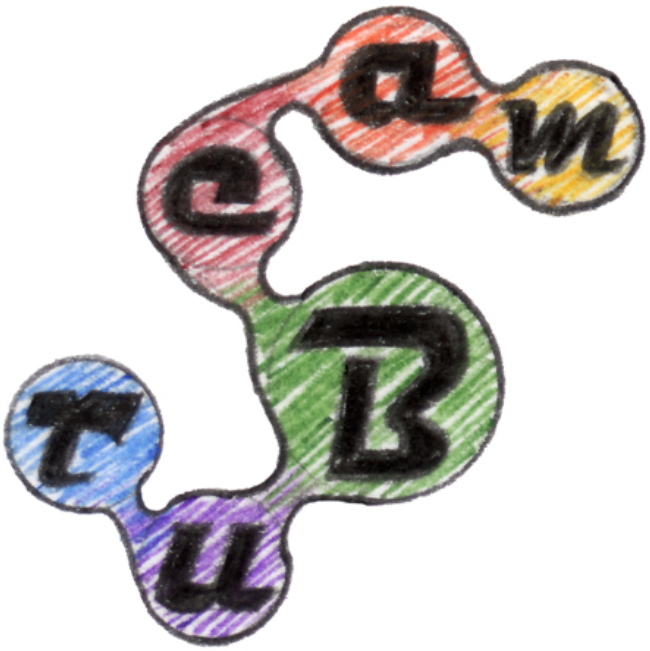


ruBeam Documentation v1.0 $beta$

Contact address: emanuel.bombasaro@cademia.org, christian.koch@cademia.org
Homepage: http://www.rubeam.cademia.org/

Contents

Chapter 1

ruBeam Model

To start with the basics in this chapter the ruBeam model is introduced. The ruBeam model is the core of both ruBeam engine and ruBeam plugin which builds up the structural model, handles all the components and solves the structure as well as preparing the results.

1.1 Introduction

ruBeam model is a simple open source 2D structural analyzer created in Java. The scope of ruBeam model is to have a simple and platform independent structural generator and solver which can be used in bash mode, ruBeam engine, or as plugin, ruBeam plugin, on the CAD program CADEMIA[1]. Furthermore the differential equation for an element is solved generally and so no discretization error is made.

1.1.1 What can be done with ruBeam?

- Generating a 2D Analysation Structure
- Uses Exact Solution for the Differential Equation of an Element
- Solving a Structure
- Showing Deformation and Force Results
- Used in Bash Mode; ruBeam engine, see chapter 2
- Used as Plugin; ruBeam plugin on CADEMIA, see chapter 3
- Used in Structural, Optimization and Study Processes

1.1.2 General

The ruBeam model consist of nodes, elements and loads which are totally independent from each other when generated and form the *base* components of a structure. These *base* components can be extended with different components *extend*-... and modified. Table 2.1 describes every single *base* and *extend* component being part of ruBeam model. Fig. 1.1 shows the structure of ruBeam model and how the ruBeam engine and the ruBeam plugin is linked to it.

[1] CADEMIA Version 1.5 b13 intermediate Copyright © 2002-2009 B. Firmenich, www.cademia.org

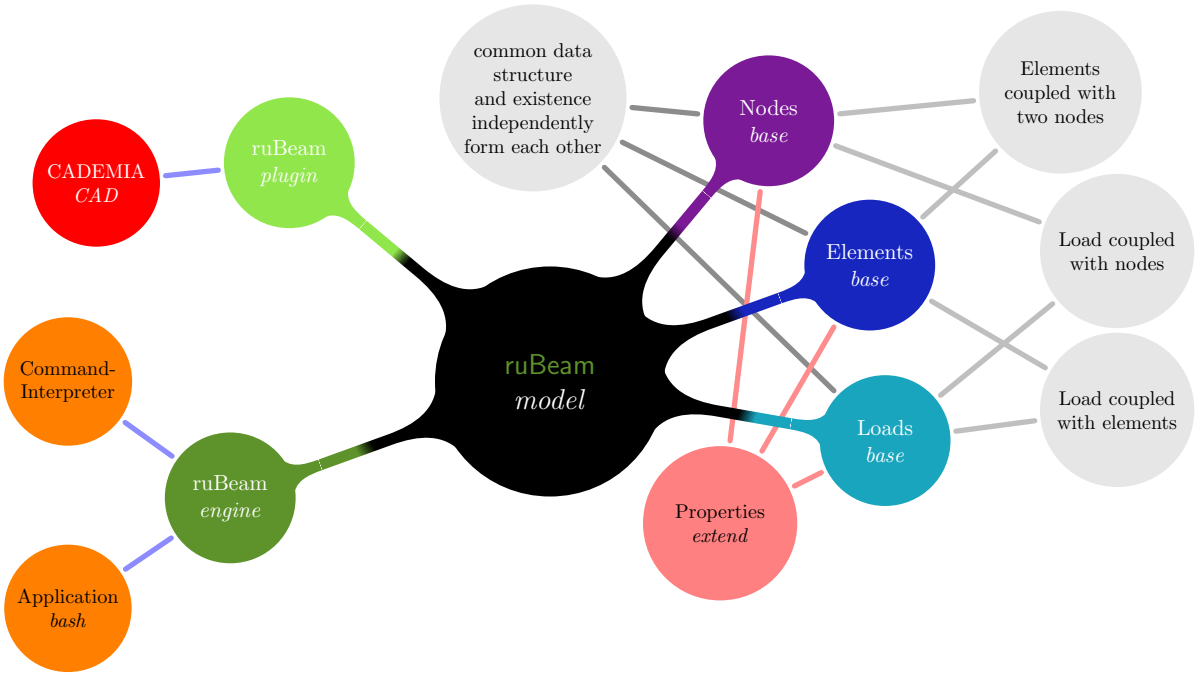


Figure 1.1: Schema of ruBeam model and its components

So as long as the same data structure for every type of component is guaranteed without any problems new types of load or elements, can be developed and implemented to the ruBeam model.

Of course it makes no sense to generat not assigned loads or elements so for this reason the commands in ruBeam engine directly assign elements to nodes and loads to elements, this is explained in more detail in section 2.2.

But where it is of great importance to be able to handle every single component on its own is in the case of using strictly the CAD concept of CADEMIA. So its possible to *edit, copy, paste,...* every single component and agreeing on the basic concept of CAD program. When running ruBeam engine we can take advantage of editing every single component without reassigning the component relations.

1.1.3 Coordinate System Definitions

The coordinate in ruBeam model is defined as seen in Fig. 1.2.

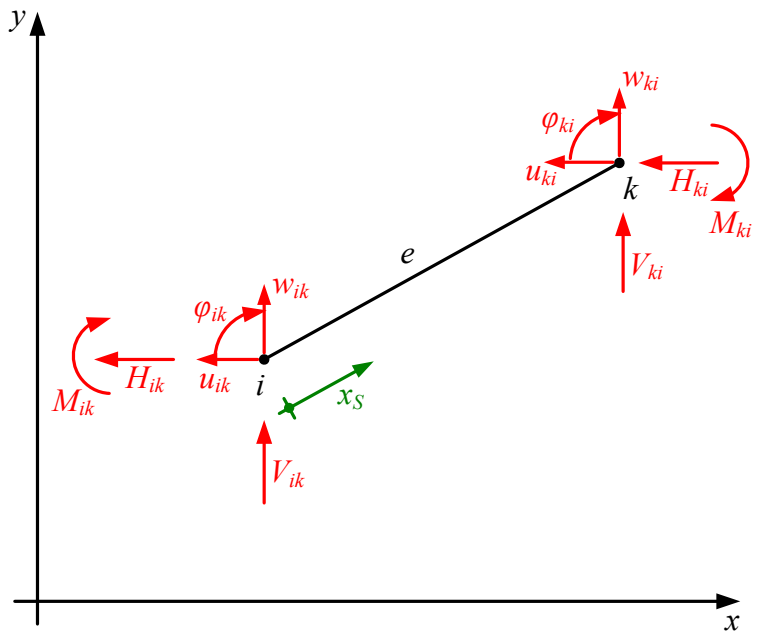


Figure 1.2: Orientation of condition vectors in global coordinate system and element directional orientation

Geometrical Coordinates

The geometrical coordinates are defined positive to ***left***, $x-axis$ and ***up***, $y-axis$.

Displacement/Force Coordinates

The displacement/force coordinates are defined positive to ***right***, $u-axis$ horizontal displacement $H-axis$ horizontal force, ***up***, $w-axis$ vertical displacement $V-axis$ vertical force and ***clockwise***, φ rotation M moment.

Element Coordinates

The element positive direction is always from the point i to the point k, see Fig. 1.2

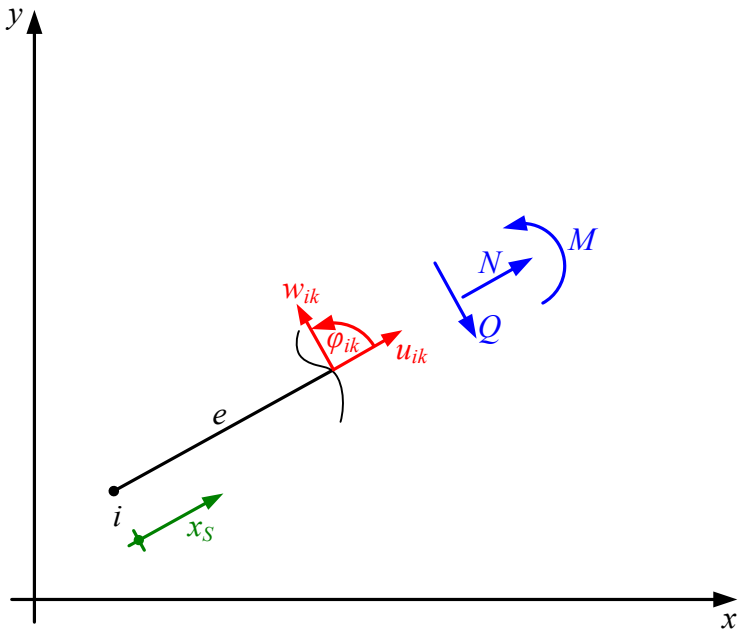
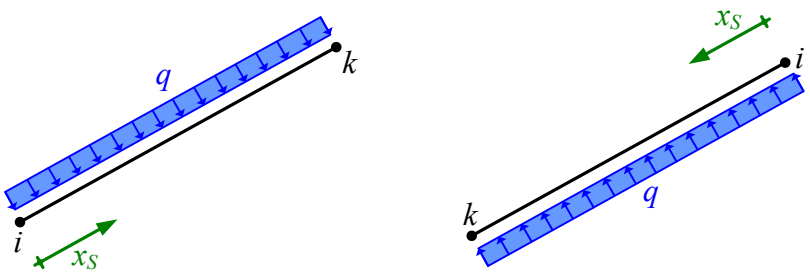


Figure 1.3: Orientation of condition vectors in local coordinate system relative to element.

Load Coordinates

Be aware that loads are linked to the directional element orientation vector, which starts at the first node and points to the second node. This means that the loads are constructed in relation to this vector, flipping over an element if the load is assigned to an element, leads to flipping over the load of course too, see Fig. 1.4. What is the same for all types of loads, see Tab. 2.1 for detailed informations regarding the loads.



(a) (b)

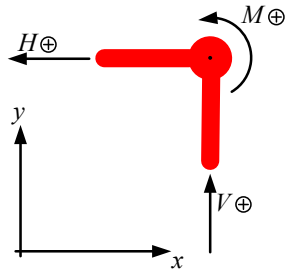
Figure 1.4: Load behavior for flipped over elements

Node Result Vector Coordinates

The displacement coordinates are defined positive to ***left***, $u-axis$ horizontal displacement, ***up***, $w-axis$ horizontal displacement and ***clockwise***, φ.

