBrick_upU | 10 | VariableNodeBrick | 11 |
| VariableNodeBrick_up | 12 | VariableNodeBrick_upU | 13 |
| EightNodeBrickOrderOne | 14 | EightNodeBrickOrderOne_up | 15 |
| EightNodeBrickOrderOne_upU | 16 | TwentyNodeBrickOrderOne | 17 |
| TwentyNodeBrickOrderOne_up | 18 | TwentyNodeBrickOrderOne_upU | 19 |
| TwentySevenNodeBrickOrderOne | 20 | TwentySevenNodeBrickOrderOne_up | 21 |
| TwentySevenNodeBrickOrderOne_upU | 22 | VariableNodeBrickOrderOne | 23 |
| VariableNodeBrickOrderOne_up | 24 | VariableNodeBrickOrderOne_upU | 25 |
| EightNodeBrickOrderTwo | 26 | EightNodeBrickOrderTwo_up | 27 |
| EightNodeBrickOrderTwo_upU | 28 | TwentyNodeBrickOrderTwo | 29 |
| TwentyNodeBrickOrderTwo_up | 30 | TwentyNodeBrickOrderTwo_upU | 31 |
| TwentySevenNodeBrickOrderTwo | 32 | TwentySevenNodeBrickOrderTwo_up | 33 |
| TwentySevenNodeBrickOrderTwo_upU | 34 | VariableNodeBrickOrderTwo | 35 |
| VariableNodeBrickOrderTwo_up | 36 | VariableNodeBrickOrderTwo_upU | 37 |
| EightNodeBrickOrderThree | 38 | EightNodeBrickOrderThree_up | 39 |
| EightNodeBrickOrderThree_upU | 40 | TwentyNodeBrickOrderThree | 41 |
| TwentyNodeBrickOrderThree_up | 42 | TwentyNodeBrickOrderThree_upU | 43 |
| TwentySevenNodeBrickOrderThree | 44 | TwentySevenNodeBrickOrderThree_up | 45 |
| TwentySevenNodeBrickOrderThree_upU | 46 | VariableNodeBrickOrderThree | 47 |
| VariableNodeBrickOrderThree_up | 48 | VariableNodeBrickOrderThree_upU | 49 |

Table 1.1: Class Tags for Real-ESSI Elements (Part -1)

| Element Type | Class Tag | Element Type | Class Tag |
|---|---|---|---|
| EightNodeBrickOrderFour | 50 | EightNodeBrickOrderFour_up | 51 |
| EightNodeBrickOrderFour_upU | 52 | TwentyNodeBrickOrderFour | 53 |
| TwentyNodeBrickOrderFour_up | 54 | TwentyNodeBrickOrderFour_upU | 55 |
| TwentySevenNodeBrickOrderFour | 56 | TwentySevenNodeBrickOrderFour_up | 57 |
| TwentySevenNodeBrickOrderFour_upU | 58 | VariableNodeBrickOrderFour | 59 |
| VariableNodeBrickOrderFour_up | 60 | VariableNodeBrickOrderFour_upU | 61 |
| EightNodeBrickOrderFive | 62 | EightNodeBrickOrderFive_up | 63 |
| EightNodeBrickOrderFive_upU | 64 | TwentyNodeBrickOrderFive | 65 |
| TwentyNodeBrickOrderFive_up | 66 | TwentyNodeBrickOrderFive_upU | 67 |
| TwentySevenNodeBrickOrderFive | 68 | TwentySevenNodeBrickOrderFive_up | 69 |
| TwentySevenNodeBrickOrderFive_upU | 70 | VariableNodeBrickOrderFive | 71 |
| VariableNodeBrickOrderFive_up | 72 | VariableNodeBrickOrderFive_upU | 73 |
| EightNodeBrickOrderSix | 74 | EightNodeBrickOrderSix_up | 75 |
| EightNodeBrickOrderSix_upU | 76 | TwentyNodeBrickOrderSix | 77 |
| TwentyNodeBrickOrderSix_up | 78 | TwentyNodeBrickOrderSix_upU | 79 |
| TwentySevenNodeBrickOrderSix | 80 | TwentySevenNodeBrickOrderSix_up | 81 |
| TwentySevenNodeBrickOrderSix_upU | 82 | VariableNodeBrickOrderSix | 83 |
| VariableNodeBrickOrderSix_up | 84 | VariableNodeBrickOrderSix_upU | 85 |
| HardContact | 86 | SoftContact | 87 |
| Truss | 88 | ElasticBeam | 89 |
| ThreeNodeAndesShell | 90 | FourNodeAndesShell | 91 |
| ShearBeam | 92 | rank_one_deficient_elastic_pinned_fixed_beam | 93 |
| DispBeamColumn3d | 94 | Cosserat_8node_brick | 95 |

Table 1.2: Class Tags for Real-ESSI Elements (Part -2)

`Partition`

`Partition` array contains the domain or process id on which element tags were defined in case of a parallel simulation. For sequential runs, this dataset is not available. If a element tag does not exists, the corresponding partition process id is output as -1. Figure 1.12 shows how to read `Partition` array. In the given example Listing 1.2, element tag 2 is assigned to process id 2. Similarly, element tag 4 is assigned to process id 1 and so on.

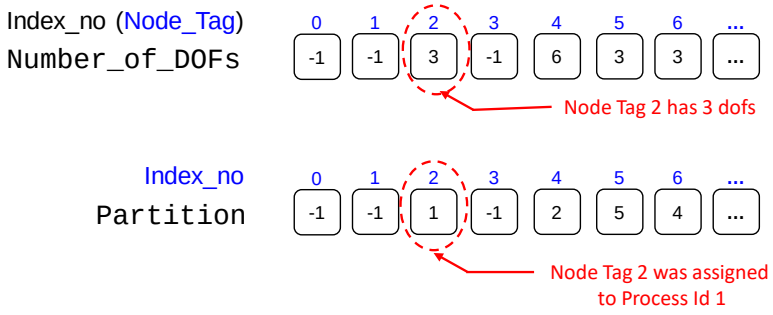


Figure 1.13: Arrays describing node information in `Nodes` group directory of HDF5 file.

`Material_Tags`

`Material_Tags` array defines the material tag number for each element defined in the model. It is an `integer` array of length equal to the maximum element tag + 1 (including tag 0). If a element tag does not exists, the corresponding number of outputs is stored as-1. Figure 1.12 shows how to read `Material_Tags` array. In the given example Listing 1.2 , element tag 2 has material tag of 1. Similarly, element tag 4 has material tag 2. Whereas, element tag 5 have material tag of -1.

`Connectivity`

`Connectivity` array stores the list of node tags in the same order as they were defined along with element declaration in the model. It is an `integer`. It is indexed by the `Index_to_Connectivity` array and the number of rows is defined by `Number_of_Nodes`. Figure 1.14 shows how to read `Connectivity` array for a particular element. In the given example Listing 1.2 , element tag 2 includes node tag 1 and 2. Similarly, element tag 4 includes 8 node tas 1,8,6, 4, 3, 9, 2 and 5 respectively.

`Gauss_Point_Coordinates`

`Gauss_Point_Coordinates` array stores the coordinates of the gauss points declared inside for each element defined in the model. It is a `float` array indexed by the `Index_to_Gauss_Point_Coordinates` array and the number of rows is defined by 3 $(x, y, z)$ times `Number_of_Gauss_Points`. Figure 1.15 shows how to read `Gauss_Point_Coordinates` array for a particular element. In the given example Listing 1.2, element tag 2 and 5 has no gauss points. Since, element tag 4 has 8 gauss points, the total length of gauss point coordinates output for that element is $8 \times 3 = 24$. The index from which the coordinates information start is 0. Coordinate values for first 3 index $(0, 1, 2)$ corresponds to gauss point 1 and next 3 index $(3, 4, 5)$ corresponds to gauss point 2 and so on.

