ČERVENKA CONSULTING

Červenka Consulting s.r.o.
Na Hrebenkach 55
150 00 Prague
Czech Republic
Phone: +420 220 610 018
E-mail: cervenka@cervenka.cz
Web: http://www.cervenka.cz

ATENA 2024
Program Documentation

User's Manual for
ATENA-PRE

Written by

Jan Červenka, Tomáš Altman, Zdeněk Janda,
Jiří Rymeš, Michaela Herzfeldt,
Pavel Pálek and **Radomír Pukl**

Prague, October 2024

Acknowledgements:

Various ATENA software modules were developed with partial support of various national and international funding programs:

ATENA 2024 Engineering development has been supported by the Technology Agency of the Czech Republic under the project TM01000059 "Reducing material demands and enhancing structural capacity of multi-spiral reinforced concrete columns – advanced simulation and experimental validation" within the DELTA Programme.

ATENA3DPrint module has been developed within the project TF04000051, "digiCON2 – Simulation Service for Digital Concrete Production" funded by TA CR under the DELTA Programme.

ATENA TwinBridge module containing the support for TwinBridge platform integration and models for long time material degradation: chloride ingress, carbonation, reinforcement and bond corrosion model, alkali-silica reaction material degradation was developed under the project CK03000023 "Digital twin for increased reliability and sustainability of concrete bridges" within the DOPRAVA 2020+ Programme.

T A
Č R

CONTENTS

1 INTRODUCTION

1.1 About ATENA-PRE

ATENA-PRE, developed by Červenka Consulting s.r.o., serves as a direct preprocessor for **ATENA** kernel, a software used for non-linear simulations of structures. In **ATENA-PRE**, users can define a finite element model, including geometry, boundary conditions, and specific data required for **ATENA** analysis.

ATENA's primary objective is to accurately simulate the behavior of reinforced concrete structures. This is accomplished by incorporating non-linearities into the material models and employing iterative solution techniques. Through the evaluation of the dissipated fracture energy, **ATENA** can simulate the cracking of concrete when its tensile strength is exceeded. Similarly, the plasticity approach allows for the simulation of concrete crushing under high compression stresses. Moreover, **ATENA** possesses the capability to simulate the yielding and eventual rupture of reinforcement bars and prestressing tendons. Additionally, **ATENA** has the capability to evaluate the non-linear behavior of soils and simulate the behavior of other brittle or plastic materials.

This User's Manual serves as a guide for utilizing **ATENA-PRE**. Once the model definition is completed within **ATENA-PRE**, an input file for ATENA (with a file name extension of *.inp) is generated, and the simulation can be initiated. The solution and result postprocessing can be performed using either **ATENA Studio** or **ATENA Engineering.**

1.2 Workflow in ATENA-PRE

This document represents the User's manual for the newly developed **ATENA** preprocessing software called **ATENA-PRE**. The following text aims to explain all important aspects of the program and is structured in a way that follows a typical workflow of developing models for finite element method analysis. Following topics are explained:

- Select one of the problem types for **ATENA**,
- Create a geometrical model,
- Select material models, define parameters,
- Assign materials and element type to the geometry,
- Create Load cases and assign boundary conditions such as supports and load,
- Create loading history by defining in Tasks,
- Generate finite element mesh,
- Execute finite element analysis with **ATENA Studio** or **ATENA Console**.

Initial part of the manual is dedicated to the definition of basic analysis setting and creating geometry for the model. The geometry can be created directly in **ATENA-PRE**, standard export file types from other programs are supported for import. Also, **ATENA-PRE** supports the definition by scripting which is very helpful for various geometry shape that can be easily defined by functions along with direct definition of points etc. In the next step, materials are defined and assigned to the geometrical entities. The program includes several tools for automatic generation of material parameters, which help the user with correct setting of the non-linear material models. Then, loads and supports are assigned to load cases, which are further used to specify the loading history for the analysis. It should be noted that for non-linear analysis, the final stress state condition cannot be determined as a simple superposition of all load case, but it is affected by the loading history, mainly by the concrete cracking. Therefore, it is recommended to base the loading history in the analysis on the actual history of the analysed structural element or structure. Finally, mesh settings are defined and the finite element mesh is generated. The outcome of the preprocessing is the **ATENA** input file (**.inp*), which contains a series of commands for solution of the problem in **ATENA** kernel.

1.3 Program layout

The basic window of **ATENA-PRE** is shown in Fig. 1-1. In the center, a graphical view of the model is displayed. The viewing position, angle and zoom can be adjusted using the mouse commands as in standard CAD software. The viewing preferences allow user customisation of the settings, including object transparency, thickness, and colour, scrolling options for in and out zooming, or displaying different types of entities and the vectors that define them.

On top of the model view, the **Main menu** and the modelling and viewing toolbars are located. The **Main menu** allows access to all features of the preprocessor, including the modelling and analysis preferences, modelling, material definition. The toolbars icons suit for quick access to the most common features, such as creating and modifying geometrical entities and model viewing options.

By default, on the left side, these are an **Input data tree** window designed in the way that it guides the user through the different stage of the pre-processing from geometry definition until mesh settings and specification and loading history specification.

On the right side, **Layers window** is located. It is used to prescribe the geometrical entities to different groups (i.e., layers). This is useful for dividing model to separate parts, which can be then easily accessed separately. Furthermore, a given layer can be frozen to avoid any unintentional changes in it when working with the rest of the model or to excluding the layer from the model before executing the solution.

Below the workspace view, a window with four tabs is located. The **Data table** tab shows a tabulated summary of all inputs for an active category form the **Input data tree**. This is especially useful for reviewing the model. For instance, the coordinates and the lower entities are shown for geometrical objects or the most important material parameters are shown for the defined materials. When generating the mesh, it allows reviewing the output, message, and error logs. Important feature of **ATENA-PRE** is support of scripting. In the **Script history** tab, the user actions are recorded in a form of a script commands. This code can be copied, reviewed, and modified and then used for automatic re-generation of the model. Either a single line script command or block of commands can be executed.

On the very bottom of **ATENA-PRE,** a log of each program operation is written.

It is also recommended to go through the **ATENA 2024 CeSTaR-2** Example and validation manual [2] before starting with one's own modelling. For more detailed description of material input parameters, analysis execution, postprocessing the reader should consult the Example and Validation manual [2].

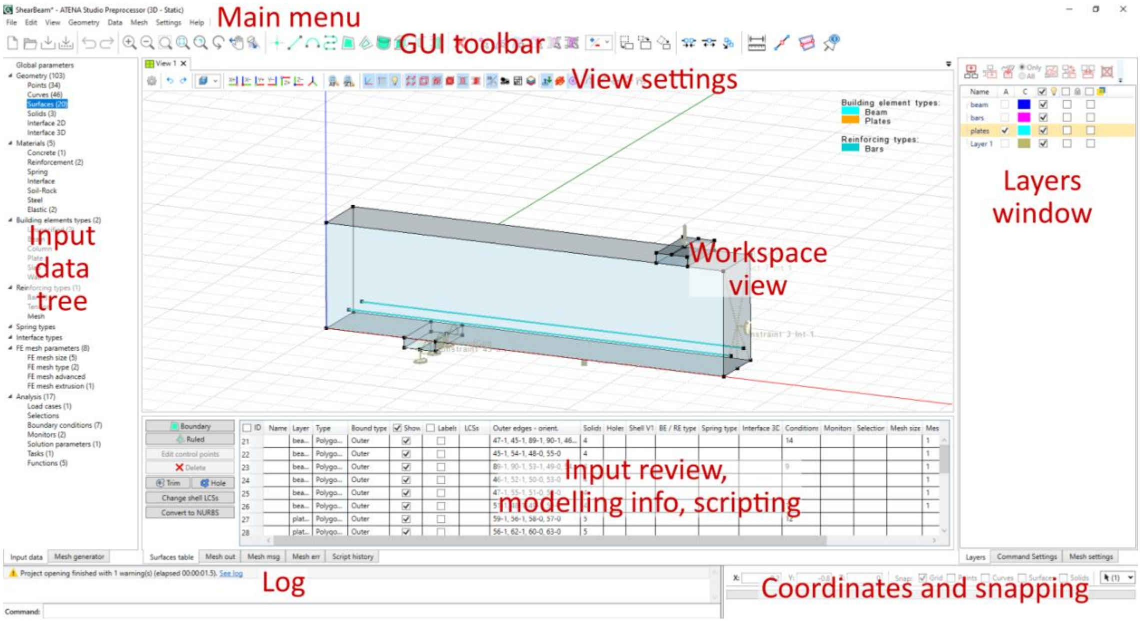
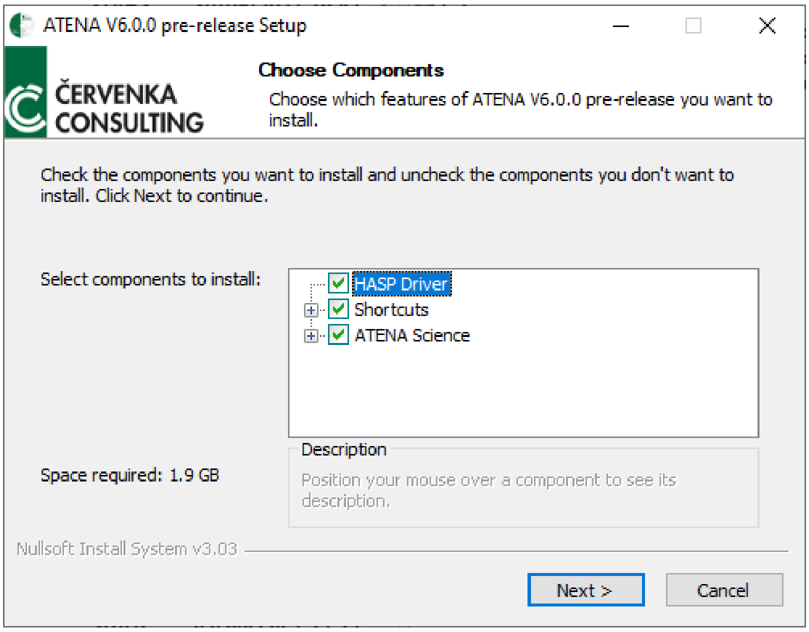


Fig. 1-1: ATENA-PRE interface.

1.4 Installing and Registration

For installation download the installer package available at company website www.cervenka.cz. In section **Download**, choose the **ATENA 2024** package and when the downloading is finished run the installer. **ATENA-PRE** is inherent part of the installer therefore no special selection is needed. The view of the installer is shown in Fig. 1-2.



ATENA V6.0.0 pre-release Setup

ČERVENKA CONSULTING

Choose Components

Choose which features of ATENA V6.0.0 pre-release you want to install.

Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

Select components to install:

- ☑ HASP Driver
- ☑ Shortcuts
- ☑ ATENA Science

Space required: 1.9 GB

Description

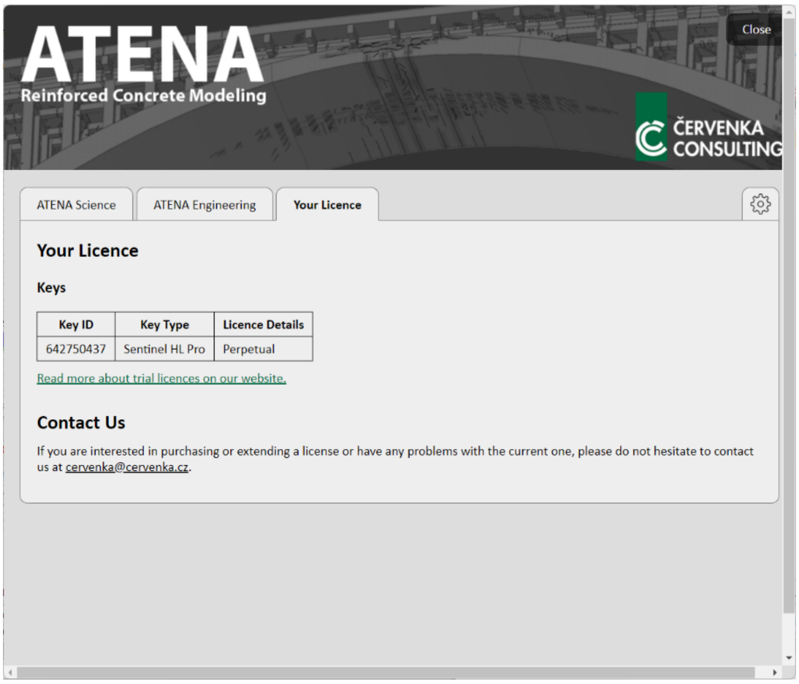
Position your mouse over a component to see its description.

Nullsoft Install System v3.03

Fig. 1-2: ATENA installer

After installation, the user is advised to access **ATENA** software package through **ATENA Center**, which serves as a basic starting point for accessing different features of the software.

Among running the software, it allows reviewing the current licence status (use the HASP HW Key or join network with NetHASP) and sending automatic request for a trial license. A screenshot summarising the information about an ATENA license is shown in Fig. 1-3. It is worth mentioning that licence-related questions are answered in **ATENA Troubleshooting manual** [3].



Your Licence

Keys

Key ID	Key Type	Licence Details
642750437	Sentinel HL Pro	Perpetual

Read more about trial licences on our website.

Contact Us

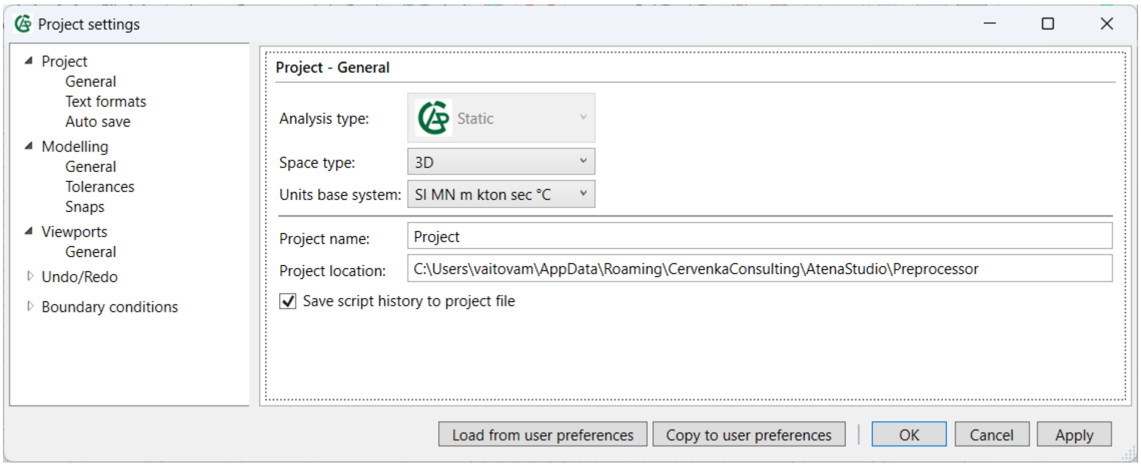
If you are interested in purchasing or extending a license or have any problems with the current one, please do not hesitate to contact us at cervenka@cervenka.cz.

Fig. 1-3: ATENA Center – Your licence.

2 STARTING A NEW PROJECT

When the user creates the new model in **ATENA-PRE**, the type of simulation and general analysis information must be defined. To set the problem type, in the main menu select **Settings | Project settings**. In the pop-up window user selects one of the **ATENA** analysis types (Fig. 2-1):

- **Space type** 3D, 2D, axisymmetric
- **Unit base system** SI, Imperial
- **Project name**
- **Project location**



Analysis type: Static

Space type: 3D

Units base system: SI MN m kton sec °C

Project name: Project

Project location: C:\Users\vaitovam\AppData\Roaming\CervenkaConsulting\AtenaStudio\Preprocessor

☑ Save script history to project file

Fig. 2-1 Project settings menu.

At this moment, the user defines whether a model for a full 3D analysis will be developed or if a 2D or axisymmetric approximation will be used. Furthermore, the base unit system is defined at this moment. The default **ATENA** unit system (i.e., meter, radian, megaton, meganewton, seconds, °C) can be changed to the imperial unit system. It should be noted that these units will be used for the definition of the geometry and finally for generating the input file; however, the material definition and boundary conditions prescription can be done in a separate unit system and **ATENA-PRE** automatically converts the inputted value into the basic unit system and generates input file with consistent data in terms of geometry, material parameters, boundary condition, and, eventually, time-dependent variables.

The remaining general information is the project name and the location for storing the **ATENA-PRE** project file. The option **Save script history** allows saving the history of user actions during development. If activated, each user operation is saved as a command, which can be later executed through the script execution line individually or as a block of commands. This is especially useful for automatic scripting (see chapter *10 Examples of scripting* in ATENA for further details).

The individual customization of the working space can be done in the **Viewports | General** category. This includes the view style and transparency of the geometrical entities, including the point and line widths and colors. The visibility of several objects of the workspace, such as the legend, coordinate system cross, boundary conditions labelling, etc. can be adjusted. The font size can be increased or decreased as well. Furthermore, in the **Viewports | Grid** tab, the coordinate system of the workspace can be modified. The most important viewing settings are also directly available through the view toolbar.

Among other customizable features, in the **Project | Text formats** tab, user can modify the decimal value format and its displayed precision for numerical inputs.

Under the **Modelling** category, several settings regarding the navigation in the workspace are listed. Mainly, zooming in and out using the scrolling wheel can be adjusted and snapping options to the grid and geometrical entities are available.

For the autosave option, the user defines the saving interval and the maximum number of files to be stored. Modifying these settings might be useful when preprocessing very large models.

It should be noted that the user preferences can be exported from an existing project and then used in a new **ATENA-PRE** model for better customizing of the program interface and modelling preferences.

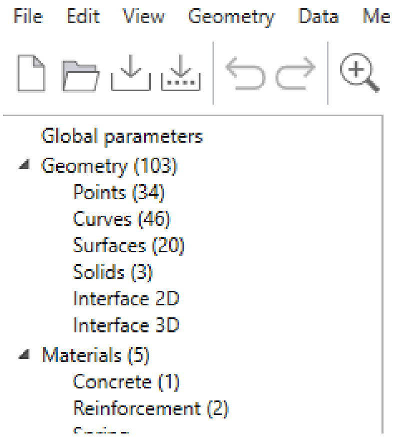
3 CREATING GEOMETRY

Generally, there are three methods how to define a model geometry for a new model in **ATENA-PRE**. These methods are as follows:

- Direct modelling using GUI,
- Running series of command for geometry definition,
- Importing geometry from other software.

Additionally, the options listed above can be combined during development of a single model.

The geometrical entities in **ATENA-PRE** are points, lines/curves, surfaces, solids, and 2D and 3D interfaces. The standard modelling method through the GUI is using the so-called bottom-to-top modelling approach, where the definition starts from creation points. The points are then connected to construct lines, arc segments, or NURBS curves, which are further used for surface definition. Finally, based on the surfaces, solids are constructed. This workflow is also suggested by the input data tree shown in Fig. 3-1. In **ATENA-PRE**, a unique name can be assigned to each entity, which can be further use to address it.



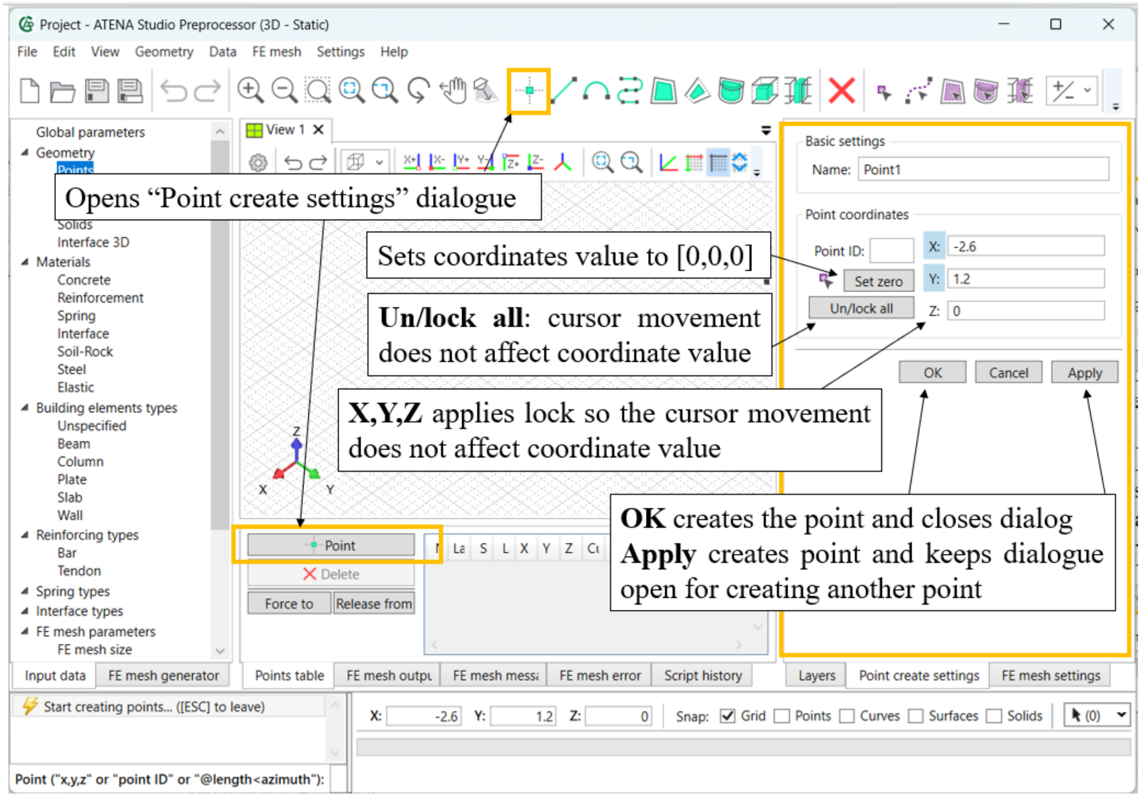
Global parameters
Geometry (103)
Points (34)
Curves (46)
Surfaces (20)
Solids (3)
Interface 2D
Interface 3D
Materials (5)
Concrete (1)
Reinforcement (2)

Fig. 3-1 Input data tree - Geometry.

View of geometry in the ATENA can be rotated by holding Shift key + Left mouse click while the cursor changes into symbol of rotating view. And the pan of the view is done similarly by holding Shift key + Right mouse click.

3.1 Create new entity

A new point can be created either by selecting **Create points** button from the toolbar or by clicking the **Point** button once the points category is activated in the input data tree. Both options are illustrated in Fig. 3-2. On top of that, points and other entities can be created also through the main menu. Its coordinates can be defined either by clicking into the workspace or by editing the coordinates in **Point create settings** dialog. Higher entities are created in a similar manner. The toolbar for creating new entities is shown in Fig. 3-3. Surfaces and solids can be defined either by a direct selection of the lines and surfaces, respectively, or through direct input of their numbers. After each creating a new entity, it is listed in the summary table, which summaries its properties.



Opens "Point create settings" dialogue

Sets coordinates value to [0,0,0]

Un/lock all: cursor movement does not affect coordinate value

X,Y,Z applies lock so the cursor movement does not affect coordinate value

OK creates the point and closes dialog **Apply** creates point and keeps dialogue open for creating another point

Name: Point1

X: -2.6 Y: 1.2 Z: 0

Start creating points... ([ESC] to leave)

Point ("x,y,z" or "point ID" or "@length<azimuth"):

X: -2.6 Y: 1.2 Z: 0 Snap: ☑ Grid ☐ Points ☐ Curves ☐ Surfaces ☐ Solids (0)

Fig. 3-2 Creating point entity.

While creating points, the coordinates are automatically updated as the user moves the cursor in the workspace window. By clicking the **Set zero** button, the initial (i.e., zero value) coordinates are restored. Furthermore, by clicking on the coordinate label **X,Y,Z**, the user can lock the current coordinate value and thus prevent it from changing by cursor movement however the coordinated is still editable in dialogue. All coordinates can be locked together by using button **Un/lock all**. By doing so, points can be created in a given plane or along a given line. In Fig. 3-2, values of x and y-coordinates are locked on -2.6 and 1.2 meters, respectively, and only z-coordinate value changes as the user moves the cursor in the workspace window.

Deleting of the existing geometrical entity can be done either through the **Delete** button in the tabulated entities list or using its dedicated button in the toolbar shown in Fig. 3-3.

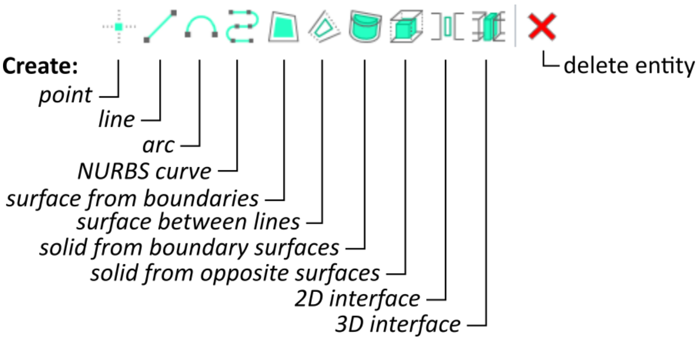


Fig. 3-3 Toolbar for creating geometrical entities.

To select an existing geometrical entity through the GUI, the designed button from the selection toolbar needs to be chosen as shown in Fig. 3-4.

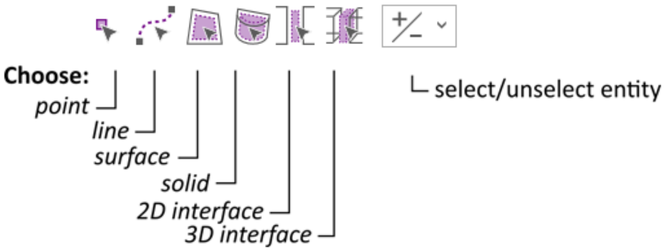
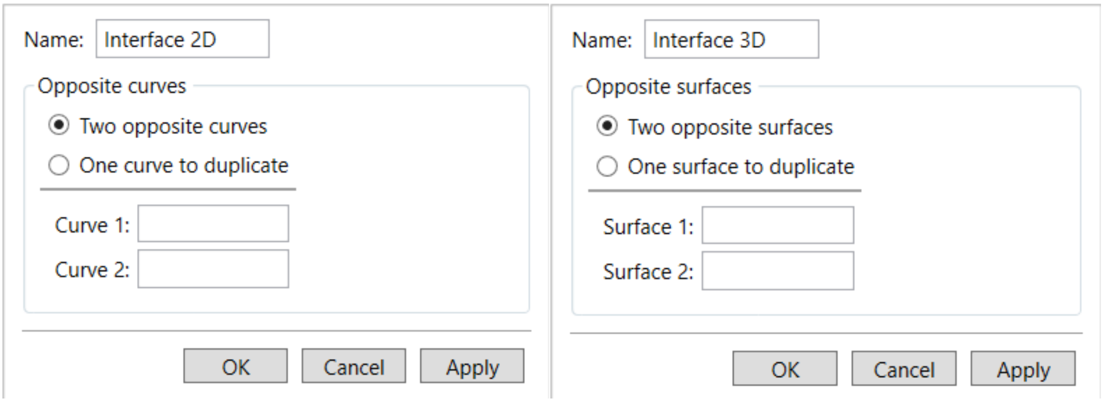


Fig. 3-4 Quick selecting/unselecting bar geometrical entities.

Create interface 2D/3D is tool that makes geometry with specific requirements for **Interface** material. Interface 2D geometry is made from two topologically identical curves that cannot share any point. Moreover, the normal of the curves have to have same direction and the sequence of curves (curve1 and curve 2 of interface) has to follow the normal direction. Similarly, the geometry must be created in 3D without shared entities within surface geometry. The normal of surfaces have to point in same direction and the surface 1 and surface to have to follow the normal orientation. Please check the Fig. 3-5. User can create Interface geometry from one curve/surface (option **One curve to duplicate/One surface to duplicate**) and the normal direction and correct the sequence of entities is created automatically.


Name: Interface 2D

Opposite curves
● Two opposite curves
○ One curve to duplicate

Curve 1:
Curve 2:

Name: Interface 3D

Opposite surfaces
● Two opposite surfaces
○ One surface to duplicate

Surface 1:
Surface 2:

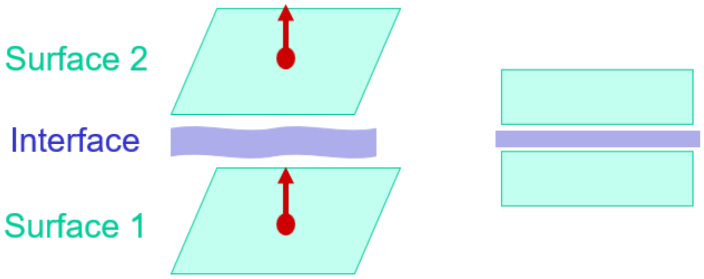


Fig. 3-5 Interface geometry orientation and sequence.

3.2 Modify entity

Standard tools are available for the modification of the existing geometry. This includes tools like copy, translation, rotation, or scaling, the toolbar of such operations is shown in Fig. 3-6. When modifying the geometry, **ATENA-PRE** offers a preview of the operation before the user confirms it, where selected geometry is highlighted in red, and preview of created geometry is in black color, see Fig. 3-7. Once the user confirms it by pressing Enter or clicking Apply/OK, the created points turn dark blue, curves lighter blue, surfaces purple and solids turquoise, see Fig. 3-8.

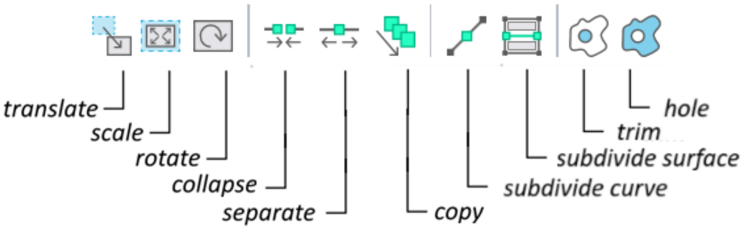
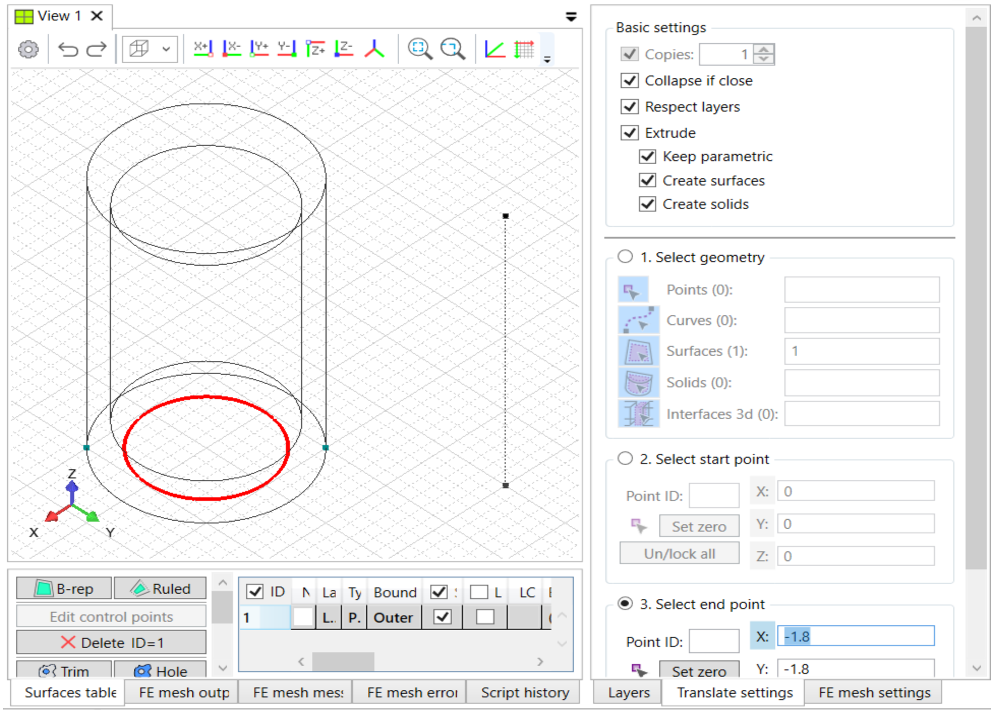


Fig. 3-6 Modify geometry toolbar.



☑ ID	N	La	Ty	Bound	☑	□ L	LC
1	□	L.	P.	Outer	☑	□	

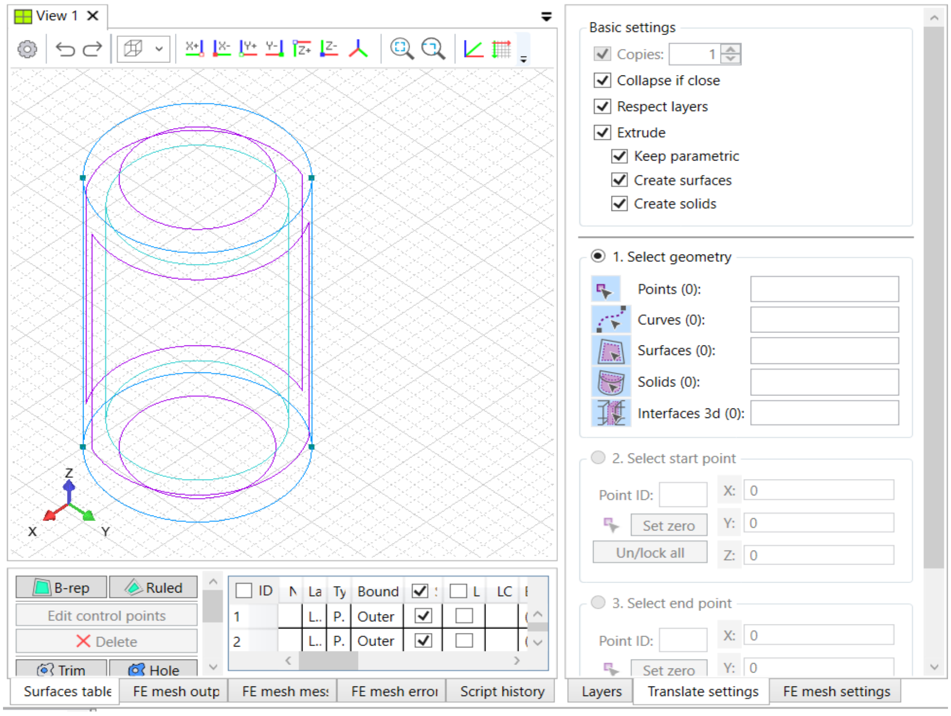
Basic settings
☑ Copies: 1
☑ Collapse if close
☑ Respect layers
☑ Extrude
☑ Keep parametric
☑ Create surfaces
☑ Create solids

○ 1. Select geometry
Points (0):
Curves (0):
Surfaces (1): 1
Solids (0):
Interfaces 3d (0):

○ 2. Select start point
Point ID:
X: 0
Set zero
Y: 0
Un/lock all
Z: 0

● 3. Select end point
Point ID:
X: -1.8
Set zero
Y: -1.8

Fig. 3-7 Preview of geometry created by extrusion of a surface.