The force coordinates are defined positive to ***right***, $H-axis$ horizontal force, ***up***, $V-axis$ vertical force and ***counter clockwise***, φ rotation M moment.
The values are stored in the following order

$$Y_n = \begin{bmatrix} \varphi_n \\ w_n \\ u_n \\ M_n \\ V_n \\ H_n \end{bmatrix}$$



Element Result Matrix Coordinates

global Matrix

Same definition like in Displacement/Force Coordinates see Fig. 1.2.
The values are sort in the following order

$$\begin{bmatrix} i & 1 & 2 & 3 & \cdots & k \\ \hline u_{ik,i} & u_{ik,1} & u_{ik,2} & u_{ik,3} & \cdots & u_{ik,k} \\ w_{ik,i} & w_{ik,1} & w_{ik,2} & w_{ik,3} & \cdots & w_{ik,k} \\ \varphi_{ik,i} & \varphi_{ik,1} & \varphi_{ik,2} & \varphi_{ik,3} & \cdots & \varphi_{ik,k} \\ M_{ik,i} & M_{ik,1} & M_{ik,2} & M_{ik,3} & \cdots & M_{ik,k} \\ V_{ik,i} & V_{ik,1} & V_{ik,2} & V_{ik,3} & \cdots & V_{ik,k} \\ H_{ik,i} & H_{ik,1} & H_{ik,2} & H_{ik,3} & \cdots & H_{ik,k} \end{bmatrix} \tag{1.1}$$

local Matrix

Same definition like in Displacement/Force Coordinates see Fig. 1.3.
The values are sort in the following order

$$\begin{bmatrix} i & 1 & 2 & 3 & \cdots & k \\ \hline \bar{u}_{ik,i} & \bar{u}_{ik,1} & \bar{u}_{ik,2} & \bar{u}_{ik,3} & \cdots & \bar{u}_{ik,k} \\ \bar{w}_{ik,i} & \bar{w}_{ik,1} & \bar{w}_{ik,2} & \bar{w}_{ik,3} & \cdots & \bar{w}_{ik,k} \\ \bar{\varphi}_{ik,i} & \bar{\varphi}_{ik,1} & \bar{\varphi}_{ik,2} & \bar{\varphi}_{ik,3} & \cdots & \bar{\varphi}_{ik,k} \\ \bar{M}_{ik,i} & \bar{M}_{ik,1} & \bar{M}_{ik,2} & \bar{M}_{ik,3} & \cdots & \bar{M}_{ik,k} \\ \bar{Q}_{ik,i} & \bar{Q}_{ik,1} & \bar{Q}_{ik,2} & \bar{Q}_{ik,3} & \cdots & \bar{Q}_{ik,k} \\ \bar{N}_{ik,i} & \bar{N}_{ik,1} & \bar{N}_{ik,2} & \bar{N}_{ik,3} & \cdots & \bar{N}_{ik,k} \end{bmatrix} \tag{1.2}$$

1.1.4 Preliminaries

The basic concept of formulating and solving the differential equation for the structural system is done following the approach given by Rubin/Schneider[2]. The differential equation for a single element is solved by a series expansion of the solution function, which if enough terms are considered, gives the exact result.

For simple load types normally less then 6 terms have to be considered to obtain the exact solution for the differential equation. So for any element, see Fig. 1.2 the transfer matrix can be easily formulated

$$\begin{bmatrix} u_{ik} \\ w_{ik} \\ \varphi_{ik} \\ H_{ik} \\ V_{ik} \\ M_{ik} \end{bmatrix} = \begin{bmatrix} g_{11} & g_{12} & g_{13} & g_{14} & g_{15} & g_{16} \\ g_{21} & g_{22} & g_{23} & g_{24} & g_{25} & g_{26} \\ g_{31} & g_{32} & g_{33} & g_{34} & g_{35} & g_{36} \\ g_{41} & g_{42} & g_{43} & g_{44} & g_{45} & g_{46} \\ g_{51} & g_{52} & g_{53} & g_{54} & g_{55} & g_{56} \\ g_{61} & g_{62} & g_{63} & g_{64} & g_{65} & g_{66} \end{bmatrix} \cdot \begin{bmatrix} u_{ki} \\ w_{ki} \\ \varphi_{ki} \\ H_{ki} \\ V_{ki} \\ M_{ki} \end{bmatrix} + \begin{bmatrix} g_1^L \\ g_2^L \\ g_3^L \\ g_4^L \\ g_5^L \\ g_6^L \end{bmatrix} \tag{1.3}$$

in which with some basic mathematics the deformations $u_{ik}, w_{ik}, \varphi_{ik}; u_{ki}, w_{ki}, \varphi_{ki}$ can be separated from the forces $H_{ik}, V_{ik}, M_{ik}; H_{ki}, V_{ki}, M_{ki}$. $g_{11} \dots g_{66}$ mixed element stiffness and displacement properties, $g_1^L \dots g_6^L$ element load properties. Without further explanations and the help of condition vectors

$$S_{ik} = \begin{bmatrix} H_{ik} \\ V_{ik} \\ M_{ik} \end{bmatrix}, \quad S_{ki} = \begin{bmatrix} H_{ki} \\ V_{ki} \\ M_{ki} \end{bmatrix}, \quad V_i = \begin{bmatrix} u_{ik} \\ w_{ik} \\ \varphi_{ik} \end{bmatrix}, \quad V_k = \begin{bmatrix} u_{ki} \\ w_{ki} \\ \varphi_{ki} \end{bmatrix}, \quad S_{ik}^0 = \begin{bmatrix} g_{\dots}^L \\ g_{\dots}^L \\ g_{\dots}^L \end{bmatrix} \tag{1.4}$$

we can express the base equation for the transfer matrix approach for any element s

$$\begin{aligned} S_{ik} &= K_{is} \cdot V_i + K_{ik} \cdot V_k + S_{ik}^0 \\ S_{ki} &= K_{ks} \cdot V_k + K_{ki} \cdot V_i + S_{ki}^0 \end{aligned} \tag{1.5}$$

where V_i, V_k are the displacement vectors, S_{ik}^0, S_{ki}^0 the load vectors of the element s; $K_{is}, K_{ik}, K_{ks}, K_{ki}$ can be seen as stiffness matrixes of the element s.

So for every node of the structure we can formulate the equilibriums conditions

$$K_{ii} \cdot V_i + \sum_k K_{ik} \cdot V_k + S_i^0 = 0 \tag{1.6}$$

with

$$K_{ii} = \sum_k K_{is} \tag{1.7}$$

and

$$S_i^0 = \sum_k S_{ik}^0 + S_i^e \tag{1.8}$$

where $\sum_k$ means the sum over all elements connected to the node i with opposite element node k and S_i^e is on the node applied load vector, S_{ik}^0, S_{ki}^0 are the element node forces caused by the element loads.

[2]Baustatik Theorie I. und II. Ordnung, 4. Auflage, Werner Verlag 2002
The book is as fare as known only published in German; some papers dealing with this argument exists in English.

1.1.5 Dimensions

ruBeam model is dimension free, which means that as long as combinations of different dimensions are conform the result dimension will be directly derivable from the input dimensions. As input dimensions we only have **Length** and **Force** which other input parameters have to be deviated. The **Angle** as input and output parameter is generally [°] for loads, [rad] for rotations and boundary conditions, this is exactly specified for the very ruBeam component.

Example

We use for **Length** [m] and for **Force** [N], the young's modulus turns to be [N/m^2], the moment [Nm] and so on. So if once decided for on unit system every value has to be set in the same unit system.

1.2 Further Development on the ruBeam model

Elements For the moment it is better to use always $h_k < 2h_i$ and start with the bigger section dimension at node i of the element, this purpose will be removed in the following versions of ruBeam.

Element description function In order to obtain exact result even for conical sections of all different types the element description function will be updated.

Theory of Second Order Due to consider pre deformations of the elements second order effects will be taken into consideration.

Stability Stability of elements and structures will be analysed, buckling shapes and buckling loads of the structure will computed.

Dynamics Linear dynamic analysation of structures computation of Eigen frequencies and Eigen shapes of the system.

Chapter 2

ruBeam Engine

In this chapter introduction to the ruBeam engine is given as well as the Installation Instructions for the ruBeam engine, a Quick Start Guide for the ruBeam engine, and a full documentation of all components included in ruBeam model.
If you are interested in the ruBeam plugin you can directly go to chapter 3 ruBeam Plugin on page 26.

2.0.1 General

ruBeam engine is the link of ruBeam model to a command-line interpreter and uses its own language. ruBeam engine connects the loads directly to the elements in order to prevent losing the overview of all components being part of the structure or not.

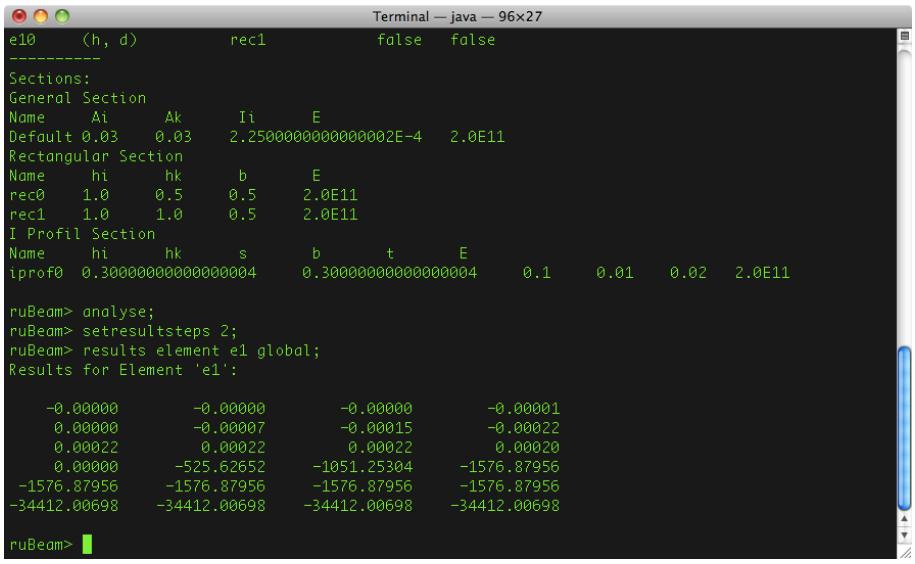


Figure 2.1: Screenshot of the running ruBeam engine in bash mode

In this chapter the ruBeam engine is described. In the next chapter the ruBeam plugin on CADEMIA is explained. The documentation is concluded with a set of examples and some special usage of the ruBeam engine combined with *MATLAB* is shown. For detailed information of the Java source code see java doc.

2.0.2 Installing ruBeam Engine

The installation of the ruBeam engine is quite easily done with a few steps.

System Requirements: *Java SE Development Kit (JDK) or Java SE Runtime Environment (JRE) Version 5.0* and higher must be installed on the system in order to run ruBeam properly.

Linux, Mac OS, Windows

1. Make sure *Java* is installed and runs properly, type `java` in a command-line interpreter of your system in order to see if *Java* is running on the system.
2. Generate a workspace folder.
3. Copy the ruBeam engine file *ruBeamEngine.jar* into that folder.
4. Lunch any prefered command-line interpreter no graphical user interface is needed.
5. Type `java -jar ruBeamEngine.jar [-options] arg` with any arguments, see 2.2.1, and press return.
6. With `help;` a list of all commands contained in ruBeam engine are shown.
7. Enjoy driving the ruBeam engine.

2.1 ruBeam Engine Quick Start

1. **Lunch rubeam Engine**
   use a command-line interpreter or call it out from any other program.
2. **Create a Structure**
   or use a ruBeam import file.
3. **Analyse the Strucutre**
   and export the results to the screen or to a file.