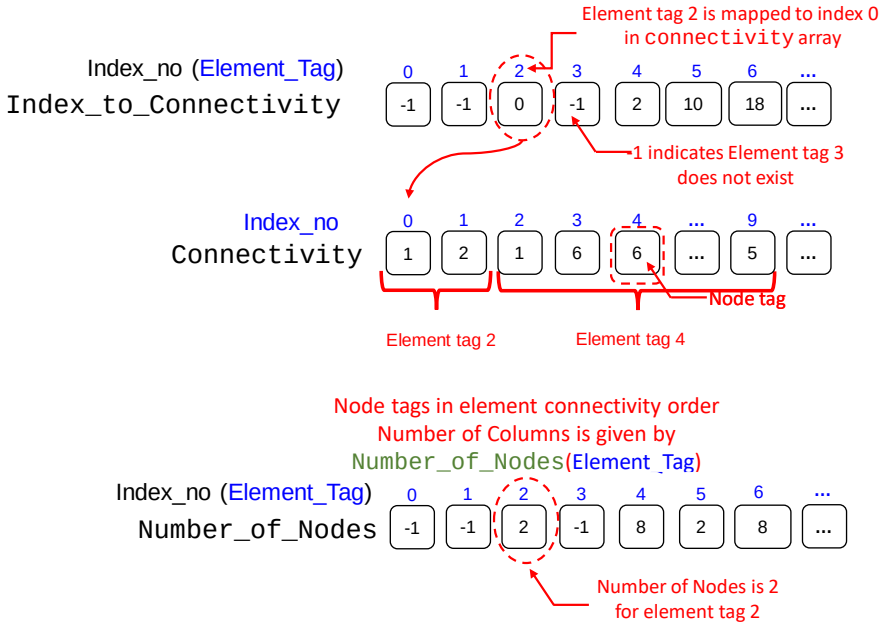


Figure 1.14: Arrays describing connectivity information in `Elements` group directory of HDF5 file.

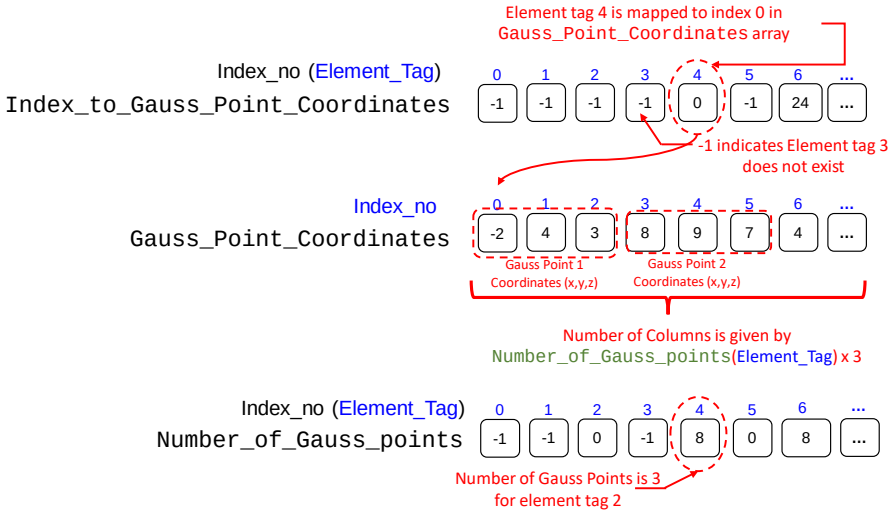


Figure 1.15: Arrays describing gauss coordinates information in `Elements` group directory of HDF5 file.

`Gauss_Output`

`Gauss_Outputs` array stores the coordinates of the gauss points declared inside for each element defined in the model. It is a 2D `float` array indexed by the `Index_to_Gauss_Outputs` array and the number of rows is defined by 18 (6 total strain, 6 plastic strain and 6 stress) times `Number_of_Gauss_Points`. The column index is represented by the time step of the simulation. Time index $0$ represents initial state conditions, i.e. the state before the start of new stage and end of previous stage.

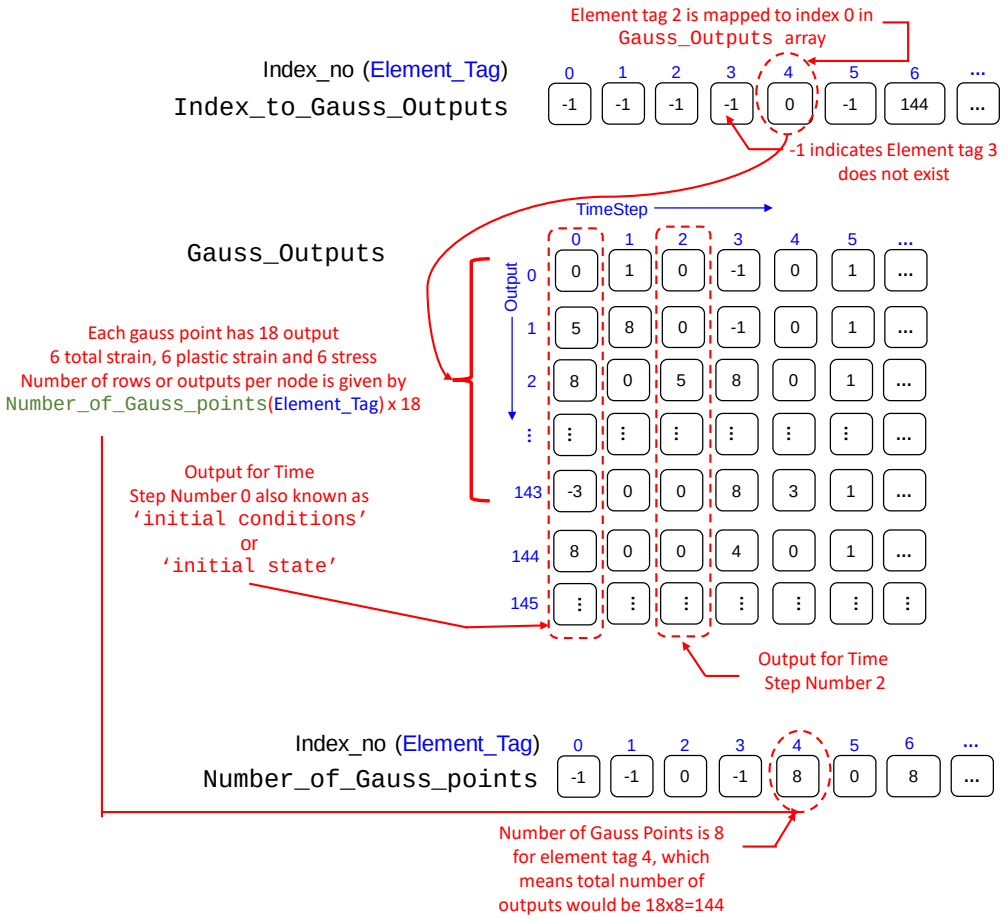


Figure 1.16: Arrays describing gauss output information in `Elements` group directory of HDF5 file.

