COMPASS
www.compassis.com

CompassLIB Manual

A toolkit for the creation of problemtypes for GiD

Compass Ingeniería y Sistemas http://www.compassis.com
Tel.: +34 932 181 989 - Fax.: +34 933 969 746 - E: info@compassis.com - C/ Tuset 8, 7o 2a, Barcelona 08006 (Spain)

Table of Contents

1 CompassLIB manual

CompassLIB is a collection of tools, which facilitates the development of advanced problem types for customizing the personal pre and post processor system GiD for computer simulation with Finite Element Method. A problem type is a collection of utilities, which allow the user to interaction easily with them by means of a Graphical User Interface (GUI), and facilitate the definition and introduction of all the data necessary for carrying out a particular calculation. In order for GiD to prepare data for a specific analysis program, it is necessary to customize it. The customization is defined in GiD by means of a problem type.

The traditional GiD problemtype is defined with a directory called the problemtype name and extension gid, and this directory contains several files. This set of files define the problemtype and contain the full functionality for customizing the preprocess. The next step is that the result of the analysis should be output in the GiD format in order to visualize results in the GiD postprocess. This step can be performed by using the gidpostlibrary, provided with GiD.

The new CompassLIB for problemtypes creation is based on these capabilities and files but uses additional capabilities provided with the toolkit and does not need anymore most of the preceding files.

Introduction

GiD is a pre/post processing system for Computer Simulation with methods like Finite Element, Finite differences, etc.

In order for **GiD** to prepare data far a specific analysis program, it is necessary to customize it. Once the customization step is completed, **GiD** can:

- Allows the user to introduce boundary conditions like constraints, loads, fixed and initial velocity, etc.
- Deal with the materials database and with other properties of the geometry
- Allows the user to introduce all the necessary data for the analysis
- Create or import the geometry and apply the data to it
- Generate the mesh
- Output all the data to a file in the native format of the analysis program

The customization is defined in **GiD** by means of a problem type. The traditional **GiD problemtype** is defined with a directory with the problemtype name and extension **gid** this directory contains several files. Most of them have the same name than the directory with different file extensions.

Let's assume that the name of the problemtype is **PROJECT**. In this case, we would have the following items:

- One directory with name PROJECT.gid
- Inside this directory several files, among others:
  - PROJECT.prb

- PROJECT.cnd
- PROJECT.mat
- PROJECT.bas
- ...
- several TCL files

This directory will be installed in the **problemtypes** directory in the **GiD** distribution or in a subdirectory of it.

This set of files define the problemtype and contain the full functionality for customizing the pre process. The next step is that the result of the analysis should be output in the GiD format in order to visualize results in the GiD post process. This step can be performed by using the **gidpostlibrary,** provided with **GiD**.

The new **Compass toolkit** for problemtypes creation is based on these capabilities and files but uses additional capabilities provided with the toolkit and does not need anymore most of the preceding files.

The new problemtype is defined with a directory similar to the previous that contains the following files:

- PROJECT.prb (not used anymore)
- PROJECT.cnd (used but should not be modified by the problemtype creator)
- PROJECT.mat (not used)
- PROJECT.bas (not used. Should be existing but void
- PROJECT_default.spd (the main configuration file. It is defined in XML format and contains all the definition of all the data that defines the analysis like boundary conditions, loads, materials, loadcases, etc.
- PROJECT.tcl main TCL file. Contains the initialization routines
- scripts/b_writecalcfile.tcl Contains the output description to the file for the analysis. It is programmed in TCL.
- several other TCL files

A new system for Problem Types creation

Why a new toolkit for problem types creation?

- It couples all the common features of the different problem types, which makes easy its creation.
- It adds additional capabilities to the traditional GiD Problem Types.
- It takes advantage of the XML format features and its hierarchical structure. It stores data more efficiently.
- It permits to process automatically XML documents on a physical data tree view on the GiD window for interfaces creation.
- It facilitates the automatic creation of standard windows in the data tree to enter input dates.

- It couples geometry or mesh entities with identical properties into the called groups using these standard windows.
- It allows to apply efficiently geometry properties and boundary conditions (i.e. constraints, loads, materials…) into groups.
- It is also possible to edit easily the groups properties.
- In order to configure GiD for a specific type of analysis it is possible to set the data tree hiding the required parts automatically.

Advantages of using the XML format for interfaces creation

- XML (Extensible Markup Language) is an open and flexible standard.
- It is endorsed by software market leaders, it supports Unicode and it is a W3C recommendation (World Wide Web Consortium). It is becoming more and more popular in the area of storing and transporting information.
- The syntax rules are very simple, logical, concise, easy to learn and to use. The XML documents are human-legible, clear and easy to create.
- The information is stored in plain text format. It can be viewed in all major of browsers and it is designed to be self-descriptive.
- The elements in a XML document form a tree-structure that starts at "the root" and branches to "the leaves" with different relationships between the nested elements. It allows to aggregate efficiently elements.
- The new Toolkit takes advantage of this hierarchical structure to convert automatically the main XML file to a physical tree on the GiD window.
- The XML elements are defined using not fixed "tags" and can have "attributes", which provide additional information about elements.