2.2 ruBeam Engine Language

The ruBeam engine Language is a easy to use language, where with a set of very simple commands the hole system can be generated solved and the system and its results exported into `ascii` files. Firstly some basics are given, followed by the basic components, section 2.2.2, and the handling commands, section 2.2.3. In the section 2.2.8 the recommended command sequence is shown.

2.2.1 Basics

The basic idea of ruBeam engine language is that a component is created with default properties and afterwards its wished properties are set.

Be careful it can happen that the system will be analyzed with not proper set properties!

Commends

Commends can be added line wise with a `//` symbol at the beginning of every line supposed to be ignored by the ruBeam engine.

End of Command

Commands are ended with a `;` so more than one command can be written in one line.

Help

With `help;` a short help document is displayed containing a list off all commands.

Mathematical Expression

- You can compute values by simple writing the mathematical expression, like `1+4` will use the value `5` for the set variable.
- So `+` stands for adding, `-` for subtracting, `*` for multiplying, `/` for dividing and `^` for power.
- Furthermore you can use `2e10` for exponential number input, of course `2E+10` ore `2E-10` is accepted.
- For performing mathematical calculations `calc expression;`, e.g. `calc 1+2^(2.2/4);`, can be called every time without taking any effect on the structural model.
- To prevent round of errors due to integer numbers always use at least one value in the decimal notation `1.0`.

Import/Export

- With `import "filename";` the components (in the file `filename`; can be absolute path) will be added to the structure and all handling commands (in the file `filename`) will perform its operations on the structure, if no structure exist a new one will be generated.
- When calling the ruBeam engine with `-f filename` in the shell the file will be read directly after start up of ruBeam engine and the engine will quit after reading and processing the input file.
- With `export "filename";` the hole structure is exported to the file `filename`, can be absolute path, in form of a sequence of ruBeam engine commands. If a file with the same name exist it will be overwritten.
- With `write type "filename";` if `type` is `results` all results will be written to the file `filename`; if `type` is `structure` the structures informations are written to the file. If no file is specificated the result will just be shown. If a file with the same name exist it will be overwritten.

Show

- With `show;` the hole components in the current workspace are shown.

Clear, Remove

- With `clear;` the hole structure is deleted without any system prompt, and is not undoable!
- With `remove type name;` the component with the `type`, node, element, nodalload, elementload and the `name` will be removed. This command is not undoable.
- When a node with linked elements is removed all linked elements will be removed, if an element is deleted the nodes won't be deleted.

Analysing the structure, Clean

- With `analyse;` the structure will be analysed and all values for the computation of the results exist.
- With `clean;` the analysation is cleaned, which means no results exist anymore.

Results

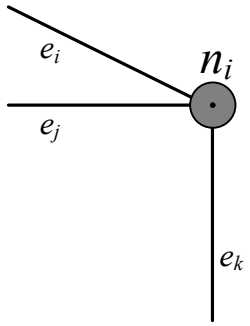
- With `results type name global/local;` the result for the component `node` or `element` and with `global` for global element condition vector and `local` for the local element condition vector, will be displyed on the screen. When component `node` is chosen for `global/local` nothing has to be set and the commands end after the nodes name.
- With `optiresultv nodename "filename";` the displacement vector of the `node` will be saved to the file `filename` without any additional information. If a file with the same name exists it will be overwritten.

Exit

- With `exit;` ruBeam engine will be quit, be aware all data are lost and no save prompt will be sent.

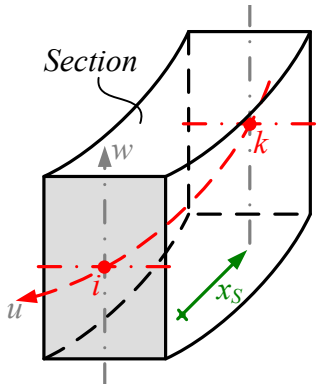
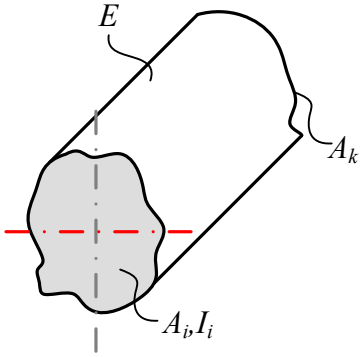
2.2.2 ruBeam Components and their Commands

Table 2.1: Components in the ruBeam application

Figure	Description
ruBeam Components	
	**Component:** Node *2D, 3 degrees of freedom, [base]* **Description:** The component node is described by its coordinates `x y` and a designate name `name`. The node has 3 degrees of freedom; *horizontal, vertical, rotation* [rad]. Nodes are use to construct elements, of course the number of nodes needed by the element must exist. Nodes can be used to define boundary conditions for the structure. Node loads can be applied on nodes. **Command:** `addnode x y name;` `x y` coordinates of node `name` wished name of node ***Command Example:*** `addnode 0.1 1.2 a;`

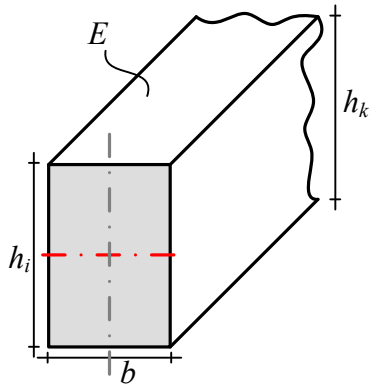
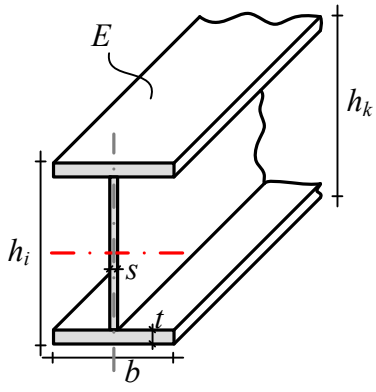
Continued on next page

Table 2.1 – continued from previous page

Figure	Description
	**Component:** Element *conical, flexible, linear elastic [base]* **Description:** Element from point i to k, with linear elastic material property; young's modulus. Element is linear conic in high, so the height has to be set at the beginning and the end of the element, see element sections for further informations and section 1.2. Section can be set as element properties. Element loads can be applied on elements. **Command:** `addelement ni nk name;` `ni` node for point i `nk` node for point k `name` wished name of element ***Command Example:*** `addelement a b e1;`
ruBeam Element Sections	
	**Component:** Section *conical, general [base]* **Description:** General conical element section, fits with conical elements. The element is linear conical in area so the area can be set at the beginning and end. **Command:** `addgensection Ai Ak Ii E name;` `Ai` area at the beginning of the element `Ak` area at the end of the element `Ii` moment of inertia at the beginning of the element `E` young's modulus of the element material `name` wished name of section ***Command Example:*** `addrecsection 0.03 0.04 0.0023 2e10 sg1;`

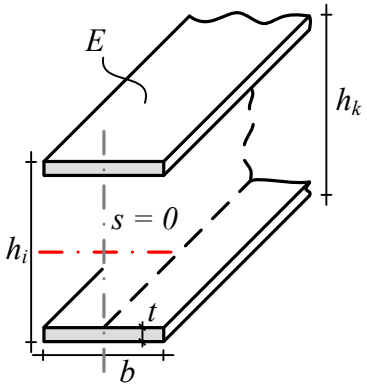
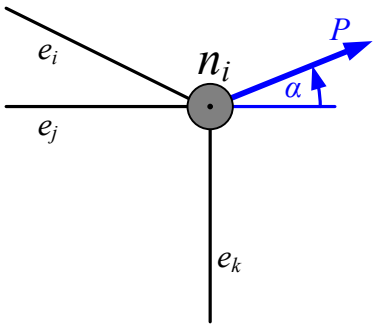
Continued on next page

Table 2.1 – continued from previous page

Figure	Description
	**Component:** Section *conical, rectangular [base]* **Description:** Rectangular conical element section, fits with conical elements. The element is linear conical in high so the high can be set at the beginning and end. **Command:** `addrecsection hi hk b E name;` `hi` high at the beginning of the element `hk` high at the end of the element `b` breadth of the element `E` young's modulus of the element material `name` wished name of section ***Command Example:*** `addrecsection 0.5 0.6 0.3 2e10 sr1;`
	**Component:** Section *conical, I-profile [base]* **Description:** I shaped profile conical element section, fits with conical elements. The element is linear conical in height so the height can be set at the beginning and end. **Command:** `addiprofsection hi hk s b t E name;` `hi` height at the beginning of the element `hk` height at the end of the element `s` thickness of the elements web `b` breadth of the elements flange `t` thickness of the elements flange `E` young's modulus of the element material `name` wished name of section ***Command Example:*** `addiprofsection 0.5 0.6 0.01 0.3 0.02 2e10 si2;`

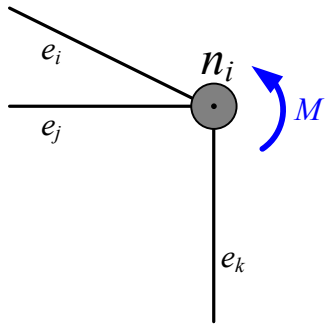
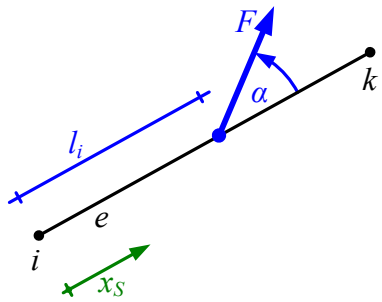
Continued on next page

Table 2.1 – continued from previous page

Figure	Description
	**Component:** Section *conical, sandwich [base]* **Description:** Sandwich means the web has no bending stiffness only flanges are considerted. Sandwich conical element section, fits with conical elements. The element is linear conical in height so the height can be set at the beginning and end. **Macro Command:** `addsandsection hi hk b t E name;` `hi` height at the beginning of the element `hk` height at the end of the element `b` breadth of the elements flange `t` thickness of the elements flange `E` young's modulus of the element material `name` wished name of section ***Command Example:*** `addsandsection 0.5 0.6 0.3 0.02 2e10 ss3;`
ruBeam Nodal Loads	
	**Component:** nodal load *concentrated, angular [base]* **Description:** Concentrated off horizontal angular nodal load, applied on a node. Angel positive counter clockwise, load positive radial from node. **Command:** `addnodalloadang ni P alpha name;` `ni` name of node `P` value of load `alpha` value of load off horizontal angle in [°] `name` wished name of load ***Command Example:*** `addnodalload a 1000 10 P1;`

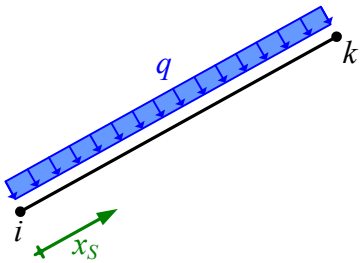
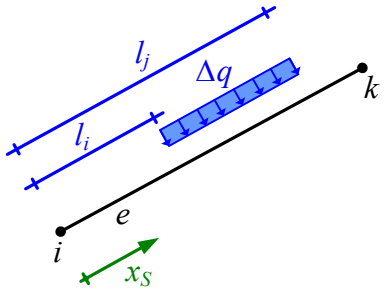
Continued on next page

Table 2.1 – continued from previous page

Figure	Description
	**Component:** nodal moment *[base]* **Description:** Nodal moment, applied on a node. Moment positive clockwise. **Command:** `addnodalmoment ni M name;` `ni` name of node `M` value of moment `name` wished name of load ***Command Example:*** `addnodalmoment a 1300  Mn1;`
ruBeam Element Loads	
	**Component:** element concentrated load *angular to element longitudinal axis [base]* **Description:** Concentrated off horizontal angular element load, applied on an element. Angel positive counter clockwise from element direction xs, load positive radial from point. **Command:** `addelementconload e F alpha li name;` `e` name of element `F` value of load `li` load distance from element node i `alpha` value of load off horizontal angle in [°] `name` wished name of load ***Command Example:*** `addelementconload e1 1000 1.1 12.3  F1;`

