

GiD

THE PERSONAL PRE
AND POSTPROCESSOR

The universal, adaptive and user-
friendly pre- and post-processing
system for computer analysis
in science and engineering

Reference Manual
Version 8

Developers

Ramon Ribó
Miguel de Riera Pasenau
Enrique Escolano
Jorge Suit Pérez Ronda
Abel Coll Sans

Cover design

Lluís Font González

For further information please contact

International Center for Numerical Methods in Engineering
Edificio C1, Campus Norte UPC
Gran Capitán s/n, 08034 Barcelona, Spain

<http://www.gidhome.com>
gid@cimne.upc.edu

Depósito legal: B-34.736-02
ISBN Reference Manual: 84-95999-95-1
ISBN Obra Completa: 84-95999-96-x
© CIMNE (Barcelona, Spain)

Table of contents

INTRODUCTION.....	1
USING THIS MANUAL.....	2
GID BASICS.....	3
INVOKING GID.....	6
USER INTERFACE	8
MOUSE OPERATIONS	12
COMMAND LINE	12
USER BASICS	13
POINT DEFINITION	13
<i>Picking in the graphical window.....</i>	<i>14</i>
<i>Entering points by coordinates.....</i>	<i>14</i>
<i>Base</i>	<i>16</i>
<i>Selecting an existing point.....</i>	<i>16</i>
<i>Point in line</i>	<i>17</i>
<i>Point in surface</i>	<i>17</i>
<i>Tangent in line.....</i>	<i>17</i>
<i>Normal in surface.....</i>	<i>17</i>
<i>Arc center</i>	<i>18</i>
<i>Grid</i>	<i>18</i>
ENTITY SELECTION.....	18
QUIT.....	20
ESCAPE	21
FILES.....	22
NEW.....	22
OPEN.....	22
SAVE.....	23
SAVE AS	24
IMPORT	24
<i>IGES.....</i>	<i>24</i>
<i>DXF.....</i>	<i>25</i>
<i>Parasolid</i>	<i>26</i>
<i>ACIS</i>	<i>26</i>
<i>VDA</i>	<i>26</i>
<i>Rhino</i>	<i>27</i>
<i>Shapefile</i>	<i>27</i>
<i>NASTRAN mesh.....</i>	<i>27</i>
<i>STL mesh</i>	<i>28</i>

<i>VRML mesh</i>	28
<i>3DStudio mesh</i>	29
<i>GiD mesh</i>	29
<i>Surface mesh</i>	31
<i>Batch file</i>	31
<i>Insert GiD geometry</i>	33
EXPORT.....	33
<i>IGES</i>	33
<i>DXF</i>	34
<i>ACIS</i>	34
<i>GiD mesh</i>	35
<i>Text data report</i>	35
<i>ASCII project</i>	35
<i>ON layers</i>	35
<i>Calculation file</i>	36
<i>Using a .bas template</i>	36
PREPROCESS/POSTPROCESS.....	37
PRINT TO FILE.....	37
PAGE/IMAGE SETUP.....	38
PRINT.....	39
RECENT FILES.....	39
QUIT.....	39
VIEW	40
ZOOM.....	40
ROTATE.....	41
<i>Rotate trackball</i>	41
<i>Rotate screen axes</i>	42
<i>Rotate object axes</i>	42
<i>Rotate center</i>	42
<i>Rotate angle</i>	43
<i>Rotate points</i>	43
<i>Plane XY (Original)</i>	44
<i>Plane XZ</i>	44
<i>Plane YZ</i>	44
PAN.....	44
REDRAW.....	45
RENDER.....	45
PERSPECTIVE.....	48
CLIP PLANES.....	48
LABEL.....	49
ENTITIES.....	50

NORMALS.....	50
HIGHER ENTITIES	51
SAVE/READ VIEW	51
BACKGROUND IMAGE	51
IMAGE TO CLIPBOARD	52
MULTIPLE WINDOWS.....	52
MODE.....	52
GEOMETRY.....	54
VIEW GEOMETRY	54
CREATE.....	54
<i>Point creation</i>	54
<i>Straight line creation</i>	55
<i>NURBS line creation</i>	55
<i>Parametric line</i>	56
<i>Polyline creation</i>	58
<i>Arc creation</i>	59
<i>NURBS surface creation</i>	60
<i>Parametric surface</i>	62
<i>Planar surface creation</i>	63
<i>4-sided surface creation</i>	63
<i>Automatic 4-sided surface creation</i>	64
<i>Contact surface creation</i>	65
<i>Surface mesh</i>	65
<i>Volume creation</i>	65
<i>Contact volume creation</i>	66
<i>Object</i>	67
DELETE	69
EDIT	69
<i>Move point</i>	70
<i>Divide</i>	70
<i>Line operations</i>	71
<i>Swap arc</i>	72
<i>Polyline</i>	72
<i>SurfMesh</i>	73
<i>Edit NURBS line/surface</i>	73
<i>Convert to NURBS line/surface</i>	76
<i>Simplify NURBS line/surface</i>	76
<i>Hole NURBS surface</i>	76
<i>Collapse</i>	76
<i>Uncollapse</i>	77
<i>Intersection</i>	77

<i>Surface boolean operations</i>	79
<i>Volume boolean operations</i>	79
UTILITIES	81
UNDO	81
PREFERENCES	82
LAYERS	92
TOOLS	94
TOOLBARS	95
SAVE WINDOW CONFIGURATION	96
MOVE SCREEN OBJECTS	96
COORDINATES WINDOW	96
READ BATCH WINDOW	98
COMMENTS	99
ANIMATE CONTROLS	99
MACROS	100
SELECTION WINDOW	101
CALCULATOR	104
REPORT	104
NOTES	105
COPY	105
MOVE	110
STATUS	110
LIST	111
RENUMBER	112
ID	112
SIGNAL	112
SWAP NORMALS	113
DISTANCE	114
DIMENSIONS	114
REPAIR MODEL	115
DATA	116
PROBLEM TYPE	116
TRANSFORM PROBLEM TYPE	117
INTERNET RETRIEVE	117
LOAD	118
UNLOAD	118
DEBUGGER	119
CONDITIONS	120
<i>Assign condition</i>	120
<i>Draw condition</i>	121
<i>Unassign condition</i>	121

MATERIALS.....	122
<i>Assign material</i>	122
<i>Draw material</i>	122
<i>Unassign material</i>	123
<i>New material</i>	123
<i>Exchange material</i>	123
DATA UNITS.....	123
PROBLEM DATA.....	124
INTERVALS.....	124
INTERVAL DATA.....	125
LOCAL AXES.....	125
MESH.....	127
UNSTRUCTURED.....	127
STRUCTURED.....	131
ELEMENT CONCENTRATION.....	135
SEMI-STRUCTURED.....	136
QUADRATIC.....	137
ELEMENT TYPE.....	137
MESH CRITERIA.....	140
RESET MESH DATA.....	141
DRAW.....	141
<i>Sizes</i>	141
<i>Element Type</i>	142
<i>Mesh / No mesh</i>	142
<i>Structured Type</i>	142
<i>Skip entities (Rjump)</i>	143
GENERATE MESH.....	143
ERASE MESH.....	144
EDIT MESH.....	144
<i>Move node</i>	144
<i>Split Elements</i>	144
<i>Smooth Elements</i>	145
<i>Collapse</i>	146
<i>Delete nodes/elements</i>	146
SHOW ERRORS.....	147
VIEW MESH BOUNDARY.....	147
CREATE BOUNDARY MESH.....	147
MESH QUALITY.....	148
CALCULATE.....	152
CALCULATE.....	152

CALCULATE REMOTE	152
CANCEL PROCESS	152
VIEW PROCESS INFO	153
CALCULATE WINDOW	153
HELP	155
REGISTER GiD	155
REGISTER PROBLEM TYPES	156
<i>Customizing Problem type registration</i>	<i>158</i>
VISIT GiD WEB	158
ABOUT	158
POSTPROCESS OPTIONS	159
INTRODUCTION	159
FILES MENU	160
UTILITIES MENU	162
POINT AND LINE OPTIONS	165
DISPLAY STYLE	167
TEXTURES	169
COVER MESH	170
POSTPROCESS RESULTS	171
CONTOUR FILL	172
SMOOTH CONTOUR FILL	175
CONTOUR RANGES	175
CONTOUR LINES	176
SHOW MINIMUM AND MAXIMUM	176
DISPLAY VECTORS	177
ISOSURFACES	178
STREAM LINES	179
LINE DIAGRAMS	180
LEGENDS	181
GRAPHS	182
<i>Graph Lines description</i>	<i>182</i>
<i>Graph Lines options</i>	<i>183</i>
<i>Graph Lines File Format</i>	<i>183</i>
RESULT SURFACE	184
DEFORM MESH	184
DO CUTS	186
ANIMATION	188
AUTOMATIC COMMENTS	190
SEVERAL RESULTS	191
CUSTOMIZATION	193

INTRODUCTION	193
CONFIGURATION FILES	196
<i>XML file</i>	196
<i>Conditions file (.cnd)</i>	198
<i>Materials file (.mat)</i>	203
<i>Problem and intervals data file (.prb)</i>	207
<i>Conditions symbols file (.sim)</i>	209
<i>Unit System file (.uni)</i>	212
<i>Special fields</i>	214
TEMPLATE FILE.....	219
<i>General description</i>	220
<i>Commands used in the .bas file</i>	222
<i>Detailed example - Template file creation</i>	242
EXECUTING AN EXTERNAL PROGRAM.....	260
<i>Commands accepted by the GiD command.exe</i>	261
<i>Showing feedback when running the solver</i>	270
<i>Managing errors</i>	270
<i>Examples</i>	271
POSTPROCESS DATA FILES	272
POSTPROCESS MESH FORMAT: PROJECTNAME.POST.MSH, PROJECTNAME.FLAVIA.MSH.....	273
<i>Mesh example</i>	277
POSTPROCESS RESULTS FORMAT: PROJECTNAME.POST.RES, PROJECTNAME.FLAVIA.RES ...	278
<i>Gauss Points</i>	279
<i>Result Range Table</i>	285
<i>Result block</i>	286
<i>Result group</i>	289
<i>Results example</i>	295
RE-MESHING AND ADAPTIVITY	299
OLD POSTPROCESS RESULTS FORMAT	304
<i>Gauss Points (Old format)</i>	309
OLD POSTPROCESS MESH FORMAT	310
<i>Old file format: ProjectName.flavia.msh</i>	312
<i>Old file format: ProjectName.flavia.bon</i>	313
<i>Old file format: ProjectName.flavia.dat</i>	316
TCL/TK EXTENSION	318
EVENT PROCEDURES	318
CONTROL FUNCTIONS.....	322
<i>Process function</i>	322
<i>Info function</i>	323
<i>Special functions</i>	336

MANAGING MENUS.....	346
HTML SUPPORT	350
<i>HelpWindow</i>	350
<i>GiDCustomHelp</i>	351
CUSTOM DATA WINDOWS	354
<i>TkWidget</i>	354
<i>Data Windows Behaviour</i>	357
GiD VERSION.....	358
USING EXEC IN GiD.....	359
DETAILED EXAMPLE - TCL/Tk EXTENSION CREATION	359

INTRODUCTION

GiD is an interactive graphical user interface used for the definition, preparation and visualization of all the data related to a numerical simulation. This data includes the definition of the geometry, materials, conditions, solution information and other parameters. The program can generate a mesh for finite element, finite volume or finite difference analysis and write the information for a numerical simulation program in its desired format. It is also possible to run these numerical simulations from within GiD and then visualize the results of the analysis.

GiD can be customized and configured by users so that the data required for their own solver modules may be generated. These solver modules may then be included within the GiD software system.

The program works, when defining the geometry, in a similar way to a CAD (Computer Aided Design) system but with some differences. The most important of these is that the geometry is constructed in a hierarchical mode. This means that an entity of higher level (dimension) is constructed over entities of lower level; two adjacent entities will then share the same lower level entity.

All materials, conditions and solution parameters can be defined on the geometry itself, separately from the mesh as the meshing is only done once the problem has been fully defined. The advantages of this are that, using associative data structures, modifications to the geometry can be made and all other information will automatically be updated and ready for the analysis run.

Full graphic visualization of the geometry, mesh and conditions is available for comprehensive checking of the model before the analysis run is started. More comprehensive graphic visualization features are provided to evaluate the solution results after the analysis run. This postprocessing user interface can also be customized depending on the analysis type and the results provided.

Using this manual

This manual has been split into five clearly differentiated parts.

The first part, **General aspects**, provides information on the basic aspects of the program. In this way, you can gain confidence and become more familiar with the system in order to take advantage of all the available facilities.

The second part, **Preprocessing**, describes the preprocessing functionality. You will learn how to configure a project and define all its components - geometry, data and mesh.

The third part, **Analysis**, concerns to the calculation process. Although it will be performed by an independent solver, it forms part of the integrated GiD system in that the analysis can be run from inside GiD.

The fourth part, **Postprocessing**, emphasizes aspects relating to the visualization of results.

The fifth part, **Customization**, explains how to customize your files so that you can introduce and run different solver modules according to your own requirements.

Different kinds of font are used to help you follow all the possibilities offered by the code:

1. font is used for the options found in the menus and windows and for literal code.
2. font is used for special references in some parts.

GID BASICS

GiD is a geometrical system in the sense that, having defined the geometry, all the attributes and conditions (i.e. material assignments, loading, conditions, etc.) are applied to the geometry without any reference to a mesh. Only when everything has been defined is the meshing of the geometrical domain carried out. This methodology facilitates alterations to the geometry while maintaining the definitions of the attributes and conditions. Alterations to the attributes or conditions can be made simultaneously without needing to reassign the geometry. New meshes can also be generated if necessary and all the information will automatically be assigned correctly.

GiD also provides the option of defining attributes and conditions directly to the mesh once it has been generated. However, if the mesh is regenerated, it is not possible to maintain these definitions and therefore all attributes and conditions must then be redefined.

In general, the complete solution process can be defined as:

1. define geometry - points, lines, surfaces, volumes;
use other facilities;
import geometry from CAD;
2. define attributes and conditions;
3. generate mesh;
4. carry out simulation;
5. view results.

Depending upon the results in step (5) it may be necessary to return to one of the previous steps to make alterations and re-run the simulations.

Building a **geometrical domain** in GiD is based on four levels of geometrical entity: points, lines, surfaces and volumes. Entities of higher level are constructed over entities of lower level; two adjacent entities can therefore share the same level entity. Here are a few examples:

- **Example 1:** One line has two lower level entities (points), each of them at an extreme of the line. If two lines are sharing one extreme, they are really sharing the same point, which is a unique entity.
- **Example 2:** When creating a new line, what is really being created is a line plus two points or a line with existing points created previously.

- **Example 3:** When creating a volume, it is created over a set of existing surfaces, which are joined to each other by common lines. The lines are, in turn, joined to each other by common points.

All domains are considered in 3-dimensional space but if there is no variation in the third coordinate (into the screen) the geometry is assumed to be 2-dimensional for the purposes of analysis and the visualization of results. Thus, to build a geometry with GiD, the user must first define the points, join these together to form lines, create closed surfaces from the lines and define closed volumes for the surfaces. Many other facilities are provided for creating the geometrical domain; these include: copying, moving points, automatic surface creation, etc.

The geometrical domain can be created in a series of layers where each one is a separate part of the geometry. Any geometrical entity (points, lines, surfaces or volumes) can belong to a particular layer. It is then possible to view and manipulate some layers and not others. The main purpose of these layers is to offer a visualization and selection tool as they are not used in the analysis. An example of the use of layers might be a chair where the four legs, seat, backrest and side arms are the different layers.

With GiD you can import a geometry or mesh created with an external CAD program. The formats supported at present are: DXF, IGES, Parasolid, ACIS, VDA, Rhino, Shapefile, STL, VRML, 3DStudio and NASTRAN.

Attributes and conditions are applied to the geometrical entities (points, lines, surfaces and volumes) using data input dialog boxes. These menus are specific to the particular solver that will be employed for the simulation and, therefore, the solver needs to be defined before attributes are defined. The form of these menus can also be configured for the user's own solver module, as is explained below and later in this manual.

Once the geometry and attributes have been defined, a mesh can be generated using the **mesh generation tools** supplied within the system. Structured and unstructured meshes containing triangular and quadrilateral surface meshes or tetrahedral and hexahedral volume meshes may be generated. The automatic mesh generation facility uses a background mesh concept for which the user is required to supply a minimum number of parameters.

Simulations are carried out from within GiD by using the **calculate** menu. Indeed, specific solvers require specific data that must have been prepared previously. A number of solvers may be incorporated together with the correct preprocessing interfaces.

The final stage of **graphic visualization** is flexible in order to allow the user to critically evaluate the results quickly and easily. The menu items are generally determined by the results supplied by the solver module. This not only reduces the amount of information stored but also allows a certain degree of user customization.

One of the major strengths of GiD is that the user can **define and configure his own graphic user interface within GiD**. The first step is to create some configuration files which define new windows, where the final user will enter data, such as materials or conditions. The format that GiD uses to write a file containing the necessary data in order to run the numerical simulation program must also be defined in a similar way. This preprocessor or data input interface will thus be tailored specifically to the user's simulation program, but employing the facilities and functionality of the GiD system. The second step is to include the user's simulation program within GiD so that it may be run utilizing the calculate menu option. The third step consists of writing an interface program, or using the 'gidpost' library, which provides the results information in the format required by the GiD graphic visualizer, thereby configuring the postprocessing menus. This post-analysis interface may be included fully in the GiD system so that it runs automatically once the simulation run has terminated.

Details on this configuration can be found in later chapters.

INVOKING GID

When starting the GiD program from a shell or script it is possible to supply several options in the same command line.

With

```
gid -help
```

the program will list the possible command line options.

Command line syntax:

```
gid [-b[+/-}g][+/-}i][+/-}w] batchfile] [filename] [-h] [-p
problem] [-e cmd] [-n] [-n2] [-c]
```

All options and filename are optional. filename is the name of a problem to be opened (the .gid extension is optional).

Options are:

- **-b batchfile** executes `batch_file` as a script file (see [Batch file](#)).
 - **+/- g** Enable/Disable Graphics (if -g, GiD does not redraw until the batch file has finished).
 - **+/- i** Enable/Disable GraphInput (enable or disable peripherals while the batch file is being executed: mouse, keyboard, etc.).
 - **+/- w** Enable/Disable Windows (GiD displays - or does not display - windows which require interaction with the user).
- **-h** shows GiD's command line arguments.
- **-p problem** loads `problem` as the type of the problem to be used for a new project.
- **-e cmd** can continue until the end of the line. It executes `anything` as if it were a group of commands entered into GiD.
- **-n** runs the program without any window. It is most useful when used with the `batchfile` option.
- **-n2** runs the program without any window but the **Tk** library is loaded. This option is useful if you use **Tcl** commands in a batch file.
- **-c conffile** takes the window configuration from `conffile`. (See [Save window configuration](#) for information about window configuration.)

- **-openglconfig** (*Only for Windows*): this allows you to choose between the accelerated OpenGL, if present, or the generic implementation, if you experience troubles using the accelerated libraries of the graphics card.

Other useful options are:

```
gid -compress [ -123456789ad] file_name_in file_name_out
```

in order to compress (gzip) a file, e.g. to compress '.dat' files or new postprocess formatted data files.

And:

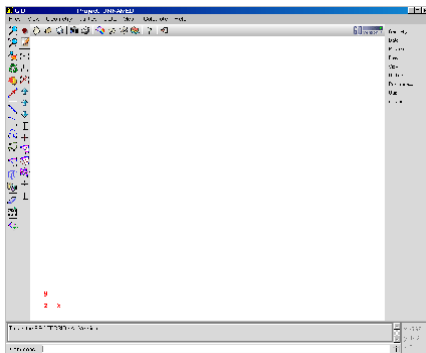
```
gid [ -PostBinaryFormat { 1.0 / 1.1}] -PostResultsToBinary file_in  
file_out
```

in order to transform ASCII results files into compressed binary ones. You can select whether to use the binary format 1.0 or 1.1. The default format (recommended) is 1.1.

USER INTERFACE

The user interface allows you to interact with the program. It is composed of buttons, windows, icons, menus, text entries and the graphical output of certain information. You can configure the interface to display things in a certain way, and may use as many menus and windows as required.

The initial layout of GiD consists of a large graphical area with pull-down menus at the top, 'click on' menus on the right-hand side, a command line at the bottom, a message window above it and an icon bar. The project that is being run is displayed in the window title bar. The pull-down and 'click on' menus are used to access GiD commands quickly. Some of them offer a shortcut for easier access - these are activated by holding the **Ctrl** key and pressing the appropriate letter key(s).



GiD main window

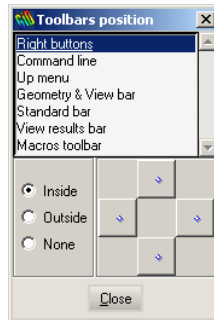
Right-clicking the mouse while the cursor is over the graphical area opens an on-screen menu with some visualization options. To select one of them, right- or left-click on the option; to quit, left-click anywhere outside the menu.

The first option in this menu is called `Contextual`. It will give different options depending on the function currently being used.

The pair of icon bars contain some facilities that also appear in the graphical area of the window or in the menu bar. When left-clicking on the icon, the corresponding command is performed or an icon menu with several options will be shown. When right-clicking (or using the center button if there is one), a menu with the options `help` and `configure toolbars` will appear, allowing you to get a description of the icon or to configure the position of the toolbars. The description also appears when the cursor remains over the icon for a couple of seconds.

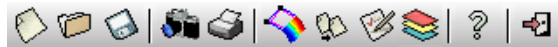
To configure the position and view of the toolbars, the **Toolbars position** window can be called from *Utilities*→*Tools*→*Toolbars*, or by right-clicking over a toolbar.

Using the **Toolbars position** window it is possible to enable the **Right buttons** menu. Only advanced users should use these buttons.



Toolbars position window

The **Standard bar** has common options for both pre- and postprocessing components, including: open, take a snapshot, print, preferences, help, exit and others.



Standard toolbar

The different icons represent, from left to right:

- New
- Open
- Save
- Take a snapshot
- Print
- Toggle between preprocess and postprocess
- Copy
- Preferences
- Layers
- Help search
- Help

- Exit

The **Geometry & view bar** has some of the view options common to pre- and postprocessing, such as zooming, panning, rotating, etc. But certain icons are specific to preprocessing and others to postprocessing.



Icon toolbox menu in Preprocess

The different icons represent, from left to right:

- Zoom in
- Zoom out
- Zoom frame
- Redraw
- Rotate trackball
- Pan dynamic
- Create line
- Create arc
- Create NURBS line
- Create polyline
- Create NURBS surface
- Create volume
- Create object (rectangle, polygon, circle, sphere, cylinder, cone, prism, torus)
- Delete
- List entities
- Toggle between geometry view and mesh view

Note: The position of the icons depends on how the window is positioned.

If you left-click on **Delete**, GiD opens another window with the different entities to be deleted: **Point**, **Line**, **Surface**, **Volume** or **All** types.

If you left-click on **List entities**, GiD opens another window with the different entities able to be listed: **Points**, **Lines**, **Surfaces** or **Volumes**.



Icon toolbox menu in Postprocess

On the postprocess toolbar, the first six commands are the same as on the preprocess toolbar, i.e. from `Zoom in` to `Pan`.

The remaining icons represent, from left to right:

- Change Light Vector: allows you to change the light direction (see [Render](#)).
- Display style: when you click here, a menu appears with each icon corresponding to a display option: Boundaries, Hidden Boundaries, All Lines, Hidden Lines, Body, Body Boundaries and Body Lines.
- Culling: allows you to switch on or switch off the front faces and/or the back faces.
- Mesh selection
- Set selection
- Cut selection
- Cut meshes/sets
- Set maximum value
- Set minimum value
- Reset maximum and minimum values
- List nodes and elements information

Notes:

If windows are used to enter the data, it is generally necessary to **accept** this data before closing the window. If this is not done, the data will not be changed.

Usually, commands and operations are invoked by using the menus or mouse, but all the information can be typed into the command line.

When an option is selected and a secondary window is opened, it generally appears over the main window and cannot be hidden by it. This behavior can be changed by deselecting the `Always on top` flag in the Window system menu (right-click on the window title bar to do this).

Mouse operations

As well as selecting the functions to be used, the left mouse button is used to select entities, either individually or picking several within a given area (see [Entity selection](#)), and to enter points in the plane $z=0$ (see [Point definition](#)).

The middle mouse button is equivalent to `escape` (see [Escape](#)).

The right mouse button opens an on-screen menu with some visualization options. To select one of them, use the left or right mouse button; to quit, left-click anywhere outside the menu.

The first option in this menu is called `Contextual`. You can select from different options relevant to the function currently being used.

When the mouse is moved to different windows, depending on the situations, different cursor shapes and colors will appear on the screen.

In some windows a help option will appear when you click the middle or right mouse buttons over an icon.

Command line

All commands may be entered via the command line (found at the bottom of the GiD window) by typing the full name or only part of it (long enough to avoid confusion with other commands); commands are not case-sensitive. Any function from the Right buttons menu can be used by typing all or part of its name in the command line. Special commands are also available for viewing (zoom, rotation and so on) and these can be typed or used at any time when working from within another function. A list of these special commands is given in `View` (see [VIEW](#)).

Commands entered by typing are word oriented. This means that the same operation is achieved if one writes the entire command and then presses `enter` or if one writes a part of it, presses `enter` and then writes the rest.

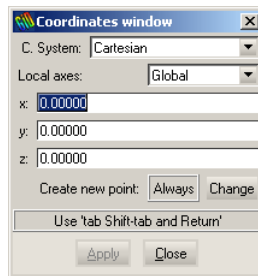
All these typed commands can be retrieved using of the up arrow (to recover past commands) and down arrow (to return to more recent commands).

USER BASICS

The following features are essential to the effective use of the GiD system. They are, therefore, described apart from the preprocessing facilities section.

Point definition

Many functions inside GiD need points to be defined by the user. Points are the lowest level of geometrical entity and therefore the most commonly used. Consequently, it is important that you have a thorough understanding of how to do this. Sometimes an existing point is required and sometimes a new point must be defined.



Window for entering coordinates

All the options explained in this section are available through the window shown above (see [Coordinates window](#)). This window is accessed via the pull-down menu `Utilities`→`Tools`. Here you can choose not only the kind of reference system - cartesian, cylindrical or spherical - but also whether to use a global or local coordinate system and whether the origin of coordinates is fixed or relative (where new coordinates are relative to the last origin point entered).

In general you can enter points in the following ways:

1. Picking in the graphical window.
2. Entering points by coordinates.
3. Selecting an existing point.
4. Using the `Base` button.

Picking in the graphical window

Points are picked in the graphical window in the plane $z=0$ according to the coordinates viewed in the window. Depending on the activated preferences (see [Preferences](#)), if you select a region located in the vicinity of an existing point, GiD asks whether it should create a new point or use the existing one.

Entering points by coordinates

GiD offers a window for entering points in order to create geometries easily, defining fixed or relative coordinates as well as different reference systems - cartesian, cylindrical or spherical.

The coordinates of a point can be entered either in the enter points window or in the command line by following one of two possible formats:

1. The format: x, y, z
2. The format: $x \ y \ z$

The Z-coordinate can be omitted in both cases.

The following are valid examples of point definitions:

$5.2, 1.0$	$5.2, 1$
$8 \ 9 \ 2$	$8 \ 9, 2$

All of a point's coordinates can be entered as local or global and through different reference systems in addition to the cartesian one.

1. Local/global coordinates
2. Cylindrical coordinates
3. Spherical coordinates

Local/global coordinates

Local coordinates are always considered relative to the last point that was used, created or selected. The `Utilities`→`Id` command allows you to make a reference to one point (see [Id](#)). Then, to define points using local coordinates referring to the same point, use `Options` and `Fixed Relative` when entering each point. The last point selected or created before using this will be the origin of the local coordinate system. It is also possible to enter this central point by its coordinates.

The following are valid examples of defining points using local coordinates:

Example (1):

```
1,0,0
@2,1,0 (actual coordinates 3,1,0)
@0,3,0 (actual coordinates 3,4,0)
2,2,2
@1,0,3 (actual coordinates 3,2,5)
```

Example (2):

```
1,0,0
Fixed Relative (when creating the point)
@2,1,0 (actual coordinates 3,1,0)
@0,3,0 (actual coordinates 1,3,0)
2,2,2
@1,0,3 (actual coordinates 2,0,3)
```

Example (3):

```
'local_axes_name'2.3,-4.5,0.0
```

The last example shows how to enter a point from a local coordinate system called 'local_axes_name' (any name inside the quotation marks will work), previously defined via the option `define local axes` (see [Local axes](#)).

All the examples have been presented using a cartesian notation. However, cylindrical or spherical coordinates can also be used.

Cylindrical coordinates

Cylindrical coordinates can be entered as: `r<angle,z`

The Z-coordinate may be omitted and angles are defined in degrees. Cylindrical coordinates can be applied to global and local coordinate systems.

The following are valid examples of the same point definitions:

Example (1):

```
1,0,0
1.931852<15
```

Example (2):

```
1,0,0
@1.0<30
```

Spherical coordinates

Spherical coordinates can be entered as `r<anglexy<anglez`

`Anglez` may be omitted and angles are defined in degrees. Spherical coordinates can be applied to global and local coordinate systems.

The following are valid examples of the same point definitions:

Example (1):

```
1,0,0
1.73205<18.43495<24.09484
```

Example (2):

```
1,0,0
@1.0<45<45
```

Base

Mouse menu



If the `Base` button is selected (it is set by default to `No Base`), a point can be retrieved from any of the other modes. Then, the coordinates of this point, instead of being used immediately, are written in the command line and can be edited before they are confirmed.

It is possible to change the way that GiD works with points by default via preferences (see [Preferences](#)).

Selecting an existing point

Menu



When using a function that asks for a point, e.g. line creation, GiD will expect you either to enter a new point (the cursor is a cross) or select an existing one (the cursor is a box). To change from the first mode to the second, click the `Join` button in the **Right buttons** menu or the **Contextual** mouse menu, or use the shortcut (Ctrl-a); the option will then change to `No Join`.

Simply select an existing point to pick it. (Ctrl-a) switches from `Join` to `No Join` and vice versa.

The special options `FJoin` and `FNoJoin` force GiD to change either to `Join` mode or `No Join` mode independently of the previous mode.

Point in line

Mouse menu



With this option selected, when creating a new point or line, etc., you can only select points that lie on existing lines. To switch it off, simply select `No Point in line`.

Point in surface

Mouse menu



With this option selected, when creating a new point or line, etc., you can only select points that lie on existing surfaces. To switch it off, simply select `No Point in surface`.

Tangent in line

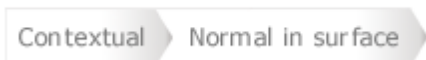
Mouse menu



Using this option, you can pick over a line in the graphical window. A vector will be returned that is the tangent to the line at the point you have picked.

Normal in surface

Mouse menu



Using this option, you can pick over a surface in the graphical window. A vector will be returned that is the normal to the surface at the point you have picked.

Arc center

Mouse menu



Using this option, you can left-click on an arc in the graphical window and a point will be created at its center.

Grid



It is possible to use an auxiliary grid of lines to define 2D points easily. The 'snap' function can be activated to force points to grid intersections.

From the preferences window (see [Preferences](#)) it is possible to set the separation between lines and to show the origin, extents, etc. of the coordinates.

There is a small button in the bottom right-hand corner that activates or deactivates the `grid` and 'snap' functions.

Entity selection

Many commands need to be supplied with entities before they can be applied and the method of selection is always the same. Before selecting entities, you are prompted to decide whether to select points, lines, surfaces or volumes (in some cases this decision is obvious or it is made within the context of the option).

Within one of the generic groups (points, lines, surfaces, volumes, nodes or elements) it does not matter what type of entity is selected (for example, an arc or a spline, both line entities are selected in the same way). After this, if one entity of the desired group is selected, it is colored red to indicate it has been selected and you are prompted to enter more entities. If you select away from any entity, a dynamic box is opened that can be defined by picking again in another place. All entities that are either totally or partly within this box are selected. Once again, you are then prompted to enter more entities. The normal selection mode is `SwapSelection`: If one entity is selected a second time, it becomes deselected and its color reverts to normal. In addition there are the options `AddToSelection` and `RemoveFromSel`, the former always adding to the selection, the latter always removing entities from the selection.

Note: Instead of picking a start point and an end point for the selection box, it is possible to press and hold left mouse and move the cursor.

The `Clear selection` option, which is found in the **Contextual** mouse menu, deselects all previously selected entities.

It is also possible to select entities by entering their label in the command line. For instance, to select the entity with number 2, input this number, 2, in the command line. To select the entities 3 to 7, input 3:7 in the command line. Entering 3: will select all entities from number 3 to the end and entering :3 will select all options from the beginning to number 3.

If a layer named 'a' exists, it is possible to select all entities belonging to that layer with command `layer:a`. Using the command `layer:` selects all entities not belonging to any layer.

Another way of selecting points or nodes is to write:

```
plane:a,b,c,d,r
```

where a,b,c,d and r are real numbers that define a plane and a tolerance in the following way: $ax+by+cz+d < r$. Points close to that plane are chosen.

In some commands, another item is added to the selection group. This item, called `AllTypes`, means that entities of all levels (points...volumes) will be selected at the same time. In this case, only selection via a dynamic box is possible in the graphical window and all entities (points, lines, surfaces and volumes) in the box are selected.

To finish the entity selection, use `escape` (see [Escape](#)).

If the `Fast Selection` option is used, entities are not colored red when selected and choosing an entity twice does not deselect it. This option is available via the **Right buttons** menu (see [Tools](#)), in `Utilities`→`Variables`.

Caution: Only use `Fast Selection` when you need to select a large number of entities, for example in a large mesh, as there is a risk of repeating entities.

Entities belonging to frozen layers (see [Layers](#)) are not taken into account in the selection. Entities belonging to OFF layers cannot be selected directly in the graphical window, but can be selected by giving a number or range of numbers.

It is possible to add filters to the selection so that, after selecting some entities, only the ones satisfying the filter criteria will remain selected. To enter one filter, you must enter the word `filter:` in the command line followed by one option. The available options are:

- HigherEntity
- MinLength
- MaxLength
- EntityType
- BadMinAngle
- BadMaxAngle

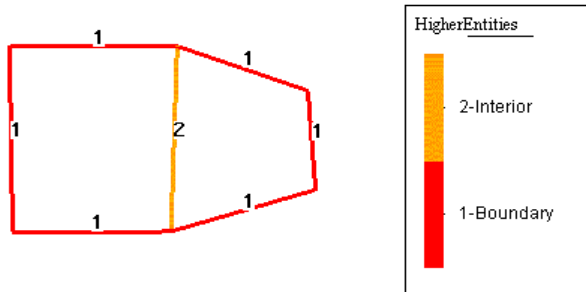
Note: To apply selection filters you can also use the **Selection window** (see [Selection window](#)).

Note: `MinLength` and `MaxLength` can be used either in geometry lines or in elements of the mesh.

For example, the following command:

```
filter:HigherEntity=1
```

means that only the entities that have higher entity equal to one will be selected.



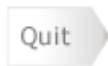
Note: A typical use of `filter` is to select only boundary lines (`HigherEntity=1`).

Quit

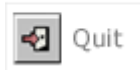
Menu



Mouse menu



Toolbar



The `quit` command is used to finish the working session. If there have been changes since the last time a session was saved, GiD asks if you wish to save them.

Escape

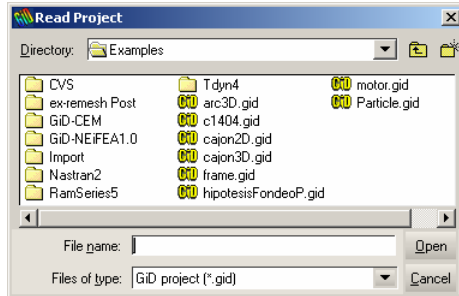
The escape command is used for moving up a level within the **Right buttons** menus, for finishing most commands, or for finishing selections and other utilities. This command can be applied by:

1. pressing the middle mouse button;
2. pressing the `ESC` key;
3. pressing the `escape` button in the **Right buttons** menu;
4. writing the reserved word `escape` in the command line. This is useful in scripts (see [Batch file](#)).

All the above options give the same result.

Caution: `Escape` is a reserved word. It cannot be used in any other context.

FILES



Browser to read and write files and projects.

GiD includes the usual ways of saving and reading saved information (*Save*, *Read*) as well as other operations, such as importing external files, saving in other formats and so on.

New

Menu

Files New

Toolbar



Selecting *New* opens a new project with no title assigned to it.

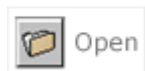
If a project is currently open and changes have been made since it was last saved, GiD will give the option to save before opening the new project.

Open

Menu



Toolbar



With this command, a project previously saved with `Save` (see [Save](#)) or with `Save ASCII project` (see [ASCII project](#)) can be opened.

Generally, there is no difference between using a project name with the `.gid` extension or using one without it.

Save

Menu



Menu



Using the `Save` command saves all the information relating to a project - geometry, conditions, materials, mesh, etc. - to the disk.

When a project is saved, GiD creates a directory with the project name and the extension `.gid`. Files containing all the information are written to this directory. Some of these files are binary and others are ASCII. You can then work with this project directory as if it were a file.

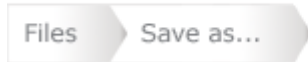
You do not need to write the `.gid` extension because it will automatically be added to the directory name.

Caution: Be careful if changing some files manually in the `project_name.gid` directory. If done in this way, some information may be corrupted.

Advice: It is advisable to save the project at regular intervals so as not to lose any important information. It is possible to back up files automatically by selecting this option in the **Preferences** menu (see [Preferences](#)).

Save as

Menu



With this command, GiD allows you to save the current project with another name.

When it is selected, an auxiliary window appears with all the existing projects and directories to facilitate the introduction of the project's new name and directory.

Import

GiD lets you import geometrical models or meshes in the following formats.

IGES

Menu



With this option it is possible to import a file in IGES format (version 5.3); GiD is able to read most of the entities, which are:

Entity number and type (Notes)	
100	Circular arc
102	Composite curve
104	Conic arc (ellipse, hyperbola and parabola)
106	Copious data (forms 1, 2, 12 and 63)
108	Plane (form1 bounded)
110	Line
112	Parametric spline curve
114	Parametric spline surface
116	Point
118	Ruled surface
120	Surface of revolution
122	Tabulated cylinder
124	Transformation matrix (form 0)

126	Rational B-spline curve
128	Rational B-spline surface
140	Offset surface entity
141	Bounded entity
142	Curve on a parametric surface
143	Bounded surface
144	Trimmed surface
186	Manifold solid B-rep object
308	Subfigure definition
402	Associativity instance
408	Singular subfigure instance
502	Vertex
504	Edge
508	Loop
510	Face
514	Shell

The variable `ImportTolerance` (see [Preferences](#)) controls the creation of new points when an IGES file is read. Points are therefore defined as unique if they lie further away than this tolerance distance from another already defined point. Curves are considered identical if they have the same points at their extremes and the "mean proportional distance" between them is smaller than the tolerance. Surfaces can also be collapsed.

Entities that are read in and transformed are not necessarily identical to the original entity. For example, surfaces may be transformed into planes, Coons or NURBS surfaces defining their contours and shape.

DXF

Menu



With this option it is possible to read a file in DXF format (AutoCAD 2002 version).

Almost all of the geometry in a DXF file can be read, with the exception of solid modeled entities.

A very important parameter to consider is how the points must be joined. This means that points that are close to each other must be converted to a single point. This is done by defining the

variable `ImportTolerance` (see [Preferences](#)). Points closer together than `ImportTolerance` will be considered as a single point. Straight lines that share both points are also converted to a single line.

You can use the `Collapse` function (see [Collapse](#)) to join more entities.

Parasolid

Menu



With this option it is possible to read a file in the Parasolid format (version 14000 - ASCII or binary).

The most usual Parasolid file extension is `.x_t` for ASCII and `.x_b` for binary format.

The variable `ImportTolerance` (see [Preferences](#)) controls the creation of new points when a Parasolid file is read. Points are therefore defined as unique if they lie further away than this tolerance distance from another already defined point. Curves are considered identical if they have the same points at their extremes and the "mean proportional distance" between them is smaller than the tolerance. Surfaces can also be collapsed.

ACIS

Menu



With this option it is possible to read a file in ACIS format (version 7.0). GiD reads the ASCII version with the SAT Save File Format. ACIS files (in ASCII) have the `.sat` extension.

VDA

Menu



With this option it is possible to read a file in VDA 2.0 format.

A very important parameter to consider is how the points must be joined. This means that points that are close to each other must be converted to a single point. This is done by defining the

variable `ImportTolerance` (see [Preferences](#)). Points closer together than `ImportTolerance` will be considered as a single point. Straight lines that share both points are also converted to a single line.

The `Collapse` function (see [Collapse](#)) can be used to join more entities.

Rhino

Menu



With this option it is possible to read Rhino 3.0 CAD files.

Shapefile

Menu



With this option it is possible to read a GIS file written in ESRI Shapefile format (version 1000). Shapefiles have the `.shp` extension.

NASTRAN mesh

Menu



With this option it is possible to read a file in NASTRAN format (version 68), with GiD accepting most of its entities, which are:

Entity name (Notes)

CBAR CBEAM CROD CCABLE CBUSH CELAS1 CELAS2 CELAS3 RBAR (translated as 2 node bars)

CQUAD4 CQUADR

CHEXA

CTETRA

CPENTA

CTRIA3 CTRIAR

CONM1 CONM2 (translated as 1 node element)

```
CORD1C CORD1R CORD1S  
CORD2C CORD2R CORD2S  
GRID
```

There are two options that can be used when reading a mesh if GiD already contains a mesh:

- a) Erasing the old mesh (`Erase`);
- b) Adding the new mesh to the old one without sharing the nodes; the nodes will be duplicated although they may occupy the same position in the space (`AddNotShare`).

The properties and materials of elements are currently ignored, because of the difficulties in associating the NASTRAN file properties with the requirements of the analysis programs. Therefore, you have to assign the materials *a posteriori* accordingly. However, in order to make this easier, the elements will be partitioned in different layers, each with the name `PIn`, where `n` is the property identity number associated with the elements as defined in the NASTRAN file. Note that CELAS2 elements do not have associated property identities so these will be created by default when the file is read.

STL mesh

Menu



With this option it is possible to read a mesh in STL format. The STL binary format is also supported.

The variable `ImportTolerance` (see [Preferences](#)) controls the creation of new points when the file is read.

VRML mesh

Menu



With this option it is possible to read a mesh in VRML 2.0 format. The compressed gzip format is also supported.

3DStudio mesh

Menu



With this option it is possible to read a mesh in .3ds 3DStudio format.

GiD mesh

Menu



With this option it is possible to read a GiD ASCII mesh (saved with `Export GiD Mesh`) in order to visualize it within GiD.

It is also possible to read a new mesh and add it to the existing one. In this case, you are prompted to keep the former one or join it to the new mesh.

The format of the file describing the mesh must have the following structure:

```
mesh dimension = 3 elemtype tetrahedra nnode = 4
coordinates
1 0 0 0
2 3 0 0
3 6 0 0
4 3 3 0
5 3 1.5 4
6 3 1.5 -4
7 1.5 0 2
end coordinates
elements
1 1 2 4 5 1
2 2 3 4 5 1
3 1 4 2 6 1
4 2 4 3 6 1
5 1 2 5 7 1
end elements
```

The code `nnode` means the number of nodes per element and `dimension` can be either:

- 2: 2 dimensions. Nodes have just two coordinates.
- 3: 3 dimensions. Nodes have three coordinates.

Where `elemtype` must be:

- Linear
- Triangle
- Quadrilateral
- Tetrahedra
- Hexahedra

Every element may have an optional number after the definition of the connectivity. This number usually defines the material type and it is useful to divide the mesh into layers to visualize it better. GiD offers the possibility of dividing the problem into different layers according to the different materials through the option `Material` (see [Layers](#)).

Note: The = sign is optional, but if it is present it is necessary to leave a space.

If it is necessary to enter different types of elements, every type must belong to a different mesh. More than one mesh can be entered by writing one after the other, all of them in the same file. The only difference is that all meshes except the first one have nothing between `coordinates` and `end coordinates`. They share the first mesh's points. Example: to enter tetrahedron elements and triangle elements,

```
mesh dimension = 3 elemtype tetrahedra nnode = 4
coordinates
1 0 0 0
2 3 0 0
3 6 0 0
4 3 3 0
5 3 1.5 4
6 3 1.5 -4
7 1.5 0 2
end coordinates
elements
1 1 2 4 5 1
```



```
2 2 3 4 5 1
3 1 4 2 6 1
4 2 4 3 6 1
5 1 2 5 7 1
end elements
mesh dimension = 3 elemtype triangle nnode = 3
coordinates
end coordinates
elements
1 1 2 4 1
2 2 3 4 1
3 1 4 2 1
4 2 4 3 1
5 1 2 5 1
end elements
```

Surface mesh

Menu



With this option a mesh can be read from a file in GiD or STL format (see [GiD mesh](#)). Elements of this mesh must be triangles or quadrilaterals. This mesh is converted by GiD into a set of surfaces, points and lines. The geometric definition of surfaces is the mesh itself, but GiD treats them as truly geometric entities. For example, these surfaces can be used as the boundary of a volume, and a new mesh can be generated over them.

You are asked for the value of an angle. An angle between elements bigger than this value is considered to be an edge, and lines are inserted over them. As a consequence, a set of boundary and interior lines are created and attached to the surfaces to mark their edges.

Batch file

Menu



Sometimes, you may wish to organise a number of commands into a group outside GiD, ready to be implemented in one go. To do so, commands can be written in a file and GiD will read this

file and execute the commands. These commands are the same ones as are used in GiD when entered in the command line or using the commands in the **Right buttons** menu.

Example: Many points have been digitalized and their coordinates saved in a file. These points are to be joined with straight lines to create the outline of the geometry. To do so, the file would look similar to this:

```
geometry create line
3.7 4.5 8
2 5 9
4,5,6
...
1 7 0.0
escape
```

A batch file can also be loaded into GiD by giving its name with the `-b` option when opening GiD (see [INVOKING GiD](#)). Another way to read batch files to create dynamic presentations is with the `Read batch` window (see [Read batch window](#)). One GiD session can be registered in a batch file. This can be useful for checking the batch commands or to repeat one session (see [Preferences](#)).

BATCH FILE COMMANDS

There are some special commands to be added to a batch file that are treated differently from regular GiD commands. Their format is one or several words after the control string `*****` (five asterisks) and everything in one line.

- **Write a log file**

```
*****OUTPUTFILENAME filename
```

`filename` is substituted with a real file name where all the session warnings (those which appear in the GiD messages warning line) are written. This can be useful when running GiD in batch mode with the option `-n` (see [INVOKING GiD](#)) and GiD output is desired.

- **Execute a Tcl command in a batch file**

```
*****TCL tcl_command
```

Note: If this command is used in a batch file and GiD is invoked with the option `-n`, it will not work. So that Tcl commands are executed when GiD is run without a window, you should use the `-n2` option (see [INVOKING GiD](#)).

- **Insert comments in the code of a batch file**

```
geometry create line 1,2
*****COMMENTS -this is a comment-
2,3 escape
```

- **Print messages in the lower GiD messages line**

```
geometry create line 1,2
*****PRINT -This is a message that will appear in the messages
line-
2,3 escape
```

- **Print messages in a window**

```
geometry create line 1,2
*****PRINT1 -This is a message that will appear in a new
window-
2,3 escape
```

Insert GiD geometry

Menu



This command lets you insert a previously created GiD model inside another one. Entities from the old and the new model are not collapsed.

You can perform one `Collapse` operation (see [Collapse](#)) to join the old and new models.

Export

GiD lets you export geometrical models or meshes in the following formats.

IGES

Menu



GiD can export the geometry in IGES format (version 5.3).

If the preference 'IGES:B-Rep output style' is set (see [Preferences](#)), then the output file is written in Boundary representation solid model style; otherwise the surfaces are written as separated trimmed surfaces, without topological information, and the volumes are ignored.

The IGES geometric entities generated are:

```

116    Point
110    Line
102    Composite curve
126    Rational B-spline curve
128    Rational B-spline surface
142    Curve on a parametric surface
144    Trimmed surface

```

and the topological entities are (B Rep style):

```

186    Manifold solid B-rep object
502    Vertex
504    Edge
508    Loop
510    Face
514    Shell

```

DXF

Menu



GiD can export the geometry in DXF format (AutoCAD 2002 version). Points and curves are correctly exported, but a surface must be converted into a mesh of triangles, because DXF does not support Trimmed NURBS Surfaces.

ACIS

Menu



GiD can export the geometry in ACIS ASCII format, version 5.0 (files with `.sat` extension).

GiD mesh

Menu



With this option a file is written with all of the project's mesh or meshes inside. This file can be read with `Import GiD Mesh` (see [GiD mesh](#)).

Text data report

Menu



With this option a file is written containing all the information within the project. It is created in a way that is easily understood when read with an editor. This is useful for checking the information.

Note: This ASCII format is only used to check information. It cannot be read again by GiD. To write ASCII files that can be read again use the option `SaveAsciiProj` (see [ASCII project](#)).

ASCII project

Menu



This option saves a project in the same way as regular `Save` (see [Save](#)) but files are written in ASCII. It may be useful for copying projects between incompatible machines. GiD also allows this information to be written in a file (see [Text data report](#)).

Projects saved in this way may be read with the same `open` command (see [Open](#)).

ON layers

Menu



With this option, only the geometrical entities with their layers set to ON will be saved in a new project (see [Layers](#)).

Note: Lower entities necessary to define the saved entities will be also saved in the new project (e.g. the two extreme points of a line are also saved if the line is saved).

Calculation file

Menu



If GiD runs the solver module automatically, this command is not necessary. However, it is useful if the solver program has to be run outside GiD, or to check the data input prior to any calculations.

This command writes the data file needed by the solver module.

The format of this file must be defined in a **Template File** (see [Template File](#)). GiD uses the template file of the current **Problem Type** to write the data file; so, to run this command, a problem type must be selected.

When testing a new problem type definition, GiD produces messages about errors within the configuration. When the error is corrected, the command can be used again without leaving the example and without having to reassign any conditions or meshing.

Using a .bas template

Menu



This command does the same thing as `Export→Calculation file` (see [Calculation file](#)), but it uses a `.bas` file provided by the user, instead of using the template file of the current problem type. This means it is not necessary to select a problem type in order to run this command.

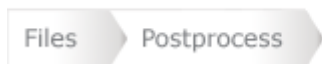
When choosing 'Others...' from the submenu, GiD asks for a `.bas` file (see [Template File](#)) and, using that file, writes the data file needed by the solver module. There are some `.bas` codes available in the submenu which write output files in some formats (DXF, NASTRAN, STL, VRML). These example `.bas` files are located in the `Templates` directory of the main GiD directory. It is possible to add other `.bas` files to that directory so they appear in the submenu.

Preprocess/Postprocess

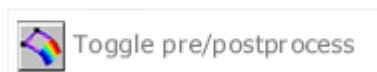
Menu



Menu



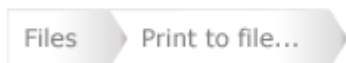
Toolbar



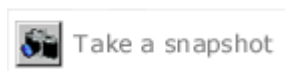
This command allows you to move between GiD Preprocess and Postprocess.

Print to file

Menu



Toolbar



This option asks you for a file name and saves an image in the required format. The properties of the image (resolution, size, etc.) can be assigned in `Page/image setup` (see [Page/Image setup](#)).

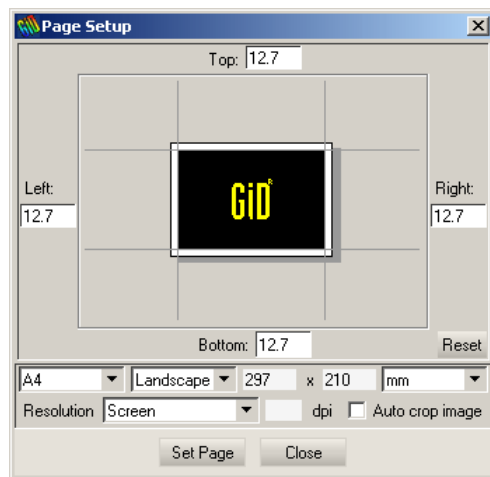
The accepted formats are as follows:

- **Postscript screen:** Postscript. Useful for sending to a postscript printer. It is a snapshot of the screen.
- **Postscript vectorial:** Postscript. Useful for sending to a postscript printer. It gives a higher quality result, but can only be used for small models. Otherwise, very large files are created and it takes a long time to print them.
- **EPS screen:** Encapsulated postscript. Useful for inserting into documents.
- **EPS vectorial:** Encapsulated postscript. Useful for inserting into documents. It gives a higher quality result than EPS screen, but the resulting file is much bigger.

- **BMP**: Windows Bitmap image file.
- **GIF**: Graphics Interchange Format image file.
- **JPEG**: Joint Photographic Experts Group image file.
- **PNG**: Portable Network Graphics image file.
- **TGA**: Truevision TarGA image file.
- **TIFF**: Tagged Image File Format.
- **VRML**: Writes a VRML model file with the current visualization.

Page/Image setup

Menu



Page setup window

This is the window where some print properties (page size, borders, etc.) and image properties (image resolution and Auto crop image option) can be set up. These settings are applied when sending an image to a printer (see [Print](#)), or to a file (see [Print to file](#)).

Print

Menu



Sends the current image to the selected printer.

Recent files

Menu



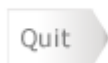
You can quickly gain access to files opened recently with GiD.

Quit

Menu



Mouse menu



Toolbar



The `Quit` command is used to finish the working session. If there have been changes since the session was last saved, GiD asks you to save them.

VIEW

Visualization commands change the way information is displayed in the graphical window. They have no effect on the definition of the geometry or any other data.

Generally, they can be used within any other command without leaving it. When the visualization process finishes, the first command continues.

They can all be accessed from the **View** pull-down menu, and most of them also by clicking the right mouse button.

Zoom

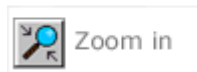
Menu



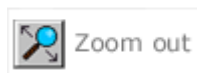
Mouse menu



Toolbar



Toolbar



Zoom is used to change how large or small objects appear in the window.

- **Zoom in:** Pick inside the graphical window. A dynamic box is opened. Pick again and the visualization changes to display only the part within the defined box.
- **Zoom out:** Pick inside the graphical window. A dynamic box is opened. Pick again and the visualization changes so that everything in the graphical window is reduced to the size of the box.
- **Zoom dynamic:** Left-click on the point that is to be the centre of the zoomed image. Moving the mouse to the right enlarges the image and moving the mouse to the left reduces its size. Left-click once more to finish.
- **Zoom previous:** GiD goes to the previous saved zoom.

- **Zoom next:** If Zoom previous has been selected, this option goes back to next view in the list.
- **Zoom frame:** Choose a visualization size so as to display everything inside the window.
- **Zoom points:** Enter two points (see [Point definition](#)), and a visualization size is chosen so as to display these two points inside the window. This option only appears in the **Right buttons** menu (see [USER INTERFACE](#)).

Note: Instead of picking twice to begin and end the rectangle, hold down the left mouse button and move the cursor.

Rotate

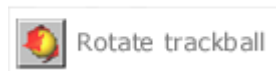
Menu



Mouse menu



Toolbar



There are various ways to rotate the image in order to view it from different angles. This does not affect the geometry.

Note: Instead of picking twice to begin and end the rotation, hold down the left mouse button and move the cursor.

Rotate trackball

With this option you can rotate the image as if using a trackball device. This means that when you left-click on a point and move the mouse, the geometric point tries to follow the mouse pointer. This can be imagined as a ball over the graphical window which is moved with the mouse.

The left mouse button can be pressed several times to engage and disengage the movement. To cancel this function, use `escape` (see [Escape](#)).

Rotate screen axes

This option allows a dynamic rotation about the screen axes. Screen axes are defined as:

- `X-axis`: The horizontal axis.
- `Y-axis`: The vertical axis.
- `Z-axis`: The axis at a right angle to the screen.

When entering this command, `Z-axis` is set by default and moving the mouse to the left or to the right will rotate the geometry around this axis. Clicking the left mouse button changes the axis. To cancel this function, use `escape` (see [Escape](#)).

The axis about which the image is rotated can be changed by entering the letters `x`, `y` or `z` in the command line.

To move the geometry by a fixed angle, enter the number of degrees, positive or negative, in the command line.

Rotate object axes

This option allows a dynamic rotation of the object about its own axes. These are displayed in the bottom left-hand corner of the screen.

When entering this command, `Z-axis` is set by default and moving the mouse to the left or to the right will rotate the geometry around this axis. Clicking the left mouse button changes the axes. To cancel this function, use `escape` (see [Escape](#)).

The axis about which the image is rotated can be changed by entering the letters `x`, `y` or `z` in the command line.

To move the geometry by a fixed angle, enter the number of degrees, positive or negative, in the command line.

Rotate center

The default center of rotation is defined as a point approximately in the center of the geometry.

If you wish to change this center point, use this command to enter a point (see [Point definition](#)). This new centre of rotation will be maintained until the next zoom frame (see [Zoom](#)).

In the **Contextual** mouse menu (the menu which appears when you right-click over the graphical window) the option 'Automatic rotation center'/'No automatic rotation center' is listed.

If this option is active, for each 'Zoom In'/'Zoom Out'/'Pan' the point of the geometry or mesh nearest to the center of the screen will be selected as the center of rotation for subsequent rotations. This variable is also present in the **Right buttons** menu under Utilities→Variables.

If a new `Rotation center` is selected, this option is deactivated.

Rotate angle

This option only appears in the **Right buttons** menu (see [USER INTERFACE](#)).

The new position of the geometry after the rotation can be defined as the direction orthogonal to the screen via a pair of angles:

1. The **angle in the plane XY** starting from the X-axis.
2. The **elevation angle** from the XY plane.

As an example, the initial view (at a right angle to the Z-axis and with the X-axis horizontal) can be obtained with:

```
rotate angle 270 90
```

Rotate points

This option only appears in the **Right buttons** menu (see [USER INTERFACE](#)).

The new position of the geometry after the rotation can be defined as the direction orthogonal to the screen via a pair of points:

1. The **target point**, the point you are looking at.
2. The **viewpoint**, the point you are looking from.

Plane XY (Original)

This option changes the view to the original one, i.e. with the screen at a right angle to the Z-axis and with the X-axis lying horizontally and pointing to the right.

Plane XZ

This option changes the view so that the screen is at a right angle to the Y-axis with the X-axis lying horizontally and pointing to the right.

Plane YZ

This option changes the view so that the screen is at a right angle to the X-axis with the Y-axis lying horizontally and pointing to the right.

Pan

Menu



Mouse menu



Toolbar



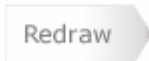
- **Two points:** This command allows the geometry to be moved within the graphical window. To do this, pick two points in the graphical window.
- **Dynamic:** In this case the object can be moved around following the movements of the mouse.

Redraw

Menu



Mouse menu



Toolbar



This command redraws the geometrical model or the mesh (depending on the visualization mode, see [Mode](#)) in the graphical window. For those machines that include overlays, none of the layers that stay underneath is affected, so the redraw is carried out more quickly and the drawings underneath remain untouched.

Render

Menu

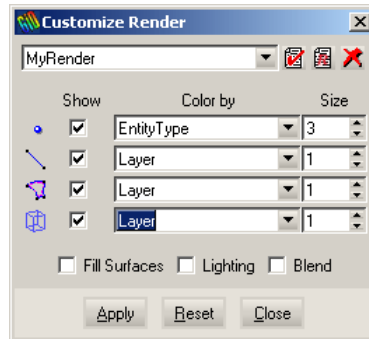


Mouse menu



Using this option changes the way the model is viewed. There are three principal options:

- **Normal:** This is the usual way of viewing the image. You can see both the geometry and mesh including all definition lines.
- **Flat lighting:** Solid model with flat illumination and lines.
- **Smooth lighting:** Solid model with smooth illumination (better quality).
- **Change light direction:** With this option you can change the **Vector** of the light direction interactively; this can also be done by entering the **Vector** components in the **command line**.
- **Customize:** You can define your own rendering with its own properties.

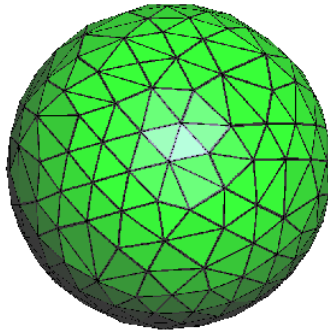


Customize render window

You can choose how to visualize each entity, and save these properties in a new `Render mode`, which will appear in the `View→Render` menu or in the mouse menu, next to standard GiD render modes.

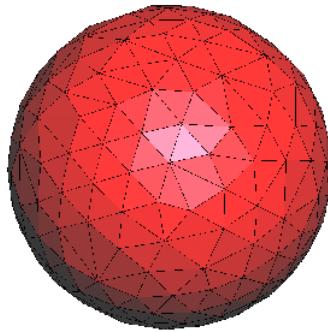
The rendering of a volume mesh can be distinguished from the rendering of its contour surfaces by the different ways of representing elements (boundary elements in the case of volume, and elements in the case of surfaces):

- `Volume mesh render`: Boundary faces of volume mesh elements are shown in the color of the layer that the volume belongs to, with thick black borders.



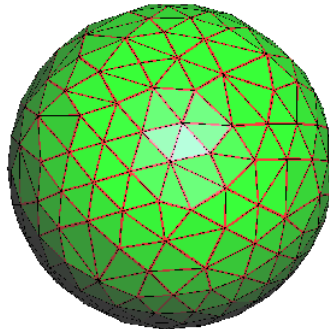
Volume mesh render

- `Surface mesh render`: Surface elements are represented as usual, in the color of the layer that the surface belongs to.



Surface mesh render

- `Volume and surface mesh render together`: Boundary faces of volume mesh elements are shown as in `Volume mesh render`, but the thick borders are colored as the layer that the contour volume surface belongs to. In the picture below, the volume layer is green and the contour volume surface layer color is red.



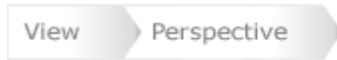
Volume and surface mesh render together

Note: The rendering of a volume mesh can be viewed with the boundary faces visible as if they were surface elements by using the option `View mesh boundary` (see [View mesh boundary](#)).

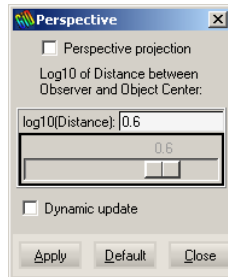
Note: The quality of visualization is controlled via preferences (see [Preferences](#)).

Perspective

Menu



By default, a model is viewed inside GiD using an orthogonal projection.

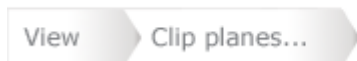


Perspective window

With this option it is possible to change to a perspective projection. In this mode, you can choose a distortion factor for the perspective. This can be updated at any time.

Clip planes

Menu



This window lets you hide the front or back of the view.

`Clip planes` is a way to prevent GiD from drawing elements of the geometry or mesh that are either very close to or far from the observer.

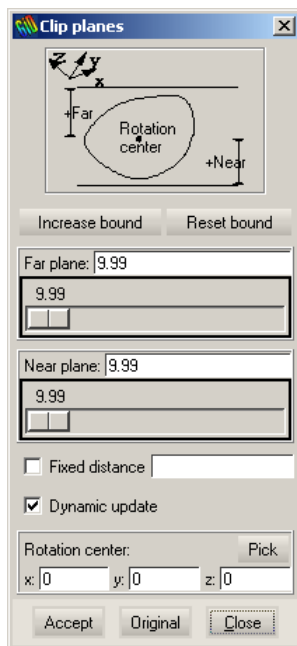
By moving the `Near plane` bar, the geometry that is closer to the viewer is hidden, while moving the `Far plane` bar hides the geometry that is further from the viewer.

To see the changes as they are performed, select `Dynamic update`. Selecting `Fixed distance` leaves a constant distance between both planes.

A new center of rotation can be entered or picked in the graphical window.

The `Increase boundaries` option adjusts the maximum setting possible for the `Far plane` and `Near plane` bars.

Note: All this information is reset when performing a zoom frame (see [Zoom](#)).



Clip planes window.

Label

Menu



Mouse menu



With this option you can choose whether or not the entities should have their labels displayed. As suggested below, there are three options: show all entities with labels, show no entities with labels, or show some with and some without. The options are:

- **All / All in:** All entities (**All**), or all entities of a particular entity type (**All in**), will have their labels displayed.
- **Select→Points, lines, surfaces or volumes:** To select some entities of a particular type, choose the desired entity type, then select entities in the usual way (see [Entity selection](#)).
- **Off:** No entity has its label displayed.

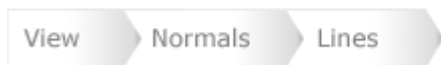
Entities

With this option, it is possible to choose just some of the points, lines, surfaces or volumes to be drawn. It is useful for making drawings faster or clearer in some instances.

Note: This option is only available in the **Right buttons** menu (see [USER INTERFACE](#)).

Normals

Menu



Menu



With this command GiD draws the direction of lines and the normal of surfaces.

- **Normals→Lines:** draws the direction of the selected lines. If the line lies on the plane $z=0$, GiD also displays the normal of the line in 2D.
- **Normals→Surfaces:** draws the normals of the selected surfaces. There are two ways of viewing surface normals: **Normal** (as an array) or **Colored** (the front and back faces of the surface are colored differently). Any surfaces belonging to the plane $z=0$ will, by default, have their normals oriented towards Z-positive. In this case, they are defined as anti-clockwise surfaces in 2D.

Viewing commands (`zoom`, `rotation`, etc.) can be applied and the normals will remain on the screen.

It is possible to swap the direction of lines or surfaces using the `Swap normals` command (see [Swap normals](#)).

Higher entities

Menu



With this option, it is possible to color-code geometrical entities according to their `Higher Entity number`; this is the number of other entities that a given entity belongs to.

If `mesh` mode is set, you can view the higher entities of mesh nodes.

Save/Read View

Menu



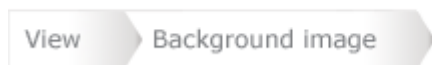
Menu



The first of these two options lets you save the actual view configuration to a file. That configuration can then be loaded at any time using the `View→Read` command.

Background Image

Menu

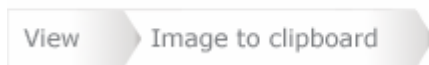


GiD allows an image to be used as a background for visualization purposes (the supported image formats are `.bmp` and `.ppm`). There are the following options:

- **Fit screen:** an image will be shown in GiD's background window; it will be modified if necessary so that it fills the screen correctly.
- **Real size:** the image will not be deformed; the image is placed in a plane. Three points must be entered: two of them to define the line where the bottom line of the image lies, and a stand-up point, which defines the upper direction of the image.
- **Default:** use this option to display the default background.

Image to clipboard

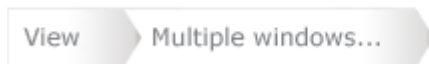
Menu



This option takes the image of the model in the actual view and sends it to the clipboard; it can then be pasted wherever you wish.

Multiple windows

Menu



The `Multiple windows` command lets you have several views of the same project. Different views can be displayed inside the program main window or in supplementary windows.

Mode

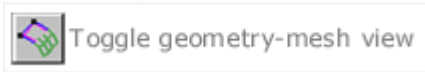
Menu



This command lets you choose the GiD mode you wish to work with (Preprocess or Postprocess), and the different visualization options in each mode:

In Preprocess: This command allows `Mesh` or `Geometry` visualization to be set.

Toolbar



The `Toggle` command changes from one visualization to another. If mesh visualization is selected and no mesh exists, you are asked to generate one.

In Postprocess: This command allows `Graphs` or `Mesh` visualization to be set.

GEOMETRY

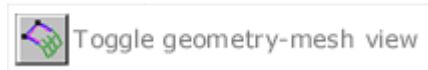
All available geometrical operations - generating, manipulating and deleting entities - are included in this chapter.

View geometry

Menu



Toolbar



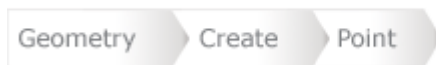
This command changes from mesh visualization to geometry visualization.

Create

This menu is for the generation of all the different possible geometrical entities. Usually, new entities are created inside the current layer (see [Layers](#)).

Point creation

Menu



Individual points are created by entering each point in the usual way (see [Point definition](#)). The points can then be joined together to form lines.

Caution: It is impossible to create new points joining old ones.

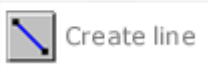
The `Number` option lets you choose the label that will be assigned to the next point created. If a point with this number already exists, the old line changes its number.

Straight line creation

Menu



Toolbar



To create a straight line, start by entering just two points (see [Point definition](#)), and then continue entering points in order to create more lines from the first one. Every part of the total line created is an independent line.

It is important to note that when creating lines, new points are also being created (if existing ones are not used).

The `Close` option joins the first point and the last point created with a straight line and finishes.

The `Undo` option undoes the creation of the last point (if new) and the last line. It is possible to continue undoing all the way back to the first point.

The `Number` option lets you choose the label that will be assigned to the next line created. If a line with this number already exists, the old line changes its number.

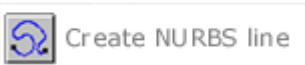
If `Join` is chosen, it is maintained for all points until `No join` is selected.

NURBS line creation

Menu



Toolbar



NURBS are non-uniform rational B-splines. They are a type of curve that can interpolate a set of points. NURBS can also be defined by their control polygon, another set of points that the curve approximates smoothly.

There are two ways of creating a NURBS line using this command, either by entering some interpolated points or entering the points that form the control polygon.

The `Undo` option undoes the creation of the last point; this can be done all the way back to the first point.

By default, a NURBS will be a cubic polynomial passing through all the points. However, this option can be changed by calling `ByControlPts`, which defines NURBS by their control polygon. This polygon is a set of points where the first and the last points match the first and last points of the curve. The rest of the points do not lie on the curve. It can be assumed that the curve approximates the points of the polygon in a smooth way. In this case, the user chooses the degree of the curve, which will be the degree of the connected polynomials that define the NURBS.

Instead of entering interpolated points, the `Fitting` option lets you approximate a line using a minimum squared criteria. You also have to select the degree of approximation for this curve.

When defining interpolating curves, you can choose to define the tangents to one or both ends (using the `Tangents` option). These tangents can be customized, in that they can either be defined by picking their direction on the screen or by considering an existing line as a tangent to the NURBS if it follows a previous curve (the option `ByLine`). The `Next` option allows only one tangent to be defined.

In this way, it is possible to create a closed NURBS by selecting the initial point as the end one and choosing one of the options '`Tangent`', '`Next`', or '`ByLine`'.

When a NURBS has been created, all the interior points (except the first and last) are not really entity points unless they previously existed.

The `Number` option lets you choose the label that will be assigned to the next created line. If a line with this number already exists, its number is changed.

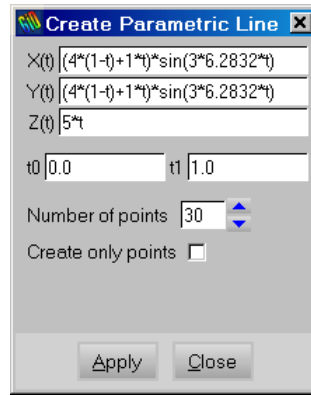
To enter rational weights on the curve, the `Edit NURBS line/surface` command (see [Edit NURBS line/surface](#)) can be used.

Parametric line

Menu



This is a tool to create a parametric approximated curve



The data that must be input are the mathematical formulae of the coordinates $X(t)$, $Y(t)$ and $Z(t)$, where 't' is the parameter of the curve, and its value belongs to the interval $[t_0-t_1]$. The curve is created by approximation and is a NURBS (Non-Uniform Rational B-Spline) which is created with N points. In GiD these kinds of curves are cubic (order 3).

The valid mathematical functions are all Tcl functions:

+ - * / %

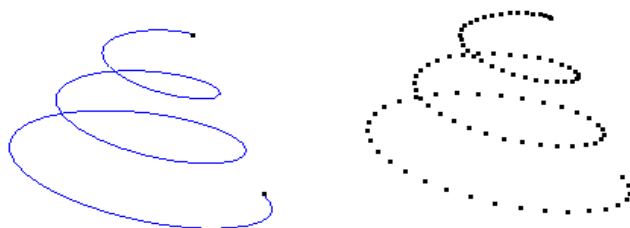
abs cosh log sqrt acos double log10 srand asin exp pow tan atan floor
rand tanh atan2 fmod round ceil hypot sin cos int sinh

Example:

We fill the formula bars with the expression of a conic helix.

That helix starts with radius $R_0=4$ and finishes with radius $R_1=1$, performing $N=3$ turns from $t=0.0$ to $t=1.0$, the height also changes from 0 to $H=5$.

$$\begin{aligned}
 x(t) &= (R_0 \cdot (1-t) + R_1) \cdot \sin(N \cdot 2\pi \cdot t) \\
 y(t) &= (R_0 \cdot (1-t) + R_1) \cdot \cos(N \cdot 2\pi \cdot t) \\
 z(t) &= H \cdot t
 \end{aligned}$$



Example of a conic helix with a unique curve or with points only

Polyline creation

Menu



Toolbar



A polyline is a set of at least two other lines of any type (including polylines themselves). Every line must share one or two of its endpoints with the endpoints of other lines.

There are two possible ways to create a polyline, either by selecting one line and searching the rest until a corner or end is reached, or by selecting several lines (see [Entity selection](#)). In the case of the latter, the order of selection is not important but all of them must join each other by sharing common points.

Polylines are drawn in green to show the difference between the other lines, which are drawn in blue.

Polylines are widely used when creating 4-sided surfaces (see [4-sided surface creation](#)) and automatic 4-sided surfaces (see [Automatic 4-sided surface creation](#)).

When deleting a polyline, all its lines are deleted. When exploding it (see [Polyline](#)), the polyline will disappear and its individual lines will appear.

It is not possible to create third level polylines: one former polyline can be included inside another, but this is the limit and these two cannot be included within a further polyline.

