ETABS .e2k Model Storage Scheme

STORY-centric data structure

The data structure in the `.e2k` file is highly efficient and object-oriented from an ETABS perspective, but it is fundamentally different from the explicit, nodal-based structure required by programs like OpenSeesPy. (See the attached Kosmos_Plat.e2k file)

Figure 1 illustrates the ETABS `.e2k` file's story-centric data structure. It's a highly efficient scheme that defines the building as a "stack of stories," where each story acts as a template to which structural elements are associated. It might be understood like a STORY based data structure where different building entities belong to a floor. Figure 1 shows STORY 08_P4 and its associated elements corresponding to:

- Column “C43”
- Beam “B10”
- Point “22” (In the top end of column “C43”)
- Point “23”
- Point “45”.

All these elements and points belong to STORY “08_P4”. Thus a STORY is defined by all the entities contained in it (beams, nodes, etc) and/or all the entities “hanging” from it. In Figure 1 Column “C43” and point “45” hang from STORY 08_P4.

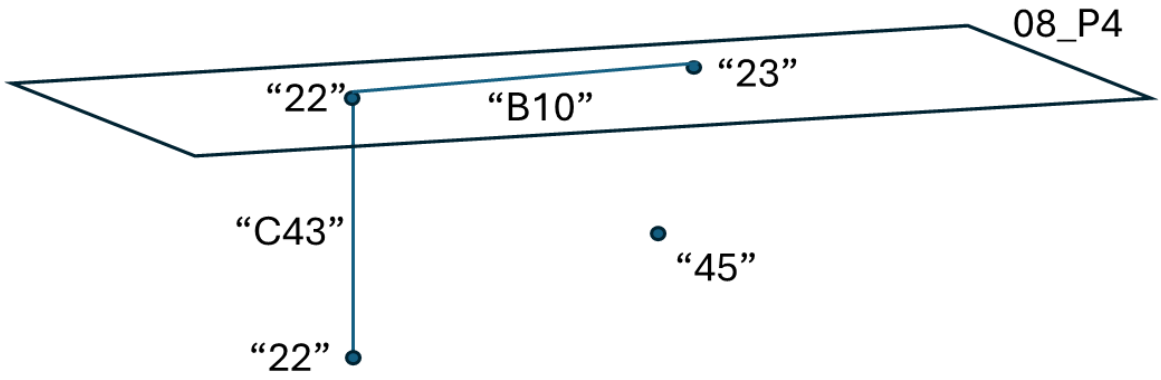

Figure 1. STORY based storage scheme

The generation of an OpenSeesPy model from an ETABS model (stored in the .e2k) file corresponds to the generation of OpenSeesPy entries for each and everyone of the entities stored in the STORIES defining the building. If every STORY is defined in this way the building is then formed by a stack of STORIES with all of its entities, see Figure 2.