Figure 1.16 shows how to read `Gauss_Outputs` array for a particular element. In the given example Listing 1.2, element tag 2 and 5 has no gauss points. Since, element tag 4 has 8 gauss points, the total length of gauss output for that element is $8 \times 18 = 144$. The index from which the coordinates information start is 0. gauss values for first $18$ index $(0, 1..17)$ corresponds to gauss point 1 and next 18 index $(18..35)$ corresponds to gauss point 2 and so on. For gauss point 1, first $6$ index $(0, 1..5)$ corresponds to total strain, next $6$ index $(6..11)$ corresponds to plastic strain and next $6$ index $(12..17)$ corresponds to stress. Section 1.7 describes how gauss output is stored and what are the units and meaning of those outputs.

### `Element_Outputs`

`Element_Outputs` array stores output for the elements except those stored at gauss points (i.e. stress, total strain and plastic strain). It is a 2D `float` array indexed by the `Index_to_Element_Outputs` array and the number of rows is defined `Number_of_Element_Outputs`. he column index is represented by the time step of the simulation. Time index $0$ represents initial state conditions, i.e. the state before the start of new stage

and end of previous stage.

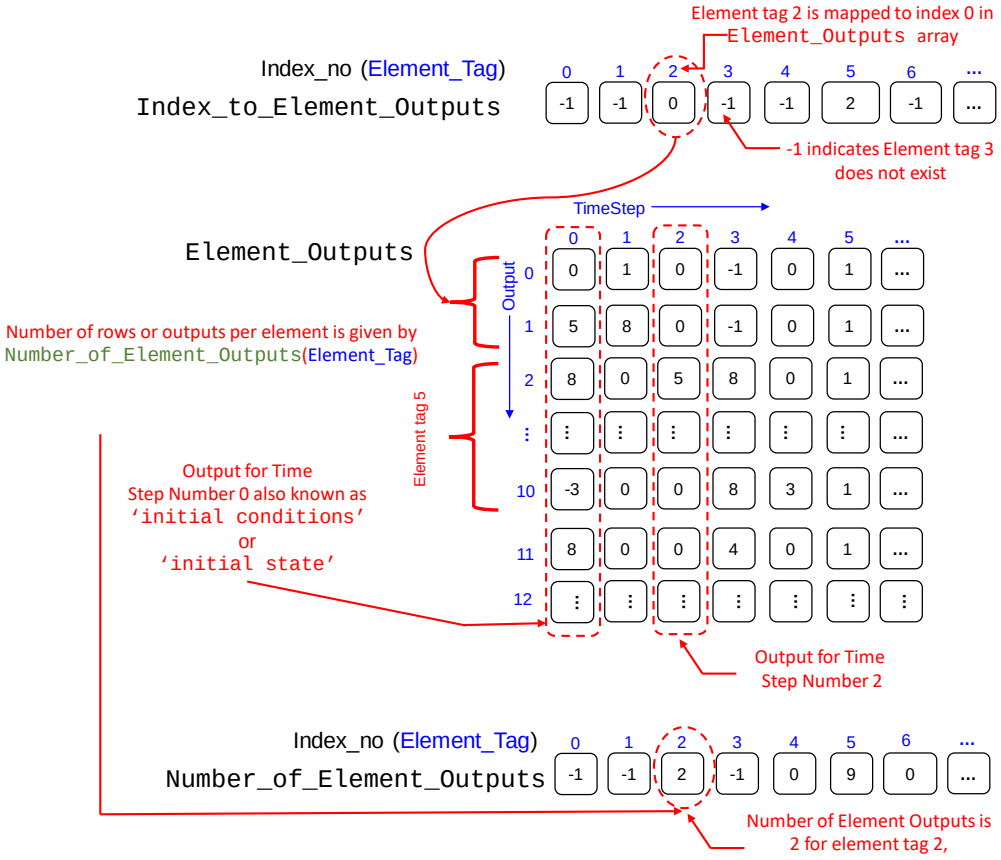


Figure 1.17: Arrays describing output information in `Elements` group directory of HDF5 file.

Figure 1.17 shows how to read `Element_Outputs` array for a particular element and particular time step. In the given example Listing 1.2, element tag 2 has two outputs. Similarly, element tag 5 has 9 outputs. On the other hand element tag $4$ has no output. For element 2, the output starts at row index $0$ and ends at $1$. Similarly, for element tag 4, the output is stored in row index $2..10$.

Since the `Element_Outputs` array format depends on the elements present in the model, one must refer to each element specifically (Section 1.8) to identify what each output component means.

### 1.5.5 The `Physical_Groups` group

`Physical_Groups` group contains the physical groups of nodes or elements defined in the analysis. It contains two subgroups: `Physical_Node_Groups` and `Physical_Element_Groups` as shown in Figure 1.18 and is described in the following sections.

For example, lets imagine that the user has defined one phyical group of nodes and elements respectively as shown below :

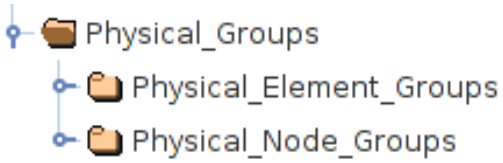


Figure 1.18: `Physical_Groups` group directory of HDF5 file.

Listing 1.3: PhyGrp_Example

```
1 // defining physical groups
2 define physical_node_group ``Physical_Node_Group#1'';
3 define physical_element_group ``Physical_Element_Group#1'';
4
5 // adding items to allready defined physical groups
6 add nodes (1,4,5,7,2,30,42) to physical_node_group ↩
    ``Physical_Node_Group#1''
7 add elements (1,4,5,7,2,30,42)  to physical_node_group ↩
    ``Physical_Element_Group#1''
```

The data arrays for the given example would look like the following as shown in the subsections ahead.

### The `Physical_Element_Groups`

This group contains information about physical groups of elements defined in the analysis. For each of the physical group defined, a new `integer` data array is created inside `Physical_Element_Groups` which stored the element tags belonging to that group. Figure 1.19 shows how to read `Physical_Element_Groups` dataset array for a particular defined physical group.

In the given example Listing 1.3, a physical group array with name ‘‘`Physical_Element_Group#1`’’ is created which contains the list of element tags that were part of that physical group. Figure 1.19 shows that ‘‘`Physical_Element_Group#1`’’ contains element tags 1,4,5,7, and so on;

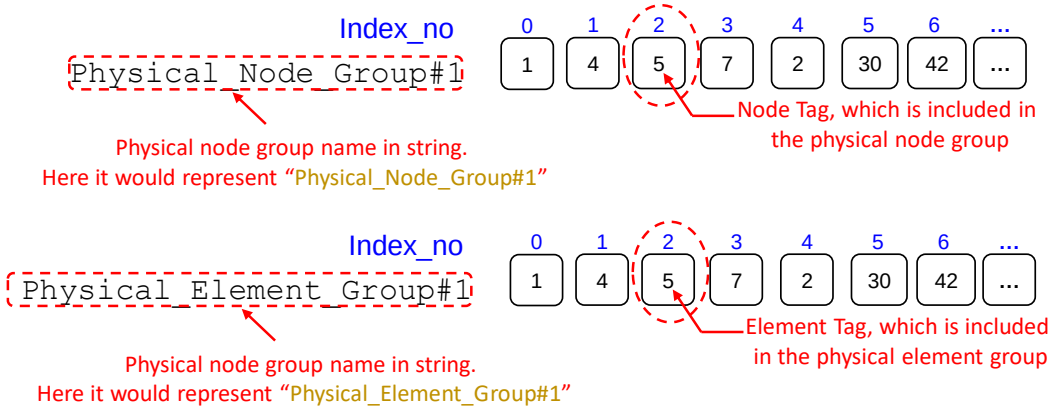


Figure 1.19: Arrays describing physical groups information in `Physical_Groups` group directory of HDF5 file.