The new Problem Type structure

File extension	Description	New Problem Type
name.prb	Problem and intervals data	Not used
name.cnd	Conditions definition	Used, should not be modified
name.mat	Material properties	Not used
name.bas	Information for data input file	Not used, should be void
name.tcl	Main TCL file, initialization	Used
name_default.spd	Main configuration file, XML-based	Used
scripts/writecalcfile.tcl	Output description to the file for analysis	Used

Table 1: Main files that configure the new Problem Types using the Toolkit

It is defined using a directory with the problem type name and a set of files, but does not need most of the original files anymore.

1. A TCL initialization file is used to create complex windows or menus.

2. The main configuration file in XML format contains the definition of all the data (except the geometry) necessary to perform an analysis. It is necessary to modify this XML document in order to add conditions or general data to the Problem Type.

3. A TCL file introduced in the <scripts> folder determines the way in which the final information has to be written inside the input files that will be read by the solver.

Toolkit conventions

The Toolkit defines its own XML tags, which clearly describe its content. It permits to create automatically a physical data tree and standard windows, which facilitates the introduction of data.

- *TDOM library*

The tDOM library is used by the Toolkit due to it combines high performance XML data processing with easy and powerful Tcl scripting functionality. tDOM is one of the fastest ways to manipulate XML and it uses very little memory in the process of creating a DOM tree from a XML document.

In tDom terminology we call 'field' to the 'Element name' and 'parameter' to the 'attribute'. All data is stored in fields and parameters, where the parameters can contain a value, a xpath expression or a [TCL command].

- *XPATH*

XPath is a language for addressing parts of an XML document using path notations.

It is used for navigating through the hierarchical structure of a XML document to extract information. It is based on a tree of nodes representing the XML file.

It provides the ability to navigate around the tree, selecting nodes from it and computing string-values.

xpath expression -> A search is performed and the result is substituted in the parameter when necessary.

[TCL command] -> The command between brackets is executed when necessary and the return value is replaced inside the parameter.

Conclusions

- Using the Toolkit it is possible to create tree interfaces easily and to store data more efficiently and automatically.
- It couples all the common features of the different Problem Types.

- It facilitates the introduction of all the data to transfer to an analysis program.
- It adds additional capabilities to the traditional GiD Problem Types.
- It is based on a XML hierarchical structure and an automatic physical tree view.
- It facilitates the automatic creation of standard windows to enter data.
- It permits to couple entities with identical properties into groups.
- It makes possible to apply efficiently geometry properties and boundary conditions into groups and to edit their properties easily.
- It allows to fix the data tree hiding concrete parts if this is convenient for a specific type of analysis.
- The Plaxis-GiD interface program is a realistic application of this powerful tool.

Features and advantages

- It couples all the common features of the different problem types, which makes easy its creation.
- It adds additional capabilities to the traditional GiD Problem Types.
- It takes advantage of the XML format features and its hierarchical structure. It stores data more efficiently.
- It permits to process automatically XML documents on a physical data tree view on the GiD window for interfaces creation.
- It facilitates the automatic creation of standard windows in the data tree to enter input dates.
- It couples geometry or mesh entities with identical properties into the called groups using these standard windows.
- It allows to apply efficiently geometry properties and boundary conditions (i.e. constraints, loads, materials□) into groups.
- It is also possible to edit easily the groups properties.
- In order to configure GiD for a specific type of analysis it is possible to set the data tree hiding the required parts automatically.

Learn the advantages of using the XML format for GUI creation:

- XML (Extensible Markup Language) is an open and flexible standard.
- It is endorsed by software market leaders, it supports Unicode and it is a W3C recommendation (World Wide Web Consortium). It is becoming more and more popular in the area of storing and transporting information.
- The syntax rules are very simple, logical, concise, easy to learn and to use. The XML documents are human-legible, clear and easy to create.
- The information is stored in plain text format. It can be viewed in all major of browsers and it is designed to be self-descriptive.
- The elements in a XML document form a tree-structure that starts at □the root□ and

branches to □the leaves□ with different relationships between the nested elements. It allows to aggregate efficiently elements.

- The new Toolkit takes advantage of this hierarchical structure to convert automatically the main XML file to a physical tree on the GiD window.
- The XML elements are defined using not fixed □tags□ and can have □attributes□, which provide additional information about elements.

gid_group_conds library

In order to add **conditions**, **general data**, or **units** information to the *problemtype*, it is necessary to modify file **{PROBLEMTYPE}_default.spd.** This is a file in XML format. This file contains all the definition of all the data necessary for the analysis.

A description of the file follows:

The XML tree below contains the definition of all the data (except the geometry), necessary for a computer simulation analysis. It contains all the data that must be filled by the user in order to perform the analysis

We call 'field' to the 'Element name' in TDOM terminology and parameter to the 'attribute' in TDOM terminology. All data is contained in fields and parameters and there is no data as text leaf of a element.