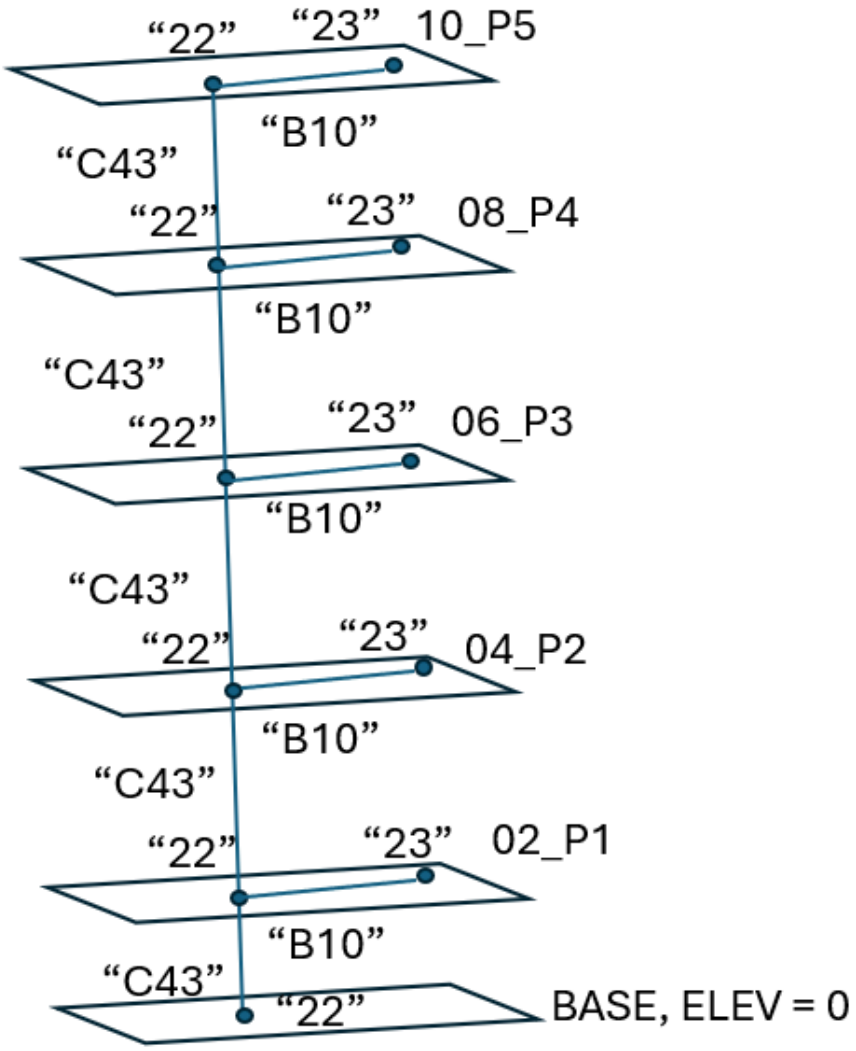

Figure 2. Stack of STORIES representing a building model

In what follows we will define the relevant parts of the .e2k file needed for creation of the basic structure, that is a framed building conformed initially by STORIES, Nodal Points, Beams and Columns. Once this basic model is built into OpenSees we progressively add complexities to the model.

The basic components of the .e2k file

The $ STORIES section

This section of the .e2k file defines the basic database unit in the .e2k storage scheme. It is identified by the keyword `$ STORIES` followed by the definition of each one of the STORIES forming the building stack. This stack is defined in sequence from the top and for coding purposes we can adopt an indexing system where (with reference to the figure) STORY "11_P6" corresponds to index = 0, STORY "08_P5" to index = 1 and so on and so forth until running all the stack. The stack of floors also stores the definition of the global vertical (Z) reference system of the model. The datum (Z = 0.0) is located at the STORY with the parameter ELEV 0 in the bottom of the stack, in this case corresponding to the STORY labeled "Base". The Z coordinate of the remaining STORIES is cumulative defined by the HEIGHT parameter in each STORY. Accordingly, STORY "01_P2_m170" with HEIGHT 2.0 is at Z = 2.0; STORY "02_P2" with HEIGHT 1.7 is at Z = 2.0+1.7; STORY "03_P3_m170" with HEIGHT 1.7 is at Z = 2.0+1.7+1.7 and this pattern follows all the way to the last level of the stack.

```
$ STORIES - IN SEQUENCE FROM TOP

  STORY "11_P6"  HEIGHT 1.475

  STORY "08_P5"  HEIGHT 1.55 SIMILARTO "04_P3"

  STORY "07_P5_m155"  HEIGHT 1.55 SIMILARTO "03_P3_m170"

  STORY "04_P4"  HEIGHT 1.55 SIMILARTO "04_P3"

  STORY "05_P4_m155"  HEIGHT 1.55 SIMILARTO "03_P3_m170"

  STORY "04_P3"  HEIGHT 1.7 MASTERSTORY "Yes"

  STORY "03_P3_m170"  HEIGHT 1.7 MASTERSTORY "Yes"

  STORY "02_P2"  HEIGHT 1.7

  STORY "01_P2_m170"  HEIGHT 2

  STORY "Base"  ELEV 0
```

The $ POINT COORDINATES section

This section of the .e2k file defines the plan or global X, Y coordinates of all the points existing in the building model. This is done in terms of a 2D "master plan" containing all potential structural points, but only with their **X and Y coordinates**. We could think of this like a plane storing the normal projection of all the points in the model, see Figure 3.

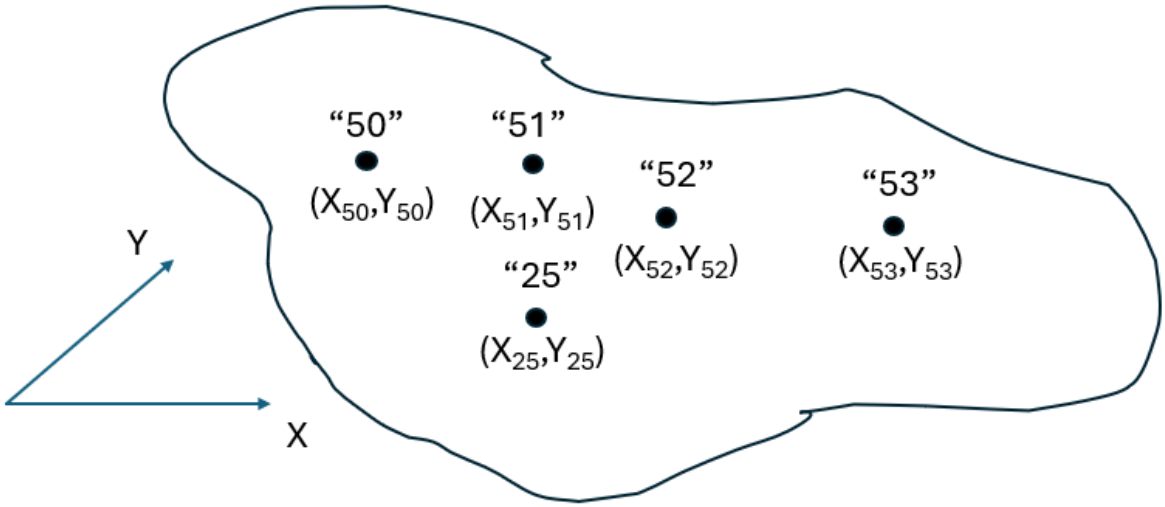

Figure 3. General floor storing the projection of all the nodal points in the building over a horizontal plan.

The start of the section is identified by the keyword `$ POINT COORDINATES` followed by the X,Y coordinates of the projection of every point on the plane

```
$ POINT COORDINATES

  POINT "50"  46.9 27.45

  POINT "51"  47.95 27.45

  POINT "52"  49 27.45

  POINT "53"  50.05 27.45

  POINT "25"  51.1 27.45  2.5
```

The $ POINT ASSIGNS section

The full definition of the nodal points in the building corresponding to their vertical distribution using the projections from the previous section is defined in the $ POINT ASSIGNS section. To illustrate how this works consider a hypothetical stack of 3 floors as shown in Figure 4. Notice that in the stack of floors column "C1" spans two different floors and each column segment is shown with initial and end points described by the same point tag, in this case nodal point "50". Similarly, column "C2" spanning one STORY is formed by identical end points in this case "51". This multispan representation of the column will be explained later.

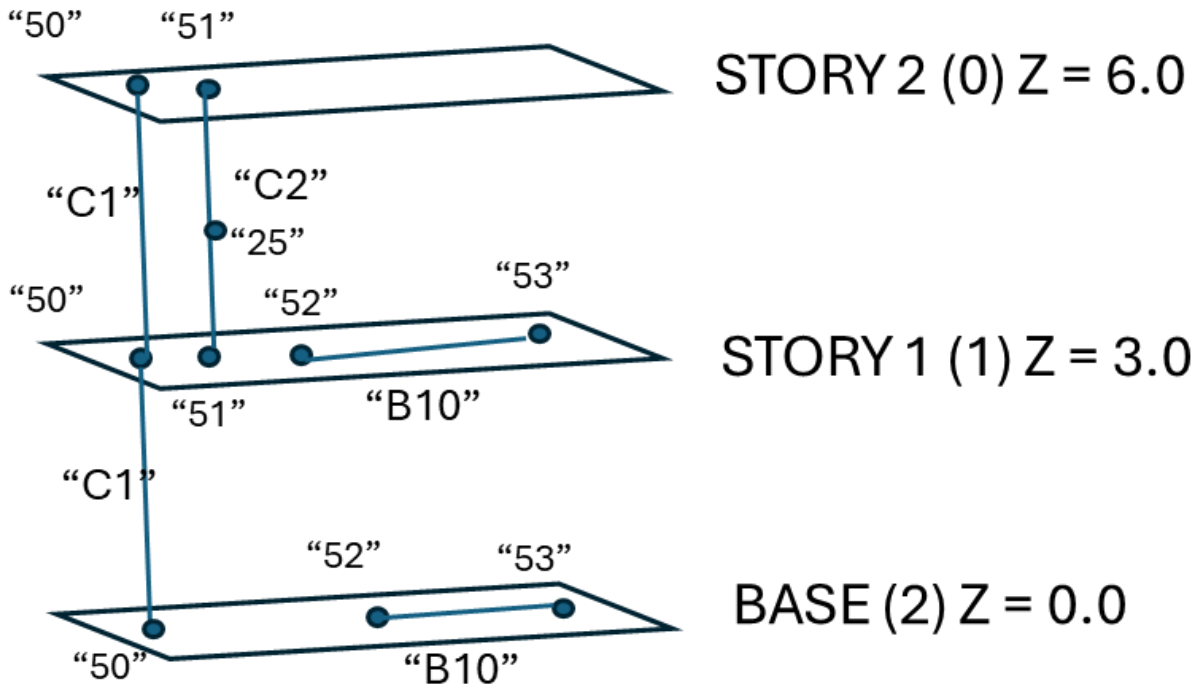

Figure 4. Stack of hypothetical floors.

Now, the X,Y locations of these nodes appear in the general 2D master plan shown in Figure 5. Off courseTo precisely locate the points in space we need both, their X,Y location and their Z location.

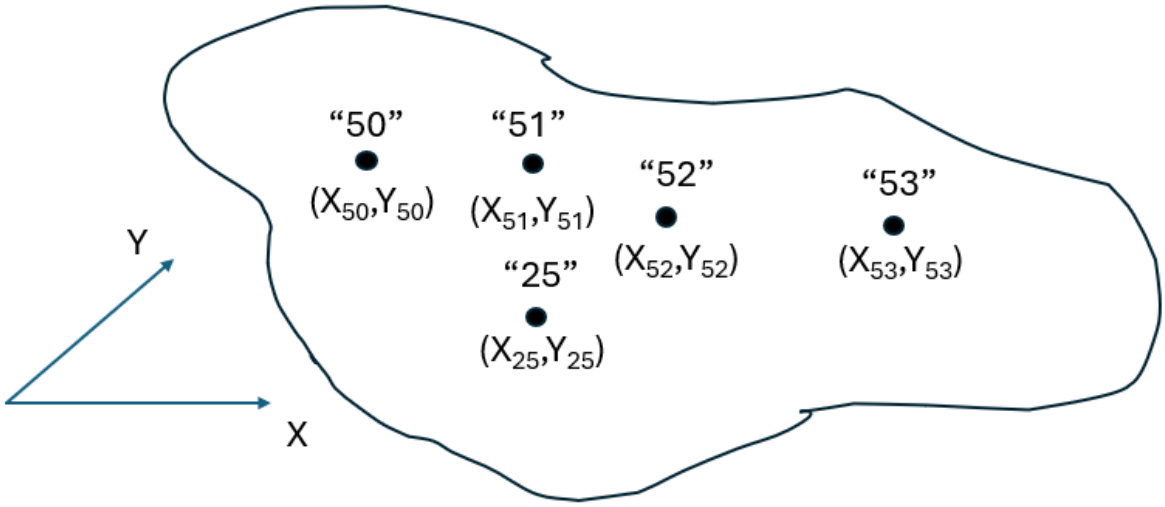

Figure 5. General 2D master plan corresponding to the physical nodes shown in Figure 4.

To define the Z coordinate the .e2k file uses the `$ POINT ASSIGNS` section and the `$ POINT COORDINATES` section as described next. Although in the general master plan a Point appears only once, each appearance of the point in the vertical elevation of the building is described by an entry in the `$ POINT ASSIGNS` section of the file indicating the location of the POINT in the corresponding STORY. For instance while in the `$ POINT COORDINATES` section, and in the general 2D master plan, point "50" appears only once, in the `$ POINT ASSIGNS` section (and in the stack of Figure 4) it appears 3 times each one corresponding to its association to STORIES **BASE, STORY 1** and **STORY 2** respectively.

```
$ POINT ASSIGNS

POINTASSIGN "50" "BASE"

POINTASSIGN "50" "STORY 1"

POINTASSIGN "50" "STORY 2"

POINTASSIGN "51" "STORY 1"

POINTASSIGN "51" "STORY 2"
```

```
POINTASSIGN "52" "BASE"

POINTASSIGN "52" "STORY 1"

POINTASSIGN "53" "BASE"

POINTASSIGN "53" "STORY 1“

POINTASSIGN “25" "STORY 2"
```

Accordingly, the X, Y coordinates of the nodes appearing in the `$ POINT ASSIGNS` section is read from the general 2D master plan (`$ POINT COORDINATES` section) while the Z coordinate is the one from the associated STORY. For instance there is a point “50” is located at `X = 46.9 Y =27.45` $Z = Z_{BASE}$; a second point “50” located at at `X = 46.9 Y =27.45` $Z = Z_{STORY1}$ and a third point “50” located at at `X = 46.9 Y =27.45` $Z = Z_{STORY2}$. Clearly these are 3 different physical points which eventually would have three different point tags when translated to OpenSeesPy.

Notice that all the points in the $ POINT COORDINATE section, except point “25”, have only their X,Y coordinates defined and we already know how to obtain their Z location. However point “25”, (shown in the elevation view of Figure 4) is declared in that section like:

```
POINT "25"  51.1 27.45  2.5
```

The third numerical entry corresponding in this case to 2.5 defines the distance from the node to its associated STORY. According to the $ POINT ASSIGNS section definition for this node which is:

```
POINTASSIGN “25" "STORY 2"
```

This node belongs to STORY 2 but it is located at a distance of 2.5 below (as it hangs from the STORY). Therefore the Z coordinate of point “25” is equal to:

$Z_{25} = Z_{STORY2}-2.5$

This rule is to be applied **ALWAYS** whenever there is a node with explicit Z value defined in the `$ POINT COORDINATES` section of the file.

The $ LINE CONNECITIVITIES section

The `$ LINE CONNECTIVITIES` section defines the abstract topology or the "wiring diagram" of the frame. It connects the points from the 2D master plan to create logical elements. Again, in this section the connectivities are defined in terms of abstract points and using the multispan column representation of columns while the actual element creation is completed in the $ LINE ASSIGNS section. To clarify, consider once again the general 2D master plan shown in Figure 6 where we now observe a first abstract beam element “B10” connecting nodes “52” and “53”, a second abstract beam element “B5” connecting nodes “50” and “51”, and an abstract column element “C43” connecting nodes “50” and “50”. These are abstract elements since they are not fully defined until they are not assigned to specific STORIES.

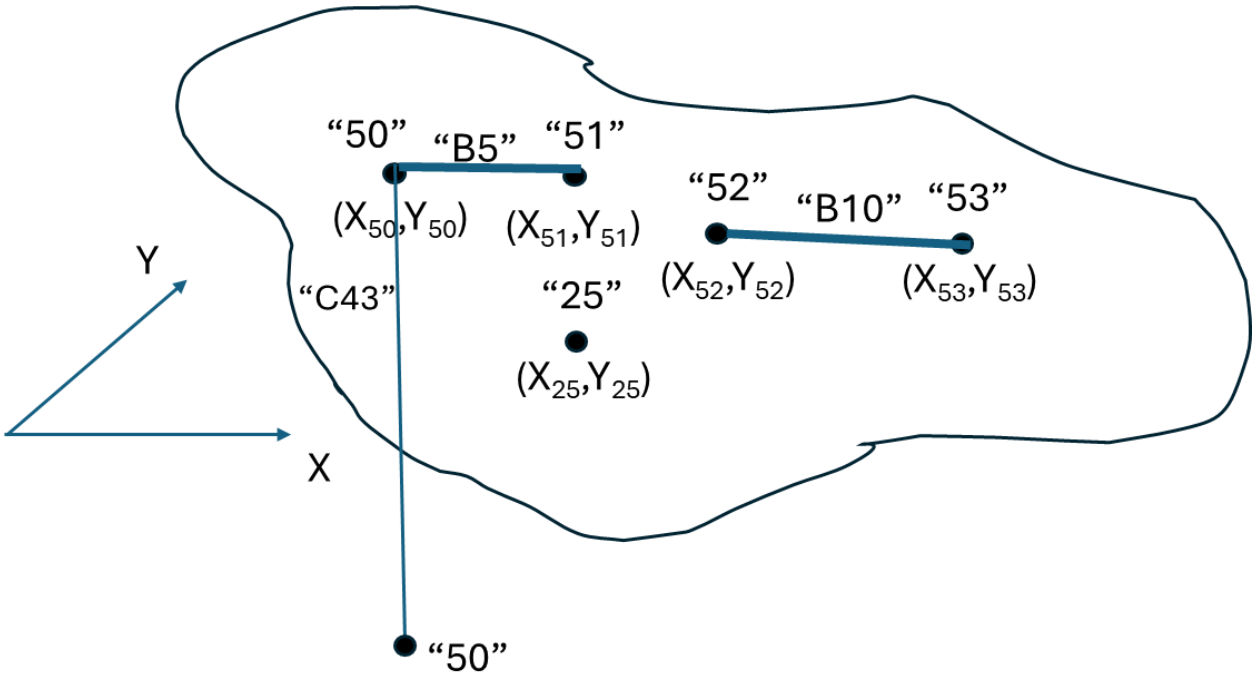

Figure 6. General 2D Master plan with column and beam elements connectivities

The representation of the elements in the general 2D master plan is then described in terms of the `$ LINE CONNECTIVITIES` of the .e2k file. For instance the definition of

the abstract elements corresponding to the plan in Figure 6 would be as follows:

```
$ LINE CONNECTIVITIES
LINE "C43"  COLUMN "50" "50"
LINE "B10"  BEAM   "52" "53"
LINE "B5"   BEAM   "50" "51"
```

The $ LINE ASSIGNS section

The `$ LINE ASSIGNS` section of the .e2k file allows to precisely define the location and particular features of the beam and column elements in the building. For instance beam "B10" as defined in the `$ LINE CONNECTIVITIES` section in Figure 6 might exist in several stories but under different particular conditions. For instance it might have ***end releases*** in one story but ***rigid connections*** in a different one. Its location and particular conditions are indicated in the `$ LINE ASSIGNS` section in exactly the same way as the `$ POINT ASSIGNS` specifies particular conditions for the abstract points defined in the `$ POINT COORDINATES` section.

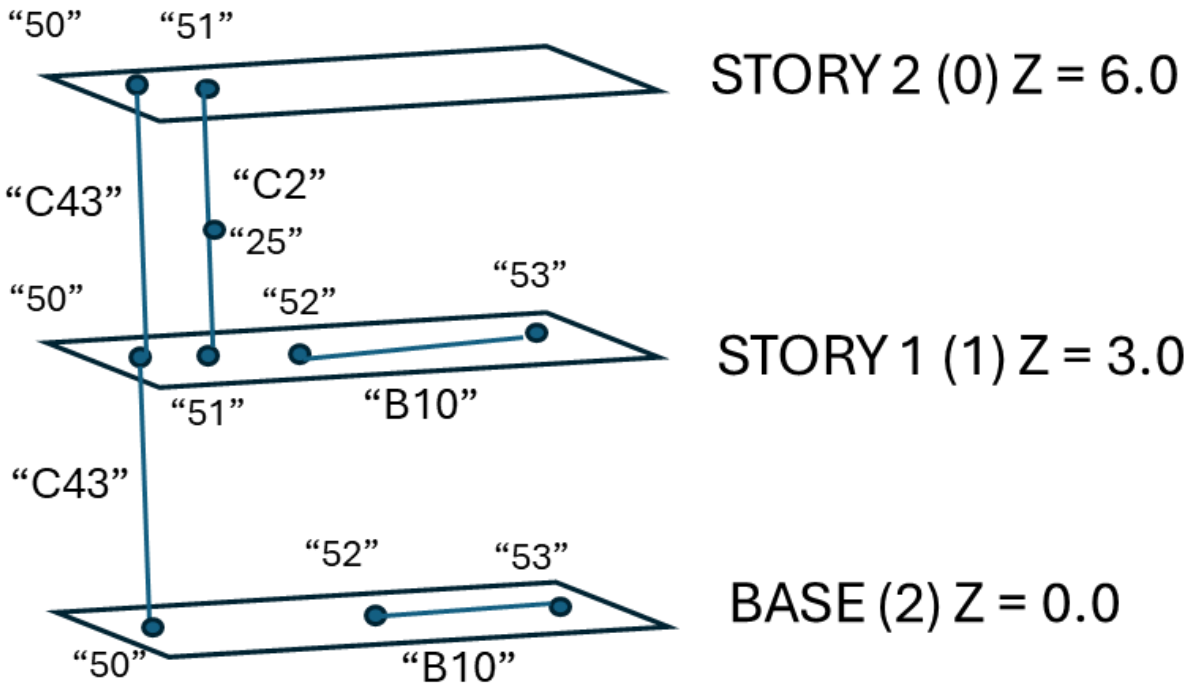

Figure 7. Stack of floors with column and beams

Consider the stack of floors shown in Figure 7. This stack, having now a precise location and distribution of columns and beams will be defined by its corresponding `$ LINE ASSIGNS` section as follows:

```
$ LINE ASSIGNS
LINEASSIGN "C43"          "STORY 1"
LINEASSIGN "C43"          "STORY 2"
LINEASSIGN "C2"           "STORY 2"
LINEASSIGN "B10"           "BASE"
LINEASSIGN "B10"          "STORY 1"
```