#### The `Physical_Node_Groups`

This group contains information about physical groups of nodes defined in the analysis. For each of the physical group defined, a new `integer` data array is created inside `Physical_Node_Groups` which stored the node tags belonging to that group. Figure 1.19 shows how to read `Physical_Node_Groups` dataset array for a particular defined physical group.

In the given example Listing 1.3, a physical group array with name ‘‘`Physical_Node_Group#1`’’ is created which contains the list of node tags that were part of that physical group. Figure 1.19 shows that ‘‘`Physical_Node_Group#1`’’ contains node tags 1,4,5,7, and so on;

### 1.5.6 The `Eigen_Mode_Analysis` group

`Eigen_Mode_Analysis` group gets created in HDF5 .feioutput file after an eigen mode analysis. The data arrays available inside this group are described below and is also shown in Figure 1.20.

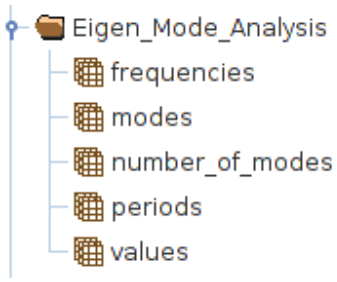


Figure 1.20: `Eigen_Mode_Analysis` group directory of HDF5 file.

`Number_of_Eigen_Modes`

`Number_of_Modes` is an `integer` that stores information about the number of modes solved for the eigen problem. Figure 1.21 shows how to read it.

`Eigen_Frequencies`

`Frequencies` is an `float` array that stores the natural frequencies of the model corresponding to different modes. The length of the array is number of modes + 1. Index ‘0’ does not correspond to any eigen mode and thus stores -1. Figure 1.21 shows how to read it.

`Eigen_Periods`

`Frequencies` is an `float` array that stores the natural periods of the model corresponding to different modes. The length of the array is number of modes + 1. Index ‘0’ does not correspond to any eigen mode and thus

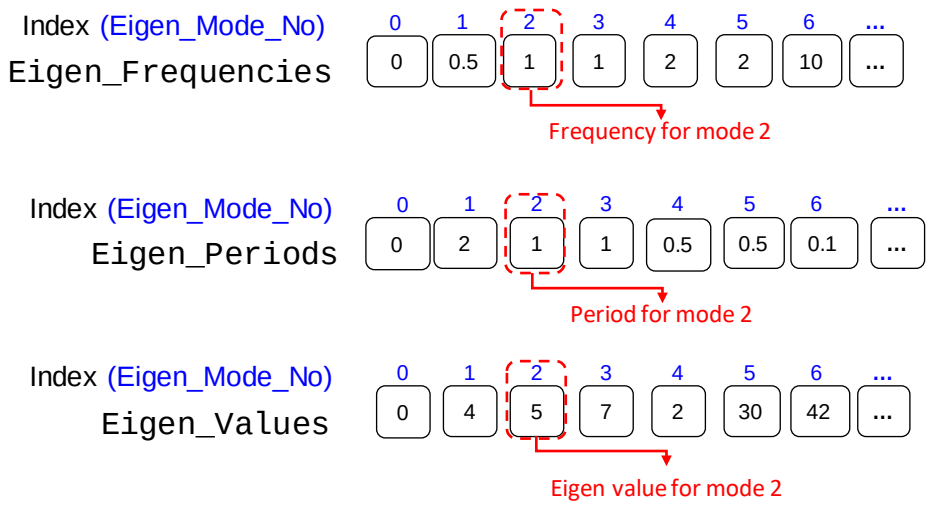


Figure 1.21: `Eigen_Mode_Analysis` group directory of HDF5 file.

stores -1. Figure 1.21 shows how to read it.

`Eigen_Values`

`Frequencies` is an `float` array that stores the natural eigen values of the model corresponding to different modes. The length of the array is number of modes + 1. Index '0' does not correspond to any eigen mode and thus stores -1. Figure 1.21 shows how to read it.

`Modes`

`Modes` is a 2-D `float` array that stores the generalized displacements of the nodes defined in the model corresponding to different modes. The column index no. n this 2-D array defines the mode no i.e. column index 1 corresponds to mode number 1 and so on. Figure **??** shows how to read it.

### 1.5.7 The `Material` data array

Material is a 1-D `string` array that stores information about all materials defined in the model. The length of the array is the maximum material tag + 1 ( including tag 0). The index of this array corresponds to the `material tag`. A value of '-1' means that the material tag was not defined. Figure 1.23 shows the `Material` data array for the example Listing 1.2. Here, index no 1 and index no 2 stores the information corresponding to material tag 1 and 2 respectively.

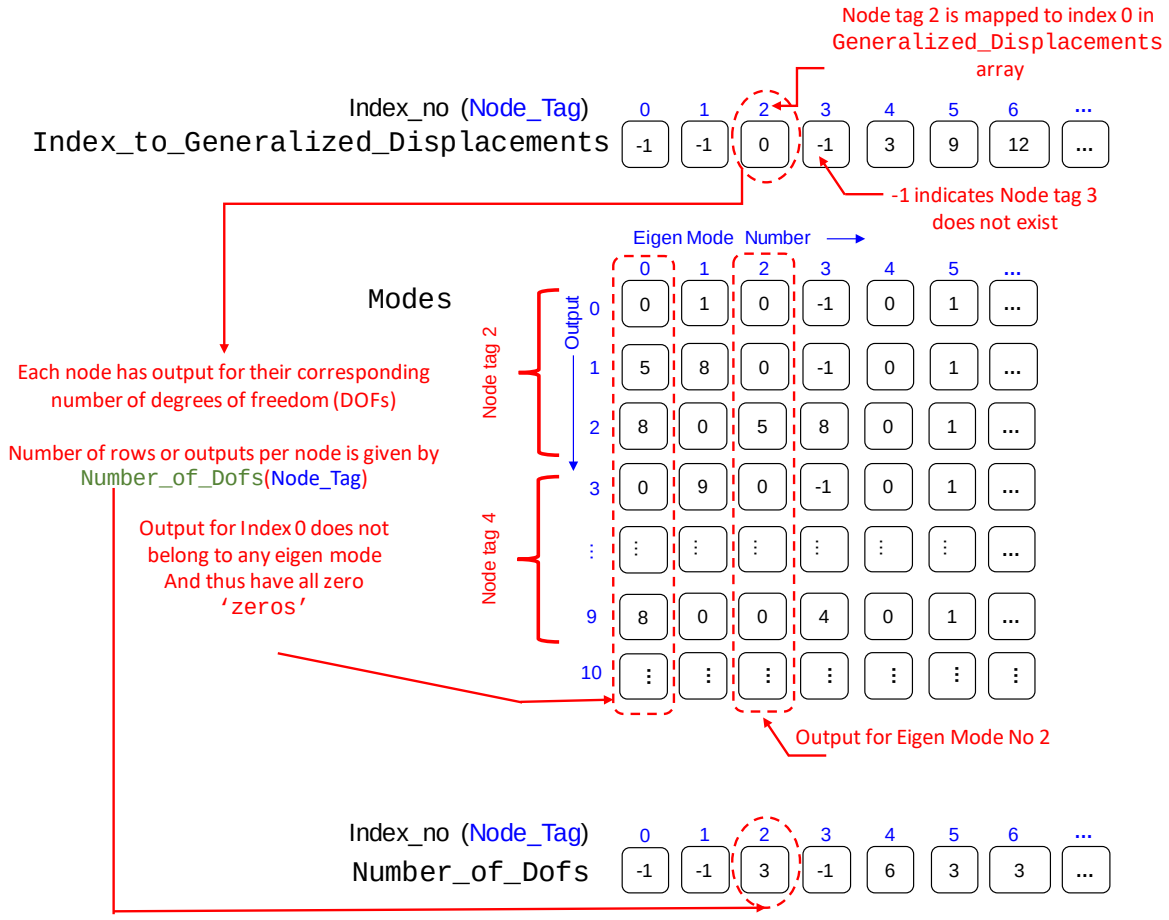


Figure 1.22: `Eigen_Mode_Analysis` group directory of HDF5 file.

| | |
|---|---|
| 0 | 0 |
| 1 | Elastic tag: 1<br>Epos: 1 Eneg: 1 eta: 0 |
| 2 | ElasticIsotropic3D::<br>Tag: 2<br>Elastic_Modulus: 2e+08<br>Poissons_Ratio: 0.3<br>Density: 2000 |

Figure 1.23: `Materials` data array in `Model` directory of HDF5 file.