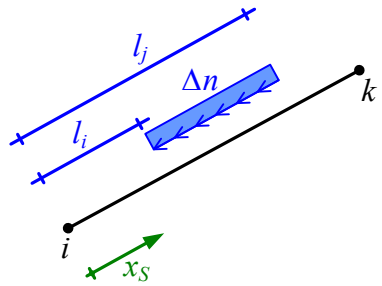
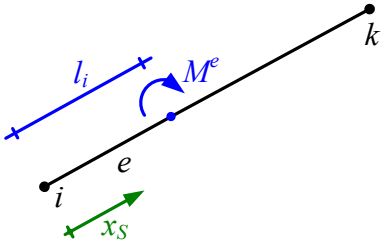
Continued on next page

Table 2.1 – continued from previous page

Figure	Description
	**Component:** element uniform distributed load *normal to element longitudinal axis [base]* **Description:** Uniform distributed element load, applied on an element, orientated right normal to element direction xs. **Command:** `addelementunidisloadnor e q name;` `e` name of element `q` value of load `name` wished name of load ***Command Example:*** `addelementunidisloadnor e1 10000 q1;`
	**Component:** element uniform distributed load sectional *normal to element longitudinal axis [base]* **Description:** Uniform distributed element load sectional, applied on an element, orientated right normal to element direction xs. **Command:** `addelementunidisloadsecnor e dq li lj name;` `e` name of element `dq` value of load `li` load distance from element node i `lj` load distance from element node i `name` wished name of load ***Command Example:*** `addelementunidisloadsecnor e1 10000 1.1 2.3 dq1;`

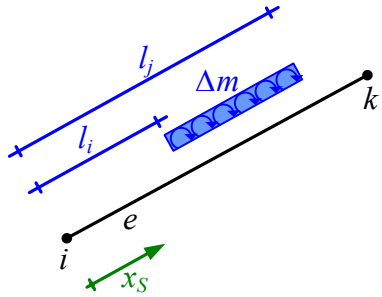
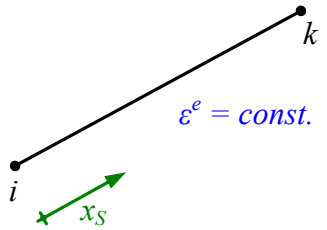
Continued on next page

Table 2.1 – continued from previous page

Figure	Description
	**Component:** element uniform sectional distributed load *directional to element longitudinal axis [base]* **Description:** Uniform distributed element load sectional, applied on an element, orientated directional contrary to element direction xs. **Command:** `addelementunidisloadsecdir e dn li lj name;` `e` name of element `dn` value of load `li` load distance from element node i `lj` load distance from element node i `name` wished name of load ***Command Example:*** `addelementunidisloadsecdir e1 3000 0.4 1.6 dn1;`
	**Component:** element moment *bending uniaxial, major axis [base]* **Description:** Moment at point li of element directional to xs, moment positive clockwise. **Command:** `addelementmoment e M li name;` `e` name of element `M` value of moment `li` load distance from element node i `name` wished name of load ***Command Example:*** `addelementmoment e1 1200 2.3 Me1;`

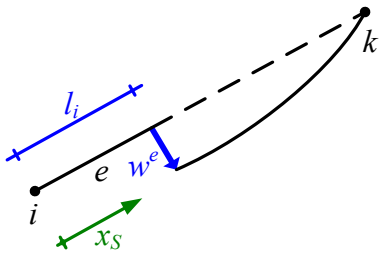
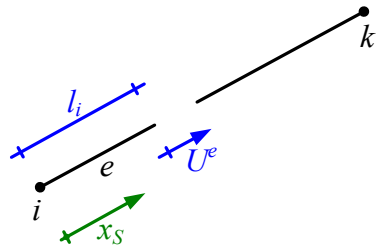
Continued on next page

Table 2.1 – continued from previous page

Figure	Description
	**Component:** element uniform moment sectional *bending uniaxial, major axis [base]* **Description:** Uniform distributed element moment sectional, applied on an element, positive clockwise. **Command:** `addelementunidismomentsec e dm li lj name;` `e` name of element `dm` value of moment `li` load distance from element node i `lj` load distance from element node i `name` wished name of load ***Command Example:*** `addelementunidismomentsec e1 2300 0.5 1.1 dm1;`
	**Component:** element prestressing *longitudinal prestressing [base]* **Description:** Prestress in manner of applying a constant strain of the value ε^e a positive ε^e will cause pressure in the element, a negative ε^e tension in the element. **Command:** `addelementprestress e eps name;` `e` name of element `eps` value of prestress `name` wished name of load ***Command Example:*** `addelementprestress e1 11e+6 pS1;`

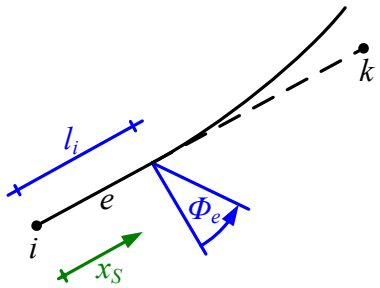
Continued on next page

Table 2.1 – continued from previous page

Figure	Description
	**Component:** element leap *normal to element longitudinal axis [base]* **Description:** Leaps the element orientated right normal to the element direction at point li with the given value. **Command:** `addelementleapnor e we li name;` `e` name of element `we` value of leap `li` load distance from element node i `name` wished name of leap ***Command Example:*** `addelementleapnor e1 0.01 1.2 lwe1;`
	**Component:** element leap, *directional to element longitudinal axis [base]* **Description:** Leaps the element orientated directional to the element direction at point li with the given value. **Command:** `addelementleapdir e ue li name;` `e` name of element `ue` value of leap `li` load distance from element node i `name` wished name of leap ***Command Example:*** `addelementleapdir e1 0.21 1.4 lue1;`

Continued on next page

Table 2.1 – continued from previous page

Figure	Description
	**Component:** element bend *bend uniaxial, major axis [base]* **Description:** Bends the element at point li positive counter clockwise. **Command:** `addelementbend e phie li name;` `e` name of element `phi` value of bend [rad] `li` load distance from element node i `name` wished name of leap ***Command Example:*** `addelementbend e1 12.3 0.4 lphie1;`

2.2.3 ruBeam Handling Commands

2.2.4 Set Node Properties *[extend]*

All node handle commands are operated on the node named `name`.

Set Name

`setnodename name newname`
Sets the name of the node to the name `newname`.

Set x-Coordinate

`setnodex name newX;`
Sets the x-coordinate to `newX`.

Set y-Coordinate

`setnodey name newY;`
Sets the y-coordinate to `newY`.

Set Coordinates

`setnodecoord name newX newY;`
Sets the node coordinate to `newX` and `newY`.

Values for Constraints

if `0`, this degree of freedom is fixed
if `?`, this degree of freedom is free, which is default setting, `?` will create a `NaN` so of course instead of `?`, `NaN` can be typed case sensitiv.
if `number`, this degree of freedom is imposed with the value `number`, for rotation φ the value must be set in [rad].

Set x-Constraint

`setnodeconstx name newfx;`
Sets the node x-constraint to `newfx`.

Set y-Constraint

`setnodeconsty name newfy;`
Sets the node y-constraint to `newfy`.

Set phi-Constraint

`setnodeconstphi name newfphi;`
Sets the node phi-constraint to `newfphi`.

Set Constraints

`setnodeconsts name newfphi newfy newfx;`
Sets the node constraints to `newdPhi newdY newdX`.

Set Fixed

`setnodefixed name;`
Sets all node constraints to 0.

2.2.5 Set Element Properties *[extend]*

All element handle commands are operated on the element named `name`.

Set Name

`setelementname name newname;`
Sets the name of the element to the name `newname`.

Set Node i

`setelementnodei name newnodei;`
Sets the node i of the element to the node with the name `newnodei`.

Set Node k

`setelementnodek name newnodek;`
Sets the node k of the element to the node with the name `newnodek`.

Set Section

`assignsection section name`
Sets the element section to the section with the name `section`.

Set Hinge Node i

`setelementhingei name newstate;`
Sets the element node i as hinged when `newstate=true`, default is `newstate=false`

Set Hinge Node k

`setelementhingek name newstate;`
Sets the element node k as hinged when `newstate=true`, default is `newstate=false`

2.2.6 Set Section Properties *[extend]*

All section properties con only be set if the section `name` has the specific value, otherwise the value will not be set!

Set Name

`setsectionname name newname`
Sets the name of the section to the name `newname`.

Set high i

`setsectionhi name newhi`

Sets the value h_i of the section to the value `newhi`.

Set high k

`setsectionhk name newhk`

Sets the value h_k of the section to the value `newhk`.

Set (flange) breadth b

`setsectionb name newb`

Sets the value b of the section to the value `newb`.

Set web thickness s

`setsections name news`

Sets the value s of the section to the value `news`.

Set flange thickness t

`setsectiont name newt`

Sets the value t of the section to the value `newt`.

Set Young's Modulus

`setsectione name newE`

Sets the element young's modulus to `newE`.

Set area i

`setsectionai name newAi`

Sets the value A_i of the section to the value `newAi`.

Set area k

`setsectionak name newAk`

Sets the value A_k of the section to the value `newAk`.

Set moment of inertia at i

`setsectionii name newIi`

Sets the value I_i of the section to the value `newIi`.

2.2.7 Set System Properties *[base]*

Visualize Steps

`setvisualizesteps value`

Sets the number of visualize steps to `value`, this command is not affecting the structure when working with the ruBeam engine.

Result Steps

`setresultsteps value`

Sets the number of element result intermediate steps to `value`, this command is not affecting the analysation, only the amount of inter steps when computing the element condition matrix. Only integer numbers can be set.

Analyse automatically

`autoanalyse state`

Sets if the analysation is performed automatically after every command, `state on` for yes or `off` for not.

2.2.8 Recommended Command Sequence

Following this sequence guaranties to not forget any component and have a full functional structure! Of course any commands can be part of the file as long as they manipulated existing components.

Preamble

1. Basic Informations

Corpus

1. Nodes; `addnode`
2. Elements; `addelement`
3. Element Hinges; `setelementhinge...`
4. Constraints; `setnodeconsts`
5. Sections; `add...section`
6. Section Assignement; `assignsection`
7. Nodal Loads; `addnodal...`
8. Element Load; `addelement...`

Addendum

1. Visualize Steps; `setvisualizesteps`
2. Analyse; `analyse`

2.2.9 Generating ruBeam input files

When generating input files which should work with both, ruBeam engine and ruBeam plugin and are not created by ruBeam only this commands are allowed to be used. If the input file is only used with the ruBeam engine all commands described in section 2.2 can be used. At any time a redundancy free export file can be created by using the export command either in ruBeam engine or ruBeam plugin.

Components

- `addnode`
- `addelement`
- `addgensection`
- `addrecsection`
- `addiprofsection`
- `addsandsection`
- `addconloadang`
- `addnodalloadang`
- `addelementconload`
- `addunidistribloadnor`

Set Node Properties

- `setnodename`
- `setnodex`
- `setnodey`
- `setnodeconstx`
- `setnodeconsty`
- `setnodeconstphi`
- `setnodeconsts`
- `setnodefixed`

Set Element Properties

- `setelementname`

- setelementnodei
- setelementnodek
- assignsection
- setelementhingei
- setelementhingek

Set Section Properties

- setsectionname
- setsectionai
- setsectionak
- setsectionb
- setsectione
- setsectionhi
- setsectionhk
- setsectionii
- setsections
- setsectiont

Set Element Properties

- show
- analyse
- setvisualizesteps
- autoanalyse

Chapter 3

ruBeam Plugin

This chapter deals with the ruBeam plugin. After introducing CADEMIA plugin development in general, the architecture and the functionality of the ruBeam plugin are described.