All parameters that must contain a list of values, like the 'ov' or 'values' fields, contain a comma separated list of these values. It must be taken care that no one of the individual values can contain a comma in its name

All parameters can contain a literal value or: {xpath expression} or [TCL command]. In the first case, an Xpath search is performed in the tree and the result is substituted in the parameter when necessary. In the second case, the TCL command between brackets is executed when necessary and the return value is replaced inside the parameter. Typically, the TCL procs defined with the field 'proc' are used. Example:

```
<condition n='elements' ... state='[pt]'/>
<proc n='pt'>
  <![CDATA[
  set problemtypes [split [$domNode selectNodes string(@pt)] ,]
  set pt [$domNode selectNodes {string(/*/blockdata[@n='general_data']/
            container[@n='problem']/value[@n='problemtype']/@v)}]
  if { $pt eq "" } { error "error: pt cannot be void" }
  if { [lsearch -exact $problemtypes $pt] != -1 } {
```

```
    return normal
    } else { return hidden }
    ]]>
  </proc>

  <value ... dict='{/beasy_data/units/units_system/@dict}'
     values='{/beasy_data/units/units_system}' v='SI'>
```

Description of the main fields

- **blockdata:** Represents a set of properties with some kind of relationship. It contains several 'values' fields with the actual data. It can also contain other 'blockdata' and 'condition' fields If it has the 'sequence' parameter activated, it is possible for the user to create consecutive repetitions of the full block data in order to represent, for example, loadcases with all its conditions inside.
- **condition:** It contains values and it can be applied to groups. For every applied group, a dependent set of values will be created that belong to that group for this condition.
- **container:** Is is a simple way for grouping data for better visualization
  - **value:** The main unit to store data. One or several of them are contained inside any of the previous field.
- **dependencies:** When one 'value' changes its value, the 'dependencies' permit to force a change in other values.
  - **value:** It allows to define a condition to execute a dependence.
  - **att, atti, i=1,2:** Indicates to which attributes of a node affect a change in one value.
  - **v,vi, i=1,2:** Indicates the new value for atti, i=1,2. It can be normal, hidden or disabled.
  - **default:** Default value for the condition. It permits to execute a dependence.
  - **condition:** Allows to define the condition in order to execute a dependence.
  - **actualize:** Permits to actualize the specified field.
  - **check**: Boolean value that permits the user changes the data tree showed information, depending on the dependence value specified.
- **proc:** Permits to define a TCL proc. The code will receive an implicit argument with name 'domNode' that represents the TDOM node in the calling field context.

Description of the main parameters

- **n:** Name used to reference the field, especially when writing the .dat file
- **pn:** Label that will be visualized by the user (can be translated)
- **dict:** name-show name pairs that affect current leaf of the tree and that can be translated
- **values:** Finite number of values that the user can apply to the property
- **editable:** Boolean value that permits or not to the user to edit the property
- **v:** The value or default value for a 'value' field

- **help:** Permits to create a pop-up help for the fields
- **state:** Can be: normal or disabled. It permits to define a Tcl function.
- **active:** Can be 0 or 1
- **sequence:** Permits a 'blockdata' field to by duplicated and copied by the user in order to create several sets of something. Like several load cases with its 'value' and 'condition' included
- **sequence_type:** For *blockdata*. Can be:
  - **any** The list can be void (this is the default)
  - **non_void_disabled** At least there needs to be one element. It can be disabled.
  - **non_void_deactivated** At least there needs to be one element. It can be deactivated.
- **editable_name:** can be '' or 'unique'. Unique means that it is not possible to use the same name ('pn' field), for two different 'sequence' 'blockdata'
- **is_value:** This is a simplification for 'condition' fields that should have a unique 'value' field inside with the same name. If this flag is set to '1', it is not necessary to define the 'value' field but works as if it were defined
- **ov, ovi, i=1,2:** Indicates to which entity types can a 'condition' be applied. Can be one or several of: point,line,surface,surface_as_volume,volume.
- **ov_default:** Indicates the default entity type which a 'condition' can be applied.
- **ovm, ovmi, i=1,2:** Indicates to which entity can a 'condition' be applied. It can be element, node, face element or void.
- **function:** Contains a Tcl command, which is executed when is called. It permits to create or edit a function for a determined field.
- **function_func:** Permits to define a TCL function.
- **edit_command:** Permits to call a TCL proc when necessary.
- **icon:** Permits to put an image in the conditions menu.
- **fieldtype:** Permits to introduce a text. It can be
  - **longtext:** A text box is created.
- **actualize_tree:** Allows to actualize all the input dates in data tree.
- **actualize:** Permits to actualize a specified field in data tree.