## 1.6 Node-specific output format

In Real-ESSI simulator, nodes can be defined with different number of degree of freedoms (DOFs). Nodes with with different dofs can be thought of different types of nodes. As a result, their corresponding output also

changes. Here is described, all the different node types available and the output format and their descriptions for each of them. The following subsections would describe the definition of outputs that are expected in `Generalized_Displacements` and `Support_Reactions` data arrays in `Nodes` group of HDF5 output file for node tags of different dof types.

### 1.6.1 3DOF

Nodes defined with 3DOF type has $u_x, u_y, u_z$ degrees of freedom. They correspond to the displacement degrees of freedom in x, y and z direction respectively. The dof id, generalized_displacement and support reactions for 3dof type node are summarized in the Table 1.6.1.

| DOF Id | Description | `Generalized_Displacements` | `Support_Reactions` |
|---|---|---|---|
| 0 | disp. in x-dir | $u_x[m]$ | $F_x[N]$ |
| 1 | disp. in y-dir | $u_y[m]$ | $F_y[N]$ |
| 2 | disp. in z-dir | $u_z[m]$ | $F_z[N]$ |

Table 1.3: DOF id and output for 3DOF type node

### 1.6.2 4DOF

Nodes defined with 3DOF type has $u_x, u_y, u_z, p$ degrees of freedom. They correspond to the displacement degrees of freedom in x, y and z direction and pressure respectively. The dof id, generalized_displacement and support reactions for 4dof type node are summarized in the Table 1.6.2. $up$ elements have nodes with 4 dofs.

| DOF Id | Description | `Generalized_Displacements` | `Support_Reactions` |
|---|---|---|---|
| 0 | disp. in x-dir | $u_x[m]$ | $F_x[N]$ |
| 1 | disp. in y-dir | $u_y[m]$ | $F_y[N]$ |
| 2 | disp. in z-dir | $u_z[m]$ | $F_z[N]$ |
| 3 | pressure | $p[Pa]$ | $p[Pa]$ |

Table 1.4: DOF id and output for 4DOF type node

### 1.6.3 6DOF

Nodes defined with 6DOF type has $u_x, u_y, u_z, r_x, r_y, r_z$ degrees of freedom. They correspond to the displacement and rotational degrees of freedom in x, y and z direction respectively. The dof id, generalized_displacement and support reactions for 6 dof type node are summarized in the Table 1.6.3. Beam

and Shell elements have nodes with 6 dofs.

| DOF Id | Description | Generalized_Displacements | Support_Reactions |
|---|---|---|---|
| 0 | disp. in x-dir | $u_x[m]$ | $F_x[N]$ |
| 1 | disp. in y-dir | $u_y[m]$ | $F_y[N]$ |
| 2 | disp. in z-dir | $u_z[m]$ | $F_z[N]$ |
| 3 | rotation about x-axis | $r_x[radian]$ | $M_x[N-m]$ |
| 4 | rotation about y-axis | $r_y[radian]$ | $M_y[N-m]$ |
| 5 | rotation about z-axis | $r_z[radian]$ | $M_z[N-m]$ |

Table 1.5: DOF id and output for 6DOF type node

### 1.6.4 7DOF

Nodes defined with 6DOF type has $u_x, u_y, u_z, p, U_x, U_y, U_z$ degrees of freedom. They correspond to the solid-displacement , pore-fluid pressure and and fluid-displacement in x, y and z direction respectively. The dof id, generalized_displacement and support reactions for 7dof type node are summarized in the Table 1.6.3. upU elements have nodes with 7 dofs.

| DOF Id | Description | Generalized_Displacements | Support_Reactions |
|---|---|---|---|
| 0 | solid disp. in x-dir | $u_x[m]$ | ${F^s}_x[N]$ |
| 1 | solid disp. in y-dir | $u_y[m]$ | ${F^s}_y[N]$ |
| 2 | solid disp. in z-dir | $u_z[m]$ | ${F^s}_z[N]$ |
| 3 | pore-pressure | $p[Pa]$ | $p[Pa]$ |
| 4 | fluid disp. in x-dir | $U_x[m]$ | ${F^f}_x[N]$ |
| 5 | fluid disp. in y-dir | $U_y[m]$ | ${F^f}_y[N]$ |
| 6 | fluid disp. in z-dir | $U_z[m]$ | ${F^f}_z[N]$ |

Table 1.6: DOF id and output for 7DOF type node

## 1.7 Element-gauss output format

`Gauss_Outputs` array stores the coordinates of the gauss points declared inside for each element defined in the model. It is a 2D `float` array indexed by the `Index_to_Gauss_Outputs` array and the number of rows is defined by 18 (6 total strain, 6 plastic strain and 6 stress) times `Number_of_Gauss_Points`. The column index is represented by the time step of the simulation. Time index $0$ represents initial state conditions, i.e. the state before the start of new stage and end of previous stage.