3.1 Introduction

3.1.1 General

*"As a solution, a new **platform for geometry-oriented AEC applications** has been developed at Bauhaus University Weimar: CADEMIA. While CADEMIA was originally written for teaching and research purposes it is now available as **open source software**. A long time experience in the development of these systems in the building industry forms the base on which CADEMIA is built. [...] CADEMIA is a modular constructed software and offers lots of **possibilities to integrate other functionalities**. Due to this flexibility CADEMIA can easily be custom-modeled to fit individual needs. CADEMIA is programmed in JAVA and therefore platform independent. This way it serves as an inexpensive alternative to proprietary CAD software."*[1]

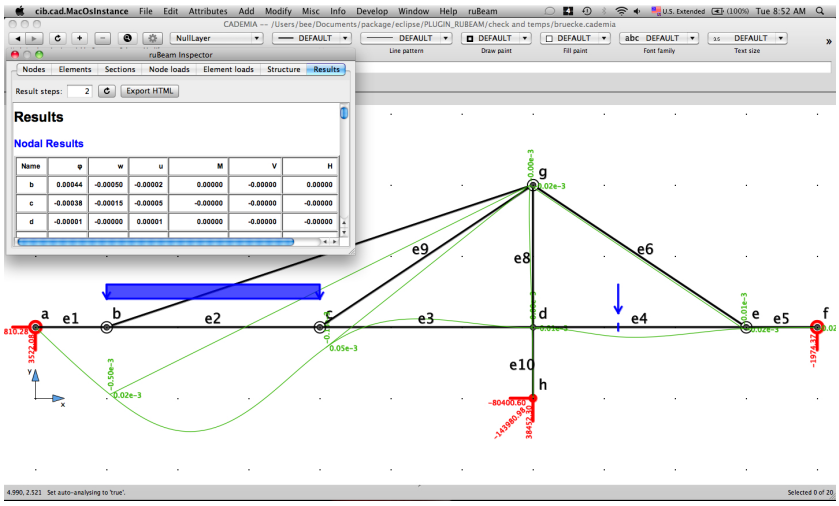


Figure 3.1: Screenshot of the running ruBeam Plugin on CADEMIA

[1] see http://www.cademia.org

3.1.2 Installing ruBeam Plugin

In order to add the ruBeam plugin to CADEMIA proceed with the following steps.

System Requirements: *Java SE Development Kit (JDK) or Java SE Runtime Environment (JRE) Version 5.0* and higher must be installed on the system in order to run ruBeam properly.

Linux, Mac OS, Windows

1. Lunch CADEMIA.
2. Use menu *Misc* → *Add plugin* and the select the downloaded and extracted plugin file `ruBeamPlugin.cademia_plugin`.
3. Enjoy using the ruBeam plugin.

3.2 Preliminaries to CADEMIA plugin development

The CADEMIA platform consists of four subsystems:

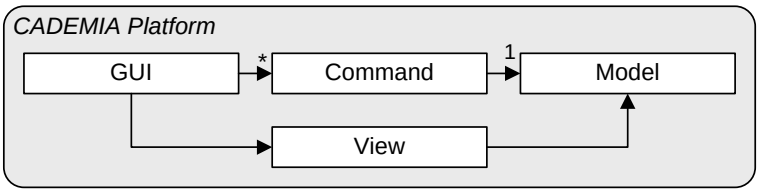


Figure 3.2: CADEMIA platform architecture

Graphical user interface (GUI) In the GUI the user can issue commands via text input, menu bar, tool bar and mouse input. The model is visualized by the view subsystem that is part of the GUI.

Command subsystem Commands are internally represented by text based on the CADEMIA command language. The model is edited via commands that can be undone and redone.

Model subsystem The model includes application objects that have a geometric representation and can be processed via commands.

View subsystem The graphical view of the model objects is handled by the view subsystem.

A more detailed introduction to CADEMIA plugin development can be found on http://www.cademia.org/frontend/index.php?page_id=10615.

3.3 ruBeam Plugin Architecture

In this section the main packages of the ruBeam plugin are introduced, see Fig. 3.3.

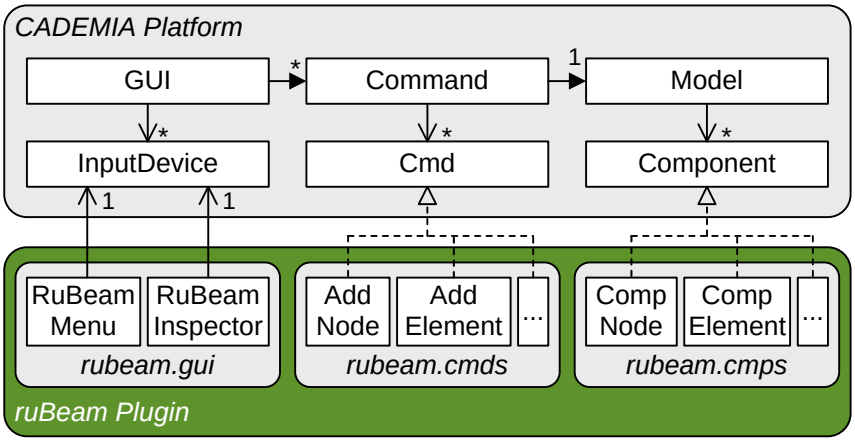


Figure 3.3: ruBeam Plugin architecture

rubeam.gui This package includes the classes for the plugin menu (RuBeamMenu) and for the ruBeam Inspector (RuBeamInspector). Both the menu and the inspector use a CADEMIA input device (InputDevice) in order to issue ruBeam plugin commands.

rubeam.cmps This package contains structural ruBeam plugin components (e.g. ComponentNode, ComponentElement, ...) that are vizualized in the graphical user interface.

rubeam.cmds In this package the classes representing ruBeam plugin commands for editing the structure (e.g. AddNode, AddElement, ...) are summarized.

3.3.1 ruBeam Plugin Components

According to the CADEMIA architecture the model includes geometric components that have to implement the component interface defined in *cib.cad.db.comp.Component*. For this reason all ruBeam plugin components are implemented as CADEMIA components. Fig. 3.4 illustrates the package *rubeam.cmps* consisting of ruBeam plugin components.

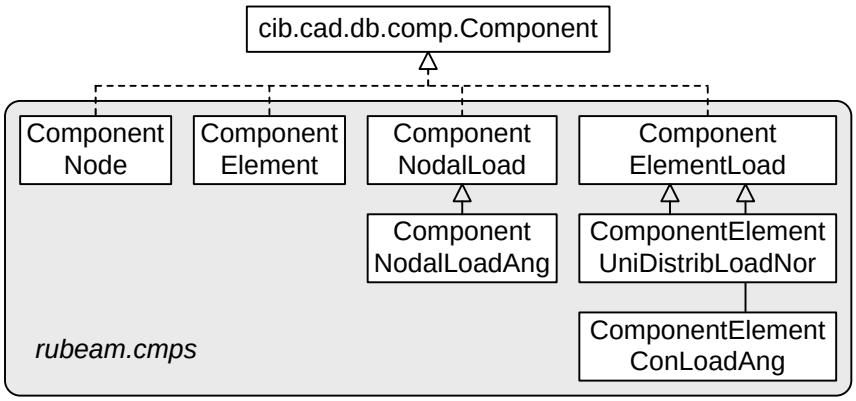


Figure 3.4: ruBeam Plugin Components

While the CADEMIA component interface defines the visual representation of a ruBeam plugin component in the CAD environment, geometry information and structural properties are stored in the ruBeam components introduced in secion 2.2.2. For this reason each structural ruBeam component is mapped to or wrapped by a ruBeam plugin component. Tab 3.1 illustrates this mapping for the components currently implemented in the ruBeam plugin.

Table 3.1: Component mapping

ruBeam component	ruBeam plugin component
Node	ComponentNode
Element	ComponentElement
NodalLoadAng	ComponentNodalLoadAng
ElementUniDistribLoadNor	ComponentElementUniDistribLoadNor
ElementConLoadAng	ComponentElementConLoadAng

3.3.2 ruBeam Plugin Commands

According to the CADEMIA architecture the model is processed by commands that have to implement the Cmd interface defined in *cib.util.cmd.Cmd*. Therefore, all ruBeam plugin commands are implemented as CADEMIA commands. Fig. 3.5 illustrates the package *rubeam.cmds* containing the ruBeam plugin commands.

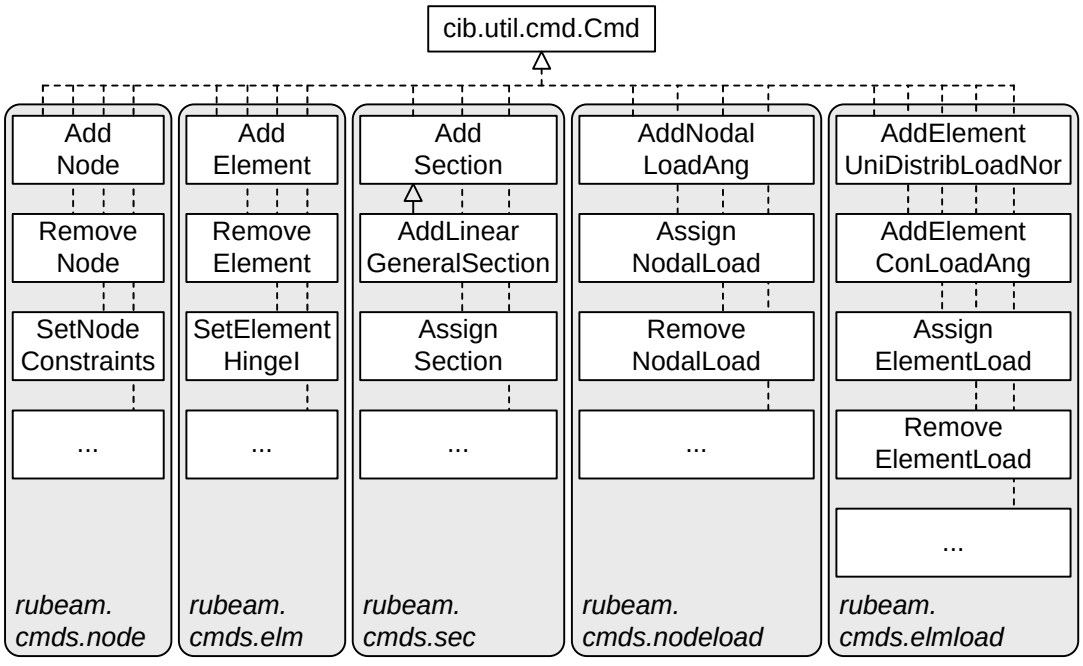


Figure 3.5: ruBeam Plugin Commands

In contrast to the ruBeam commands running directly on the ruBeam structure (see section 2.2.9) the ruBeam plugin commands run inside the CADEMIA environment and are therefore undoable and redoable.

3.3.3 ruBeam Inspector

The CADEMIA architecture allows to issue commands via an input device defined in *cib.cad.kernel.InputDevice*. Each user interaction in the ruBeam inspector (e.g. in the node table) is represented as a ruBeam plugin command that is issued via the inspector's input device. Fig. 3.6 illustrates the inpector (RuBeamInspector) and the related classes.

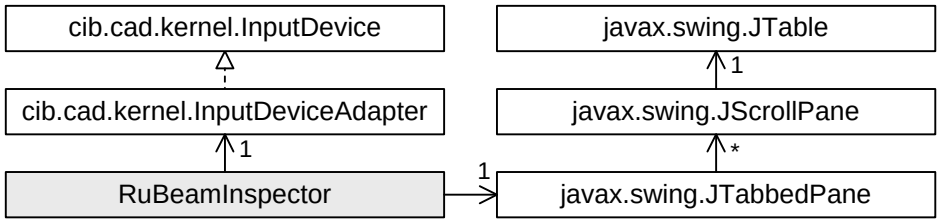


Figure 3.6: ruBeam Inspector

3.4 ruBeam Plugin Functionality

3.4.1 General

There are three ways of using ruBeam on CADEMIA

- One is to type all the commands on by on into the CADEMIA command line.
- The more common way is to use the ruBeam plugin Menu and
- the most fashionable way is to use the ruBeam plugin Inspector window.