☐ ID	N	La	Ty	Bound	☑	☐ L	LC
1		L..	P.	Outer	☑	☐	
2		L..	P.	Outer	☑	☐	

Basic settings

☑ Copies: 1

☑ Collapse if close

☑ Respect layers

☑ Extrude

☑ Keep parametric

☑ Create surfaces

☑ Create solids

● 1. Select geometry

Points (0):

Curves (0):

Surfaces (0):

Solids (0):

Interfaces 3d (0):

○ 2. Select start point

Point ID: X: 0

Set zero Y: 0

Un/lock all Z: 0

○ 3. Select end point

Point ID: X: 0

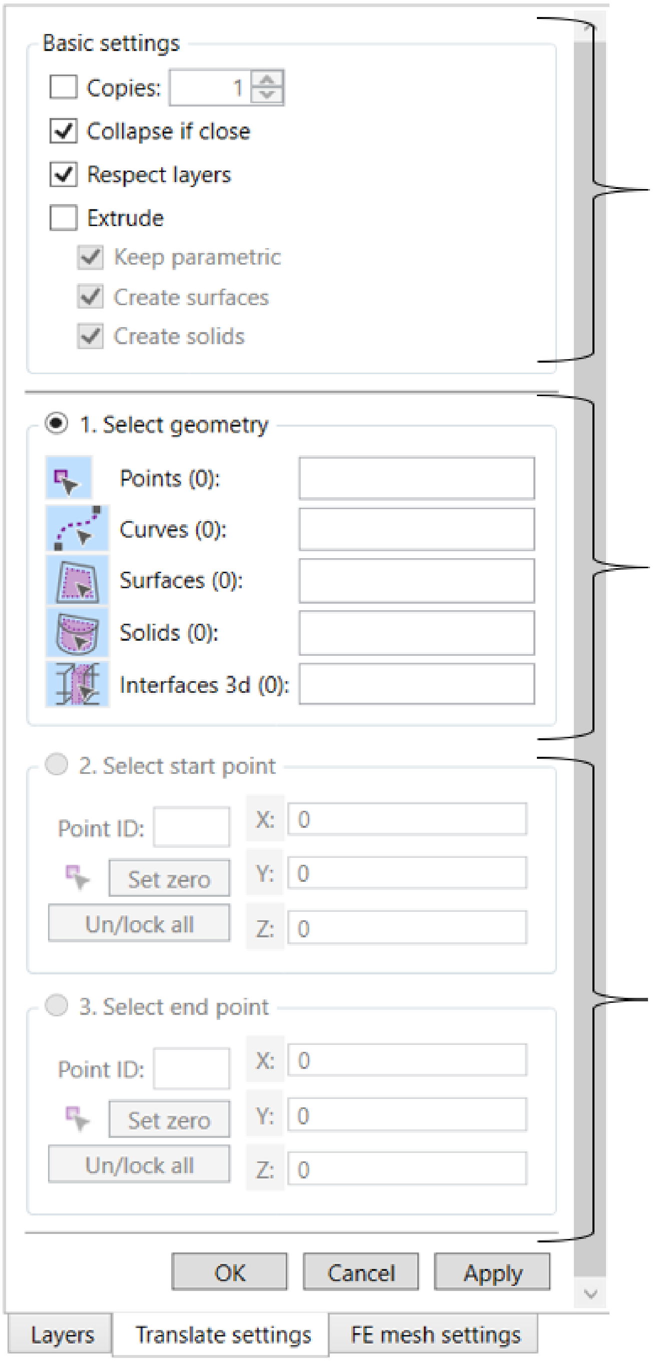
Set zero Y: 0

Fig. 3-8 Extruded surface into solid after confirmation execution of command.

Translate tool is used for operation Translate (only moves the selected geometry), Copy (creates copy/copies of selected geometry) and Extrude (selected entities are extruded along the defined direction, surfaces and solids can be enabled/disabled). User specifies which operation wants to execute, then selects the entity, and finally defines vector by start and end point that determines the operation. Fig. 3-9 shows Translate geometry settings. “Select entity” might be specified directly by ID numbers or selected in the geometry view by clicking on them, to select more entities in a row Ctrl key needs to be pressed throughout selection.

Scale tool enlarges or shrinks original object, operation Copy is available as well, then the original object is kept along with the scaled one. The scale is defined by scale factor in direction X,Y and Z, the lock sign keeps same factor in all directions, and by scale center which point of reference for scaling factor, Fig. 3-10 depicts the dialogue.

Rotate tool causes circular movement of geometry around center, Copy keeps the original object in its place and Revolve extrudes the geometry, Fig. 3-11.



Basic settings

□ Copies: 1

☑ Collapse if close

☑ Respect layers

□ Extrude

☑ Keep parametric

☑ Create surfaces

☑ Create solids

● 1. Select geometry

Points (0):

Curves (0):

Surfaces (0):

Solids (0):

Interfaces 3d (0):

○ 2. Select start point

Point ID: X: 0

Y: 0

Z: 0

○ 3. Select end point

Point ID: X: 0

Y: 0

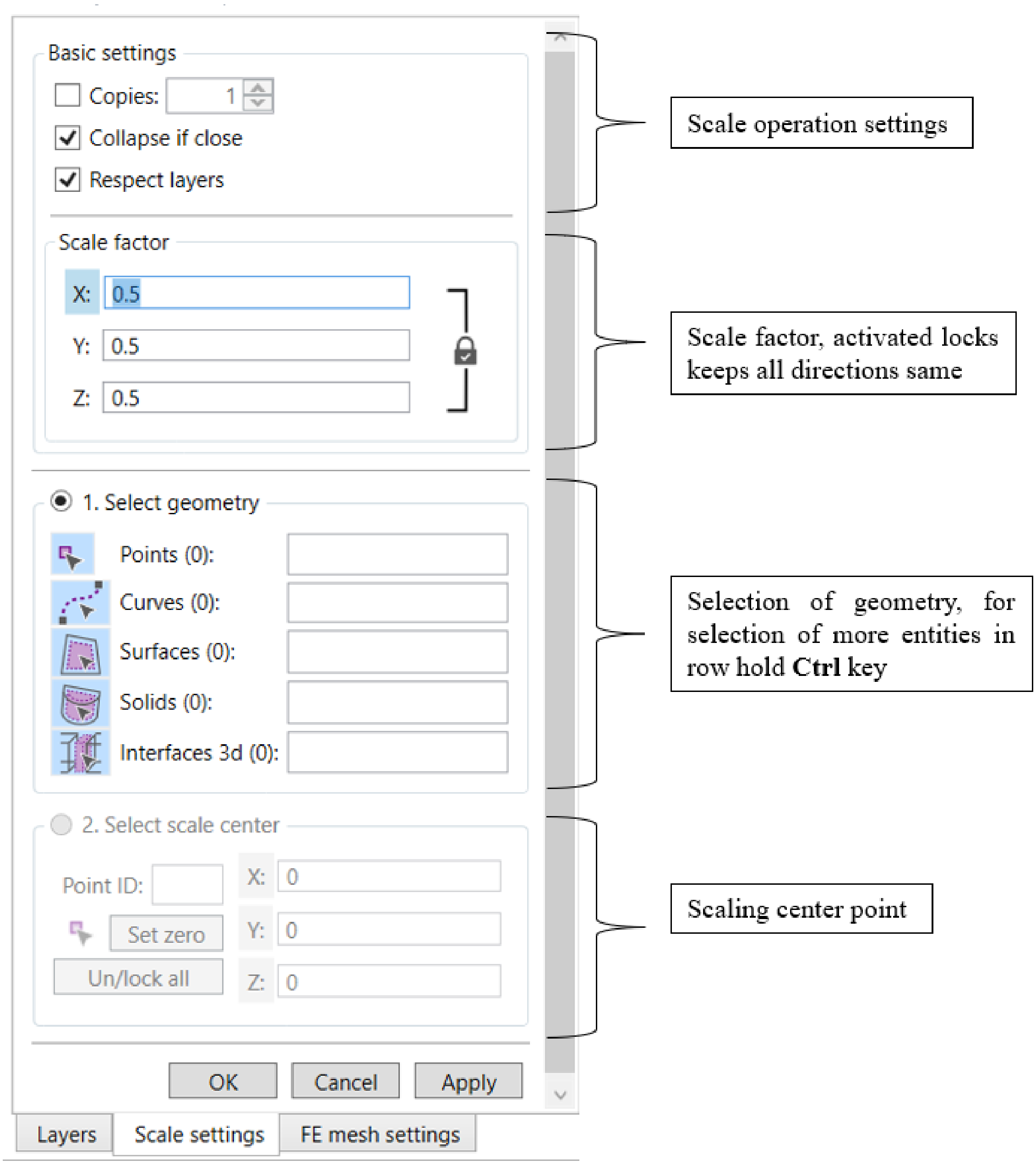
Z: 0

Operation settings Copy/Extrusion

Selection of geometry, for selection of more entities in row hold **Ctrl** key

Selection of points (vector) defining operation

Fig. 3-9 Translate geometry dialogue settings.



Basic settings

□ Copies: 1

☑ Collapse if close

☑ Respect layers

Scale factor

X: 0.5

Y: 0.5

Z: 0.5

● 1. Select geometry

Points (0):

Curves (0):

Surfaces (0):

Solids (0):

Interfaces 3d (0):

○ 2. Select scale center

Point ID:

X: 0

Y: 0

Z: 0

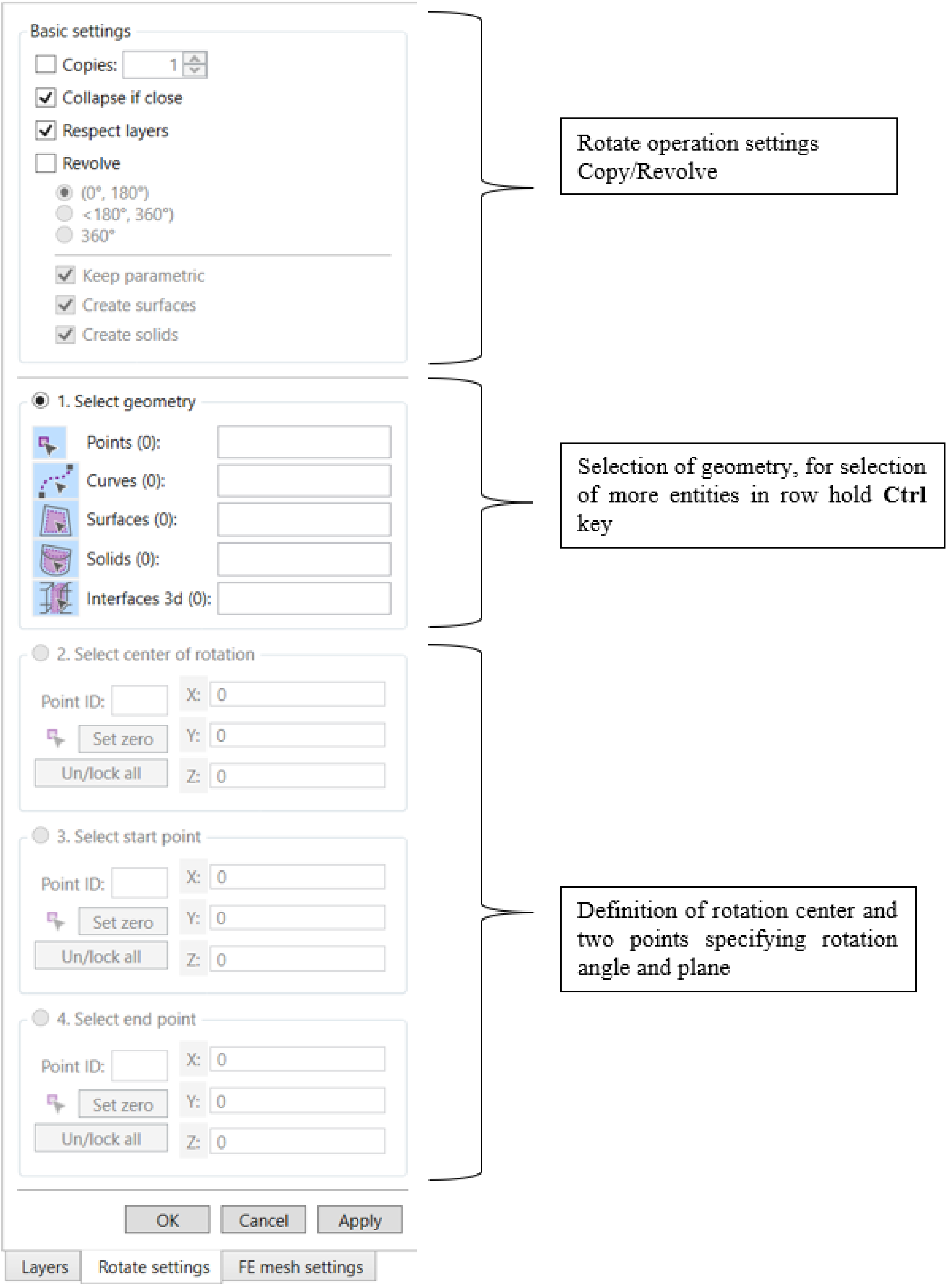
Scale operation settings

Scale factor, activated locks keeps all directions same

Selection of geometry, for selection of more entities in row hold **Ctrl** key

Scaling center point

Fig. 3-10 Scale geometry dialogue settings.


Basic settings

☐ Copies: 1

☑ Collapse if close

☑ Respect layers

☐ Revolve

● (0°, 180°)

○ <180°, 360°)

○ 360°

☑ Keep parametric

☑ Create surfaces

☑ Create solids

● 1. Select geometry

Points (0):

Curves (0):

Surfaces (0):

Solids (0):

Interfaces 3d (0):

○ 2. Select center of rotation

Point ID: X: 0

Y: 0

Z: 0

○ 3. Select start point

Point ID: X: 0

Y: 0

Z: 0

○ 4. Select end point

Point ID: X: 0

Y: 0

Z: 0

Rotate operation settings Copy/Revolve

Selection of geometry, for selection of more entities in row hold **Ctrl** key

Definition of rotation center and two points specifying rotation angle and plane

Fig. 3-11 Rotate geometry dialogue settings.

Collapse, separate, duplicate geometry are used for entities having the same coordinates can be collapsed or overlapping entities can be created by separating geometry. Duplicate creates duplicate of geometry. It can be applied on points, curves, surface, solids or interface.

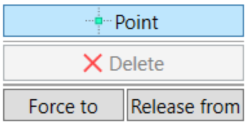
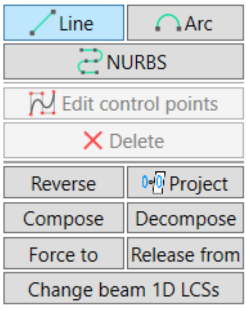
Subdivide curve/surface divides the curves or surfaces into several smaller parts. The curve division is defined by one of following method: number of divisions, near point, parameter, relative length, step length, length. For surfaces the division is defined by number of divisions together with two edge curves.

Trim nonplanar surfaces can be trimmed along selected curves.

Hole can be created within the surface, the definition of hole is either by its edge curves or by surface.

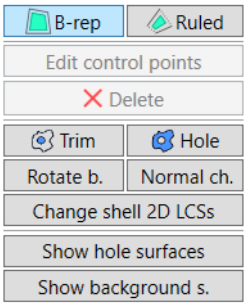
Beside the main toolbar buttons for modifying the geometry, the extra buttons are included in Table area for all entities: points, curves, surfaces, solids and interface. Each button with its use are listed in following table Tab. 3-1.

Tab. 3-1 Geometry tables – modification buttons

Button	Description
POINTS  	
Force to	Creates FE mesh node in point that is forced to curve or surface, tolerance is set by User defined epsilon in FE mesh generator (1E-05 is default)
Release from	Cancelation of command “Force to”, the point is released from curve or surface
CURVES  Line Arc NURBS Edit control points Delete Reverse Project Compose Decompose Force to Release from Change beam 1D LCSs	
Edit control points	Applies on NURBS curves, selected NURBS can be adjusted via its control point by either moving them or changing their weight
Reverse	Changes the orientation of curve normal
Compose	Creates polycurve from several separate curves
Decompose	Reverts the “Compose” command, the polycurve is decomposed to original separate curves
Force to	Creates FE mesh nodes on curve that is forced to surface, tolerance is set by User defined epsilon in FE

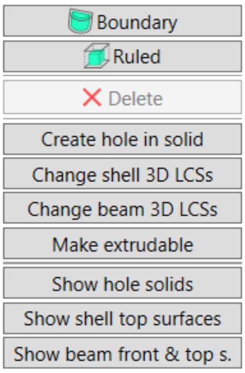
	mesh generator (1E-05 is default)
Release from	Cancelation of command “Force to”, the curve is released from surface
Change beam 1D LCSs	When static model Beam 1D is assigned, the local coordinate system (LCS) can be adjusted with this tool, V3 can be set, V1 can be reversed

SURFACES

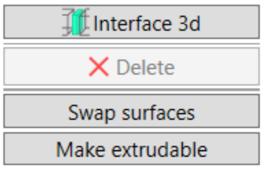


Edit control points	Applies on NURBS surfaces that are imported with control points
Rotate b.	Rotates boundary curves, user can specify number of rotations and the direction (by using positive or negative number of rotations)
Normal ch.	Changes orientation of surface normal to opposite direction
Change shell 2D LCSs	When static model Shell 2D is assigned, the local coordinate system (LCS) can be adjusted with this tool, V1 can be set, normal can be reversed
Show hole surfaces	Shows hole surfaces and dims rest of the geometry
Show background s.	Shows original surfaces that have been trimmed

SOLIDS



Create hole in solid	Hole within solid can be created by selecting boundary surfaces of inner object, or solid. The boundary of inner object cannot lie on the boundary of main solid
Change shell 3D LCSs	When static model Shell 3D is assigned, the local coordinate system (LCS) can be adjusted with this tool, V1 can be set, and V3 or top surface is defined
Change beam 3D LCSs	When static model Beam 3D is assigned, the local coordinate system (LCS) can be adjusted with this tool, V1 and V3 can be set, or front surface and top

	surface is defined
Make extrudable	Rotates curves defining boundary surface, changes orientation their orientation if necessary and also surface normal in order to make solid extrudable, then semi-structured FE mesh can be applied
Show hole solids	
Show shell top surfaces	Shows shell 3D top surface, only if it is manually assigned (default LCS is defined by V1 and V3)
Show beam front & top s.	Shows beam 3D front and top surface, only if it is manually assigned (default LCS is defined by V1 and V3)
INTERFACE 2D/3D 	
Swap curves (2D) / surfaces (3D)	Changes the sequence of curves/surfaces defining Interface 2D/3D, more details find in 3.1 Create Interface
Make extrudable	Rotates curves defining boundary surface, changes orientation their orientation if necessary and also surface normal in order to make solid extrudable, then semi-structured FE mesh can be applied

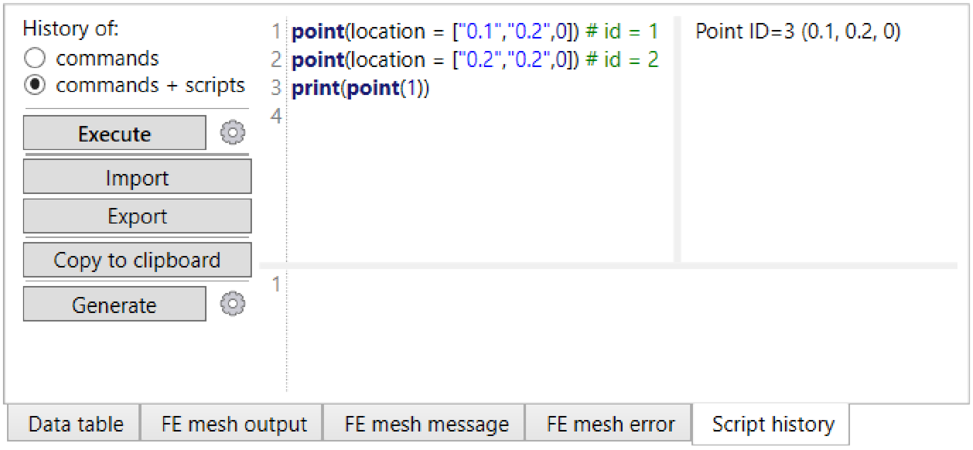
3.3 ATENA Scripting and Python

ATENA preprocessor supports scripting using ATENA-specific commands which are listed in Annex A – ATENA commands. ATENA script is being written automatically when the user uses GUI for defining the model and analysis. User can find the script in window **Script history** where the script is shown and also new script commands can be executed with corresponding button, Fig. 3-12. Beside that Scripting history can be exported and stored into specific file and then imported into another ATENA project, or simply the script can be copied to clipboard and pasted into command line in new project.

ATENA scripting supports analytical expressions which enables convenient definition of geometrical entities of complex shapes.

Another great feature in scripting section is implementing of **Python** (specifically Iron Python at the moment) allowing users to create user-defined functions. The Python script is visible only when user switches to **History of: commands + script**, this option enables the output window next to command history. **History of: commands** shows only ATENA commands, see in Fig. 3-12.

Examples of ATENA commands and also Python user-defined function are shown in chapter 10 Examples of scripting in ATENA preprocessor.



```
point(location = ["0.1","0.2",0]) # id = 1
point(location = ["0.2","0.2",0]) # id = 2
print(point(1))
```

```
Point ID=3 (0.1, 0.2, 0)
```

Fig. 3-12 Script history.

3.4 Import geometry

On top of that, **ATENA-PRE** supports import of the geometry from CAD systems in IFC or IGES format. Also, importing existing projects from **ATENA Engineering 2D** and **ATENA Engineering 3D** is supported, as well as import of **ATENA Science** models developed in **GiD**. The import functionality is activated through the menu item **File | Import**, see Fig. 3-13.

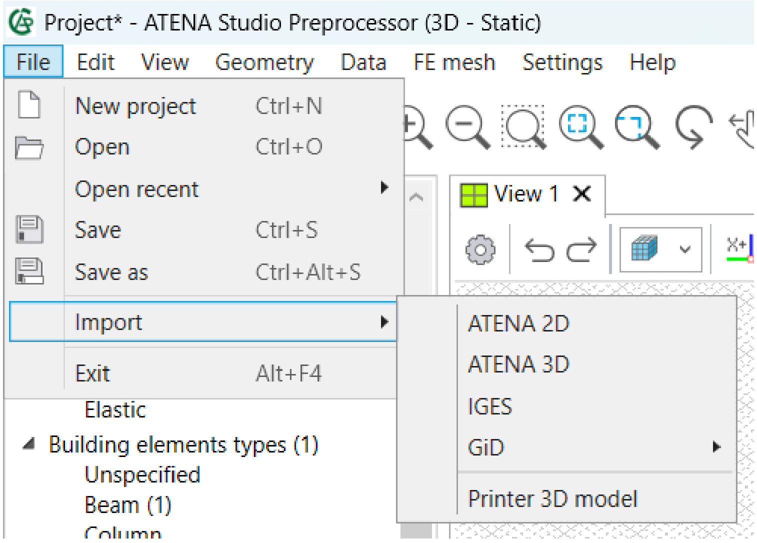


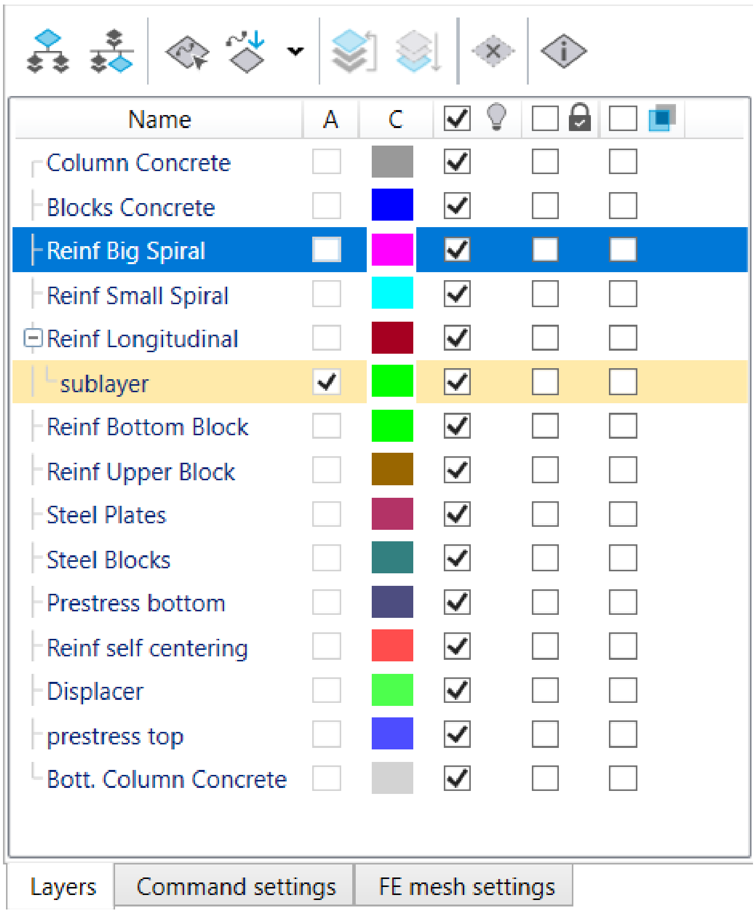
Fig. 3-13 Import of data into ATENA-PRE.

3.5 Layers

Parts of geometry can be sorted into layers. Generally, it can be extremely helpful to have the model divided into different layers, their visibility assists users especially while assigning various properties such as materials via building element types, FE mesh parameters, boundary conditions etc.

Each layer has set of properties: color (C), visibility (💡), freeze (🔒) and transparency (■). Only one layer can be active (A), newly created geometry is placed in this layer. Active layer is highlighted in orange color on the other hand selected layer is highlighted in blue color, see Fig. 3-14.

User can create new layers or sub-layers (sub-layer can be turned into layer by button). Geometry that has been already created can be moved into another layer, firstly the layer has to be selected, then the geometry (using geometry selection toolbar) and finally button Add selected geometry to selected layer finishes the request. Geometry belonging to layer can be selected at once by button . More layers can be joint together by button . Finally, there is a button for deleting the layer and information what entities are placed in the layer .



Name	A	C	☑	☐	☐
Column Concrete	☐	■	☑	☐	☐
Blocks Concrete	☐	■	☑	☐	☐
Reinf Big Spiral	☐	■	☑	☐	☐
Reinf Small Spiral	☐	■	☑	☐	☐
Reinf Longitudinal	☐	■	☑	☐	☐
sublayer	✓	■	☑	☐	☐
Reinf Bottom Block	☐	■	☑	☐	☐
Reinf Upper Block	☐	■	☑	☐	☐
Steel Plates	☐	■	☑	☐	☐
Steel Blocks	☐	■	☑	☐	☐
Prestress bottom	☐	■	☑	☐	☐
Reinf self centering	☐	■	☑	☐	☐
Displacer	☐	■	☑	☐	☐
prestress top	☐	■	☑	☐	☐
Bott. Column Concrete	☐	■	☑	☐	☐

Fig. 3-14 Import of data into ATENA-PRE.

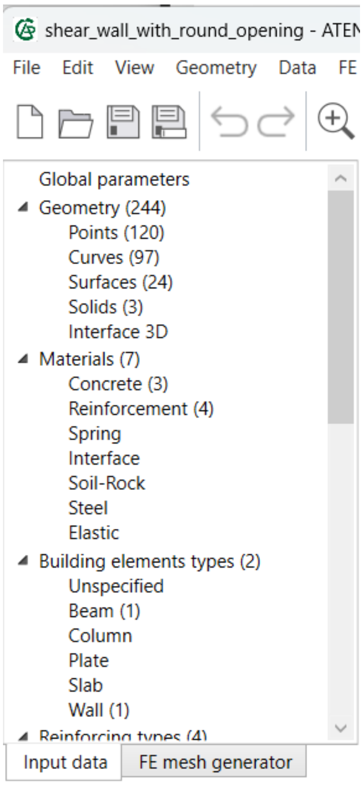
4 MATERIALS

ATENA analyses offer various material models, where the user can define parameters based on standards or directly from experimental results on test samples.

Within the Input data tree (Fig. 4-1) the **Materials** branch offers the following material classes:

- Concrete,
- Reinforcement,
- Interface,
- Spring,
- Soil-Rock,
- Steel,
- Elastic.

A comprehensive summary of the materials is given in Tab. 4-1 including a brief description of each of them.


shear_wall_with_round_opening - ATEN
File Edit View Geometry Data FE
Input data
FE mesh generator

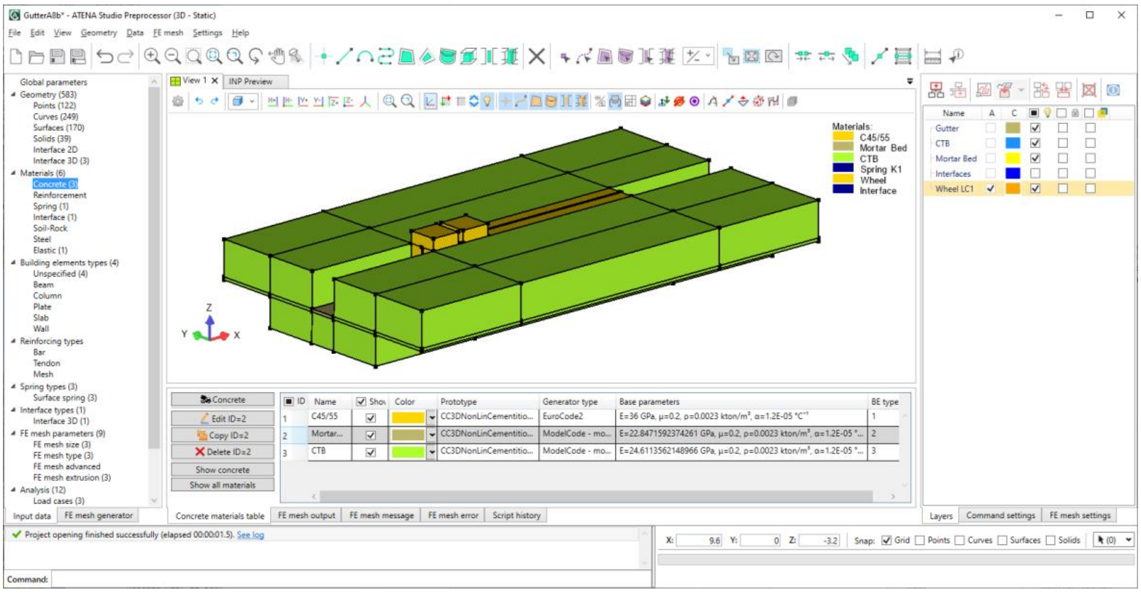
Global parameters
- Geometry (244)
  - Points (120)
  - Curves (97)
  - Surfaces (24)
  - Solids (3)
  - Interface 3D
- Materials (7)
  - Concrete (3)
  - Reinforcement (4)
  - Spring
  - Interface
  - Soil-Rock
  - Steel
  - Elastic
- Building elements types (2)
  - Unspecified
  - Beam (1)
  - Column
  - Plate
  - Slab
  - Wall (1)
- Reinforcing types (4)

Fig. 4-1 Input data tree - Materials.

User chooses a material class from the **Input data tree** to define a material that will be later assigned to geometrical entities. Generally, ATENA is developed to simulate nonlinear behaviour of mainly reinforced concrete structures, therefore **Concrete** material is often assigned to 2D and 3D entities and reinforcement is mostly represented by 1D entities. A material is assigned to a geometric entities via the **Building element types,** which can be found in the **Input data tree** after the **Materials** branch.

Individual material types can be defined or modified by selecting the appropriate type from the **Materials** branch. When a required material class is selected in **the Input data tree** a table is displayed at the bottom of the main program screen as shown in Fig. 4-2.

This table can be used for the definition of a new material as well as for the modification of previously defined material types.