Description of the units

There is an initial units definition section that include all the relevant units for the problem It includes:

- **units_system:** All the units can belong to one of the units systems or not specify it, that means that the unit belongs to any of them. All units used in a problem must belong to the same unit system
- **unit_magnitude:** Every magnitude that can contain several units
- **units:** every unit inside a magnitude
- **factor:** It is the factor that permits: 1 unit=factor*unit(SI) example: 1mm=10e-3 m

Every property that needs it, contains its magnitude and units. Example:

```
<value n='x' pn='x' state='normal' v='0.0' unit_magnitude='L' units='m'/>
```

Description of the symbols

Every condition can have a symbol, that will be drawn when the user selects **Draw symbols** in the interface. The symbol is defined by a field **symbol** inside the condition. The available parameters are:

- **proc:** Includes the name of a TCL proc to be defined in the TCL files of the problemtypes. File 'drawsymbols.tcl' inside directory 'scripts' contains some of the definitions. In that proc, OpenGL is used to make the real drawing. There are functions to automatically import GiD mesh files
- **orientation:** can be 'local' or 'global'. 'global' means that the symbol defined in the proc will be draw with its axes corresponding to that of the global axes of the model. 'local' means that the symbol will be drawn related to a local axes system dependent on the entity. For lines, this local axes system will have x' axe parallel to the line. For surfaces, the z' axe will be parallel to the surface normal

New units system

A new units system has been implemented that will permit to deal with any type of unit in RamSeries, Tdyn and SeaFEM.

RamSeries has already been adapted to work with this system. It can be used as a base to update the other problemtypes.

Thinks to check for updating a problemtype:

- The contents of the "units" sub-tree in the XML can be deleted. All this information is now in file **units.xml** inside package **gid_groups_cond**. Once everything is working, this section can be used to change some defaults or to add specialized units. The contents of this sub-tree needs to have the same format that the one in file units.xml and can have part or all of the information contained there for magnitudes/units.
- The field to choose the **Units System** is special. It has the attribute units_system_definition="1", does NOT contains **v** attribute and contains a unique dependency referring to the **units fields**.
- The fields to choose the **Default units**" are special. They contain the attribute: unit_definition="MAGNITUDE" being **MAGNITUDE** the one to be used in that field. They contain NO dependencies.
- The units and magnitudes for the normals fields are defined as before. As now there are more derived magnitude names (like **Pressure, P**), It is convenient to change magnitudes like **F/L^2** by **P**. Check units.xml for all the magnitude names.
- The fields for **units** in the **Start window** have been modified. They use now a mega-widget called **entry_units**.
- The code for writing data to the calculation file has been modified. Some new functions are used. Among them, to print the units in an ASCII representation use:

gid_groups_conds::**give_printable_unit** and
gid_groups_conds::**convert_value_to_printable_unit**

- After the changes, increase the version number in the XML tree.
- Packages to upgrade: **gid_groups_conds, compass_utils, fulltktree**

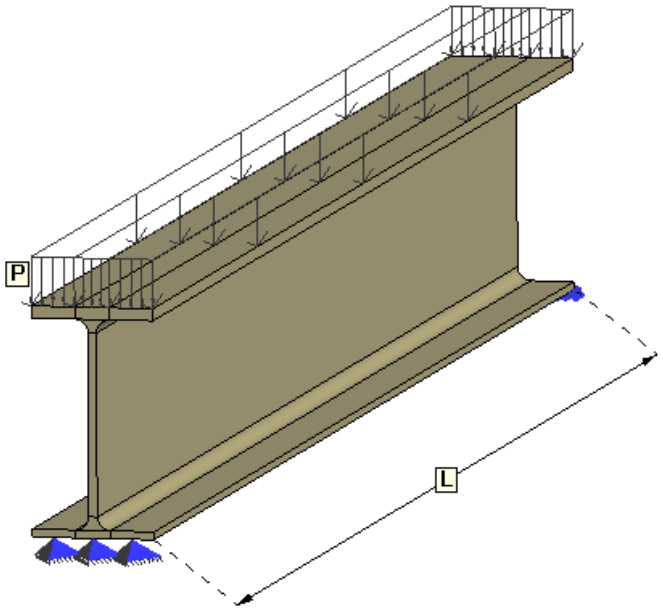
Parametric models library

The parametric model library is a set of powerful features contained in CompassLIB, also called gid_groups_conds, which allows to create parametric geometrical models in GiD automatically. Furthermore, this library also permits to apply appropriate groups directly in the data tree, obtaining the necessary input data for the finite element analysis.

In this way, the *param_creator::create_window* procedure can be used in order to generate a default window, which will read all the necessary data for creating the parametric model. It is a SVG file describing a two-dimensional graphic together with an input file containing the groups information and the rest of dimensions. Notice that both files should be in XML format and located inside the same folder.

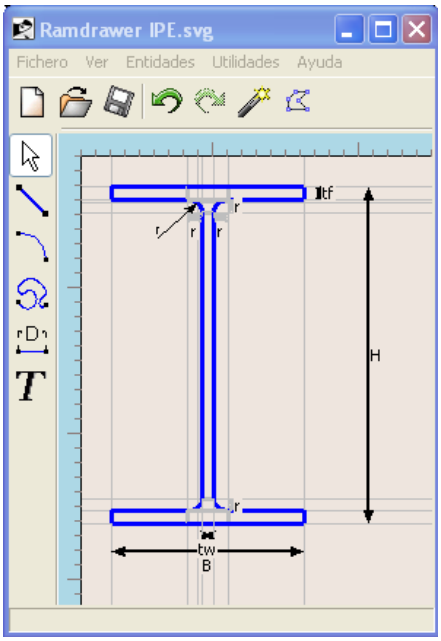
A 3D application example of the parametric model library is presented here, where all the input data necessary for the analysis will be created automatically in the case of a beam. Input data to be defined by the user is the following:

- Dimensions of the I-shape cross section of the beam.
- Length of the beam **L.**
- Steel type and a distributed load with magnitude **P**.