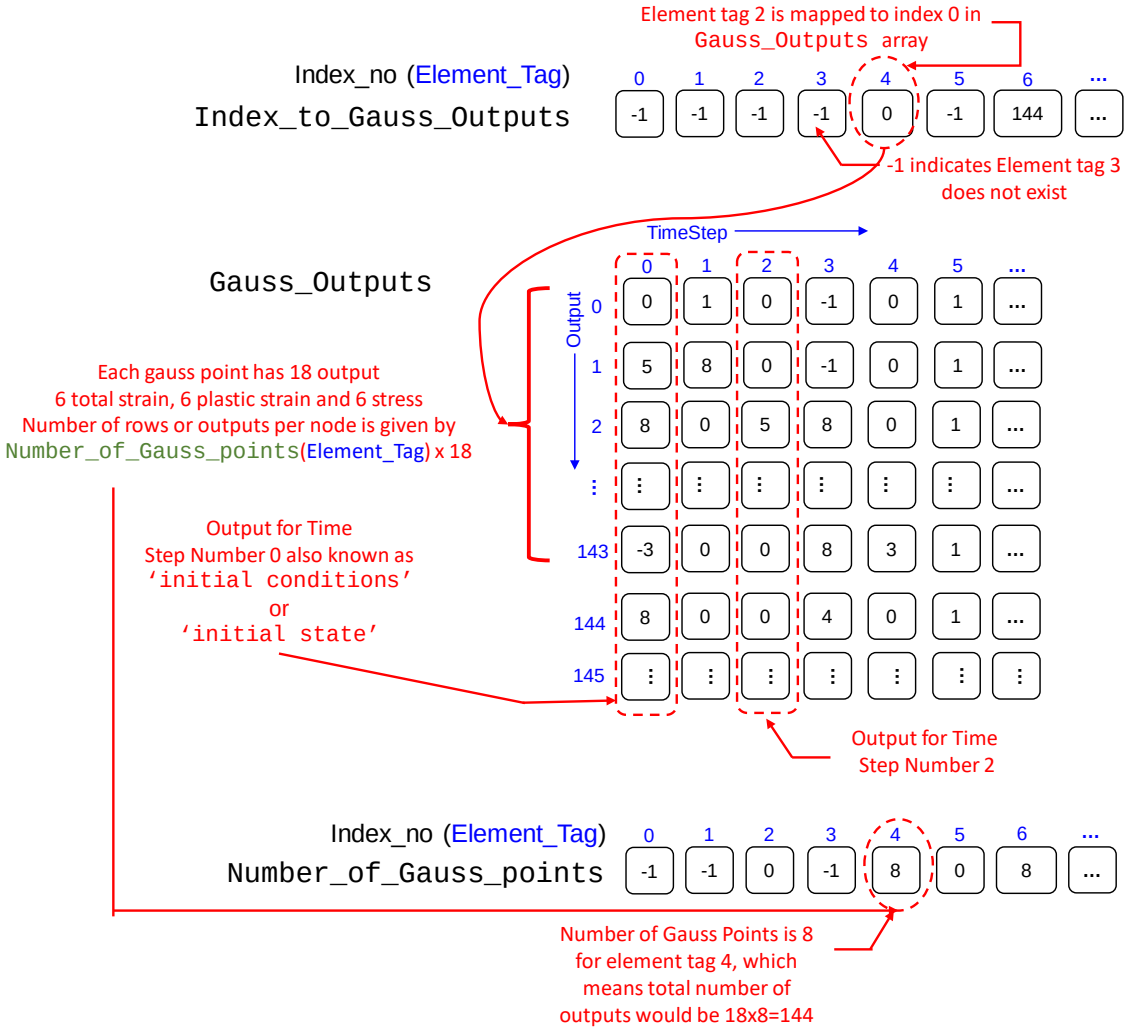


Figure 1.24: Arrays describing gauss output information in `Elements` group directory of HDF5 file.

Figure 1.24 shows how to read `Gauss_Outputs` array for a particular element. In the given example Listing 1.2, element tag 2 and 5 has no gauss points. Since, element tag 4 has 8 gauss points, the total length of gauss output for that element is $8 \times 18 = 144$. The index from which the coordinates information start is 0. gauss values for first $18$ index $(0, 1..17)$ corresponds to gauss point 1 and next 18 index $(18..35)$

corresponds to gauss point 2 and so on. For gauss point 1, first $6$ index $(0, 1..5)$ corresponds to total strain, next $6$ index $(6..11)$ corresponds to plastic strain and next $6$ index $(12..17)$ corresponds to stress. Table 1.7 shows how data is stored for each gauss points. The strains are unit less and stress have unit of Pascal $Pa$.

In the Table 1.7 `start` corresponds to the starting position for the elements output as determined with reference to the `Index_to_Element_Outputs` for the element of interest. To this number we add the corresponding `offset` as determined by each table below and interpret the row according to the meaning established below.

For each gauss outputs there are 6 components of strain tensor, 6 components of plastic-strain tensor and 6 components of stress tensor. This makes a total of $3 \times 6 \times NumGauss$ rows of output per element. The specific meaning of the rows is as follows. For Gauss-point 1 (with starting index given in the `Index_to_Gauss_Outputs` array), the outputs are stored as the following

| Position (start+offset) | Content meaning | Position (start+offset) | Content meaning | Position (start+offset) | Content meaning |
|---|---|---|---|---|---|
| start+0 | $\epsilon_{xx}[]$ | start+6 | $\epsilon^{\text{plastic}}_{xx}[]$ | start+12 | $\sigma_{xx}[Pa]$ |
| start+1 | $\epsilon_{yy}[]$ | start+7 | $\epsilon^{\text{plastic}}_{yy}[]$ | start+13 | $\sigma_{yy}[Pa]$ |
| start+2 | $\epsilon_{zz}[]$ | start+8 | $\epsilon^{\text{plastic}}_{zz}[]$ | start+14 | $\sigma_{zz}[Pa]$ |
| start+3 | $\epsilon_{xy}[]$ | start+9 | $\epsilon^{\text{plastic}}_{xy}[]$ | start+15 | $\sigma_{xy}[Pa]$ |
| start+4 | $\epsilon_{xz}[]$ | start+10 | $\epsilon^{\text{plastic}}_{xz}[]$ | start+16 | $\sigma_{xz}[Pa]$ |
| start+5 | $\epsilon_{yz}[]$ | start+11 | $\epsilon^{\text{plastic}}_{yz}[]$ | start+17 | $\sigma_{yz}[Pa]$ |

Table 1.7: Output Format for each gauss point defined inside element

Gauss-point 2 will then start at position 18 through 36 with the same meaning for each row. And so-on for the other Gauss-points.

## 1.8 Element-specific output format

`Element_Outputs` array stores output for the elements except those stored at gauss points (i.e. stress, total strain and plastic strain). It is a 2D `float` array indexed by the `Index_to_Element_Outputs` array and the number of rows is defined `Number_of_Element_Outputs`. he column index is represented by the time step of the simulation. Time index $0$ represents initial state conditions, i.e. the state before the start of new stage and end of previous stage.