ID	Name	Show	Color	Prototype	Generator type	Base parameters	BE type
1	C45/55	☑	🟨	CC3DNonLinCementitio...	EuroCode2	E=36 GPa, μ=0.2, ρ=0.0023 kton/m³, α=1.2E-05 °C⁻¹	1
2	Mortar...	☑	🟫	CC3DNonLinCementitio...	ModelCode - mo...	E=22.8471592374261 GPa, μ=0.2, ρ=0.0023 kton/m³, α=1.2E-05 °...	2
3	CTB	☑	🟩	CC3DNonLinCementitio...	ModelCode - mo...	E=24.6113562148966 GPa, μ=0.2, ρ=0.0023 kton/m³, α=1.2E-05 °...	3

Name	A	C	■
Gutter	□	🟫	☑
CTB	□	🟦	☑
Mortar Bed	□	🟨	☑
Interfaces	□	🟦	□
Wheel LC1	✓	🟧	☑

✓ Project opening finished successfully (elapsed 00:00:01.5). See log

X: 9.6 Y: 0 Z: -3.2 Snap: ☑ Grid □ Points □ Curves □ Surfaces □ Solids

Fig. 4-2 Material table can be used to define new materials and to list or modify the existing ones

Tab. 4-1 ATENA material collection in ATENA-PRE

ATENA-PRE name	*ATENA name (input file command)*	*Description*
Concrete		
Cementitious2	CC3DNonLinCementitious2	Materials suitable for rock or concrete like materials. This material is identical to 3DNONLINCEMENTITIOUS except that this model is fully incremental. Material properties can be generated according the EC2, ModelCode (FRC), SP63.
	CC3DNonLinCementitious2 FRC	The material is suitable for fibre reinforced concrete.
	CC3DNonLinCementitious2Fatigue	This material is based on the CC3DNonLinCementitious2 material, extended for fatigue calculation.
	CC3DNonLinCementitious2WithTempDepProperties	This model is to be used to simulate change of material properties due to current temperature. The temperature fields can be imported from a previously performed thermal analysis.
Cementitiou2 RadWaste	CC3DNonLinCementitious2RadWaste	Material model is developed for concrete structures exposed to radiation. The effect of neutron and gamma radiation dose is reflected.
Cementitious2 SHCC	CC3DNonLinCementitious2SHCC	Strain Hardening Cementitious Composite material. Material suitable for

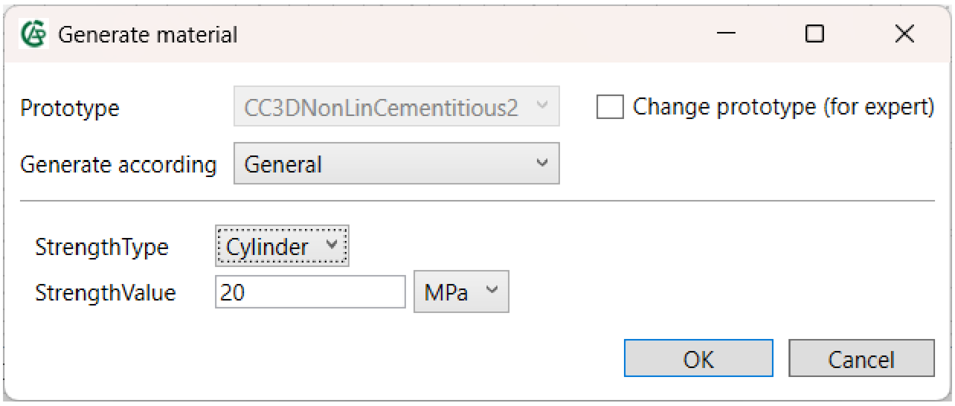
	CC3DNonLinCementitious2SHCC WithTempDepProperties	fiber reinforced concrete, such as SHCC and HPFRCC materials. Identical to CC3DNonLinCementitious2User except for the shear response definition.
Cementitious2 User	CC3DNonLinCementitious2User	Materials suitable for rock or concrete like materials. This material is identical to CC3DNonLinCementitious2 except that selected material laws can be defined by user curves.
Cementitious2 Variable	CC3DNonLinCementitious2Variable	In this material model the parameters of material behaviour are developing in time.
Cementitious3	CC3DNonLinCementitious3	Materials suitable for rock or concrete like materials. This material is an advanced version of CC3DNonLinCementitious2 material that can handle the increased deformation capacity of concrete under triaxial compression. Suitable for problems including confinement effects.
Reinforced Concrete	CCCombinedMaterial	This material can be used to create a composite material consisting of various components, such as for instance concrete with smeared reinforcement in various directions. Unlimited number of components can be specified. Output data for each component are then indicated by the label #i. Where i indicates a value of the i-th component.
Material From File	CCFromFile	Material model defined by user directly in material.inp file.
Material Random Fields	CCMaterialWithRandomFields	Material definition to be used in connection with SARA software to model material with random spatial distribution of material parameters.
Microplane	CCMicroplane7	Bazant Microplane material models for concrete
SBETA Material	CCSBETAMaterial	Older version of the basic material for concrete, only suitable for 2-D plane stress models
Reinforcement		
Reinforcement	CCReinforcement	Material for discrete reinforcement – bars and cables. You can generate material properties according the EC2
	CCCyclingReinforcement	This material is used for cycling loading
	CCReinforcementWithTemp Dep Properties	Material suitable for analyses with temperature dependency
Interface		
Interface	CC2DInterface, CC3DInterface	Interface (GAP) material for 2D and 3D analysis.
Spring		
Spring Material	CCSpringMaterial	Material for spring type boundary condition elements, i.e., for truss element modelling a spring.
Soil-Rock		

Drucker Prager	CC3DDruckerPragerPlasticity	Plastic materials with Drucker-Prager yield condition.
Steel		
Steel Von Mises 3D	CC3DBiLinearSteelVonMises	Plastic materials with Von-Mises yield condition, e.g., suitable for steel.
	CC3DBiLinearVonMisesWithTempDepPropertiess	This model is to be used to simulate change of material properties due to current temperature. The temperature fields can be imported from a previously performed thermal analysis.
Elastic		
Elastic 3D	CC3DElastIsotropic	Linear elastic isotropic materials for 3D

4.1 Concrete

The **Concrete** material library contains various version types of material models that are suitable for modelling quasi-brittle materials such as for instance concrete, rock or masonry, for more details on the available concrete material models see Tab. 4-1.

Specific **Concrete** material can be either generated or defined directly using material parameters. The dialog for material generation is shown in Fig. 4-3. The **Prototype** of material defines ATENA material model which parameters can be adjusted in **Material properties** dialog. Default material is Cementitious2 that can be generated according to EuroCode 2, Model Code (FRC) or SP63, user can select one of the mentioned standards and based on the Strength Class, Safety Format, and in some cases Strength Type (sample shape) and specifics for FRC: Type of Fibres and Weight of Fibres. Also general concrete material can be generated based on cubic or cylindric compressive strength, the formulas for general material parameters is presented in ATENA Theory [1] chapter 2.1.13. When user wants to use other Prototype, the checkbox **Change prototype (for experts)** has to be selected. Then the Prototype of Concrete material can be switched to different version, see Fig. 4-4.



Generate material

Prototype: CC3DNonLinCementitious2

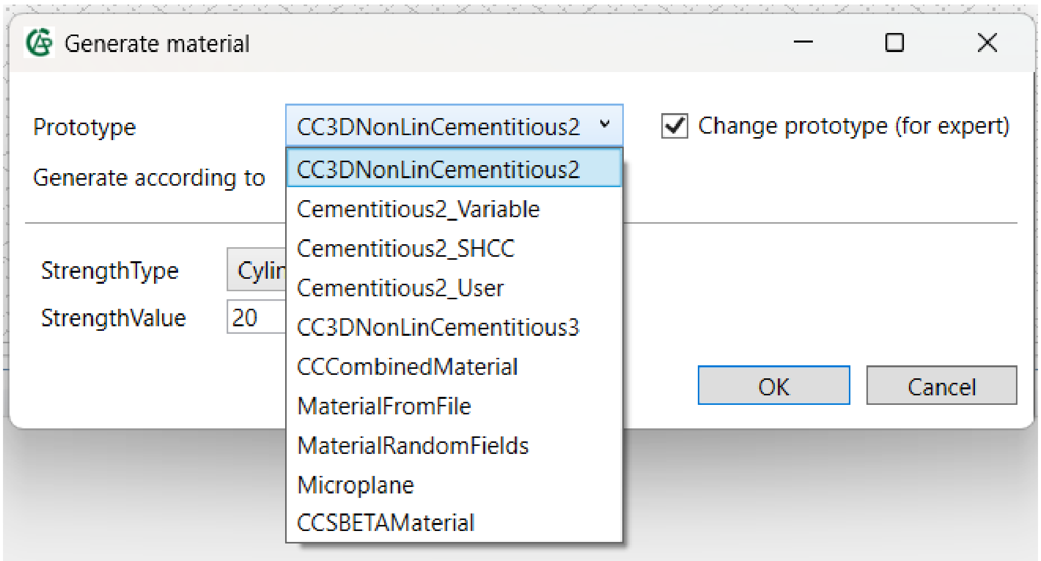
☐ Change prototype (for expert)

Generate according: General

StrengthType: Cylinder

StrengthValue: 20 MPa

Fig. 4-3 Concrete – Generation of material.



Generate material

Prototype: CC3DNonLinCementitious2

☑ Change prototype (for expert)

Generate according to

StrengthType: Cylin

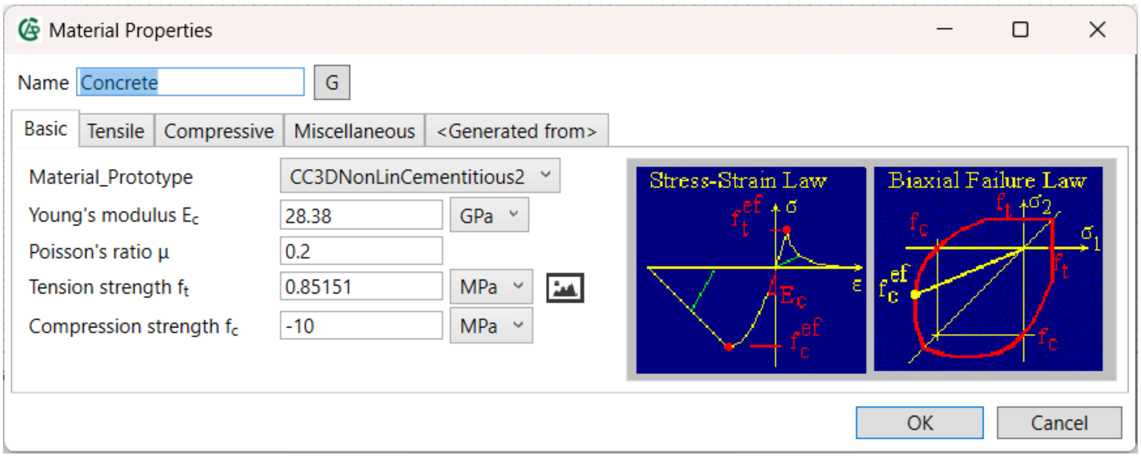
StrengthValue: 20

Fig. 4-4 Concrete – Selection of Concrete Prototypes.

Default Concrete material that is generated according to selected standard has Material properties divided in dialog window into tabs: Basic, Tensile, Compressive, Miscellaneous and Generated from, shown in Fig. 4-5. If user wishes to re-generate material values, button G offers the generate material dialog again.

The material prototype list box from the **Basic** tab allows to select the basic CC3DNonLinCementitious2, or CC3DNonLinCementitious2WithTempDepProperties, where some of the material values can depend on temperature, or CC3DNonLinCementitious2Fatigue for modelling high-cycle tensile fatigue.

The basic material parameters are defined in the **Basic** dialog – the Young's modulus of elasticity E, the Poisson's coefficient of lateral expansion, the strength in direct tension Ft, and the cylinder compressive strength Fc.



Material Properties

Name: Concrete

Material_Prototype: CC3DNonLinCementitious2

Young's modulus E_c: 28.38 GPa

Poisson's ratio μ: 0.2

Tension strength f_t: 0.85151 MPa

Compression strength f_c: -10 MPa

Fig. 4-5 Concrete – Basic material properties of Cementitious2.

The advanced parameters related to tension are defined at the **Tensile** tab: Fracture energy Gf, Fixed Crack coefficient (0 = rotated, 1 = fixed, more details you can find in ATENA Theory in section "2.1.6 Two Models of Smeared Cracks"), Crack Spacing, Tension Stiffening, Aggregate Interlock, manual definition of Shear Factor, and Unloading Factor (0 = the default unloading to the origin, 1 = unloading parallel to the initial elastic stiffness). The meaning of the parameters should be clear from the figures in the dialog and the help texts. For details on these (and also other) parameters, see the ATENA Theory Manual [1].

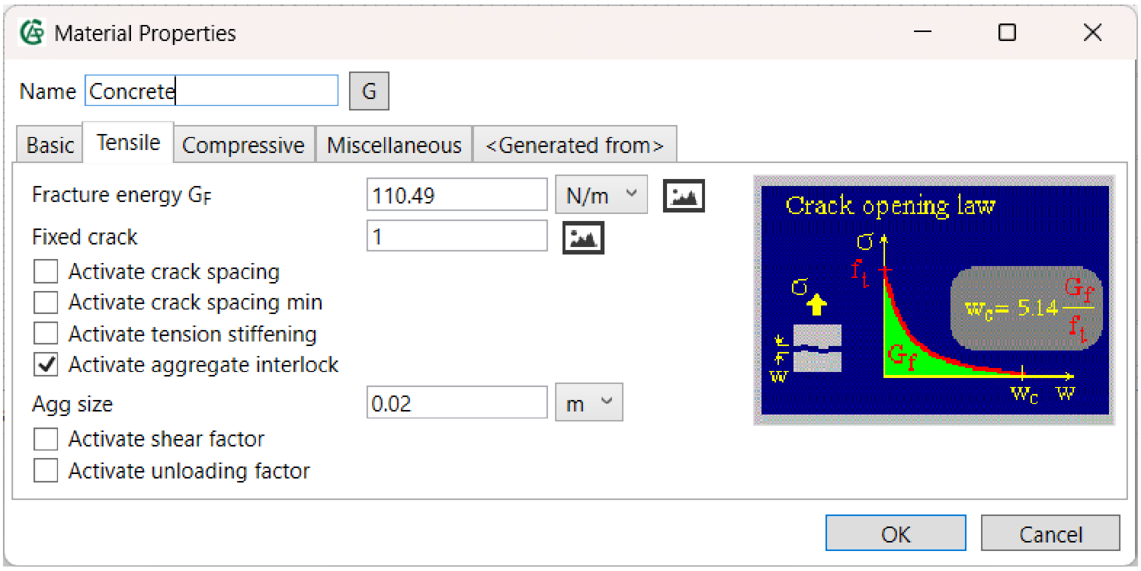
Crack Spacing option should be used when the element size is larger than the expected crack width. Typically, it should be used in reinforced concrete elements, and is equal to the expected crack spacing. In the simplest case, the spacing of ties or stirrups can be used to estimate its value.

Tension Stiffening should be used only if reinforcement is present in the model. It defines a relative tensile stress minimal limit for cracked concrete. This means the tensile stress in the cracked concrete cannot drop below this relative level (i.e., ft times tension_stiffening).

Aggregate size for the calculation of **aggregate interlock** based on the modified compression field theory by Collins. When this parameter is set, the shear strength of the cracked concrete is calculated using the modified compression field theory by Collins. The input parameter represents the maximal size of aggregates used in the concrete material.

Shear factor that is used for the calculation of cracking shear stiffness. It is calculated as a multiple of the corresponding minimal normal crack stiffness that is based on the tensile softening law.

Unloading factor controls crack closure stiffness.



Name: Concrete

Fracture energy G_F: 110.49 N/m

Fixed crack: 1

□ Activate crack spacing

□ Activate crack spacing min

□ Activate tension stiffening

☑ Activate aggregate interlock

Agg size: 0.02 m

□ Activate shear factor

□ Activate unloading factor

Fig. 4-6 Concrete – Tensile material properties of Cementitious2.

The advanced parameters influencing the compressive response are defined at the **Compressive** tab:

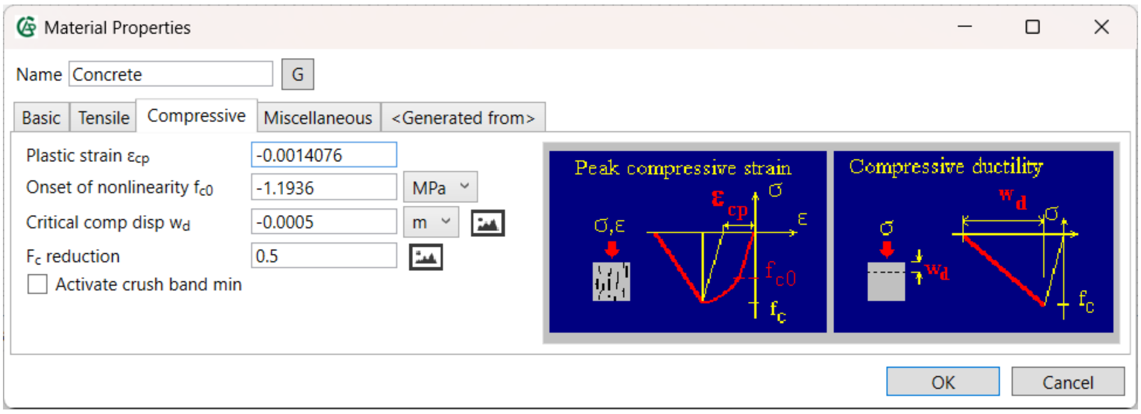
Plastic Strain at peak load eps_cp (ε_c^p) – this corresponds to the total strain at Fc from which the elastic part should be subtracted.

Onset of Crushing Fc0 represents the linearity limit.

Critical Compressive Displacement wd is the crushing compressive displacement after which the compressive stress drops to zero.

The relative limit for reduction of compressive strength due to cracking **Fc Reduction** controls the limit of Fc reduction due to cracking.

The checkbox **Activate Crush Band Min** is used to specify the minimal value of the crush band size for compressive concrete crushing. Normally crush band size is calculated as a projection of the finite element size in the direction of the highest compressive stress. If this value is defined, the crush band size is limited to the provided value, i.e., if the calculated crush band size is smaller than the limiting value, the minimal values are used. It is recommended to define it as the smallest dimension of the compressive element. See ATENA Theory manual [1] chapter on Cementitious2 material for more details.



Name Concrete G

Plastic strain ε_{cp} -0.0014076

Onset of nonlinearity f_{c0} -1.1936 MPa

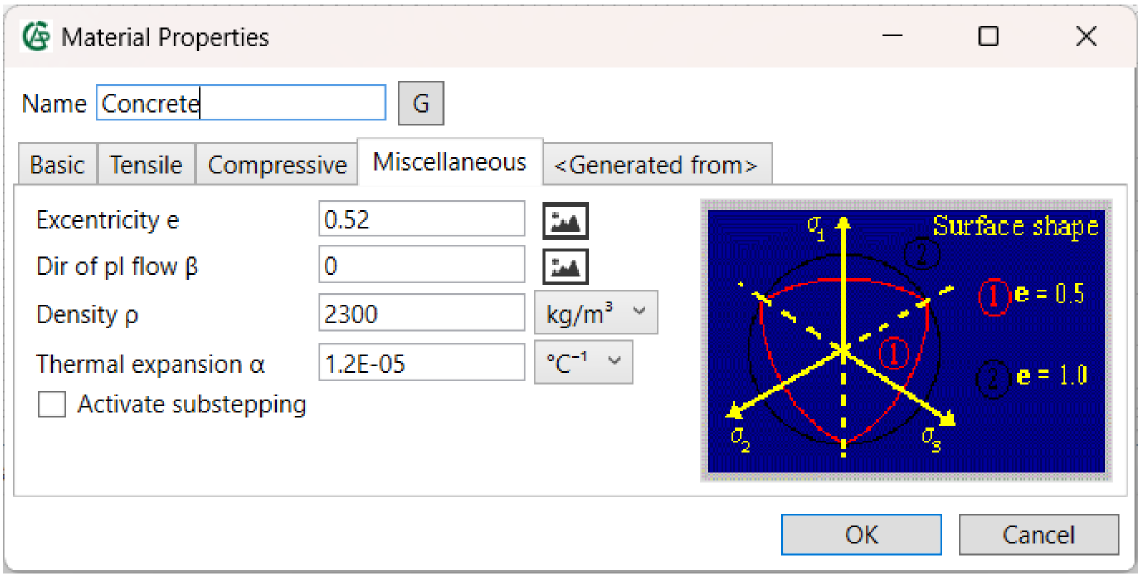
Critical comp disp w_d -0.0005 m

F_c reduction 0.5

□ Activate crush band min

Fig. 4-7 Concrete – Compressive material properties of Cementitious2.

The **Miscellaneous** tab contains two additional plasticity-related parameters, the **Eccentricity Exc** defining the shape of the failure surface, and the **Direction of Plastic Flow Beta**, determining volume compaction (Beta<0) or expansion (Beta>0) during crushing, i.e., plasticization, and two general parameters: **Density Rho** (only used in dynamic analysis) and the coefficient of **Thermal Expansion Alpha** (only used when the thermal load is applied).



Name Concrete G

Excentricity e 0.52

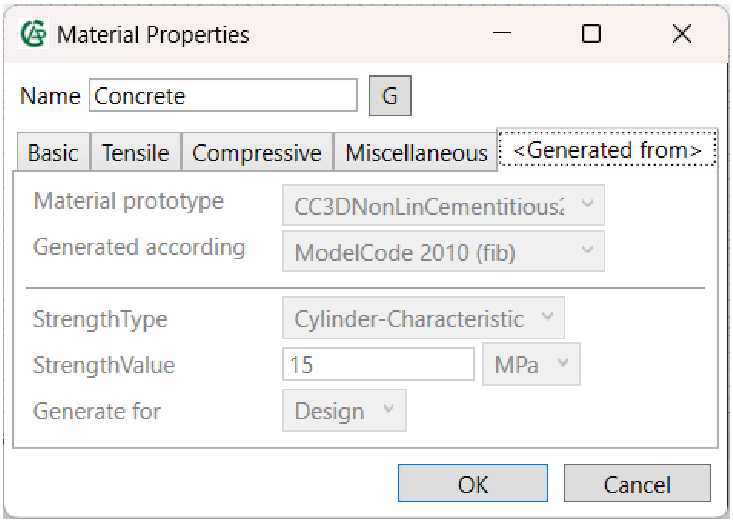
Dir of pl flow β 0

Density ρ 2300 kg/m³

Thermal expansion α 1.2E-05 °C⁻¹

□ Activate substepping

Fig. 4-8 Concrete – Miscellaneous material properties of Cementitious2.



Name: Concrete

Material prototype: CC3DNonLinCementitious2

Generated according: ModelCode 2010 (fib)

StrengthType: Cylinder-Characteristic

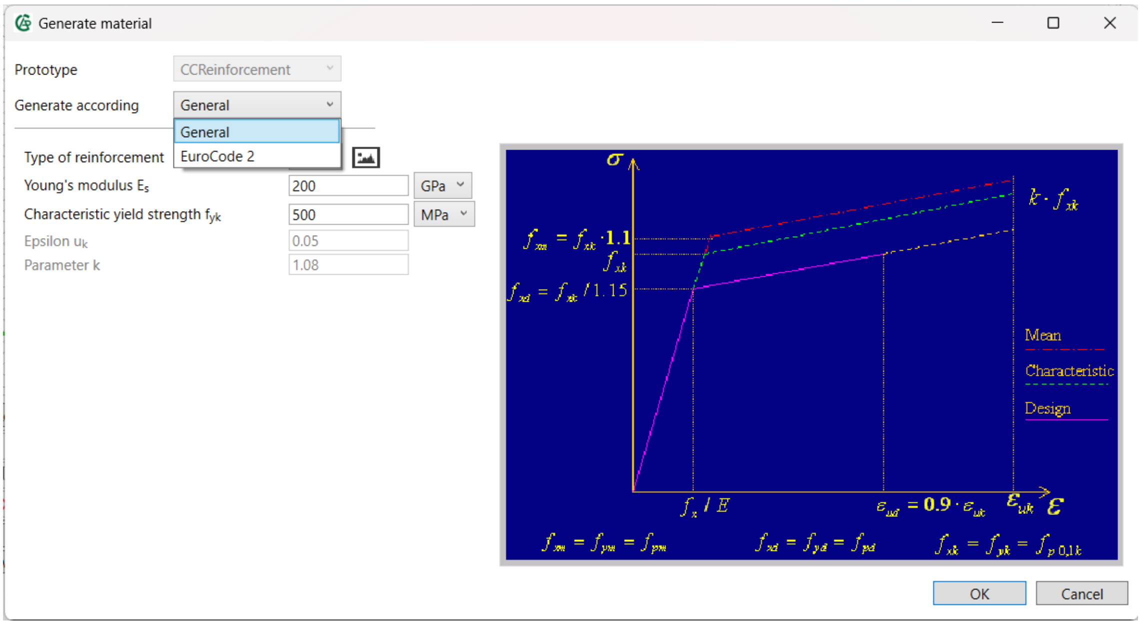
StrengthValue: 15 MPa

Generate for: Design

Fig. 4-9 Concrete – Generated from info of Cementitious2.

4.2 Reinforcement

The basic material parameters for one-dimensional reinforcement bars is defined in material **Reinforcement**. The dialog Generate material creates stress-strain function for bars or tendons based on the reinforcement steel strength class, a few basic parameters (elastic modulus, characteristic yield strength, …) and safety format, in Fig. 4-10, the generator is either General or EC2.



Prototype: CCReinforcement

Generate according: General

Type of reinforcement

Young's modulus E_s: 200 GPa

Characteristic yield strength f_{yk}: 500 MPa

Epsilon u_k: 0.05

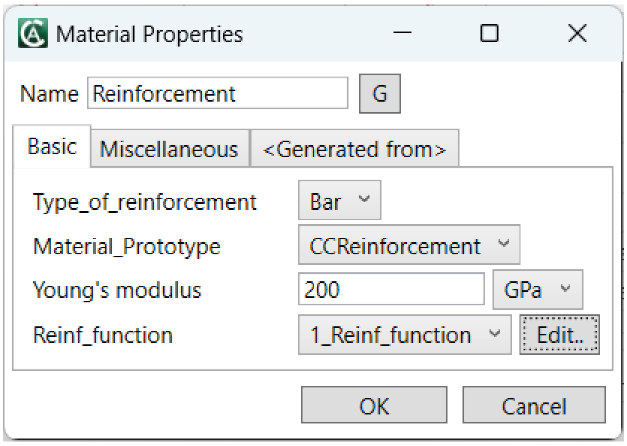
Parameter k: 1.08

Fig. 4-10 Reinforcement – Generate material.

When material is generated, the stress-strain function can still be edited by user in dialog window in Fig. 4-11 and Fig. 4-12. Also following material prototypes developed for cycling loading or temperature dependent properties can be selected.

CCCyclingReinforcement - Material for cyclic reinforcement. There is a tab Menegotto-Pinto where special parameters can be defined. Detailed information about these parameters can be find in ATENA Theory Manual [1], section 2.7.5.

CCReinforcementWithTempDepProperties - This model is used to simulate change of material properties due to current temperature. The temperature fields can be imported from a previously performed thermal analysis. Reinforcement parameters can be generated according to the production method.

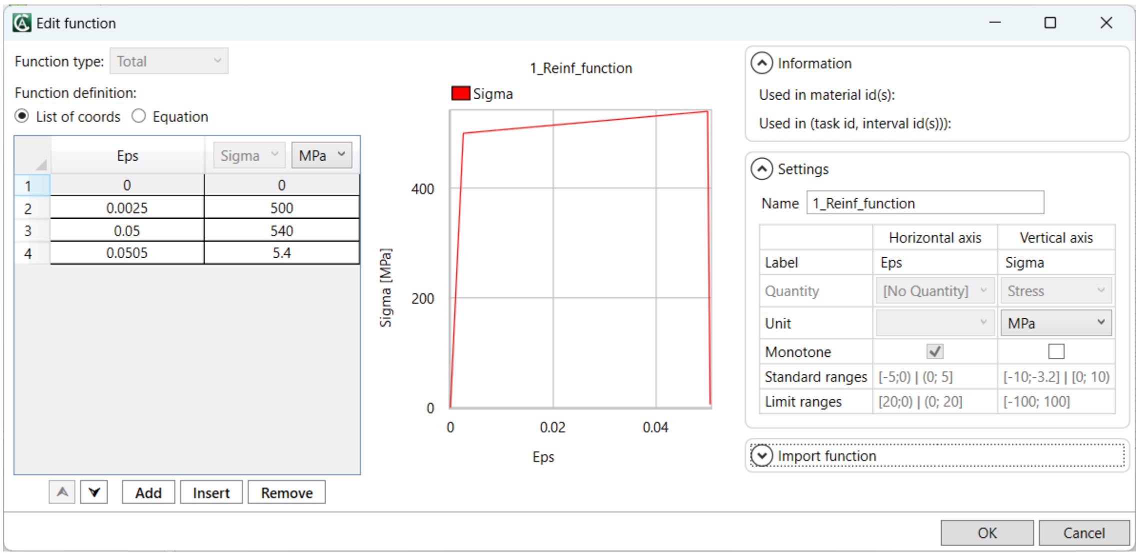


Material Properties

Name: Reinforcement

Type_of_reinforcement	Bar
Material_Prototype	CCReinforcement
Young's modulus	200 GPa
Reinf_function	1_Reinf_function

Fig. 4-11 Reinforcement – Material properties edit dialogue.



Edit function

Function type: Total

Function definition: ● List of coords ○ Equation

	Eps	Sigma MPa
1	0	0
2	0.0025	500
3	0.05	540
4	0.0505	5.4

Information

Used in material id(s):

Used in (task id, interval id(s)):

Settings

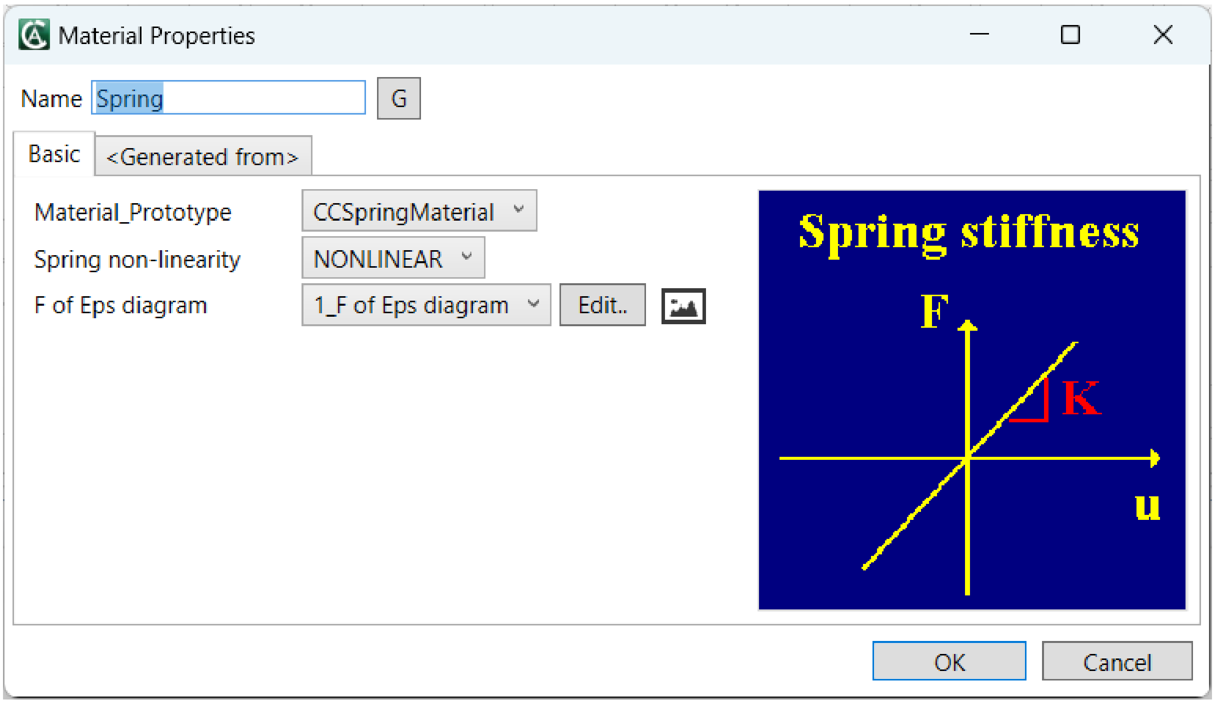
Name: 1_Reinf_function

	Horizontal axis	Vertical axis
Label	Eps	Sigma
Quantity	[No Quantity]	Stress
Unit		MPa
Monotone	☑	□
Standard ranges	[-5;0) | (0; 5]	[-10;-3.2] | [0; 10)
Limit ranges	[20;0) | (0; 20]	[-100; 100]

Fig. 4-12 Reinforcement – Edit function – stress-strain diagram.

4.3 Spring

Spring material can be used for point, surface and line springs. Spring can be linear and also nonlinear. The linear spring are dfined with stiffness while nonlinear ones are defines by force-deformation diagram, Fig. 4-13.



Name: Spring

Material_Prototype: CCSpringMaterial

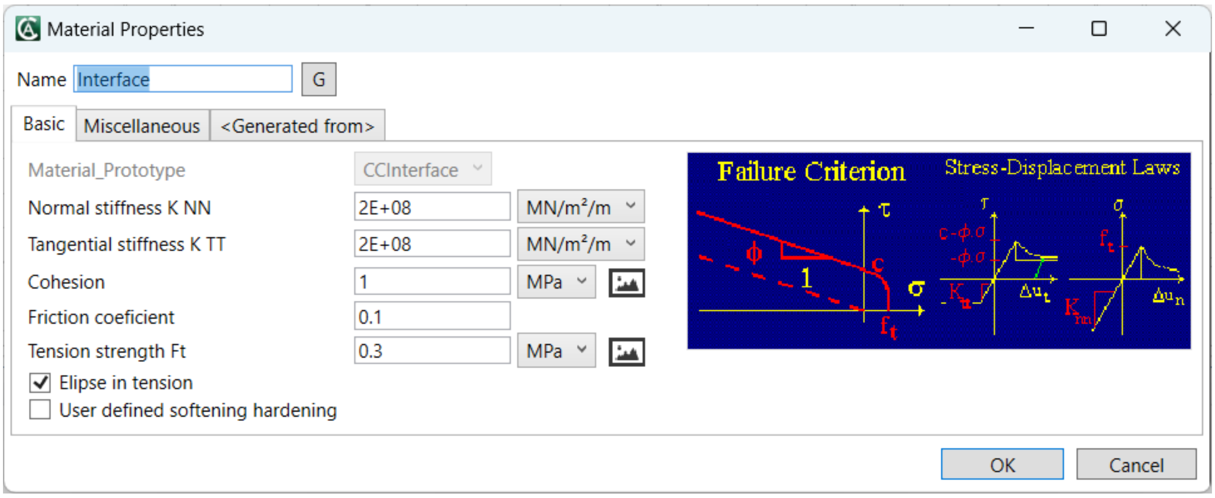
Spring non-linearity: NONLINEAR

F of Eps diagram: 1_F of Eps diagram

Fig. 4-13 Spring – Basic material properties.

4.4 Interface

The **Interface** material (also called GAP) has been developed to model behaviour of contacts between volumes, e.g., concrete - steel or thin layers of, e.g., mortar. This material should only be assigned to *contact volumes* (in 3D) or *contact surfaces* (in 2D and axisymmetry), in Fig. 4-14 and Fig. 4-15.