SVG is a language for describing two-dimensional graphics in XML, which provides complete access to all elements, attributes and properties of a vector graphic shape. In this way, the following figure shows the 2D cross-section shape of the beam. Notice that its corresponding SVG file (see IPE.svg file) was created using Ramdrawer, it is our Vector Graphics Editor.

In order to create new groups and to assign geometrical entities into them automatically using the parametric models library it is necessary to add the **ramdraw:groups** field and a group name as attribute when writing the SVG file. For instance, "*<point x='$x' y='$y' z='$z' ramdraw:groups='Top flange'/>*" assigns a group called '*Top flange*' into the field of type 'point'. For more information about SVG format see Scalable Vector Graphics (SVG) specification web site http://www.w3.org/TR/SVG.



Once the SVG file has been created containing the I cross section shape of the beam, it is also necessary to write an input file in XML format with the rest of information regarding to groups and dimensions of the model (see IPE_3D.xmlfile). In this way, the 'stage' field couples data into different blocks of type:

- **geometry** Field used to describe geometrical entities.
  - **delete_entities** Delete group entities
    - **groups** Set of groups to be deleted.
    - **also_delete_groups** A Boolean field (1 - 0) is used to delete groups.
  - **create_from_svg** SVG id.
  - **nurbsurface** It allows to create surfaces.
    - **creation_type** Drawing type option.

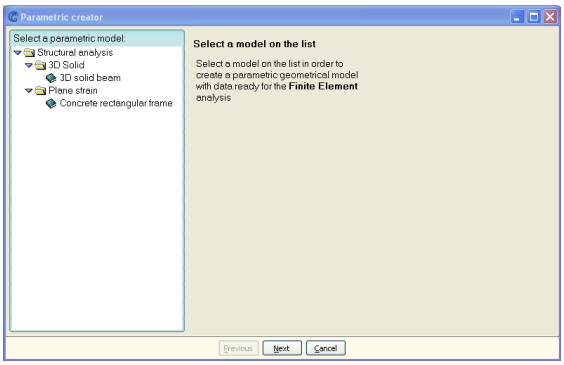
- **automatic** All possible surfaces with the number of sides given by the user will be created automatically.
- **numlines** Selected number of sides.
- **lines** Set of groups to be assigned to line entities.

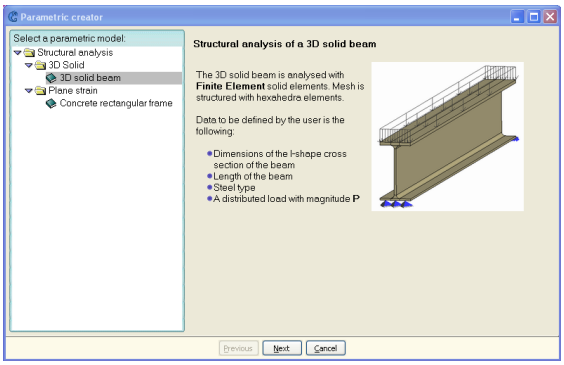
- **rename_groups:** It permits to rename groups. In order to separate the final group names // must be used .
  - **names** A set of groups is required.
  - **prefix** A prefix can be added.
  - **suffix** A suffix can be added.
  - **also_layers** Boolean field (1 - 0) used to rename also layers names.
- **condition** It permits to assign groups in data tree. Field 'href' contains a xpath expression and 'groups' indicates the list of groups to be applied.
- **values/container/blockdata** It permits to select 'values', 'container' or 'blockdata' fields in data tree.
  - **value** Field 'href' contains a xpath expression and field 'v' its applied value.
- **operations** It allows to select a group of entities and copy them with a movement operation performed, either translation, rotation or sweep.
  - **operation** Field used to define each operation type.
    - **type** The type of movement needs to be chosen. The options are:
      - **translation** Two points are defined and relative movements can be obtained by defining the first point as 0,0,0 and considering the second point as the translation vector.
        - **copy** It duplicates entities
        - **extrude** Extrude selected entities
        - **vector1** Translation vector
        - **entities_type** Indicates to which entity types will be applied the condition
        - **entities** Indicates the list of groups.
        - **extrude_prefix** Indicates a prefix name for the groups along the length.
        - **dest_prefix** Indicates a destination prefix name for the groups along the length.
      - **rotation** It is necessary to enter two points in 3D, which define the axis of rotation and its orientation.
        - **copy** It duplicates entities
        - **extrude** Extrude selected entities
        - **angle** Enter the angle of rotation in degrees.
        - **entities_type** Indicates to which entity types will be applied the condition
        - **entities** Indicates the list of groups.
      - **align** This is an option for moving figures from a generic position to the desired one. Set the new location specifying three source points and three destination points to define the axes movement. Field 'entities_type' indicates to which entity types will be applied the condition and 'entities' indicates the list of groups.
      - **sweep** This is an option for copying figures along a line (a path line).