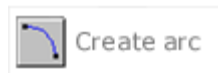
The `Number` option lets you choose the label that will be assigned to the next created line. If a line with this number already exists, its number is changed.

Arc creation

Menu



Toolbar



To create an arc you can either enter three points (By 3 points, see [Point definition](#)) or enter a radius and the two tangent lines at the arc's ends (`Fillet curves`).

It is important to note that when creating an arc, new points are also being created (if existing ones are not being used).

An arc that begins and ends at the same point (i.e. where the first and third points are the same) will be created as a circle. An arc will always include the second point that is entered, though this one is only used as a reference and, if it is not an existing point, is automatically erased when the arc is created.

The `Undo` option undoes the creation of the last point (if it is a new one). It is possible to continue undoing all the way back to the first point.

The `Fillet curves` option lets you input a radius and select two lines that share one common point. An arc will then be created and the two lines will be modified to be tangent and continuous with this new arc.

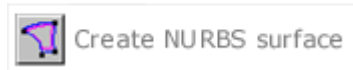
To convert one arc to another one with the same center and in the same plane but with a complementary angle, the `Swap arc` command can be used (see [Swap arc](#)).

NURBS surface creation

Menu



Toolbar



NURBS are non-uniform rational B-splines. They are a type of surface that is defined by its control polygon (one set of points that the surface approximates smoothly), one set of knots for the two directions u and v (a non-decreasing list of real numbers between 0 and 1) and, optionally, one set of rational weights.

To draw the isoparametric lines in $u, v=0.5$, check the surface drawing type option in the [Preferences](#) window.

- **By contour:** this creates a NURBS according to its contour lines. GiD automatically calculates the interior information of the surface so as to interpolate the boundaries smoothly. To create a NURBS surface, some lines must be selected (see [Entity selection](#)). The order of selection is not important but all of them must join each other by sharing common points and must form a closed contour. The number of lines must be equal to or greater than one and their shape must be topologically similar to a triangle or a quadrilateral in the space if the algorithm is to work correctly. This last argument is not necessary if all the lines lie in one plane. In this case, the surface is created as a trimmed one and any problems with the shape are avoided. It is possible to select the boundary lines and the boundary lines of interior holes at the same time, if all the lines belong to a plane.

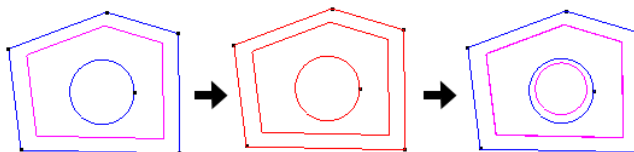
Note: The `No try planar` option (found in the **Contextual** mouse menu) avoids the creation of a trimmed NURBS surface when lines are coplanar.

Note: To enter rational weights for the surface, use the `Edit NURBS surface` command (see [Edit NURBS line/surface](#)).

- **Automatic:** this automatically creates all possible surfaces with the number of sides given by the user. Every new surface will be created in the current layer.

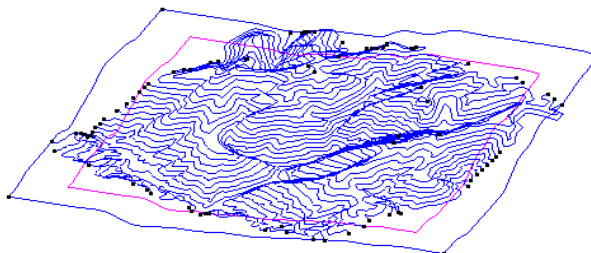
Caution: When creating more than one surface in one go, it is possible that some undesired surfaces may also be created. It is necessary to check the surfaces after creation and erase the undesired ones.

- **Trimmed:** this option lets you select one existing NURBS surface and a set of closed lines that are inside the surface. Some of these lines may already belong to the contour of the existing surface. Some other lines may be created with an intersection with another surface. Another new surface will be created without changing the old one. It is possible to select the boundary lines and the boundary lines of interior holes at the same time, if all the lines belong to the surface:



Creation of a new trimmed surface with a hole

- **Untrimmed:** this constructs one new surface with the selected surface as its base and with the natural contours of the NURBS surface as its contours. The resulting surface is not trimmed.
- **Parallel lines:** this lets you create one surface given a set of parallel lines in the space. The new surface will interpolate all the selected lines.
- **By points / By line points:** these two options are available in the **Contextual** mouse menu after the NURBS surface creation tool is selected. **By points** creates a NURBS surface from a cloud of points, and **By line points** creates a NURBS surface from level curves. These two functions are very useful for creating relief and terrains. In the image below there is a NURBS surface created from level curves:



Note: This surface is an approximation to the selected points/lines, but there is no interpolation.

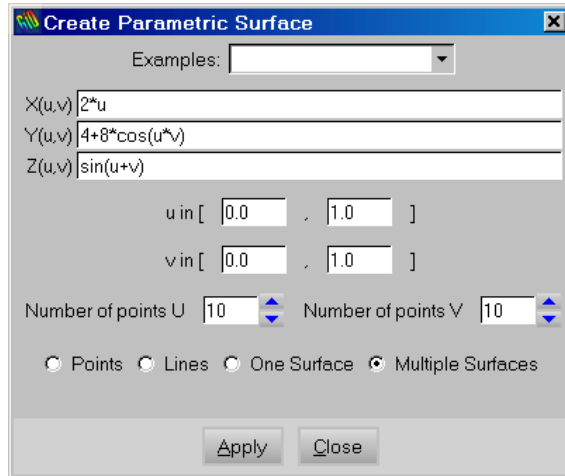
- **Search:** this lets you select one line and then creates one surface that contains that line.

Parametric surface

Menu



This is a tool to create a parametric approximated surface.



The required input data are the mathematical formulae of the coordinates $X(u, v)$, $Y(u, v)$ and $Z(u, v)$, where 'u' and 'v' are the parameters of the surface, and its value belongs to the intervals set in 'u in' and 'v in' respectively. The surface is created by approximation and is a NURBS (Non-Uniform Rational B-Spline), which is created with 'Number of points U' x 'Number of points V' points. In GiD these kinds of surfaces are cubic (order 3).

The valid mathematical functions are all Tcl functions:

+ - * / %

abs cosh log sqrt acos double log10 srand asin exp pow tan atan floor
rand tanh atan2 fmod round ceil hypot sin cos int sinh

Planar surface creation

Note: The planar surface has been substituted by the NURBS surface (see [NURBS surface creation](#)). The latter automatically detects if boundary lines lie in a plane and creates a planar NURBS.

Note: It is still possible, however, to access this function with the **Right buttons** menu (see [Tools](#)).

A planar surface is an entity formed by a closed set of lines, all of them lying on the same plane. These lines must share some common endpoints.

To create a planar surface, some lines must be selected (see [Entity selection](#)). The order of selection is not important but all of them must join each other by sharing common points and must form a closed contour. If all lines are not in the same plane the surface is not created.

It is possible to add holes to a planar surface. To do so, it is first necessary to create the outside planar surface. After this, press the `Hole` button and select the created surface. Then select lines that form every hole, one by one. Finish with `escape` (see [Escape](#)). It is also possible to define the surface and holes at the same time, by selecting all the curves.

If the surfaces lie on the plane $z=0$, the orientation of the surfaces will be anti-clockwise in this plane (the normal vector points towards Z-positive). Otherwise, orientation will be arbitrary. This can be checked with the `DrawNormals` command (see [Normals](#)).

4-sided surface creation

Note: The 4-sided surface has been substituted by the NURBS surface (see [NURBS surface creation](#)). This new entity has all the functionality of the old one.

Note: It is still possible, however, to access to this function with the **Right buttons** menu (see [Tools](#)).

A 4-sided surface is an entity formed by a closed set of four lines in the space. Its mathematical definition is a bilinear Coon's surface. The surface is totally defined by the shape of the lines, with no information about the interior. This means that it will sometimes be necessary to use more surfaces to obtain a good shape definition.

To create a 4-sided surface, several lines must be selected (see [Entity selection](#)). For the creation of a 4-sided surface defined by three lines, it is necessary to divide one of the lines into two pieces (see [Divide](#)). The order of selection is not important, but all of them must join each

other by sharing common points and they must form a closed contour. If it cannot be created, information about the endpoints is displayed in a window.

In order to make one or more lines form parts of a polyline (see [Polyline creation](#)), select the entire polyline as one of the lines and GiD will automatically select the piece or pieces of the polyline that are required. Using this facility, non-conforming surfaces can be created. This means creating a surface by using the entire line on one side of the polyline, and creating more than one 4-sided surface by using parts of it on the other side.

When selecting more than four lines, GiD will automatically search for all the possible 4-sided surfaces that can be created with these lines. This allows the creation of many surfaces at the same time.

The `Automatic` button is equivalent to `automatic 4-sided surface creation` (see [Automatic 4-sided surface creation](#)).

If the surfaces lie on the $z=0$ plane, the orientation of the surfaces will be anti-clockwise in this plane (normal vector points towards Z-positive). Otherwise, the orientation will be arbitrary. This can be checked with the `DrawNormals` command (see [Normals](#)).

The `Number` option lets you choose the label that will be assigned to the next created surface. If a surface with this number already exists, the old line changes its number.

Caution: When creating more than one surface at a time, some undesired surfaces may also be created. It is necessary to check the surfaces after creation and erase the undesired ones.

Automatic 4-sided surface creation

Note: The 4-sided surface has been substituted by the NURBS surface (see [NURBS surface creation](#)). This new entity has all the functionality of the old one.

Note: It is still possible, however, to access to this function with the **Right buttons** menu (see [Tools](#)).

Inside this option, GiD creates as many 4-sided surfaces as it can find. Every new surface will be created in the current layer.

Caution: When creating more than one surface at a time, some undesired surfaces may also be created. It is necessary to check the surfaces after creation and erase the undesired ones.

Contact surface creation

Menu

A horizontal menu bar with three items: 'Geometry', 'Create', and 'Contact surface'. Each item is inside a light gray arrow-shaped button pointing to the right. The 'Contact surface' button is highlighted with a darker gray background.

Contact surfaces are defined as being between two lines that are physically in the same place, but which have different line and point entities. From a contact surface, it is possible to generate contact elements, to be used by some calculation algorithms, which define a special contact between these two bodies.

Using contact surface entities is like a meshing specification. In this way, equal meshes will be generated for the two lines, ensuring a one-to-one relationship between nodes.

Choose the `Contact surface` option from the menu, and then select some lines on both bodies.

Contact elements are, by default, 4-node planar quadrilaterals. However, you can select 2-node lines for all cases (see [Element type](#)).

The 4-node planar quadrilaterals can be converted to the 8-node or 9-node for the quadratic case.

You can also select no mesh for the contact entity. This makes it possible to have exactly the same mesh for both lines but without any additional element between them.

Surface mesh

Menu

A horizontal menu bar with three items: 'Geometry', 'Create', and 'Surface mesh'. Each item is inside a light gray arrow-shaped button pointing to the right. The 'Surface mesh' button is highlighted with a darker gray background.

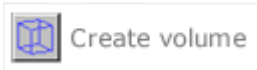
With this option a Surface mesh can be created by selecting triangular or quadrilateral mesh elements (see [Surface mesh](#)).

Volume creation

Menu

A horizontal menu bar with three items: 'Geometry', 'Create', and 'Volume'. Each item is inside a light gray arrow-shaped button pointing to the right. The 'Volume' button is highlighted with a darker gray background.

Toolbar



A volume is an entity formed by a closed set of surfaces that share the lines between them.

To create a volume, some surfaces must be selected (see [Entity selection](#)) using the `By contour` option. The order of selection is not important but all of them must join each other by sharing common lines and they must form a closed contour.

If there is an error and the volume is not created, a window appears with some useful information.

The `Search` option lets you select one surface and create one of the volumes that contains this surface.

It is possible to add holes to a volume. To do so, start by creating the outside and interior volumes as independent volumes. After this, click the `Hole` button and select the outside volume. Then, select the interior volumes that form every hole, one by one. Finish with `escape` (see [Escape](#)).

Volumes and their surfaces are automatically orientated so that they are meshed correctly.

An additional feature allows the selection of surfaces that form the outer part of the volume as well as the ones that form the holes at the same time. In this case, GiD automatically recognizes the holes.

The `Automatic 6-sided volumes` option creates all possible volumes that have 6 sides (contour surfaces). It can be applied several times over the geometry and volumes are not repeated. Every new volume will be created in the current layer.

This can be useful for structured meshing (see [Structured](#)).

Contact volume creation

Menu



Contact volumes are defined between two surfaces that are physically in the same place but with different surfaces, lines and points. From a contact volume, it is possible to generate

contact elements, to be used by some calculation algorithms, which define special contact between two bodies.

Those equivalent surfaces can be in the same location or can be separated by a movement (separated contact volume). The result will be equal meshes, ensuring a one-to-one relationship between nodes.

Choose `contact volume` from the menu, and then select the surfaces. GiD automatically searches for possible contacts, combining the selected surfaces in pairs.

Contact elements are, by default, 8-node hexahedra or 6-node prisms (depending on the surface mesh). However, you can select 2-node lines for all cases (see [Element type](#)).

The result elements can be also quadratic.

You can also select no mesh for the contact entity. This makes it possible to have exactly the same mesh for both surfaces but without any additional element between them.

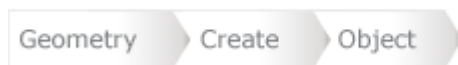
When creating contact volumes, GiD internally checks what surfaces occupy the same location in the space and creates the contact, therefore there is no need to specify what surfaces have to be in contact. For this reason, several surfaces can be selected at once and GiD performs the contact automatically, indicating the number of contact volumes that have been generated.

One feature of GiD is the option to create 'contact separated volumes' for surfaces that are not physically in contact.

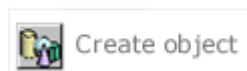
For these separated volumes, GiD internally checks whether a unique solid-rigid movement exists between two surfaces and creates the contact. There is the possibility that multiple solid-rigid movements may exist. In this situation, GiD asks for the point image of a source point to define the movement and, consequently, applies the right contact.

Object

Menu



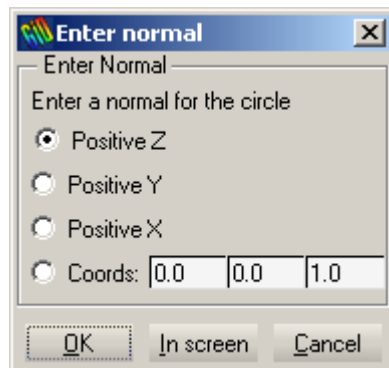
Toolbar



With this command it is possible to create the following kinds of objects:

- Rectangle
- Polygon
- Circle
- Sphere
- Cylinder
- Cone
- Prism
- Torus

When creating an object, GiD asks for a center and a normal. To enter the coordinates of the center you can click on the screen or select an existing point (see [Point definition](#)). To enter the normal, GiD displays a window where you can choose one of the three axes or enter the coordinates of a point.



Window to define the direction of the normal to the plane

The `In screen` button in the `Enter normal` window lets you manually enter the coordinates of the point which defines the normal: you can directly click on the screen or pick an existing point using the `Join` option in the **Contextual** mouse menu.

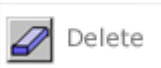
When using the commands `sphere`, `cylinder`, `cone`, `prism` or `torus`, the volume of the object is also created.

Delete

Menu



Toolbar



The deletion of entities can be done in two ways: at one level (point, line, surface or volume) or erasing all entities at once. A selection is made (see [Entity selection](#)) in both cases. After pressing `escape` (see [Escape](#)), the entities are erased.

To undo the selection of entities, press `Clear selection` in the **Contextual** menu.

Entities that form the basis of higher entities cannot be erased. For example, if a surface is created over some lines, it is necessary to erase the surface before erasing the lines.

Edit

These are the GiD editing options for geometrical entities:

- Move point
- Divide lines, polylines or surfaces
- Join lines end points
- Force lines to be tangent
- Swap arc
- Explode or edit polylines
- Edit SurfMesh
- Edit NURBS lines or surfaces
- Convert to NURBS lines or surfaces
- Simplify NURBS lines or surfaces
- Hole NURBS surface
- Collapse or Uncollapse entities or model
- Intersections between entities
- Surface or volume boolean operations

Move point

Menu



By using this command, an existing point is selected and moved. The new position is entered in the usual way (see [Point definition](#)). If the new position is an existing point (when using `Join`), GiD will determine the distance between the points and ask if they should be joined. If the answer is yes, both points are converted into one. Any lines or surfaces that include the point in question will be moved accordingly in order that any links are maintained; this may lead to these lines or surfaces being distorted.

Divide

Menu



The `Divide` command can be applied either to lines, polylines or surfaces (including trimmed surfaces).

In the case of polylines, an existing interior point must be chosen. The polyline will be converted into two lines that may or may not be polylines.

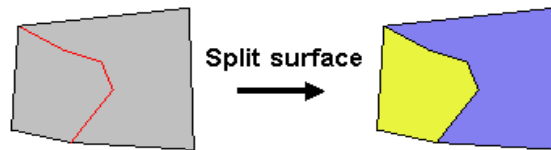
Polyline division has the option `Angle` which allows you to divide the polyline at all the points where the angle between the sub-lines is greater than a given value.

Caution: An interior point must belong to the first level of a polyline (see [Polyline creation](#)).

In the case of lines and surfaces, once the entity has been selected the division can be done in several ways:

- `Number of divisions`: The line or surface will be converted into equally spaced pieces.
- `Near point`: With this option one point must be selected near the line or the surface. Points inside the entities can be selected (see [Point in line](#), or [Point in surface](#)). The line or the surface will be divided into two entities near that point.
- `Parameter`: One factor is given between 0.0 and 1.0 and the entity will be divided where the parametric variable takes that value.
- `Lines`:

- **Relative length:** One factor is given between 0.0 and 1.0 to divide the line with relative arc length ratio equal to the selected factor. (Same concept as **Parameter** if the curve was arc length parameterized).
- **Length:** The length of the resulting divided lines is given, and GiD divides the line into as many lines as it can. If the length given is bigger than the length of the selected line, no division is made.
- **Surfaces:**
 - **Split:** The surface will be divided following the divide lines. These lines must intersect the surface contour (**Geometry Edit SplitSurf**).



In the case of surfaces it is necessary to give the division direction as **u** or **v**. This can be checked beforehand with the command **Edit NURBS Surface** (see [Edit NURBS line/surface](#)).

Note: After the division, the old entity disappears and the new entities are created.

Line operations

Menu



With this option you can edit groups of lines with respect to their topology and shape.

Join lines end points:

With the command **Join lines end points**, two lines must be selected. GiD determines the distance between the two closest endpoints, draws both points, and asks for confirmation. If one of the lines is a polyline, interior points are also considered. If accepted, the points are converted into one and the lines are distorted. The new point will then take the place of the first line's point.

(See [Move point](#) for another method of converting two points to one.)

Caution: The second selected line cannot have higher entities (the second point is moved to the first).

Force to be tangent:

With the command `Force to be tangent`, two lines (which share at least one point) must be selected. They must be NURBS lines, otherwise they will be rejected. You are asked to enter the maximum angle between lines to accept the operation, and GiD will modify the selected NURBS lines and force them to be tangents at their common point.

Swap arc

Menu



This command lets you select and alter arcs. Lines that are not arcs are rejected. When you confirm the operation, the arc is converted to a new arc with the same center and in the same plane but opposite the old one. The old arc disappears and the angle of the new arc will be complementary to the angle of the old arc.

Caution: Arcs belonging to higher entities cannot be swapped.

Polyline

Menu



Explode polyline:

This command lets you select which polylines you wish to explode; lines that are not polylines or have higher entities or conditions are rejected. After confirmation, the polylines are exploded and converted back to their original lines. Polylines then disappear (see [Polyline creation](#)).

Edit polyline:

The `Edit Polyline` command allows you to select which polylines you wish to edit; lines that are not polylines are rejected. It is possible to choose several options for the polylines:

- **Use points:** When meshing this polyline, there will be at least one node at every point location that defines the polyline. These will be the endpoints of interior lines.
- **Not use points:** When meshing this polyline, the mesh generator ignores the points and therefore the nodes will be placed anywhere. This is the default option. Nodes will only be put in the position of a point if there is a 4-sided surface over a part of a polyline (see [Automatic 4-sided surface creation](#)).
- **Only points:** When meshing this polyline, the nodes will only be placed where the geometry points are.

Note: If one condition is assigned to one interior point of a polyline (see [Conditions](#)), one node of the mesh will be placed over that point.

SurfMesh

Menu



Select one or several surface meshes (see [Surface mesh](#)). The options are:

- **Draw mesh:** Surface will be drawn as a mesh.
- **No draw mesh:** Surface will be drawn as a regular surface with magenta lines close to the boundary lines.

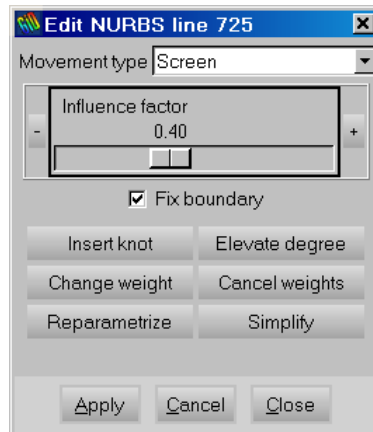
Edit NURBS line/surface

Menu



Edit NURBS line:

Tool to modify some NURBS geometric properties, like control points, degree, etc.



Edit NURBS line window

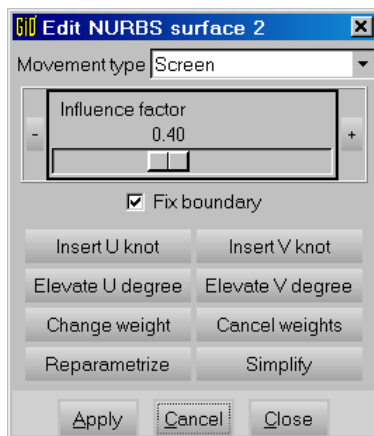
Once a NURBS line is selected (use the **Pick** button in the **Edit NURBS Line** window), you can edit its control points (see [NURBS line creation](#)). Select the control points as if they were regular points and enter their new positions in the usual way (see [Point definition](#)).

The Influence factor affects the movement propagation of the neighboring control points.

Available options:

- **Fix boundary** Check the fix boundary option if you do not want to move the boundary control points of the line.
- **Insert knot:** You are asked for a knot value between 0.0 and 1.0 and this is then inserted. The program checks that the knot multiplicity is not greater than the order ($\text{order} = \text{degree} + 1$). As the number of knots increases, the number of control points also increases, so this option can be used to have more points defining the same curve.
- **Elevate degree:** With this option the degree of the curve is raised by one. The new curve will have the same shape but with more control points and knots.
- **Change weight:** A new positive weight can be introduced for any control point, with the exception of the end points.
- **Cancel weights:** All weights of the NURBS are converted to 1.0 and the curve is no longer rational.
- **Reparameterize:** With the same control points a new curve is calculated to get a better curve with a more uniform parameterization.
- **Simplify:** This option converts the curve to a simplified curve, which is an approximation of the original one.

Edit NURBS surface:

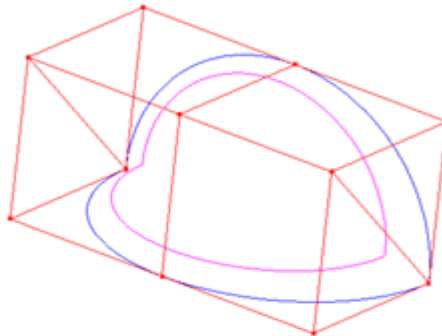


Edit NURBS surface window

Once a NURBS surface is selected (use the `Pick` button of the `Edit NURBS Surface` window), you can edit its control points interactively (see [NURBS surface creation](#)). Select the control points as if they were regular points and enter their new positions in the usual way (see [Point definition](#)).

Available options:

- **Insert knot:** You are asked for a knot value between 0.0 and 1.0 and it is then inserted. The program checks that the knot multiplicity is not greater than the order. This option can be used to have more points defining the same surface.
- **Elevate degree:** With this option the degree of the surface is raised by one. The new surface will have the same shape but with more control points and knots.
- **Change Weight:** A new positive weight can be introduced for any control point, with the exception of the end points.
- **Cancel weights:** This converts the weights of all the control points to 1.0.
- **Simplify:** This option simplifies the surface (if possible), removing the less significant knots.
- **Reparameterize:** This reparameterizes the surface obtaining an optimized surface. When a Nurbs surface is not well parameterized, the mesh is of a lower quality.



Control polygon of a NURB surface

The `Movement` type menu of the `Edit NURBS Surface` window determines the way the selected knots will move. This movement can be along an axis (`X-axis`, `Y-axis`, `Z-axis`), can describe the Normal of the surface (`Normal`), can follow the screen movement of the mouse (`Screen`), or the new location of the knot can be defined by introducing the coordinates of a point (`Point`).

Note: The `Insert knot` and `Degree elevate` options can be chosen for either the `u` or the `v` parameter directions.

Convert to NURBS line/surface

Menu



This option converts the selected lines or surfaces to NURBS lines or NURBS surfaces.

Note: Some algorithms only work with NURBS entities.

Simplify NURBS line/surface

Menu



This option converts the selected NURBS lines or surfaces to other ones very similar to the originals but with a less complicated definition. It can be useful when importing data where a control polygon is too complex for GiD to display or mesh quickly.

Hole NURBS surface

Menu



With this option you can select one existing NURBS surface and a set of closed lines that are inside it and that form a hole. The lines may be created by an intersection with another surface. The hole will be added to the existing surface.

Collapse

Menu



The `Collapse` function converts coincident entities, i.e. entities that are very close to each other, into one.

The `ImportTolerance` variable (see [Preferences](#)) determines which entities will be collapsed. Where the distance between two points is less than the tolerance, they will be converted to one. With lines and surfaces, the maximum distance between both entities is calculated and if it is less than `ImportTolerance`, they are converted to one.

Select the type of entities - point, line, surface or volume - when in **geometry** mode. All the lower entities that belong to the selected entities will automatically be computed. On pressing `escape`, the collapse operation will be performed.

The `Model` option performs the operation over all the geometrical entities in the model.

Uncollapse

Menu



The `Uncollapse` function lets you select lines, surfaces or volumes and duplicate all common lower entities.

Typically, if two surfaces share one line as an edge, after applying this function to both surfaces, that line and its shared points will be duplicated and every line will belong to a different surface.

This feature is interesting, for example, if you want to disconnect joined bodies or generate a non-conformal mesh with fewer elements than a conformal one.

Intersection

Menu



Using this option, the intersection of many geometrical entities can be performed.

Intersection: Line-line

With the command `Intersect lines`, two lines must be selected. GiD then searches for the closest points between the two lines. If the two lines already touch, a new point will be created where they cross.

If the lines do not intersect, GiD determines the point where the two lines are the closest, it draws both points and asks for confirmation. If confirmed, the new point on the line selected second is moved to the new point on the line selected first, creating a unique point. In the case of an arc and a straight line, for example, this effectively creates four lines out of two, but with two straight lines, where an endpoint is highlighted, GiD simply extends the second line until it reaches the first.

The `No Divide Lines` option creates the intersection point, i.e. draws the points where the lines are closest, but does not modify the lines.

Note: Polylines cannot accept this option.

Caution: The second selected line can only have higher entities if it is not necessary to extend this line.

Intersection: Multiple lines

This option lets you select several lines for which GiD then tries to find as many intersection points as possible. Lines are divided where applicable.

The `No Divide Lines` option creates an intersection point but does not modify the lines.

Intersection: Surface-2 points

You need to select one surface and two points that lie approximately over it. GiD calculates the line intersection between the surface and a plane defined by the two given points and the average normal to the surface of these points.

Note: Planar surfaces cannot be used with this option.

Note: See [Point in line](#) and [Point in surface](#) which can be used to define the points.

Intersection: Surface-lines

You need to select one NURBS surface and several lines. GiD then calculates the intersection between the surface and the lines. Lines will be divided at the intersection point.

The `No Divide Lines` option creates the intersection point but does not modify the lines.

The `Extend/Divide lines` option extends the lines until they reach the surface.

Intersection: Surface-surface

This command creates the intersection lines between two surfaces. If these lines intersect surface contour lines, they are divided.

The `No Divide Lines` option creates the intersection point but does not modify the contour lines.

By default the surfaces are divided, unless the `No divide surface` option is selected.

Intersection: Multiple surfaces

This command creates the intersection lines between surfaces.

Surface boolean operations

Menu



You need to select two 2D surfaces located in the XY plane (order is important when dealing with subtraction).

The valid surface boolean operations are:

- **Union:** Fuses two surfaces wherever they intersect to create a single, more complex volume.
- **Intersection:** Creates a surface based on the intersecting points of two separate volumes.
- **Subtraction:** Negates a specific portion of a surface to create a hole or indentation.

Volume boolean operations

Menu



The GiD Volume Boolean Modeler has been designed to accomplish geometric feats such as physically punching a hole through a volume, combining two volumes into one, and creating a new volume from the intersecting points of two separate volumes.

The valid volume boolean operations are:

- **Union:** Fuses two volumes wherever they intersect to create a single, more complex volume.
- **Intersection:** Creates a volume based on the intersecting points of two separate volumes.
- **Subtraction:** Negates a specific portion of a volume to create a hole or indentation.

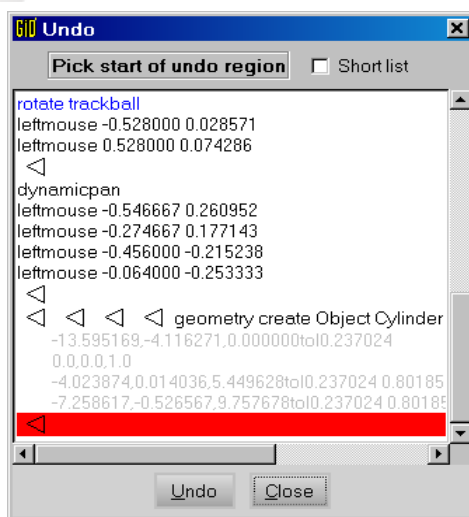
Two volumes must be selected (order is important when dealing with subtraction).

UTILITIES

Within **Utilities** you can find commands that apply to the geometry and/or the mesh, as well as other functions that apply to the whole project.

Undo

Menu



Undo window

With this command you can undo any previous commands executed since the the project was last saved or read. To do this, select from the list of operations in the window so that they are highlighted red, and click **undo**.

Preferences

Menu



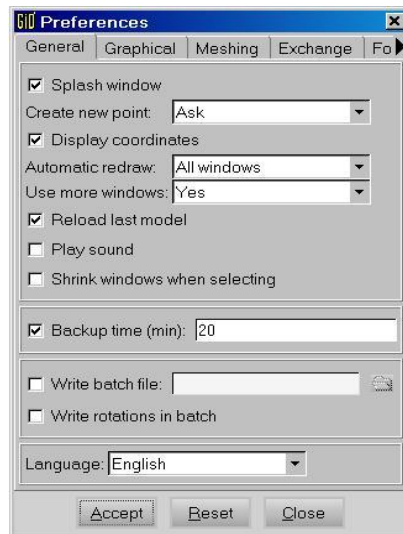
Toolbar



Note: There are many settings in GiD that have a predefined value, but that can be modified by the user. They can be accessed in one of two ways. Firstly, by opening the **Preferences** window from the `Utilities` pull-down menu, and secondly via the `Variables` command in the `Utilities` section of the **Right buttons** menu (the latter is also available in the **Contextual** mouse menu). Almost all the preferences variables are present in the Preferences window, but some advanced ones are only available in `Variables` command.

In the following section the different options available in the Preferences window are shown and their different settings explained.

The first group of **Preferences** are general options, and are used to set the different ways of working with GiD.

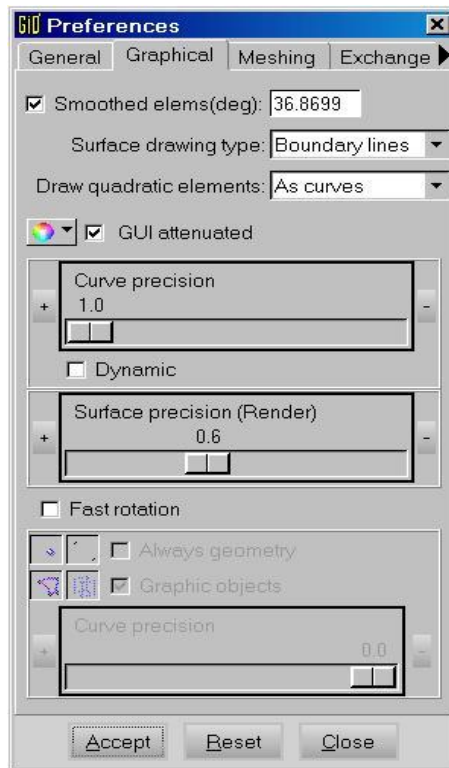


General preferences

- **Splash window:** If this option is set, when the program is opened again a welcome window is displayed. Variable: `SplashWindow`. Values: 1,0. Default is 1 (Yes).
- **Create new point:** Alters the way in which the points are entered in GiD (see [Point definition](#)). Options are:
 - **Always:** If trying to create a new point in the vicinity of an existing one, the new point is always created.
 - **Ask:** If trying to create a new point in the vicinity of an existing one, GiD asks whether to make use of the existing point or create a new one.
 - **Never:** Only allows existing points to be selected. You can also change this when in point creation mode by setting No join until all points are entered.Variable: `CreateAlwaysNewPoint`. Respective values: 1,0,-2. Default is 0 (Ask).
- **Display coordinates:** If set, GiD shows the coordinates in the graphical window. Variable: `DisplayCoordinates`. Values: 1,0. Default is 1 (Yes).
- **Automatic redraw:** If this option is not set, redraws caused by window expositions or internal functions are not performed. This avoids spending a lot of time redrawing when working with large models. Variable: `AutomaticRedraw`. Values: 1,0. Default is 1 (Yes).
- **Use more windows:** Depending on the selected field - yes, no or beginner - any messages are either displayed in new windows or in the regular messages window. Variable: `UseMoreWindows`. Values: 1,0,2. Default is 2 (beginner).
- **Reload last model:** If set, when GiD is opened it reloads the last model that was saved. Variable: `ReloadLastModel`. Values: 1,0. Default is 0 (No).
- **Play sound:** If this option is set, a sound is played to indicate a task has finished. Variable: `Sound`. Values: 1,0. Default is 0 (No).
- **Shrink windows when selecting:** If this option is set, some windows are resized when selecting, in order to facilitate the selection. Variable: `SmallWinSelecting`. Values: 1,0. Default is 0 (No).
- **Backup time:** If this option is set and a model name is given, the model is saved automatically with the frequency given (in minutes). **Note:** The mesh is not saved in automatic backup to save time. Variable: `BackUpMinutes`. Default is 20 minutes.
- **Write batchfile:** If this option is set and a filename is given, all commands used during the current session are saved to this file. It can be executed later by means of a script file with the predefined commands to be run in GiD (see [Batch file](#)). Variable: `BatchFileToWrite`.
- **Write rotations in batch:** If this option is set and the batch file name is given, all dynamic rotations and movements are stored there as well. In this way you can see these movements when reading this batch file with 'Read batch window'. Variable: `WriteRotationsInBatch`. Values: 1,0. Default is 0 (No).

- **Language:** This option sets the language GiD is working in. RamTranslator is used to deal with message catalogues for GiD and the translation of problem types (to see more information about RamTranslator visit <http://www.gidhome.com>).

The second group of **preferences** are graphical options, and are used to set different ways of visualizing the model. They do not change the geometry or the model information.




Graphical preferences

- **Smoothed Elements:** If this option is set, when rendering a mesh (see [Render](#)) the intersection between elements with a small angle between their normals will be illuminated as if it were a continuous solid. If it is not set, illumination is made considering every element as planar. Variable: LightSmoothedElements. Values: 1,0. Default is 1 (Yes). The angle (in degrees) is the maximum angle between the normals of two elements to allow smooth lighting between them. If the angle is greater than this,

one edge is drawn between them. Variable: `CosSmoothedElems`. Saved as the cosine of this angle. Default is 0.8.

- **Surface drawing type:** Lets you choose how to draw the surfaces when in wire frame (normal) mode. Options are:
 - **None:** Surfaces are not drawn.
 - **Boundary lines:** One magenta line is drawn for every contour line. This set of lines has a small offset towards the interior of the surface.
 - **Isoparametric lines:** Two yellow lines are drawn for every NURBS surface, one for $u=0.5$ and one for $v=0.5$.
 - **Both:** Draws both Boundary lines and Isoparametric lines.

Variable: `DrawSurfaceMode`. Default is 1 (Boundary lines).

- **Draw quadratic elements:** With this option you can specify how to visualize quadratic elements. Options are:
 - **No:** If this option is set, quadratic elements are drawn as if they were linear.
 - **As lines:** If this option is set, elements are drawn taking into account all quadratic nodes, but edges are drawn using straight lines between nodes.
 - **As curves:** If this option is set, elements are drawn taking into account all quadratic nodes, and edges are represented using quadratic curves. This option is more realistic, but it may make some visualization aspects slower when dealing with large models.
- **Change color:**  It is possible to change the default color of several visual items in GiD. One possibility is to change the background color of the GiD graphical window. Variable: `BackgroundColor`. Value: hexadecimal RGB char. Default is (255,255,255) (white). Other possibilities include the color of the entities in normal mode (no render). These are also accessible through the Variables menu.
- **GUI attenuated:** Set the GUI appearance to standard or attenuated color. The change is updated after restarting GiD.
- **Curve precision:** This option determines the precision with which curves are drawn. The internal definition of curves does not change. Selecting the Dynamic option allows this setting to be changed while drawing a curve in order to test the level of precision. Variable: `CurvePrecision`. Values: 1.0 to 0.0 from best to worst. Default is 1.0.
- **Surface precision:** This option determines the precision with which surfaces are drawn in render mode. The internal definition of surfaces does not change. Variable: `SurfacePrecision`. Values: 1.0 to 0.0 from best to worst. Default is 0.6
- **Automatic rotation center:** If this option is active, each time 'Zoom In' / 'Zoom Out' / 'Pan' is used, the point of the geometry or mesh nearest to the center of the screen will be taken as the center of rotation for subsequent rotations. Variable: `AutomaticRotationCenter`. Values: 0,1. Default is 1 (yes). This is also available

by right-clicking the mouse, under `Rotation` → `Center`. If a new center of rotation is selected (see [Rotate center](#)), then this option is deactivated.

- `Fast rotation`: This allows fast rotation of the object. Variable: `FastRotation`. Value: 0,1. Default is 0 (No).

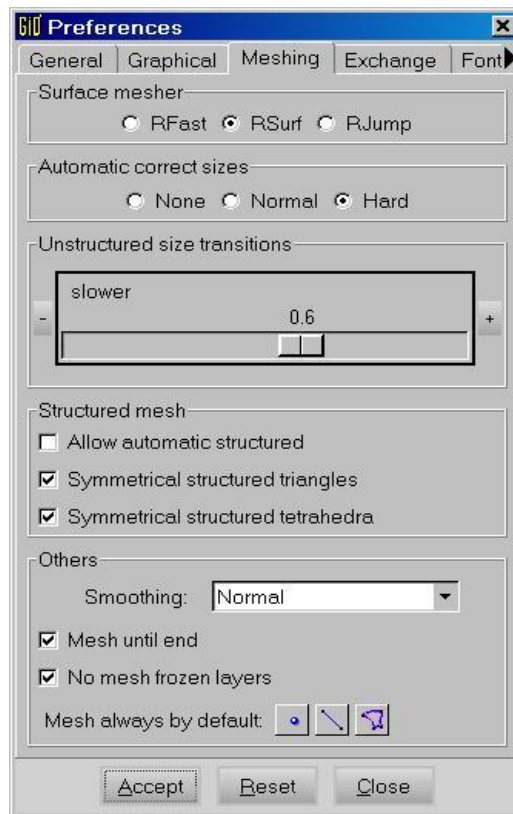
If this option is set various further options are available, though these only apply when rotating.

- `Points, lines, surfaces, volumes`: This determines whether the entities will be drawn when rotating. Variables: the same. Values: 0,1. Default is 0 (No).
- `Always Geometry`: With this option set, when you view and rotate the mesh, the geometry is drawn instead. Variable: `UseAlwaysGeom`. Values: 0,1. Default is 0 (No).
- `Draw graphic objects`: If this option is not set, when rotating the geometry, some graphical and temporal objects like normals or materials or conditions symbols are not drawn. Variable: `DrawGraphicObjects`. Values: 0,1. Default is 1 (Yes).
- `Curve precision`: This is the same as the general item `Curve precision`, but only applies when rotating. Variable: `CurvePrecision`. Values: 1.0 to 0.0 from best to worst. Default is 0.8

The third group of **preferences** are meshing options.

- `Surface mesher`: GiD can use three kinds of surface mesher.
 - `RFast` meshes are the most efficient in speed and reliability. With deformed surfaces they can give distorted elements.
 - `RSurf` meshes are generated directly in the space. Quality is better but it is slower and can fail for distorted surfaces.
 - `RJump` meshes are generated directly in the space, and contact lines between surfaces which are almost tangent (less than 10 degrees between tangent vectors) are skipped when meshing. Contact points between almost tangent lines (less than 10 degrees between tangent vectors) are skipped too. With this surface mesher, you can generate meshes with fewer elements than other meshers because it is less dependent on the dimensions of geometrical surfaces. However, it is slower and can fail for distorted surface patches.

If any entity has `Skip` or `NoSkip` Mesh Criteria (see [Mesh criteria](#)), the surfaces involved are meshed with the `RJump` mesher, even if another mesher is set in this window. These mesh generators are based on the advancing front generation mesh technique in order to improve speed and portability. Variable: `SurfaceMesher`. Values: 0 (`RFast`), 1 (`RSurf`) and 4 (`RJump`). Default is 0 (`RFast`). **Note:** GiD can try another mesher internally when one of them fails to generate the mesh for one surface.



Meshing preferences

- Automatic correct sizes:** This preference lets GiD make an automatic mesh size correction when meshing begins. If *None* is set, no size correction is made; if *Normal* is set, a size correction is made according to the sizes of geometrical entities and the compatibility between meshing sizes of neighboring entities; if *Hard* is set, the *Normal* correction is made and, furthermore, an automatic chordal error criteria is applied to assign sizes to surfaces which are the contours of some volume, so as to improve volume meshes. Variable: *AutomaticCorrectSizes*. Values: 0 (*None*), 1 (*Normal*) and 2 (*Hard*). Default is 1 (*Normal*).
- Unstructured size transitions:** This controls whether the transitions between different element sizes are slow or fast. Variable: *SizeTransitionsFactor*. Values: 1.0 to 0.0 from slower to faster transition. Default is 0.6.

- **Allow automatic structured:** If this preference is set, functions like Assign sizes by Chordal Error will define some surfaces as structured with highly distorted elements over them. Variable: `AllowAutomaticStructured`. Values: 0,1. Default is 0 (No).
- **Symmetrical structured triangles:** If this preference is set, structured triangle meshes will be topologically symmetrical. Variable: `SymmetricalStructuredTriangles`. Values: 0,1. Default is 1 (Yes).
- **Symmetrical structured tetrahedra:** If this preference is set, structured tetrahedron meshes will be topologically symmetrical. Variable: `SymmetricalStructuredTetrahedra`. Values: 0,1. Default is 1 (Yes).
- **Smoothing:** You can choose the level of smoothing required to enhance the mesh after the generation. Options are:
 - **Normal:** only the standard smoothing is performed.
 - **HighAngle:** an additional smoothing with angle criteria is performed.
 - **HighGeom:** an additional smoothing with chordal error criteria is performed.
 Variable: `HighQualitySmoothing`. Values: 0 (Normal), 1 (HighAngle) and 2 (HighGeom). Default is 1 (HighAngle).
 - **Mesh until End:** If this preference is set, the mesh generator will continue until it has completed the operation even if there are surfaces or volumes that cannot be meshed. Variable: `MeshUntilEnd`. Values: 0,1. Default is 1 (Yes).
- **No Mesh Frozen layers:** If this preference is set, entities belonging to a frozen layer will not be meshed. Variable: `NoMeshFrozenLayers`. Values: 0,1. Default is 1 (Yes).
- **Mesh always by default:** Changes the default meshing criteria. Entities will always be meshed even if they have higher entities. Example: If surfaces are checked when meshing a volume, volume elements and surface elements will be obtained. Variable: `ForceMeshEntities`. Values: 0 (No entity), 1 (Points), 2 (Lines), 3 (Points and Lines), 4 (Surfaces), 5 (Points and Surfaces), 6 (Lines and Surfaces), 7 (Points, Lines and Surfaces). Default is 0 (No entity).

The fourth group of **preferences** are geometry exchange (import and export) options.

Import options:

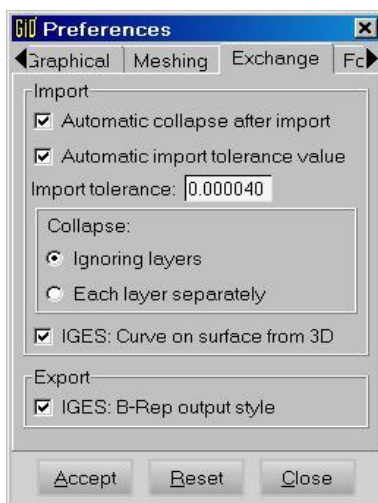
- **Automatic Collapse After Import:** If this option is set then after reading one IGES file, one global collapse is made. If it is not set, all surfaces and lines will be independent of each other. Variable: `AutoCollapseAfterImport`. Default is active (1).
- **Automatic Import tolerance:** When importing a file or collapsing entities, any points closer together than this distance are considered to be the same (see [IGES](#)). Lines and surfaces can also be collapsed. Variable: `ImportTolerance`. Default is 0.001.

- Collapse
 - Ignoring Layers: Entities are also collapsed if they belong to different layers.
 - Each layer separately: Entities in different layers are not collapsed. (Entities belonging to frozen layers are never considered.)

Variable: CollapseIgnoringLayers. Value: 0 (Each layer separately), 1 (Ignoring Layers). Default is 1 (Ignoring Layers).

- IGES: Curve on surface from 3D: If this option is set, IGES curves on surface entities are created from the direct 3D space definition (recommended); if the option is not set, IGES curves are created from the surface space parameter definition.

Variable: IGESCurveOnSurfaceFrom3D. Value: 0 (space parameter definition), 1 (3D space definition). Default is 1 (3D space definition).

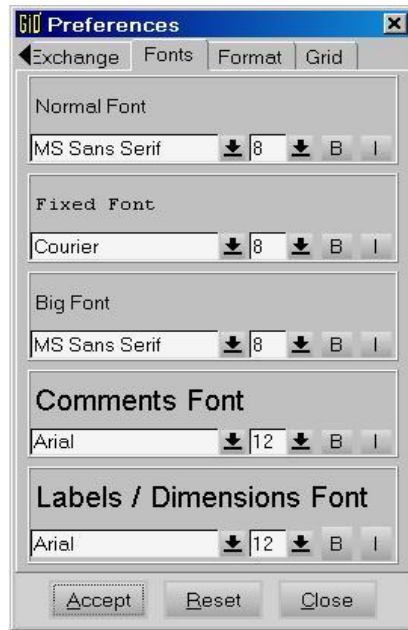


Exchange preferences

Export options:

- IGES: B-Rep output style: If this option is set, exported IGES entities are written with Boundary Representation Solid Model style; otherwise the surfaces are written as separated trimmed surfaces without topological information. Variable: IGESSolidsManifoldBRep. Value: 0 (no B-Rep output style), 1 (B-Rep output style). Default is 1 (B-Rep output style).

The fifth group of **preferences** deals with the fonts used inside GiD.

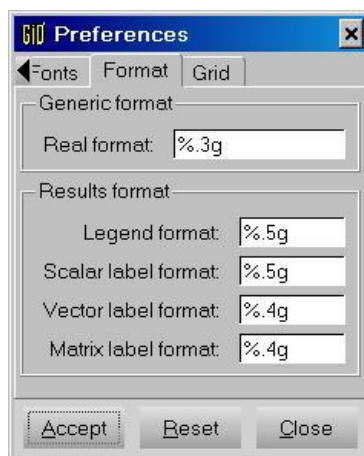


Font preferences

- `Normal font`: This is the font normally used inside GiD.
- `Fixed font`: This font must have the same spacing for every letter. It is used in places where this property is necessary.
- `Big font`: Used in some dialog boxes.
- `Comments font`: Used for comments.
- `Labels/Dimensions font`: Used in labels and dimensions.

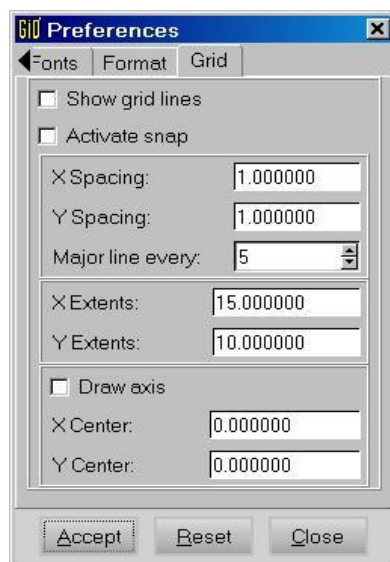
The sixth group of **preferences** deals with numerical formats used inside GiD.

- `Generic format`: This option refers to numerical formats used in other GiD utilities (for example in the coordinates display).
- `Results format`: This option refers to numerical formats in results (Postprocess).



Format preferences

The seventh group of **preferences** contains grid options (see [Grid](#)).



Grid preferences

- **Show grid lines:** If this option is set, grid lines are shown. Variable: `Grid(Show)`. Value: 0,1. Default is 0 (No show).
- **Activate snap:** If this option is set, snap is activated. Variable: `Grid(Active)`. Value: 0,1. Default is 0 (No activate).
- **X/Y Spacing:** These options determine the spacing between grid lines in the X and Y directions. Variable: `Grid(SpacingX)` and `Grid(SpacingY)`. Default value is 1.0 in both directions.
- **Major line every:** This option specifies the number of lines between principal lines in the grid. Variable: `Grid(MajorLineEvery)`. Default value is 5 lines.
- **X/Y Extents:** These options determine the extension of the grid in the X and Y directions. Variable: `Grid(ExtentsX)` and `Grid(ExtentsY)`. Default value is 15.0 in X direction and 10.0 in Y direction.
- **Draw axis:** If this option is set, 2D axes are shown in the grid (X and Y). Variable: `Grid(DrawAxis)`. Value: 0,1. Default is 0 (No axis drawn).
- **X/Y Center:** These options determine the position of the center of the grid. Variable: `Grid(CenterX)` and `Grid(CenterY)`. Default value is 0.0 (center is at x=0.0,y=0.0).

Layers

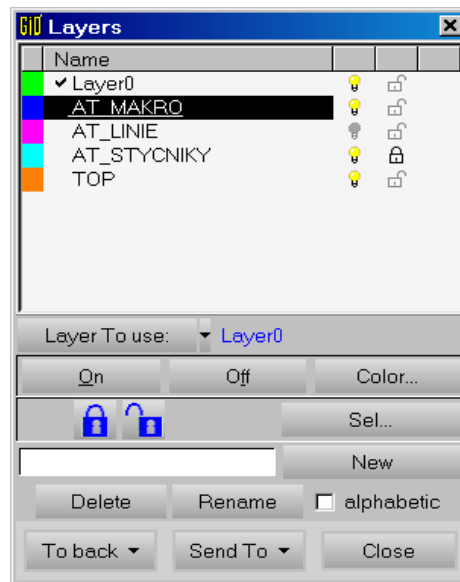
Mouse menu



Toolbar





Layers are a way to split a complex drawing up into separate pieces. The idea is that any entity can belong to one layer or to none (an entity cannot belong to more than one layer). In this way, it is possible to view only some layers and not others. It is also useful for making it easier to select entities in the graphical window.



Layers window

The commands relating to layers are as follows:

- **Layer To use:** Selects a layer to be used as a default. All the new entities will be created within this layer. All the layers can be selected here. Next to the Layer To use button, there is an arrow facing down. This is a menu which lets you select a point, a line, a surface or a volume; the layer to which that entity belongs will be set as the default layer.
- **On:** Entities belonging to this layer will be drawn and can be selected in the graphical window.
- **Off:** Entities belonging to this layer will not be drawn and cannot be selected in the graphical window.
- **Color:** Changes the color of the layer. This color is used when performing a rendering (see [Render](#)). The color can also be chosen by clicking on the colored box next to the layer name in the list.
- **Lock/Unlock**   (Freeze/Unfreeze): Entities belonging to this layer will be drawn but cannot be selected in any way. When copying entities (see [Copy](#)) and sharing old entities, entities belonging to frozen layers are not taken into account and not even a range of numbers (see [Entity selection](#)) can be selected. The opposite command is **Unlock**, where the entities belonging to this layer can be selected, copied and shared.

- **Sel...**: In the layer window, there is the option `select` which allows the selection of several layers. This can be useful when using a large number of layers. After pressing `Sel...` a new window appears to allow the input of a string of characters that will match the layer's name. In this string, the characters `*` and `?` are wildcards that match any characters or character, respectively, in the layer's name. So, the string `select` will match `select`, `selection`, `select-surface` and so on.
- **New**: Creates a new layer. If no name is given, the layer will be called `Layer#`. The new layer will be used as the default layer until the end of the session or until another layer is selected as the default.
- **Delete**: Deletes a layer. A layer can only be deleted if it has no entities in it.
- **Rename**: Changes the name of a layer.
- **alphabetic**: If this option is set, layers are sorted alphabetically (by name).
- **To back**: Sends entities to the back of its layer. When an entity is "at the back" it is not visible and cannot be selected, moved, copied or deleted. To bring the entities of a layer back "to the front" again, select a layer and choose the `Bring to front` option from the `To back` menu. There is another option, `Bring All to front`, which brings all entities of all layers to the front. If the `Opposite` option is flagged, entities which are **not** selected will go to the back.
- **Send To**: Moves entities to a new layer. No new entities are created. They are only moved from one layer to another. When in **geometry** mode and selecting point, line, surface, volume or all, the all option can only be used to select entities in the graphical window and to change all the entities in the dynamic box to a new layer. When in **mesh** mode, it is possible to choose `nodes`, `elements` or `all`. With the special option `Also lower entities`, entities that are lower entities of the selected ones can also be sent to a new layer. **Example**: If this flag is set and one surface is selected, its lines and its points are also sent to the layer.
- **Close**: Closes the **Layers** window.

Important: Mesh generation does not depend on the state of the layers when the generation is performed (see [Layers](#)). All layers are meshed and every node and element will be assigned to the layer where the original geometrical entity was. The only exception to this rule is in the case of a frozen layer if the `No mesh frozen layers` option is selected (see [Preferences](#)→`Meshing`).

Tools

Menu



In this menu there are some options to change the appearance of windows and some tools related to different aspects of GiD.

Toolbars

Menu



In this menu you can customize GiD toolbars. They can be displayed inside another window, can appear independently in their own dialog box, or can be hidden.

When you select this option, the following window appears:



Toolbars layout window

Within this window you can choose where on the screen the toolbar is to be displayed - inside, outside, top left, bottom right, etc. - or you can switch it off. The GiD toolbars are:

- **Right buttons:** These are the buttons that usually appear on the right-hand side of the window. They can perform most of the functions available in the program.
- **Command line:** The bar where commands can be written using the keyboard. It usually appears at the bottom of the screen.
- **Up menu:** The pull-down menus located above the graphical window.
- **Geometry & View bar and Standard bar:** These are icon toolbars used to perform certain operations. Click the middle or right mouse button over an icon to get help.

- `Macros toolbar`: This is an icon toolbar where default and user macros are placed.

Save window configuration

Menu



It is possible to save a window configuration to a file. Then, if GiD is opened again with the `-c` option (see [INVOKING GiD](#)) and the file in question, the windows are opened in the same place and are the same size as before.

Move screen objects

Menu



This option lets you move screen objects (objects that do not belong to the model) such as axes, comments, legends, etc.

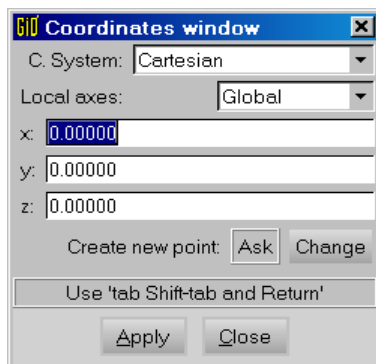
Coordinates window

Menu



This option opens a window used to enter points (see [Point definition](#)). It can be used in any place where it is possible to enter a point.

To accept one point in this window, click `Apply` or press the `Return` key.



Entering coordinates window

The Coordinate system option lets you select between:

- **Cartesian:** Cartesian coordinate system.
- **Cylindrical:** Enter the radius in the XY plane, the anti-clockwise angle in the XY plane from the X-axis (theta), and the Z-coordinate.
- **Spherical:** Enter the radius, the anti-clockwise angle in the XY plane from the X-axis (theta), and the angle with the Z-axis (phi).

The Local axes option lets you choose:

- **Global:** Global axes.
- **Relative center:** Define a center of coordinates. All new points created with this window will be related to this center. In the window where the relative center is entered, you can select a point from the graphical window with the **Pick** button.
- **Relative last:** The last point entered is the relative center of coordinates.
- **Define new...:** This lets you define a new local axes. Once defined, you can select it.
- **Any local axes:** All local axes defined with this option (see [Local axes](#)), will be listed here. Points entered will be related to these axes.

Create new point shows the current way of entering points. The **Change** button opens the preferences window (see [Preferences](#)→General) where it can be changed.

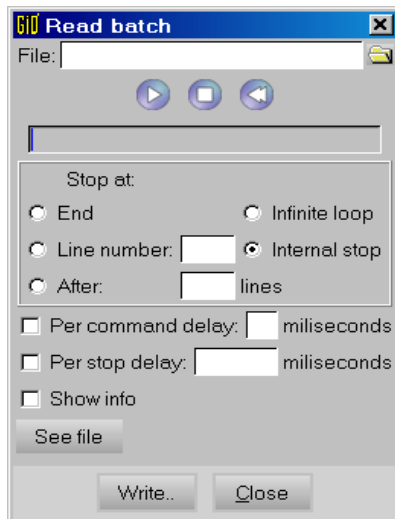
The **Pick** button allows you to select a point from the window which is then inserted in the point fields. From here it can be edited and used.

Read batch window

Menu



A batch file can be read to execute some functions (see [Batch file](#)) or to create an animated view of these operations. This latter case can be performed with the `Read batch` window.



Read batch window

Once a file is selected, it is possible to pause it at certain points to highlight interesting parts and execute it interactively. To allow all the movements (rotations and so on) be executed in the same way as originally, the `Write rotations in batch:` option must be flagged in the preferences window (see [Preferences](#)→General) when creating the batch.

There are several ways to halt the batch while it is running. One of them is to include stops in the show file section with the `Mark break` button, and selecting `Internal stop`. These marks can be saved in the batch file by clicking the `Write...` button.

The `Show info` option lets the program write all the usual messages in the GiD messages window.

Comments

Menu



Comments can be added to images created with GiD by using this command. Click `Apply` to display the text on the screen as it will appear when printed - either to a file (see [Print to file](#)) or otherwise; comments can be changed at any time by clicking the `Comments` button in the `Utilities` menu (either pull-down or **Right buttons** menus).

Animate Controls

Menu



Animate controls window

This window lets you create animations while using GiD. Any of these formats can be selected: MPEG, AVI True Color, AVI 15bpp (reduced number of colors: 32768) and GIF. AVI with MJPEG compression is also supported. After giving a name and a delay time between frames (for 20 frames per second, a delay of $1/20 = 50$ ms. should be entered) the process can be started by clicking on the `start` button (film roll and arrow icon); the green LED will change to red.

Simply click on the film roll and arrow button to add frame by frame the pictures you want to include in the animation. The **Save redraws** option tells GiD to save every redraw automatically to the animation file, so no user intervention is needed to store frames.

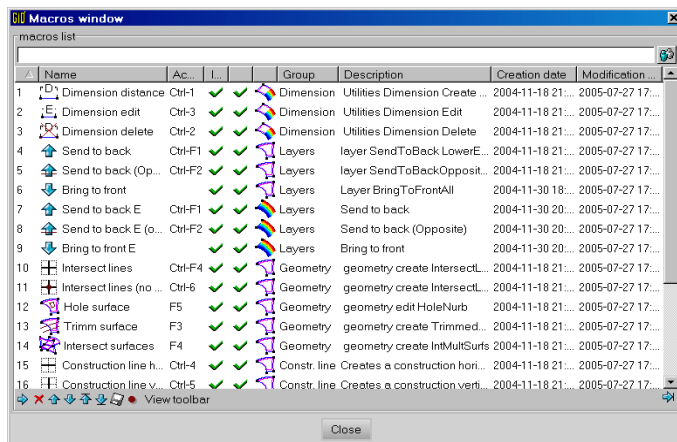
The animation can be finished at any time by clicking the `end` button (closed film roll icon); the red LED will change to green.

Note: To avoid problems when trying to view an MPEG format animation in Microsoft Windows, it is strongly recommended that you use the `Default` menu to select a 'standard' size and click the `Resize` button. The graphical window will change to this 'standard' size. When you have finished the animation, simply select `Default` on the menu and click the `Resize` button and the previous size will be restored.

Note: AVI files with MJPEG compression can be viewed with xanim on Linux and Unix machines, and with DirectX 8.x installed on Microsoft Windows machines.

Macros

Menu



Window for the creation and recording of macros

Windows 'Macros' allow you to create sequences of commands and give them a name. This group of commands can also be recorded from one execution set inside the program.

It is possible to assign a keyboard shortcut to a given macro.

Note: Macros are considered as a user preference and not related to the active model. So, to transfer a set of macros from one user to another it is necessary to check the files: gid.ini or .gidDefaults.

Selection window

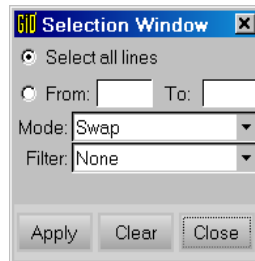
Menu



Mouse menu



For those functions where some entities are to be selected (creation of a surface or a volume, copying entities, etc.), the **Selection window** can be used.



Selection window

Note: The `Selection window` option is only available in the mouse menu during the selection process.

The Selection window has the following options:

- `Select all`: If this option is chosen all entities are selected. If a filter is selected, it is applied to all entities.

- **From, To:** This option lets you select a range of entities. If a filter is selected, it is only applied to that range of entities. To see the ID numbers of entities, use the `Label` command (see [Label](#)).
- **Mode:** There are three selection modes:
 - **Swap:** If you select an entity that is already selected, the entity is deselected, and vice versa.
 - **Add:** In this mode it is impossible to deselect an entity. Only new entities are added to the current selection.
 - **Remove:** In this mode it is impossible to add entities to the selection. This mode is used to remove entities from the current selection.
- **Filter:** If a filter is selected, only the entities that satisfy the filter criteria will be selected. This menu changes depending on what type of entity is being selected:
 - **Points**
 - ◆ **Higher entities:** You are asked for a value, and all points with this Higher Entity number are selected (Higher Entity number is the number of lines that a point belongs to).
 - ◆ **Label:** Selects all points where a Label is shown or not (On or Off).
 - ◆ **Material:** All points with a chosen material assigned are selected.
 - ◆ **Condition:** All points with a chosen condition assigned are selected.
 - **Lines**
 - ◆ **Higher entities:** You are asked for a value, and all lines with this Higher Entity number are selected (Higher Entity number is the number of surfaces that a line belongs to).
 - ◆ **Label:** Selects all lines where a Label is shown or not (On or Off).
 - ◆ **Material:** All lines with a chosen material assigned are selected.
 - ◆ **Condition:** All lines with a chosen condition assigned are selected.
 - ◆ **Min. length:** Selects only the lines whose length is greater than the given length.
 - ◆ **Max. length:** Selects only the lines whose length is smaller than the given length.
 - ◆ **Entity type:** Selects only the lines which fit the specified type: StLine (Straight Line), ArcLine, PolyLine or NurbLine.
 - **Surfaces**
 - ◆ **Higher entities:** You are asked for a value, and all surfaces with this Higher Entity number are selected (Higher Entity number is the number of volumes that a surface belongs to).
 - ◆ **Label:** Selects all surfaces where a Label is shown or not (On or Off).

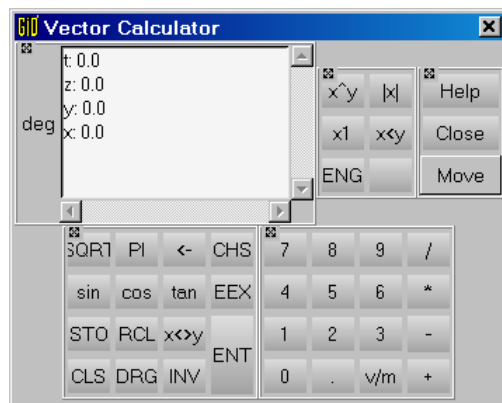
- ♦ **Material:** All surfaces with a chosen material assigned are selected.
- ♦ **Condition:** All surfaces with a chosen condition assigned are selected.
- ♦ **Entity type:** Type of surface.
- **Volumes**
 - ♦ **Label:** Selects all volumes where a Label is shown or not (On or Off).
 - ♦ **Material:** All volumes with a chosen material assigned are selected.
 - ♦ **Condition:** All volumes with a chosen condition assigned are selected.
 - ♦ **Entity type:** Type of volume.
- **Nodes**
 - ♦ **Label:** Selects all nodes where a Label is shown or not (On or Off).
 - ♦ **Condition:** All nodes with a chosen condition assigned are selected.
- **Elements**
 - ♦ **Label:** Selects all elements where a Label is shown or not (On or Off).
 - ♦ **Material:** All elements with a chosen material assigned are selected.
 - ♦ **Condition:** All elements with a chosen condition assigned are selected.
 - ♦ **Min Angle:** Selects only the elements with an angle greater than the figure specified in degrees (see [Mesh quality](#)).
 - ♦ **Max Angle:** Selects only the elements with an angle smaller than the figure specified in degrees (see [Mesh quality](#)).
 - ♦ **Min Edge:** Sets the minimum edge length accepted (see [Mesh quality](#)).
 - ♦ **Max Edge:** Sets the maximum edge length accepted (see [Mesh quality](#)).
 - ♦ **Shape quality:** Sets the minimum shape quality accepted (see [Mesh quality](#)).
 - ♦ **Minimum Jacobian:** Sets the minimum jacobian accepted (see [Mesh quality](#)).
 - ♦ **Entity type:** Type of element (see [Element type](#)).
- **Clear:** Clears the current selection.

Calculator

Menu



This option opens a scalar and vector calculator. Clicking the help button open a dedicated help section.

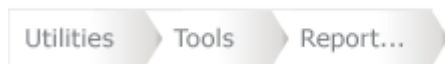


Vector calculator

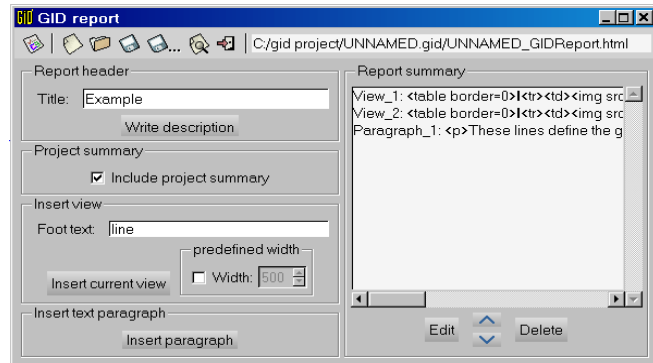
It is possible to transfer scalar and vector points and distances between the calculator and the main graphical display.

Report

Menu



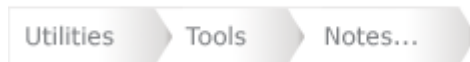
This option creates a report in html format to which figures, comments, titles, etc. can be added. This report can be saved to a file and then reloaded in another GiD session where it can be edited or have more information added to it.



Report window

Notes

Menu



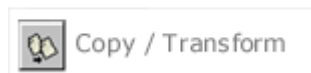
This is a simple text editor for writing small notes regarding the model. The content is saved in a file named `ModelName.txt`, with `utf-8` encoding.

Copy

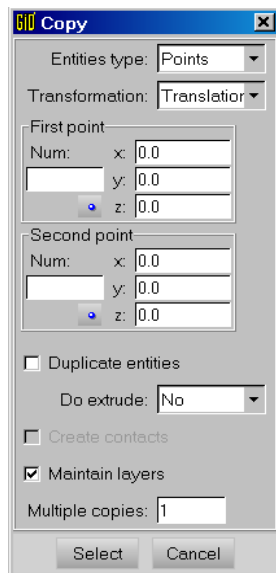
Menu



Toolbar



`Copy` is a general function that allows you to select a group of entities and copy them with a movement operation performed, either `translation`, `rotation`, `mirror`, `scale` or `offset`.



Copy window

Select the type of entities to copy. In **geometry** mode choose between point, line, surface and volume; in **mesh** mode choose between nodes and elements. All the lower entities belonging to the selected one will automatically be computed. Next, the type of movement needs to be chosen and its parameters defined. The options are:

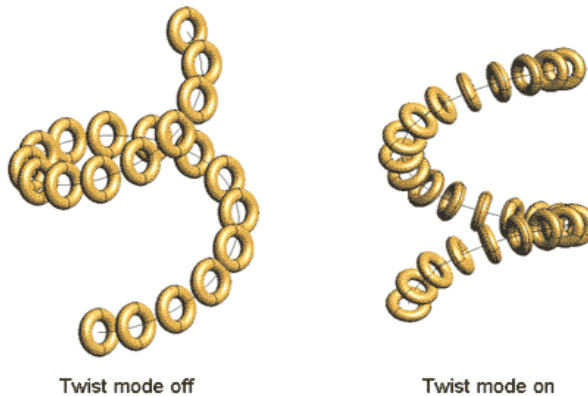
- **Rotation:** It is necessary to enter two points in 3D, or one point in 2D. These two points define the axis of rotation and its orientation. In 2D, the axis goes from the defined point towards Z-positive. Enter the angle of rotation in degrees; it can be positive or negative. The direction is defined by the right hand rule. In 2D, the direction is counter-clockwise.
- **Translation:** Two points are defined. Relative movements can be obtained by defining the first point as 0,0,0 and considering the second point as the translation vector (see [Point definition](#)).
- **Mirror:** Three points are defined (they cannot be in a line). These points form the mirror plane. In 2D, the mirror line is defined by two points.
- **Scale:** This is defined by a center and a point. Every coordinate of the point is the scale factor for every X-,Y- and Z-axis. A scale factor greater than one increases the size, while a scale factor less than one decreases the size. It may also be negative, changing the sign of the corresponding coordinates.
- **Offset:** This is defined by one positive or negative scalar magnitude. Each entity will be moved in the direction of its normal, by the magnitude given. In 2D, the normal is

considered to lie in the plane $z=0$. This option works either for lines, surfaces or mesh elements.

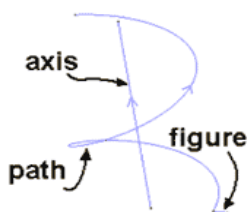
- **Sweep:** This is an option for copying figures along a line (path line). You can simply copy the figures (and then specify a number of copies) or extrude them. Both methods have basically the same options.



The 'end scale' factor determines how the figure is scaled along the path curve (the scale value starts at 1.0 and varies in a linear way until the 'end scale' value). This scale value must be greater than zero. The extrusion (and the copy) always starts on the start point of the path line. If you select 'twist mode' as on, then the relative position of the figure to copy with respect to the start of the path line is conserved along the line.

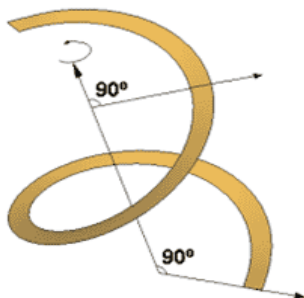


In a non-planar curve, there is not only curvature, but also torsion. This may cause some unexpected behavior, because the figure also has a rotation along the tangent direction of the path:



Natural twist option

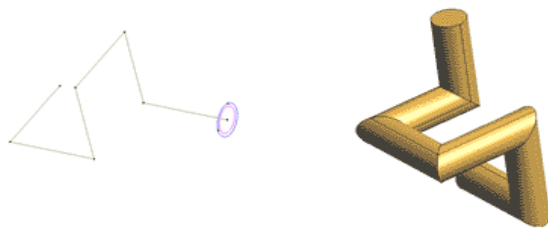
Next the 'XY plane' option is used, which makes certain that the initial vector will remain in this plane during extrusion. (In this example, the axis is Z).



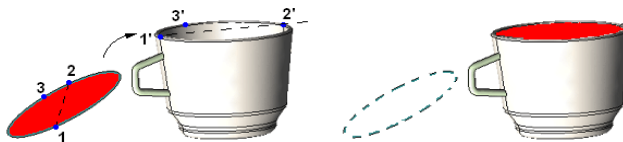
The option 'ByDer2' uses the second derivate of the path line as a normal to the plane in which that initial vector has to remain. A rotation along the path line can be forced using the 'Angle' parameter (degrees):



Finally, if the path line is a polyline, then the extrusion will be divided into several parts:



- **Align:** This is an option to move figures from a generic position to the desired one. Set the new location specifying three source points and three destination points: the first point defines the exact destination of the source point; the second point is not necessarily the exact destination, but a point over the destination straight line; the third point is not necessarily the exact destination, but a point over the destination plane



Other available options are:

- **Extrude:** This option can be set to either lines, surfaces or volumes. When a movement is selected, the copy is made and lines connecting the old and new points are created. These lines will either be straight lines or arcs depending on the movement type. If extrude surface is chosen, NURBS surfaces connecting old and new lines will also be created. If Volumes is chosen, the volume contained between old and new surfaces is also created. This option is not allowed when copying volumes.
- **Mcopy:** By selecting this option and giving the number of repetitions, the selected operation is performed this number of times. This option is not available for mirror.
- **Create contacts:** Creates separated contact volumes (see [Contact volume creation](#)) for every copied surface. This option is only available when copying surfaces.
- **Duplicate entities:** If this option is not set and after the copy operation an entity occupies the same position as an existing one that does not belong to a frozen layer, both entities are converted into one.
- **Maintain layers:** If this option is not set the new entities created will be placed in the layer to use; otherwise, the new entities are copied to the same layers as their originals.

Note: Entities belonging to a frozen layer (see [Layers](#)), are not checked when sharing old entities.

Move

Menu



This command works like `Copy` but moves the entities instead of copying them. The program automatically checks to see if any of the entities must be copied instead of being moved (for example, if they also belong to other higher level entities) and performs the appropriate operation.

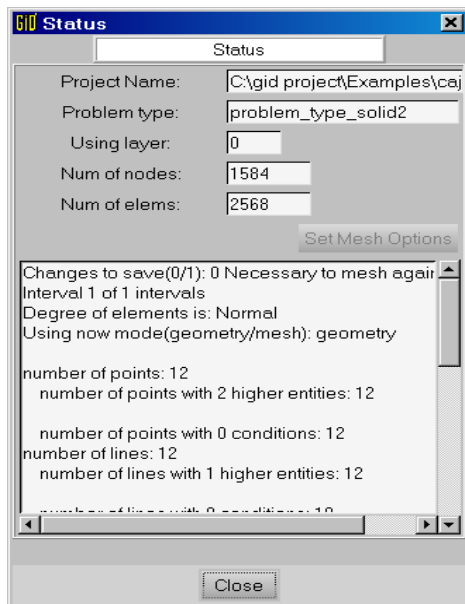
Options like `Extrude`, `Multiple copy` and `Create contacts` are disabled for movements.

Status

Menu



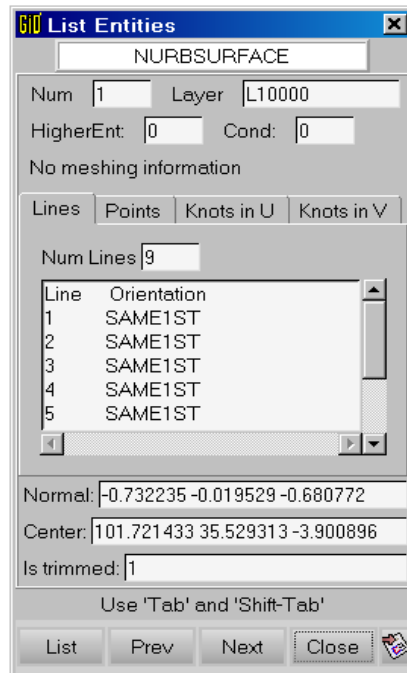
The `Status` option gives useful information about general project data.



Project status window

List


Menu



List entities window

The `List` command gives information about the selected entities. This information is read-only.

If the `Mass` option is checked, information about physical properties is given: lengths of lines, center of mass, areas of surfaces, volumes of solids. It works for both the geometry and the mesh.

All this information can be sent to the active report (see [Report](#)) by using the  button.

Renumber

Menu



When creating new entities, the label of the new entity will be the lowest number greater than zero that still does not exist for this entity type. If an entity is deleted, a gap is left in the labels list. This gap will be filled with a new entity, but it is also possible to renumber the geometry, changing the previous entity labels. There are no problems with materials and conditions applied to entities.

In **geometry** mode, the renumbered entities are the geometrical ones. In **mesh** mode, the renumbered entities are the mesh ones. In this case, renumbering not only fills the gaps in the labels list but also changes the node numbers so as to minimize the difference in node numbers within each element. This can be useful when the calculating module uses band or skyline storage methods.

Note: When generating a mesh using GiD it is not necessary to use this command because it is automatically applied when generating the mesh.

Id

Menu



This command gives the label and coordinates of an existing or new point. Different options for getting information about an existing point are available in the **Contextual** menu.

Signal

Menu



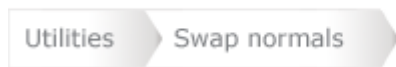
With this option you can select one entity (a point, line, surface or volume), and a pair of crossed red lines will signal the center of the entity in the graphical window.

They must be existing entities, except the special case of points or nodes, where they can be existing or defined with any of the usual methods.

The `Superpose Lines` option (in the **Contextual** menu) is useful when in render mode. Depending whether it is set or not, the crossed red lines will either be in front of the object or partially hidden by the model.

Swap normals

Menu



This command can be applied to lines or surfaces (in geometry mode) or to elements (in mesh mode). A selection is made (see [Entity selection](#)) and the orientation of the selected entities is inverted.

Viewing commands (`zoom`, `rotation`, etc.) can be applied and the normals will remain on the screen.

When this command is applied to surfaces or mesh elements, you can choose one of the following options:

- `Select`: Inverts the direction of the normals of the selected surfaces.
- `Make coherent`: Inverts the normals of all selected surfaces or elements so that all of them have their normal in the same direction as adjacent entities. This direction is arbitrary, but is common to all adjacent entities.
- `Select by normal`: You are prompted to enter a vector and all selected entities' normals are swapped so as to have their normal in the same direction (dot product positive) as the given vector.

Note: With the `Color` option set (available in the **Contextual** mouse menu), all surfaces or elements are drawn in filled color. Their front side will be drawn in their regular color, and their back face in yellow.

Note: Volumes are correctly oriented by GiD, regardless of their surface orientation.

Distance

Menu

Utilities Distance

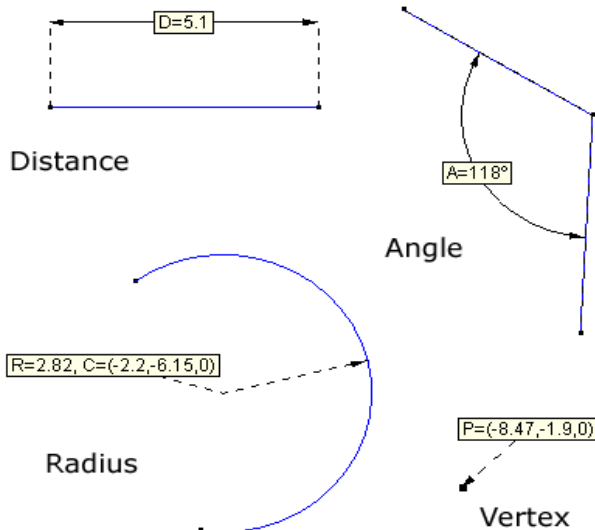
The **Distance** command gives the distance between two existing or new points.

Dimensions

Menu

Utilities Dimension

With the **dimensions** option it is possible to add textual information to your model. This information can be moved to a different layer or deleted.



The following options are available:

Menu

Utilities Dimension Create

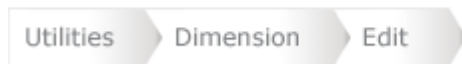
- **Vertex:** Shows the coordinates of a vertex. Click over an existing point and then click where you want the dimension to be written.
- **Distance:** Shows the distance between two points. You have to select two points and then click where you want the dimension to be written.
- **Angle:** Shows the size of an angle in degrees. You have to select three points and then click where you want the dimension to be written.
- **Radius:** Shows the center and radius of an arc. You have to select an existing arc and then click where you want the dimension to be written.
- **Text:** Shows a text string defined by the user. Enter the text and click where you want the text to be written.

Menu



Deletes a dimension. Select the dimension you want to delete and press `escape`.

Menu



Select the dimension you want to edit, change the text, and click `OK`.

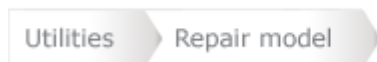
Menu



This option lets you change the appearance of a dimension; a dimension can be drawn with or without a box. Choose `ShowBox On` or `Off`, select a dimension, and press `escape`.

Repair model

Menu



This option checks the coherence of the database information. Only use it if there are problems. When used, a window notifies you of any repaired items and may give some warnings about incorrect entities.

DATA

All the data that defines the problem and that is managed in the data menus, depends on the **Problem Type** and will change for every different problem type. The following help will describe the common interfaces to all the possible data.

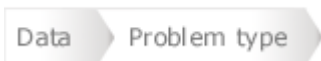
Data for a problem is defined by the following parameters: conditions (see [Conditions](#)), materials properties (see [Materials](#)), units (see [Data units](#)), problem data (see [Problem data](#)), and intervals data (see [Interval data](#)) that define the problem. The conditions and materials have to be assigned to geometrical entities. It is also possible to define new reference systems (see [Local axes](#)).

The various windows will differ according to the specifications of the problem (thermoelastic, impact, metal-forming, etc.), the different types of elements (2D or 3D, beams, shells, plates, etc.) and also the different requirements and specifications of the particular solver.

All the commands and facilities that are explained in this manual are generally available for the different solvers. However, there may be some commands and facilities that are only available to some of them and therefore some displays may look slightly different from those explained in these pages.

Problem type

Menu



This option lets you choose between all installed **Problem Types**. When selecting a new problem type, all information about materials, conditions and other data parameters that were already selected or defined is lost.

Note: When defining a new problem type which is not already installed, it must be selected by other means. One possibility is to select `Problem type`→`Load...`. Another possibility is to select `Data`→`Defaults`→`ProblemType` the **Right buttons** menu, or enter it in the command line.

A problem type is considered to be installed when it has been copied to the GiD Problem Types directory or to another subdirectory within this.

Note: Instead of copying the problem type to the Problem Types directory, it is also possible to create a Windows shell link (a direct access), or a UNIX link, pointing to the real location. This option is particularly interesting for developers, as it avoids duplicating the code.

Transform problem type

Menu



This option can be found inside the `Problem Type` menu and is useful for updating a model from an old problem type to a newer one that is similar to the first.

When converting, it tries to maintain all the conditions and materials assigned to the geometry or mesh. Also, it tries to maintain the rest of the data.

It will typically be used when a problem type has been updated and it is necessary to reuse a model defined with the old version.

Internet Retrieve

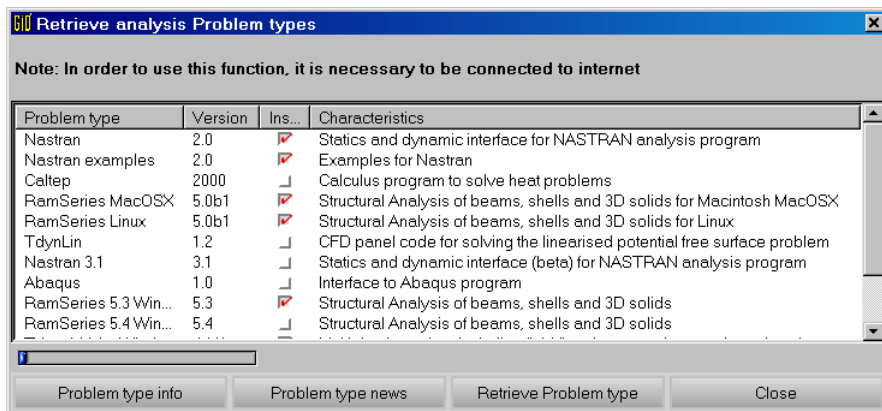
Menu



Internet retrieve window

With this option it is possible to download new problem types or update existing ones.

Note: You need to be connected to the internet to use this option.



Select a list item and use the **Problem type info** and **Problem type news** buttons to get information about them.

Use the **Retrieve Problem type** button to download the selected problem type. The problem type will be installed inside the Problem Types directory.

Load

Menu



The **Load...** option allows you to load a previously installed problem type from the current or another directory. This possibility is useful when developing a new problem type which cannot be installed until it is finished, or if the developer does not have permission to write to the Problem Types directory.

Unload

Menu



This option unloads the problem type currently in use.

Sometimes it is easier to work with a model that does not have an associated problem type. It is also useful for sending a model to another user who does not have the problem type in question.

Debugger

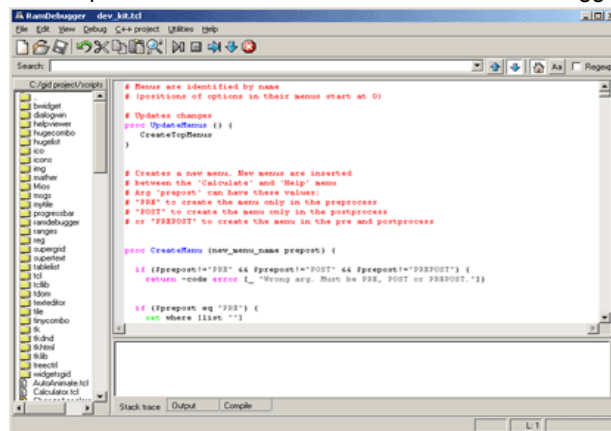
Menu



This tool is a graphical debugger for the scripting language **Tcl/Tk**.

This debugger has additional capabilities such as:

- Editing the code. It is possible to edit the code inside its own editor and resend the new code without closing the debugged program.
- The Tcl/Tk source code is colored and supports automatic indentation.
- When one source code line stops the debugger, it is possible to view all the variables and expression values, as well as change them.
- It has additional options to measure execution times in the debugged program.



Tcl/Tk debugger window

It is possible to use Tcl/Tk to enhance a problem type (see [TCL/TK EXTENSION](#)).

Tcl code can be invoked inside GiD from a problem type (`.tcl`, `.bas`, `.bat` or `.xml` files), from user-defined macros, from a batch file, etc.

More information can be found in this debugger's own help section.

Conditions

Menu



Conditions are all the properties of a problem (except materials) that can be assigned to an entity.

An example would be the boundary forces and displacement constraints in a solid mechanic analysis or initial velocities in a CFD analysis. Information about contact between master-slave nodes can also be considered as conditions.

Caution: Once a mesh has been generated, any changes made to the condition assignments require you to regenerate the mesh in order to transfer these new conditions. If this new generation has not been performed, GiD will warn you when the data for the analysis is being written.

Assign condition

A condition is assigned to geometric entities or layers that have the given field values.

If you are using the `AssignCond` command in the **Right buttons** menu, the `Change` option allows you to define the field values. Do not forget to change these values before assigning. Selecting `DeleteAll` erases all the entities that have this particular condition assigned.

Conditions can be assigned both to the geometry and to the mesh, but it is advisable to assign them to the geometry because in this way the conditions will then be transferred automatically to the mesh. If assigned to the mesh, any re-meshing will cause the conditions to be lost.

The `UnAssign` command inside `AssignCond` allows a condition to be unassigned, either from all entities, or just from those that have a specified field value.

Caution: Once a mesh has been generated, any changes made to the condition assignments require the mesh to be regenerated.

Some conditions may have a behavior that depends on the type of the chosen axes. You can choose whether to use global axes or any of the local axes previously defined with the `local axes` command (see [Local axes](#)), or, alternatively, another kind of automatic local axes calculations. In this second case, different axes are created according to the adopted criteria of tangency and orthogonality with the geometry.

Draw condition

The `DrawAll` option draws all the conditions assigned to all the entities. This means that a graphical symbol or condition name will be drawn over every entity that has this condition.

If one particular condition is selected, you can choose `Draw` for just one field. `Draw` is like `DrawAll`, but for one particular condition only. If one field is chosen, the value of this field is written over all the entities that have this condition assigned.

If the condition has any field which refers to the type of axes, the latter can be visualized by means of `Draw local axes`.

The `Draw colors` option draws groups of entities with different colors depending on the values assigned to them for this condition.

You can apply all graphical functions (`zoom`, `rotate`, ...) and all the condition symbols are maintained in active mode in the graphical window until you select `escape` or click `Finish` in the conditions window.

Unassign condition

When using the `UnAssign` window, you can choose between several possibilities:

- Unassign one condition from some selected entities.
- Unassign one condition from all the entities that may have this assigned.
- Unassign all conditions of a book from all the entities that may have them assigned.
- Unassign all conditions from all the entities that may have them assigned.

When using the command line or the **Right buttons** menu, `UnAssign` works as the fourth option above, i.e. all conditions. To unassign only one condition, use the `DeleteAll` command (see [Assign condition](#)).

Materials

Menu



For any problem that needs a definition of materials, there is a database of existing materials that can be assigned to entities. You can also create new materials derived from existing ones and assign them as well.

Caution: Once a mesh has been generated, any changes made to the assigned materials require you to regenerate the mesh or reassign these materials to the mesh directly. If only the material definition is changed (i.e. some field value) then is not necessary to re-mesh again.

Assign material

This option is used for assigning a material to some selected entities.

When working in **geometry** mode, the kind of entity to which you wish to assign a material must be selected, i.e. point, line, surface or volume; when working in **mesh** mode, you select directly the elements to which the material is to be assigned.

Note: A material cannot be applied over nodes; a material only assigned to points will be transferred to 1-node elements if generated, but not to nodes themselves.

If assigning from the command line, the `UnAssignMat` option erases all the assignments of a particular material.

Caution: Once a mesh has been generated, any changes made to the assigned materials require you to regenerate the mesh or reassign these materials directly to the mesh.

Draw material

This option draws a color indicating the selected material for all the entities that have it assigned. It is possible to draw just one material type or, alternatively, to draw all materials. To select just some of them use `a : b` and all material numbers that lie between `a` and `b` will be drawn.

Unassign material

When using the `UnAssign` window, you are presented with several possibilities:

- Unassign one material from some selected entities.
- Unassign one material from all the entities that may have this assigned.
- Unassign all materials from all the entities that may have them assigned.

When using the command line, `UnAssign` works as the third option here, i.e. all materials. For only one material, use `UnAssignMat` (see [Assign material](#)).

New material

When using the `NewMaterial` command, a new material is created taking an existing one as a base material. This means that the new material will have the same fields as the base one. Following this, all the new values for the fields can be entered in the command line.

It is also possible to redefine an existing material.

To create a new material or redefine an old one using the `materials` window, write a new name or an existing one and change some of the properties. Then click `Accept`.

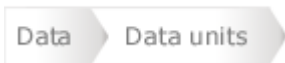
Exchange material

It is possible to import and export materials between the model database and an external one. Typically, one centralized database of materials is maintained and every new model gets its properties from there.

Note: If you wish to exchange materials with the problem type database, it is necessary to check that you have permission to read/write in that directory.

Data units

Menu



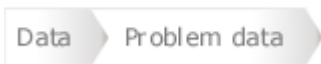
`Data units` refer to the units defined in the problem. This option only appears if the problem type loaded has units defined.

You have to declare the length units of the current model and the unit system to be used when writing coordinates and data properties in the calculation file (values will be converted from the current units to the selected system units).

It is possible to set a user-defined unit system, but this feature can be disabled (see [Unit System file \(.uni\)](#)).

Problem data

Menu



`Problem data` refers to all the data that is associated generally with the problem. This means that it is not related to a geometrical entity and it does not change for every interval of the analysis.

It can be entered with the `ProblemData` command in the **Right buttons** menu or in the `Problem Data` window.

If entered in a window, the data is not accepted until the you click the `Accept` button.

This data can be entered before or after meshing.

Intervals

Menu



`Intervals` are a way of separating information into several groups; information for every group can also be duplicated, if desired. When a new interval is defined, you can choose whether or not to copy all the new information about conditions assigned to entities.

Therefore, the correct way to work is to define all the conditions first and afterwards create the new intervals.

The options are:

- **New:** You can define as many intervals as you wish using this command. When creating a new one, you can choose whether or not to copy the assigned conditions. To copy them, conditions must already have been assigned (see [Conditions](#)).
- **Current:** Chooses the current interval to use. Any subsequent conditions or interval data (see [Interval data](#)) will be considered inside this interval.
- **Delete:** Deletes one existing interval and all its related data.

These groups, or intervals, can be used to change some conditions or information, like the increment factor in an incremental analysis. They can also be useful in order to define load states for the same geometry.

Interval data

This is the information that is specific to each individual interval (see [Intervals](#)).

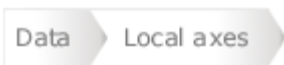
It can be entered with the `IntervalData` command or in the `Interval Data` window.

If entered in the window, the data is not accepted until you click the `Accept` button.

This data can be entered before or after meshing.

Local axes

Menu



With this option, GiD lets you define new coordinate reference systems. They can be written not only using cartesian reference systems, but also with reference to Euler angles. All user-defined systems are automatically calculated and can be visualized one by one or all together.

There are several ways to define new local axes:

- **3 Points XZ:** Enter three points that corresponds to the origin, the X-direction and the Z-direction. The origin and the last introduced point define the Z-axis, whereas the second point indicates the side of the XZ plane where the point lies.

- `X and angle`: Enter two points and one angle. The first point is the center, the second point indicates the X-axis and the angle indicates the position of the Y- and Z-axes. In the graphical window it is possible to set this angle by moving the mouse. It also indicates where the origin of the angle is. The angle can be entered either by clicking the mouse or by entering the exact value in degrees.

When defining local axes, the definition mode is done through three points `3PointsXZ`.

Local axes can be used later when creating a point (see [Point creation](#)) or in some conditions that have a field related to these axes (see [Conditions](#)).

MESH

Generating a mesh is the process by which a finite element mesh is calculated from the geometry definition. This mesh will be used for the FEM analysis at a later stage. Conditions (see [Conditions](#)) and materials (see [Materials](#)) assigned to geometric entities will be transferred to the nodes and elements of the new mesh.

What is meshed and how it is meshed is controlled by some default options which can be changed with the commands described later.

The generation does not depend on whether layers are ON or OFF at the moment of generation (see [Layers](#)), but frozen layers are not meshed if the preference `NoMeshFrozenLayer` is selected (see [Preferences](#)→Meshing). Every node and element will be assigned to the layer in which the original geometrical entity was defined.

The defaults are:

- An entity is meshed if does not belong to a higher level entity.
- A line mesh is made of two-noded elements and a surface mesh is made of non-structured triangular elements. The default for Structured meshes is quadrilateral elements. Unstructured Volume Meshes are composed of non-structured tetrahedral elements, Structured Volume Meshes are composed of hexahedra and Semi-Structured Volume Meshes are composed of prisms.

All these default elements use linear interpolations for the unknown variables.

Unstructured

Menu



Note: Size is given by the average side length (edge) of the corresponding mesh element.

Assign sizes to points, lines, surfaces or volumes:

It is possible to assign different sizes to different entities of the mesh. This means that in the vicinity of these entities, the generated elements will be approximately of that size. All the entities that do not have an assigned size when meshing take the default one. Points do not take any size if none is given.

Assigning the size 0.0 to an entity is the same as setting the default size. The transition between different sizes is controlled by a parameter in preferences (see [Preferences](#)→Meshing).

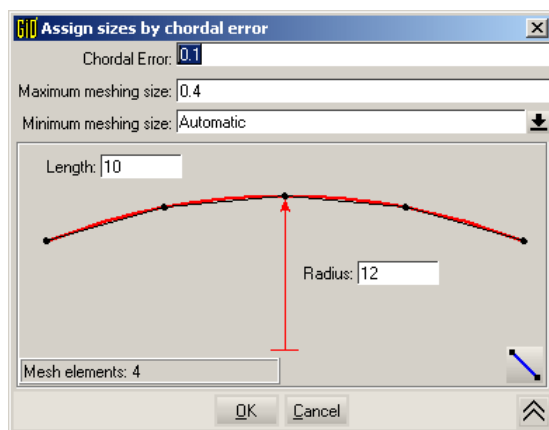
Assign sizes by chordal error:

Menu



The **By chordal error** option asks for a **chordal error** (the maximum distance between the generated element and the real geometry) and also minimum and maximum size limits. GiD assigns the corresponding sizes to all the entities to satisfy this condition. It will only change the current sizes if the new one is smaller than the one defined previously. In structured surfaces, stretching is permitted. This means that if necessary, elements can have very different sizes in the two principal directions.

Note: Entities are assigned a size between the minimum and maximum; however, when generating the mesh, GiD may adapt the size of the elements if necessary, sometimes exceeding the minimum and maximum limits.

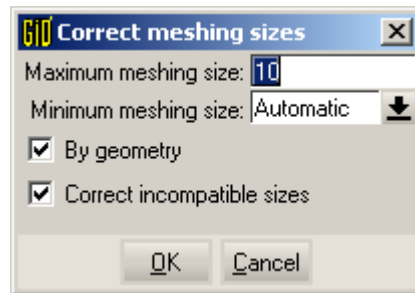


Assign sizes by chordal error window

In this window you can view the effect of different chordal error values on the number of elements that will be created over a line. Any line of the geometry can be picked and be used in this window.

Correct sizes:**Menu**

When the `Correct sizes` option is selected, a window appears. In this window it is possible to enter a minimum and a maximum mesh size.



Correct meshing sizes window

If the `By geometry` option is activated, sizes are assigned to all the entities depending on the shape of the geometry. This means that smaller surfaces will have smaller elements.

If the `Correct incompatible sizes` option is activated, some sizes are reduced to ensure that the transitions between sizes in close entities are not too fast.

It will only change the existing sizes if the new size is smaller than that previously defined.

Note: Applying the last two options with default values is the same as setting the Automatic correct sizes preference to Normal in the preferences window (see [Preferences](#)).

Note: When meshing a difficult volume, instead of trying to adjust element sizes and geometry detail, it may be useful to use the `By geometry` and `Correct incompatible sizes` options, setting the `Maximum meshing size` as equal to the default size for meshing, and setting the `Minimum meshing size` to reflect the details (10 times smaller, for example). Sometimes it is also necessary to assign sizes by chordal error.

Assign sizes by background mesh:

Menu



With this option it is possible to assign sizes using a background mesh of triangles or tetrahedra. When this option is selected GiD asks for a file; that file must contain the background mesh. The background mesh must cover the whole domain, so it will usually be a previous mesh of the same model. The format of the file containing the background mesh is the following:

First line: BackgroundMesh V 1.0

Description of the mesh. The format of that mesh is described in the **Mesh read** section (see [GiD mesh](#)).

Desired sizes in the following format:

DesiredSize (Nodes or Elements)

number of size

node/element

...

Background mesh file example:

BackgroundMesh V 1.0

MESH dimension 2 ElemType Triangle Nnode 3

Coordinates

1	5.61705	4.81504	0.00000
---	---------	---------	---------

...

51	-5.64191	-1.53335	0.00000
----	----------	----------	---------

end coordinates

Elements

1	24	16	26
---	----	----	----

2	16	10	14
---	----	----	----

...

76	34	31	28
----	----	----	----

end elements

DesiredSize Elements

1	0.20000
---	---------

```

                2                0.20000
    . . .
                75                1.50000
                76                1.50000
End DesiredSize

```

Assign entities:

This option is used to assign an unstructured mesh to geometrical entities (lines, surfaces or volumes). Using this option it is not necessary to specify an unstructured size for entities; the default size will be set for them.

CAUTION: Be careful when assigning large sizes to entities close to others where a small size has been given. It may be impossible to obtain a mesh.

CAUTION: When using contact elements (see [Contact surface creation](#) and [Contact volume creation](#)), the same size must be used for contact and duplicate entities.

Structured

Menu

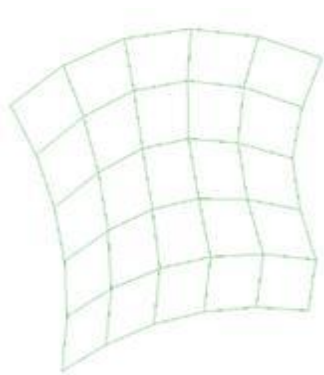


A structured mesh is defined as a mesh where all the nodes have the same number of elements around them.

The size of the elements is defined in a different way than for a non-structured mesh. In this case, the mesh is not defined by the size but by the number of elements that are required on every line. This number must be the same for all lines that are opposite each other on each surface. When meshing volumes, this definition must be the same for opposite surfaces.

To create a structured mesh, choose `Structured→Volumes/Surfaces/Lines`. After selecting `escape`, the number of elements per line is given. Later, lines can be selected and related lines (when dealing with surfaces or volumes) are added or deleted from the group. This process can be repeated as many times as necessary until all lines have a new value. Lines with no numbering given will have two elements over them. All non-selected lines will also have two elements by default.

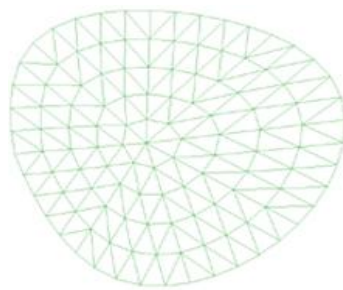
In the case of surfaces, structured meshes can be four-sided, three-sided or centered structured.



Four-sided structured surface
mesh

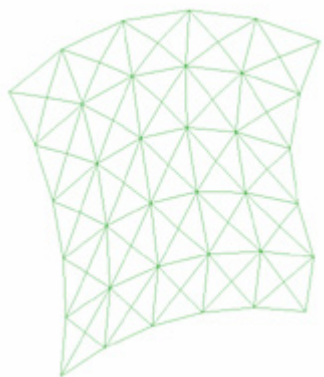


Three-sided structured surface
mesh

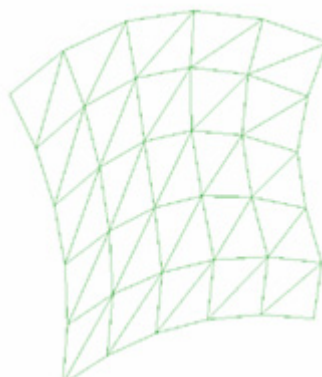


Centered structured surface
mesh

By default, the generated elements in four-sided structured meshes are quadrilaterals, but they can be triangles. In this case, triangles can be symmetrical or non-symmetrical (see [Preferences](#)→Meshing).

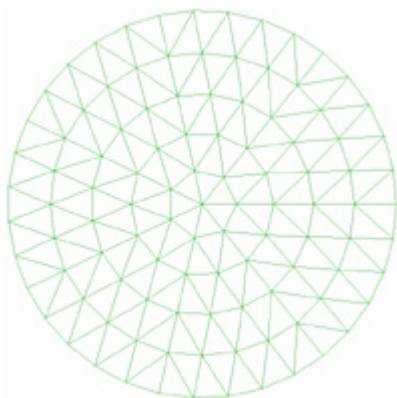


Symmetrical structured triangle mesh

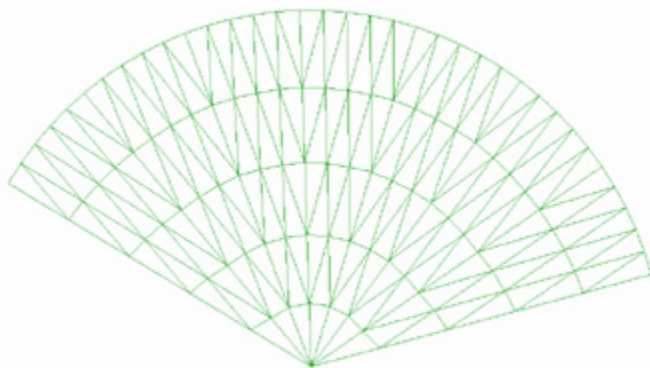


Non-symmetrical structured triangle mesh

Three-sided structured meshes (relating to a three-sided surface) and centered structured meshes can only be meshed with triangles. Centered structured meshes can be centered either at a point on the surface itself, or at a point located on a particular surface boundary; for this, use the `Set center` option. If a center is not set, GiD will locate it automatically.

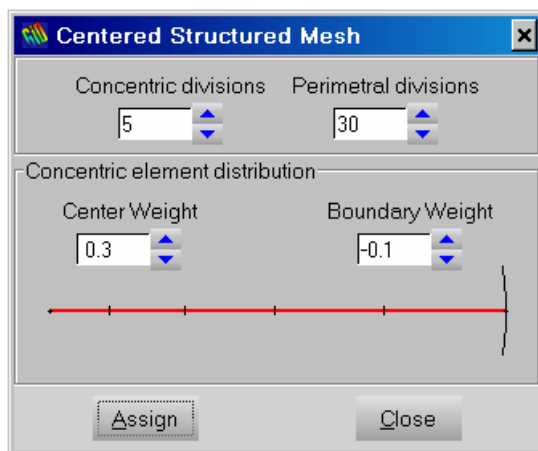


Centered structured mesh with center inside the surface



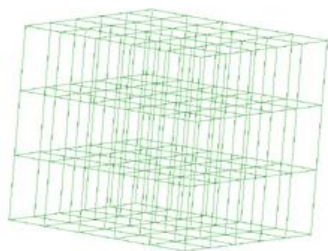
Centered structured mesh with center in the surface boundary

When selecting this kind of structured mesh (i.e. centered), the following window appears where you need to enter the number of concentric and perimetral divisions, as well as the two weights to concentrate elements in the center of the structure or in the boundary.

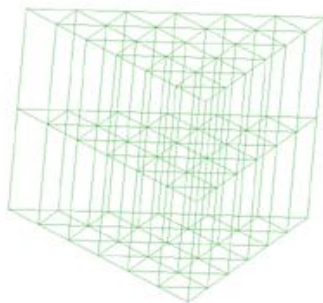


Centered structured meshing window

In the case of volumes, a structured mesh is usually six-sided, but it can also be five-sided.



Six-sided structured volume mesh

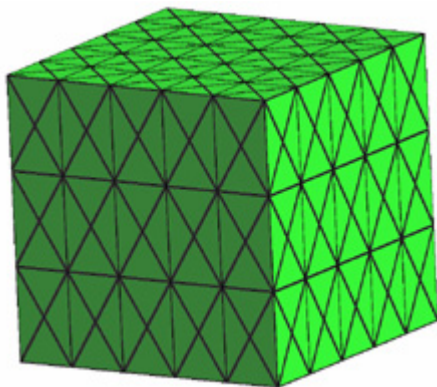


Structured volume mesh with a three-sided structured surface mesh at each end

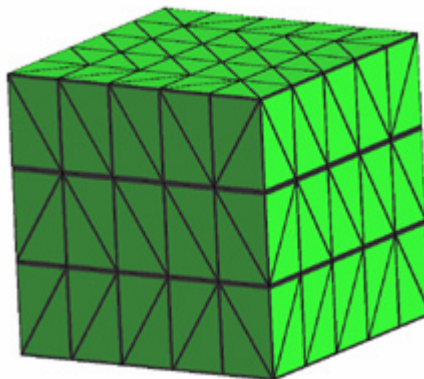


Structured volume mesh with centered structured surface mesh at each end

By default, the generated elements in six-sided structured volume meshes will be hexahedra, but they can be tetrahedra or prisms. In the case of tetrahedra, they can be symmetrical or non-symmetrical (see [Preferences](#)→Meshing).



Symmetrical structured tetrahedra mesh



Non-symmetrical structured tetrahedra mesh

Where the ends of the volume are meshed with a centered structured or three-sided surface, the default element type is prism, but you can also choose to use tetrahedra.

In the case of a six-sided structured volume mesh, volumes must have six contour surfaces.

It is possible to mix some entities with structured meshes and others with unstructured ones.

To convert a structured entity to a non-structured one, select `reset` (see [Reset mesh data](#)) or assign an unstructured mesh to it (see [Unstructured](#)).

To change the default element type see [Element type](#).

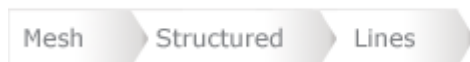
Note 1: One NURBS surface can be structured with any number of contour lines but it must have a good shape form. This means that it must have four large angles and the other angles must be small (four corners). With this criterion, the shape will be topologically similar to one quadrilateral.

Note 2: When assigning structured divisions to a line with difficult topology, GiD may need to reassign some of the divisions to make the structured mesh conformal; this will be done automatically. If it is impossible to create compatibility between surfaces, a message is displayed.

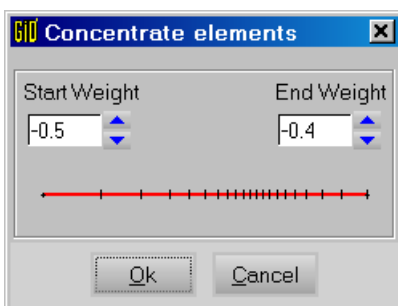
Note 3: It is possible to assign a number of structured divisions to the boundary line of a surface or volume, and then create an unstructured mesh for the surface/volume.

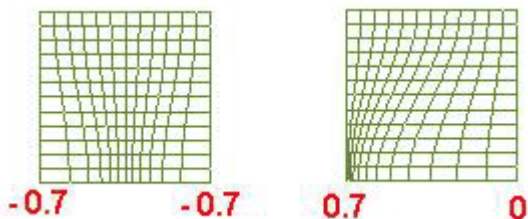
Element concentration

Menu



By default, all partitions in one structured line have the same approximate length. This command lets you select one line, which will be shown in the graphical window with an arrow indicating its direction. You then have to enter a positive or negative weight. If the weight is positive the elements will be concentrated towards the extremities of the line; if negative, the elements will be repelled.





As the magnitude of the weight increases, the difference between element sizes will be greater.

Semi-Structured

Menu

Mesh

SemiStructured

A semi-structured mesh is a mesh that is structured in only one direction of the volume and is unstructured in the other two directions. For example, in a prismatic volume, the meshing on the sides of the object could be structured, while the meshing on the surfaces at each end could be unstructured. The surfaces on both ends must be topologically equal.

Depending on the element type selected for each surface of the volume, the elements will be hexahedra, prisms or tetrahedra.

Note: When assigning divisions in the structured direction, GiD may need to reassign some number of divisions to make the structured mesh conformal with the surrounding ones; this will be done automatically. If it is impossible to create compatibility between surfaces, a message is displayed.

Set Master surface or Structured direction:

The structured direction of the prismatic volume can be set by using this option. If no structured direction is set, GiD will assign it automatically. This option makes sense in the case of volumes with multiple prismatic directions, so you can specify which one you want to use.

With semi-structured volumes (since they are prismatic) two end surfaces can be distinguished. They will have topologically identical meshes, so one of them is obtained from the other. The **master surface** is the one that is meshed first, so its mesh will determine the topology of the **slave mesh** at the opposite end. By selecting `Set→Master surface` one top surface can be forced to be the master one, and the structured direction is then set automatically.

Quadratic

Menu

The image shows a menu path starting with 'Mesh' in a grey button, followed by a right-pointing arrow, and then 'Quadratic elements' in a white button with a grey border.

The `Quadratic` option applies to all the elements of a problem. If chosen, the elements will be:

- **Linear:** 3 nodes.
- **Triangle:** 6 nodes.
- **Quadrilateral:** 8 nodes.
- **Tetrahedra:** 10 nodes.
- **Hexahedra:** 20 nodes.
- **Prisms:** 15 nodes.

The `Quadratic9` option is similar to `Quadratic`, but will generate 9-noded quadrilaterals and 27-noded hexahedra.

Element type

Menu

The image shows a menu path starting with 'Mesh' in a grey button, followed by a right-pointing arrow, and then 'Element type' in a white button with a grey border.

With this command, the type of element you wish to use is selected. It is only necessary to do this when the element type is different from the default (see [MESH](#)).

The types are as follows:

- **Default:** For surfaces and volumes. This option lets GiD assign a compatible element type to geometric entities, assigning the default ones if possible (see [MESH](#)).
- **Linear:** For lines.
- **Triangle and Quadrilateral:** For surfaces.
- **Tetrahedron, Prism and Hexahedron:** For volumes.
- **Only points:** Just for volumes. Point elements are generated.
- **Linear and Quadrilateral:** For contact surfaces.
- **Linear, Prism and Hexahedron:** For contact volumes.

By default, the elements are of minimum order: 3-noded triangle, 4-noded quadrilateral and so on. To increase the degree, use the `Quadratic` command (see [Quadratic](#)). Quadratic applies to all the elements of a problem.

- **Point:** 1 node.

Point connectivity:



- **Linear:** 2 or 3 nodes.

Line connectivities:



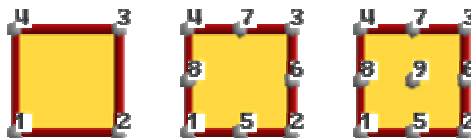
- **Triangle:** 3 or 6 nodes.

Triangle connectivities:



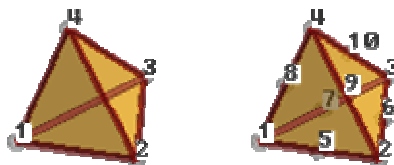
- **Quadrilateral:** 4, 8 or 9 nodes.

Quadrilateral connectivities:



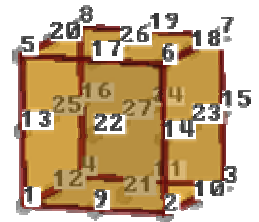
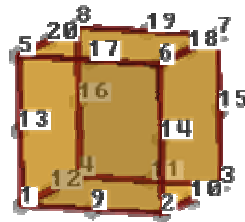
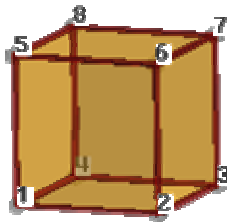
- **Tetrahedra:** 4 or 10 nodes.

Tetrahedron connectivities:



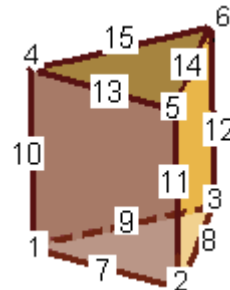
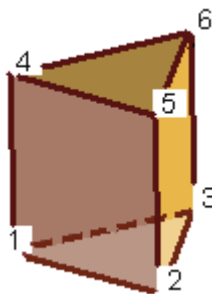
- **Hexahedra:** 8, 20 or 27 nodes.

Hexahedron
connectivities:



- **Prism:** 6 or 15 nodes.

Prism connectivities:



The `Linear` option assures not only the meshing of lines, but also the creation of 2-node contact lines between surfaces or volumes if desired.

At contact surfaces, GiD elements are:

- **Linear:** 2 or 3 nodes.
- **Triangle:** 3 or 6 nodes.
- **Quadrilateral:** 4, 8 or 9 nodes.

At contact volumes (and separated contacts), elements are:

- **Linear:** 2 or 3 nodes.
- **Prism:** 6 or 15 nodes.
- **Hexahedron:** 8, 20 or 27 nodes.

To decide which parts of the geometry should be meshed, use the `Mesh criteria` command (see [Mesh criteria](#)).

Mesh criteria

Menu

Mesh

Mesh criteria

GiD provides five different criteria to generate the mesh. The `Default` option skips meshing the boundaries, that is, lines for surface meshes and surfaces for volume meshes.

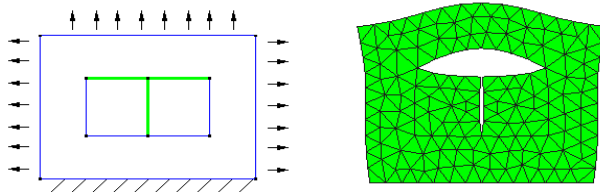
The `Mesh` option lets you choose the entities to be meshed, while the `No Mesh` option does the opposite.

The `Skip` option forces GiD to skip a geometrical entity when meshing (so the entity will not have mesh), while the `No Skip` option forces GiD not to skip the geometrical entity (so the entity will have mesh) when the RJUMP surface mesher is used (see [Preferences](#)→Meshing).

The `Automatic skip` option lets GiD decide if the geometrical entity should be skipped or not skipped when the RJUMP surface mesher is used. This decision is taken according to the tangency between entities: those entities that are tangent enough will be skipped when meshing.

The `Force points to` option forces the selected points to belong to the surface or volume mesh, even though they do not necessarily belong to the surface or the volume.

Use the `Duplicate` option when you want to create a discontinuity in the mesh in a particular place by duplicating nodes. This is interesting, for example, when dealing with very thin shapes where it is difficult to represent the domain with two overlapped surfaces, and it is easier to have a single surface marked with this meshing option (like sails embedded in a volume, material cracks, etc.).



Marked lines and the deformed mesh after an structural analysis

It is possible to mark lines embedded in a 2D domain, or surfaces in the case of 3D domains.

Reset mesh data

Menu



Mesh > Reset mesh data

This command resets all the sizes assigned to entities. This means that all of them will be unassigned.

To unassign only certain entities, assign the size 0.0 (see [Unstructured](#)) to the entities where the default size is required.

The information about element types, mesh criteria and quadratic parameters is also reset.

Draw

Menu



Mesh > Draw

This option is used to draw meshing properties in geometrical entities.

Sizes

Menu

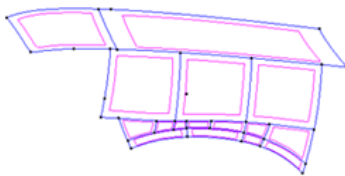


Mesh > Draw > Sizes

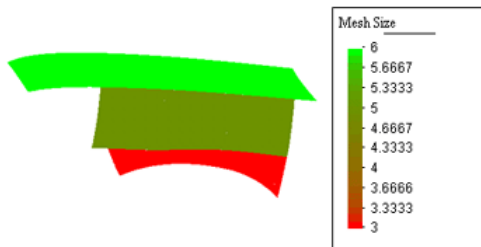
When sizes are assigned to points, lines, surfaces or volumes using the `Assign Unstructured sizes` option, it is possible to draw the different assigned sizes in different colors.

Example:

In the following example some different sizes are assigned to surfaces. Sizes of 3, 5 and 6 are assigned depending on the surface.



After choosing the `Draw Sizes` option (over surfaces) we get the following result:



Element Type

Menu



With this option you can see which element types have been assigned to each geometric entity. If no element type has been assigned, it is shown as `Default` (see [MESH](#)).

Mesh / No mesh

Menu



With this option you can see the entities that GiD has either been forced to mesh or forced not to mesh (see [Mesh criteria](#)). If the meshing criteria have not been assigned to an entity, it is shown as `Default` (see [MESH](#)).

Structured Type

Menu



With this option you can see what kind of mesh will be generated for geometrical entities, i.e. Unstructured, Structured or Semi-Structured. If no level of structure has been assigned, it is shown as `Default` (see [MESH](#)).

Skip entities (Rjump)

Menu



With this option you can see which lines and points will be skipped when meshing if the Rjump mesher (see [Preferences](#)→Meshing) is used.

Generate mesh

Menu



When everything is ready for mesh generation, select this command. If there is a previously generated mesh, GiD asks if this should be erased. It will be lost from the memory, but will remain on the disk until the project is next saved (see [Save](#)).

The mesher or mesher combination can be chosen in **preferences** (see [Preferences](#)).

Next, GiD asks for a general element size which will be applied to all lines, surfaces and volumes that do not have one previously defined (see [Unstructured](#)). GiD offers two default possibilities:

- One default size automatically calculated by the program to define a coarse mesh.
- The last size given by the user in a previous meshing.

You can choose one of these or enter a new one.

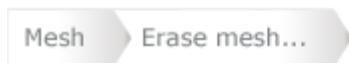
The size is given by the average side of the corresponding triangle or quadrilateral.

During meshing a progress bar indicates the number of generated surfaces or volumes in relation to the total number of surfaces or volumes.

Meshing can be stopped at any time by clicking the `Stop` button. Sometimes it is necessary to press the button repeatedly or keep it pressed for a few seconds.

Erase mesh

Menu



If the model has a mesh this option erases it.

Edit mesh

Menu



This option lets you modify a mesh. All modifications will be lost when the mesh is generated again.

Move node

Menu



By using this command, an existing node is selected and moved. The new position is entered in the usual way (see [Point definition](#)).

Split Elements

Menu



To split elements, select them in the usual way (see [Entity selection](#)), and then press `escape` (see [Escape](#)) to perform the action. Triangles can be split into triangles, quadrilaterals into two or four triangles, hexahedra into tetrahedra, and prisms into tetrahedra.

When splitting triangles, the new nodes can be located in the mid-edge or with an enhanced interpolation (modified Butterfly scheme) in order to obtain a smooth mesh.

When splitting quadrilaterals, if `SymmetricalStructuredTriangles` is set (see [Preferences → Meshing](#)), then four triangles are generated for each quadrilateral; otherwise only two triangles are created per element.

When splitting hexahedra, if `SymmetricalStructuredTetrahedra` is set (see [Preferences → Meshing](#)), then 24 tetrahedra are generated for each hexahedron; otherwise six tetrahedra are created per element.

If original elements are quadratic, the triangles or tetrahedra obtained are quadratic. Otherwise, if original elements are linear, the triangles or tetrahedra obtained are linear.

In the case of quadratic quadrilaterals, extra nodes are generated. These nodes are not associated with geometric entities, they have simply been obtained by interpolating mesh node coordinates.

In the case of hexahedra or prisms, all selected elements must be of the same quadratic type.

Note: Currently, the operation `Split triangles` only works for linear elements.

Smooth Elements

Menu



To smooth elements, select them in the usual way (see [Entity selection](#)), and then press `escape` (see [Escape](#)) to perform the action.

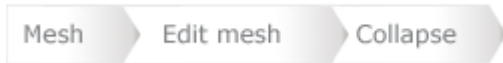
Currently only triangles, tetrahedra and hexahedra can be smoothed.

In the case of tetrahedra and hexahedra, element connectivity is conserved after the smoothing; however, with triangle elements this may be modified during the smoothing process.

All selected elements must be of the same quadratic type.

Collapse

Menu



The `Collapse` function converts coincident entities, i.e. entities that are close each other, into one.

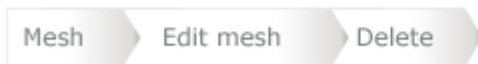
It is possible to collapse edges, nodes, elements or the whole mesh.

- `Collapse mesh`: collapses all the nodes of the mesh.
- `Collapse edges`: joins nodes that are connected by edges shorter than the `ImportTolerance` value.
- `Collapse nodes`: asks you to select some nodes. Nodes closer together than the `ImportTolerance` value are collapsed.
- `Collapse elements`: asks you to select some elements. Then, the nodes of these elements that are closer together than the `ImportTolerance` value are collapsed.

Note: Entities belonging to a frozen layer (see [Layers](#)) are not checked when collapsing.

Delete nodes/elements

Menu



To delete elements or nodes, select them in the usual way (see [Entity selection](#)), and then press `escape` (see [Escape](#)) to perform the action.

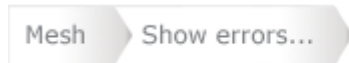
Nodes that no longer belong to any element after the operation are also erased.

Note: It is possible to filter the selection, e.g. to select only triangles but not quadrilaterals (see [Selection window](#)).

Note: Only lonely nodes (nodes of the mesh that do not belong to any element) can be deleted.

Show errors

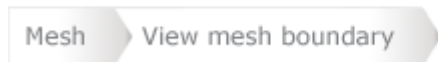
Menu



This option opens the `mesh errors` window. This window presents a list of the entities that GiD could not mesh, and some information about the problems that occurred during the meshing process. By right-clicking over an item in the list, advice will be displayed about how to solve the meshing problems for each geometrical entity.

View mesh boundary

Menu



This option draws the boundaries of the mesh on the screen.

Boundaries for triangular or quadrilateral meshes are line elements.

Boundaries for tetrahedra or brick meshes are triangles or quadrilaterals. This option can be useful when rendering a volume mesh (see [Render](#)).

Create boundary mesh

Menu



This option creates the boundary mesh of the existing mesh.

The boundary mesh for triangular or quadrilateral meshes is a line element mesh.

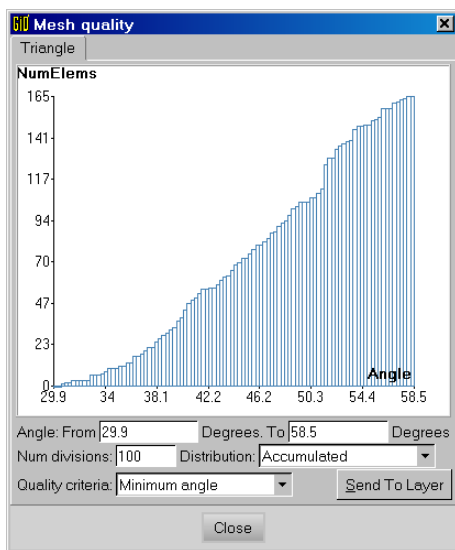
The boundary mesh for tetrahedra or brick meshes is a triangular or quadrilateral mesh.

Mesh quality

Menu

Mesh → Mesh quality

This option opens a window that shows information about the quality of the mesh elements.



Mesh Quality window

There are six criteria used to measure the quality of the elements:

1. **Minimum angle:** The quality criterion is the minimum angle in surface elements and the minimum dihedral angle for volume elements. This means that elements with a small angle are considered to be of a worse quality than ones with bigger angles.
2. **Maximum angle:** This gives the maximum angle for every element. Elements with bigger angles are considered worse.
Typically, the **Minimum angle** criterion is good for qualifying triangles and tetrahedra and the **Maximum angle** criterion is good for quadrilaterals and hexahedra
3. **Element vol:** The quality criterion is the size of elements (distance for lines, area for surfaces and volume for volumes). Elements with small "volume" are considered worse.
4. **Minimum edge:** The quality criterion is the size of the smaller edge of each element. Elements with smaller edges are considered worse.

5. **Maximum edge:** The quality criterion is the size of the largest edge of each element. Elements with bigger edges are considered worse.
6. **Shape quality:** The quality criterion measures the likeness of the element to the reference one (an equilateral triangle in the case of triangles, a regular tetrahedron in the case of tetrahedra, a square in the case of quadrilaterals and a cube in the case of hexahedra). Its value is 1 for a perfect element (the reference one), and it decreases as the element becomes worse. If it reaches a negative figure it means that the element has a negative Jacobian at some point. The mathematical expression of this quality measure for each type of element is as follows.

- **Triangles:** The shape quality q of triangles is measured as

$$q = \frac{4 \cdot \sqrt{3} \cdot Area}{\sum_{i=1}^3 l_i^2}$$

where **Area** is the area of the triangle, and l_i ($i=1\dots3$) are the lengths of the triangle's edges.

- **Tetrahedra:** The shape quality q of tetrahedra is measured as

$$q = \frac{6 \cdot \sqrt{2} \cdot Volume}{\sum_{i=1}^6 l_i^3}$$

where **Volume** is the volume of the tetrahedron, and l_i ($i=1\dots6$) are the lengths of the tetrahedron's edges.

- **Quadrilaterals:** The shape quality q of quadrilaterals is measured as the quality of the worst quality node. The quality of a node q_n is given by

$$q_n = \frac{2 \cdot Area_n}{l_1^2 + l_2^2}$$

where l_1 and l_2 are the lengths of concurrent edges of the node, and **Area_n** is the area of a fictitious parallelogram, made with these two edges.

- **Hexahedra:** The shape quality q of hexahedra is measured as the quality of the worst quality node. The quality of a node q_n is given by

$$q_n = \frac{Volume_n}{\left(\frac{(l_1^2 + l_2^2 + l_3^2)^2}{3} - (A_1^2 + A_2^2 + A_3^2) + (Volume_n)^{4/3} \right)^{3/4}}$$

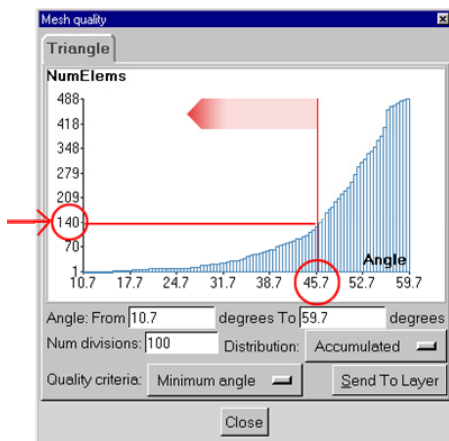
where $l_1, l_2, l_3, A_1, A_2, A_3$ are the lengths and areas of concurrent edges and faces of the node, and $Volume_n$ is the volume of a fictitious parallelepiped, made with these concurrent faces.

7. **Minimum Jacobian:** The quality criterion is the value of the minimum Jacobian between the Jacobians calculated at each element gauss point. If there are elements with negative Jacobians, problems may be encountered in some calculation processes.

There are two visualization modes:

1. **Normal:** The graph shows the number of elements that have an angle of a certain size.
2. **Accumulated:** The graph shows the number of elements which have an angle of a given size or smaller.

In the **Mesh quality** window, if you double click on a value, the elements below this value are selected in red. These selected elements can be sent to a layer using the `Send To Layer` button in the Mesh quality window.



Example of a mesh quality study

In this example we are studying the mesh using the minimum angle criterion. We can see that **140** elements of our mesh have an angle of less than **45.7** degrees. If we double-click on the graphic, the 140 elements which have an angle smaller than 45.7 degrees will be selected.

CALCULATE

With this menu, you can initiate and manage the analysis of a problem, hereafter referred to as a "process".

You will see in the sections that follow that several analyses, or processes, can be run at the same time.

Calculate

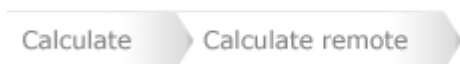
Menu



This option begins the process module. Once it is selected, you can continue working with GiD as usual.

Calculate remote

Menu



This option begins the process module on a remote machine. Once it is selected, you can continue working with GiD as usual.

Note: ProcServer with the same model problem type must be installed and running on the remote machine in order to use this option. (ProcServer is not included with GiD.)

Cancel process

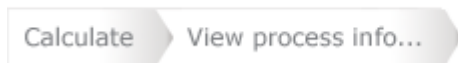
Menu



Selecting a process that is currently running and clicking this option will halt its execution.

View process info

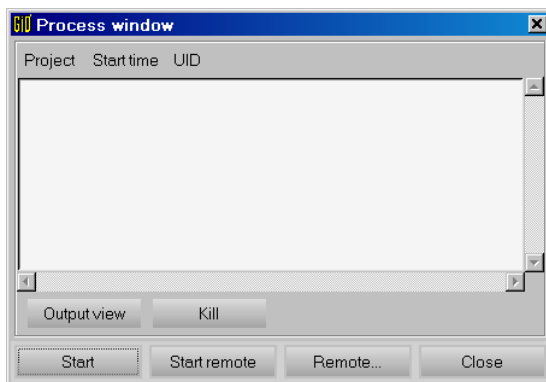
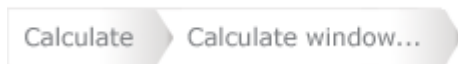
Menu



Select a process that is currently running and click this option to open a window that shows information relating to the process, such as iterations, convergence, etc. Clicking `Close` will close the window, but will not halt the process.

Calculate window

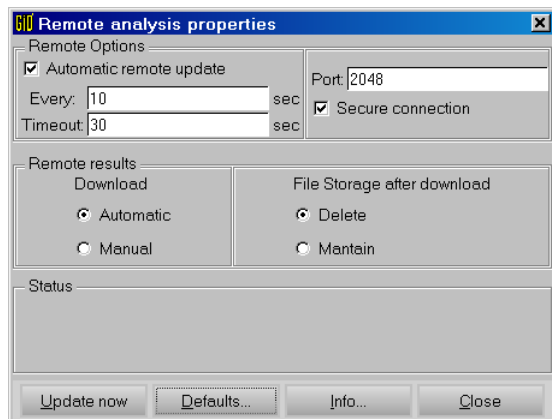
Menu



Process manager windows

Selecting this option opens a window in which a list of all the running processes is shown, along with some useful information like name, starting time, etc.

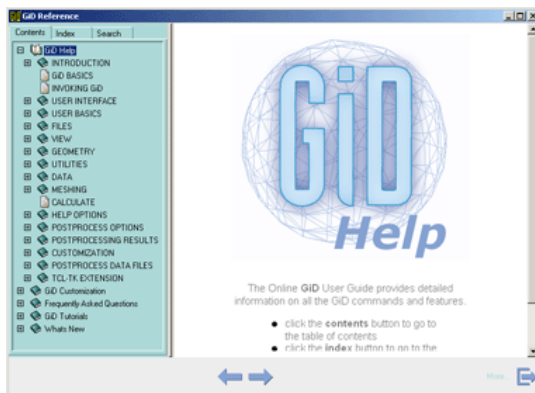
The buttons in this window let you control some running process features, such as terminating the process (`Terminate`), starting a remote calculation (`Start remote`), or setting remote analysis properties (`Remote...`). These remote analysis properties are shown in the next figure.



Window to customize remote calculation options.

HELP

GiD provides a help system based in html format. In the same window you can access the following sections: “GiD Help”, “GiD Customization”, “FAQs”, “GiD Tutorials” and “What's New”. These contents can also be accessed directly from Help Menu. The help window provides three ways of accessing a given topic: a) through a table of contents shown as a tree, b) through an indexed list of terms appearing inside the help, and c) through a search engine.

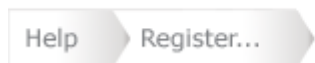


GiD help window

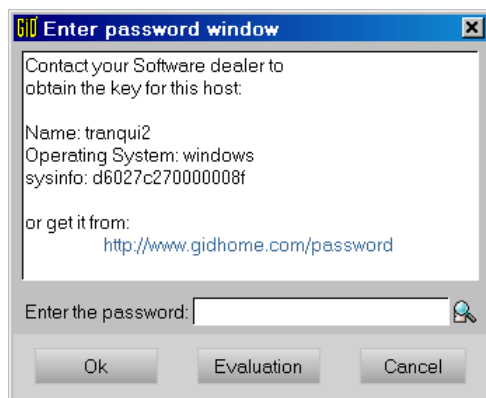
From the Help menu you can also register **GiD** (see [Register GiD](#)) and **Problem Types** (see [Register Problem types](#)).

Register GiD


Menu



In order to get the most of GiD, you need to register a password that can be obtained from <http://www.gidhome.com>. From this site you can obtain a permanent or temporary password which must be typed in the window shown below.



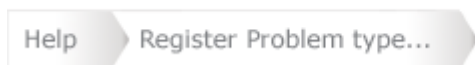
GiD register window

If you have previously registered your current copy of GiD (official versions only), the password can be reloaded by clicking the  icon and selecting the folder where the old password is.

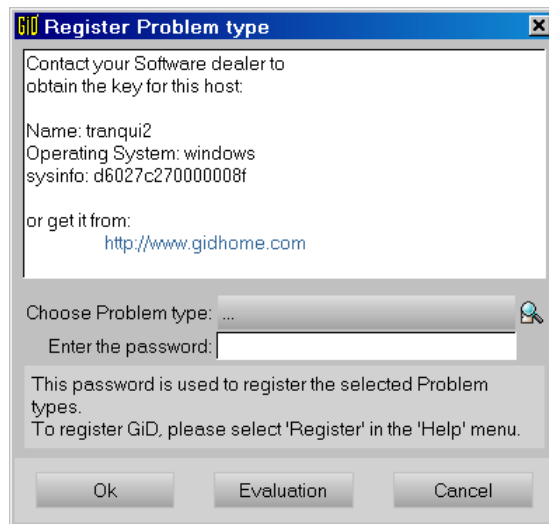
After registering either a permanent or temporary password you will be able to generate and postprocess an unlimited number of nodes and elements.

Register Problem types


Menu



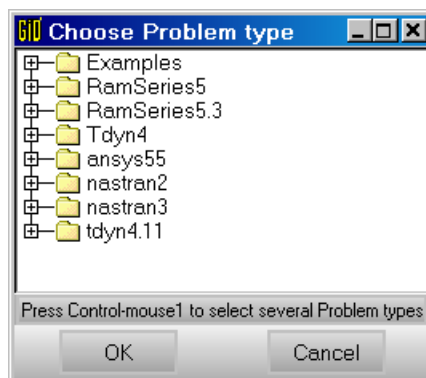
In the same way that some of GiD's capabilities have restricted access before it is registered, so not all problem types are available without registration. The layout of the **Register Problem type** window is shown below:



Problem type register window

If a problem type has been registered previously, the password can be reloaded by clicking  and selecting the folder where the old password is.

A problem type does not need to be loaded to be registered. In this case, click the **Choose a Problem type** bar and a window similar to the one below will appear where you can choose a set of problem types to be registered. If a problem type is currently in use then it is selected by default.



Customizing Problem type registration

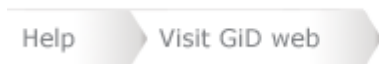
GiD provides a default validation when registering a module or problem type. This consists of checking that the password is not empty. If the password is valid, GiD appends a line to the file <problem type>/password.txt similar to this:

```
hostname password # 2005 01 19 Password for Problem type '/pathroot-  
to-problem-type/problemtype.gid'
```

This default validation can be overridden, but this involves Tcl programming. For a description of how to provide a custom password validation see [ValidatePassword node](#).

Visit GiD web

Menu



This command takes you to the GiD homepage (<http://www.gidhome.com>).

About

Menu



This command gives information about the program such as the version being run, the system or libraries.

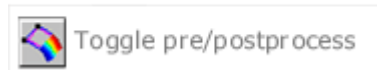
POSTPROCESS OPTIONS

Introduction

Menu

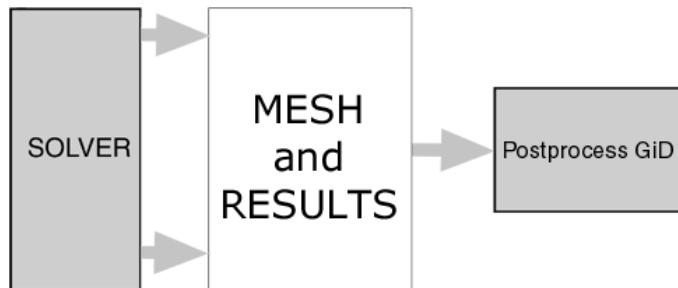


Toolbar



This chapter describes some relevant aspects of the postprocessing step and the way to load results from a numerical analysis into GiD.

In GiD Postprocess you can study the results obtained from a solver program. GiD receives mesh and results information from the solver module, and if the solver module does not create any new mesh, the preprocess mesh is used.



The solver and GiD Postprocess communicate through the transfer of files.

The solver program has to write the results in a file that must have the extension `.post.res`, or the old `.flavia.res`, and its name must be `ProjectName`. If the solver writes a mesh, the file must have the extension `.post.msh`, or the old `.flavia.msh` (see [POSTPROCESS DATA FILES](#)).

The extensions `.msh` and `.res` are also allowed, but only files with the extensions `.post.msh` and `.post.res` (and the old ones `.flavia.msh` and `.flavia.res`) will automatically be read by GiD when postprocessing the GiD project.

There are two ways to postprocess inside GiD:

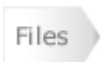
- **Postprocess inside a GiD project:** Start GiD and open the GiD project (`ProjectName.gid`) you want to postprocess, and then select the `Postprocess` option from the files menu or click on the `pre/post` toolbar button. This way the file `ProjectName.post.res`, or the old `ProjectName.flavia.res`, will automatically be read by GiD, along with the file `ProjectName.post.msh`, or `ProjectName.flavia.msh`, if present.
- **Postprocess other files:** Start GiD and select the `Postprocess` option directly from the files menu or click on the `pre/post` toolbar button. Now open the pair files `OtherModel.msh` and `OtherModel.res`.

Once inside the postprocessing component of GiD, all the visualization features and management options of the preprocessing section are available: Zoom, Rotate (Rotate screen/object axes, Rotate trackball, etc.), Pan, Redraw, Render, Label, Clip Planes, Perspective, etc.

Note: There is no need to load a project into GiD to use its postprocessing facility; you can open mesh and results information directly from GiD Postprocess (see [Files menu](#)).

Files menu

Menu



Several useful options can be found in the Files menu.

- **New:** Clears all postprocess information present in GiD.
- **Open:** Reads postprocess information in GiD.
- **Open multiple:** With this option you can load multiple meshes (pairs of `.msh` and `.res`, or `.bin` files) into GiD. This is useful, for instance, when performing an analysis where some or all steps require re-meshing. By now, each different mesh should be written in a separate `Project.post.msh` file with its own `Project.post.res` results file. From the file browser window, you can select several files in one go by left-clicking one file and then `<Shift>`-left-clicking another so that all the files in between are highlighted; further files can be added or removed individually with `<Ctrl>`-left-click. Normal operations, such as animation, displaying results and cuts, can be done over

these meshes, and they will be actualized when the selected analysis/step is changed, for example by means of `View results`→`Default analysis/step` (see [Re-meshing and adaptivity](#)).

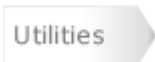
- **Merge:** Reads mesh and results information from an ASCII or a binary file and adds them to the current ones. If nodes are already present inside GiD, they will be overwritten. No renumeration is done.
- **Import:**
 - `NASTRAN mesh`: Reads a NASTRAN mesh file.
 - `FEMAP`: Reads FEMAP Neutral ASCII files and binary files.
 - `TECPLOT ASCII`: Reads TECPLOT 9.0 ASCII files.
 - `3DStudio file`: Reads a mesh in `.3ds` 3DStudio format.
 - `Cuts`: Reads cut planes, cut wires and iso-surface cuts in GiD so that the same cuts and cut-spheres can be used among several postprocess meshes.
 - `Graphs`: Adds graphs to those that may or may not have already been created inside GiD.
- **Export:**
 - `Post information`
 - ♦ **ASCII files**: saves meshes, sets and results to ASCII files using the new PostProcess format (see [Postprocess results format](#): `ProjectName.post.res`, `ProjectName.flavia.res`, [Postprocess mesh format](#): `ProjectName.post.msh`, `ProjectName.flavia.msh`). If multiple meshes are read with the `Open multiple` option explained above, only two files will be saved, one for the meshes and another one for the results, where each post information file (or pair `.msh` + `.res`) will define a Group (see [Re-meshing and adaptivity](#)).
 - ♦ **Binary (whole model)**: Saves meshes, sets and results to one binary file. If multiple meshes are read with the `Open multiple` option explained above, only two files will be saved, one for the meshes and another one for the results, where each post information file (or pair `.msh` + `.res`) will define a Group (see [Re-meshing and adaptivity](#)).
 - ♦ **Binary (only results)**: Saves results information only, so the transition between Pre- and Postprocess will be quicker.
 - ♦ **ASCII boundaries**: Saves the boundaries of the meshes and sets, with its nodal results, to ASCII files (one for mesh information `.msh`, one for the results `.res`). The boundary of a hexahedron/tetrahedron mesh is a quadrilateral/triangle mesh. The boundary of a quadrilateral/triangle mesh is a line mesh. There is no boundary of a line/point mesh.

- **Cut:** Saves cut planes, cut wires and iso-surface mesh cuts so that the same cuts can be used among several postprocess meshes. Cut spheres can also be saved.
- **Graph:** Saves graphs in ASCII (gnuplot) format. If the option **All** is chosen, you will be asked for a prefix. GiD will then create a file for each graph with the names `prefix-1.cur`, `prefix-2.cur`, `prefix-3.cur` and so on ...
- **Cover Mesh:** After visualizing the cover mesh of the points/nodes, this mesh can be saved for other uses.
- **Preprocess:** Selecting this option will open an 'Are you sure?' dialog box. Clicking **Yes** will return you to the preprocessing component of GiD.

The rest of the options are the same as when in preprocessing mode.


Utilities menu

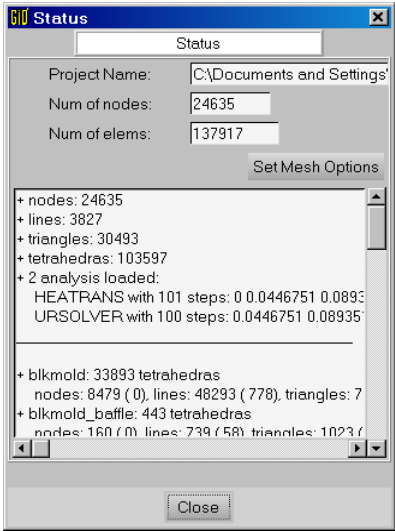
Menu



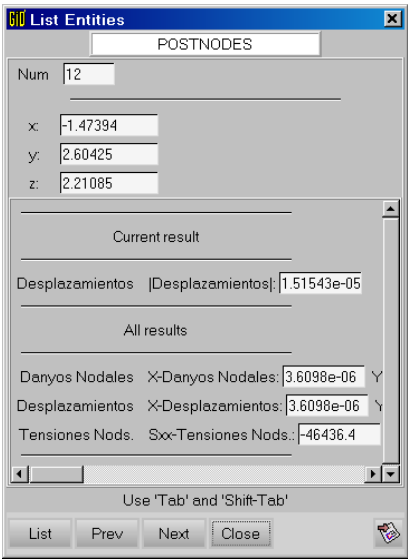
Inside the **Utilities** menu, the options **Id**, **Signal**, **Distance** and **Calculator** have the same functionality as in preprocessing mode (see [UTILITIES](#)). Other options in the **Utilities** menu are slightly different.

- **Status:** A window appears showing the general postprocess status: number of meshes, elements, etc.
- **List**
 - **Nodes:** A window appears showing information about selected nodes: coordinates, the current visualized result, and all other nodal results defined over the selected nodes.
 - **Elements:** A window appears showing information about selected elements: connectivity, the gauss points defined on the selected elements, the current visualized result, and all other results defined over the gauss points of the selected elements.

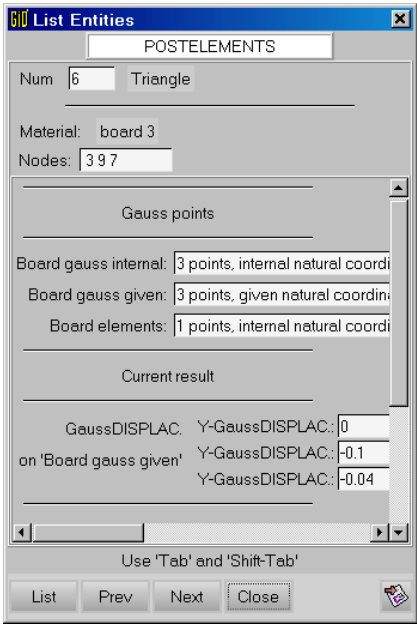
All this information can be sent to the active report (see [Report](#)) by using the  button.



Status window



List nodes window



List elements window

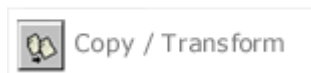
- **Collapse:** Collapses nodes that are together in a set.
- **Join:** Joins several sets into one.
- **Delete:** Deletes meshes, sets and cuts.
- **Texture:** Adds textures to sets (see [Textures](#)).

The **Copy** tool introduces some changes:

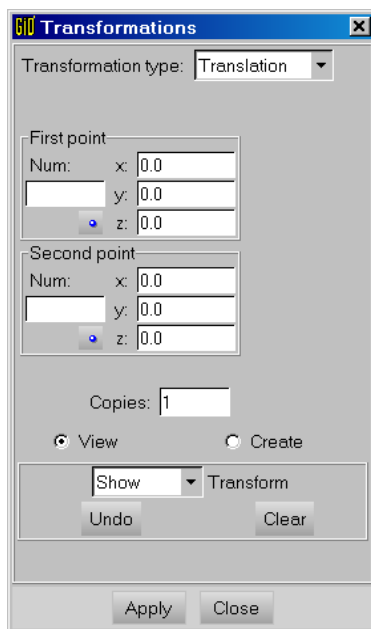
Menu



Toolbar



When you select the **Copy** tool, the `Transformations` window appears. This window allows you to repeat the visualization using translation, rotation and mirror transformations, so that GiD can draw a whole model when only a part of it was analyzed.



Transformations Window

The types of transformation are:

- **Translation:** This is defined by two points. Relative movements can be obtained by defining the first point as 0,0,0 and considering the second point as the translation vector (see [Point definition](#)).
- **Rotation:** It is necessary to enter two points in 3D or one point in 2D. In 3D, these two points define the rotation axis and its orientation. In 2D, the axis goes from the defined point towards z positive. Enter the angle of rotation in degrees. It can be positive or negative. In 3D, the direction is defined by the right hand rule. In 2D, it is counter-clockwise.
- **Mirror:** This is defined by three points that cannot be in a line. These points form a plane that is the mirror plane. In 2D, the mirror line is defined by two points.

The other available options are:

- **Copies:** By entering the number of repetitions, the operation selected is performed this number of times.
- **View:** The transformation is only for viewing purposes. *Show/Hide*, *Undo* and *Clear* options are enabled.
- **Create:** All the Post information (meshes & results) will be duplicated when performing the transformation. The options *Show/Hide*, *Undo* and *Clear* are disabled.
- **Show/Hide Transform.:** Allows you to see either the original model or the one with the transformations active.
- **Undo:** Deletes the last transformation performed. It can be used repeatedly to clear all transformations.
- **Clear:** Clears all the transformations in one go.

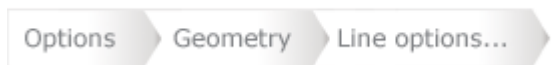
The **DoIt** button tells the program to perform the transformation that has been selected.

Point and Line options

Menu

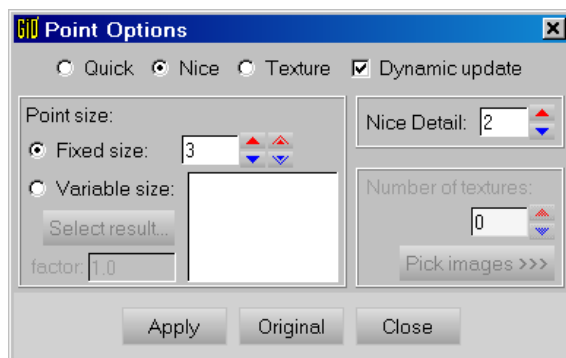


Menu



As **points** and **lines** can be viewed, there are several interesting options for each of them.

- Point options:

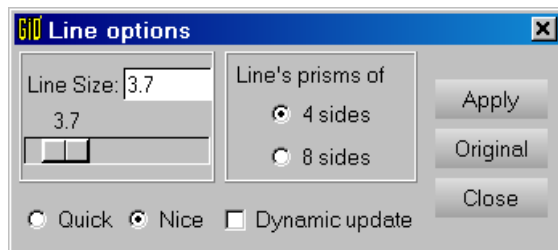


Point options window

Here you can select whether to draw the points **Quick**, **Nice** or with the center of a **Texture** glued to the point. Quick: points will be drawn as big dots. Nice: points will be drawn as little spheres (quadrilateral meshes). For every draw style, the Point Size can be changed, but ranges vary between Quick, which depends on the graphics library, and Nice and Texture. When the Nice style is selected, the Nice detail level can be adjusted. The number represents the number of vertical and horizontal subdivisions of a sphere.

Note: The changes affect the representation of all the points – point elements and the arrow heads of vectors when using the `Point detail level` (see [Display vectors](#)).

- Line options:



Line options window

Here you can select whether to draw the lines with a **Quick** style or a **Nice** one. Selecting a Line Size of 0.0 in the Nice style, will cause the lines not to be drawn. Quick: lines are drawn as 3-point thick lines. The line size is fixed. Nice: lines are drawn as long 4-sided or 8-sided prisms.

The size and width of the lines can be changed. When the lines are drawn nicely, the number of sides of the prism used to draw them can be changed between 4 and 8 sides.

Note: Line options no longer affect the stream lines as before, but only the line elements. For stream lines options see [Stream Lines](#).

Display Style

Menu



Below it is an example of the `View Style` window, where almost all the interesting visualization options can be adjusted. It only deals with meshes, sets or cuts, and not with results.



Display Style Window

Selecting `Volumes`, `Surfaces` and/or `Cuts`, you can switch them **On** and **Off**, Delete them or Rename them. By clicking on a Volume, Surface or Cut, and pressing `Color...` you can adjust

the appearance of the selected set; you can change the color, including the `Ambient`, `Diffuse`, `Specular` and `Shininess` components, and return it to its `Default` color. The `Diffuse` component is used for all the representations, while the others (`Ambient`, `Specular` and `Shininess`) have more effect in `Render` visualizations.

In the **Style** menu you can choose how volumes, surfaces and cuts should be drawn. These options are:

- **Boundaries:** All the edges of the boundaries and surface elements will be displayed in black. An edge is the line that belongs to just one surface element, or that belongs to two surface elements as long as the faces form an angle smaller than has been predefined (you can specify this angle in `Results Options BorderAngle` in the **Right buttons** menu).
- **Hidden Boundaries:** The same as before, but here the edges that are behind the volumes, surfaces and cuts are removed.
- **All Lines:** All the lines of surface elements are drawn. When drawing volumes, GiD only draws the surface/boundary elements of this volume mesh. How GiD can draw these interior elements is explained below. Each volume, surface and cut is drawn with its own colors.
- **Hidden Lines:** The same as before, but here the lines that are behind the volumes, surfaces and cuts are removed.
- **Body:** The elements of volumes, surfaces and cuts, are drawn in filled mode, i.e. they are drawn as solid elements. Each volume, surface and cut is drawn with its own colors. It can be very hard to recognize the shape of the meshes if no illumination is active.
- **Body Boundaries:** Same as before, but with `Hidden Boundaries` drawn too.
- **Body Lines:** Same as before, but with `Hidden Lines` drawn too.
- **Points:** The nodes of the meshes are drawn.

With the **Render** menu you can select how the mesh should be displayed:

- `Normal` – no lighting;
- `Flat` – lighting with sharpened edges;
- `Smooth` – lighting with smoothed edges.

The light direction can also be changed by clicking on the **lamp** icon.

Near the `Culling` label, you can choose whether the `Front Faces`, `Back Faces`, `Front and Back Faces` are culled, i.e. not drawn, or `No Faces` are culled. This option is useful for looking at volume meshes, etc.

Both geometry and mesh conditions can also be drawn if they are present in preprocess.

The following two options can be changed separately for each mesh/set/cut:

Massive lets you see the elements that are inside a volume mesh, and draw all the vectors inside a volume/surface/cut when a Boundaries display style is used.

Transparent determines whether the volumes/surfaces/cuts will be drawn as transparent or opaque, so that, for instance, isosurfaces inside them can be viewed easily.

Textures

Menu



In GiD it is also possible to assign a texture to a Set. Inside the **Utilities**→**Texture** menu there are several options:

- **View** No / Fast / Nice: switches between viewing the textures over the sets or not. When the texture is small and a pixel of the textures must be drawn over several pixels on the screen, the Fast mode just draws the pixels using the 'nearest neighbor' policy, while the Nice mode tries to interpolate the colours of the pixels from the original.
- **Add** Screen Map / To 4 Sided / BoundImg to BoundSet: maps a texture to a set.
 - **ScreenMap**: by picking four points over the screen, GiD projects the texture parallel to the screen, over the underlying sets.
 - **4 sided**: GiD tries to match the 4 sides of the texture to the 4 sides of the set. The best results are achieved using quadrilateral sets which are more or less flat, at least in one direction.
 - **BoundImg to BoundSet**: GiD looks for the border of the texture inside the image file and tries to glue it to the border of the set, for instance a texture circle to a circle set.
- **Change** Flip Horiz. / Vert.: changes the orientation of the texture once it is applied to the Set.

Note: When the display style of the visualization is changed, for instance from Body to Body Boundaries, the visualization of the texture is switched off. To view the texture, just select **Utilities**→**Texture**→**View**→**Fast / Nice**.

Once the texture is applied to a Set, any deformed of the Set leads to a corresponding deformation of the texture. In this way the deformed texture can be visualized.

Cover mesh

Menu



Another feature in GiD is the calculation of the involving mesh of a set of points or nodes. To switch the visualization of this mesh on and off just select `Options→Geometry→Covering mesh`. After saying `Yes` to the visualization of the covering mesh, you will be asked for a number. This number is the distance between the covering mesh and the points.

This option is not only available for points, but also for every mesh/set present in GiD.

Note: This covering mesh is recalculated when the mesh is deformed. So in a particle movement system the covering mesh will also move along with the particles.

This mesh can also be saved through `Files→Export→Cover mesh`.

POSTPROCESS RESULTS

The kinds of results that will be displayed on screen can be grouped into five major categories:

- **Scalar view results:** Show Minimum & Maximum, Contour Fill, Contour Text Ranges, Contour Lines, Iso Surface and its configuration options.
- **Vector view results:** Mesh deformation, Display Vectors, Stream Lines (Particle Tracing)
- **Line diagrams:** Scalar line diagram and vector diagrams.
- **Graph lines:** XY plots.
- **Animation:** Animation of the current result visualization.

The **View Results** menu and window are used to manage the visualization of the different type of results.

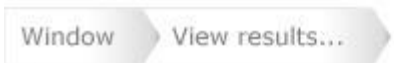
Menu



The **View Results menu** lets you select any type of result:

- Contour fill
- Contour ranges
- Contour lines
- Display vectors
- Show min/max
- Iso surfaces
- Stream lines
- Graphs
- Deformation
- Line diagrams

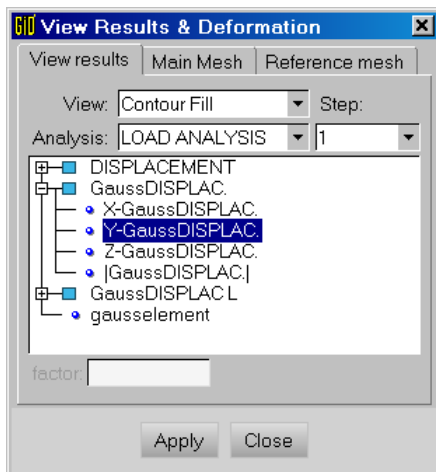
Menu



The **View Results window** manages the visualization of the following type of results:

- Contour fill
- Contour ranges (if Result Range Tables are present)

- Contour lines
- Display vectors
- Show min/max
- Scalar and Vector line diagram (if Linear elements are present)



View Results Window

As the results are grouped into steps and analyses, GiD must know which analysis and step is currently selected for displaying results. The **Analysis** menu of the **View Results** window is used to select the current analysis and step to be used for the rest of the results options. If some of the results view requires another analysis or step, you will be asked for it.

Note: If you no longer wish to view results, you can select **No Result** in this window or in the **View results** menu.

Contour Fill

Menu



This option allows the visualization of colored zones, in which a variable or a component varies between two defined values. GiD can use as many colors as permitted by the graphical capabilities of the computer. When a high number of colors is used, the variation of these colors

looks continuous, but the visualization becomes slower unless the `Fast-Rotation` option is used. A menu of the variables to be represented will be shown, and the one that is chosen will be displayed using the default analysis and step selected.

Vectors will be unfolded into their X, Y, and Z components and module. Symmetrical matrix values will be unfolded into the Sxx component, Syy component, Szz component, Sxy component, Syz component and Sxz component of the original matrix and also into the Si component, Sii component and Siii component in 3D problems or angular variation in 2D problems. Any of these components can be selected to be visualized.

When using results defined over gauss points, they are extrapolated to the nodes so that discontinuities can appear between adjacent elements. If the gauss points results cannot be extrapolated to the nodes, they are drawn as coloured spheres. The radius and detail level of these spheres can be configured as well (see [Point and Line options](#)).

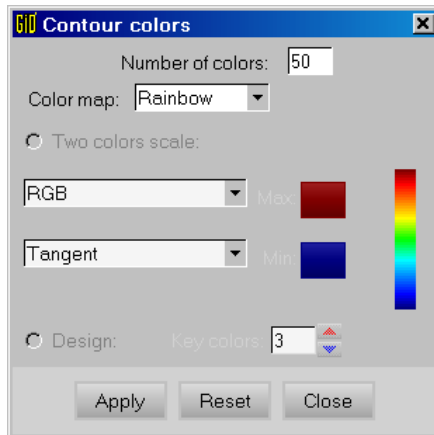
Several configuration options can be accessed via the `Options` menu.

Menu



- **Number Of Colors:** Here the number of colors of the results color ramp can be specified.
- **Width Intervals:** Fixes the width of each color zone. For instance, if a `width interval` of 2 is set, each color will represent a zone where the results differ by two units at the most.
- **Set Limits:** This option is used to tell the program which contour limits it should use when there are no user defined limits: the absolute minimum and maximum of all sets or shown sets, for the actual step or for all steps.
- **Define limits:** When choosing this option the `Contour Limits` window appears (this option is also available in the Postprocess toolbar). With this window you can set the minimum/maximum value that Contour Fill should use. Outliers will be drawn in the color defined in the `Out Min Color / Out Max Color` option.
- **Reset Limit Values:** This option resets the values defined in the `Define limits` option.
- **Reset All:** This option sets all Contour Fill options by default.
- **Max/Min Options:** Inside this group, several options for the minimum or maximum value can be defined:
 - **ResetValue:** Here you can reset the maximum/minimum value, so that the default value is used.

- **OutMaxColor / OutMinColor:** With this option you can specify how the outlying values should be drawn: Black, Max/Min Color, Transparent or Material.
 - **Def. MaxColor / Def. MinColor:** This option lets you define the color for the minimum or maximum value for the color scale to start with.
- **Color scale:** Specifies the properties of the color scale:
 - **Standard:** The color scale will be the default: starting from blue (minimum) through green to red (maximum).
 - **Inverse Standard:** The color scale will be the inverse of the default: starting from red (minimum) through green to blue (maximum).
 - **Terrain Map:** A physical map-like color ramp will be used.
 - **Black White:** Black for Minimum and White for Maximum, and a grey scale between these.
 - **Scale Ramp:** This option lets you specify how the ramp should change from minimum color to the maximum color: Tangent, ArcTangent or Linear. The default is ArcTangent.
 - **Scale Type:** Tells GiD how the colors between the minimum color and the maximum color should change: RGB or HSV.
- **Color window:** a window opens to let you configure the colour scale of the contours easily



Contour Colors Window

Smooth Contour Fill

Menu

A horizontal menu bar with two buttons. The first button is labeled 'View results' and the second button is labeled 'Smooth Contour Fill'. Both buttons have a light gray background and a dark gray border.

This option displays a `Contour Fill` (as explained in the previous section) with a local smoothing of the results defined over gauss points.

With the `Smoothing type` option inside the menu `Options`→`Contour` you can choose between these types of local smoothing:

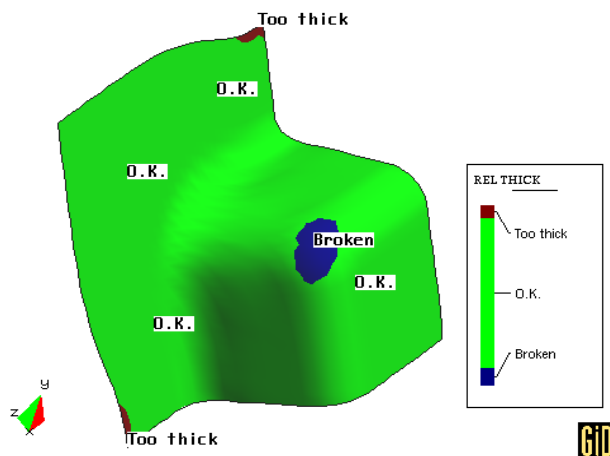
- **Minimum value:** The minimum value between two adjacent elements is the result that is used.
- **Maximum value:** The maximum value between two adjacent elements is the result that is used.
- **Mean value:** A mean value of the points between two adjacent elements is the result that is used.

Contour Ranges

Menu

A horizontal menu bar with two buttons. The first button is labeled 'View results' and the second button is labeled 'Contour ranges'. Both buttons have a light gray background and a dark gray border.

This is the same as the `Contour Fill` visualization type, but the coloured areas are created following a '`Result range table`' specified in the results file (see [Result Range Table](#)), and the names of these areas are visualized as text labels.



Example of Contour ranges

Contour Lines

Menu

View results > Contour Lines

This display option is quite similar to **Contour Fill** (see [Contour Fill](#)), but here the isolines of a certain nodal variable are drawn. In this case, each color ties several points with the same value of the variable chosen.

Here the configuration options are almost the same as the ones for **Contour Fill**, with the only difference being that the number given in the **Number of Colors** option will be used as the number of lines for this contour lines representation.

Show Minimum and Maximum

Menu

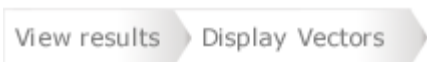
View results > Show Min Max

With this option you can see the minimum and maximum of the chosen result.

Note: This minimum and maximum can be absolute - for all the meshes/sets/cuts - or relative (local) to the ones displayed. This can be selected from the pull-down menu `Options` → `Contour` → `Set Limits`.

Display vectors

Menu



This option displays a menu with results from vectors and matrices (where the principal values have been previously evaluated by the program). From the menu of variables, choose the one you wish to see displayed; it will be shown with the default analysis and step (this can be changed with the `Default Analysis/Step` option in the menu). Once a result is chosen, the program will display the nodal vectors of the chosen result. The vectors that are drawn can be scaled interactively. The factor can be applied several times and every time it changes to the new input value.

You can modify the colour of the vectors, which by default are drawn in green, so that all the vectors are drawn in the same colour, or you can let the color of the vectors vary according to the result module.

When drawing a matrix result (3x3 symmetrical), such as the stress tensor, only a single color representation of principal values is available: blue when negative (also drawn as `>--<` representing compressions), and red when positive (also drawn as `<-->` representing tensions).

A vector will be separated into its X, Y, and Z components and its module. So either the X, Y or Z component or the whole vector (represented by the module) can be drawn. Symmetrical matrices will be unfolded into: `Si` component, `Sii` component, `Siii` component, and 'All' components. Any of these components can be selected to be visualized. `Si`, `Sii` and `Siii` represent the eigen values and vectors of the matrix results which are calculated by GiD, and which are ordered according to the eigen value.

Several configuration options can be accessed via the `Options` menu.

Menu



Color Mode: This option lets you choose if vectors are drawn in one color (`Mono Color`) or in several colors (`Color Modules`).

Number of Colors: If you use `Color Modules` to draw vectors, it is possible to choose the number of colors.

Offset: With this option, you control where the vector should be in relation of the node, ranging from 0, to tie the tail of the arrow to the node, to 1, to tie the tip of the arrow to the node.

Note: These options have no effect when viewing a matrix result.

Color (mono): When vectors are drawn in (`Mono Color`) `Color Mode`, they are drawn in green. You can change this green color selecting this entry.

Detail: The level of detail can be adjusted to draw the vectors quicker or nicer. The different options are:

- **Point:** the arrow head will be drawn as a point. The style used to draw these point like arrow heads can be modified through the point options window described before (see [Point and Line options](#)).
- **Lines:** the arrow head will be drawn as four lines.
- **2 Triangles:** the arrow head will be drawn with two intersecting triangles.
- **4 Triangles:** the arrow head will be drawn as cones of four triangles.
- **8 Triangles:** the arrow head will be drawn as cones of eight triangles.

Note: This last option affects also matrix and local axes results.

Isosurfaces

Menu



Here a surface is drawn that ties a fixed value inside a volume mesh; for surface meshes a line is drawn. To create isosurfaces there are several options:

- **Exact:** After choosing a result or a result component of the current analysis and step, you can input several fixed values and then for each given value an isosurface is drawn.

- **Automatic:** Similarly, after choosing a result or a result component, you are asked for the number of isosurfaces to be created. GiD calculates the values between the Minimum and the Maximum (these are not included).
- **Automatic Width:** After choosing a result or result component, you are asked for a width. This width is used to create as many isosurfaces as are needed between the Minimum and Maximum defined values (these are included).

Several configuration options can be accessed via the `Options` menu.

Menu



- **Display Style:** For isosurfaces there is also a `Display Style` option, like for Volumes/Surfaces/Cuts. The available options are: `All Lines`, `Hidden Lines`, `Body`, `Body Lines` (see [Display Style](#)).
- **Render:** The render option for isosurfaces renders the surfaces `Flat` or `Normal` (see [Display Style](#)).
- **Transparency:** The isosurfaces can be set to `Transparent` or `Opaque` (see [Display Style](#)).
- **Convert to cuts:** Another interesting option is `Convert To Cuts`. With this options all the isosurfaces drawn will be turned into cuts, so that they can be saved, read, and have results drawn over them.

Stream Lines

Menu



With this option you can display a stream line, or in fluid dynamics, a particle tracing, in a vector field. After choosing a vector result, using the default analysis and step selected, the program asks you for a point from which to start plotting the stream line. This point can be given in several ways:

- **Clicking on the screen:** The point will be the intersection between the line orthogonal to the screen and the plane parallel to the screen and containing the center of rotation.
- **Joining a node:** The selected node will be used as a start point.
- **Selecting nodes:** Here several nodes can be selected to start with.

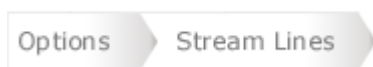
- **Along a line:** With this option you can define a segment along which several start points will be chosen. The number of points will also be asked for, including the ends of the segment. In the case of just one start point, this will be the center of the segment.
- **In a quad:** Here you can enter four lines that define a quadrilateral area which will be used to create a $N \times M$ matrix of points. These points will be the start for the stream lines. When giving $N \times M$, N lies on the first and third line, and M on the second and fourth. So, points $(0, 0..N)$, $(M, 0..N)$, $(0..M, 0)$ and $(0..M, N)$ will lie on the lines. But in case of $N=1$ or $M=1$, this will be the center of the line, and if $N = 1$ and $M = 1$, this will be the center of the Quad.
- **Intersect set:** The point will be the intersection between the line orthogonal to the screen and the current viewed sets nearest the viewpoint.

When viewing `Stream Lines`, labels can be drawn to show the times at the start and end points of the stream line.

Stream lines can also be deleted and their color changed (green by default).

The following options can be chosen for Stream Lines:

Menu



Color: The color of the stream lines.

Delete: This option lets you select the stream lines to be deleted.

Label: This option lets you select the kind of label:

- **None:** No labels are drawn.
- **0_End:** Labels are drawn according to the following convention: 0 at the start of the stream line, and the total time taken for the particle to travel at the end.
- **Ini_End:** Labels are drawn according to the following convention: 0 at the chosen point, with - time before at the beginning of the stream line and + time after at the end.

Size & detail: Here you can adjust the size, width and detail of the stream lines using the `Stream line options` window (see [Point and Line options](#)).

Line diagrams

Menu



This result visualization option is only active when line elements are used in the mesh, and will only be represented over these line elements. When using this result visualization option, graph-style lines will be drawn over the line elements.

When drawing a **Scalar Diagram**, the graph-style lines are drawn on a plane parallel to the screen (with its normal vector pointing out of the screen) when this result view is selected. The positive 'axis' will be the vector resulting from the cross product between this normal vector and the one that the line defines.

When drawing a **Vector Diagram**, the graph-style lines are drawn on a plane that includes the result vector and the vector that the line defines. The graph-style lines represent the module of this vector. The positive 'axis' is also defined by the result vector. As modules are positive, to allow negative values, the input format for vector results allows the introduction of a fourth component: the signed vector module (see [Postprocess results format: ProjectName.post.res, ProjectName.flavia.res](#)).

There is a Show Elevations option only accessible through the **Right buttons** menu under Results→Line Diagram→Options. Elevations are lines that connect the nodes and the gauss points of the line element and the graph-style line that represents the result. The options are:

- **None:** to switch the elevations off;
- **Nodes only:** to draw the elevation lines only on the nodes;
- **Whole line:** to draw the elevation lines along the whole line, using nodes and gauss points;
- **Filled line:** to draw orange filled elevations;
- **Contour filled line:** the colors used to draw the filled elevations will be the same as a contour fill done along the line.

Legends

Menu



Legends appear when contour visualization, isosurfaces or color vector visualization is used:

- **Show:** Legends can be switched **On** or **Off**.
- **Opaque:** Legends can be transparent, showing the result visualization behind them, or opaque.

- **Show title:** With this option the result name of the current visualization appears at the top of the legend.
- **Outside:** Activating this option, the legend is shown in a separate window, thus leaving more space in GiD's windows.
- **Automatic comments:** If this option is activated, information about the type of analysis, the actual step and the kind of result, is added at the bottom of the screen (see [Automatic comments](#)).

Graphs

Menu



Menu



Graph Lines description

Here you can draw graphs in order to take a closer look at the results. Several graph types are available: “point evolution against time”, “result 1 vs. result 2 over points”, and “result along a boundary line”. You can also save or read a graph (see [Files menu](#)). The format of the file will be described later (see [Graph Lines File Format](#)).

In the `View Results`→`Graphs` pull-down menu, there are the following options:

- `Show`: Here you can switch between the graphs view and the postprocessing view.
- `ClearGraphs`: To reset all the graphs and start again.
- `Point Evolution`: This graph shows the evolution of a result on a point along all the steps in the current analysis.
- `Point Graph`: After choosing a point or points, you can contrast a result against another.
- `Border Graph`: After selecting a border, you can see how the results vary along this boundary. You can also use the `Border Graph` window (`Windows` menu) to configure this type of graphs.

You can view labels for the points of a graph. These labels include not only the graph point's number, but also its X and Y values. If some points on the graph are labeled, when returning to the normal results view, the labels also appear in the results view.

Graph Lines options

Menu

A rectangular button with a light gray gradient and a dark gray border, containing the word "Options" in a sans-serif font.A rectangular button with a light gray gradient and a dark gray border, containing the word "Graphs" in a sans-serif font.

- **Grids:** Tell GiD whether or not to draw grids.
- **Current Style:** Choose what the new graphs should look like. The possible styles are: Dot, Line, and Dot-Line.
- **Change Style Graph:** Change the style of the selected graph.
- **Change Colour Graph:** Change the colour of the selected graph.
- **Change Line Width Graph:** Change the width of the graph lines.
- **Change Line Pattern Graph:** Switch between different line patterns, useful with a b/w printer.
- **Change Point Size Graph:** Change the point size for Dot and Dot-Line styles.
- **Change Title Graph:** Change the title of the selected graph.
- **Title:** Change the title, change its position, or reset its value.
- **X Axis:** Set min and max values and divisions for the X-axis, reset them, and change the label.
- **Y Axis:** Set min and max values and divisions for the Y-axis, reset them, and change the label.

Graph Lines File Format

The graph file that GiD uses is a standard ASCII file.

Every line of the file is a point on the graph with X and Y coordinates separated by a space.

Comment lines are also allowed and should begin with a '#'.

The title of the graph and the labels for the X- and Y-axes can also be configured.

If a comment line contains the keyword 'Graph: ' the string between quotes that follows this keyword will be used as the title of the graph. The string between quotes that follows the keywords 'X: ' and 'Y: ' will be used as labels for the X- and Y-axes respectively.

Example:

```
# Graph: "Nodes 26, 27, 28, ... 52 Graph."
#
# X: "Szz-Nodal_Streess" Y: "Sxz-Nodal_Stress"
-3055.444 1672.365
-2837.013 5892.115
-2371.195 666.9543
-2030.643 3390.457
-1588.883 -4042.649
-1011.5 1236.958
# End
```

Result surface

This option uses a result component, or a scalar value, and draws a 3D surface above the mesh following the normals of this mesh. It can be seen as an extrusion of the mesh along its normals with the result as factor, like the beam diagrams but with surfaces.

With the `Show elevations` option inside the menu `Options→Result Surface`, you can choose how the elevations (lines or faces that connect the result surface with the underlying mesh) are drawn:

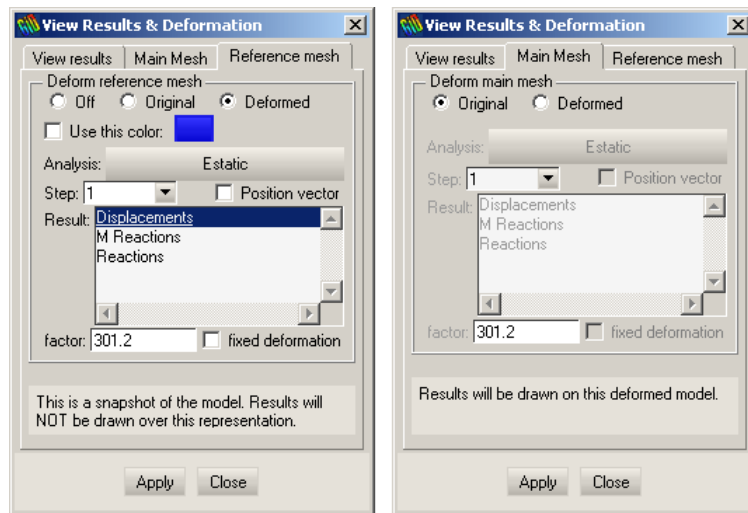
- **None:** The mesh and its result surface are drawn separately, without interconnecting lines or faces.
- **Extruded nodes:** Lines are drawn between the original nodes of the mesh and the extruded ones, i.e. the nodes of the result surface.
- **Extruded edges:** Faces are drawn between the original edges (of the elements) of the mesh and the extruded ones, i.e. the edges of the result surface.
- **Contour fill:** Switches **On** or **Off** the contour fill representation of the result used to create the 3D surface.

Deform Mesh

Volumes, surfaces and cuts can be deformed according to a nodal vector and a factor. When doing this all the results are drawn on the deformed volumes, surfaces and cuts. In GiD this is called `Main Geometry`. Thus, when the `Main Geometry` is deformed, results are also drawn distorted; and when `Main geometry` is in its original state, results also drawn in their original state.

The `View results` window allows you to do this.

Menu



Deformation window

In the upper part of this window, you choose between the `Original` state of the `Main Geometry` and the `Deformed` state, for which a nodal vectorial result, and an analysis and step, must be selected and a factor entered.

There is also a `Reference Geometry` option. This lets you visualize volumes, surfaces or cuts like `Main Geometry` do, but **NO** results can be displayed over these volumes, surfaces or cuts. It is merely provided as a reference, to contrast several deformations or changes in the original geometry. Remember that the current `Mesh Display Style` will be used for the `Reference Geometry` and it can only be changed when redoing the mesh.

In the lower part of the window, the `Reference Geometry` can be configured. You can choose between:

- `Off`, so this reference visualization is not displayed;

- **Original**, if Main Geometry is deformed and you wish to compare it to its original state without losing a results representation;
- **Deformation**, where after providing an analysis, step, result and factor, you can use it to contrast two deformation states, or a deformed state and an original geometry; or
- **Color**, where the color of the reference meshes can be specified as the same as, or different from, the original meshes.

Do Cuts

Menu



Here you can cut and divide volumes, surfaces and cuts. A cut of a volume mesh results in a cut plane. The cut is done for all the meshes, even those that are switched **Off**. When cutting surfaces, a line set will be created. Here only those surfaces that are switched **On** are cut.

Another feature is that a cut can be deformed, if meshes are also told to do so (see [Deform Mesh](#)). A cut of a deformed mesh, when changing to the original shape, will be deformed accordingly.

When dividing volumes/surfaces, only those elements of the volumes/surfaces that are switched **On** and which lie on one side of a specified plane will be used to create another volume/surface.

- **Cut Plane:** Specify a plane which cuts the volumes/surfaces. Several options can be used to enter this plane.
 - **2 Points:** the plane is defined by two points and the visual direction orthogonal to the screen.
 - **3 Points:** the plane is defined by three points. When choosing the points, the nodes of the mesh can also be used.
 - **Succession:** this option is an enhancement of **Cut Plane**. Here you specify an axis that will be used to create cut planes orthogonal to this axis. The number of planes is also asked for.
- **Divide by selection:** To send the selected elements to an old or new set.
- **Divide Volume Sets:** Specify a plane which is used to divide the meshes. Several options can be used to enter this plane. With **Two Points** (the default), the plane is defined by the corresponding line and the direction orthogonal the screen. With **Three Points**, the plane is defined by three points. When choosing the points, the nodes of the mesh can also be used. After defining the plane, you should choose which section of the mesh should be saved by selecting the side of the plane to use. Only those

elements that lie entirely on this side are selected. Only those volume meshes that are shown are divided.

- **Divide Surface Sets:** Specify a plane which is used to divide the sets. Several options can be used to enter this plane. With `Two Points` (the default), the plane is defined by the corresponding line and the direction orthogonal to the screen. With `Three Points`, the plane is defined by three points. When choosing the points, the nodes of the mesh can also be used. After defining the plane, you should choose which section of the mesh should be saved by selecting the side of the plane to use. Only those elements that lie entirely on this side are selected. Only those surface meshes that are shown are divided. When doing a division, there are several useful options inside the **Contextual** mouse menu (right-click in the graphical window):
 - `exact`: to do an exact division, i.e. elements are cut to create the division;
 - `parallel planes`: the remaining elements will be the ones between two parallel planes. A distance can also be entered, after choosing this option from the **Contextual** menu.
- **Divide lines:** Specify a plane which is used to get the lines on one side of this plane
- **Cut Wire:** Here you can define a 'wire' tied to the edges of the elements of the volumes/surfaces. So when the volumes/surfaces are deformed, the wire is deformed too. And vice versa, if a wire is defined while the volumes/surfaces are deformed, when turning them into its original shape, the cut-wire is "undeformed" with them.
- **Cut Spheres:** Here you can define a sphere which will be used to cut the volume and surface meshes, resulting in triangle or line cut meshes respectively.
- **Convert cuts to surface sets:** with these options cuts can be converted to surface sets so they can be saved, or cut again.

Cuts can also be read from and written to a file. The information stored to the file from a 'Cut Plane' is the plane equation that defines the plane ($Ax + By + Cz + D = 0$), so it can be used with several models. The information stored in the file from a 'Cut Wire' is the points list of the cut-wire, i.e. the intersection between the wire and the edges of the meshes/sets. These files are standard ASCII files. A line of a 'Cut Plane' archive contains the four coefficients of the plane that defines the cut, separated by spaces. A line of a 'Cut Wire' archive contains the three coordinates of a point of the wire. Comment lines are allowed and should begin with a '#'.

An example of a 'Cut Plane' file where three planes were written:

```
# planes created from a 'cut succession'
-10.82439 0.5740206 0 51.62557
-10.82439 0.5740206 0 12.45994
-10.82439 0.5740206 0 -26.70569
```

An example of a 'Cut Wire' file:

```

-2.444425 3.883427 2.487002
2.130787 2.762815 3.885021
0.8411534 4.458836 3.215301
4.270067 3.795048 2.037187
5.66561 3.414776 0.8219391
2.945865 3.600701 3.29012
0.4487007 3.764661 3.574121

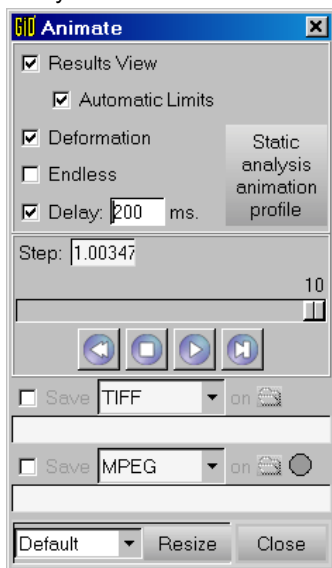
```

Animation

Menu



With this window a little bit of automatization has been done to create animations inside GiD. Nowadays, almost all results visualizations are animated. Only stream-lines are not automatized along all the steps of the current analysis.



Animation Window

This window lets you create an animation of the current `Results View`, where the limits can be fixed along the animation with `Automatic Limits`, and/or an animation of the Deformation of the meshes. To the right of the `Step:` label, the step value is shown. On the slide bar, the step number is shown.

The four buttons under the slide bar are self-explanatory: they 'Rewind', 'Stop', 'Play' and 'Step' the animation. Clicking on the slide bar will rewind or advance the animation. The green LED, which indicates that an animation is ready, will change to red while the animation is being saved to a file. This LED will change back to green when the animation is finished, or the `Stop` button is pressed.

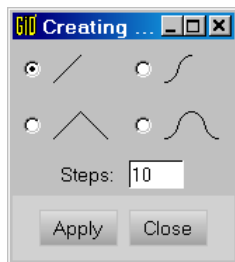
Options are:

- `Automatic Limits`: GiD searches for the minimum and maximum values of the results along all the steps of the analysis and uses them to draw the results view through all the steps.
- `Endless`: The animation continues indefinitely.
- `Delay`: Specify a delay time between steps in milliseconds.
- `Save TIFF/JPEG/GIFs on`: Save snapshots, in TIFF, JPEG or GIF format, of each step when the `Play` button is pushed. Here the filename given will be used as a prefix to create the TIFF/JPEG/GIFs; for instance, if you write `MyAnimation`, TIFF/JPEG/GIF files will be created with names `MyAnimation-01.tif/.jpg/.gif`, `MyAnimation-02.tif/.jpg/.gif`, and so on.
- `Save MPEG/Avi mjpeg/AVI MS Video 1/AVI raw True Color/AVI raw 15 bpp (VD)/AVI raw 16 bpp (MS)/GIF on`: Entering a filename here, a MPEG/AVI/GIF file will be created when the `Play` button is pressed.