Name: Interface

Material_Prototype: CCInterface

Normal stiffness K NN: 2E+08 MN/m³/m

Tangential stiffness K TT: 2E+08 MN/m³/m

Cohesion: 1 MPa

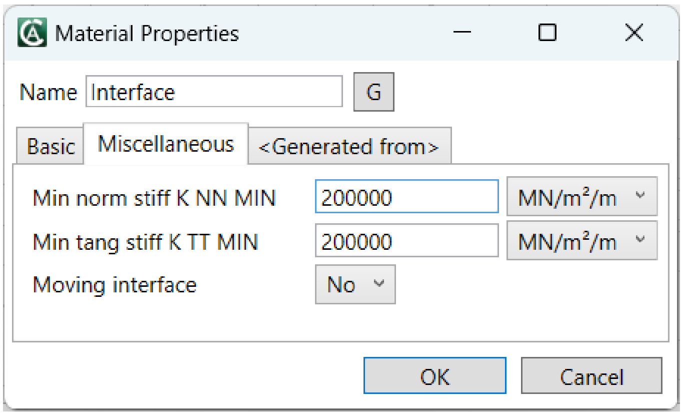
Friction coeficient: 0.1

Tension strength Ft: 0.3 MPa

☑ Elipse in tension

□ User defined softening hardening

Fig. 4-14 Interface – Basic parameters.



Name Interface

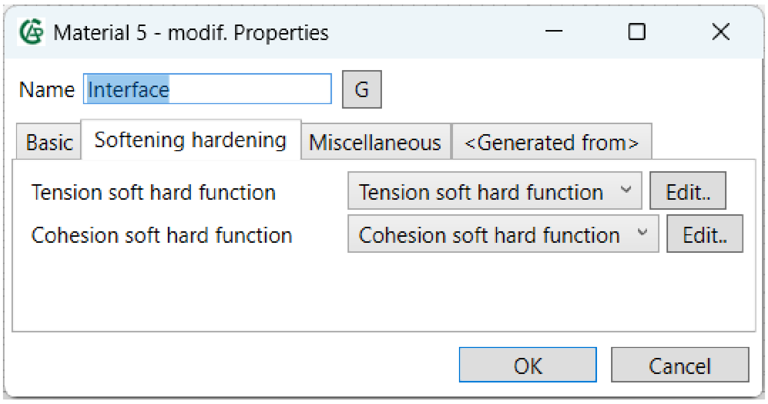
Min norm stiff K NN MIN 200000 $MN/m^2/m$

Min tang stiff K TT MIN 200000 $MN/m^2/m$

Moving interface No

Fig. 4-15 Interface – Miscellaneous.

User defined softening/hardening of the material depends on the relative displacement in tension du or in shear dv, see .



Name Interface

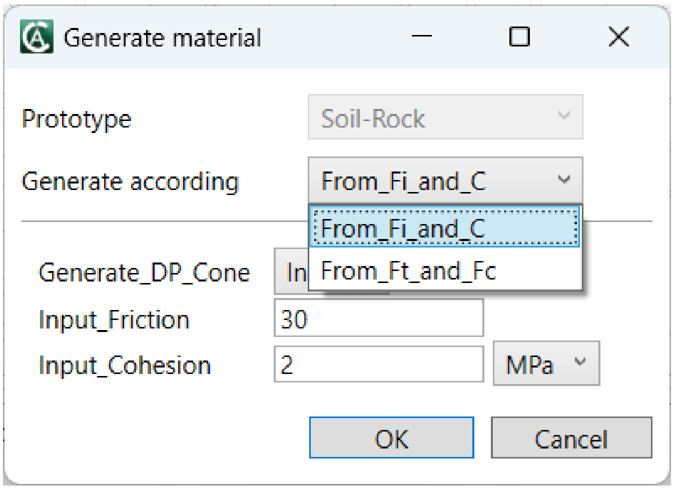
Tension soft hard function Tension soft hard function

Cohesion soft hard function Cohesion soft hard function

Fig. 4-16 Interface – Softening hardening.

4.5 Soil-Rock

Soil-Rock material is useful for defining subsoil. It utilizes Drucker-Prager plasticity model that can be generated by combination of friction angle and cohesion or by compressive and tensile strength, see Fig. 4-17. Generated parameters of model can be adjusted, Fig. 4-18.



Prototype Soil-Rock

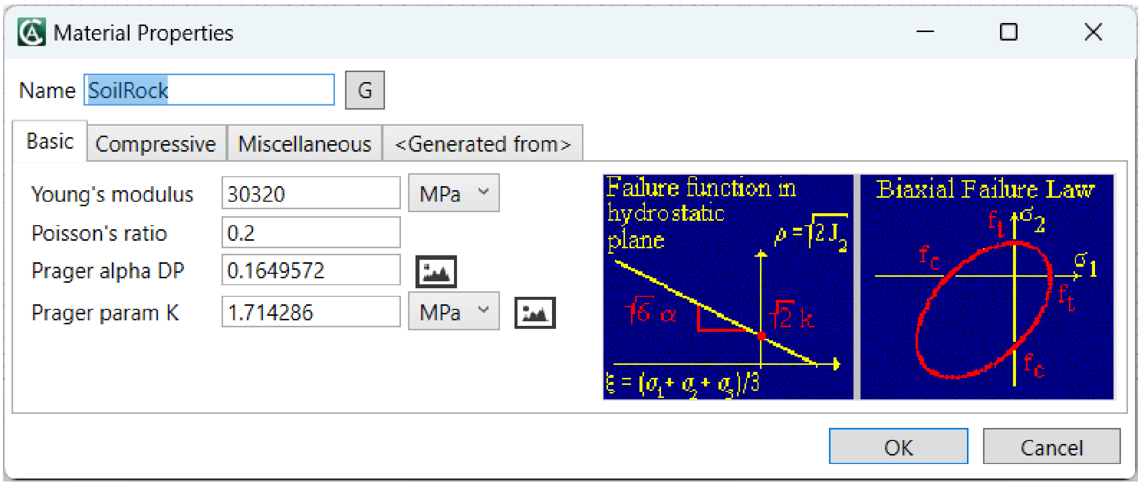
Generate according From_Fi_and_C

Generate_DP_Cone In

Input_Friction 30

Input_Cohesion 2 MPa

Fig. 4-17 Soil-Rock – Generate material dialog.



Material Properties

Name SoilRock

Basic Compressive Miscellaneous <Generated from>

Young's modulus 30320 MPa

Poisson's ratio 0.2

Prager alpha DP 0.1649572

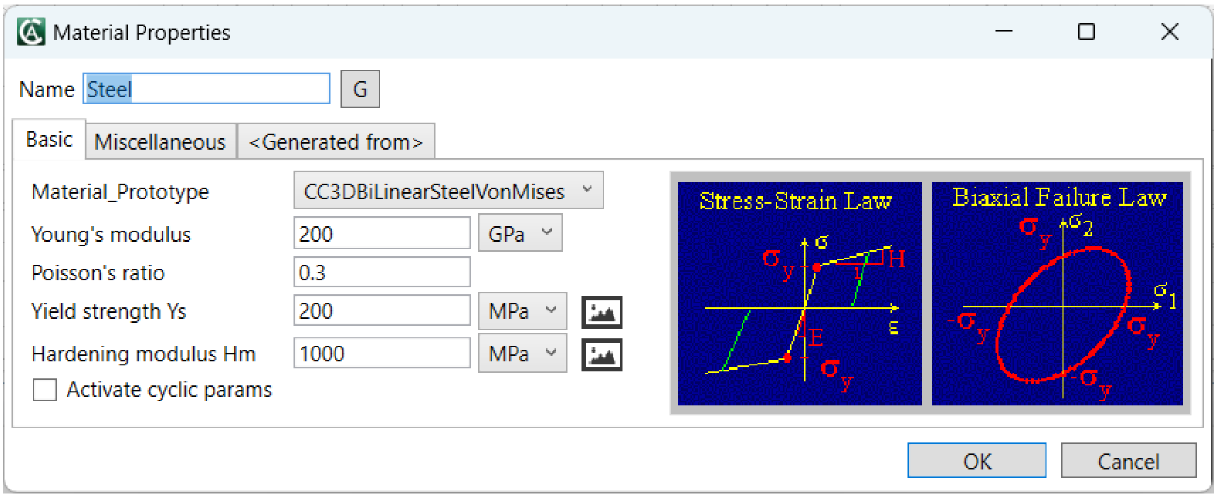
Prager param K 1.714286 MPa

Fig. 4-18 Soil-Rock – Material parameters.

4.6 Steel

Material Steel includes two material prototypes, the "normal" **CC3DBilinearSteelVonMises** and a variant supporting thermal degradation **CC3DBilinearVonMisesWithTempDepProperties**. It is also possible to define cyclic properties through enabling the option **Activate Cyclic Params**.

This material model is targeted for volume members made from steel and other similar (ductile) materials, i.e., elements which can not be well represented with the 1D reinforcement (truss) elements. Note the hardening is not limited, which means it has to be checked in post-processing if the ultimate strain has been exceeded.



Material Properties

Name Steel

Basic Miscellaneous <Generated from>

Material_Prototype CC3DBiLinearSteelVonMises

Young's modulus 200 GPa

Poisson's ratio 0.3

Yield strength Ys 200 MPa

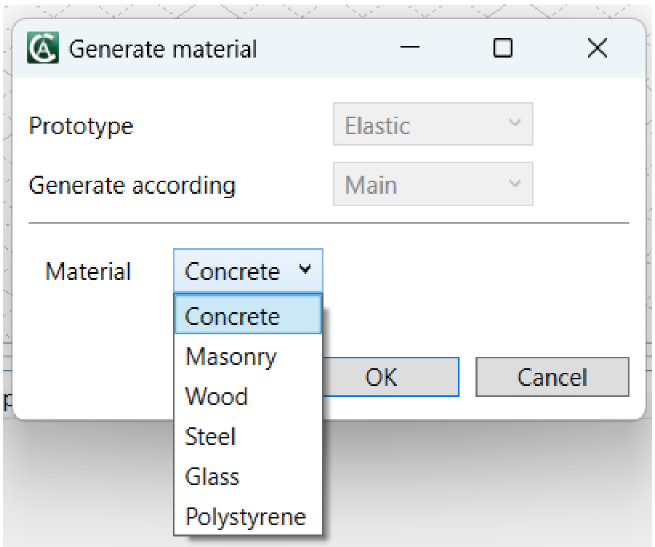
Hardening modulus Hm 1000 MPa

☐ Activate cyclic params

Fig. 4-19 Steel – Material parameters.

4.7 Elastic

Last of material choices in ATENA-PRE is **Elastic**. The material is defines by Young’s modulus and Poisson’s ratio, there is selection of various common materials like concrete, masonry, wood etc. that are available with their respective properties, Fig. 4-20 and Fig. 4-21.



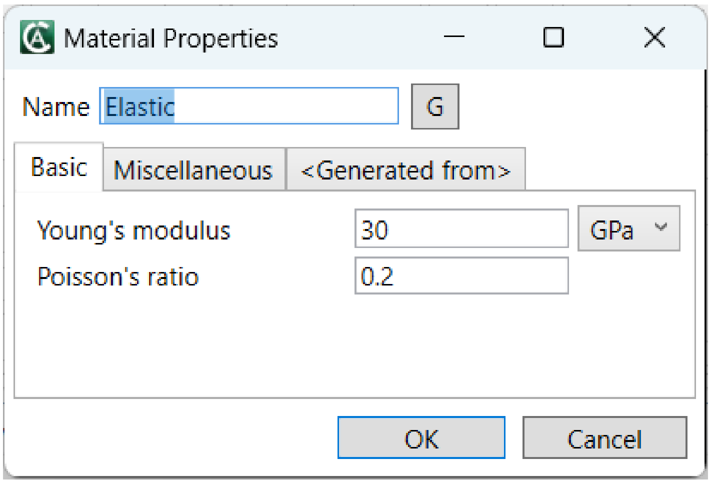
Generate material

Prototype: Elastic

Generate according: Main

Material: Concrete

Fig. 4-20 Elastic – Generate material dialog.



Material Properties

Name: Elastic

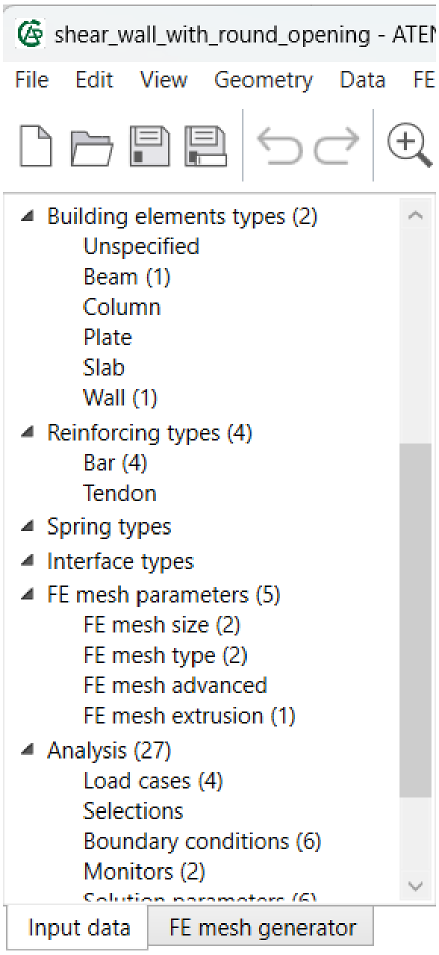
Young's modulus: 30 GPa

Poisson's ratio: 0.2

Fig. 4-21 Elastic – Material parameters.

5 BUILDING ELEMENT TYPES

Building element types interconnect model features including geometry, material and finite element mesh. The main **Building element types** are directly in the **Input data tree** (Fig. 5-1) while other types are available to choose through **Unspecified Building** element type, see complete list of options in Fig. 5-2. However, the choice of building element type is not affecting the model at the moment, it is rather preparation for import and export of IFC files that are going to be included in **ATENA-PRE** in foreseeable future.



shear_wall_with_round_opening - ATEN

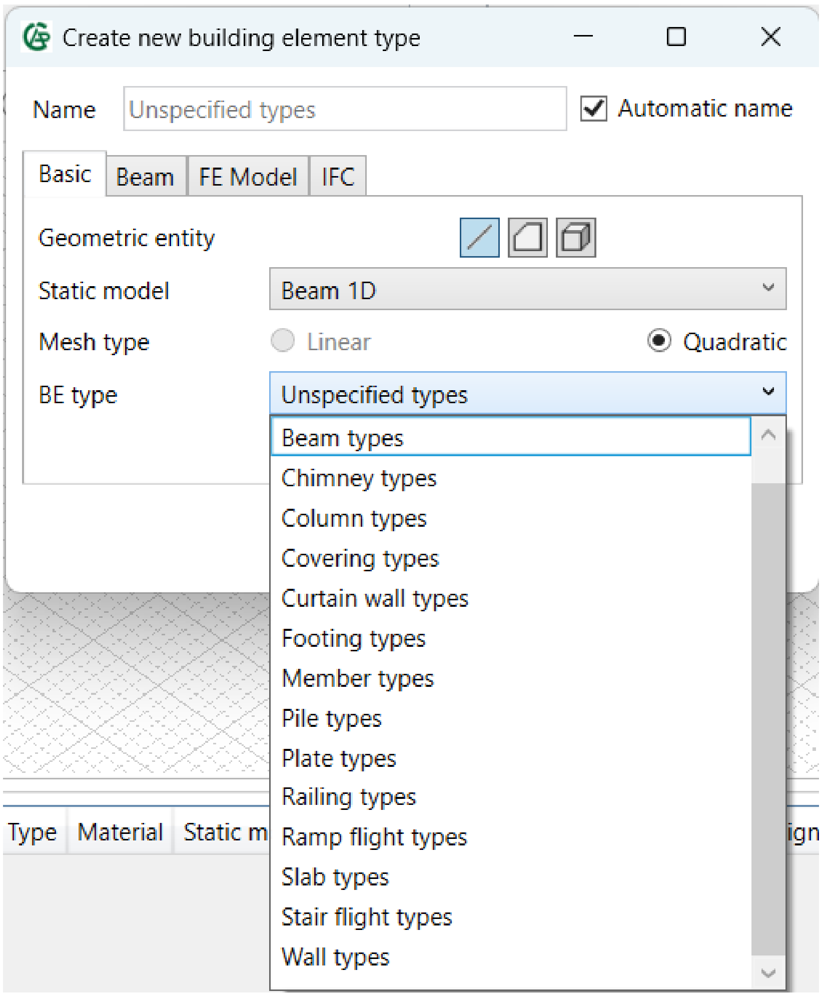
File Edit View Geometry Data FE

- Building elements types (2)
  - Unspecified
  - Beam (1)
  - Column
  - Plate
  - Slab
  - Wall (1)
- Reinforcing types (4)
  - Bar (4)
  - Tendon
- Spring types
- Interface types
- FE mesh parameters (5)
  - FE mesh size (2)
  - FE mesh type (2)
  - FE mesh advanced
  - FE mesh extrusion (1)
- Analysis (27)
  - Load cases (4)
  - Selections
  - Boundary conditions (6)
  - Monitors (2)
  - Solution parameters (6)

Input data | FE mesh generator

Fig. 5-1 Input data tree – Building element types.

Each **Building element type** is assigned to geometrical entity along with specific material that has been defined by user in **Material section**. Based on the type of entity (curve/surface/solid) the Static model defining finite element (2D, 3D, shell or beam elements) is offered to adjust. Other properties include type of FE mesh (linear/quadratic) and other idealization specifics when the user works with beam or shell approximation, or e.g. plane-stress or plain-strain etc.



Create new building element type

Name: Unspecified types

☑ Automatic name

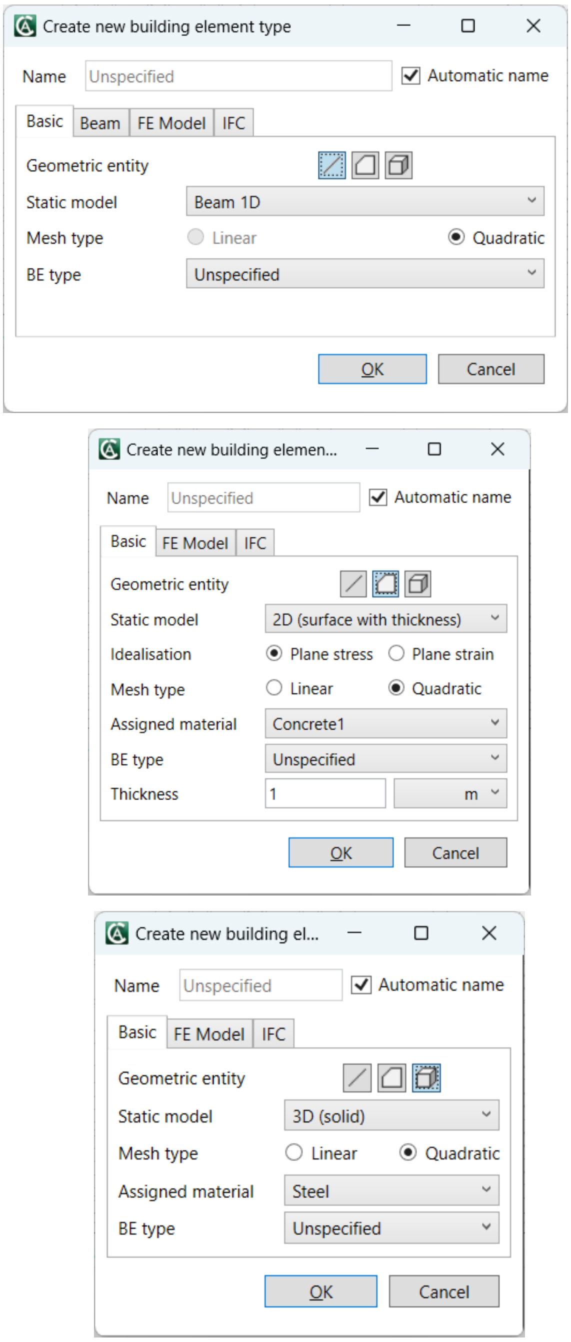
Static model: Beam 1D

Mesh type: ○ Linear ● Quadratic

BE type: Unspecified types

Fig. 5-2 Building element types options.

BE type properties differ for curve, surface or solid, Fig. 5-3.



Create new building element type

Name Unspecified ☑ Automatic name

Basic | Beam | FE Model | IFC

Geometric entity

Static model: Beam 1D

Mesh type: ○ Linear ● Quadratic

BE type: Unspecified

OK Cancel

Create new building elemen...

Name Unspecified ☑ Automatic name

Basic | FE Model | IFC

Geometric entity

Static model: 2D (surface with thickness)

Idealisation: ● Plane stress ○ Plane strain

Mesh type: ○ Linear ● Quadratic

Assigned material: Concrete1

BE type: Unspecified

Thickness: 1 m

OK Cancel

Create new building el...

Name Unspecified ☑ Automatic name

Basic | FE Model | IFC

Geometric entity

Static model: 3D (solid)

Mesh type: ○ Linear ● Quadratic

Assigned material: Steel

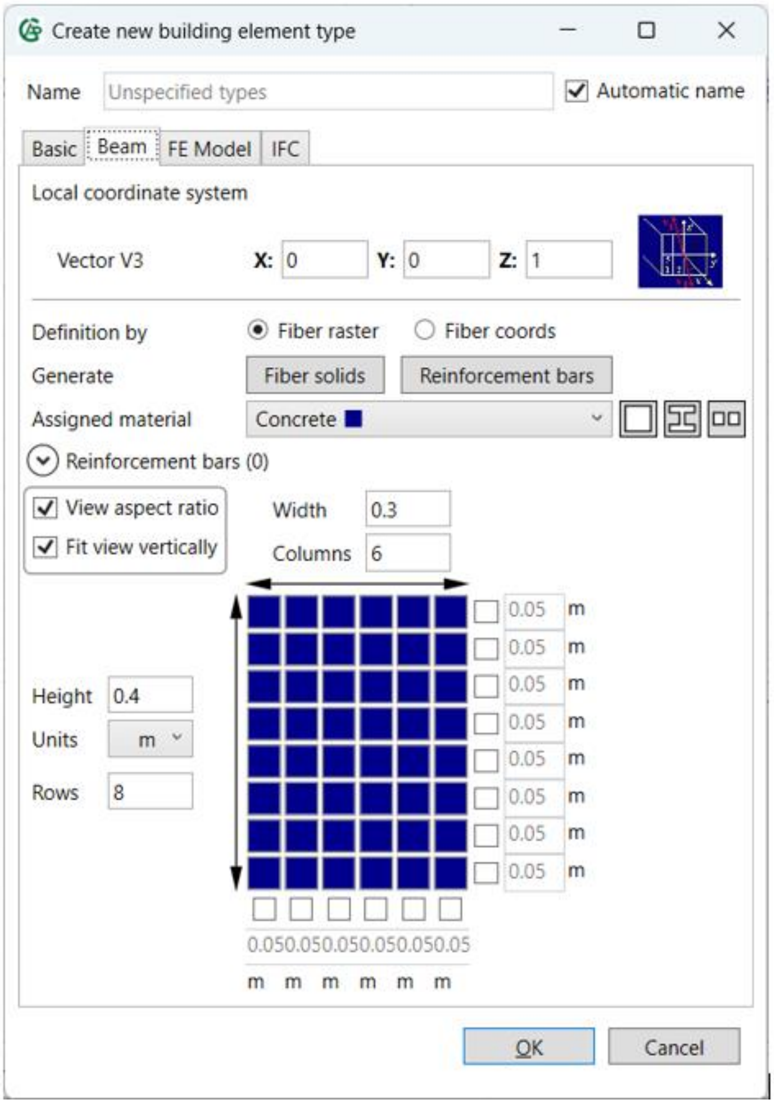
BE type: Unspecified

OK Cancel

Fig. 5-3 Building element types – 1D/2D/3D properties.

1D entities (curves) use beam static model with quadratic mesh only. Beam cross-section is defined by one material and the reinforcing bars can be added to it. The beam cross-section can be defined via either raster or coordinates with specific material representing each fibre, see Fig. 5-4.

The local coordinate system of beam element is specified by vector V3 that represents vertical orientation of beam cross-section; the longitudinal direction of beam is automatically assumed in direction of assigned geometry.



Create new building element type

Name Unspecified types ☑ Automatic name

Basic | Beam | FE Model | IFC

Local coordinate system

Vector V3 X: 0 Y: 0 Z: 1

Definition by ● Fiber raster ○ Fiber coords

Generate: Fiber solids | Reinforcement bars

Assigned material: Concrete

Reinforcement bars (0)

☑ View aspect ratio

☑ Fit view vertically

Width: 0.3

Columns: 6

Height: 0.4

Units: m

Rows: 8

□ 0.05 m
□ 0.05 m
□ 0.05 m
□ 0.05 m
□ 0.05 m
□ 0.05 m
□ 0.05 m
□ 0.05 m

□ □ □ □ □ □

0.050.050.050.050.050.05

m m m m m m

OK Cancel

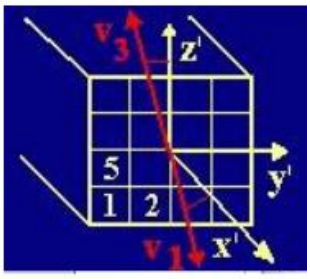
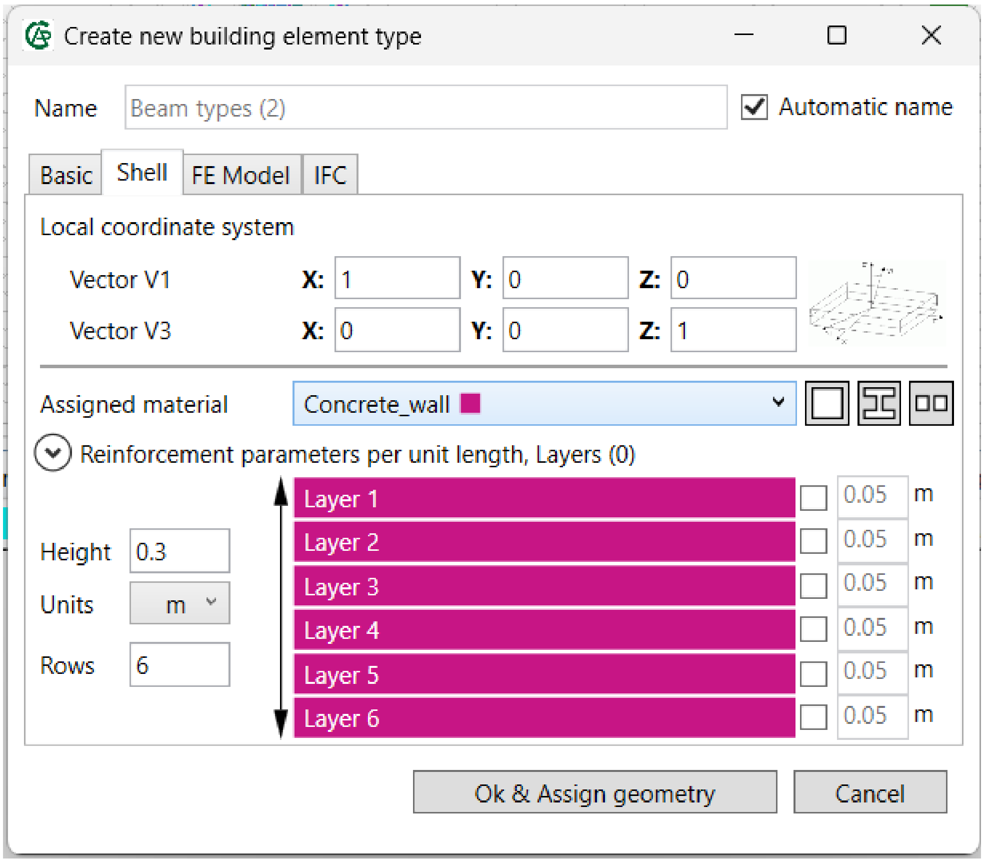


Fig. 5-4 Building element types – beam fibres.

For 2D entities (surface), user can choose between static model with 2D elements with thickness or layered 2D shell elements. The plane-stress or plain-strain idealization, and linear or quadratic mesh are combined with 2D elements with thickness. The shell elements are quadratic only and its layers can combine different materials, see Fig. 5-5. Vertical vector V3 is defined automatically from geometry while longitudinal vector V1 needs to be defined by user.



Create new building element type

Name: Beam types (2) ☑ Automatic name

Basic | Shell | FE Model | IFC

Local coordinate system

Vector V1	X:	1	Y:	0	Z:	0
Vector V3	X:	0	Y:	0	Z:	1

Assigned material: Concrete_wall 🟥

Reinforcement parameters per unit length, Layers (0)

Height: 0.3

Units: m

Rows: 6

Layer 1	□	0.05	m
Layer 2	□	0.05	m
Layer 3	□	0.05	m
Layer 4	□	0.05	m
Layer 5	□	0.05	m
Layer 6	□	0.05	m

Ok & Assign geometry | Cancel

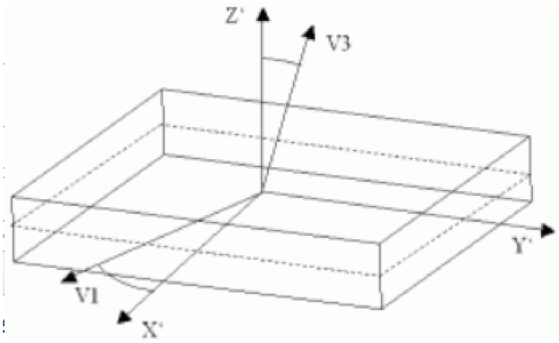
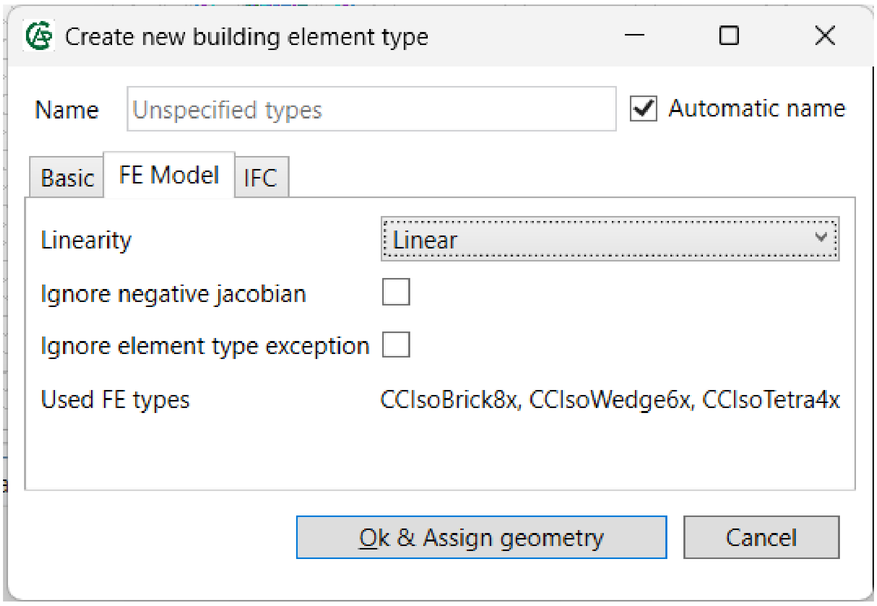


Fig. 5-5 Building element types – shell layers.

Finally, 3D entities (solid) offer three static models: 3D solid, Beam or Layered shell. Solid offers both linear and quadratic mesh while beam and shell use just quadratic mesh. Beam and shell can combine base material with reinforcement in various cross-section shapes as is shown in previous figures for 1D beam and 2D shell, the only difference is definition of both vectors V1 and V3.

FE Model options offer choice of geometrical linearity of the model, more information can be found in input file manual [4], also in Fig. 5-6. Along with negative jacobian and exception that can be triggered during analysis and terminate the calculation, the choice of ignoring them leads to disputable results but might be used in specific cases e.g. to get results when the structure is already heavily damaged.



Name: Unspecified types

☑ Automatic name

Linearity: Linear

Ignore negative jacobian □

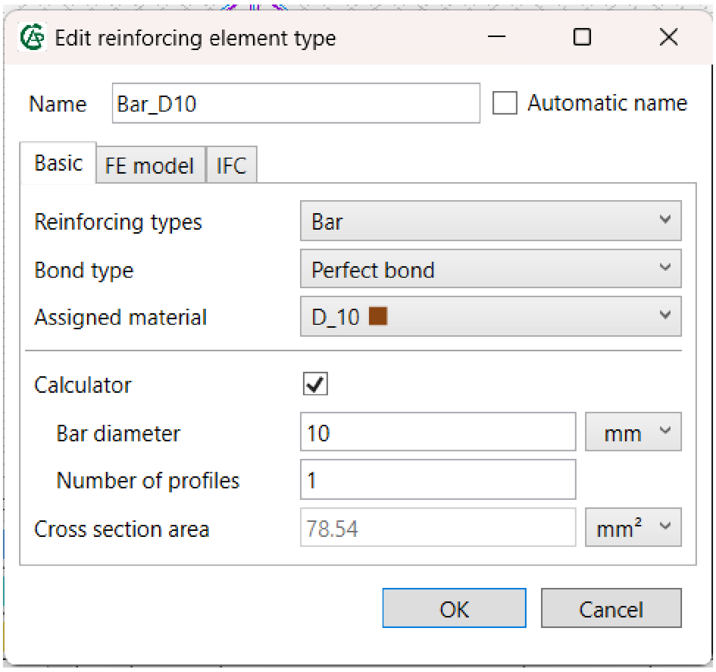
Ignore element type exception □

Used FE types: CCIsoBrick8x, CCIsoWedge6x, CCIsoTetra4x

Fig. 5-6 Building element types – FE model.

6 Reinforcing Types

Reinforcing types serves for the definition of properties of 1D reinforcement elements. Beside the assignment of material to geometry, the bond properties and cross-section area of bar/tendon or the whole region of reinforcement are defined, Fig. 6-1.