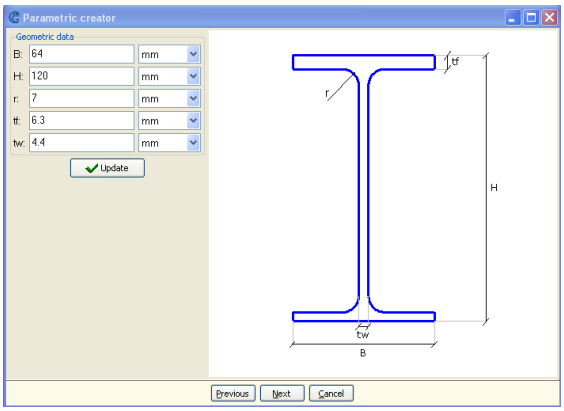
- **copy** It duplicates entities
- **extrude** Extrude chosen entities
- **pathline** Path line selected for copying figures along it.
- **entities_type** Indicates to which entity types will be applied the condition
- **entities** Indicates the list of groups.
- **extrude_prefix** Indicates a prefix name for the groups along the length.
- **dest_prefix** Indicates a destination prefix name for the groups along the length.

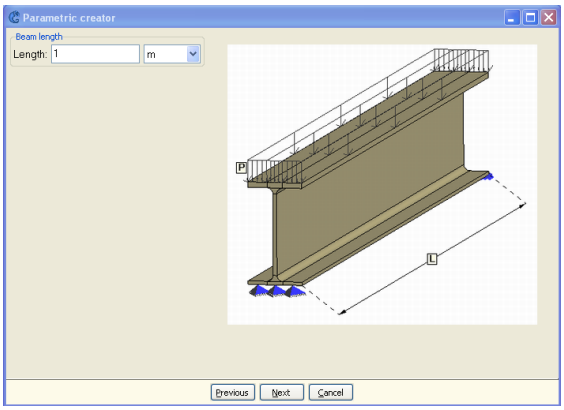
- **meshing** It allows to select meshing options.
  - **structured** Field 'entities_type' indicates to which entity types will be applied the condition and field 'entities' indicates the list of group names.
  - **divisions** Field 'lines' indicates the group names which the condition will be applied and 'number' indicates the number of divisions.

The following figures show the automatic wizard, which has been created using the parametric model library.



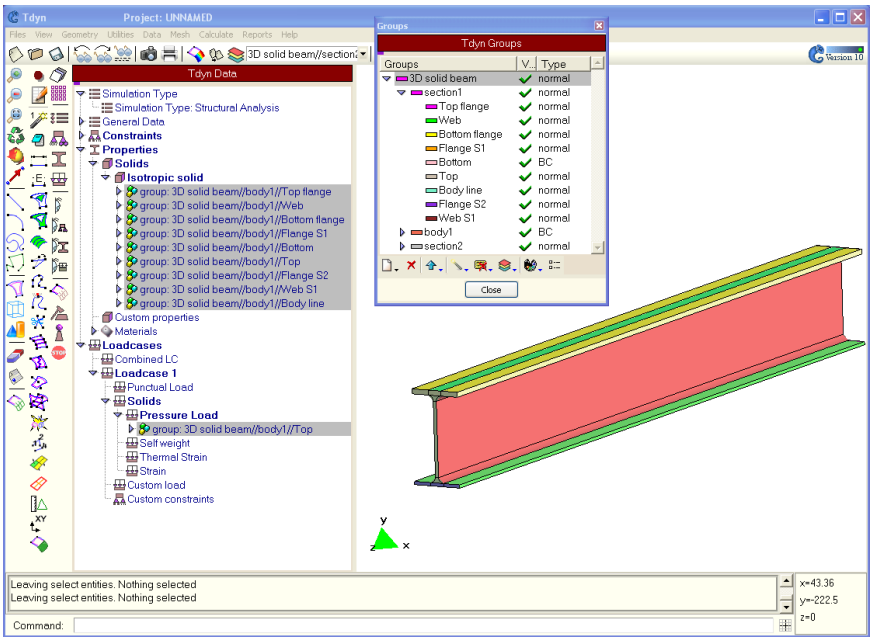






After all the data for the analysis has been introduced and the process of creation has finished, a new geometrical model will be created and the conditions previously defined will be applied to it. Therefore, after calling the creation process all the input data necessary for

the finite element analysis will be obtained. Actually the figure below shows the final geometry together with all the groups created and applied automatically to their corresponding entities.