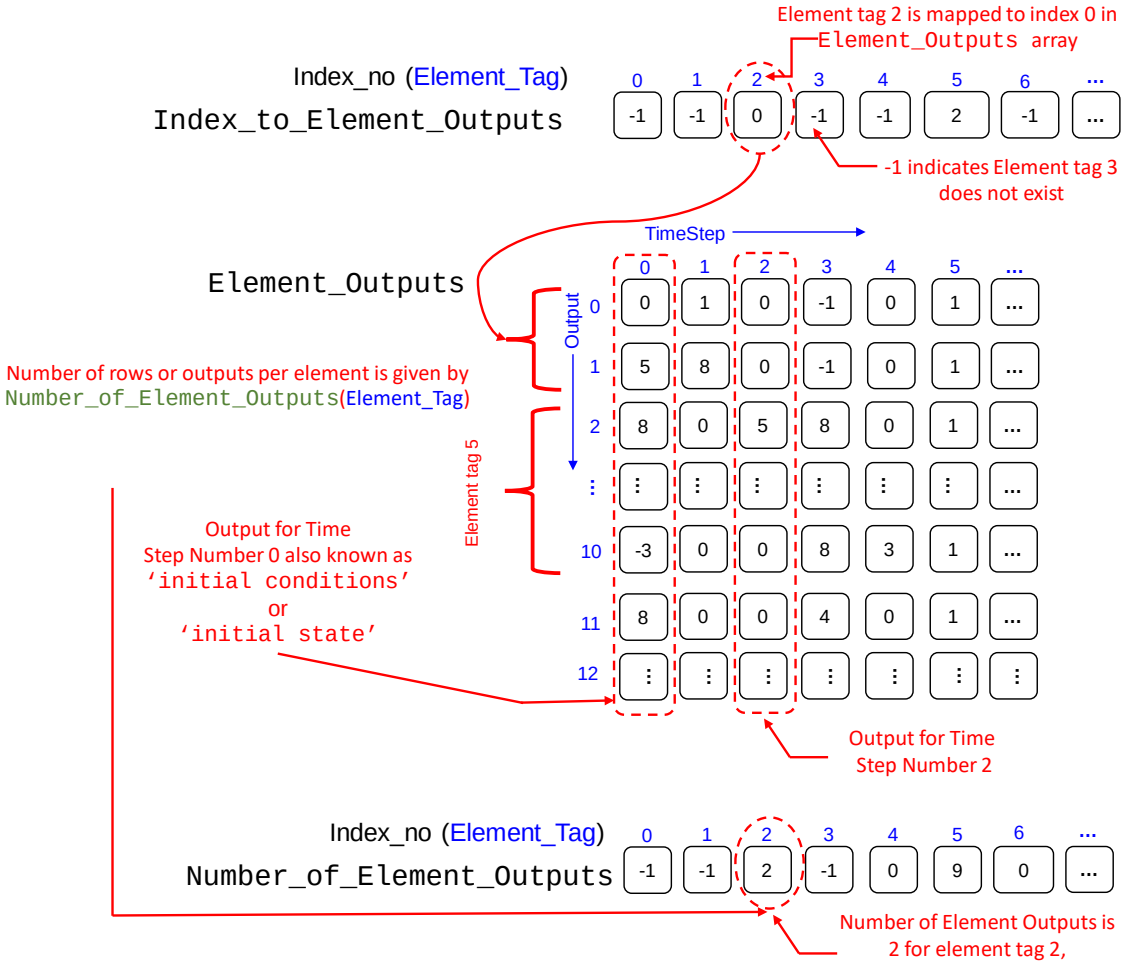


Figure 1.25: Arrays describing output information in `Elements` group directory of HDF5 file.

Figure 1.25 shows how to read `Element_Outputs` array for a particular element and particular time step. In the given example Listing 1.2, element tag 2 has two outputs. Similarly, element tag 5 has 9 outputs. On the other hand element tag $4$ has no output. For element 2, the output starts at row index $0$ and ends at $1$. Similarly, for element tag 4, the output is stored in row index $2..10$.

Since the `Element_Outputs` array format depends on the elements present in the model, one must refer

to each element specifically. Each element writes information into the `Element_Outputs` array in a different way. The user can determine the rows of the `Element_Outputs` array that belong to a given element by looking into the `Index_to_Element_Outputs` array which relates element *tag* to the starting position of that element's output within the `Element_Outputs` array. Additionally, the `Number_of_Element_Outputs` tells the user how many rows after the starting position correspond to the given element output.

The actual meaning of each row is element dependent and is detailed in the following pages. Please note that elements not in this list have no output defined at the moment.

In what follows `start` corresponds to the starting position for the elements output as determined with reference to the `Index_to_Element_Outputs` for the element of interest. To this number we add the corresponding `offset` as determined by each table below and interpret the row according to the meaning established below.

### 1.8.1 Truss

This element outputs 2 rows in total. First row is the uniaxial change in length while second component is the axial force. Truss does not have any gauss points and thus do not have any gauss outputs. The description of each of these rows are shown in Table 1.8:

| Position (start+offset) | Content meaning |
|---|---|
| start+0 | $\Delta L[m]$ |
| start+1 | $ForceF[N]$ |

Table 1.8: Element Output description for Truss

### 1.8.2 Brick Elements

All the bricks (u, up, upU and variable) have $0$ element outputs. But, they do output total strain, plastic strain and stress for all their corresponding number of gauss points involved in elasto-plastic integration. The format in which gauss outputs are stored for each gauss point is described in Section 1.8

### 1.8.3 ShearBeam

Like brick elements, Shear Beam element also does not have any element output. But it does output for the *one* gauss point it has. Thus, in total it has 18 outputs for the only one gauss point. The format in which gauss outputs are stored for each gauss point is described in Section 1.8

### 1.8.4 ElasticBeam

This element outputs 6 components of local nodal displacements at each of its two nodes, and 6 components of end forces at each of its two nodes. Total number of outputs is thus $2 \times 6 \times 2 = 24$ rows per element. The description of each of these rows are shown in Table 1.9:

| Position (start+offset) | Content meaning | Position (start+offset) | Content meaning |
|---|---|---|---|
| start+0 | $u_x^{\text{local},1}[m]$ | start+6 | $u_x^{\text{local},2}[m]$ |
| start+1 | $u_y^{\text{local},1}[m]$ | start+7 | $u_y^{\text{local},2}[m]$ |
| start+2 | $u_z^{\text{local},1}[m]$ | start+8 | $u_z^{\text{local},2}[m]$ |
| start+3 | $\theta_x^{\text{local},1}[radian]$ | start+9 | $\theta_x^{\text{local},2}[radian]$ |
| start+4 | $\theta_y^{\text{local},1}[radian]$ | start+10 | $\theta_y^{\text{local},2}[radian]$ |
| start+5 | $\theta_z^{\text{local},1}[radian]$ | start+11 | $\theta_z^{\text{local},2}[radian]$ |
| Position (start+offset) | Content meaning | Position (start+offset) | Content meaning |
| start+12 | $F_x^{\text{local},1}[N]$ | start+18 | $F_x^{\text{local},2}[N]$ |
| start+13 | $F_y^{\text{local},1}[N]$ | start+19 | $F_y^{\text{local},2}[N]$ |
| start+14 | $F_z^{\text{local},1}[N]$ | start+20 | $F_z^{\text{local},2}[N]$ |
| start+15 | $M_x^{\text{local},1}[N-m]$ | start+21 | $M_x^{\text{local},2}[N-m]$ |
| start+16 | $M_y^{\text{local},1}[N-m]$ | start+22 | $M_y^{\text{local},2}[N-m]$ |
| start+17 | $M_z^{\text{local},1}[N-m]$ | start+23 | $M_z^{\text{local},2}[N-m]$ |

Table 1.9: Element Output description for Elastic Beam

### 1.8.5 4NodeShell_ANDES

This element currently does not have any gauss or element outputs.

### 1.8.6 BeamColumnDispFiber3d

This element outputs 6 components of end forces at each of its two nodes. Each row is described in Table 1.10:

### 1.8.7 Force Based Contact/Interface Elements

This element outputs 9 components: 3 components of gap displacement and 3 components of contact/interface forces and 3 components of incremental slip in the local axis definition. The first two components of gap are transverse components $(g_{t1}, g_{t2})$, while the third is the normal gap component $g_n$. Similarly, the first two

| Position (start+offset) | Content meaning | Position (start+offset) | Content meaning |
|---|---|---|---|
| start+0 | $F_x^{\text{local},1}[N]$ | start+6 | $F_x^{\text{local},2}[N]$ |
| start+1 | $F_y^{\text{local},1}[N]$ | start+7 | $F_y^{\text{local},2}[N]$ |
| start+2 | $F_z^{\text{local},1}[N]$ | start+8 | $F_z^{\text{local},2}[N]$ |
| start+3 | $M_x^{\text{local},1}[N-m]$ | start+9 | $M_x^{\text{local},2}[N-m]$ |
| start+4 | $M_y^{\text{local},1}[N-m]$ | start+10 | $M_y^{\text{local},2}[N-m]$ |
| start+5 | $M_z^{\text{local},1}[N-m]$ | start+11 | $M_z^{\text{local},2}[N-m]$ |

Table 1.10: Element Output description for displacement based fiber beams

components of force $(F_{t1}, F_{t2})$ are transverse (shear on contact/interface plane) while the third is the normal contact/interface force $F_n$. The last three components are incremental slip ${\Delta g^{inc_s lip}}_1, {\Delta g^{inc_s lip}}_2$ in the local transverse direction and total uplift $\Delta n$ in local normal vector direction. If $\Delta n > 0$, there is uplift i.e. loss of contact/interface else it is in contact. Each row is described in Table 1.12:

| Position (start+offset) | Content meaning | Position (start+offset) | Content meaning | Position (start+offset) | Content meaning |
|---|---|---|---|---|---|
| start+0 | $g_{t1}[m]$ | start+3 | $F_{t1}[N]$ | start+6 | ${\Delta g^{inc_s lip}}_1[m]$ |
| start+1 | $g_{t2}[m]$ | start+4 | $F_{t2}[N]$ | start+7 | ${\Delta g^{inc_s lip}}_2[m]$ |
| start+2 | $g_n[m]$ | start+5 | $F_n[N]$ | start+8 | $Uplift \Delta g_n[m]$ |