Edit reinforcing element type

Name: Bar_D10 □ Automatic name

Reinforcing types: Bar

Bond type: Perfect bond

Assigned material: D_10 🟫

Calculator: ☑

Bar diameter: 10 mm

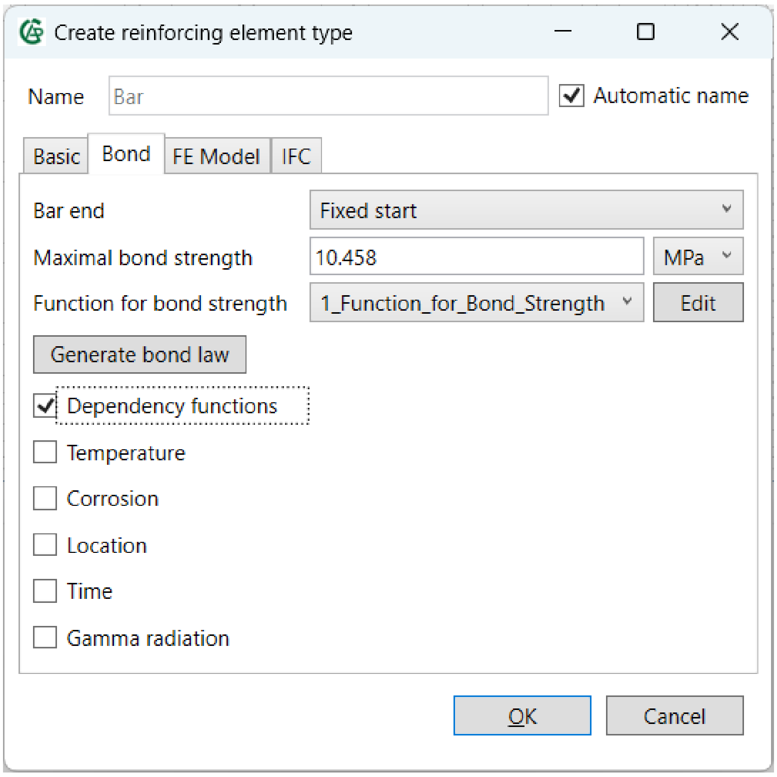
Number of profiles: 1

Cross section area: 78.54 mm^2

Fig. 6-1 Reinforcing element types – Options.

Calculator of cross-section area and its perimeter is included as well.

Then the bond type offers three options: bond, cyclic and perfect. The bond is standard option for reinforcement where slip occurs once the bond strength is reached. The bond-slip law is defined in tab Bond, Fig. 6-2. The cyclic bond has extra feature added to bond-slip which is parameters defining unloading response. Finally, the perfect bond does not allow any slip and the reinforcement is always totally connected to surrounding concrete.



Create reinforcing element type

Name: Bar — ☑ Automatic name

Bar end: Fixed start

Maximal bond strength: 10.458 MPa

Function for bond strength: 1_Function_for_Bond_Strength

☑ Dependency functions

□ Temperature

□ Corrosion

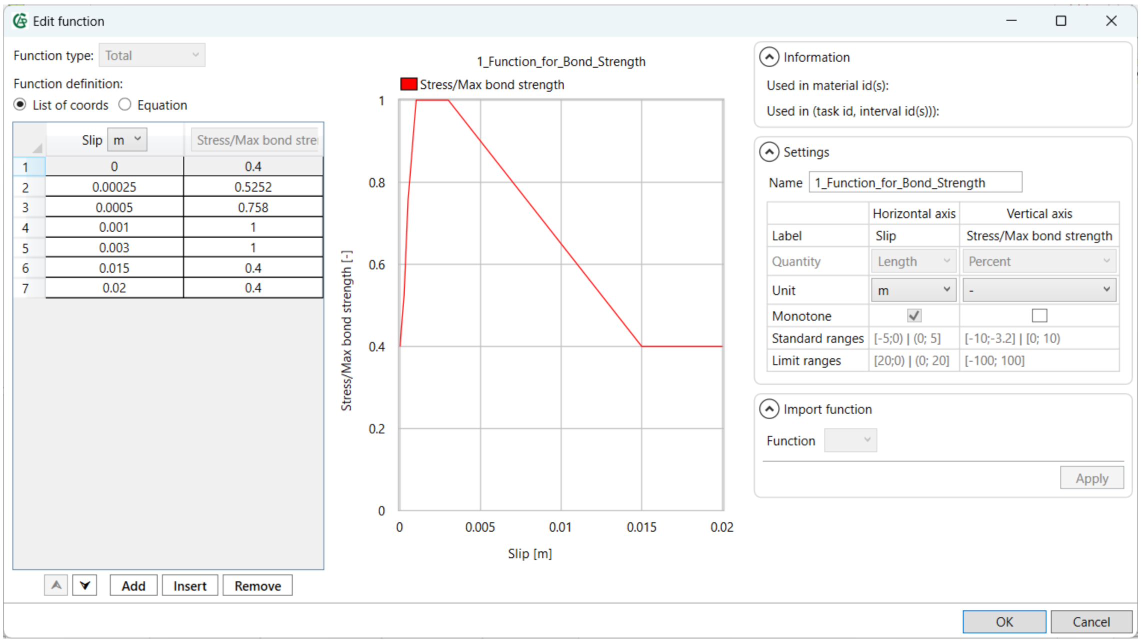
□ Location

□ Time

□ Gamma radiation

Fig. 6-2 Reinforcing element types – Bond settings.

Bar/tendon starting or ending node can be fixed which blocks it from slipping. Following bond properties specify bond-slip function which is proportional based on maximum bond strength. It can be Manually added, imported, see Fig. 6-3, or also generated according to Bigaj or Ceb Fib Model Code, Fig. 6-4. Moreover, the bond strength can be dependent on temperature, corrosion, location, time and gamma radiation.



Edit function

Function type: Total

Function definition: ● List of coords ○ Equation

	Slip [m]	Stress/Max bond stre
1	0	0.4
2	0.00025	0.5252
3	0.0005	0.758
4	0.001	1
5	0.003	1
6	0.015	0.4
7	0.02	0.4

Add
Insert
Remove
1_Function_for_Bond_Strength
🟥 Stress/Max bond strength
Stress/Max bond strength [-]
1
0.8
0.6
0.4
0.2
0
0
0.005
0.01
0.015
0.02
Slip [m]

Information

Used in material id(s):

Used in (task id, interval id(s))):

Settings

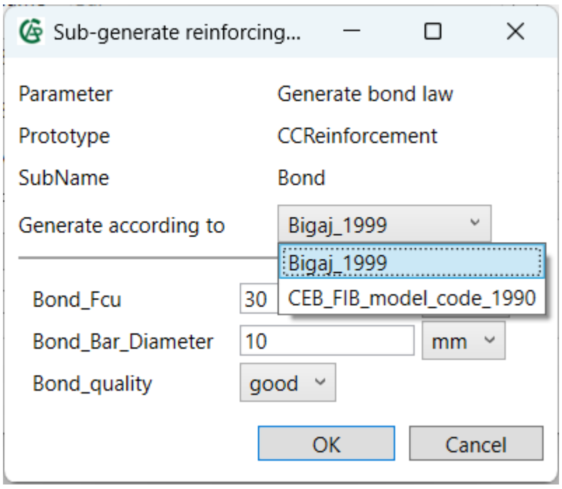
Name: 1_Function_for_Bond_Strength

	Horizontal axis	Vertical axis
Label	Slip	Stress/Max bond strength
Quantity	Length	Percent
Unit	m	-
Monotone	☑	□
Standard ranges	[-5;0) | (0; 5]	[-10;-3.2] | [0; 10)
Limit ranges	[20;0) | (0; 20]	[-100; 100]

Import function

Function:

Fig. 6-3 Reinforcing element types – Bond edit function.


Sub-generate reinforcing...

Parameter: Generate bond law

Prototype: CCReinforcement

SubName: Bond

Generate according to: Bigaj_1999 (options: Bigaj_1999, CEB_FIB_model_code_1990)

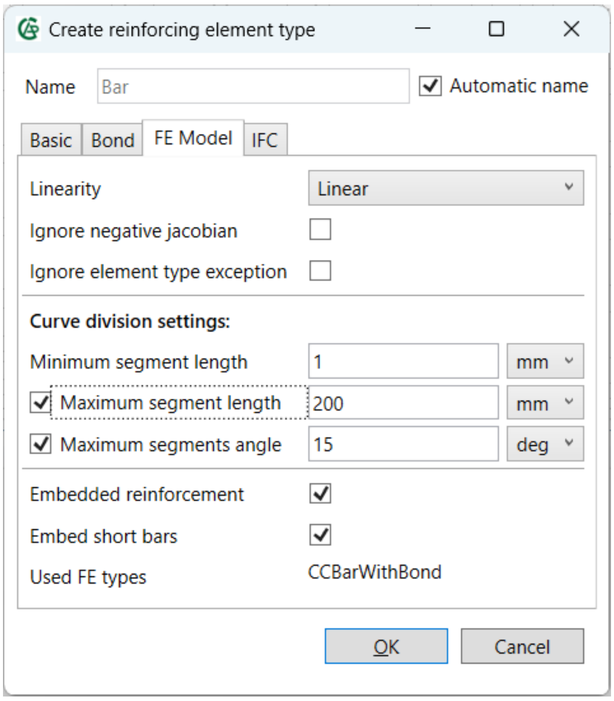
Bond_Fcu: 30

Bond_Bar_Diameter: 10 mm

Bond_quality: good

Fig. 6-4 Reinforcing element types – generate bond.

FE model tab covers specifics for linearity, negative jacobian and exception that have been introduced in BE types above. Moreover the settings of reinforcement curve division is applicable on curved reinforcement bars which are always handled as partially linear. Therefore the settings allows user to control the division of the curve into linear element parts, see Fig. 6-5. Standard division creates nodes as an intersection of curve geometry with finite element boundary (line or surface). So generally the size of linear pieces is similar to finite element mesh size while there might be curved pieces of reinforcement within one element a it is then desired to make finer mesh on the reinforcement curve. The maximum length of segment sets the maximum size of elements while the minimum skips the small parts that might arise when the node is created right next to element boundaries. The minimum segment eliminates the unnecessary small parts in curve meshing. Finally the maximum segment angle allows the maximum inclination from the adjacent elements and makes finer mesh in area with smaller radius.


Create reinforcing element type

Name: Bar ☑ Automatic name

Linearity: Linear

Ignore negative jacobian ☐

Ignore element type exception ☐

Curve division settings:

Minimum segment length: 1 mm

☑ Maximum segment length: 200 mm

☑ Maximum segments angle: 15 deg

Embedded reinforcement ☑

Embed short bars ☑

Used FE types: CCBarWithBond

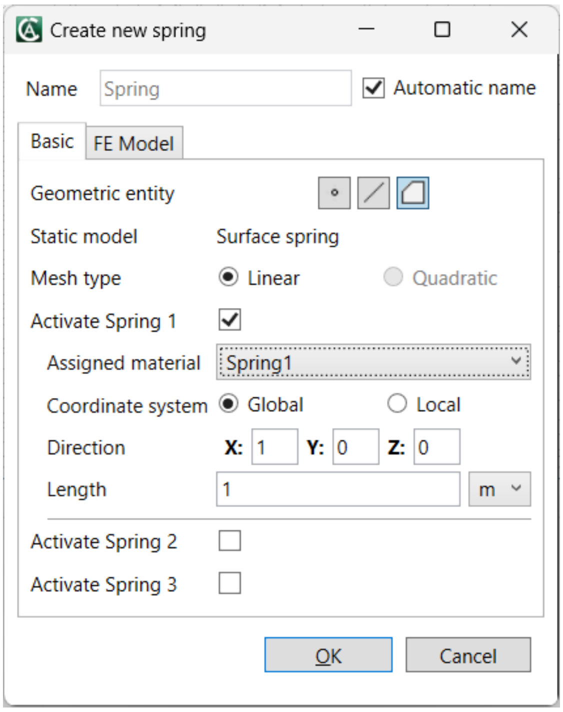
Fig. 6-5 Reinforcing element types – FE model.

7 SPRING AND INTERFACE TYPES

As in previous chapters where material and finite element properties have been assigned to geometry, Spring and Interface types serve the same purpose just for spring and interface materials.

7.1 Spring types

Spring types can be assigned to point, lines and surfaces, Fig. 7-1. In all cases the mesh of spring types is linear. Based on the geometric entity the spring is specified by cross-section area for point, thickness for curve (cross-section thickness times curve length) or the surface area which does not need to be specified within dialogue. The orientation of the spring can be defined in the local or global system. More springs can be assigned by activating another spring that has different orientation or material. Spring length determines the relative elongation/shortening defined in spring material properties, the default is recommended 1 m.



Name: Spring ☑ Automatic name

Geometric entity

Static model: Surface spring

Mesh type: ● Linear ○ Quadratic

Activate Spring 1 ☑

Assigned material: Spring1

Coordinate system: ● Global ○ Local

Direction X: 1 Y: 0 Z: 0

Length: 1 m

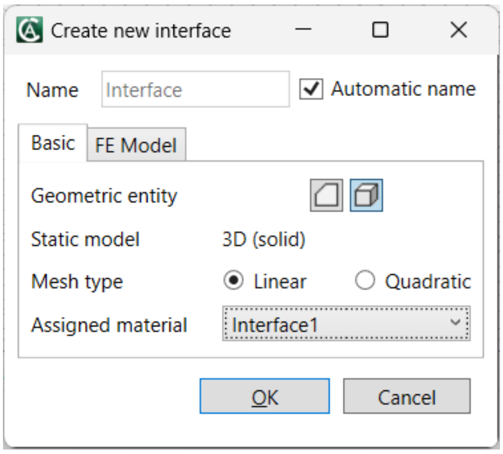
Activate Spring 2 □

Activate Spring 3 □

Fig. 7-1 Spring types.

7.2 Interface types

Interface is applicable to surface and volume entities, Fig. 7-2. The mesh assigned to interface types can be linear or quadratic and for 2D entities the interface thickness has to be specified, for axisymmetry the recommended value is 1 m.



Create new interface

Name Interface ☑ Automatic name

Basic | FE Model

Geometric entity

Static model 3D (solid)

Mesh type ● Linear ○ Quadratic

Assigned material Interface1

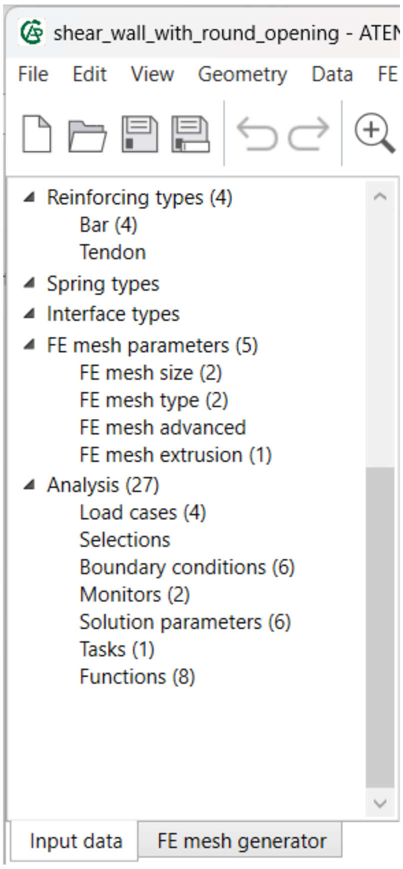
Fig. 7-2 Interface types.

8 FE Mesh Parameters and Mesh generator

The finite element mesh quality has a very high influence on the quality of the analysis results, the speed, and memory requirements. Refining only the important parts can save a lot of processor time and disk space.

Partially the finite element mesh settings are described in previous chapters mainly the finite element types and its definition, however in this chapter the mesh size, shape of finite elements and mesh compatibility are presented, Fig. 8-1.

FE mesh parameters allows to define mesh settings in specific parts of the model while general settings that are applied on the whole structure are adjustable in Mesh generator. **ATENA-PRE** uses **T3D** mesh generator that is developed by Daniel Rypl [5].

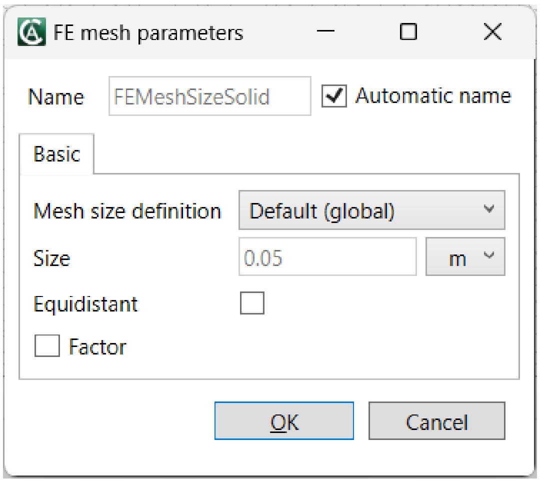


- Reinforcing types (4)
  - Bar (4)
  - Tendon
- Spring types
- Interface types
- FE mesh parameters (5)
  - FE mesh size (2)
  - FE mesh type (2)
  - FE mesh advanced
  - FE mesh extrusion (1)
- Analysis (27)
  - Load cases (4)
  - Selections
  - Boundary conditions (6)
  - Monitors (2)
  - Solution parameters (6)
  - Tasks (1)
  - Functions (8)

Fig. 8-1 Input data tree – FE mesh parameters.

8.1 Mesh size and FE mesh generator

First of mesh settings in Fig. 8-2 is **mesh size** where size of mesh can be default, uniform or user defined. The default and uniform mesh size can be assigned within FE mesh parameter while their value is possible to adjust in **FE mesh generator**, see Fig. 8-3. The user defined mesh size can be assigned and adjusted in FE mesh parameter directly. The default mesh size is applied on geometry that lack any other mesh size specifications in points. Uniform mesh size propagates through all model entities. Mesh size that is applied on the geometry is the smallest mesh size out of default, uniform or locally assigned.



FE mesh parameters

Name FEMeshSizeSolid ☑ Automatic name

Basic

Mesh size definition: Default (global)

Size: 0.05 m

Equidistant ☐

☐ Factor

Fig. 8-2 FE mesh parameters – Size of mesh.

Then there is equidistant possibility to ensure the regular mesh within selected area. Finally, the factor option is used to specify the vertex mesh size multiplication factor, the default value is 1 and it must not be negative.

There is specific command of mesh definition available for curves only. User can divide the curves not just by size but also by density and count. The density value does not have upper limit while the value cannot be negative, for example density 0.5 of curve makes mesh size double of the curve length. The count is specified number of divisions on the curve, shown in Fig. 8-4.

Other parameters within **FE mesh generator** such as Silent mode, Suppress warning messages, Include virtual entities, Enable curvature violation, Boundary conforming elements, Disable default designation, Disable nodal smoothing, Disable quad and hexa meshing, Enable convexity check are explained in more detail in T3D manual [5] in chapter 8 Command line options.

Default curve density rules the size of mesh on curves and is counted as ratio between curve length and density.

User defined epsilon is basically tolerance to solve geometrical ambiguities regarding the mesh, it is also used for definition of points forced to curves or surfaces (see 3.2) which can be useful for creating the monitor at certain point that will be mesh node.

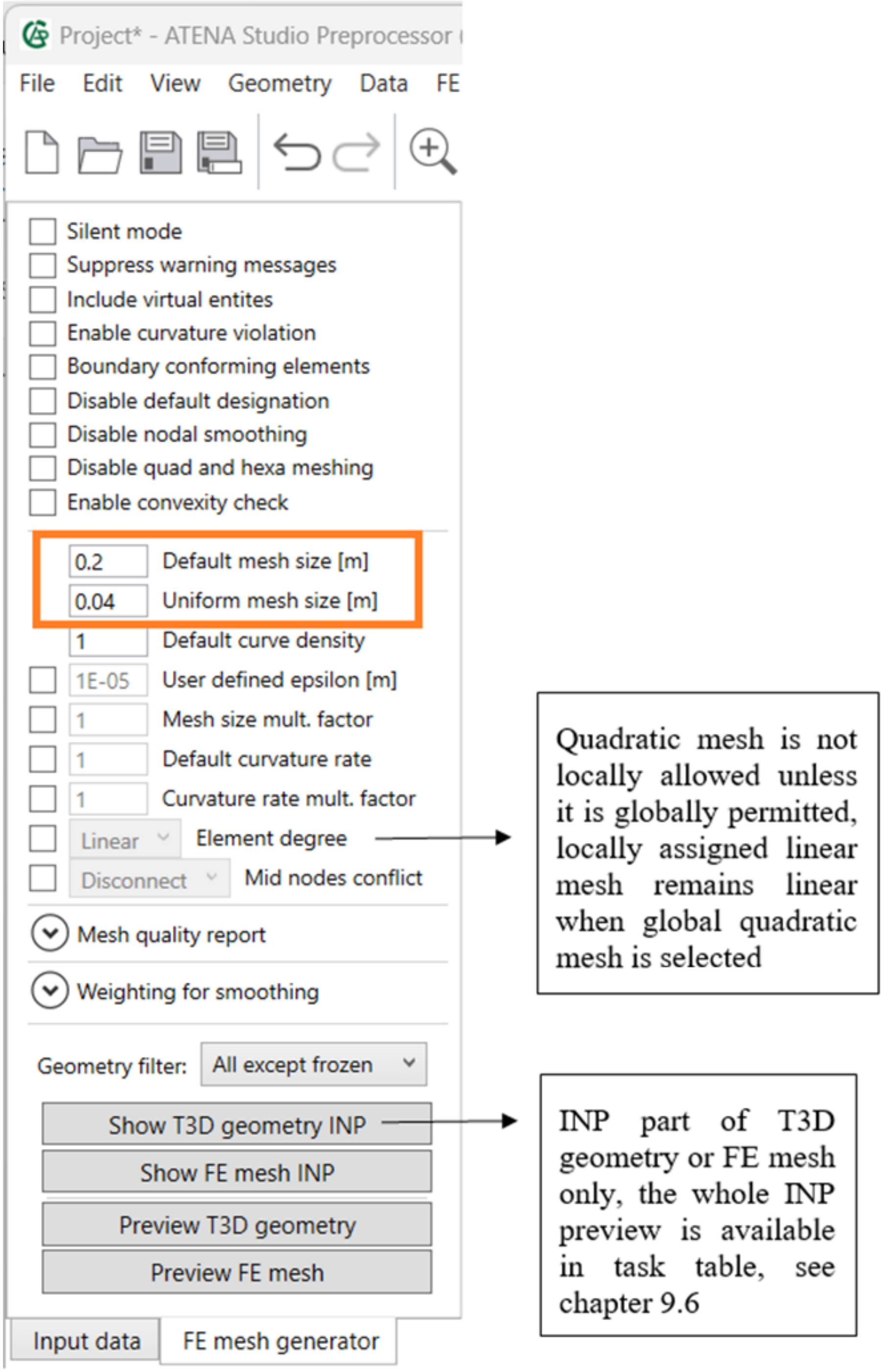
Mesh size multiplication factor multiplies the mesh size throughout the whole model so it can be useful to change the mesh size for cases in sensitivity study.

Default curvature rate controls ration between mesh size and radius of curvature. It is assigned at places without local curvature specifications.

Curvature rate multiplication factor multiplies the curvature rate within whole model so it can be used for sensitivity analysis same way as mesh size multiplication factor.

Element degree defines the global mesh as either linear or quadratic. Quadratic mesh is not locally allowed unless it is globally permitted. So linear global degree should be used only when no local quadratic mesh is assigned.

Mid node conflicts solve the local problems where linear and quadratic mesh meet because the quadratic elements contain extra mid node. When user combines both element degrees in one model, the joints between these meshes should be properly studied in postprocessing. Similarly like incorrectly assigned master-slave connection, the meshed deformation in these joints would exhibit gaps or distorted deformation that should not be there. Then three solutions to repair the mesh connection between linear and quadratic mesh are available: disconnect mid nodes, disconnect and fix, or disconnect and erase. The chosen solution can affect the concentrated loads in mid nodes so for more details please read the ATENA Input File Format in [4] (Chapter: The Command &REMOVE_NLIN_NODES_POLICY).



□ Silent mode
□ Suppress warning messages
□ Include virtual entites
□ Enable curvature violation
□ Boundary conforming elements
□ Disable default designation
□ Disable nodal smoothing
□ Disable quad and hexa meshing
□ Enable convexity check

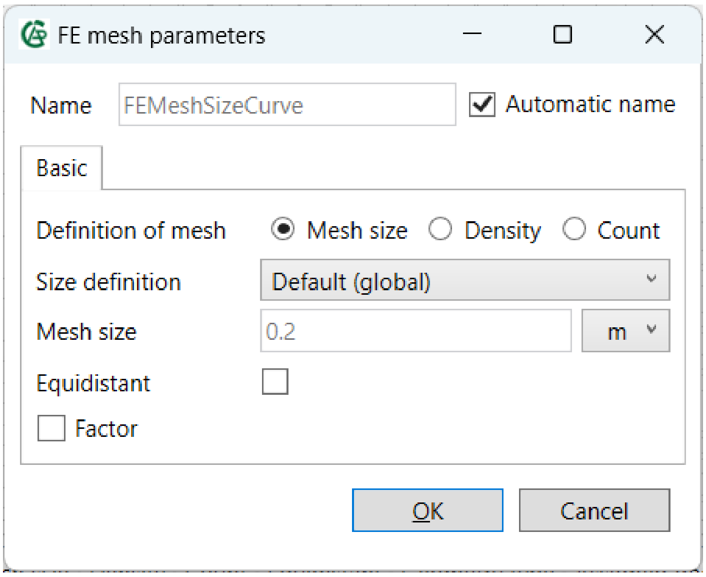
0.2 Default mesh size [m]
0.04 Uniform mesh size [m]
1 Default curve density
□ 1E-05 User defined epsilon [m]
□ 1 Mesh size mult. factor
□ 1 Default curvature rate
□ 1 Curvature rate mult. factor
□ Linear Element degree
□ Disconnect Mid nodes conflict

Geometry filter: All except frozen

Quadratic mesh is not locally allowed unless it is globally permitted, locally assigned linear mesh remains linear when global quadratic mesh is selected

INP part of T3D geometry or FE mesh only, the whole INP preview is available in task table, see chapter 9.6

Fig. 8-3 FE mesh generator.



FE mesh parameters

Name FEMeshSizeCurve ☑ Automatic name

Definition of mesh ● Mesh size ○ Density ○ Count

Size definition: Default (global)

Mesh size: 0.2 m

Equidistant □

□ Factor

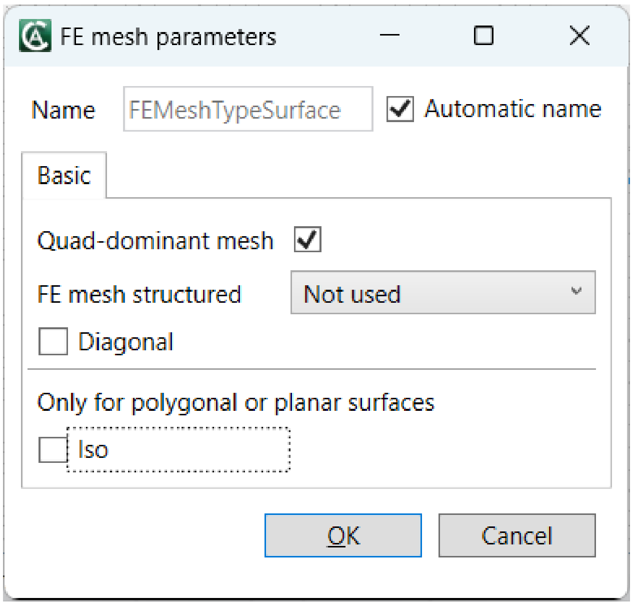
Fig. 8-4 FE mesh parameters – Size for curves.

8.2 Mesh type

Following mesh settings is **mesh type** for surfaces and solids Fig. 8-5 and Fig. 8-6. Basic option for surface mesh specifies quad-dominant mesh and whether the mesh is structured or not. When mesh is quad-dominant without structured attribute, the mesh becomes quadrilateral where possible and the rest has triangular elements. Structured mesh is usually represented by orthogonal quadrilateral elements on surfaces it is also better for identification of elements and nodes and gives mostly better results than unstructured. However some complicated geometries do not allow to use structured mesh so the mesh is unstructured with triangular elements only.

Diagonal can be used to generate structured triangular mesh by diagonal splitting each of the structured quad into two triangles. Various diagonal settings are described in [5].

Iso creates ideal equilateral elements (triangles or quadrangles) within the patch that is surrounded by unstructured mesh. The details on the Iso settings are available in T3D manual [5].



FE mesh parameters

Name FEMeshTypeSurface ☑ Automatic name

Quad-dominant mesh ☑

FE mesh structured: Not used

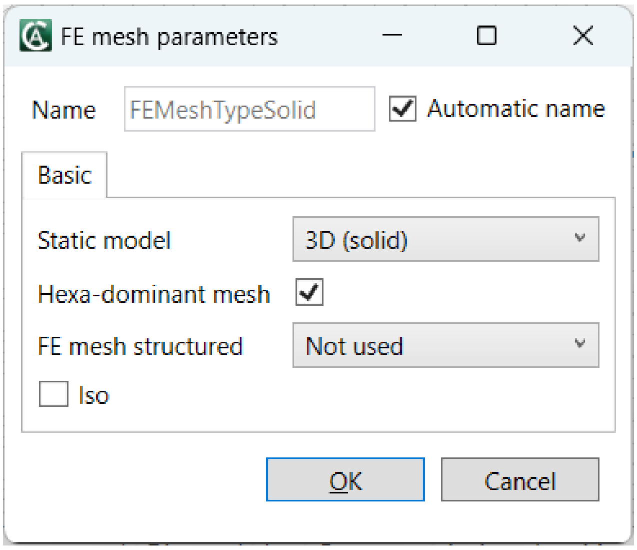
□ Diagonal

Only for polygonal or planar surfaces

□ Iso

Fig. 8-5 FE mesh parameters – Mesh type for surface.

In case of solids, the static model is specified once more to ensure compatibility with BE types. Then the hexa-dominant mesh combines the tetrahedra, wedges and hexahedra. While structured mesh uses hexahedra and unstructured mesh tetrahedra.


FE mesh parameters

Name FEMeshTypeSolid ☑ Automatic name

Basic

Static model: 3D (solid)

Hexa-dominant mesh ☑

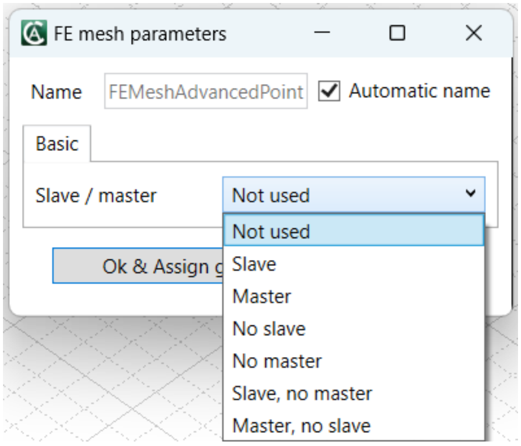
FE mesh structured: Not used

□ Iso

Fig. 8-6 FE mesh parameters – Mesh type for solid.

8.3 Mesh advanced

Section of **FE mesh parameters advanced** is dedicated to connecting of model part that contain incompatible mesh that might be given by different element shape or type, basically mostly at places where finite element nodes are not matching on the connecting point/line/surface, Fig. 8-7. This connection does not need to be duplicated by fixed contact in boundary conditions.


FE mesh parameters

Name FEMeshAdvancedPoint ☑ Automatic name

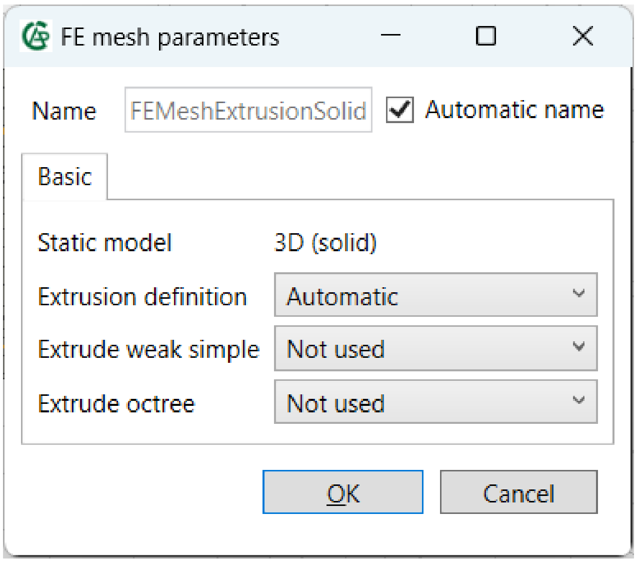
Basic

Slave / master: Not used

Fig. 8-7 FE mesh parameters – Master-slave connections.

8.4 Mesh extrusion

Last parameter in FE mesh input tree is **extrusion**. It is possible to create semi-structured mesh with hexahedra and wedges within solid that has two opposite topologically identical surfaces which are quad mappable, see Fig. 8-8. Default settings of extrusion work automatically while advanced setting might be required when the opposite surfaces are not completely identical, in such a case follow recommendations in T3D mesh generator manual, [5] where can be found more details on FE mesh setting in ATENA-PRE.