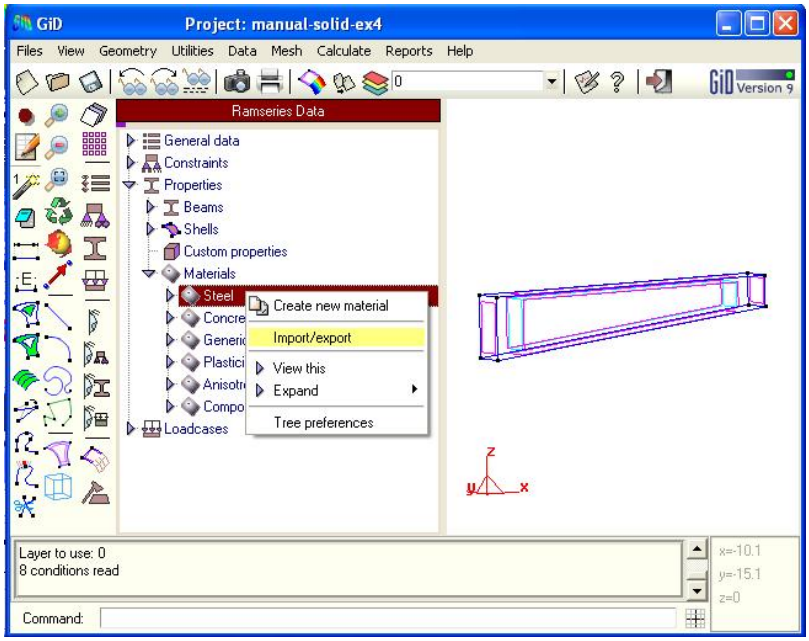


Import export materials

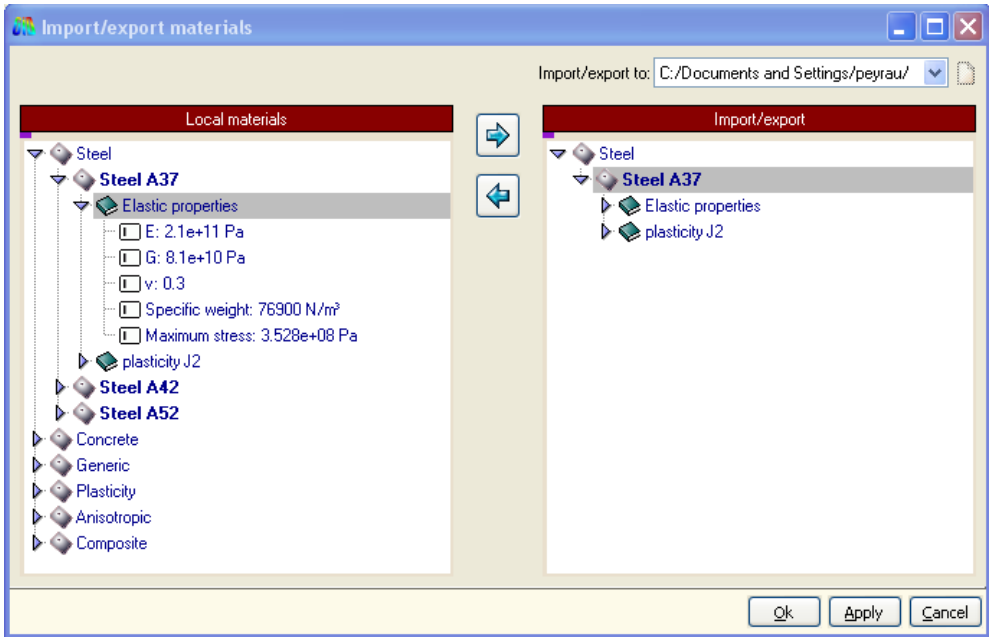
Help manual description

A material is composed by a set of material properties, which can be applied to geometry entities (i.e. beams, shells, solids, etc.). In this way, the *Import/export* tool allows to handle material properties easily. This tool is located inside the right mouse menu when a particular material or the materials collection is selected in the data tree.

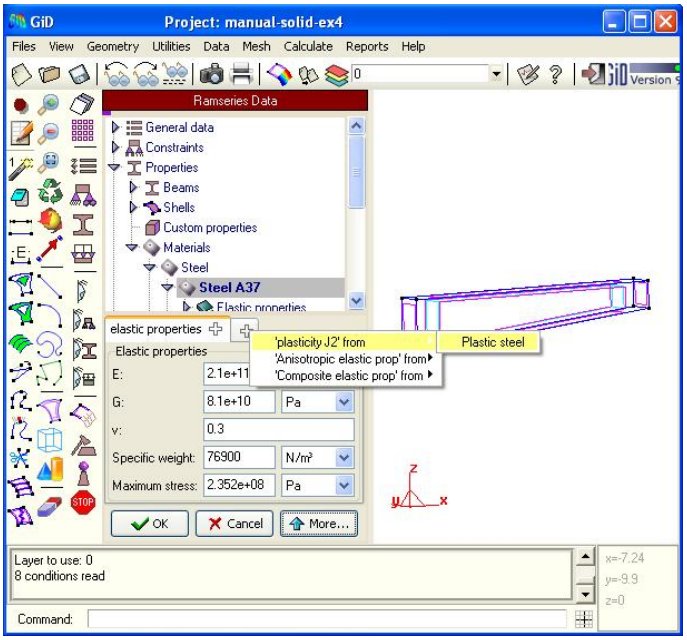
- In order to activate the Import/export materials tool, it is necessary to call n="materials" to the global container in the spd-file. Moreover, although the intermediate containers could be defined using any attribute n, each final material blockdata has to be called n="material".