In this section the coordinate system is redefined in order to be compatible with the CADEMIA coordinate system and the new symbolism of supports is explained. Furthermore, the Menu and the ruBeam Inspector are described. The available ruBeam plugin commands are listed in section 2.2.9.

3.4.2 Coordinate System Definition

In order to be consistent in the logical behaviour of the structural deformation the coordinate system orientation in the ruBeam plugin is changed slightly, be aware that this is just a changing in visualization the base computation coordinate system used in ruBeam model is not changed at all!

What was done is to change the orientation of the horizontal displacement $u-axis$ in order to follow the same direction as the CADEMIA defined positive to ***right***, $x-axis$, see Fig. 3.7.

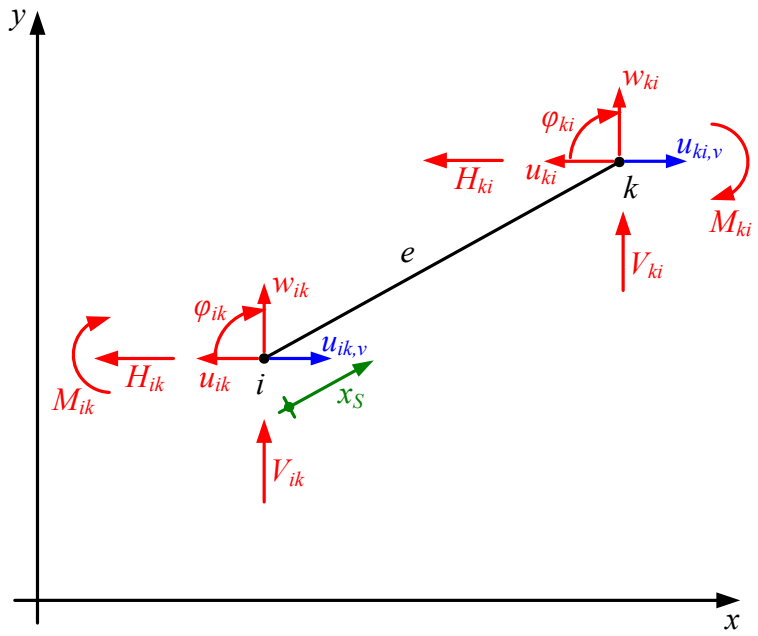
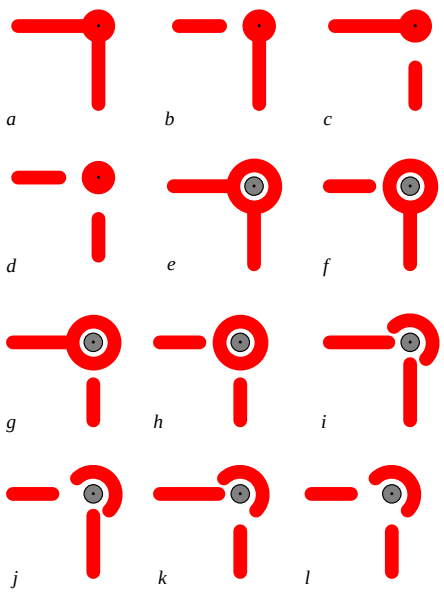


Figure 3.7: Orientation of condition vectors in global coordinate system and element directional orientation

But again this is only a change in the visualization and input parameter orientation. If the node constraint value dx is set positive in the ruBeam plugin, following the visualization schema in Fig. 3.7 the command will give back a negative value to the ruBeam model.

3.4.3 Support Symbolism

The support symbolism was recreated in ruBeam plugin, too. It was tried to find a new and more logical symbolism for the bounder conditions, supports, of the structural model.



Case	$d\varphi$	dy	dx
(a)	0	0	0
(b)	0	0	val
(c)	0	val	0
(d)	0	val	val
(e)	?	0	0
(f)	?	0	val
(g)	?	val	0
(h)	?	val	val
(i)	val	0	0
(j)	val	0	val
(k)	val	val	0
(l)	val	val	val

Figure 3.8: New symbolism of supports used in ruBeam plugin

To make it even more clear all combinations are shown in Fig. 3.8 and the set values for the node shown in table beside Fig. 3.8. Where ? means that this degree of freedom is set free and *val* means that this degree of freedom is set to a special value, 0 of course means fixed. If nothing is set for the displacements no line will appear, if no constraint is set at all for a node even the circle will not be visible.

3.4.4 ruBeam Menu

Plugin menu ruBeam

Analyse analyses the structure

Ananlyse automatically when turned on the structure will be analysed after every command automatically

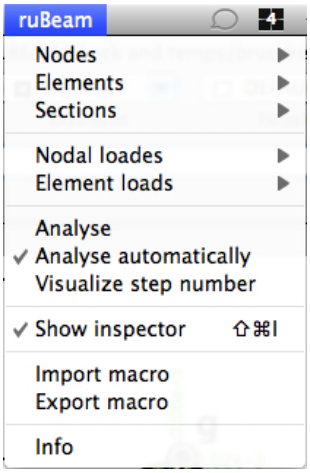
Visualize step number sets the visualize step number

Show inspector shows the ruBeam inspector window, see section 3.4.5

Import Macro imports a ruBeam macro file

Export Macro exports a ruBeam macro file

Info shows ruBeam Plugin info

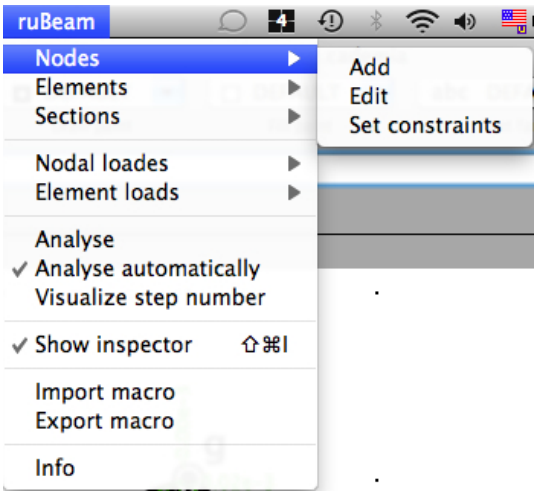


Menu item Nodes

Nodes → Add adds a node to the workspace

Nodes → Edit shows the node feature dialog

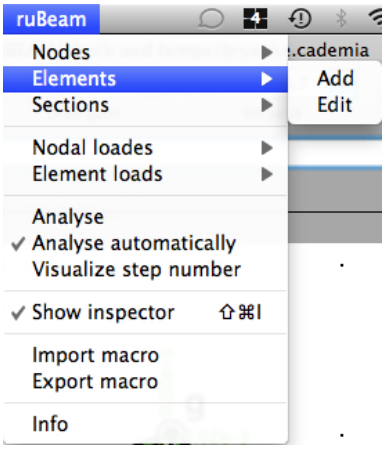
Nodes → Set constraints sets the node constraints



Menu item Elements

Elements → Add adds an element to the workspace

Elements → Edit shows the element feature dialog

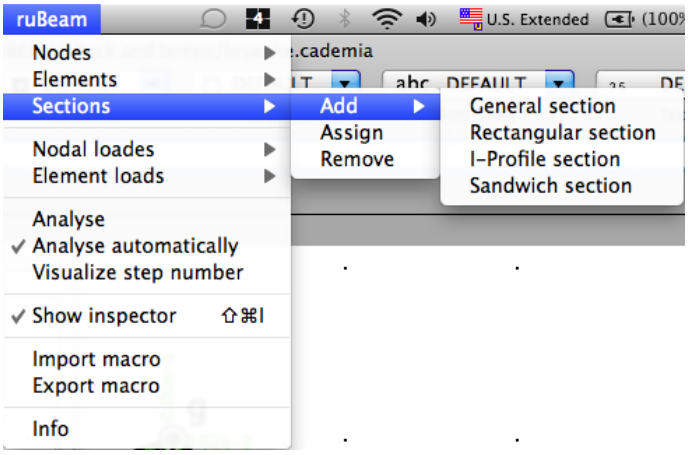


Menu item Sections

Sections → Add adds a new section to the workspace

Sections → Assign assigns a section to elements

Sections → Remove removes section

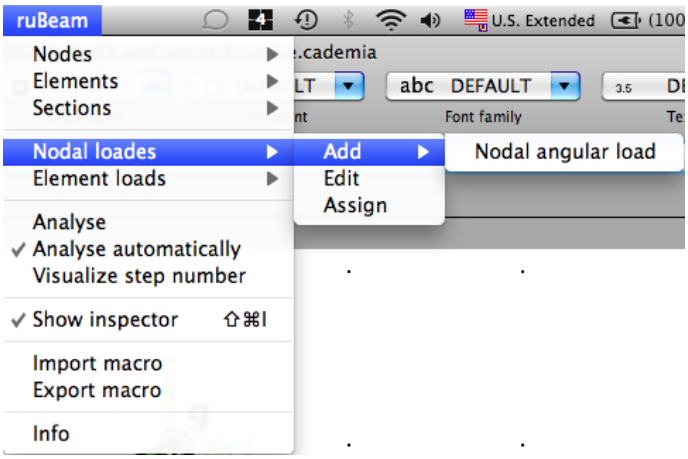


Menu item Nodal loads

Nodal loads → Add
adds a new nodal load to the workspace

Nodal loads → Edit
shows the nodal load feature dialog

Nodal loads → Assign
assigns nodal loads to one node

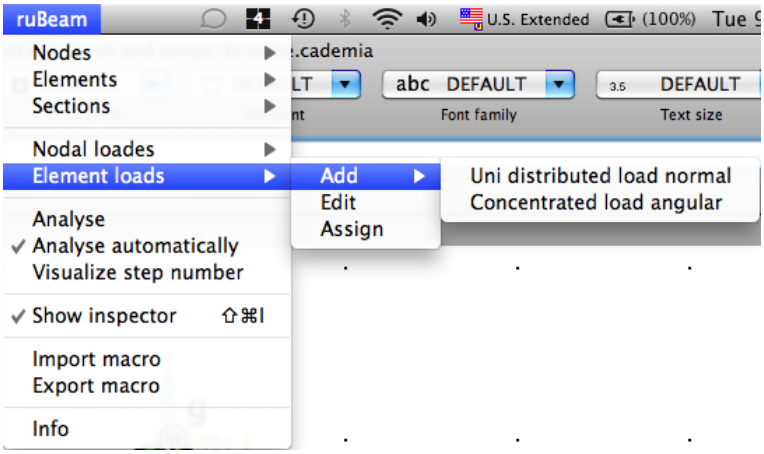


Menu item Element loads

Element loads → Add
adds a new element load to the workspace

Element loads → Edit
shows the element load feature dialog

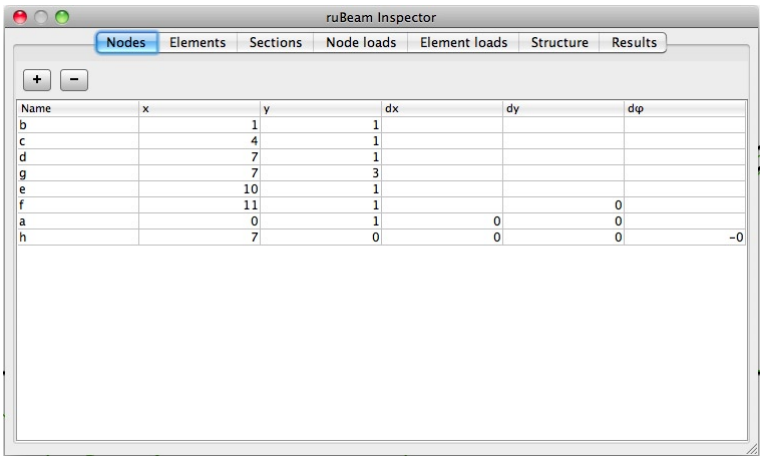
Element loads → Assign
assigns element loads to one element



3.4.5 ruBeam Inspector

Inspector Tab Nodes

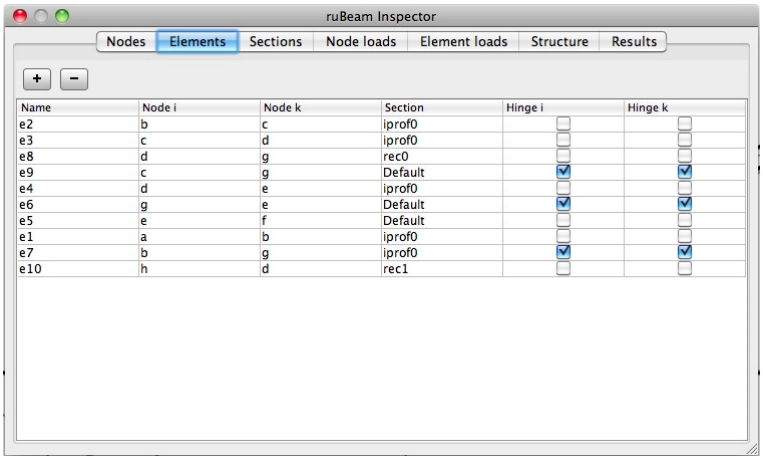
Nodes Tab displays all nodes in the workspace, all properties can be set, [+] button adds a node, [−], [×] button removes the selected line in the table.



Name	x	y	dx	dy	dφ
b	1	1			
c	4	1			
d	7	1			
g	7	3			
e	10	1			
f	11	1		0	
a	0	1	0	0	
h	7	0	0	0	-0

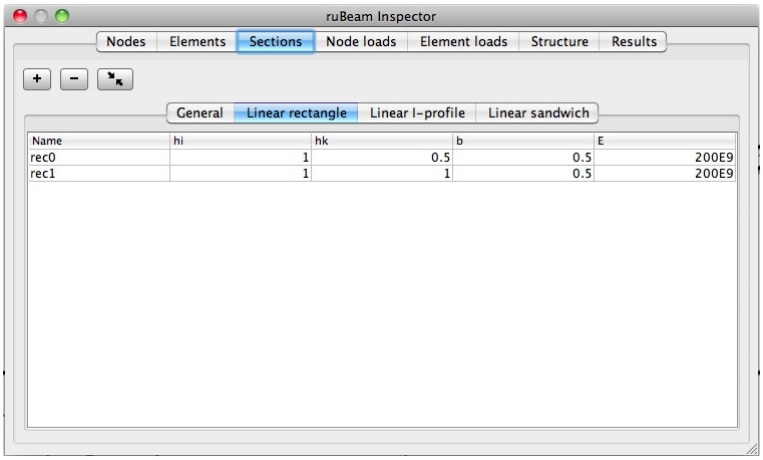
Inspector Tab Elements

Elements Tab displays all elements in the workspace, all properties can be set, [+] button adds a element, [–], [×] button removes the selected line in the table.



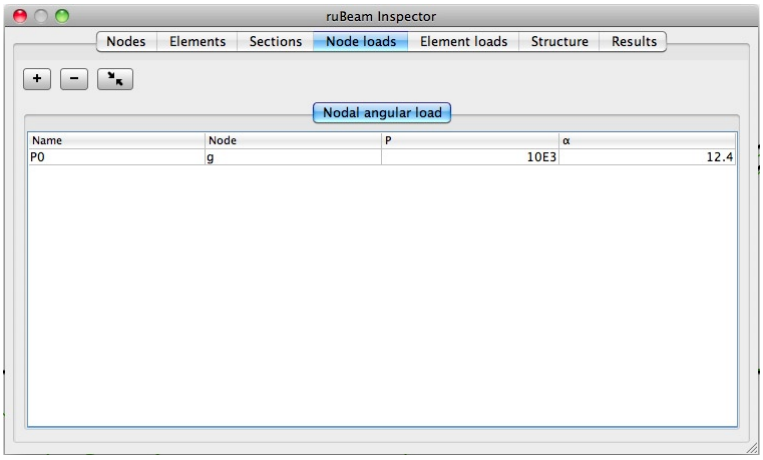
Inspector Tab Sections

Sections Tab displays all sections in the workspace type wise in different subtabs, all properties can be set, [+] button adds a section of the selected section subtab, [–], [×] button removes the selected line in the table, [↘ ↖] button assigns a section to one element.



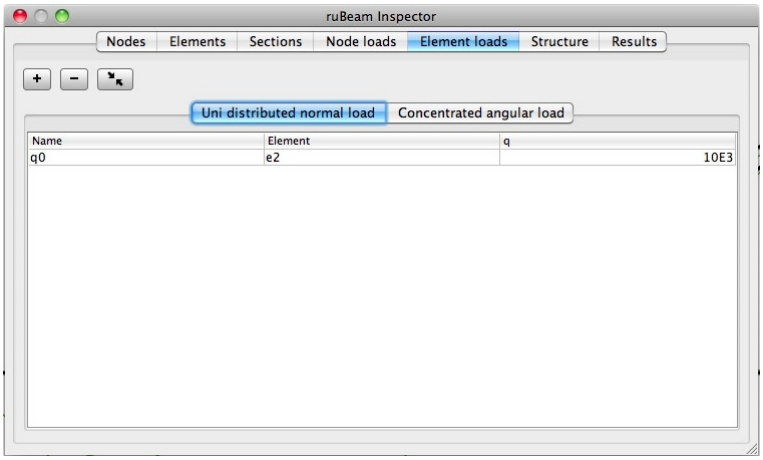
Inspector Tab Nodal loads

Nodal loads Tab displays all nodal loads in the workspace type wise in different subtabs, all properties can be set, [+] button adds a nodal load of the selected nodal load subtab, [–], [×] button removes the selected line in the table, [↘ ↖] button assigns selected nodal loads in the workspace to one node.



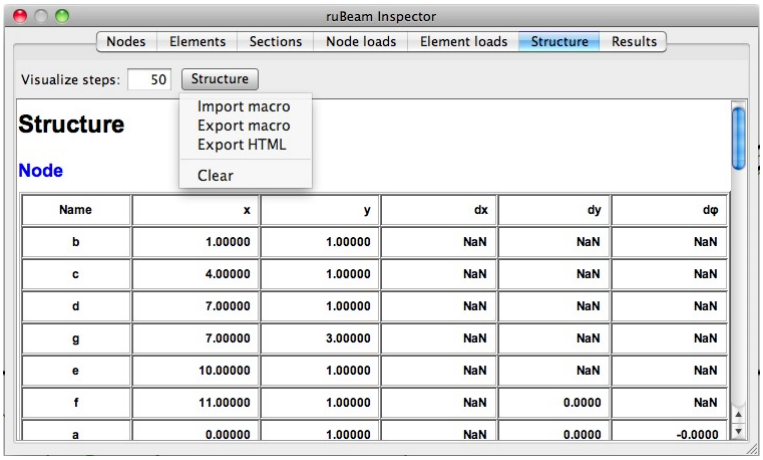
Inspector Tab Element loads

Element loads Tab displays all element loads in the workspace type wise in different subtabs, all properties can be set, [+] button adds a element load of the selected element load subtab, [–], [×] button removes the selected line in the table, [↘ ↖] button assigns selected element loads in the workspace to one element.



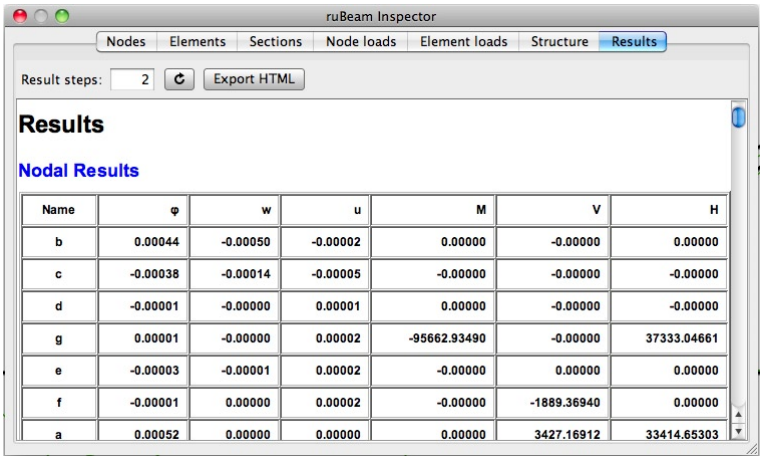
Inspector Tab Structure

Structure Tab displays the structure data, the visualize steps can be set, which do not influence in any way the structure data view, [**Structure**] button allows to import, export a ruBeam macro file and to save the structure tab content to a HTML file for later use, with Clear the structure will be cleared all ruBeam components will be deleted.



Inspector Tab Results

Result Tab displays the structure result data, the result steps can be set, [↻] button refreshes the view, [**Export HTML**] saves the result tab view to a HTML file for later use.



3.4.6 ruBeam File Management

ruBeam plugin allows to use two different file types

CADEMIA files normal CADEMIA files where both ruBeam and CADEMIA components are saved in one file, via *File* → *Save*

ruBeam macro files which only contain the ruBeam components and can be used with the ruBeam engine too, this files guarantee an upwards compatibility to newer ruBeam versions.

3.5 Useful CADEMIA Features in ruBeam Plugin

In this section a few useful CADEMIA features are introduced in order to emphasize how the ruBeam plugin makes use of existing CAD functionality. To learn more about the CADEMIA features described below, please refer to the CADEMIA Online Help (Menu *Help* → *Help contents*).

3.5.1 Construction

In order to add a node, an element or a load component to the structure usually points have to be constructed. The CADEMIA point construction processor allows four different methods that are to be selected via the context menu *Settings*:

Snap The point is constructed by combining the methods Pick, Grid, Digitze. The priority is first Pick (if a point is found), then Grid (if a grid point is found inside the pick box) and then Digitize (if nothing else is found).

Pick The point is constructed by picking/referring to existing points emphasized in the microscope.

Grid The point is constructed on the basis of the rectangular grid. It automatically moves to the nearest grid point. The grid can be redefined via menu *Window* → *Set grid*.

Digitize The point is digitized on the basis of the current mouse position.

3.5.2 Transform components

A major CAD functionality concerns the transformation of geometry. In CADEMIA the ruBeam plugin components can be transformed by applying different affine transforms:

Translate A ruBeam plugin component is translated about a vector.

Totate A ruBeam plugin component is rotated about an angle and a point.

Scale A ruBeam plugin component is scaled about a point.

Mirror A ruBeam plugin component is mirrored about a line.

3.5.3 Copy components

Another typical CAD feature is copying. Based on the existing copy functionality in CADEMIA nodes, elements and loads can be copied easily. It is distinguished between the different copy modes:

Copy translate A ruBeam plugin component is cloned first and then translated about a vector. This is the functionality most users would expect from copying.

Copy rotate A ruBeam plugin component is cloned first and then rotated about an angle and a point.

Copy scale A ruBeam plugin component is cloned first and then scaled about a point.

Copy mirror A ruBeam plugin component is cloned first and then mirrored about a line.

3.5.4 User Coordinate System

In CADEMIA the user coordinate system can be set by applying a rotation and a translation to the default coordinate system. The subsequently specified coordinates are related to the current user coordinate system. Using this feature a set of angular elements can be constructed very efficiently.

Appendix A

Examples

Here you find examples to show how to use ruBeam plugin and the ruBeam engine, as well they show some specialities when using ruBeam engine as subroutine in a Matlab program.
All files can be found on the ruBeam web page `http://rubeam.cademia.org` under the documentation section.

A.1 Production Hall

First example is a simple production hall constructed in CADEMIA with the help of the function *copy mirror*. After constructing the left side of the hall we copy mirrored it, cleaned up the duplicated nodes and added the missing elements in the middle. Enclosed the ruBeam

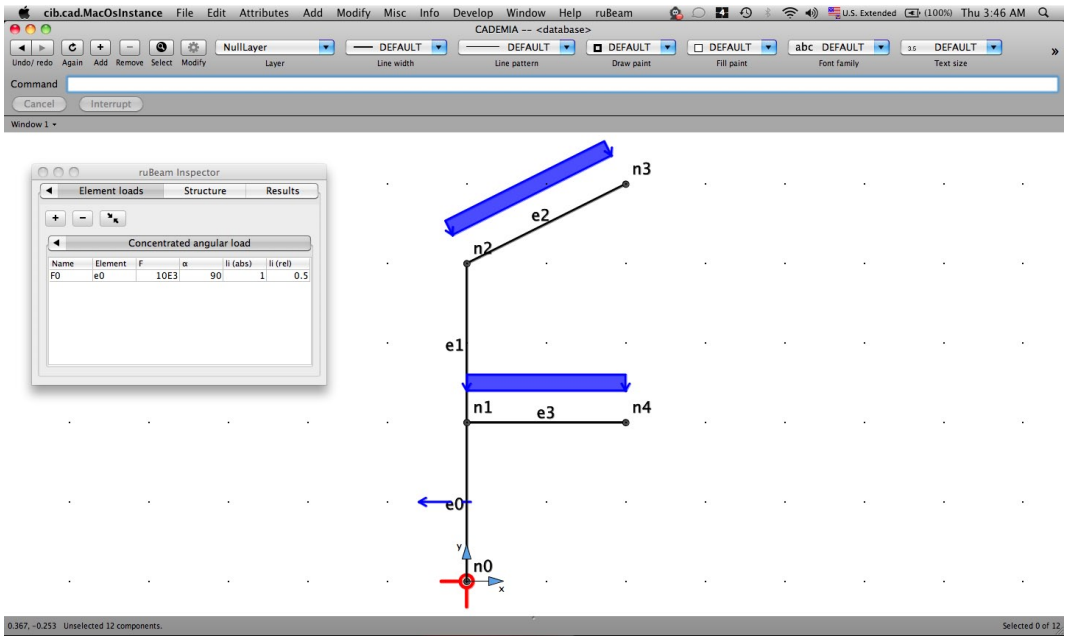


Figure A.1: First part of the production hall

file `phall.rb` for the suspension bridge is listed.

```
// ruBeam 1.0 beta, Copyright 2009 E. Bombasaro, Ch. Koch

//Nodes:
//-------------------------
addnode 0.0000000000 0.0000000000 n0;
addnode 0.0000000000 2.0000000000 n1;
addnode 0.0000000000 4.0000000000 n2;
addnode 2.0000000000 5.0000000000 n3;
addnode 2.0000000000 2.0000000000 n4;
addnode 4.0000000000 0.0000000000 n5;
addnode 4.0000000000 2.0000000000 n6;
addnode 4.0000000000 4.0000000000 n7;
addnode 2.0000000000 0.0000000000 n8;

//Elements:
//-------------------------
addelement n0 n1 e0;
addelement n1 n2 e1;
addelement n2 n3 e2;
addelement n1 n4 e3;
addelement n5 n6 e4;
addelement n6 n4 e5;
addelement n6 n7 e6;
addelement n7 n3 e7;
addelement n8 n4 e9;
setelementhingei e9 true;
setelementhingek e9 true;
addelement n4 n3 e10;
setelementhingei e10 true;
setelementhingek e10 true;

//Constraints:
//-------------------------
setnodeconsts n0 NaN 0.0000000000 0.0000000000;
setnodeconsts n5 NaN 0.0000000000 0.0000000000;
setnodeconsts n8 NaN 0.0000000000 NaN;

//Element Loads:
//-------------------------
addelementconload e0 10000.0 90.0 1.0 F0;
addelementunidisloadnor e2 10000.0 q0;
addelementunidisloadnor e3 10000.0 q1;
addelementconload e4 10000.0 -90.0 1.0 F1;
addelementunidisloadnor e5 10000.0 q5;
addelementunidisloadnor e7 10000.0 q4;

//Addendum:
//-------------------------
setvisualizesteps 50;
```

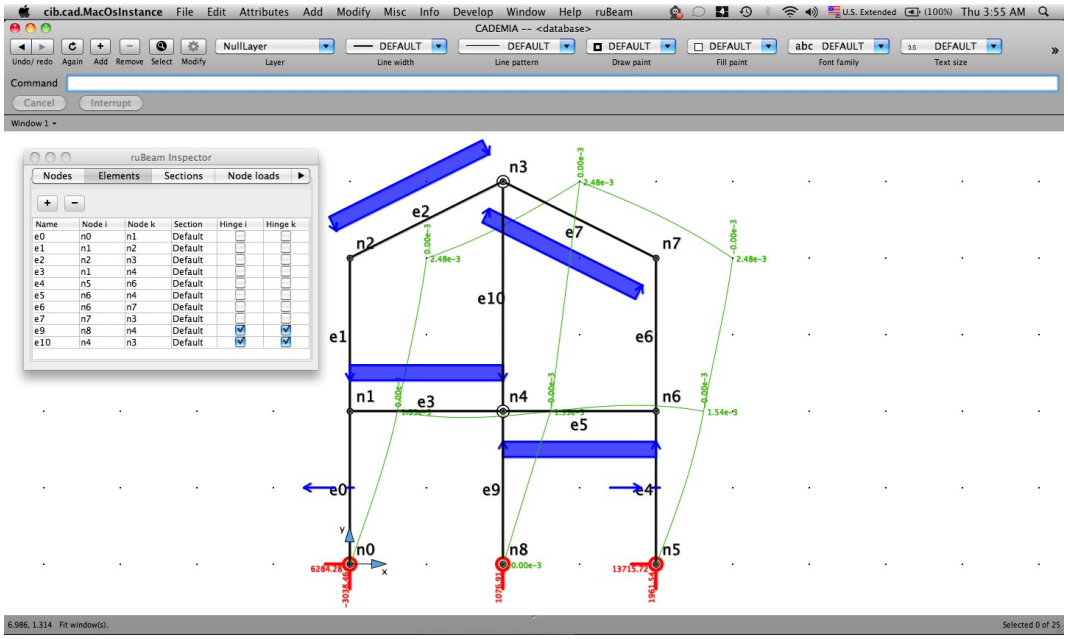


Figure A.2: The production hall after *copy mirror* command and cleaned nodes and loads

A.2 Suspension Bridge

Second example is a simple suspension bridge where the cables are modeld as thin beam elements.

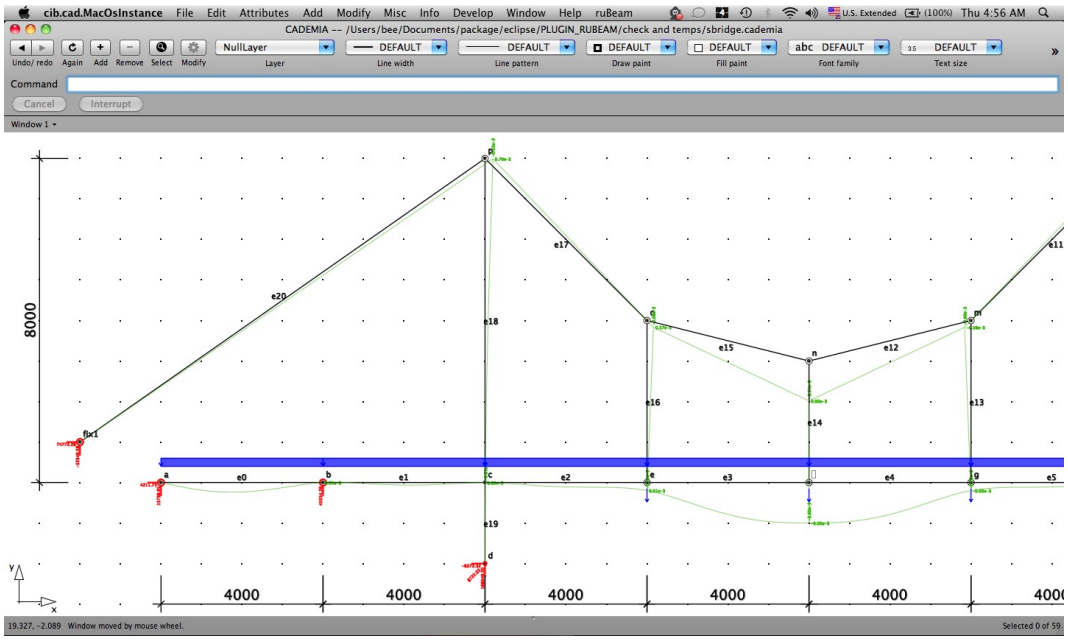


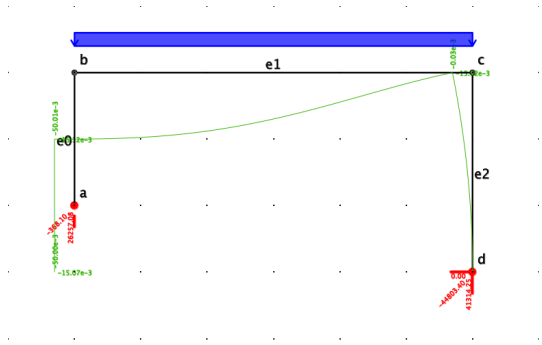
Figure A.3: Screenshot of CADEMIA while working on the suspension bridge project

The ruBeam file `sbridge.rb` for the suspension bridge is not listed here because its too long, you can download it from the ruBeam web page `http://rubeam.cademia.org`.

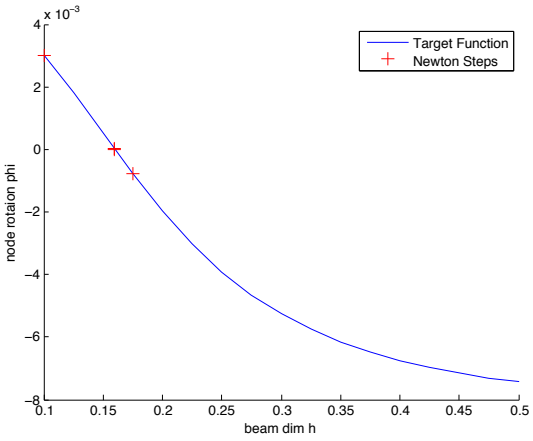
A.3 Advanced Examples using ruBeam engine

To show how smooth ruBeam engine is running, we minimize rotation angle φ of a simple frame Fig. A.4(a) by changing the height of the beam element. The frame is loaded with the beam weight plus traffic load and has a support imposed displacement of 0.05 m.

Fig. A.4(b) shows the functional dependency of the beam height h due to the rotation angel φ of the node b as well as the steps made by the optimizer. In this case Newton Raphson Method is used. Because in this example only the functionality of the ruBeam engine should be show no special care to the optimization routine was given.
As result we obtain the beam height of 0.16 m.



(a) Screenshot taken from CADEMIA optimized frame



(b) Matlab plot displaying the target function blue and the optimization steps red +

The *MATLAB* files can be found in the documentation section on the ruBeam web page `http://rubeam.cademia.org`. No special remarks are given here concerning the *MATLAB* files, you find some comments in every single file.

Index