Name FEMeshExtrusionSolid ☑ Automatic name

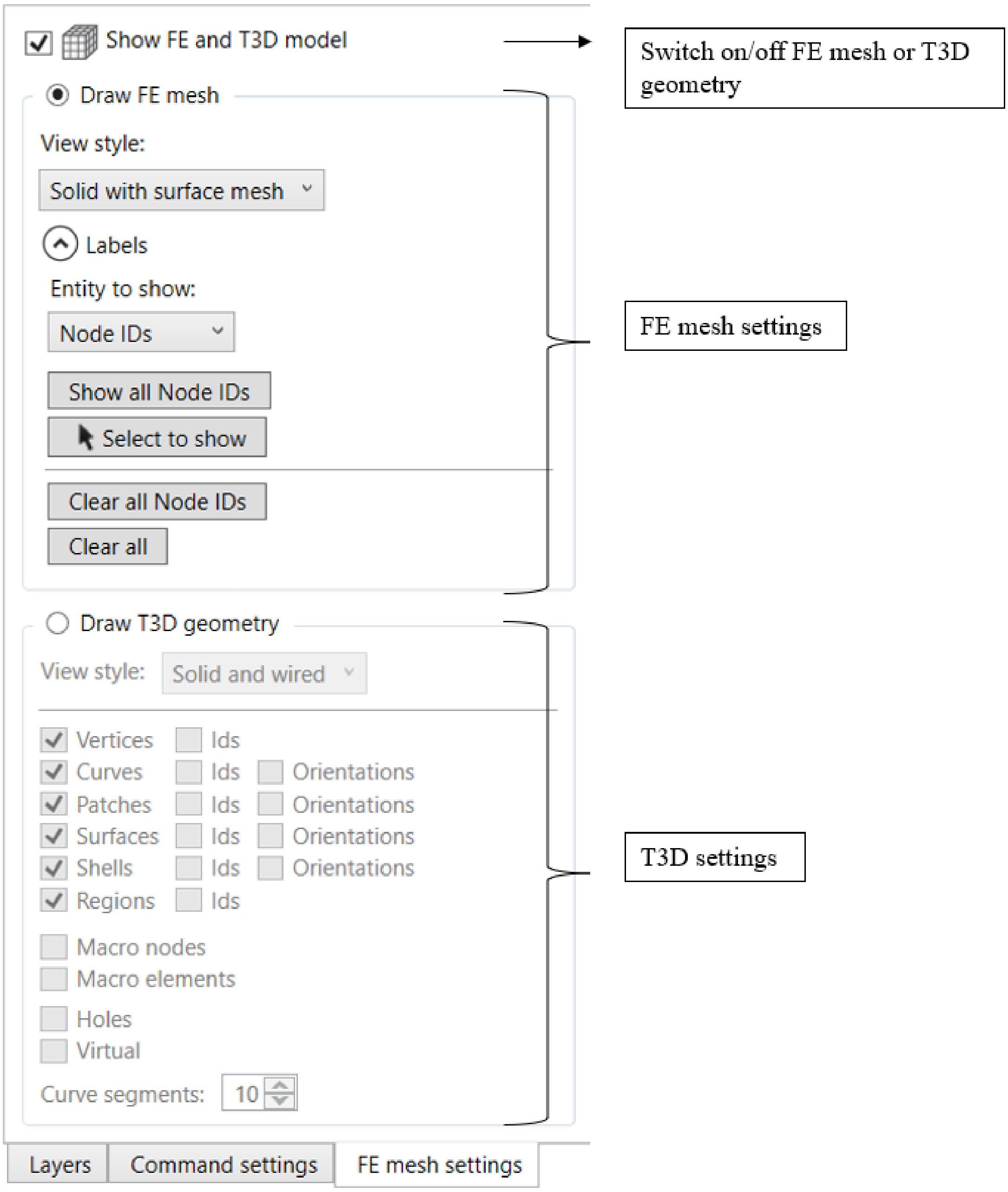
Static model 3D (solid)

Extrusion definition Automatic

Extrude weak simple Not used

Extrude octree Not used

Fig. 8-8 FE mesh parameters – Extrusion.



☑ Show FE and T3D model

● Draw FE mesh

View style: Solid with surface mesh

Labels

Entity to show: Node IDs

○ Draw T3D geometry

View style: Solid and wired

☑ Vertices □ Ids
☑ Curves □ Ids □ Orientations
☑ Patches □ Ids □ Orientations
☑ Surfaces □ Ids □ Orientations
☑ Shells □ Ids □ Orientations
☑ Regions □ Ids

□ Macro nodes
□ Macro elements

□ Holes
□ Virtual

Curve segments: 10

Switch on/off FE mesh or T3D geometry

FE mesh settings

T3D settings

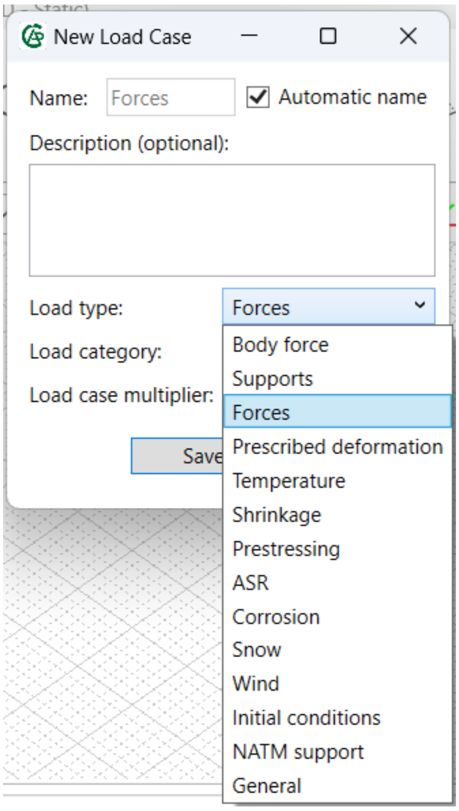
Fig. 8-9 FE mesh settings in workspace view.

9 ANALYSIS

When geometry of the model has assigned material and all necessary parameters of FE mesh, analysis data can be created. Analysis data are divided into categories: Load cases, Selections, Boundary conditions, Monitors, Solution parameters, Tasks and Functions.

9.1 Load cases

Load cases serve for creating the load types (forces, shrinkage, prestressing etc.) acting on the geometrical entity including supports, see Fig. 9-1. Defined load cases are listed in table as is shown in Fig. 9-2 and this list of Load cases appears in boundary conditions definition to set the details (values, direction etc.) and in Tasks to create the loading history of the whole analysis.



New Load Case

Name: Forces ☑ Automatic name

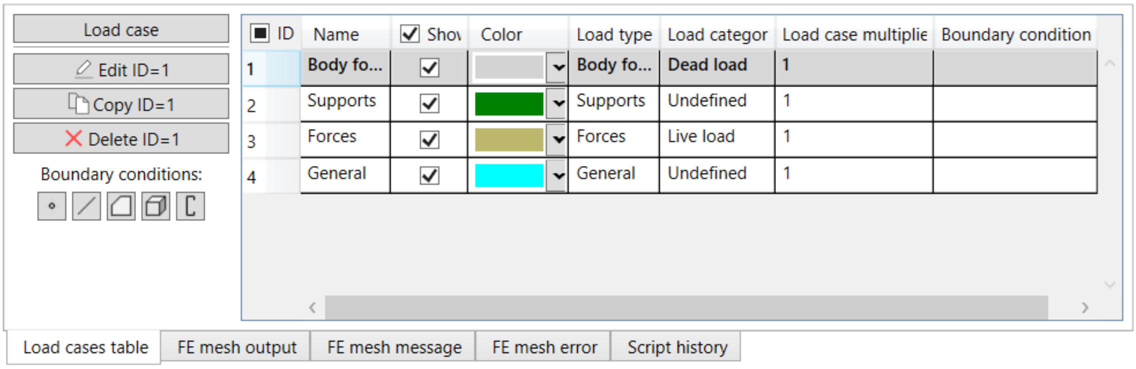
Description (optional):

Load type: Forces

Load category:

Load case multiplier:

Fig. 9-1 Load cases – Load types.



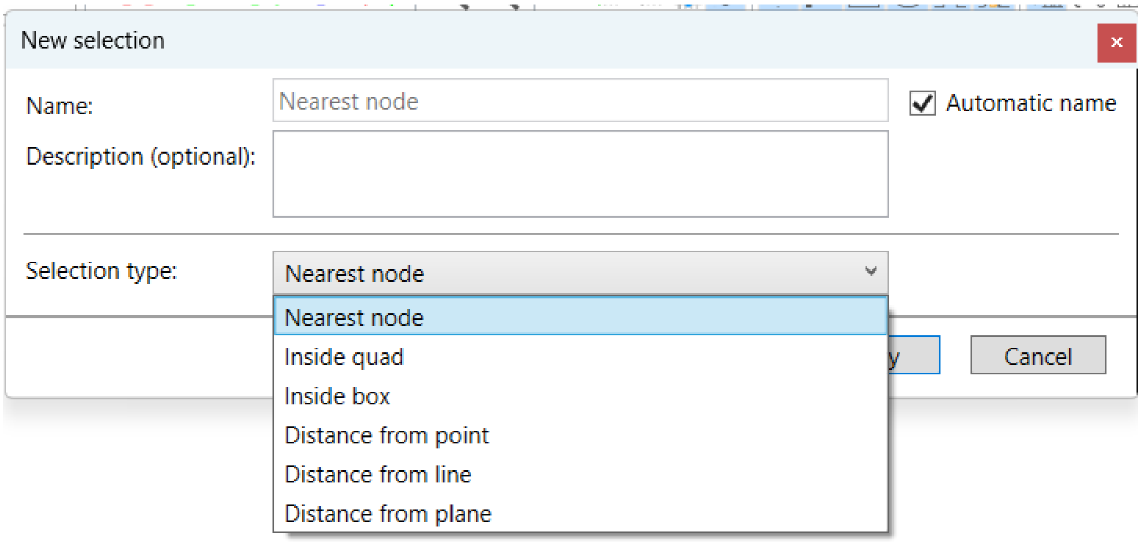
ID	Name	Shov	Color	Load type	Load categor	Load case multiplie	Boundary condition
1	**Body fo...**	☑	⬜	**Body fo...**	**Dead load**	**1**	
2	Supports	☑	🟩	Supports	Undefined	1	
3	Forces	☑	🟨	Forces	Live load	1	
4	General	☑	🟦	General	Undefined	1	

Fig. 9-2 Load cases – Table of specifications.

9.2 Selections

Selections are collection of entities that can be subsequently used for preprocessing (assigning BC, monitors etc.) or for postprocessing in ATENA Studio (output data or activities).

The selection type creates macronodes that are not specified in geometry but user can use them for specifying conditions or outputs. The selection types are defined as nearest node, inside quad which is defined by 4 nodes, inside box which is defined by 8 nodes, by distance from point, line or plane.



New selection

Name: Nearest node

☑ Automatic name

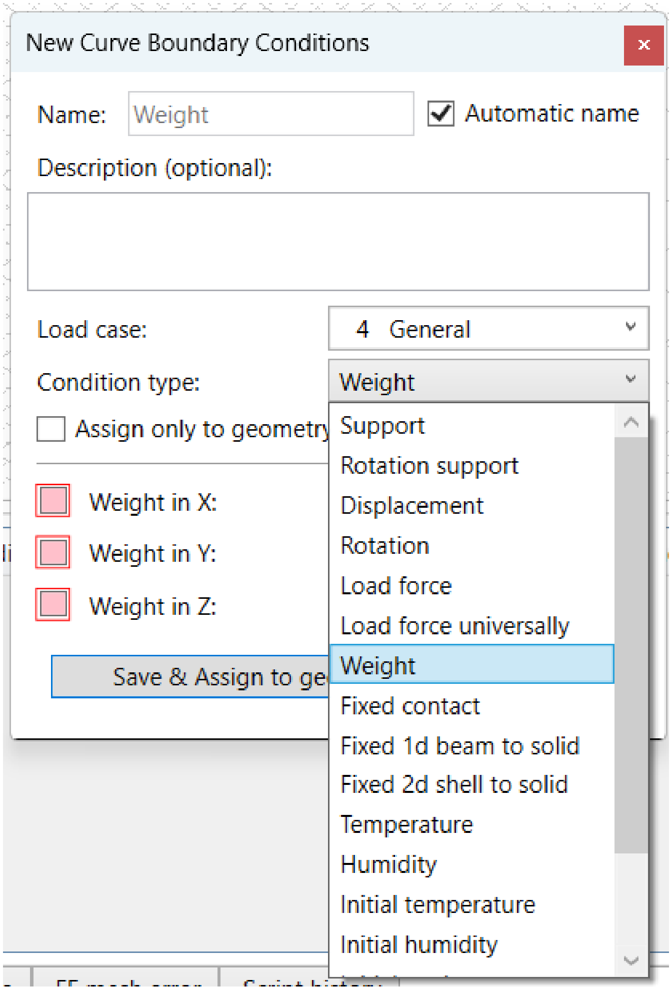
Description (optional):

Selection type: Nearest node

Fig. 9-3 Selections – Types of selections.

9.3 Boundary conditions

When the Load Cases and their load types are prepared, the specific Boundary conditions can be assigned to entities with buttons in table of specifications ∘ ╱ ▢ ⧉ [in Fig. 9-4. List of all boundary condition with description and named entities it can be assigned to is in Tab. 9-1.



New Curve Boundary Conditions

Name: Weight ☑ Automatic name

Description (optional):

Load case: 4 General

Condition type: Weight

□ Assign only to geometry

Fig. 9-4 Boundary conditions – Types of boundary conditions.

Tab. 9-1 List of boundary conditions and entities it can be assigned to

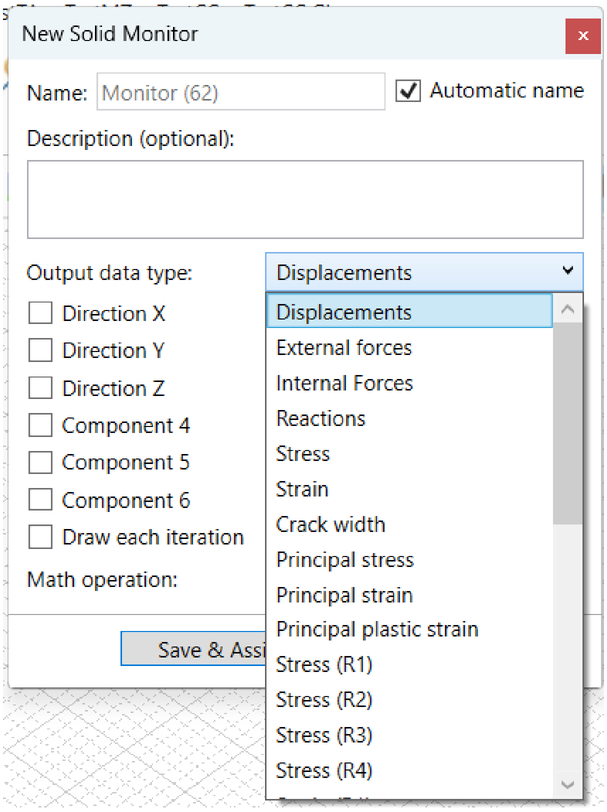
Boundary condition	Entity	Description
Support	Point Curve Surface	Constraint of entity in global or inclined coordinate system
Rotation support	Point Curve Surface	Rotation constraint for beam or shell element entities in global or inclined coordinate system
Displacement	Point Curve Surface	Prescribed displacement in global coordinate system
Rotation	Point Curve Surface	Prescribed rotation for beam and shell element entities in global coordinate system
Load force	Point Curve	Load force for point can be defined by three components in each coordinate direction. The loading for line can be prescribed only for 2D

	Surface	elements. Local coordinate system can be used to apply loading normal to the line. The projection can be used, for example, for the snow or wind load. The loading can be constant or linear. The load force for surface can be obviously defined only for 3D entities. The possible coordinate systems options are similar to the line condition.
Load force universally	Curve Surface	Load can be applied to a part of a line or surface. When entering the force magnitudes for each component, it is possible to select suitable units. You can apply this condition to any line or surface in the model. If you don’t want to mesh this entity, and you want to use the line or surface only to apply the load, you can assign a material from book Dummy to this entity. The number A DIV and B DIV define into how many parts the element will be split for the forces application.
Weight	Curve Surface Solid Reinforcement	Body weight can be applied in direction of global coordinates.
Fixed contact	Point Curve Surface Solid	Fixed contact connects either duplicated entities or parts with incompatible mesh. The Master (larger elements) and Slave (smaller elements) entities have to have identical names in order to be connected.
Fixed 1d beam to solid	Point Curve Surface Solid	Fixed contact that can connect solid with 3 DOFs with beam that has 6 DOFs
Fixed 2d shell to solid	Point Curve Surface Solid	Fixed contact that can connect solid with 3 DOFs with shell that has 5 DOFs
Temperature	Curve Surface Solid Reinforcement	Temperature increment that is constant or linear gradient for static analysis
Humidity	Curve Surface Solid	Humidity increment that is constant or linear gradient for static analysis
Initial temperature	Curve Surface Solid Reinforcement	Temperature increment that is constant or linear gradient for static analysis in first interval
Initial humidity	Curve	Humidity increment that is constant or linear

	Surface Solid	gradient for static analysis in first interval
Initial strain	Curve Surface Solid	This condition is used for simulation of pre-stressing or shrinkage. Any shortening is with negative value and elongation with positive value. Losses of pre-stress might need to be compensated for elastic deformation of concrete.
Initial stress	Curve Surface Solid	Another way to simulate pre-stressing. Initial stress remains constant and might simulate situation when the cable is repeatedly post-tensioned to compensate losses. The positive value represents tension.
Global shell nodes	Point Curve Surface	This condition specifies list of finite element joint which degrees of freedom are treated in element local coordinate system
Chlorides	Surface	Boundary condition to apply effect of chlorides on concrete, for more details on settings see ATENA Theory chapter 6.2
Carbonation	Surface	Boundary condition to apply effect of carbonation on concrete, for more details on settings see ATENA Theory chapter 6.1
ASR	Surface Solid	Boundary condition to apply effect of alkali-silica reaction on concrete, for more details on settings see ATENA Theory chapter 6.5
Reinforcement initial strain	Reinforcement	Initial strain for reinforcement is applicable on 1D elements only. It can be constant on the entity or it can be defined by function dependent on the location on entity.
Prestressing	Reinforcement	By this condition, you can define the prestressing of the reinforcement, see also ATENA Troubleshooting [9], 2.2.7 How can I model prestressing losses.

9.4 Monitors

Then there are Monitors which are very helpful for evaluation of the whole analysis because they record various result values which can be displayed in load-displacement diagrams or similar. Monitors for various types of load or reactions of the structure resulting from acting load are available in table of specifications for different geometrical entities with buttons ∘ ╱ ▢ ⧉, Fig. 9-5. The direction or component of chosen data type should be selected and on entities with more than one node the math operation can be provided such as summation, maximum, minimum, or average.



New Solid Monitor

Name: Monitor (62) ☑ Automatic name

Description (optional):

Output data type: Displacements

☐ Direction X

☐ Direction Y

☐ Direction Z

☐ Component 4

☐ Component 5

☐ Component 6

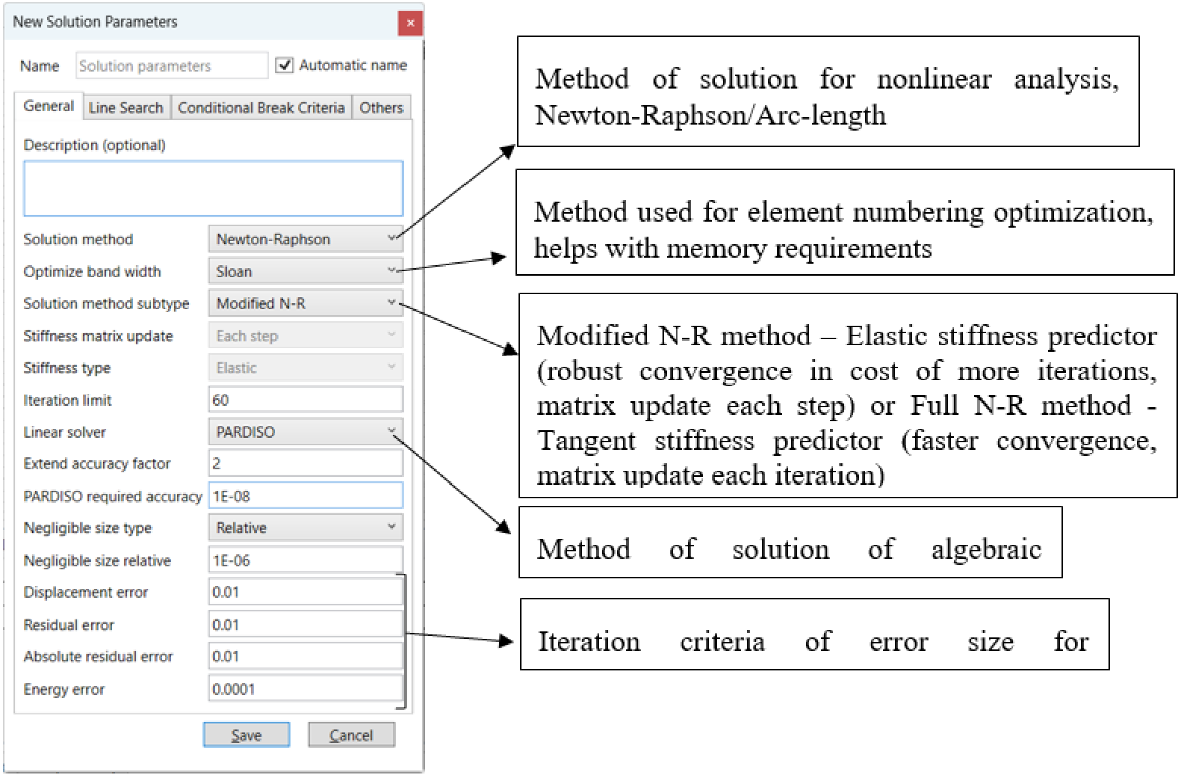
☐ Draw each iteration

Math operation:

Fig. 9-5 Monitors – Data types.

9.5 Solution parameters

Nonlinear analyses in ATENA generally offer two different type of solution method which are Newton-Raphson and Arc-length method, in Fig. 9-6 and Fig. 9-7. Both methods have their strength and limitation therefore the proper one needs to be set to find the correct solution, more information on methods and their detailed settings is available in ATENA Theory [1]. The settings of method and other solution parameters are within the Solution parameters. User can define various combinations of solution parameters that are used within different Intervals of analysis. List of Solution parameters is shown in Solution parameters table, see Fig. 9-8.



Name: Solution parameters ☑ Automatic name

Description (optional)

Solution method: Newton-Raphson

Optimize band width: Sloan

Solution method subtype: Modified N-R

Stiffness matrix update: Each step

Stiffness type: Elastic

Iteration limit: 60

Linear solver: PARDISO

Extend accuracy factor: 2

PARDISO required accuracy: 1E-08

Negligible size type: Relative

Negligible size relative: 1E-06

Displacement error: 0.01

Residual error: 0.01

Absolute residual error: 0.01

Energy error: 0.0001

Method of solution for nonlinear analysis, Newton-Raphson/Arc-length

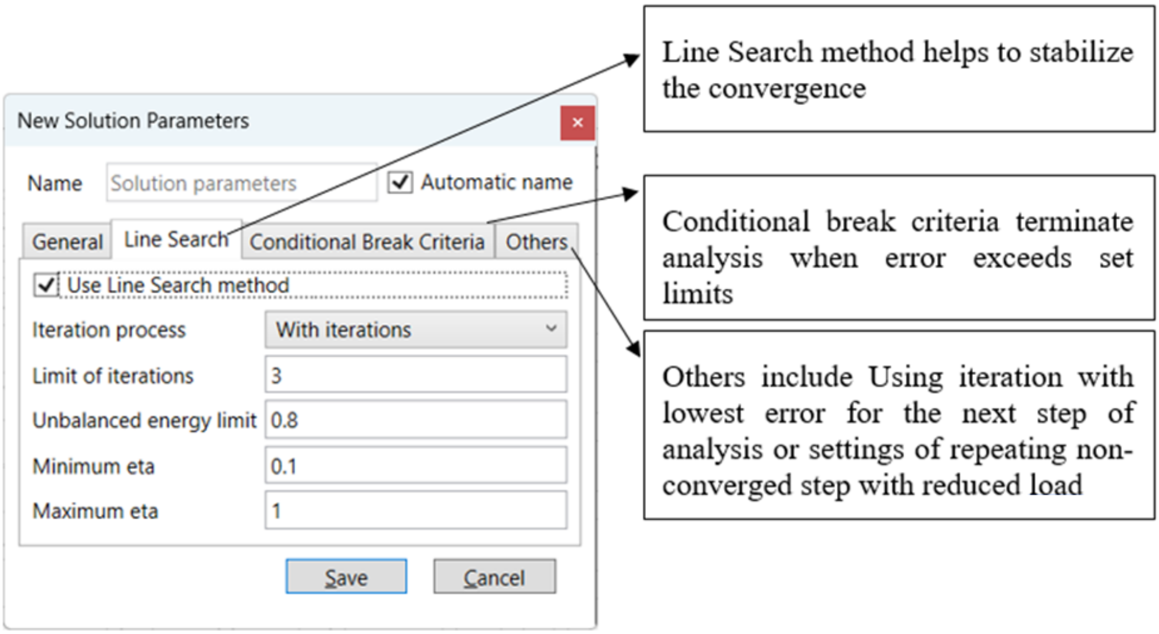
Method used for element numbering optimization, helps with memory requirements

Modified N-R method – Elastic stiffness predictor (robust convergence in cost of more iterations, matrix update each step) or Full N-R method - Tangent stiffness predictor (faster convergence, matrix update each iteration)

Method of solution of algebraic

Iteration criteria of error size for

Fig. 9-6 Solution parameters – Settings of solution parameters.



Name: Solution parameters ☑ Automatic name

☑ Use Line Search method

Iteration process: With iterations

Limit of iterations: 3

Unbalanced energy limit: 0.8

Minimum eta: 0.1

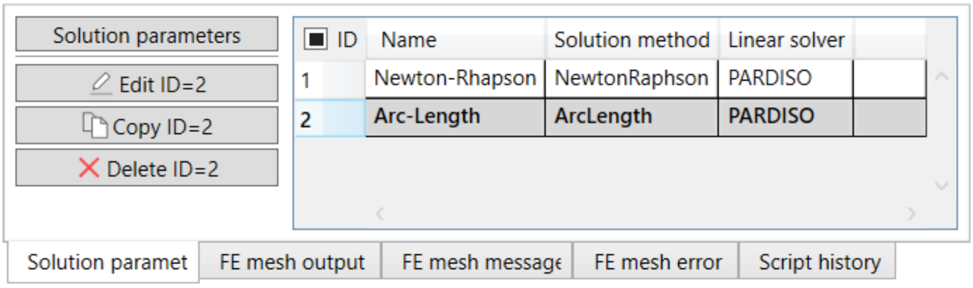
Maximum eta: 1

Line Search method helps to stabilize the convergence

Conditional break criteria terminate analysis when error exceeds set limits

Others include Using iteration with lowest error for the next step of analysis or settings of repeating non-converged step with reduced load

Fig. 9-7 Solution parameters – Line Search.



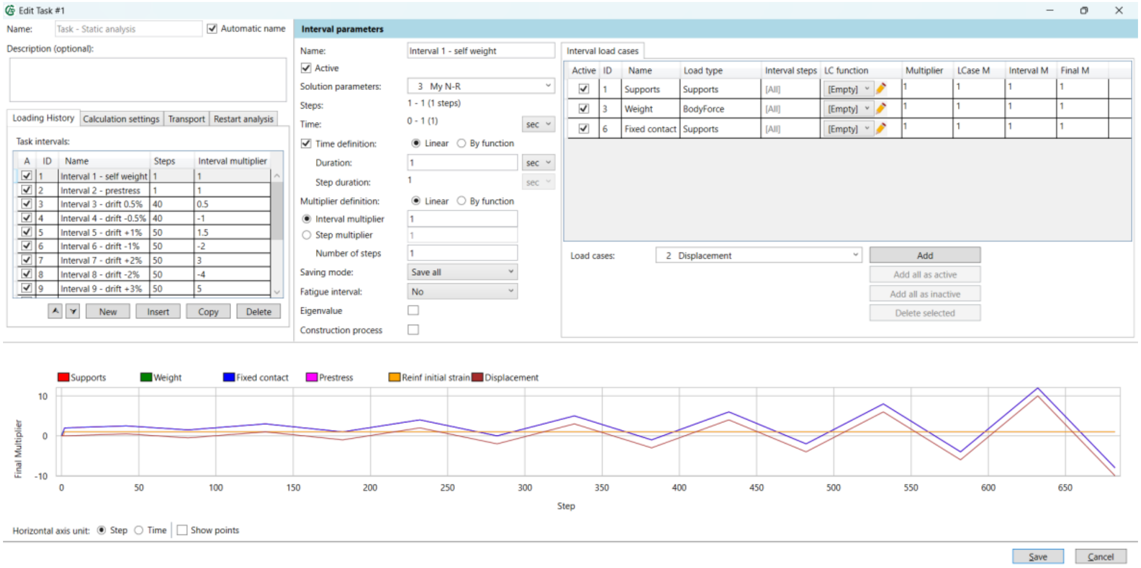
ID	Name	Solution method	Linear solver
1	Newton-Rhapson	NewtonRaphson	PARDISO
2	**Arc-Length**	**ArcLength**	**PARDISO**

Fig. 9-8 Solution parameters table.

9.6 Tasks

The main part of Analysis section is Task, there all the previous parts come together and the loading history for the analysis can be created. Within Loading history separate intervals are defined and their specifics are adjustable in Interval parameters. The solution parameters and combination of load cases are defined for each interval separately including the number of steps, interval multiplier etc. Finally, the diagram of load cases action is at the bottom of Tasks settings, see Fig. 9-10. Different loading history can be defined in another Task.

Once the loading history and solution of analysis is set the INP file can be created or just previewed, or the analysis can be executed with Run button, see Fig. 9-11.



Name: Task - Static analysis ☑ Automatic name

Description (optional):

Task intervals:

A	ID	Name	Steps	Interval multiplier
☑	1	Interval 1 - self weight	1	1
☑	2	Interval 2 - prestress	1	1
☑	3	Interval 3 - drift 0.5%	40	0.5
☑	4	Interval 4 - drift -0.5%	40	-1
☑	5	Interval 5 - drift +1%	50	1.5
☑	6	Interval 6 - drift -1%	50	-2
☑	7	Interval 7 - drift +2%	50	3
☑	8	Interval 8 - drift -2%	50	-4
☑	9	Interval 9 - drift +3%	50	5

Name: Interval 1 - self weight

☑ Active

Solution parameters: 3 My N-R

Steps: 1 - 1 (1 steps)

Time: 0 - 1 (1) sec

☑ Time definition: ● Linear ○ By function

Duration: 1 sec

Step duration: 1 sec

Multiplier definition: ● Linear ○ By function

● Interval multiplier 1

○ Step multiplier 1

Number of steps 1

Saving mode: Save all

Fatigue interval: No

Eigenvalue □

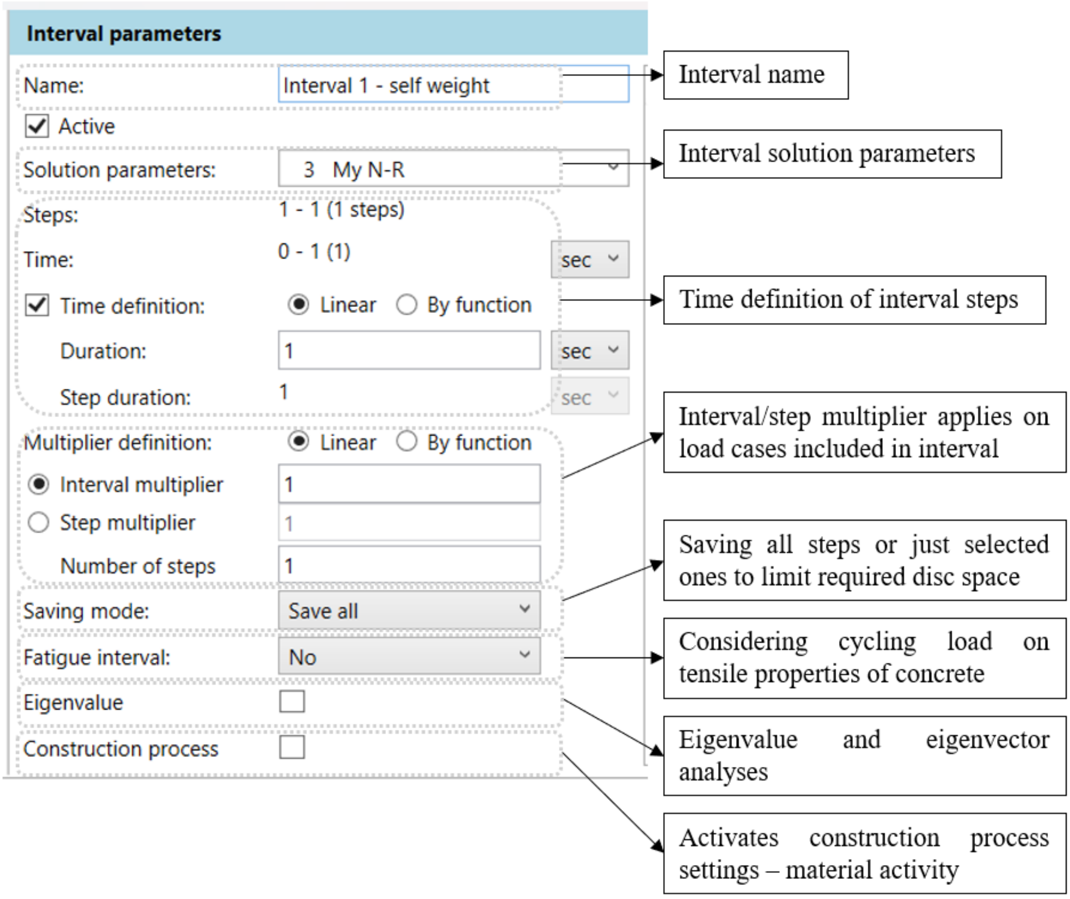
Construction process □

Active	ID	Name	Load type	Interval steps	LC function	Multiplier	LCase M	Interval M	Final M
☑	1	Supports	Supports	[All]	[Empty]	1	1	1	1
☑	3	Weight	BodyForce	[All]	[Empty]	1	1	1	1
☑	6	Fixed contact	Supports	[All]	[Empty]	1	1	1	1

Load cases: 2 Displacement

Horizontal axis unit: ● Step ○ Time □ Show points

Fig. 9-9 Tasks – Creating the loading history.



Interval parameters

Name: Interval 1 - self weight

☑ Active

Solution parameters: 3 My N-R

Steps: 1 - 1 (1 steps)

Time: 0 - 1 (1) sec

☑ Time definition: ● Linear ○ By function

Duration: 1 sec

Step duration: 1 sec

Multiplier definition: ● Linear ○ By function

● Interval multiplier 1

○ Step multiplier 1

Number of steps 1

Saving mode: Save all

Fatigue interval: No

Eigenvalue □

Construction process □

Interval name

Interval solution parameters

Time definition of interval steps

Interval/step multiplier applies on load cases included in interval

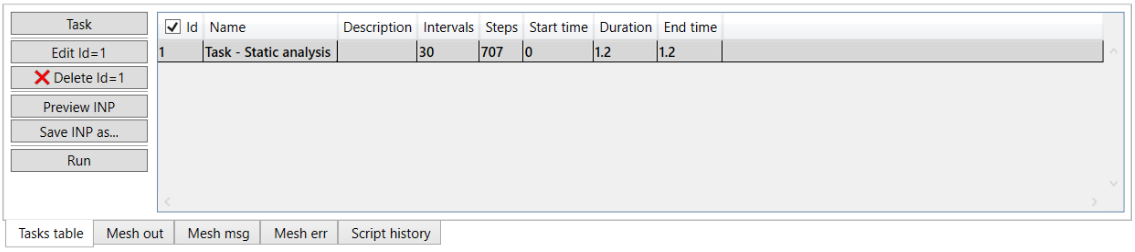
Saving all steps or just selected ones to limit required disc space

Considering cycling load on tensile properties of concrete

Eigenvalue and eigenvector analyses

Activates construction process settings – material activity

Fig. 9-10 Tasks – Creating the loading history.

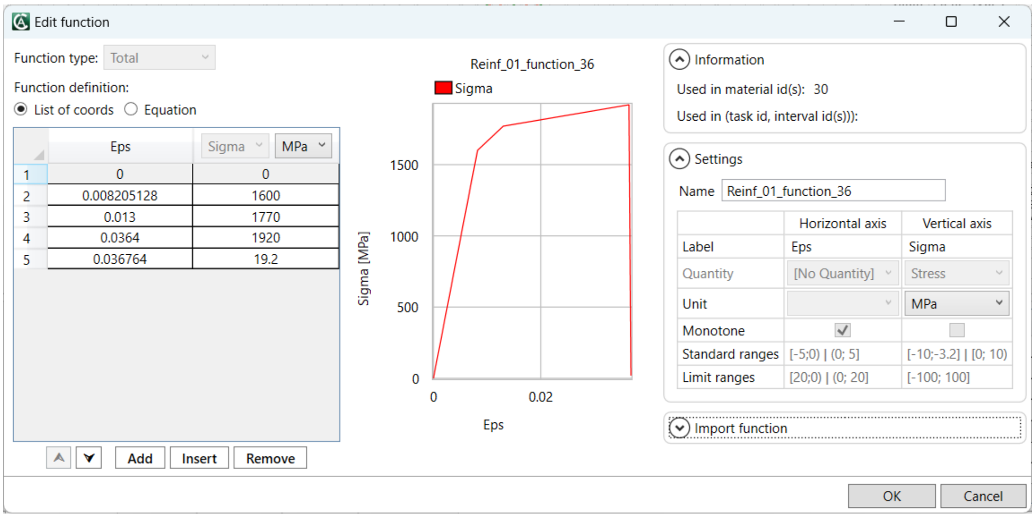


☑ Id	Name	Description	Intervals	Steps	Start time	Duration	End time
1	Task - Static analysis		30	707	0	1.2	1.2

Fig. 9-11 Tasks – INP preview/save, Run of analysis.

9.7 Functions

Finally, there is the section with defined and saved Functions. They can be found all the general functions that are utilized for the whole model. For example, the function of material laws like strass-strain diagrams, material properties dependent on time/temperature/radiation, function that defines application of load during the loading history and many others, example of function dialog in Fig. 9-12.



Edit function

Function type: Total

Function definition:

● List of coords ○ Equation

Eps	Sigma	MPa
1	0	0
2	0.008205128	1600
3	0.013	1770
4	0.0364	1920
5	0.036764	19.2

Information

Used in material id(s): 30

Used in (task id, interval id(s))):

Settings

Name Reinf_01_function_36

	Horizontal axis	Vertical axis
Label	Eps	Sigma
Quantity	[No Quantity]	Stress
Unit		MPa
Monotone	☑	□
Standard ranges	[-5;0) | (0; 5]	[-10;-3.2] | [0; 10)
Limit ranges	[20;0) | (0; 20]	[-100; 100]

Import function

Fig. 9-12 Functions – Definition of function.

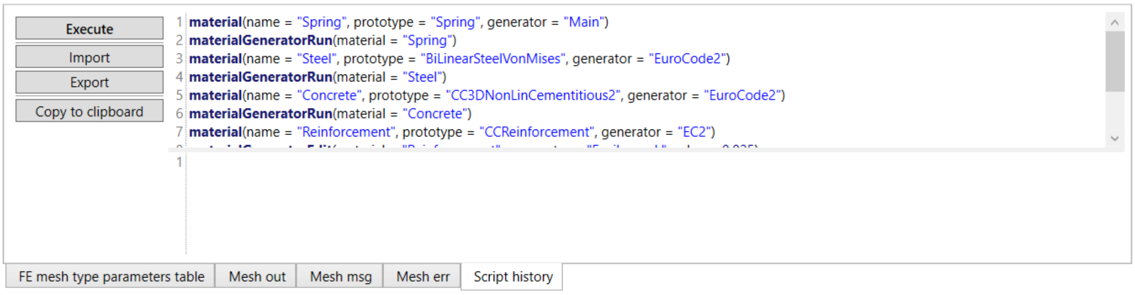
10 EXAMPLES OF SCRIPTING IN ATENA PREPROCESSOR

10.1 ATENA script of spiral reinforcement

Scripting of geometry is great advantage to create complicated geometry that can be expressed mathematically. Example of such scripts and its visual results are presented bellow.

The scripting method is especially useful for instance for the parametric modelling of standard structural elements such as for instance the spiral reinforcement as indicated below.

Working with scripts is enabled in table of specification at the last tab **Script history** in Fig. 10-1. The script can be written and executed directly in Script history or user can import or export existing scripts.



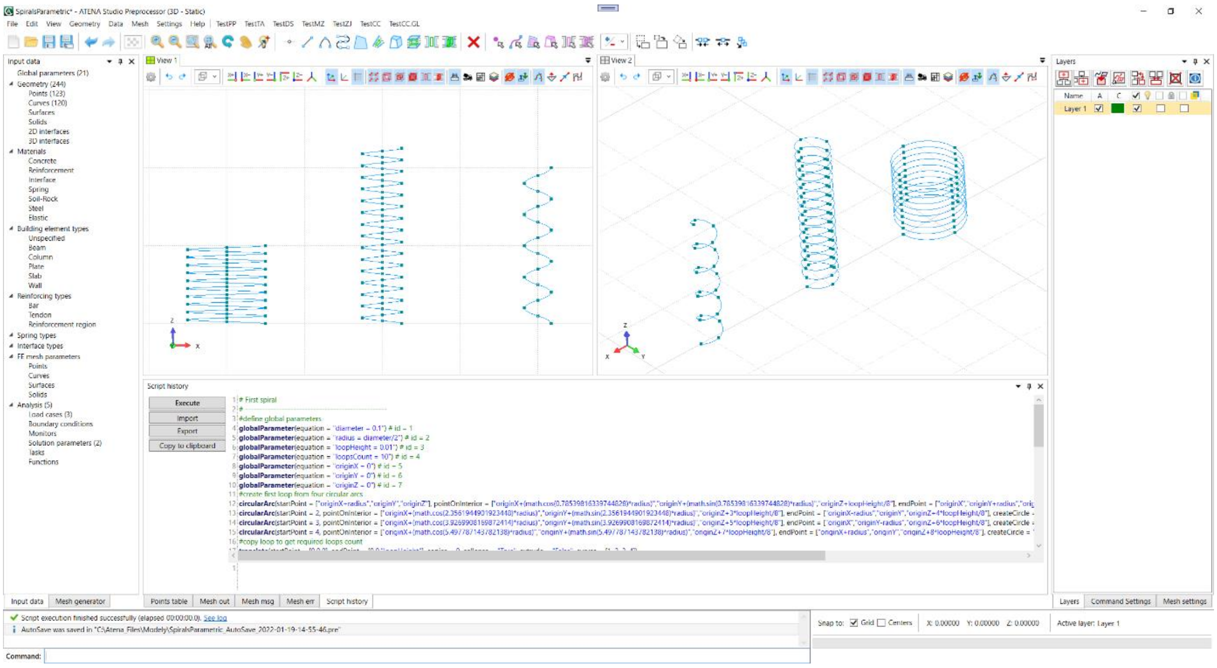
```
material(name = "Spring", prototype = "Spring", generator = "Main")
materialGeneratorRun(material = "Spring")
material(name = "Steel", prototype = "BiLinearSteelVonMises", generator = "EuroCode2")
materialGeneratorRun(material = "Steel")
material(name = "Concrete", prototype = "CC3DNonLinCementitious2", generator = "EuroCode2")
materialGeneratorRun(material = "Concrete")
material(name = "Reinforcement", prototype = "CCReinforcement", generator = "EC2")
```

Fig. 10-1 Script history.

EXAMPLE 1 – Spiral reinforcement, Fig. 10-2.

```
# First spiral

# ---------------------------------------------------

#define global parameters

globalParameter(equation = "diameter = 0.1") # id = 1

globalParameter(equation = "radius = diameter / 2") # id = 1

globalParameter(equation = "loopHeight = 0.01") # id = 2

globalParameter(equation = "loopsCount = 10") # id = 3

globalParameter(equation = "originX = 0") # id = 4

globalParameter(equation = "originY = 0") # id = 5

globalParameter(equation = "originZ = 0") # id = 6


#create first loop from four circular arcs

circularArc(startPoint = ["originX + radius", "originY", "originZ"], pointOnInterior = ["originX +
(math.cos(0.78539816339744828) * radius)", "originY + (math.sin(0.78539816339744828) * radius)",
"originZ + loopHeight / 8"], endPoint = ["originX", "originY + radius", "originZ + 2 * loopHeight /
8"], createCircle = "False", createCircularSurface = "False") # id = 2

circularArc(startPoint = 2, pointOnInterior = ["originX + (math.cos(2.3561944901923448) * radius)",
"originY + (math.sin(2.3561944901923448) * radius)", "originZ + 3 * loopHeight / 8"], endPoint =
["originX - radius", "originY", "originZ + 4 * loopHeight / 8"], createCircle = "False",
createCircularSurface = "False") # id = 2

circularArc(startPoint = 3, pointOnInterior = ["originX + (math.cos(3.9269908169872414) * radius)",
"originY + (math.sin(3.9269908169872414) * radius)", "originZ + 5 * loopHeight / 8"], endPoint =
["originX", "originY - radius", "originZ + 6 * loopHeight / 8"], createCircle = "False",
createCircularSurface = "False") # id = 2

circularArc(startPoint = 4, pointOnInterior = ["originX + (math.cos(5.497787143782138) * radius)",
"originY + (math.sin(5.497787143782138) * radius)", "originZ + 7 * loopHeight / 8"], endPoint =
["originX + radius", "originY", "originZ + 8 * loopHeight / 8"], createCircle = "False",
createCircularSurface = "False") # id = 2

#copy loop to get required loops count
```

```
translate(startPoint = [0,0.0,0.0], endPoint = [0,0.0,"loopHeight"], copies = "loopsCount - 1",
collapse = "True", extrude = "False", curves = [1,2,3,4])

# --------------------------------------------------

# Second spiral

# --------------------------------------------------

#define global parameters

globalParameter(equation = "diameter2 = 0.05") # id = 1

globalParameter(equation = "radius2 = diameter2 / 2") # id = 1

globalParameter(equation = "loopHeight2 = 0.015") # id = 2

globalParameter(equation = "loopsCount2 = 15") # id = 3

globalParameter(equation = "originX2 = 0.2") # id = 4

globalParameter(equation = "originY2 = 0") # id = 5

globalParameter(equation = "originZ2 = 0") # id = 6


#create first loop from four circular arcs

circularArc(startPoint = ["originX2 + radius2", "originY2", "originZ2"], pointOnInterior = ["originX2
+ (math.cos(0.78539816339744828) * radius2)", "originY2 + (math.sin(0.78539816339744828) * radius2)",
"originZ2 + loopHeight2 / 8"], endPoint = ["originX2", "originY2 + radius2", "originZ2 + 2 *
loopHeight2 / 8"], createCircle = "False", createCircularSurface = "False") # id = 2

circularArc(startPoint = 52, pointOnInterior = ["originX2 + (math.cos(2.3561944901923448) *
radius2)", "originY2 + (math.sin(2.3561944901923448) * radius2)", "originZ2 + 3 * loopHeight2 / 8"],
endPoint = ["originX2 - radius2", "originY2", "originZ2 + 4 * loopHeight2 / 8"], createCircle =
"False", createCircularSurface = "False") # id = 2

circularArc(startPoint = 53, pointOnInterior = ["originX2 + (math.cos(3.9269908169872414) *
radius2)", "originY2 + (math.sin(3.9269908169872414) * radius2)", "originZ2 + 5 * loopHeight2 / 8"],
endPoint = ["originX2", "originY2 - radius2", "originZ2 + 6 * loopHeight2 / 8"], createCircle =
"False", createCircularSurface = "False") # id = 2

circularArc(startPoint = 54, pointOnInterior = ["originX2 + (math.cos(5.497787143782138) * radius2)",
"originY2 + (math.sin(5.497787143782138) * radius2)", "originZ2 + 7 * loopHeight2 / 8"], endPoint =
["originX2 + radius2", "originY2", "originZ2 + 8 * loopHeight2 / 8"], createCircle = "False",
createCircularSurface = "False") # id = 2

#copy loop to get required loops count

translate(startPoint = [0,0.0,0.0], endPoint = [0,0.0,"loopHeight2"], copies = "loopsCount2 - 1",
collapse = "True", extrude = "False", curves = [41,42,43,44])

# --------------------------------------------------

# Third spiral

# --------------------------------------------------

#define global parameters

globalParameter(equation = "diameter3 = 0.035") # id = 1

globalParameter(equation = "radius3 = diameter3 / 2") # id = 1

globalParameter(equation = "loopHeight3 = 0.04") # id = 2

globalParameter(equation = "loopsCount3 = 5") # id = 3

globalParameter(equation = "originX3 = 0.4") # id = 4

globalParameter(equation = "originY3 = 0") # id = 5

globalParameter(equation = "originZ3 = 0") # id = 6


#create first loop from four circular arcs

circularArc(startPoint = ["originX3 + radius3", "originY3", "originZ3"], pointOnInterior = ["originX3
+ (math.cos(0.78539816339744828) * radius3)", "originY3 + (math.sin(0.78539816339744828) * radius3)",
"originZ3 + loopHeight3 / 8"], endPoint = ["originX3", "originY3 + radius3", "originZ3 + 2 *
loopHeight3 / 8"], createCircle = "False", createCircularSurface = "False") # id = 2

circularArc(startPoint = 127, pointOnInterior = ["originX3 + (math.cos(2.3561944901923448) *
radius3)", "originY3 + (math.sin(2.3561944901923448) * radius3)", "originZ3 + 3 * loopHeight3 / 8"],
endPoint = ["originX3 - radius3", "originY3", "originZ3 + 4 * loopHeight3 / 8"], createCircle =
"False", createCircularSurface = "False") # id = 2

circularArc(startPoint = 128, pointOnInterior = ["originX3 + (math.cos(3.9269908169872414) *
radius3)", "originY3 + (math.sin(3.9269908169872414) * radius3)", "originZ3 + 5 * loopHeight3 / 8"],
endPoint = ["originX3", "originY3 - radius3", "originZ3 + 6 * loopHeight3 / 8"], createCircle =
"False", createCircularSurface = "False") # id = 2

circularArc(startPoint = 129, pointOnInterior = ["originX3 + (math.cos(5.497787143782138) *
radius3)", "originY3 + (math.sin(5.497787143782138) * radius3)", "originZ3 + 7 * loopHeight3 / 8"],
endPoint = ["originX3 + radius3", "originY3", "originZ3 + 8 * loopHeight3 / 8"], createCircle = "False", createCircularSurface = "False") # id = 2

#copy loop to get required loops count

translate(startPoint = [0,0.0,0.0], endPoint = [0,0.0,"loopHeight3"], copies = "loopsCount3 - 1", collapse = "True", extrude = "False", curves = [101,102,103,104])
```



```
# First spiral
# ------------------------------------------
#define global parameters
globalParameter(equation = "diameter = 0.1") # id = 1
globalParameter(equation = "radius = diameter/2") # id = 2
globalParameter(equation = "loopHeight = 0.01") # id = 3
globalParameter(equation = "loopsCount = 10") # id = 4
globalParameter(equation = "originX = 0") # id = 5
globalParameter(equation = "originY = 0") # id = 6
globalParameter(equation = "originZ = 0") # id = 7
#create first loop from four circular arcs
circularArc(startPoint = ["originX+radius","originY","originZ"], pointOnInterior = ["originX+(math.cos(0.7853981633974483)*radius)","originY+(math.sin(0.7853981633974483)*radius)","originZ+loopHeight/8"], endPoint = ["originX","originY+radius","orig
circularArc(startPoint = 2, pointOnInterior = ["originX+(math.cos(2.356194490192345)*radius)","originY+(math.sin(2.356194490192345)*radius)","originZ+3*loopHeight/8"], endPoint = ["originX-radius","originY","originZ+4*loopHeight/8"], createCircle =
circularArc(startPoint = 3, pointOnInterior = ["originX+(math.cos(3.9269908169872414)*radius)","originY+(math.sin(3.9269908169872414)*radius)","originZ+5*loopHeight/8"], endPoint = ["originX","originY-radius","originZ+6*loopHeight/8"], createCircle =
circularArc(startPoint = 4, pointOnInterior = ["originX+(math.cos(5.497787143782138)*radius)","originY+(math.sin(5.497787143782138)*radius)","originZ+7*loopHeight/8"], endPoint = ["originX+radius","originY","originZ+8*loopHeight/8"], createCircle =
#copy loop to get required loops count
```

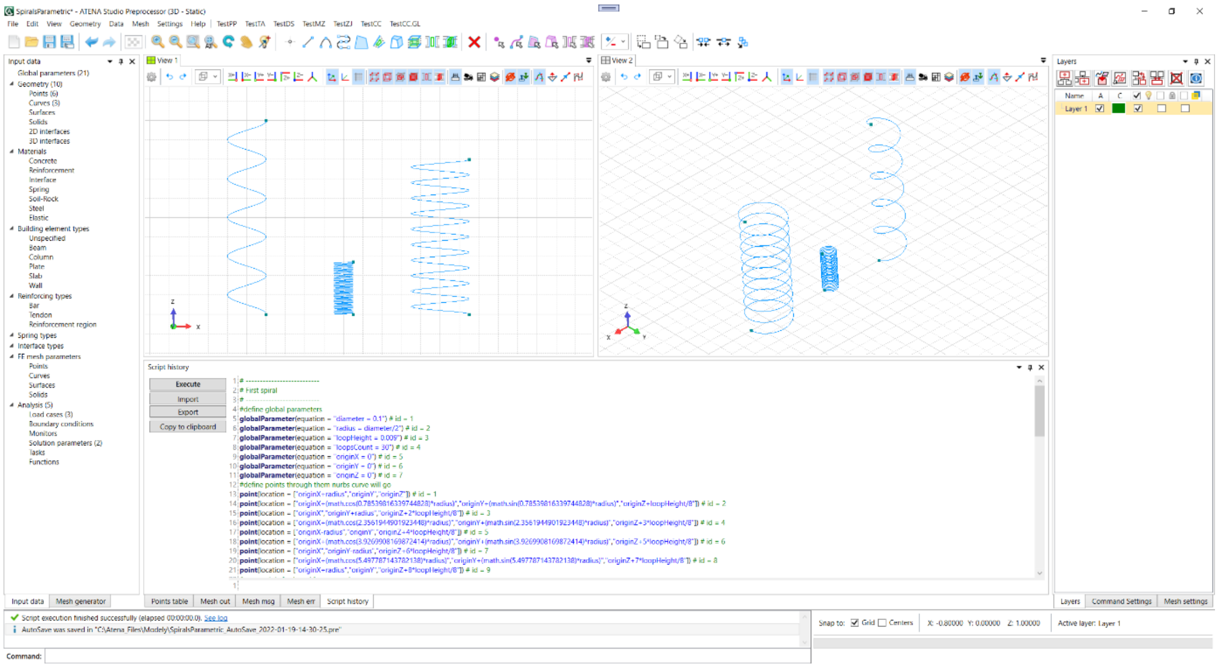
✔ Script execution finished successfully (elapsed 00:00:00.0). See log

AutoSave was saved in "C:\Atena_Files\Modely\SpiralsParametric_AutoSave_2022-01-19-14-55-46.pre"

Fig. 10-2 Script - Parametric circular arcs.

EXAMPLE 2 – Nurbs reinforcement, Fig. 10-3.

```
# --------------------------

# First spiral

# --------------------------

#define global parameters

globalParameter(equation = "diameter = 0.1") # id = 1

globalParameter(equation = "radius = diameter / 2") # id = 1

globalParameter(equation = "loopHeight = 0.009") # id = 2

globalParameter(equation = "loopsCount = 30") # id = 3

globalParameter(equation = "originX = 0") # id = 4

globalParameter(equation = "originY = 0") # id = 5

globalParameter(equation = "originZ = 0") # id = 6

#define points through them nurbs curve will go

point(location = ["originX+radius","originY","originZ"]) # id = 1

point(location = ["originX+(math.cos(0.78539816339744828)*radius)","originY+(math.sin(0.78539816339744828)*radius)","originZ+loopHeight/8"]) # id = 2

point(location = ["originX","originY+radius","originZ+2*loopHeight/8"]) # id = 3

point(location = ["originX+(math.cos(2.3561944901923448)*radius)","originY+(math.sin(2.3561944901923448)*radius)","originZ+3*loopHeight/8"]) # id = 4

point(location = ["originX-radius","originY","originZ+4*loopHeight/8"]) # id = 5

point(location = ["originX+(math.cos(3.9269908169872414)*radius)","originY+(math.sin(3.9269908169872414)*radius)","originZ+5*loopHeight/8"]) # id = 6

point(location = ["originX","originY-radius","originZ+6*loopHeight/8"]) # id = 7
point(location = ["originX+(math.cos(5.497787143782138)*radius)","originY+(math.sin(5.497787143782138)*radius)","originZ+7*loopHeight/8"]) # id = 8

point(location = ["originX+radius","originY","originZ+8*loopHeight/8"]) # id = 9

translate(startPoint = [0,0,0], endPoint = [0,0,"loopHeight"], copies = "loopsCount - 1", collapse = "True", extrude = "False", points = [1, 2, 3, 4, 5, 6, 7, 8, 9])

nurbsCurveThroughPoints(degree = 3, fromPoint = 1, toPoint = 270)


# --------------------------

# Second spiral

# --------------------------

#define global parameters

globalParameter(equation = "diameter2 = 0.3") # id = 1

globalParameter(equation = "radius2 = diameter2 / 2") # id = 1

globalParameter(equation = "loopHeight2 = 0.08") # id = 2

globalParameter(equation = "loopsCount2 = 10") # id = 3

globalParameter(equation = "originX2 = 0.5") # id = 4

globalParameter(equation = "originY2 = 0.0") # id = 5

globalParameter(equation = "originZ2 = 0") # id = 6

#define points through them nurbs curve will go

point(location = ["originX2+radius2","originY2","originZ2"]) # id = 1

point(location = ["originX2+(math.cos(0.78539816339744828)*radius2)","originY2+(math.sin(0.78539816339744828)*radius2)","originZ2+loopHeight2/8"]) # id = 2

point(location = ["originX2","originY2+radius2","originZ2+2*loopHeight2/8"]) # id = 3

point(location = ["originX2+(math.cos(2.3561944901923448)*radius2)","originY2+(math.sin(2.3561944901923448)*radius2)","originZ2+3*loopHeight2/8"]) # id = 4

point(location = ["originX2-radius2","originY2","originZ2+4*loopHeight2/8"]) # id = 5

point(location = ["originX2+(math.cos(3.9269908169872414)*radius2)","originY2+(math.sin(3.9269908169872414)*radius2)","originZ2+5*loopHeight2/8"]) # id = 6

point(location = ["originX2","originY2-radius2","originZ2+6*loopHeight2/8"]) # id = 7

point(location = ["originX2+(math.cos(5.497787143782138)*radius2)","originY2+(math.sin(5.497787143782138)*radius2)","originZ2+7*loopHeight2/8"]) # id = 8

point(location = ["originX2+radius2","originY2","originZ2+8*loopHeight2/8"]) # id = 9

translate(startPoint = [0,0,0], endPoint = [0,0,"loopHeight2"], copies = "loopsCount2 - 1", collapse = "True", extrude = "False", points = [271, 272, 273, 274, 275, 276, 277, 278, 279])

nurbsCurveThroughPoints(degree = 3, fromPoint = 271, toPoint = 360)

# --------------------------

# Third spiral - points

# --------------------------

#define global parameters

globalParameter(equation = "diameter3 = 0.2") # id = 1

globalParameter(equation = "radius3 = diameter3 / 2") # id = 1

globalParameter(equation = "loopHeight3 = 0.2") # id = 2

globalParameter(equation = "loopsCount3 = 5") # id = 3

globalParameter(equation = "originX3 = -0.5") # id = 4

globalParameter(equation = "originY3 = 0.0") # id = 5

globalParameter(equation = "originZ3 = 0") # id = 6

#define points through them nurbs curve will go

point(location = ["originX3+radius3","originY3","originZ3"]) # id = 1

point(location = ["originX3+(math.cos(0.78539816339744828)*radius3)","originY3+(math.sin(0.78539816339744828)*radius3)","originZ3+loopHeight3/8"]) # id = 2
point(location = ["originX3","originY3+radius3","originZ3+2*loopHeight3/8"]) # id = 3

point(location                                                                     =
["originX3+(math.cos(2.356194490192344 8)*radius3)","originY3+(math.sin(2.3561944901923448)*radius3)",
"originZ3+3*loopHeight3/8"]) # id = 4

point(location = ["originX3-radius3","originY3","originZ3+4*loopHeight3/8"]) # id = 5

point(location                                                                     =
["originX3+(math.cos(3.9269908169872414)*radius3)","originY3+(math.sin(3.9269908169872414)*radius3)",
"originZ3+5*loopHeight3/8"]) # id = 6

point(location = ["originX3","originY3-radius3","originZ3+6*loopHeight3/8"]) # id = 7

point(location                                                                     =
["originX3+(math.cos(5.497787143782138)*radius3)","originY3+(math.sin(5.497787143782138)*radius3)","o
riginZ3+7*loopHeight3/8"]) # id = 8

point(location = ["originX3+radius3","originY3","originZ3+8*loopHeight3/8"]) # id = 9

translate(startPoint = [0,0,0], endPoint = [0,0,"loopHeight3"], copies = "loopsCount3 - 1", collapse
= "True", extrude = "False", points = [361, 362, 363, 364, 365, 366, 367, 368, 369])

nurbsCurveThroughPoints(degree = 3, fromPoint = 361, toPoint = 405)


#delete intervening points

delete(deleteLowerEntities = "False", deleteHigherEntities = "False", deleteElementTypes = "False",
points = "all")
```



```
# ---------------------------
# First spiral
# ---------------------------
#define global parameters
globalParameter(equation = "diameter = 0.1") # id = 1
globalParameter(equation = "radius = diameter/2") # id = 2
globalParameter(equation = "loopHeight = 0.009") # id = 3
globalParameter(equation = "loopsCount = 30") # id = 4
globalParameter(equation = "originX = 0") # id = 5
globalParameter(equation = "originY = 0") # id = 6
globalParameter(equation = "originZ = 0") # id = 7
#define points through them nurbs curve will go
point(location = ["originX+radius","originY","originZ"]) # id = 1
point(location = ["originX+(math.cos(0.7853981633974483)*radius)","originY+(math.sin(0.7853981633974483)*radius)","originZ+loopHeight/8"]) # id = 2
point(location = ["originX","originY+radius","originZ+2*loopHeight/8"]) # id = 3
point(location = ["originX+(math.cos(2.356194490192345)*radius)","originY+(math.sin(2.356194490192345)*radius)","originZ+3*loopHeight/8"]) # id = 4
point(location = ["originX-radius","originY","originZ+4*loopHeight/8"]) # id = 5
point(location = ["originX+(math.cos(3.9269908169872414)*radius)","originY+(math.sin(3.9269908169872414)*radius)","originZ+5*loopHeight/8"]) # id = 6
point(location = ["originX","originY-radius","originZ+6*loopHeight/8"]) # id = 7
point(location = ["originX+(math.cos(5.497787143782138)*radius)","originY+(math.sin(5.497787143782138)*radius)","originZ+7*loopHeight/8"]) # id = 8
point(location = ["originX+radius","originY","originZ+8*loopHeight/8"]) # id = 9
```

```
Script execution finished successfully (elapsed 00:00:00.0). See log
AutoSave was saved in "C:\Atena_Files\Modely\SpiralsParametric_AutoSave_2022-01-19-14-30-25.pre"
```

Fig. 10-3 Script – Parametric nurbs curves.

10.2 Example of Python user-defined function

Example of Python scripting is shown on function that defines cuboid beam with bottom reinforcement. User inputs for the function are beam origin (placing the beam within the global coordinate system), beam sizes, concrete compressive strength, number of reinforcing bars and their diameter.

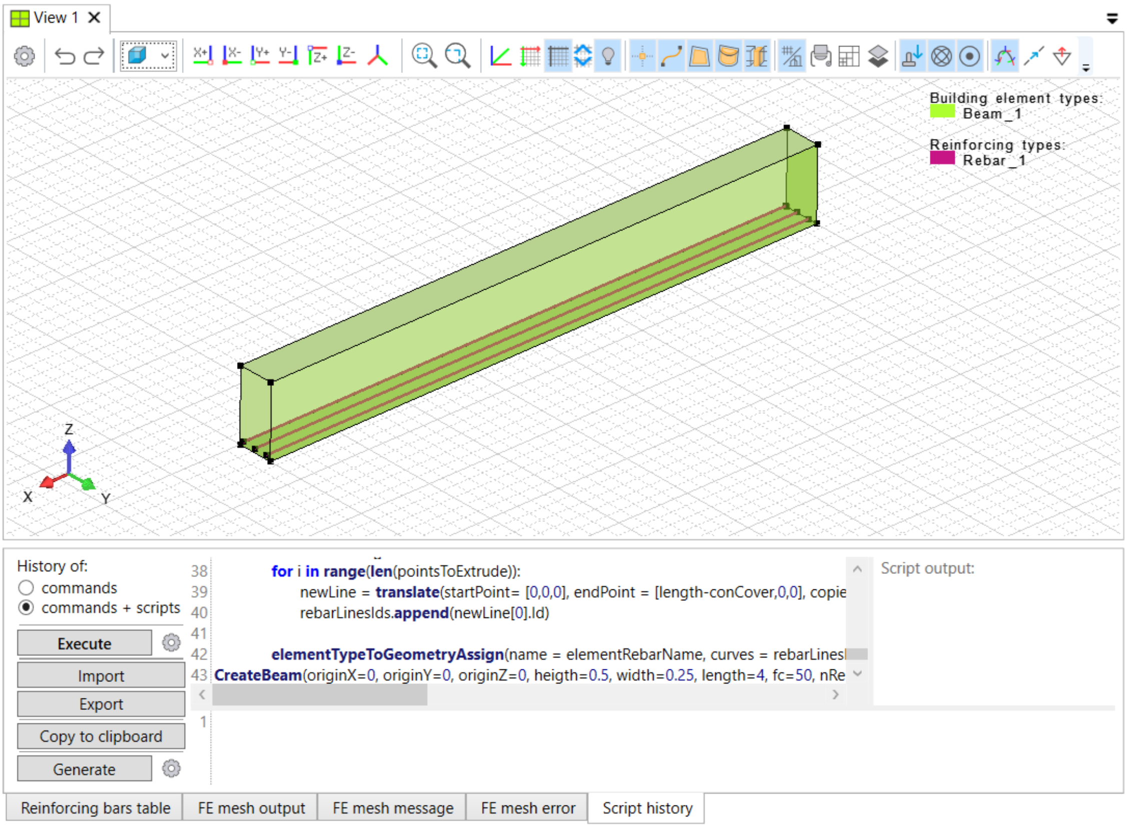
```
----------------------------------------------------------------------------------------------------

def CreateBeam(originX, originY, originZ, heigth, width, length, fc, nRebar, dRebar):

        P1 = point(location = [originX, originY, originZ]) # origin of the beam

        L1 = translate(startPoint= [0,0,0], endPoint = [0,width,0], copies = 1, collapse = "True",
extrude  =  "True",  keepParametric  =  "True",  createSurfaces  =  "True",  createSolids  =  "True",
respectLayers = "True", points = P1.Id)[0] #extruding width

        S1 = translate(startPoint = [0,0,0], endPoint = [0,0,heigth], copies = 1, collapse = "True",
extrude  =  "True",  keepParametric  =  "True",  createSurfaces  =  "True",  createSolids  =  "False",
respectLayers = "True", curves = L1.Id)[0] #extruding height
```

```
        V1 = translate(startPoint = [0,0,0], endPoint = [length,0,0], copies = 1, collapse = "True",
extrude = "True", keepParametric = "True", createSurfaces = "True", createSolids = "True",
respectLayers = "True", surfaces = S1.Id)[0] #extruding length


        concreteName ="Concrete_" + str(fc) + "MPa_" + str(V1.Id)

        material(name = concreteName, prototype = "CC3DNonLinCementitious2", generator = "General")

        materialGeneratorEdit(material = concreteName, parameter = "StrengthValue", value = fc, unit
= "MPa")

        materialGeneratorRun(material = concreteName)

        elementTypeName ="Beam_" + str(V1.Id)

        el1 = elementType(name = elementTypeName, type = "BeamType", material = concreteName)

        elementTypeToGeometryAssign(name =elementTypeName, solids = V1.Id)


        if nRebar > 0:

                rebarName="Steel_" +  str(V1.Id)

                material(name = rebarName, prototype = "CCReinforcement", generator = "General")

                materialGeneratorRun(material = rebarName)

                elementRebarName="Rebar_" +  str(V1.Id)

                elementType(name = elementRebarName, type = "BarType", material = rebarName)

                elementTypeEdit(name = elementRebarName , parameter = "ReBondType", value =
"PerfectBond")

                elementTypeEdit(name = elementRebarName, parameter = "ProfileDiameter", value =
dRebar)


                conCover = 0.03

                pointsToExtrude = []

                if nRebar > 1:

                        rebarSpacing = (width - 2*conCover)/(nRebar-1)

                        newPoint = point(location = [originX +conCover , originY + conCover,
originZ + conCover])

                        pointsToExtrude.append(newPoint)

                        for i in range(nRebar-1):

                                newPoint = point(location = [originX +conCover , originY + conCover
+ (i+1)*rebarSpacing, originZ + conCover])

                                pointsToExtrude.append(newPoint)

                else:

                        newPoint = point(location = [originX +conCover , originY + width/2, originZ
+ conCover])

                        pointsToExtrude.append(newPoint)


                rebarLinesIds=[]

                for i in range(len(pointsToExtrude)):

                        newLine = translate(startPoint= [0,0,0], endPoint = [length-conCover,0,0],
copies = 1, collapse = "False", extrude = "True", keepParametric = "True", createSurfaces = "False",
createSolids = "False", respectLayers = "True", points = pointsToExtrude[i].Id)

                        rebarLinesIds.append(newLine[0].Id)


                elementTypeToGeometryAssign(name = elementRebarName, curves = rebarLinesIds)
----------------------------------------------------------------------------------------------------
```

The function can be then applied repeatedly with various input parameters, for example, also see Fig. 10-4:

```
----------------------------------------------------------------------------------------------------
```

```
CreateBeam(originX=0, originY=0, originZ=0, heigth=0.5, width=0.25, length=4,
fc=50, nRebar=3, dRebar=0.016)
-----------------------------------------------------------------------------------------
```



```
        for i in range(len(pointsToExtrude)):
            newLine = translate(startPoint= [0,0,0], endPoint = [length-conCover,0,0], copie
            rebarLinesIds.append(newLine[0].Id)

        elementTypeToGeometryAssign(name = elementRebarName, curves = rebarLinesI
CreateBeam(originX=0, originY=0, originZ=0, heigth=0.5, width=0.25, length=4, fc=50, nRe
```

Fig. 10-4 Python function – reinforced beam.

REFERENCES

[1] Cervenka, V., Jendele, L, Cervenka, J., (2022), *ATENA Program Documentation, Part 1, Theory*, Cervenka Consulting, 2, 2022

[2] Cervenka, J., Altman, T., Janda, Z., Rymes J., Herzfeldt M., Palek, P., Pukl, R., ATENA 2024 Program Documentation, Tutorial SWO for ATENA-PRE, ATENA with CeSTaR 2 Module, Cervenka Consulting s.r.o. 2024.

[3] Pryl, D. and Cervenka, J., (2022), *ATENA Program Documentation Part 11, ATENA Troubleshooting*, Cervenka Consulting, 2022

[4] Cervenka, J., and Jendele, L., (2022), *ATENA Program Documentation, Part 6, ATENA Input File Format*, Cervenka Consulting, 2022

[5] Rypl, D. (2016), *Triangulation of 3D domains – User guide*, CTU in Prague, 2016

ANNEX A – ATENA COMMANDS

Camera

- Function:
    - def fitAll():
- Examples:
    - **fitAll**();

Global parameters

Create global parameter

- Function:
    - def globalParameter(equation):
- Examples:
    - **globalParameter**(equation = "a = 1")
    - **globalParameter**(equation = "b = a+2")

Geometry creating

Point

- Function:
    - def point(location, name = None): # id = ..
- Examples:
    - **point**(location=[0.1, 0.2, 0.3]);
    - **point**(location=[0.1, 0.2, 0.3], name="my point”);
    - **point**(location = ["a","b",0]) # id = 9
    - in 2D: **point**(location=[0.1, 0.2]);

Line segment

- Function:
    - def line(startPoint, endPoint, name = None): # id = ..
- Examples:
    - **line**(startPoint=[0.1, 0.2, 0.3], endPoint=[0.3, 0.2, 0.1])
    - **line**(startPoint=1, endPoint=[0.1, 0.2, 0.3])
        - 1 is an existing point ID
    - **line**(startPont=1, endPoint=2)
        - 1 and 2 are point IDs
    - **line**(startPoint=1, endPoint=2, name="my line")
    - **line**(startPoint = ["a",0,0], endPoint = ["b",0,0])

Circular arc

- Function:
  - def circularArc(method, startPoint, endPoint = None, interiorPoint = None, centerPoint = None, angleInDegrees = None, anglePoint = None, radius = None, radiusPoint = None, directionAtStart = None, createCircle = None, createCircularSurface = None, name = None);
- Examples:
  - **circularArc**(method = "CenterStartAngle", centerPoint = [1.2,-0.2,0], startPoint = [1.3,-1.2,0], angleInDegrees = 83.616, anglePoint = [-0.2,-0.5,0])
  - **circularArc**(method = "StartEndRadius", startPoint = [0,1,0], endPoint = [-0.4,-0.2,0], radius = 1.2649, radiusPoint = [-1.2,0.6,0])
  - **circularArc**(method = "StartInteriorEnd", startPoint = [-1,1.2,0], interiorPoint = [-1.5,0.1,0], endPoint = [-2.1,1.2,0])
  - **circularArc**(method = "StartEndInterior", startPoint = [-1,2.6,0], endPoint = [-1.3,1.9,0], interiorPoint = [-1.8,2.2,0], createCircle = "False", createCircularSurface = "True")
  - **circularArc**(method = "StartEndDirectionAtStart", startPoint = [2.3,-0.3,0], endPoint = [2.5,-1.5,0], directionAtStart = [2.1,-1.7,0])
  - **circularArc**(method = "StartDirectionAtStartEnd", startPoint = [1.4,1.4,0], directionAtStart = [1.4,0.9,0], endPoint = [1,1.1,0])
  - circle: **circularArc**(method = "CenterStartAngle", centerPoint = [1,-2.2,0], startPoint = [1.1,-3.2,0], angleInDegrees = 35.455, anglePoint = [0.6,-2.9,0], createCircle = "True", createCircularSurface = "True" ,name = "my circle")

NURBS curve through points

- Function:
  - nurbsCurveThroughPoints(degree, points = None, fromPoint = None, toPoint = None, name = None);
  - points – can be either [x,y,z] or [pointId]
- Examples:
  - **nurbsCurveThroughPoints**(degree = 3, points = [[1.3, 0.6, 0],[0.6, -0.4, 0],[0.7, 0.5, 0],[0, 1.3, 0],[-1.1, -0.4, 0],[4],[-0.3, -1.1, 0],[2]])

Surface from boundary curves

- Function:
  - def surfaceByEdges(edges, name = None): # id = ..
  - edges – IDs of existing curves
- Examples:
  - **surfaceByEdges**(edges = [12, 13, 14, 15, 16], name = "my surface")

Ruled surface (surface between two curves)

- Function:
  - def ruledSurface(curve1, curve2, name = None): #id = ..
- Examples:
  - **ruledSurface**(curve1 = 17, curve2 = 18)

Solid from boundary surfaces

- Function:
  - def solidBrep(surfaces, name = None): # id = ..

- Examples:
  - solidBrep(surfaces = [2, 1, 3, 4, 5, 6]) # id = 1

Ruled solid (solid between two surfaces)

- Function:
  - def ruledSolid(surface1, surface2, name = None): # id = ..
- Examples:
  - ruledSolid(surface1 = 1, surface2 = 2)

Interface 2D

- Function:
  - def interface2d(curves, name = None):
- Examples:
  - interface2d(curves = [1, 2])

Interface 3D

- Function:
  - def interface3d(surfaces, name = None):
- Examples:
  - interface3d(surfaces = [1, 2])

Geometry tranformations

Translation

- Function:
  - def translate(startPoint, endPoint, copies, collapse, extrude, keepParametric = None, createSurfaces = None, createSolids = None, respectLayers = None, points = None, curves = None, surfaces = None, solids = None, interfaces2d = None, interfaces3d = None):
- Examples:
  - translate(startPoint = [-0.5,0.8,0], endPoint = [-1.7,0.7,0], copies = 0, collapse = "True", extrude = "False", respectLayers = "True", curves = [1, 2])
  - translate(startPoint = [-1.1,1,0], endPoint = [-2.3,1.7,0], copies = 2, collapse = "True", extrude = "False", respectLayers = "True", solids = [1])
  - extrusion: translate(startPoint = [1.7,0.4,0], endPoint = [2.8,0.1,0], copies = 1, collapse = "True", extrude = "True", keepParametric = "True", createSurfaces = "True", createSolids = "False", respectLayers = "True", curves = [10])

Scaling

- Function:
  - def scale(center, factor, copies, collapse, respectLayers = None, points = None, curves = None, surfaces = None, solids = None, interfaces2d = None, interfaces3d = None):
- Examples:
  - scale(center = [1.3,-0.6,0], factor = [0.5,0.5,0.5], copies = 0, collapse = "True", respectLayers = "True", surfaces = [19])
  - scale(center = [1,1,0], factor = [2,2,2], copies = 2, collapse = "True", respectLayers = "True", surfaces = [19])

Rotation

- Function:
  - def rotate(center, startPoint, endPoint, copies, collapse, revolve, angleRange = None, keepParametric = None, createSurfaces = None, createSolids = None, respectLayers = None, points = None, curves = None, surfaces = None, solids = None, interfaces2d = None, interfaces3d = None):
- Examples:
  - **rotate**(center = [1.4,1.2,0], startPoint = [1.8,0.6,0], endPoint = [0.3,1.5,0], copies = 0, collapse = "True", revolve = "False", angleRange = "LessThan180Deg", keepParametric = "True", createSurfaces = "True", createSolids = "True", respectLayers = "True", surfaces = [19])
  - revolve: **rotate**(center = [0.2,3.3,0], startPoint = 22, endPoint = [0.9,2.1,0], copies = 1, collapse = "True", revolve = "True", angleRange = "LessThan180Deg", keepParametric = "True", createSurfaces = "True", createSolids = "True", respectLayers = "True", curves = [39])

Geometry modifications

Trim surface

- Function:
  - def surfaceTrim(surface, trimCurves):
- Examples:
  - **surfaceTrim**(surface = 1, trimCurves= [5,6,7,8])

Hole in surface

- Function:
  - def surfaceHole(surface, holeCurves = None, holeSurfaces = None):
- Examples:
  - **surfaceHole**(surface = 1, holeSurfaces = [5])
  - **surfaceHole**(surface = 4, holeCurves = [24, 25, 26])

Hole in solid

- Function:
  - def solidHole(solid, holeSurfaces = None, holeSolids = None):
- Examples:
  - **solidHole**(solid = 1, holeSurfaces = [10, 7, 8, 9, 11, 12])
  - **solidHole**(solid = 1, holeSolids = [2])

Copy

- Function:
  - def copy(respectLayers = None, points = None, curves = None, surfaces = None, solids = None, interfaces2d = None, interfaces3d = None):
- Examples:
  - **copy**(respectLayers = "True", surfaces = [14, 15])

Separate

- Function:
  - def separate(respectLayers = None, points = None, curves = None, surfaces = None, solids = None, interfaces2d = None, interfaces3d = None):
- Examples:
  - **separate**(respectLayers = "True", surfaces = [20])

Collapse

- Function:
  - def collapse(points = None, curves = None, surfaces = None, solids = None, interfaces2d = None, interfaces3d = None):
- Examples:
  - **collapse**(surfaces = [19, 22])

Delete

- Function:
  - def delete(deleteLowerEntities = None, deleteHigherEntities = None, globalParameters = None, points = None, curves = None, surfaces = None, solids = None, interfaces2d = None, interfaces3d = None, profiles = None, elementTypes = None, materials = None, functions = None, loadCases = None, boundaryConditions = None, monitors = None, selections = None, solutionParameters = None, tasks = None, feMeshSizeParameters = None, feMeshTypeParameters = None, feMeshAdvancedParameters = None, feMeshExtrusionParameters = None):
- Examples:
  - **delete**(deleteLowerEntities = "False", deleteHigherEntities = "False", points = [1, 2, 3, 4, 5, 6], curves = [1, 3, 5, 6, 7, 8, 9], surfaces = [2, 3])

Curve subdivision

- Function:
  - def curvesSubdivide(curves, divisions = None, nearPoint = None, parameter = None, relativeLength = None, length = None, collapseNewPoints = None):
- Examples:
  - **curvesSubdivide**(curves = [42], divisions = 3, collapseNewPoints = "True")
  - **curvesSubdivide**(curves = [37], nearPoint = [1,1,0], collapseNewPoints = "True")
  - **curvesSubdivide**(curves = [43, 44], parameter = 0.3, collapseNewPoints = "True")
  - **curvesSubdivide**(curves = [60, 67], relativeLength = 0.3, collapseNewPoints = "True")
  - **curvesSubdivide**(curves = [41], length = 1, collapseNewPoints = "False")
  - **curvesSubdivide**(curves = [46], stepLength = 0.1, collapseNewPoints = "True")

Surface subdivision

- Function:
  - def surfaceSubdivide(surface, firstCurve, secondCurve, divisions):
- Examples:
  - **surfaceSubdivide**(surface = 1, firstCurve = 2, secondCurve = 4, divisions = 2)

Curves compose

- Function:
  - def curvesCompose(curves, name = None):

- Examples:
    - **curvesCompose**(curves = [1, 2, 3, 4, 5, 6])

- ## Curves decompose

- Function:
    - def curvesDecompose(curves):
- Examples:
    - **curvesDecompose**(curves = [7])

- ## NURBS curve edit control points

- Function:
    - def nurbsCurveEdit(curve, controlPoints, startPoint = None, endPoint = None, newWeight = None):
- Examples:
    - **nurbsCurveEdit**(curve = 8, controlPoints = [2], startPoint = [1.7,0.8,0], endPoint = [2,0.9,0])
    - **nurbsCurveEdit**(curve = 8, controlPoints = [5], newWeight = 1.88884444444443)

- ## NURBS surface edit control points

- Function:
    - def nurbsSurfaceEdit(surface, controlPoints, startPoint = None, endPoint = None, newWeight = None):
- Examples:
    - **nurbsSurfaceEdit**(surface = 2, controlPoints = [11], startPoint = [4.7,-7.4,0], endPoint = [6.3,-8.7,0])
    - **nurbsSurfaceEdit**(surface = 2, controlPoints = [10], newWeight = 5.20719703703703)

- ## Force geometry to mesh

- Function:
    - def forceGeometryToMesh(curveMesh = None, surfaceMesh = None, solidMesh = None, fixedPoints = None, fixedCurves = None):
- Examples:
    - **forceGeometryToMesh**(curveMesh = 2, fixedPoints = [6])

- ## Release points from mesh

- Function:
    - def releasePointsFromMesh(points):
- Examples:
    - **releasePointsFromMesh**(points = [6])

- ## Release curves from mesh

- Function:
    - def releaseCurvesFromMesh(curves):
- Examples:
    - **releaseCurvesFromMesh**(curves = [10])

Set name to geometry

- Function:
  - def geometryNameSet(type, id, name = None):
- Examples:
  - **geometryNameSet**(type = "Point", id = 1, name = "my point")
  - **geometryNameSet**(type = "Curve", id = 2, name = "help curve")

Changing beam and shell vectors

Changing vector V1 of beams 1d

- Function:
  - def curvesBeamVectorChange(curves, mode = None, reverseV1Direction = None, manualV3 = None):
- Examples:
  - **curvesBeamVectorChange**(curves = [1], reverseV1Direction = "Opposite")
  - **curvesBeamVectorChange**(curves = [1], mode = "UserDefined", manualV3 = [0, 0, 1])

Changing vector V1 of shells 2d

- Function:
  - def surfacesShellVectorChange(surfaces, mode = None, reverseNormalDirection = None, manualV1 = None):
- Examples:
  - **surfacesShellVectorChange**(surfaces = [1], reverseNormalDirection = "Opposite")
  - **surfacesShellVectorChange**(surfaces = [1], mode = "UserDefined", manualV1 = [0, 1, 0])

Changing vector V1 and V3 of beams 3d

- Function:
  - def solidsBeamVectorsChange(solids, modeV1 = None, vectorV1 = None, modeV3 = None, vectorV3 = None, frontSurfaces = None, topSurfaces = None):
- Examples:
  - **solidsBeamVectorsChange**(solids = [1], modeV1 = "UserDefined", vectorV1 = [1, 0, 0], modeV3 = "UserDefined", vectorV3 = [0, 1, 0])
  - **solidsBeamVectorsChange**(solids = [1], frontSurfaces = [5], topSurfaces = [2])

Changing vector V1 and V3 of shells 3d

- Function:
  - def solidsShellVectorsChange(solids, modeV1 = None, vectorV1 = None, modeV3 = None, vectorV3 = None, topSurfaces = None):
- Examples:
  - **solidsShellVectorsChange**(solids = [1], modeV1 = "UserDefined", vectorV1 = [1, 0, 0], topSurfaces = [5])
  - **solidsShellVectorsChange**(solids = [1], modeV1 = "UserDefined", vectorV1 = [1, 0, 0], modeV3 = "UserDefined", vectorV3 = [0, 0, 1])

• Materials

- Available functions:

```
def material(name, prototype, generator):
def materialSetName(name, newName):
def materialEdit(name, parameter, value, unit = None):
def materialCopy(name, newName = None):
def materialGridRowEdit(name, parameter, index, value, unit = None):
def materialGridRowAdd(name, parameter):
def materialGridRowDelete(name, parameter, index):
def materialFunctionChange(name, parameter, previousFunction, newFunction):
def materialGeneratorSet(name, prototype, generator):
def materialGeneratorEdit(material, parameter, value, unit = None):
def materialGeneratorGridRowEdit(name, parameter, index, value, unit = None):
def materialGeneratorGridRowAdd(name, parameter):
def materialGeneratorGridRowDelete(name, parameter, index):
def materialGeneratorRun(material):
def materialSubGeneratorSet(material, parameter, subGenerator):
def materialSubGeneratorEdit(material, subGenerator, parameter, value,unit = None):
def materialSubGeneratorRun(material, subGenerator):
def materialSubGeneratorGridRowEdit(material, subGenerator, parameter, index, value, unit = None):
def materialSubGeneratorGridRowAdd(material, subGenerator, parameter):
def materialSubGeneratorGridRowDelete(material, subGenerator, parameter, index):
```

- Examples:

```
material(name = "Concrete (3)", prototype = "CC3DNonLinCementitious2", generator = "General")
materialGeneratorSet(name = "Concrete (3)", prototype = "Cementitious2_Variable", generator = "3D_print_generate")
materialGeneratorSet(name = "Concrete (3)", prototype = "Cementitious2_Variable", generator = "3D_print_MasterFlow_3D")
materialGeneratorSet(name = "Concrete (3)", prototype = "Cementitious2_Variable", generator = "3D_print_generate")
materialGeneratorEdit(material = "Concrete (3)", parameter = "Fc_28", value = -5, unit = "MPa")
materialGeneratorEdit(material = "Concrete (3)", parameter = "Fc_28", value = -5000, unit = "kPa")
materialGeneratorRun(material = "Concrete (3)")
materialEdit(name = "Concrete (3)", parameter = "Poisson's ratio", value = 0.3)
materialEdit(name = "Concrete (3)", parameter = "Activate crack spacing", value = "True")
materialEdit(name = "Concrete (3)", parameter = "Crack spacing", value = 0.03, unit = "m")
materialEdit(name = "Concrete (3)", parameter = "Fixed crack", value = 2)
materialEdit(name = "Concrete (3)", parameter = "Fixed crack", value = 1)
materialEdit(name = "Concrete (3)", parameter = "Activate function for EPS CP", value = "False")
materialEdit(name = "Concrete (3)", parameter = "Plastic strain EPS CP", value = -0.003)
materialEdit(name = "Concrete (3)", parameter = "Activate substepping", value = "True")
```

• Element types (building, reinforcing, spring and interface)

- Available functions:

```
def elementType(name, type, material):
def elementTypeEdit(name, parameter, value):
def elementTypeMainTypeChange(name, mainType):
def elementTypeToGeometryAssign(name, points = None, curves = None, surfaces = None, solids = None,
interfaces2d = None, interfaces3d = None):
def elementTypeMaterialSet(name, material):
def elementTypeFunctionSet(name, parameter, function):
def elementTypeSubGeneratorSet(elementType, parameter, subGenerator):
def elementTypeSubGeneratorEdit(elementType, subGenerator, parameter, value, unit = None):
def elementTypeSubGeneratorRun(elementType, subGenerator):
def elementGeometryChange(name, geometryType):
def elementGeometryEdit(name, parameter, value):
def elementGeometryMaterialSet(name, parameter, material):
def elementGeometryVectorSet(name, parameter, coord, value):
def elementGeometryIsFiberRasterSet(name, value, solidMaterial, rebarMaterial = None):
def elementGeometryRasterSizeSet(name, isColumn, value, material):
def elementGeometryRebarCoordsAdd(name, material, area, s, t, enabled):
def elementGeometryRebarCoordsDelete(name, id):
def elementGeometryRebarCoordsEdit(name, id, parameter, value):
def elementGeometrySolidCoordsAdd(name, material, area, s, t, enabled):
def elementGeometrySolidCoordsDelete(name, id):
def elementGeometrySolidCoordsEdit(name, id, parameter, value):
def elementGeometryMaterialToCellAssign(name, index, material):
def elementGeometryMaterialToAllCellsAssign(name, material):
def elementGeometryMaterialToAssignedCellsAssign(name, material):
def elementGeometryMaterialToEmptyCellsAssign(name, material):
def elementGeometryRasterRowColumnSet(name, isRow, index, value):
def elementGeometryRasterRowColumnCheck(name, isRow, index, isChecked):
def elementGeometryRasterRowSet(name, index, value):
def elementGeometryRasterRowCheck(name, index, isChecked):
def elementGeometryRowsCountSet(name, value, material):
def elementGeometryReinfLayerAdd(name, material, thickness):
def elementGeometryReinfLayerEdit(name, id, parameter, value):
def elementGeometryReinfLayerDelete(name, id):
def elementGeometryReinfLayerVectorSet(name, id, parameter, coord, value):
def elementTypeCopy(name, newName = None):
```

- Examples:

```
elementType(name = "Unspecified types (2)", type = "ProxyType", material = "Concrete1")
elementGeometryChange(name = "Unspecified types (2)", geometryType = "GeometryBeam3D")
elementTypeEdit(name = "Unspecified types (2)", parameter = "FEMeshQuadratic", value = "True")
elementTypeMainTypeChange(name = "Unspecified types (2)", mainType = "ChimneyType")
elementTypeEdit(name = "Unspecified types (2)", parameter = "Name", value = "Chimney types")
elementTypeEdit(name = "Chimney types", parameter = "Name", value = "My be type")
elementGeometryVectorSet(name = "My be type", parameter = "LocalAxisX", coord = "x", value =
0.1)
elementGeometryMaterialToCellAssign(name = "My be type", index = 32, material = "null")
elementGeometryMaterialToCellAssign(name = "My be type", index = 19, material = "Concrete 2")
elementGeometryMaterialToCellAssign(name = "My be type", index = 7, material = "Concrete 2")
elementTypeSubGeneratorEdit(elementType = "Unspecified types (2)", subGenerator =
"Fiber_raster_generator", parameter = "Material to assign", value = "Concrete 2")
```

```
elementTypeSubGeneratorEdit(elementType = "Unspecified types (2)", subGenerator = "Fiber_raster_generator", parameter = "IPE_profile", value = 160)
elementTypeSubGeneratorRun(elementType = "My be type", subGenerator = "Fiber_raster_generator")
elementGeometryRasterRowColumnCheck(name = "My be type", isRow = "True", index = 0, isChecked = "False")
elementGeometryIsFiberRasterSet(name = "My be type", value = "False", solidMaterial = "Concrete 2", rebarMaterial = "Reinforcement1")
elementGeometrySolidCoordsAdd(name = "My be type", material = "Concrete (3)", area = 0.01, s = 0, t = 0, enabled = "True")
elementGeometrySolidCoordsDelete(name = "My be type", id = 2)
elementGeometrySolidCoordsEdit(name = "My be type", id = 3, parameter = "CoordT", value = 0.001)
elementGeometrySolidCoordsDelete(name = "My be type", id = 3)
```

Functions

- Available functions:

```
def function(name):
def functionEdit(name, newName = None, column = None, newColumnName = None, newUnit = None, newSimple = None, newStandardRanges = None, newLimitRanges = None):
def functionCopy(name, newName = None):
def functionImport(targetFunction, importedFunction):
def functionEquation(name, equation, horizMin, horizMax, samplesCount):
def functionDefinitionChange(name, isCoordsDefinition):
def functionRecord(name, row, horizValue, vertValues, horizUnit = None, vertUnits = None):
def functionRecordEdit(name, row, col, value, unit = None):
def functionRecordsDelete(name, rows):
def functionRecordsMove(name, rows, moveUp):
```

- Examples:

```
function(name = "Function")
functionDefinitionChange(name = "Function", isCoordsDefinition = "False")
functionEquation(name = "Function", equation = "sin(x)", horizMin = 0, horizMax = 1, samplesCount = 10)
functionDefinitionChange(name = "Function", isCoordsDefinition = "True")
functionRecordsDelete(name = "Function", rows = "8, 9, 10")
functionRecord(name = "Function", row = 8, horizValue = 0.7, vertValues = 0.6)
functionRecord(name = "Function", row = 9, horizValue = 0.8, vertValues = 0.6)
functionEdit(name = "Function", newName = "My function")
functionEdit(name = "My function", column = 1, newColumnName = "horiz")
functionEdit(name = "My function", column = 2, newColumnName = "vert")
functionEdit(name = "My function", column = 1, newUnit = "m")
functionEdit(name = "My function", column = 2, newUnit = "m^2")
functionEdit(name = "My function", column = 2, newSimple = "True")
functionEdit(name = "My function", column = 1, newStandardRanges = "(-100 ; 100)")
functionEdit(name = "My function", column = 2, newStandardRanges = "(-10 ; 10)")
functionEdit(name = "My function", column = 2, newSimple = "False")
```

Load cases

- Available functions:

```
def loadCase(name, loadType, loadCategory):
def loadCaseEdit(name, parameter, value):
def loadCaseCopy(name, newName = None):
```

- Examples:

```
loadCase(name = "Body force (2)", loadType = "Forces", loadCategory = "LiveLoad") # id = 4
loadCaseEdit(name = "Body force (2)", parameter = "AutomaticName", value = "False")
loadCaseEdit(name = "Body force (2)", parameter = "Name", value = "My load case")
loadCaseEdit(name = "My load case", parameter = "LoadType", value = "Supports")
loadCaseEdit(name = "My load case", parameter = "LoadCategory", value = "Undefined")
loadCaseEdit(name = "My load case", parameter = "LoadCategory", value = "Earthquake")
loadCaseEdit(name = "My load case", parameter = "Multiplier", value = 2)
loadCaseCopy(name = "My load case", newName = "Supports (2)")
```

Boundary conditions

- Available functions:

```
def condition(name, subjectType, conditionType, loadCase):
def conditionTypeChange(name, newType, newName):
def conditionEdit(name, parameter, value):
def conditionLoadCaseSet(name, loadCase):
def conditionMaterialSet(name, number, material):
def conditionToGeometryAssign(name, points = None, curves = None, surfaces = None, solids = None, reTypes = None):
def conditionVectorSet(name, parameter, coord, value):
def conditionFunctionSet(name, parameter, function):
def conditionSubGeneratorSet(condition, parameter, subGenerator):
def conditionSubGeneratorEdit(condition, subGenerator, parameter, value, unit = None):
def conditionSubGeneratorRun(condition, subGenerator):
def conditionCopy(name, newName = None):
```

- Examples:

```
condition(name = "Weight", subjectType = "Curve", conditionType = "Weight", loadCase = "Body force")
conditionLoadCaseSet(name = "Weight", loadCase = "Supports")
conditionTypeChange(name = "Weight", newType = "Constraint", newName = "Support")
conditionTypeChange(name = "Support", newType = "RotationConstraint", newName = "Rotation support")
conditionEdit(name = "Rotation support", parameter = "ActivateRotationConstraintX", value = "True")
conditionEdit(name = "Rotation support", parameter = "AutomaticName", value = "False")
conditionEdit(name = "Rotation support", parameter = "Name", value = "Rotation support 1")
conditionToGeometryAssign(name = "Rotation support 1", curves = [2, 3])
conditionCopy(name = "Rotation support 1", newName = "Rotation support (2)")
```

Monitors

- Available functions:

```
def monitor(name, subjectType):
def monitorEdit(name, parameter, value):
def monitorToGeometryAssign(name, points = None, curves = None, surfaces = None, solids = None):
def monitorCopy(name, newName = None):
```

- Examples:

```
monitor(name = "Monitor", subjectType = "Surface") # id = 1
monitorEdit(name = "Monitor", parameter = "Component1", value = "True")
monitorEdit(name = "Monitor", parameter = "OutputDataType", value = "Reactions")
monitorEdit(name = "Monitor", parameter = "MathOperation", value = "Minimum")
monitorToGeometryAssign(name = "Monitor", surfaces = [1])
monitorCopy(name = "Monitor", newName = "Monitor (2)")
```

Selections

- Available functions:

```
def selection(selectionType, name):
def selectionEdit(name, parameter, value):
def selectionCopy(name, newName = None):
def selectionToGeometryAssign(name, points = None, curves = None, surfaces = None, solids = None):
```

- Examples:

```
selection(selectionType = "NearestNode", name = "Nearest node") # id = 1
selectionEdit(name = "Nearest node", parameter = "AutomaticName", value = "False")
selectionEdit(name = "Nearest node", parameter = "Name", value = "Nearest node 1")
selectionEdit(name = "Nearest node 1", parameter = "SelectionType", value = "InsideQuad")
selectionEdit(name = "Nearest node 1", parameter = "Name", value = "Inside quad")
selectionToGeometryAssign(name = "Inside quad", surfaces = [1])
selectionsCopy(name = "Inside quad", newName = "Inside quad (2)")
```

Solution parameters

- Available functions:

```
def solutionParameters(name):
def solutionParametersEdit(name, parameter, value):
def solutionParametersCopy(name, newName = None):
```

- Examples:

```
solutionParameters(name = "Solution parameters") # id = 3
solutionParametersEdit(name = "Solution parameters", parameter = "SolutionMethod", value = "ArcLength")
solutionParametersEdit(name = "Solution parameters", parameter = "OptimizationAlgorithm", value = "GibbsPoole")
solutionParametersEdit(name = "Solution parameters", parameter = "SolutionMethodSubtype", value = "FullNR")
solutionParametersEdit(name = "Solution parameters", parameter = "LinearSolver", value = "ICCG")
solutionParametersEdit(name = "Solution parameters", parameter =
"ConvergenceMainCriteria_1", value = 0.02)
solutionParametersCopy(name = "Solution parameters", newName = "Solution parameters (2)")
```

FE mesh parameters

- Available functions:

```
def feMeshParams(name, type, gmType):
def feMeshParamsEdit(name, parameter, value):
def feMeshParamsToGeometryAssign(name, points = None, curves = None, surfaces = None, solids = None):
def feMeshParamsCopy(type, name, newName = None):
```

- Examples:

```
feMeshParams(name = "FEMeshSizeCurve", type = "Size", gmType = "Curve")
feMeshParamsEdit(name = "FEMeshSizeCurve", parameter = "FEMeshDefinition", value =
"Density")
feMeshParamsEdit(name = "FEMeshSizeCurve", parameter = "DensityDefinition", value =
"UserDefined")
feMeshParamsEdit(name = "FEMeshSizeCurve", parameter = "Density", value = 0.4)
feMeshParamsToGeometryAssign(name = "FEMeshSizeCurve", curves = [2, 3])
feMeshParamsCopy(type = "Size", name = "FEMeshSizeCurve", newName = "FEMeshSizeCurve
(2)")
```

Tasks

- Available functions:

```
def task(name):
def taskEdit(name, parameter, value):
def taskRun(name):
def taskCopy(name, newName = None):
def interval(task, interval):
def intervalCopy(task, interval, newInterval):
def intervalDelete(task, interval):
def intervalUpMove(task, interval):
def intervalDownMove(task, interval):
def intervalEdit(task, interval, parameter, value):
def intervalParametersSet(task, interval, parameters):
def intervalLoadCasesAdd(task, interval, loadCases):
def intervalLoadCaseDelete(task, interval, loadCase):
def intervalLoadCaseEdit(task, interval, loadCase, parameter, value):
def intervalLoadCaseFunctionSet(task, interval, loadCase, function):
def intervalSubGeneratorRun(task, interval, subGenerator):
def intervalSubGeneratorEdit(task, interval, subGenerator, parameter, value, unit = None):
def intervalSubGeneratorSet(task, interval, parameter, subGenerator):
def intervalConstructionProcessAdd(task, interval, baseElementType):
def intervalConstructionProcessDelete(task, interval, constructionProcess):
def intervalConstructionProcessEdit(task, interval, constructionProcess, parameter, value):
```

- Examples:

```
task(name = "Task")
interval(task = "Task", interval = "Interval 0")
intervalParametersSet(task = "Task", interval = "Interval 0", parameters = "Newton-Rhapson")
interval(task = "Task", interval = "Interval 0")
intervalParametersSet(task = "Task", interval = "Interval 0", parameters = "Newton-Rhapson")
```

```
interval(task = "Task", interval = "Interval 0 (1)")
intervalParametersSet(task = "Task", interval = "Interval 0 (1)", parameters = "Newton-Rhapson")
intervalCopy(task = "Task", interval = "Interval 0", newInterval = "Interval 4 (copy of #2)")
intervalDownMove(task = "Task", interval = "Interval 0")
intervalDownMove(task = "Task", interval = "Interval 0")
intervalDelete(task = "Task", interval = "Interval 0")
taskEdit(name = "Task", parameter = "Execute", value = "False")
taskEdit(name = "Task", parameter = "ActivateBaseTemperature", value = "True")
taskEdit(name = "Task", parameter = "Temperature", value = 1)
taskEdit(name = "Task", parameter = "AutomaticName", value = "False")
taskEdit(name = "Task", parameter = "Name", value = "My task")
intervalParametersSet(task = "My task", interval = "Interval 0 (1)", parameters = "Arc-Length")
intervalEdit(task = "My task", interval = "Interval 0", parameter = "ActivateTimeDefinition", value = "True")
intervalEdit(task = "My task", interval = "Interval 0", parameter = "ActivateTimeDefinition", value = "True")
intervalEdit(task = "My task", interval = "Interval 0", parameter = "ActivateTimeDefinition", value = "True")
intervalEdit(task = "My task", interval = "Interval 0", parameter = "DurationUnit", value = "hour")
intervalEdit(task = "My task", interval = "Interval 0", parameter = "ConstructionProcess", value = "True")
intervalLoadCasesAdd(task = "My task", interval = "Interval 0", loadCases = "Body force")
intervalLoadCasesAdd(task = "My task", interval = "Interval 0", loadCases = "Supports")
function(name = "Function")
intervalLoadCaseFunctionSet(task = "My task", interval = "Interval 0", loadCase = "Supports", function = "Function")
intervalLoadCaseEdit(task = "My task", interval = "Interval 0", loadCase = "Body force", parameter = "Multiplier", value = 2)
intervalLoadCaseEdit(task = "My task", interval = "Interval 0", loadCase = "Supports", parameter = "IsActive", value = "False")
intervalEdit(task = "My task", interval = "Interval 0", parameter = "FatigueInterval", value = "Calculate")
intervalEdit(task = "My task", interval = "Interval 0", parameter = "FatigueCodLoadCoeff", value = 2)
```