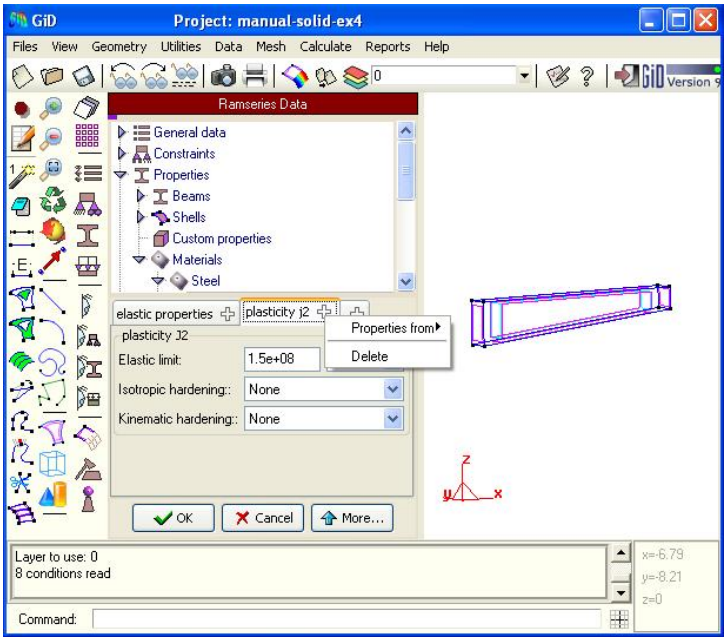


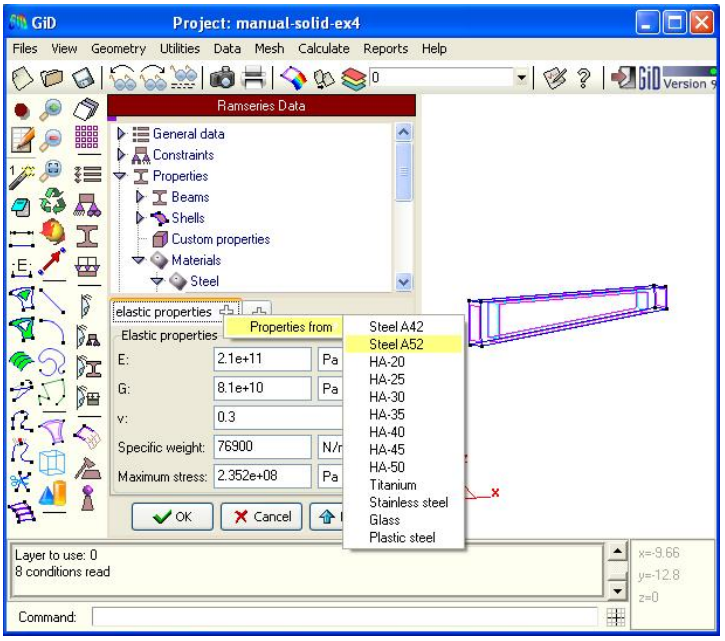
- The "Import/export materials" window contains two different material data trees. At the left side there is the local materials list associated to the current model. At the right side there is a material data tree list, which could be imported or exported depending on the user interests. It is possible to import/export materials in four different ways: to a global database active, to a global database inactive, to import from a selected file in XML-format or to export to a XML-file. The global database active is the database shown in the data tree located at the left side of the GiD window when a new model is created, whereas the global database inactive is an internal database which does not affect the default data tree for new models. It could be used to store odd materials in order to import them for a particular model. The global database active also allows to import the original default materials clicking on the button located just at the right side of the "Import/export to" combo-box, which facilitates to recover easily the original input data for materials. It is also possible to import or export selected materials from particular XML-files without modifying the global database. It should be noted that the creation of new materials can be done directly using the right mouse menu in the import/export materials window.



- The *"More..."* button permits to add other collections of material properties to a specific material. It should be noted that it is necessary to click on the addition button in order to add/delete material properties from other sets of materials for each defined material. Then the selected material properties are coupled as tab-sheets, depending on the user requirements. See the following images:




GiD
Project: manual-solid-ex4
Files View Geometry Utilities Data Mesh Calculate Reports Help
Ramseries Data
General data
Constraints
Properties
Beams
Shells
Custom properties
Materials
Steel
elastic properties
plasticity J2
Properties from
Delete
plasticity J2
Elastic limit:
1.5e+08
Isotropic hardening::
None
Kinematic hardening::
None
OK
Cancel
More...
Layer to use: 0
8 conditions read
Command:


GiD
Project: manual-solid-ex4
Files View Geometry Utilities Data Mesh Calculate Reports Help
Ramseries Data
General data
Constraints
Properties
Beams
Shells
Custom properties
Materials
Steel
elastic properties
Elastic properties
Properties from
Steel A42
Steel A52
HA-20
HA-25
HA-30
HA-35
HA-40
HA-45
HA-50
Titanium
Stainless steel
Glass
Plastic steel
E:
2.1e+11
G:
8.1e+10
v:
0.3
Specific weight:
76900
Maximum stress:
2.352e+08
OK
Cancel
Layer to use: 0
8 conditions read
Command: