ETABS .e2k Model Storage Scheme

This document provides a comprehensive description of each major section found in an ETABS-generated .e2k text file, based on the KOSMOS_Plat.e2k example and explanations useful for its translation into an OpenSeesPy equivalent model.

Components of the .e2k file

1. File Header and Program Information

- **$ File ...**: A comment line indicating the original file path and the timestamp when the .e2k file was exported from ETABS.
- **$ PROGRAM INFORMATION**: Specifies the software (ETABS) and version (20.2.0) that generated the file. This is crucial for compatibility.

2. Model Controls and Basic Definitions

- **$ CONTROLS**: Defines global settings for the model.
  - UNITS: Sets the base units for force, length, and temperature (e.g., "KGF", "M", "C").
  - MERGETOL: The tolerance distance below which separate nodes are merged into one.
  - RLLF: Parameters for Reduced Live Load Factors, a code-based provision.
- **$ STORIES - IN SEQUENCE FROM TOP**: Defines the vertical layout of the building, including story height, elevation, and master/similar story relationships.
- **$ GRIDS**: Defines the grid systems (e.g., "G1") used for modeling, including the location and labels for each grid line.
- **$ DIAPHRAGM NAMES**: Defines floor diaphragms (e.g., "D1"), which model the in-plane stiffness of floor slabs to distribute lateral loads. They can be RIGID or SEMI-RIGID.

3. Material and Section Properties

- **$ MATERIAL PROPERTIES**: Defines all materials for the model (e.g., "Concrete", "Steel"). Key properties include E (Modulus of Elasticity), U (Poisson's Ratio), WEIGHTPERVOLUME, and strength parameters like FC (for concrete) or FY (for steel).
- **$ FRAME SECTIONS**: Defines the cross-sections for all line elements (beams, columns, braces). It specifies the section's name, shape, dimensions, and assigned material.
- **$ CONCRETE SECTIONS**: Provides concrete-specific design information for the

frame sections, including reinforcement details, cover, and design type (Column or Beam).

- **$ SLAB PROPERTIES / $ WALL PROPERTIES**: Defines the properties for area sections used for slabs and walls. This includes the material, thickness, and modeling type (e.g., Membrane, ShellThin).
- **$ LINK PROPERTIES**: Defines properties for specialized link elements used to model dampers, isolators, gaps, or other specific nonlinear behaviors.
- **$ POINT SPRING PROPERTIES**: Defines spring supports used to model soil-structure interaction or other elastic supports by specifying stiffness values for each degree of freedom.

4. Geometry and Connectivity

- **$ POINT COORDINATES**: A list of all the points (nodes) in the model, defined by a unique name (typically a number) and their X, Y, and Z coordinates.
- **$ LINE CONNECTIVITIES**: Defines all line elements (beams, columns) by connecting the points defined in the coordinates table.
- **$ AREA CONNECTIVITIES**: Defines all area elements (slabs, walls) by listing the points that form their corners, typically in a counter-clockwise order.

5. Assignments (Connecting Properties to Geometry)

- **$ POINT ASSIGNS**: Assigns properties or constraints to points. This includes RESTRAINT conditions (e.g., fixed, pinned), DIAPH (diaphragm) assignments, and SPRINGPROP (spring) assignments.
- **$ LINE ASSIGNS**: Assigns properties to line elements. This includes SECTION assignments, moment RELEASE conditions (to model pins), and rigid end LENGTHOFF (offsets).
- **$ AREA ASSIGNS**: Assigns properties to area elements. This includes SHELLPROP (slab/wall property) assignments, PIER or SPANDREL labels for design, and meshing instructions.

6. Loading, Analysis, and Design

- **$ LOAD PATTERNS**: Defines the names and types of loads (e.g., "DEAD", "LIVE", "QUAKE"). It also specifies the SELFWEIGHTMULT (self-weight multiplier).
- **$ LOAD CASES**: Defines the analysis cases, such as LINEAR STATIC or MODAL analysis, and specifies which load patterns are included.
- **$ COMBINATIONS**: Defines load combinations for design, which combine the results of different load cases using specified scale factors (e.g., 1.4 * DEAD + 1.6 * LIVE).

- **$ JOINT LOADS - FORCE**: Applies point loads directly to joints/nodes for a given load case.
- **$ FRAME LOADS - DISTRIBUTED**: Applies distributed loads along frame elements.
- **$ SHELL OBJECT LOADS - UNIFORM**: Applies uniform surface loads to shell (area) elements.
- **$ DESIGN PREFERENCES / OVERWRITES**: Sections that control the parameters

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 ***EjemploNew.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” both hang from STORY 08_P4.

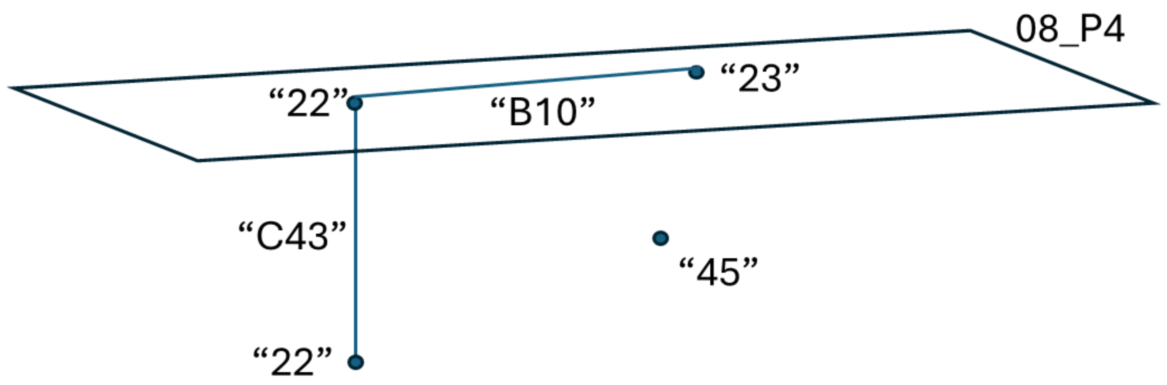

Figure 1. STORY based storage scheme

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 stored or associated to the given STORY, see Figure 2.

On the other hand in an OpenSeesPy model all the entities, that is nodes, elements, loads, materials, etc, must be explicitly defined in a traditional storage scheme as shown next in the following snippet of a typical OpenSeesPy file:

```
wipe()
    model("basic", "-ndm", ndm, "-ndf", ndf)

    # --- Nodes ---
    node(1, 60.990652, 20.9609649, 14.775)
    node(2, 59.4406246, 22.2606796, 13.3)

geomTransf('Linear', 1442376417, 1, 0, 0)
    element('elasticBeamColumn', 342376417, 20002, 1860458314,
160000, 2.5e+10, 1.04166667e+10, 8533.33333, 2133.33333,
2133.33333, 1442376417)
    geomTransf('Linear', 1205534426, 1, 0, 0)
    # [nl] COLUMN tag 105534426 ← hinge_set 'ColHinge_C50x80C'
    element('forceBeamColumn', 105534426, 1860458314,
1851532825, 1205534426, 43)
```

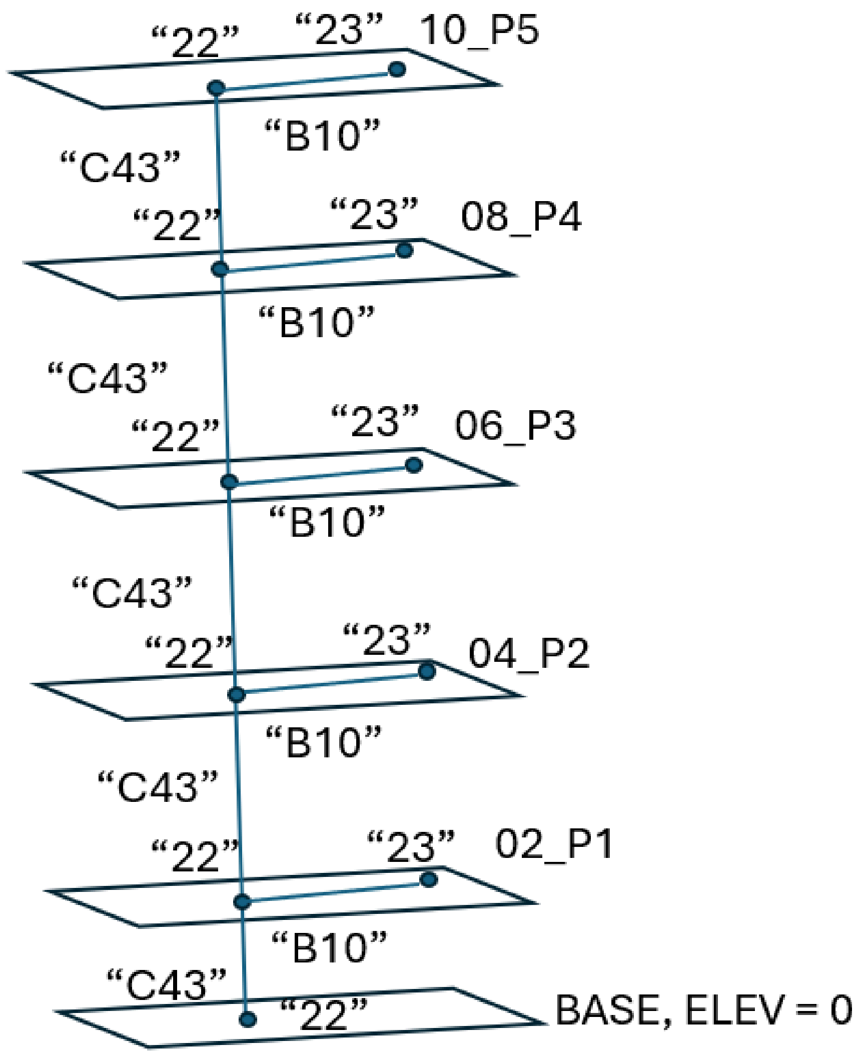

Figure 2. Stack of STORIES representing a building model

In what follows we will define the relevant parts of the .e2k file needed for creation of a basic structure, that is a framed building conformed initially by STORIES, Nodal Points, Beams and Columns. Once this basic model is built into OpenSeesPy we will 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 and relevant parameters of each one of the STORIES forming the building stack.

The stack is defined in sequence from the top and for coding purposes we can adopt an indexing system where (with reference to the sample file) STORY "11_P6" corresponds to index = 0, STORY "08_P5" to index = 1 and so on 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 cumulatively 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 the 2D "master plan" containing all potential structural points, but only with their **X and Y coordinates**.

We could think of the master plan like a plane storing the perpendicular projections of all the points existing in the model, see Figure 3 regardless of its z location. In this sense there might be for instance many points "50" in the model all of them sharing the same (X,Y) coordinates but each one with a different Z location.

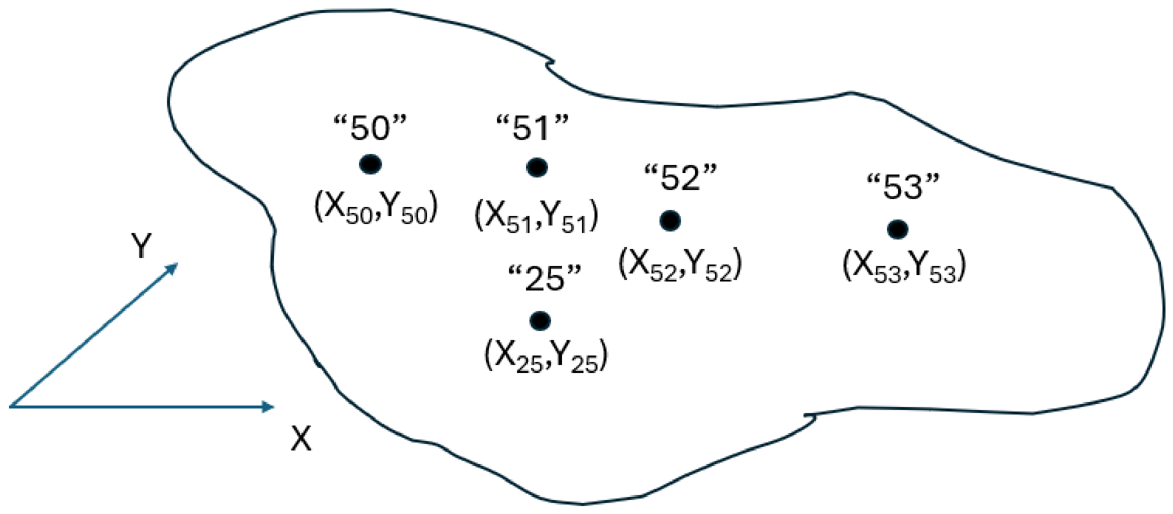

Figure 3. General floor storing the projection of all the nodal points in the building over a horizontal plan.

The start of the section plan point coordinates section is identified by the statement `$ POINT COORDINATES` followed by the X,Y coordinates of the projection of every point on the plane as follows:

```
$ 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 "point assigns" section describes now the complete spatial location of the nodal points in the building, corresponding to their vertical distribution in addition to the projections from the previous section. To illustrate how this works consider a hypothetical stack of 3 floors as shown in Figure 4. Notice that in this sample 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". Notice once again that there are in fact three nodes labeled "50" sharing the same (X,Y) coordinates but placed at different Z locations. These points will be mapped into different nodal tags in OpenSeesPy. 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.

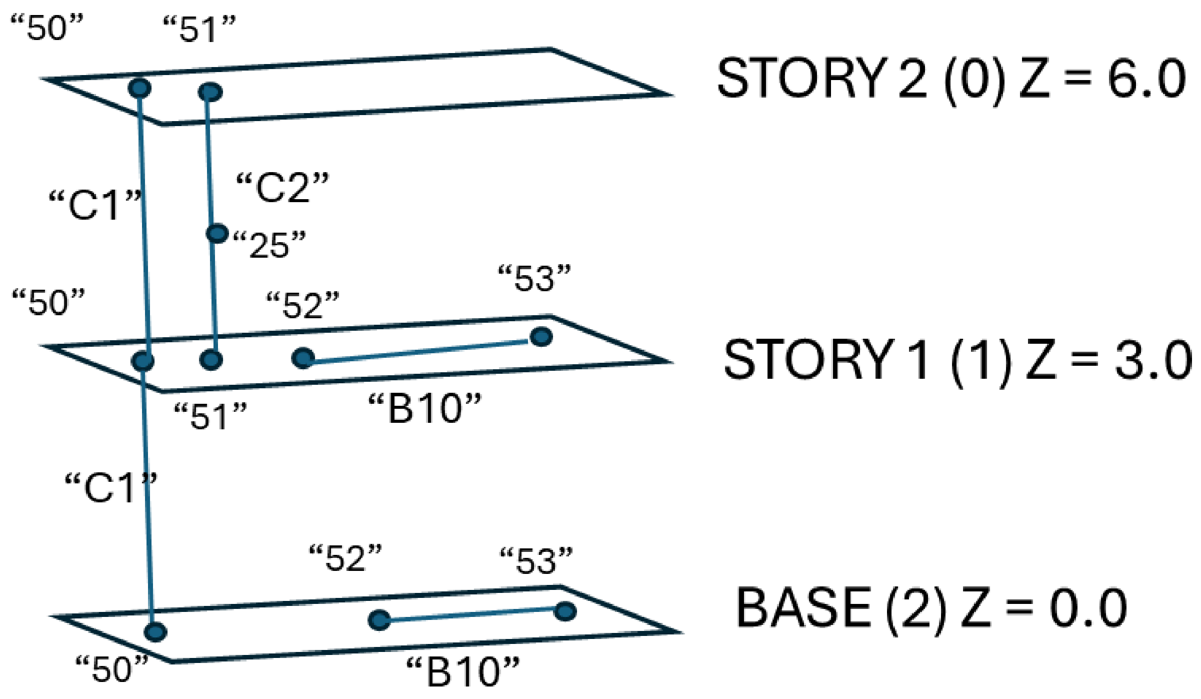

Figure 4. Stack of hypothetical floors.

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

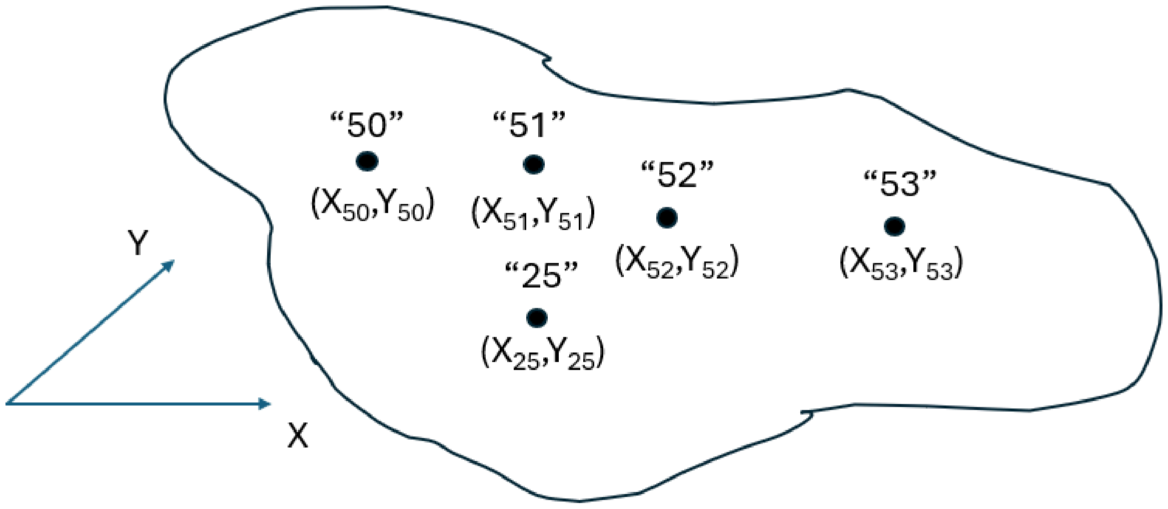

Figure 5. General 2D master plan corresponding to the physical nodes shown in Figure 4.

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 sample 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” 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.

Points with an explicit z value

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 the Z location of the story (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 CONNECTIVITIES 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.

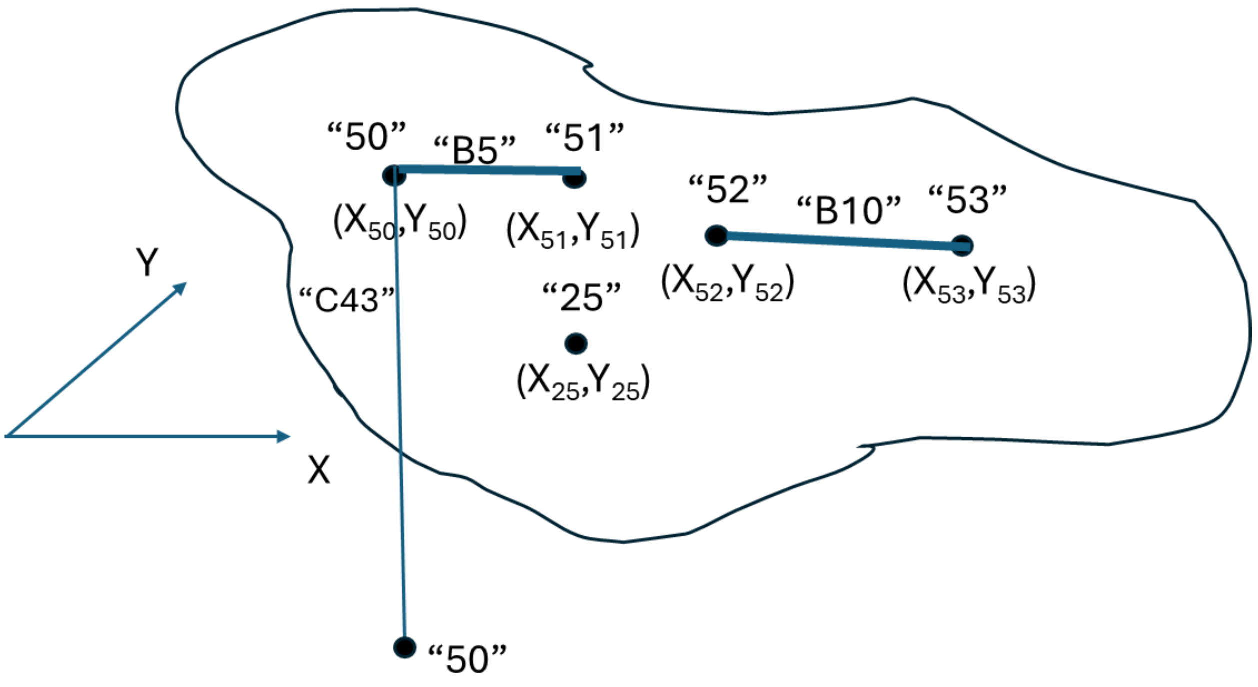

Figure 6. General 2D Master plan with column and beam elements connectivities

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 view 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 define accuratly 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.

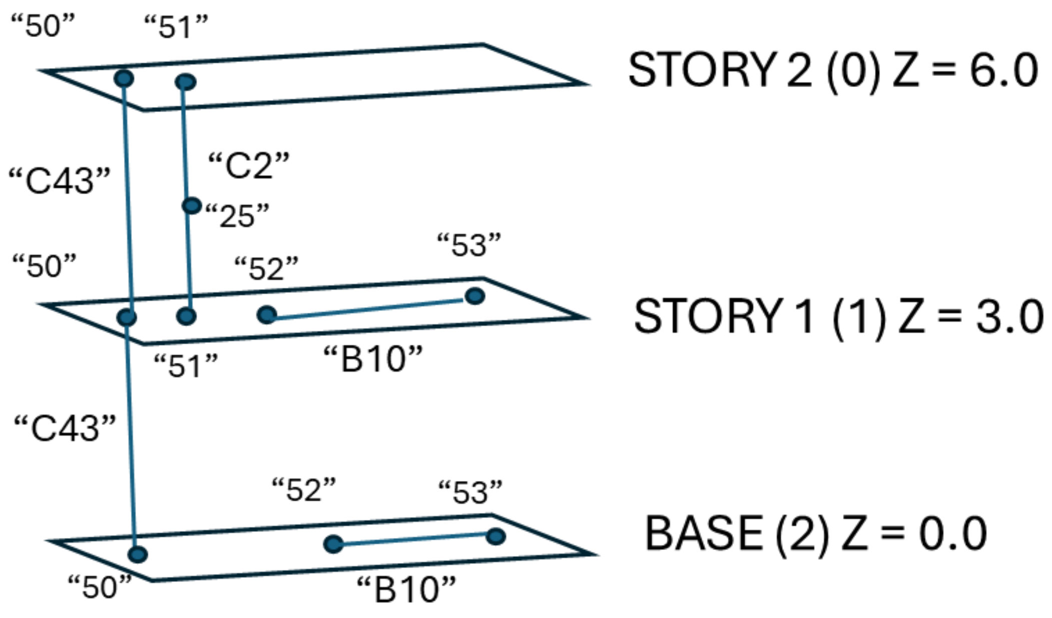

Figure 7. Stack of floors with column and beams

Consider the sample stack of floors shown in Figure 7. This stack, having now an accurate 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 sample 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 the 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.

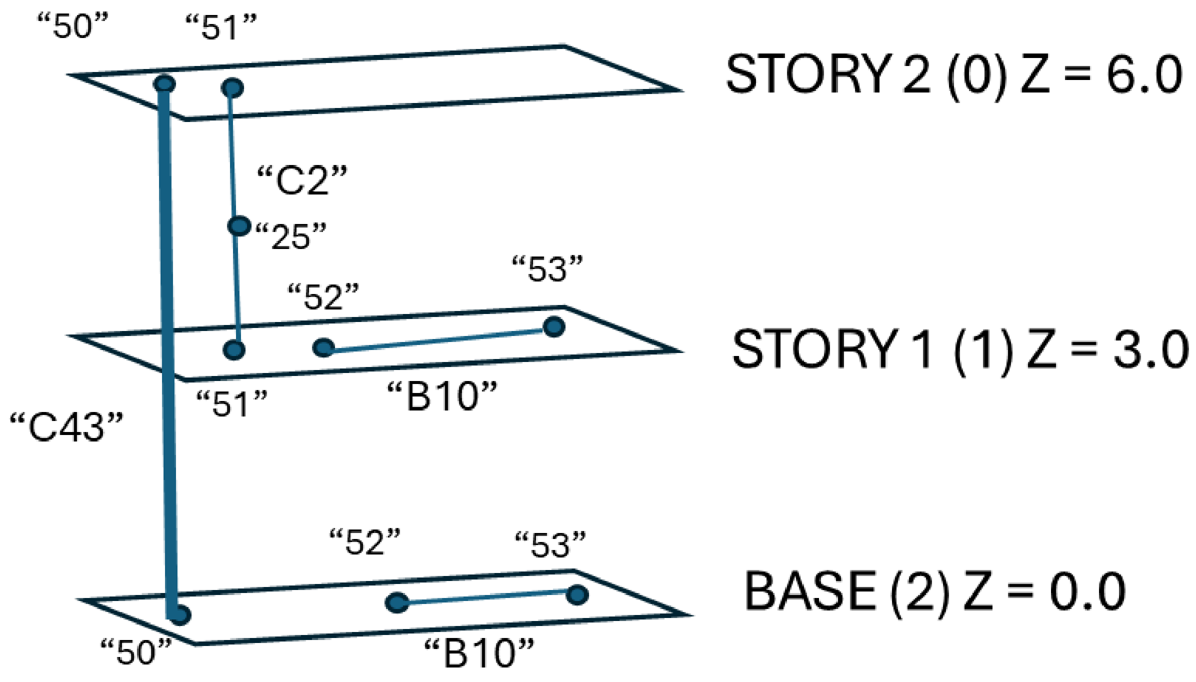

Figure 8. Special case of a column going through an intermediate story

Length offsets and end offsets

In ETABS frame section properties are assigned to frame objects. However, actual structural members have finite cross-sectional dimensions. When two members such as a beam and column are connected at a joint, there is some overlap of the cross-sections. In many structures, the dimensions of the members are large, and the length of the overlap can be a significant fraction of the total length of the frame section. ETABS provides the capability of defining ***end length offsets*** along the length of frame sections to account for these finite dimensions of structural components. ***Length offsets*** are defined in the LINEASSIGNs of the .e2k file as follows:

```
LINEASSIGN  "B522"  "02_P2"  SECTION "C50x80C"  LENGTHOFFI 0.275
LENGTHOFFJ 0.275 RIGIDZONE 1
```

Joint offsets

On the other hand when a frame object is used to model a frame section, the frame object is assumed to be located at the centroid of the frame section. Thus, when frame objects (frame sections) intersect in a model, it means that the centroids of the associated frame objects intersect. In a real structure, that is not always the case. For example, it is not unusual for one or more floor beams in a building to frame eccentrically into a column. ETABS provides the capability of defining frame ***joint offsets*** to account for these eccentric connections. ***Joint Offsets*** are defined in the LINEASSIGNs of the .e2k file as follows:

```
LINEASSIGN  "C437"  "02_P2"    OFFSETXI -0.05  OFFSETYI 0.2
OFFSETXJ -0.05  OFFSETYJ 0.2
```

The offsets parameters have the following meanings:

OFFSETXI: End offset at nodal point I along the global X direction.

OFFSETYI: End offset at nodal point I along the global Y direction.

OFFSETXJ: End offset at nodal point J along the global X direction.

OFFSETYJ: End offset at nodal point J along the global Y direction.

When none of these parameters appear within the LINEASSIGN of an element it means there are no offsets.

In OpenSeesPy the end offsets are specified as:

```
'geomTransf('Linear',  transfTag,  *vecxz,  '-jntOffset',  *dI,
*dJ)'
```

where *dI and *dJ are the components of vectors in the global coordinate system defining the offset values for element-ends node i and node j respectively.

To understand how these offsets and their translation are going to be handled within this development consider the plan view of a beam-column-beam joint shown in figure 9. The column, labeled as “C102” in the Etabs .e2k file has a squared cross section. There is an X-parallel beam, labeled “B31” excentrically connecting to the column at the white analysis node and also a Y-parallel beam labeled “B32” connecting to this same white analysis node. Note that both beams connect to the analysis node through their centroidal axes while the centroidal axis of the column (black node) has X and Y offsets with respect to the white analysis node. This is our most common practice as all our models are built through the centroidal lines of beams.

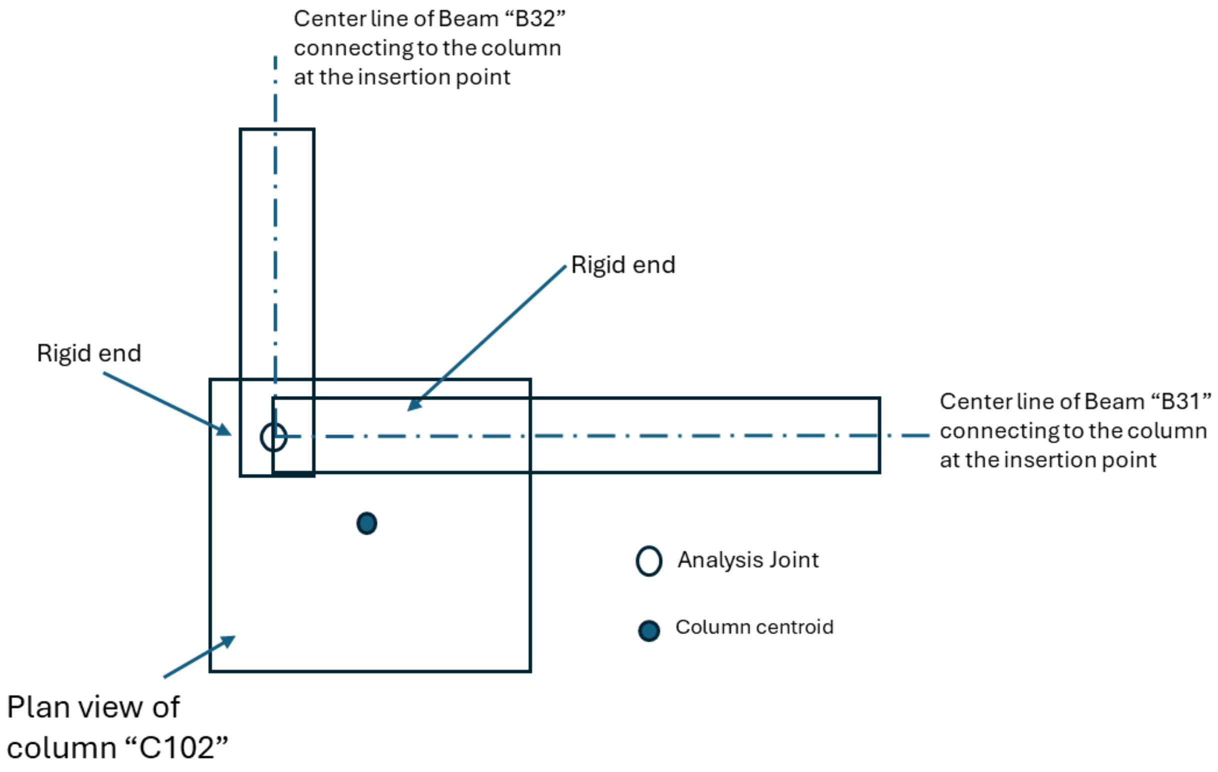
Center line of Beam "B32" connecting to the column at the insertion point
Rigid end
Rigid end
Center line of Beam "B31" connecting to the column at the insertion point
Analysis Joint
Column centroid
Plan view of column "C102"

Figure 9. Beams to column connection with rigid ends and joint offsets

Modeling convention used here

In our modeling approach we adopt the following convention:

• Node location: at the intersection of beam centerlines (see white node in figure 9).

• Beams: modeled centroidal to that node.

• Columns: at that joint, the column end is eccentric relative to the node; therefore joint offsets are assigned to the column at that story level.

• Rigid ends (end-length offsets): may exist on both beams and columns.

In ETABS the above modeling convention implies the following fields in the .e2k file

1) Rigid ends (end-length offsets)

Add rigid (non-deformable) zones by shortening the deformable span with:

```
LENGTHOFFI <L_I>  LENGTHOFFJ <L_J>  RIGIDZONE 1
```

in the element's LINEASSIGN row for the story where the member exists. These belong to the member that has the rigid zone (beam or column).

2) Joint offsets (eccentric connections)

Shift an element end off the analysis node with:

```
OFFSETXI <dxI>  OFFSETYI <dyI>  OFFSETZI <dzI>

OFFSETXJ <dxJ>  OFFSETYJ <dyJ>  OFFSETZJ <dzJ>
```

Use these on the column at the joint (since beams are centroidal in this convention). Signs follow the global axes.

OpenSeesPy translation (geometric transformation)

OpenSees represents both effects (rigid ends and offsets) through joint offsets specification on the element's transformation as follows:

```
geomTransf('Linear' or 'PDelta', transfTag, *vecxz_if_3D,

               '-jntOffset', dIx, dIy, dIz,  dJx, dJy, dJz)
```

• dI, dJ are global vectors from the node to the effective element end.

• Build them as the sum of:

  – an axial piece from end-length offsets, and

  – a lateral piece from joint offsets (eccentricity).

Let the unit vector along the member axis be ê (from I to J).

End-length offsets → axial part

```
d_I^(len) = + L_I * eˆ
```

```
d_J^(len) = - L_J * eˆ
```

Joint offsets → lateral part

Take the .e2k OFFSET* components (global) as Δ_I, Δ_J.

Total per end

```
d_I = d_I^(len) + Δ_I
```

```
d_J = d_J^(len) + Δ_J
```

In this convention:

• Beams typically have Δ = 0 (centroidal), but may have L_J > 0 rigid ends.

• The column has Δ ≠ 0 at the story joint where the beams meet, and may also have L_I/L_J if you model rigid zones in the column.

Minimal templates

.e2k (at the story of the joint)

With respect to figure 9 we have the following fields.

Column (eccentric end at J; optional rigid ends):

```
LINEASSIGN "C102" "<StoryZ>"  SECTION "<ColSec>" \

  [LENGTHOFFI <L_I_col>] [LENGTHOFFJ <L_J_col>] [RIGIDZONE 1] \

  OFFSETXJ <dX>  OFFSETYJ <dY>  [OFFSETZJ <dZ>]
```

X-beam (centroidal; rigid end at J only):

```
LINEASSIGN "B31" "<StoryZ>"  SECTION "<BeamSecX>" \

  LENGTHOFFJ <L_Jx>  RIGIDZONE 1
```

Y-beam (centroidal; rigid end at J only):

```
LINEASSIGN "B32" "<StoryZ>"  SECTION "<BeamSecY>" \
  LENGTHOFFJ <L_Jy>  RIGIDZONE 1
```

(Include LENGTHOFFI if there is a rigid zone at the I end too.)

OpenSeesPy (3D examples; use global Z as the vecxz helper)

Beams (centroidal; only rigid ends at J):

```
# B31 along +X

geomTransf('Linear', 31, 0,0,1, '-jntOffset',

               0,0,0,   -L_Jx, 0,0)

# B32 along +Y

ops.geomTransf('Linear', 32, 0,0,1, '-jntOffset',

               0,0,0,    0, -L_Jy, 0)
```

Column (vertical; eccentric at J; optional rigid ends)

```
# Column local x // global Z

dIx_col,   dIy_col,   dIz_col   =   (+L_I_col)*0,   (+L_I_col)*0,
(+L_I_col)*1  # if using column rigid I-end

dJx_col,   dJy_col,   dJz_col   =   (-L_J_col)*0,   (-L_J_col)*0,
(-L_J_col)*1  # if using column rigid J-end

# add lateral eccentricity at J:

dJx_col += dX;  dJy_col += dY;  dJz_col += dZ

 geomTransf('Linear', 102, 1,0,0, '-jntOffset',

               dIx_col, dIy_col, dIz_col,

               dJx_col, dJy_col, dJz_col)
```

Sanity checks & pitfalls

• Don’t double-count eccentricity. If the node is at the beam–beam intersection, keep beams centroidal ($\Delta=0$) and put the offset on the column end at that story.

• Signs are global. OFFSETXJ > 0 means the element’s J end is shifted toward +X from the node.

• Rigid ends ≠ joint offsets. The former are along the member axis (shorten deformable span); the latter are lateral vectors from node to the effective end.

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