Table 1.11: Element Output description for force based contact/interface elements

### 1.8.8 Stress Based Contact/Interface Elements

This element outputs 9 components: 3 components of gap displacement and 3 components of contact/interface forces and 3 components of incremental slip in the local axis definition. The first two components of gap are transverse components $(g_{t1}, g_{t2})$, while the third is the normal gap component $g_n$. Similarly, the first two components of stress $(\sigma_{t1}, \sigma_{t2})$ are transverse (shear on contact/interface plane) while the third is the normal contact/interface stress $\sigma_n$. The last three components are incremental slip ${\Delta g^{inc_s lip}}_1, {\Delta g^{inc_s lip}}_2$ in the local transverse direction and total uplift $\Delta n$ in local normal vector direction. If $\Delta n > 0$, there is uplift i.e. loss of contact/interface else it is in contact. Each row is described in Table 1.12:

| Position (start+offset) | Content meaning | Position (start+offset) | Content meaning | Position (start+offset) | Content meaning |
|---|---|---|---|---|---|
| start+0 | $g_{t1}[m]$ | start+3 | $\sigma_{t1}[N/m^2]$ | start+6 | $\Delta g^{inc_s lip}{}_1[m]$ |
| start+1 | $g_{t2}[m]$ | start+4 | $\sigma_{t2}[N/m^2]$ | start+7 | $\Delta g^{inc_s lip}{}_2[m]$ |
| start+2 | $g_n[m]$ | start+5 | $\sigma_n[N/m^2]$ | start+8 | $Uplift\Delta g_n[m]$ |

Table 1.12: Element Output description for stress based contact/interface elements

## 1.9 Energy Output Format

Energy folder in the output file stores simulation results for energy-related quantities. Energy density quantities are calculated at element integration points. Averaged energy density quantities are calculated for each element. Energy quantities are calculated for each element. Nodal input energy values are calculated at each node. All energy terms are calculated and stored at each time step.

Under the Energy folder, there are currently seven datasets. Four of them contain energy calculation results. The other three are index datasets, which can be used to find energy output for specific elements or nodes.

In general, energy output datasets are 2D float arrays. The column index is represented by the time step of the simulation. The row index represents various energy quantities that are calculated and stored. The three index datasets are used to locate the row index for specific energy quantities.

### 1.9.1 Input Energy

All types of nodal load, including DRM, are applied at nodes. Elemental loads are automatically converted to nodal loads, then also applied at nodes. Therefore, input energy or input work from all types of external load can be calculated at each node.

Under the Energy folder, the Nodal_Input_Energy dataset stores input energy at each node. Note that this is input energy accumulated from the start of simulation to the current time step. To find the input energy time history at a specific node, find the index in the Index_to_Nodal_Input_Energy dataset then find the corresponding entry in the Nodal_Input_Energy dataset.

For example, here are the steps to get the input energy time history at the node with node tag 73, Go to Index_to_Nodal_Input_Energy, find the index for node 73 to be 105. Then the accumulated input energy at your chosen node is stored at row 105 in Nodal_Input_Energy.

### 1.9.2 Energy Density Quantity at Gauss Point

Energy_Density_GP dataset stores energy density quantities at each Gauss point. To find the energy density time history at a specific Gauss point, find the index in the Index_to_Energy_Density_GP dataset then find the corresponding entry in the Energy_Density_GP dataset.

For each Gauss point, 12 data slots are occupied in the following order:

- Incremental kinetic energy density
- Accumulated kinetic energy density
- Incremental strain energy density
- Accumulated strain energy density
- Incremental plastic free energy density
- Accumulated plastic free energy density
- Incremental plastic dissipation density
- Accumulated plastic dissipation density
- The last 4 slots are currently empty, reserved for potential future use

Note that "incremental" means the change of energy density during the time step, "accumulated" means current cumulative energy density at the time step.

Here is an example showing how to obtain the energy density evolution at a specific Gauss point. Say you are interested in looking at the accumulated plastic dissipation density at the third Gauss point of an 8NodeBrick element with element tag 73. Go to Index_to_Energy_Density_GP, find the index for element 73 to be 3904. Since each Gauss point occupies 12 slots in Energy_Density_GP, the energy outputs for the third Gauss point starts at index 3904+(3-1)*12=3928. Finally, since accumulated plastic dissipation density is the 8th entry among the 12 slots, the row index for the data of your interest is 3928+(8-1) = 3935. This means that the accumulated plastic dissipation density at your chosen location is stored at row 3935 of Energy_Density_GP.

### 1.9.3 Average Energy Density Quantity for Element

Energy_Density_Element_Average dataset stores averaged energy density quantities of each element. To find the energy density time history of a specific element, find the index in the Index_to_Energy_Element dataset then find the corresponding entry in the Energy_Density_Element_Average dataset.

For each element, 12 data slots are occupied in the following order:

- Incremental kinetic energy density
- Accumulated kinetic energy density
- Incremental strain energy density
- Accumulated strain energy density
- Incremental plastic free energy density
- Accumulated plastic free energy density
- Incremental plastic dissipation density
- Accumulated plastic dissipation density
- The last 4 slots are currently empty, reserved for potential future use

Note that "incremental" means the change of energy density during the time step, "accumulated" means current cumulative energy density at the time step.

Here is an example showing how to obtain the energy density evolution of a specific element. Say you are interested in looking at the accumulated plastic dissipation density of an `8NodeBrick` element with element tag 73. Go to `Index_to_Energy_Element`, find the index for element 73 to be 612. Since accumulated plastic dissipation density is the 8th entry among the 12 slots, the row index for the data of your interest is 612+(8-1) = 619. This means that the accumulated plastic dissipation density of your chosen element is stored at row 619 of `Energy_Density_Element_Average`.

### 1.9.4 Energy Quantity for Element

`Energy_Element` dataset stores averaged energy density quantities of each element. To find the energy time history of a specific element, find the index in the `Index_to_Energy_Element` dataset then find the corresponding entry in the `Energy_Element` dataset.

For each element, 12 data slots are occupied in the following order:

- Incremental kinetic energy
- Accumulated kinetic energy
- Incremental strain energy
- Accumulated strain energy
- Incremental plastic free energy

- Accumulated plastic free energy
- Incremental plastic dissipation
- Accumulated plastic dissipation
- Incremental viscous energy dissipation
- Accumulated viscous energy dissipation
- The last 2 slots are currently empty, reserved for potential future use

Note that "incremental" means the change of energy density during the time step, "accumulated" means current cumulative energy density at the time step.

Here is an example showing how to obtain the energy evolution of a specific element. Say you are interested in looking at the accumulated plastic dissipation of an `8NodeBrick` element with element tag 73. Go to `Index_to_Energy_Element`, find the index for element 73 to be 612. Since accumulated plastic dissipation is the 8th entry among the 12 slots, the row index for the data of your interest is 612+(8-1) = 619. This means that the accumulated plastic dissipation of your chosen element is stored at row 619 of `Energy_Element`.

# Bibliography

T. H. Group. HDF5. https://www.hdfgroup.org/HDF5/, 2020.