Note that in this section there are going to be as many entries as there are elements in the actual building. For instance the stack of floors shows two beams "B10" located in the stories "BASE" and "STORY 1" respectively as declared in the two entries from the file. Similarly, there are 3 column segments in the stack of floors, two of them of the type "C43" spanning STORY 1 and STORY 2 and one column type "C2" associated with STORY 2.

Consider the special case shown in Figure 8 where a column type "C43", already defined in the `$ LINE CONNECTIVITIES` section by

```
LINE "C43"  COLUMN "50" "50"
```

goes from a point "50" located in floor BASE to a second point "50" located in STORY 2. This column clearly hangs from STORY 2 therefore it belongs to STORY 2 and it runs all the way down to the floor BASE but without having any point in the intermediate STORY 1. This column would still be defined by:

```
LINEASSIGN "C43"          "STORY 2"
```

in the `$ LINE ASSIGNS` section and the program would know that it directly connects to the BASE floor as there is no POINT "50" in STORY 1. In other words the code would know that the column goes from point "50" located in the STORY from which the column hangs to the first next point "50" in a lower STORY. This would be the general rule to locate column segments. In the case of column "C43" in Figure 8 there is only one column segment going from STORY 2 directly into the BASE floor and therefore there is no support for the column in STORY 1.

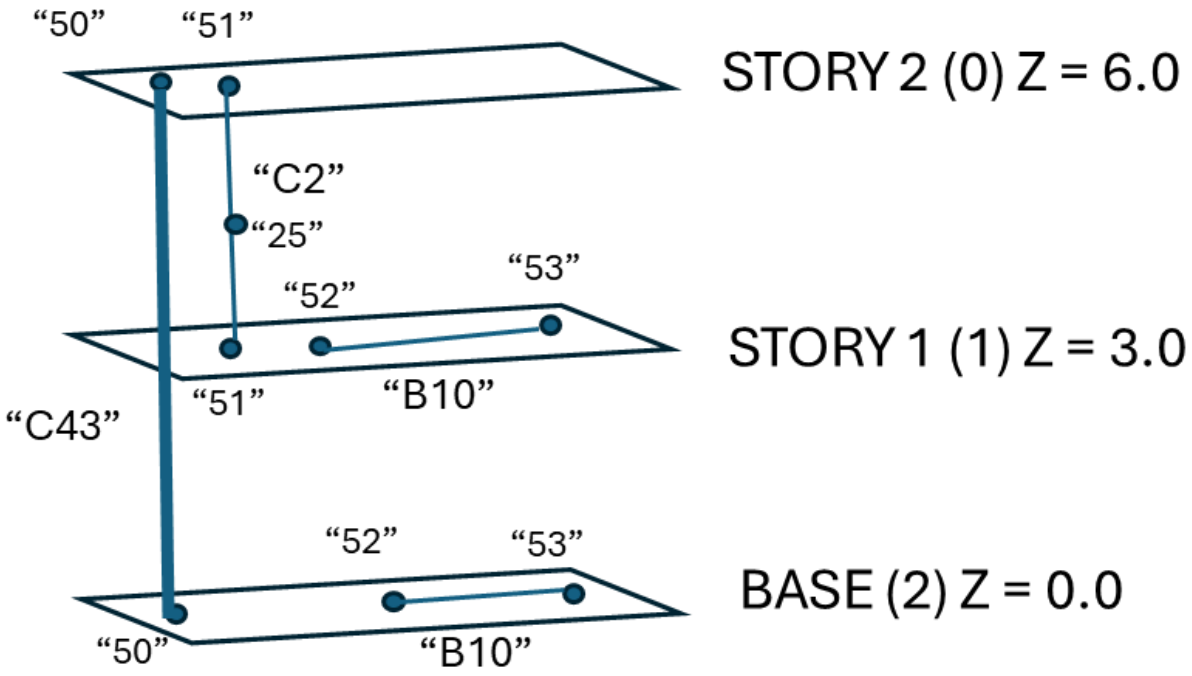

Figure 8. Special case of a column going through an intermediate story

SUMMARY OF THE RELEVANT DATA BLOCKS IN THE .e2k FILE

```
$ STORIES
STORY "STORY 2" HEIGHT 3.0
STORY "STORY 1" HEIGHT 3.0
STORY "BASE"   ELEV  0.0
```

Defines the vertical stack of floors conforming the building. The datum Z = 0 is at the STORY with the parameter ELEV 0.0

```
$ POINT COORDINATES
POINT "50"  0.0  0.0
POINT "51"  5.0  0.0
POINT "52"  2.0  0.0
POINT "53"  6.0  0.0
POINT “25"  5.0  0.0  0.5
```

Defines all the abstract points in the General 2D master plan

```
$ POINT ASSIGNS
POINTASSIGN "50" "BASE"
POINTASSIGN "50" "STORY 1"
POINTASSIGN “51" "STORY 2"
POINTASSIGN "51" "STORY 1"
POINTASSIGN "51" "STORY 2"
POINTASSIGN "52" "BASE"
POINTASSIGN "52" "STORY 1"
POINTASSIGN "53" "BASE"
POINTASSIGN "53" "STORY 1“
POINTASSIGN “25" "STORY 2"
```

Defines the actual physical points in the model or equivalently the distribution of abstract points along the stack of floors

```
$ LINE CONNECTIVITIES
LINE "C1"      COLUMN "50" "50"
LINE "C2"      COLUMN "51" "51"
LINE "B10_BASE"  BEAM  "52" "53"
LINE "B10_S1"   BEAM  "52" "53"
```

Defines the abstract elements in the general 2D master plan

```
$ LINE ASSIGNS
LINEASSIGN "C1"      "STORY 1"
LINEASSIGN "C1"      "STORY 2"
LINEASSIGN "C2"      "STORY 2"
LINEASSIGN "B10_BASE"  "BASE"
LINEASSIGN "B10_S1"   "STORY 1"
```

Defines the actual physical distribution of abstract elements in the stack of floors

Suggested Assembly Instructions

Here we propose an initial set of assembly instructions for the OpenSees model out of the .e2k file defining the ETABS model.

1. Using the General 2D Master Plan (`$ POINT COORDINATES`), the vertical stack of floors (`$ STORIES`) and the vertical distribution of the points in the stack of floors (`$ POINT ASSIGNS`) generate all the physical points in the building. Each point can now be represented in OpenSeesPy format like: ***node(node_tag, x, y, z)***.
2. Create a mapping between the ETABS points defined in the `$ POINT ASSIGNS` section and the OpenSeesPy nodal points. For instance there should be a correspondence between node "50" in STORY 1 and an OpenSeesPy node, and between node "50" in STORY 2 and a different OpenSeesPy node.
3. Generate the Beams: For each STORY and for each beam within the STORY, and using the nodal mapping created in 2 define the corresponding OpenSeesPy beams using for instance:

```
b_sec = 0.40

h_sec = 0.50

A_beam = b_sec * h_sec

Iy_beam = (b_sec * h_sec**3) / 12

Iz_beam = (h_sec * b_sec**3) / 12

J_beam = b_sec * h_sec**3 / 3

E_beam = 2.50e10

G_beam = 1.04e10

transfTag = 222


# Define the beam elements


element('elasticBeamColumn', 1064, 507, 485, A_beam, E_beam, G_beam, J_beam, Iy_beam, Iz_beam, transfTag)
```

4. Generate the columns: For each STORY and for each column within the STORY,

and using the nodal mapping created in 2 define the corresponding OpenSeesPycolumns using for instance:

```
 geomTransf('Linear', 111, 1, 0, 0)

# --- Placeholder Material and Section Properties ---

# NOTE: These are temporary values for visualization.

b_col = 0.40

h_col = 0.40

E_col = 2.5e10

nu_col = 0.2

G_col = E_col / (2 * (1 + nu_col))

A_col  = b_col * h_col

Iy_col = (b_col * h_col**3) / 12

Iz_col = (h_col * b_col**3) / 12

J_col  = b_col * h_col**3 / 3


# Define the column elements


element('elasticBeamColumn', 1, 1, 2, A_col, E_col, G_col, J_col, Iy_col, Iz_col, 111)
```