Note: To avoid problems when trying to view an MPEG format animation in *Microsoft Windows*, it is strongly recommended that you use the `Default` menu to select a 'standard' size and press the `Resize` button. The graphical window will change to this 'standard' size. After finishing the animation, simply select `Default` on the menu and press the `Resize` button, and the previous size will be restored.

Note: AVI MS Video 1 uses a simple video compression algorithm, commonly supported by all video players.

- `Static analysis animation profile`: If the project has only one step, it is possible to simulate an animation by generating intermediate steps. Use this option to automatically generate several frames, following a lineal, cosine, triangular or sinusoidal interpolation between the normal state and the deformed state.



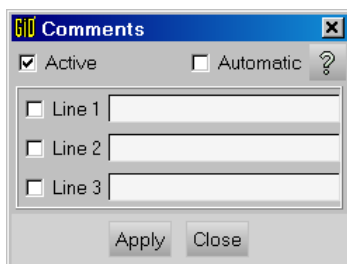
Profile factor for pseudo-animation

Automatic comments

Menu



While displaying results, comments can be automatically generated by switching **On** the **Automatic** checkbox in the **Comments** window, which appears by selecting **Utilities**→**Graphical**→**Comments**.



If this option is selected and the comment lines are empty, the program will create its own automatic comments, like these ones:

Load Analysis, step 3

Display Vectors of Displacements, |Displacements| factor 68095.4

Deformation (x127.348): Displacements of Load Analysis 2, step 8

where the fields which will change when the visualization changes are marked in bold.

You can also create your own automatic comments, just by filling in the comment lines. To tell the program where the result name, analysis name, etc. should be placed, the following fields can be used:

- `%an`: for the analysis name of the current visualized result
- `%sv`: for the step value of the current visualized result
- `%vt`: for the result visualization type of the current visualized result
- `%vf`: for the vectors factor used in the current visualized result (if any)
- `%rn`: for the result name of the current visualized result
- `%cn`: for the component name of the current visualized result
- `%da`: for the analysis name of the current deformation (if any)
- `%ds`: for the step value of the current deformation (if any)
- `%dr`: for the result name of the current deformation (if any)
- `%df`: for the factor of the current deformation (if any)

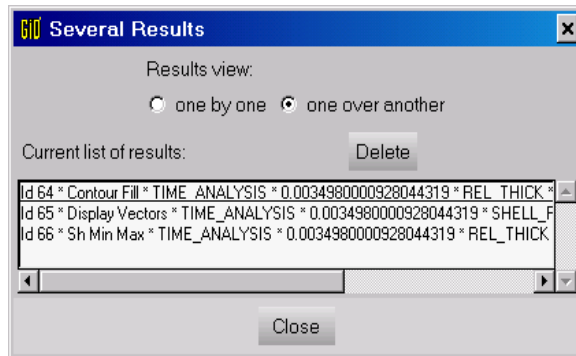
GiD will substitute the fields for the values of the current visualization. Fields which cannot be filled will be empty.

Several results

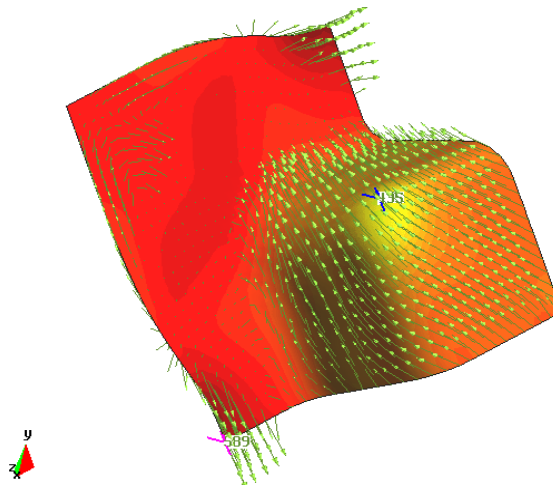
Menu



With this window, which appears under `Windows` → `Several Results`, you can select whether to view the results one by one, as usual, or to view some results visualization types at once, e.g. a contour fill of pressure and velocity vectors at the same time. From this window you can also delete the undesired results visualizations. After selecting the desired behavior, press the `Apply` button.



Several results window



Example showing contour fill, vectors and the minimum and maximum visualization types

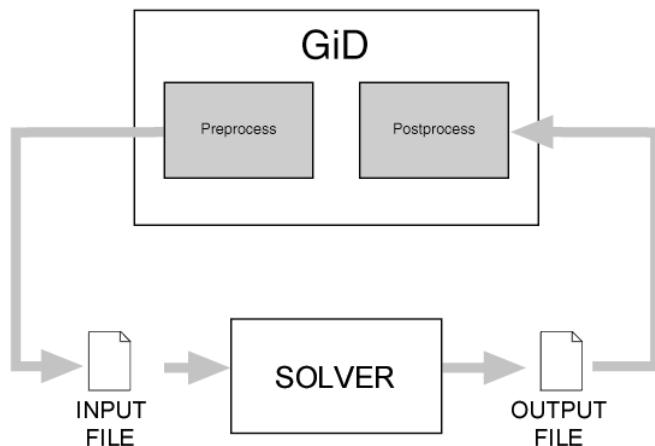
CUSTOMIZATION

Introduction

When GiD is to be used for a particular type of analysis, it is necessary to predefine all the information required from the user and to define the way the final information is given to the solver module. To do so, some files are used to describe conditions, materials, general data, units systems, symbols and the format of the input file for the solver. We give the name **Problem Type** to this collection of files used to configure GiD for a particular type of analysis.

Note: You can also learn how to configure GiD for a particular type of analysis by following the **Problem Type Tutorial**; this tutorial is included with the GiD package you have bought. You can also download it from the GiD support web page (<http://www.gidhome.com/support>).

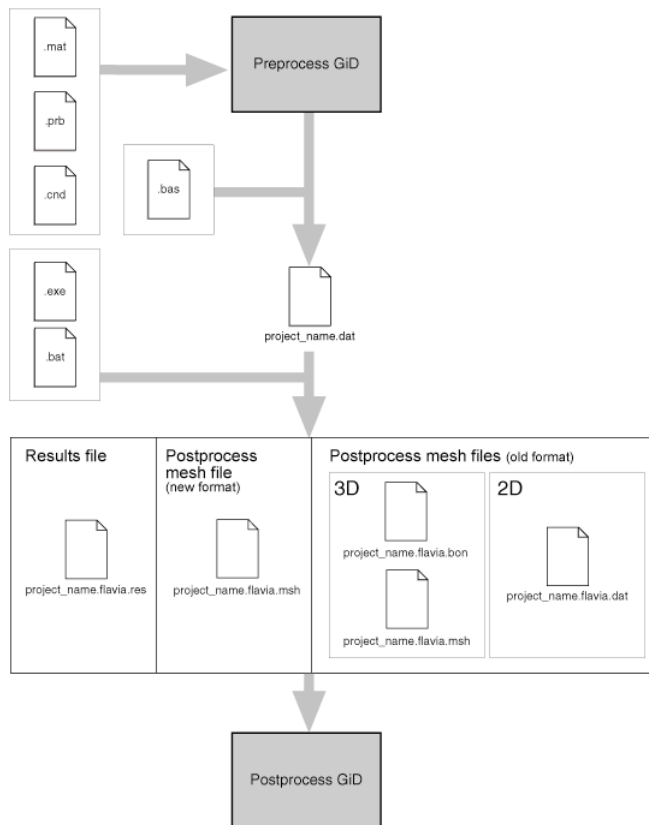
GiD has been designed to be a general-purpose Pre- and Postprocessor; consequently, the configurations for different analyses must be performed according to the particular specifications of each solver. It is therefore necessary to create specific data input files for every solver. However, GiD lets you perform this configuration process inside the program itself, without any change in the solver, and without having to program any independent utility.



To configure these files means defining the data that must be input by the user, as well as the materials to be implemented and other geometrical and time-dependent conditions. It is also possible to add symbols or drawings to represent the defined conditions. GiD offers the opportunity to work with units when defining the properties of the data mentioned above, but there must be a configuration file where the definition of the units systems can be found. It is

also necessary to define the way in which this data is to be written inside the file that will be the input file read by the corresponding solver.

The creation of a **Problem Type** involves the creation of a directory with the name of the problem type and the extension `.gid`. This directory can be located in the current working directory or the main GiD executable directory. The former can be useful during the development of the project. Once it is finished, it may be advisable to move the directory to the one where GiD is stored; in this way, your problem type will be added to those included in the system and it will appear in the GiD menu (see [Problem type](#)). In both cases, the series of files must be inside the problem type directory. The name for most of them will follow the format `problem_type_name.xxx` where the extension refers to their particular function. Considering `problem_type_name` to be the name of the problem type and `project_name` the name of the project, file configuration is described by the following diagram:



- **Directory name:** `problem_type_name.gid`
- **Directory location:** `c:\a\b\c\GiD_directory\problemtypes`
- **Configuration files**
 - `problem_type_name.xml` XML-based configuration
 - `problem_type_name.cnd` Conditions definitions
 - `problem_type_name.mat` Materials properties
 - `problem_type_name.prb` Problem and intervals data
 - `problem_type_name.uni` Units Systems
 - `problem_type_name.sim` Conditions symbols
 - `***.geo` Symbols geometrical definitions
 - `***.geo` Symbols geometrical definitions ...
- **Template files**
 - `problem_type_name.bas` Information for the data input file
 - `***.bas` Information for additional files
 - `***.bas` Information for additional files ...
- **Tcl extension files**
 - `problem_type_name.tcl` Extensions to GiD written in the Tcl/Tk programming language
- **Command execution files**
 - `problem_type_name.bat` Operating system shell that executes the analysis process

The files `problem_type_name.sim`, `***.geo` and `***.bas` are not mandatory and can be added to facilitate visualization (both kinds of file) or to prepare the data input for restart in additional files (just `***.bas` files). In the same way `problem_type_name.xml` is not necessary; it can be used to customize features such as: version info, icon identification, password validation, etc.

Configuration Files

These files generate the conditions and material properties, as well as the general problem and intervals data to be transferred to the mesh, at the same time giving you the chance to define geometrical drawings or symbols to represent some conditions on the screen.

XML file

The file `problem_type.xml` contains information related to the configuration of the problem type, such as file browser, icon, password validation or message catalog location. Besides this, the file can be used to store assorted structured information such as version number, news added from the last version, and whatever the developer decides to include. This file can be read using the Tcl extension `tcom` which is provided with **GiD**.

The data included inside the xml file should observe the following structure:

```
<Infoproblemtype version="1.0">
  <Program>
  </Program>
</Infoproblemtype>
```

We suggest that the following nodes are included (the values of these nodes are just examples):

- `<Name>Nastran 2.4</Name>` to provide a long name for the problem type.
- `<Version>2.4</Version>` dotted version number of the problem type.
- `<MinimumGiDVersion>8.0</MinimumGiDVersion>` to state the minimum **GiD** version required.
- `<ImageFileBrowser>images/ImageFileBrowser.gif</ImageFileBrowser>` icon image to be used in the file browser to show a project corresponding to this problem type. The recommended dimensions for this image are 17x12 pixels.
- `<MsgcatRoot>scripts/msgs</MsgcatRoot>` a path, relative or absolute, indicating where the folder with the name `msgs` is located. The folder `msgs` contains the messages catalog for translation.
- `<PasswordPath>..</PasswordPath>` a path, relative or absolute, indicating where to write the password information see [ValidatePassword node](#)).
- `<ValidatePassword></ValidatePassword>` provides a custom validation script in order to override the default **GiD** validation (see [ValidatePassword node](#)).

ValidatePassword node

The default action taken by **GiD** when validating a problem type password is verifying that it is not empty. When a password is considered as valid, this information is written in the file `password.txt` which is located in the problem type directory. In order to override this behavior, two nodes are provided in the `.xml` file

- **PasswordPath**: The value of this node specifies a relative or absolute path describing where to locate/create the file `password.txt`. If the value is a relative path it is taken with respect to the problem type path.

Example:

```
<PasswordPath>../PasswordPath>
```

- **ValidatePassword**: The value of this node is a Tcl script which will be executed when a password for this problem type needs to be validated. The script receives the parameters for validation in the following variables:

`key` with the contents of the password typed,

`dir` with the path of the problem type, and

`computer_name` with the name of host machine.

Note: It is like this Tcl procedure prototype:

```
proc PasswordPath { key dir computer_name } { ... body... }
```

The script should return one of three possible codes:

0 in case of failure.

1 in case of success.

2 in case of success; the difference here is that the problem type has just saved the password information so **GiD** should not do it.

Furthermore, we can provide a description of the status returned for **GiD** to show to the user. If another status is returned, it is assumed to be 1 by default.

Below is an example of a `<ValidatePassword>` node.

```
<ValidatePassword>
  #validation.exe simulates an external program to validate the
  key for this computername
  #instead an external program can be used a tcl procedure
  if { [catch {set res [exec [file join $dir validation.exe]
    $key $computername]} msgerr] } {
    return [list 0 "Error $msgerr"]
  }
}
```

```

switch -regexp -- $res {
  failRB {
    return [list 0 "you ask me to fail!"]
  }
  okandsaveRB {
    proc save_pass {dir id pass} {
      set date [clock format [clock second] -format "%Y %m
%d"]
      set fd [open [file join $dir .. "password.txt"] "a"]
      puts $fd "$id $pass    # $date Password for Problem type
'$dir'"
      close $fd
    }
    save_pass $dir $computername $key
    rename save_pass ""
    return [list 2 "password $key saved by me"]
  }
  okRB {
    return [list 1 "password $key will be saved by gid"]
  }
  default {
    return [list 0 "Error: unexpected return value $res"]
  }
}
</ValidatePassword>

```

Conditions file (.cnd)

Files with extension `.cnd` contain all the information about the `conditions` that can be applied to different entities. The condition can adopt different field values for every entity. This type of information includes, for instance, all the displacement constraints and applied loads in a structural problem or all the prescribed and initial temperatures in a thermal analysis.

An important characteristic of the conditions is that they must define what kind of entity they are going to be applied over, i.e. over points, lines, surfaces, volumes or layers, and what kind of entity they will be transferred over, i.e. over nodes, over face elements or over body elements.

- **Over nodes** This means that the condition will be transferred to the nodes contained in the geometrical entity where the condition is assigned.

- **Over face elements ?multiple?** If this condition is applied to a line that is the boundary of a surface or to a surface that is the boundary of a volume, this condition is transferred to the higher elements, marking the affected face. If it is declared as **multiple**, it can be transferred to more than one element face (if more than one exists). By default it is considered as **single**, and only one element face will be marked.
- **Over body elements** If this condition is applied to lines, it will be transferred to line elements. If assigned to surfaces, it will be transferred to surface elements. Likewise, if applied to volumes, it will be transferred to volume elements.

Note: For backwards compatibility, the command `over elements` is also accepted; this will transfer the condition either to elements or to faces of higher level elements.

Another important feature is that all the conditions can be applied to different entities with different values for all the defined intervals of the problem.

Therefore, a condition can be considered as a group of fields containing the name of the particular condition, the geometric entity over which it is applied, the mesh entity over which it will be transferred, its corresponding properties and their values.

The format of the file is as follows:

```
CONDITION: condition_name
CONDTYPE: 'over' ('points', 'lines', 'surfaces', 'volumes', 'layer')
CONDMESHTYPE: 'over' ('nodes', 'face elements', 'face elements
multiple', 'body elements')
QUESTION: field_name['#CB#'(...,optional_value_i,...)]
VALUE: default_field_value
...
QUESTION: field_name['#CB#'(...,optional_value_i,...)]
VALUE: default_field_value
END CONDITION

CONDITION: condition_name
...
END CONDITION
```

Note: #CB# means Combo Box.

Note that this file format does not allow you to put blank lines between the last line of a condition definition, `END CONDITION`, and the first one of the next condition definition.

Local Axes

QUESTION: field_name['#LA#('global', 'automatic', 'automatic alternative')]

VALUE: default_field_value

This type of field refers to the **local axes** system to be used. The position of the values indicates the kind of **local axes**.

If it only has a single default value, this will be the name of the global axes. If two values are given, the second one will reference a system that will be computed automatically for every node and will depend on geometric constraints, like whether or not it is tangent, orthogonal, etc. If a third value is given, it will be the name of the automatic alternative axes, which are the automatic axes rotated 90 degrees.

All the different user-defined systems will automatically be added to these default possibilities.

To enter only a specific kind of local axes it is possible to use the modifiers #G#, #A#, #L#.

- #G#: global axes;
- #A#: automatic axes;
- #L#: automatic alternative axes.

When using these modifiers the position of the values does not indicate the kind of local axes.

Example:

QUESTION: Local_Axes#LA#(Option automatic#A#,Option automatic_alt#L#)

VALUE: -Automatic-

Note: All the fields must be filled with words, where a word is considered as a string of characters without any blank spaces. The strings signaled between quotes are literal and the ones inside brackets are optional. The interface is case-sensitive, so any uppercase letters must be maintained. The `default_field_value` entry and various `optional_value_i` entries can be alphanumeric, integers or reals. GiD treats them as alphanumeric until the moment they are written to the solver input files.

Note: The numbers of the conditions must be consecutive, beginning with number 1. There is no need to point out the overall number of conditions or the respective number of fields for each one. This last number can vary for each condition.

One flag that can optionally be added to a condition is:

```
CANREPEAT: yes
```

It is written after `CONDMESHTYPE` and means that one condition can be assigned to the same entity several times.

Another type of field that can be included inside a condition is:

```
QUESTION: Surface_number#FUNC# (NumEntity)
VALUE: 0
```

where the key `#FUNC#`, means that the value of this field will be calculated just when the mesh is generated. It can be considered as a function that evaluates when meshing. In the above example, `NumEntity` is one of the possible variables of the function. It will be substituted by the label of the geometrical entity from where the node or element is generated.

```
QUESTION: X_press#FUNC# (Cond(3,REAL) * (x-Cond(1,REAL)) / (Cond(2,REAL) -
Cond(1,REAL)) )
VALUE: 0
```

In this second example, the `x` variable is used, which means the X-coordinate of the node or of the center of the element. Others fields of the condition can also be used in the function. Variables `y` and `z` give the Y- and Z-coordinates of this point.

Note: There are other options available to expand the capabilities of the `Conditions` window (see [Special fields](#)).

Example: Creating the conditions file

Here is an example of how to create a conditions file, explained step by step:

1. First, you have to create the folder or directory where all the problem type files are located, `problem_type_name.gid` in this case.
2. Then create and edit the file (`problem_type_name.cnd` in this example) inside the recently created directory (where all your problem type files are located). As you can see, except for the extension, the names of the file and the directory are the same.
3. Create the first condition, which starts with the line:

```
CONDITION: Point-Constraints
```

The parameter is the name of the condition. A unique condition name is required for this conditions file.

4. This first line is followed by the next pair:

```
CONDTYPE: over points
CONDMESHTYPE: over nodes
```

which declare what entity the condition is going to be applied over. The first line, `CONDTYPE: . . .` refers to the geometry, and may take as parameters the sentences "over points", "over lines", "over surfaces" or "over volumes".

The second line refers to the type of condition applied to the mesh, once generated. GiD does not force you to provide this second parameter, but if it is present, the treatment and evaluation of the problem will be more accurate. The available parameters for this statement are "over nodes" and "over elements".

5. Next, you have to declare a set of questions and values applied to this condition.

```
QUESTION: Local-Axes#LA# (-GLOBAL-)
VALUE: -GLOBAL-
QUESTION: X-Force
VALUE: 0.0
QUESTION: X-Constraint:#CB# (1,0)
VALUE: 1
QUESTION:
X_axis:#CB# (DEFORMATION_XX,DEFORMATION_XY,DEFORMATION_XZ)
VALUE: DEFORMATION_XX

END CONDITION
```

After the `QUESTION:` prompt, you have the choice of putting the following kinds of word:

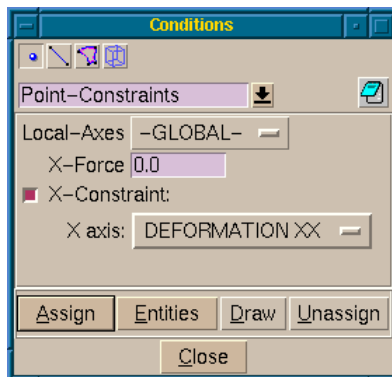
- An alphanumeric field name.
- An alphanumeric field name followed by the `#LA#` statement, and then the single or double parameter.
- An alphanumeric field name followed by the `#CB#` statement, and then the optional values between parentheses.

The `VALUE :` prompt must be followed by one of the optional values, if you have declared them in the previous `QUESTION :` line. If you do not observe this format, the program may not work correctly.

In the previous example, the `X-Force` `QUESTION` takes the value 0.0. Also in the example, the `X-Constraint` `QUESTION` includes a Combo Box statement (`#CB#`), followed by the declaration of the choices 1 and 0. In the next line, the value takes the parameter 1. The `X_axis` `QUESTION` declares three items for the combo box: `DEFORMATION_XX`, `DEFORMATION_XY`, `DEFORMATION_XZ`, with the value `DEFORMATION_XX` chosen.

Beware of leaving blank spaces between parameters. If in the first question you put the optional values (`-GLOBAL`, `-AUTO-`) (note the blank space after the comma) there will be an error when reading the file. Take special care in the Combo Box question parameters, so as to avoid unpredictable parameters.

6. The conditions defined in the `.cnd` file can be managed in the `Conditions` window (found in the `Data` menu) in the Preprocessing component of GiD.
- 7.



Conditions window in Preprocessing, showing an unfolded Combo Box

Materials file (.mat)

Files with the extension `.mat` include the definition of different materials through their properties. These are base materials as they can be used as templates during the Preprocessing step for the creation of newer ones.

You can define as many materials as you wish, with a variable number of fields. None of the unused materials will be taken into consideration when writing the data input files for the solver. Alternatively, they can be useful for generating a materials library.

Conversely to the case of conditions, the same material can be assigned to different levels of geometrical entity (lines, surfaces or volumes) and can even be assigned directly to the mesh elements.

In a similar way to how a condition is defined, a material can be considered as a group of fields containing its name, its corresponding properties and their values.

The format of the file is as follows:

```
MATERIAL: material_name
QUESTION: field_name['#CB#'(...,optional_value_i,...)]
VALUE: default_field_value
...
QUESTION: field_name['#CB#'(...,optional_value_i,...)]
VALUE: default_field_value
END MATERIAL

MATERIAL: material_name
...
END MATERIAL
```

If a material has a variable property (an example would be where a property was dependent on temperature and was defined with several values for several temperatures) a table of changing values may be declared for this property. When the solver evaluates the problem, it reads the values and applies a suitable property value.

The declaration of the table requires two lines of text:

The first is a QUESTION line with a list of alphanumeric values between parentheses.

```
QUESTION: field_name:(...,optional_value_i,...)
```

These values are the names of each of the columns in the table so that the number of values declared is the number of columns.

This first line is followed by another with the actual data values. It starts with the words `VALUE :` `#N#`, and is followed by a number that indicates the quantity of elements in the matrix and, finally, the list of values.

```
VALUE: #N# number_of_values ... value_number_i ...
```

The number of values declared for the matrix obviously has to be the number of columns multiplied by the number of rows to be declared.

This kind of material specification is most likely to be used in thermo-mechanical simulations, where the problem is exposed to a temperature variation, and the properties of the materials change for each temperature value.

All the fields must be filled with words, where a word is considered as a string of characters without any blank spaces. The strings signaled between quotes are literal and the ones within brackets are optional. The interface is case-sensitive, so any uppercase letters must be maintained. The `default_field_value` entry and various `optional_value_i` entries can be alphanumeric, integers or real numbers, depending on their type.

The numbers of the materials have to be consecutive, beginning with the number 1. There is no need to indicate the overall number of materials or the respective number of fields for each one. This last one can vary for each material.

Note that in this file, you cannot include blank lines between different material definitions, nor between questions and values.

Note: There are other options available to expand the capabilities of the `Materials` window (see [Special fields](#)).

Example: Creating the materials file

Here is an example of how to create a materials file, explained step by step:

1. Create and edit the file (`problem_type_name.mat` in this example) inside the `problem_type_name` directory (where all your problem type files are located). As you can see, except for the extension, the names of the file and the directory are the same.
2. Create the first material, which starts with the line:

```
MATERIAL: Air
```

The parameter is the name of the material. A unique material name is required for this into this materials file (do not use blank spaces in the name of the material).

3. The next two lines define a property of the material and its default value:

```
QUESTION: Density  
VALUE: 1.0
```

You can add as many properties as you wish. To end the material definition, add the following line:

```
END MATERIAL
```

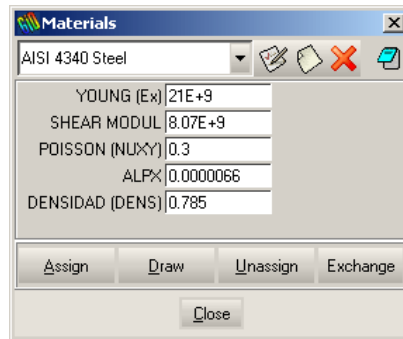
4. In this example we have introduced some materials; the .mat file would be as follows:

```
MATERIAL: Air  
QUESTION: Density  
VALUE: 1.01  
END MATERIAL
```

```
MATERIAL: AISI_4340_Steel  
QUESTION: YOUNG_(Ex)  
VALUE: 21E+9  
QUESTION: SHEAR_MODUL  
VALUE: 8.07E+9  
QUESTION: POISSON_(NUXY)  
VALUE: 0.3  
QUESTION: ALPX  
VALUE: 0.0000066  
QUESTION: DENSIDAD_(DENS)  
VALUE: 0.785  
END MATERIAL
```

```
MATERIAL: Concrete  
QUESTION: Density  
VALUE: 2350  
END MATERIAL
```

5. The materials defined in the .mat file can be managed in the **Materials** window (found in the **Data** menu) in the **Preprocessing** component of GiD.



Materials window in GiD Preprocessing

Problem and intervals data file (.prb)

Files with the extension .prb contain all the information about general problem and intervals data. The general problem data is all the information required for performing the analysis and it does not concern any particular geometrical entity. This differs from the previous definitions of conditions and materials properties, which are assigned to different entities. An example of general problem data is the type of solution algorithm used by the solver, the value of the various time steps, convergence conditions and so on.

Within this data, one may consider the definition of specific problem data (for the whole process) and intervals data (variable values along the different solution intervals). An interval would be the subdivision of a general problem that contains its own particular data. Typically, one can define a different load case for every interval or, in dynamic problems, not only variable loads, but also variation in time steps, convergence conditions and so on.

The format of the file is as follows:

```
PROBLEM DATA
QUESTION: field_name['#CB#'(...,optional_value_i,...)]
VALUE: default_field_value
...
QUESTION: field_name['#CB#'(...,optional_value_i,...)]
VALUE: default_field_value
END PROBLEM DATA

INTERVAL DATA
QUESTION: field_name['#CB#'(...,optional_value_i,...)]
```

```

VALUE: default_field_value
...
QUESTION: field_name['#CB#'(...,optional_value_i,...)]
VALUE: default_field_value
END INTERVAL DATA

```

All the fields must be filled with words, where a word is considered as a string of characters without any blank spaces. The strings signaled between quotes are literal and the ones inside brackets are optional. The interface is case-sensitive, so any uppercase letters must be maintained. The `default_field_value` entry and various `optional_value_i` entries can be alphanumeric, integers or real numbers, depending on the type.

Note: There are other options available to expand the capabilities of the Problem Data window (see [Special fields](#)).

Example: Creating the PRB data file

Here is an example of how to create a problem data file, explained step by step:

1. Create and edit the file (`problem_type_name.prb` in this example) inside the `problem_type_name` directory (where all your problem type files are located). Except for the extension, the names of the file and the directory must be the same.

2. Start the file with the line:

```
PROBLEM DATA
```

3. Then add the following lines:

```

QUESTION: Unit_System#CB#(SI,CGS,User)
VALUE: SI
QUESTION: Title
VALUE: Default_title

```

The first question defines a combo style menu called `Unit_System`, which has the `SI` option selected by default. The second question defines a text field called `Title`, and its default value is `Default_title`.

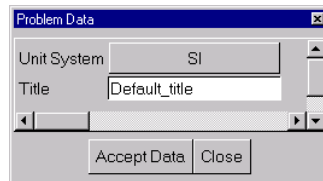
4. To end the file, add the following line:

```
END PROBLEM DATA
```

5. The whole file is as follows:

```
PROBLEM DATA
QUESTION: Unit_System#CB# (SI,CGS,User)
VALUE: SI
QUESTION: Title
VALUE: Default_title
END GENERAL DATA
```

6. The options defined in the .prb file can be managed in the Problem Data window (found in the Data menu) in the Preprocessing component of GiD.



Problem Data window in GiD Preprocessing

Conditions symbols file (.sim)

Files with the extension .sim comprise different symbols to represent some conditions during the preprocessing stage. You can define these symbols by creating ad hoc geometrical drawings and the appropriate symbol will appear over the entity with the applied condition every time you ask for it.

One or more symbols can be defined for every condition and the selection will depend on the specified values in the file, which may be obtained through mathematical conditions.

The spatial orientation can also be defined in this file, depending on the values taken by the required data. For global definitions, you have to input the three components of a vector to express its spatial direction. GiD takes these values from the corresponding conditions window. The orientation of the vector can be understood as the rotation from the vector (1,0,0) towards the new vector defined in the file.

For line and surface conditions, the symbols may be considered as local. In this case, GiD does not consider the defined spatial orientation vector and it takes its values from the line or surface orientation. The orientation assumes the vector (1,0,0) to be the corresponding entity's normal.

These components, making reference to the values obtained from the adequate conditions, may include C-language expressions. They express the different field values of the mentioned condition as `cond(type,i)`, where `type` (real or int) refers to the type of variable (not case-sensitive) and `i` is the number of the field for that particular condition.

Example: Creating the Symbols file

Here is an example of how to create a symbols file. Create and edit the file (`problem_type_name.sim` in this example) inside the `problem_type_name` directory (where all your problem type files are located). Except for the extension, the names of the file and the directory must be the same.

The contents of `problem_type_name.sim` example should be the following:

```
cond Point-Constraints
3
global
cond(int,5)
1
0
0
Support3D.geo
global
cond(int,1) && cond(int,3)
1
0
0
Support.geo
global
cond(int,1) || cond(int,3)
cond(int,3)
cond(int,1)*(-1)
0
Support-2D.geo
cond Face-Load
1
local
```

```
fabs (cond (real, 2) ) + fabs (cond (real, 4) ) + fabs (cond (real, 6) ) > 0.  
cond (real, 2)  
cond (real, 4)  
cond (real, 6)  
Normal.geo
```

This is a particular example of the .sim file where four different symbols have been defined. Each one is read from a `***.geo` file. There is no indication of how many symbols are implemented overall. GiD simply reads the whole file from beginning to end.

The `***.geo` files are obtained through GiD. You can design a particular drawing to symbolize a condition and this drawing will be stored as `problem_name.geo` when saving this project as `problem_name.gid`. You do not need to be concerned about the size of the symbol, but should bear in mind that the origin corresponds to the point (0,0,0) and the reference vector is (1,0,0). Subsequently, when these `***.geo` files are invoked from `problem_type_name.sim`, the symbol drawing appears scaled on the display at the entity's location.

Nevertheless, the number of symbols and, consequently, the number of `***.geo` files can vary from one condition to another. In the previous example, for instance, the condition called `Point-Constraints`, which is defined by using `cond`, comprises three different symbols. GiD knows this from the number 3 written below the condition's name. Next, GiD looks to see if the orientation is relative to the spatial axes (global) or moves together with its entity (local). In the example, the three symbols concerning point constraints are globally oriented.

Imagine that this condition has six fields. The first, third and fifth field values express if any constraint exist along the X-axis, the Y-axis and the Z-axis, respectively. These values are integers and in the case that they are null, the degree of freedom in question is assumed to be unconstrained.

For the first symbol, obtained from the file `Support3D.geo`, GiD reads `cond (int, 5)`, or the Z-constraint. If it is false, which means that the value of the field is zero, the C-condition will not be satisfied and GiD will not draw it. Otherwise, the C-condition will be satisfied and the symbol will be invoked. When this occurs, GiD skips the rest of the symbols related to this condition. Its orientation will be the same as the original drawing because the spatial vector is (1,0,0).

All these considerations are valid for the second symbol, obtained from the file `Support.geo`, but now GiD has to check that both constraints (&&) - the X-constraint and the Y-constraint - are fixed (their values are not zero).

For the third symbol, obtained from the file `Support-2D.geo`, only one of them has to be fixed (||) and the orientation of the symbol will depend on which one is free and which one is fixed, showing on the screen the corresponding direction for both degrees of freedom.

Finally, for the fourth symbol, obtained from the file Normal.geo, it can be observed that the drawing of the symbol, related to the local orientation will appear scaled according to the real-type values of the second, fourth and sixth field values. Different types of C-language expressions are available in GiD. Thus, the last expression would be equivalent to entering

```
(fabs(cond(real,2))>0. || fabs(cond(real,4))!=0. ||
fabs(cond(real,6))>1e-10)
```

Note: As previously mentioned, GiD internally creates a `project_name.geo` file when saving a project, where it keeps all the information about the geometry in binary format. In fact, this is the reason why the extension of these files is `.geo`. However, the file `project_name.geo` is stored in the `project_name.gid` directory, whereas these user-created `***.geo` files are stored in the `problem_type_name.gid` directory.

Unit System file (.uni)

When GiD is installed, the file `units.gid` is copied within the GiD directory. In this file a table of magnitudes is defined. For each magnitude there is a set of units and a conversion factor between the unit and the reference unit. The units systems are also defined. A unit system is a set of magnitudes and the corresponding unit.

```
BEGIN TABLE
  LENGTH : m, 100 cm, 1e+3 mm
  ...
  STRENGTH : kg*m/s^2, N, 1.0e-1 kp
END
```

```
BEGIN SYSTEM(INTERNATIONAL)
  LENGTH : m
  MASS : kg
  STRENGTH : N
  ...
  TEMPERATURE : Cel
END
```

The syntax of the unit file (`problem_type_name.uni`) within the problem type is similar. It can include the line:

```
USER DEFINED: ENABLED
(or DISABLED)
```

meaning that the user is able (or not able) to define his own system unit within the project. If the line does not appear in the file the value is assumed to be `ENABLED`.

It is possible to ignore all units systems defined by default inside the file `units.gid`:

```
USE BASE SYSTEMS: DISABLED
(or ENABLED)
```

With the command `HIDDEN: 'magnitude', 'magnitude'`, certain magnitudes will not be displayed in the `Problem units` window. For example,

```
HIDDEN: strength, pressure
```

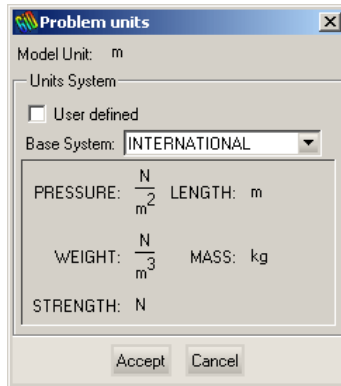
If the problem type uses a property which has a unit, then GiD creates the file `project_name.uni` in the project directory. This file includes the information related to the unit used in the geometric model and the unit system used. The structure of this file is:

```
MODEL: km
PROBLEM: USER_DEFINED
BEGIN SYSTEM
LENGTH: m
PRESSURE: Pa
MASS: kg
STRENGTH: N
END
```

In this file, `MODEL` refers to the unit of the geometric model and `PROBLEM` is the name of the units system used by GiD to convert all the data properties in the output to the solver. If this name is `USER_DEFINED`, then the system is the one defined within the file. The block

```
BEGIN SYSTEM
...
END
```

corresponds to the user-defined system.



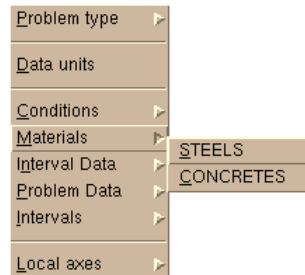
Data unit window

Special fields

These fields are useful for organizing the information within data files. They make the information shown in the data windows more readable. In this way you can better concentrate on the data properties relevant to the current context.

- **Book:** With the **Book** field it is possible to split the data windows into other windows. For example, we can have two windows for the materials, one for the steels and another for the concretes:

```
BOOK: Steels
...
All steels come here
...
BOOK: Concretes
...
All concretes come here
...
```

Options corresponding to books

The same applies to conditions. For general and interval data the **book** field groups a set of properties.

- **Title:** The **Title** field groups a set of properties on different tabs of one book. All properties appearing after this field will be included on this tab.

TITLE: Basic

...

Basics properties

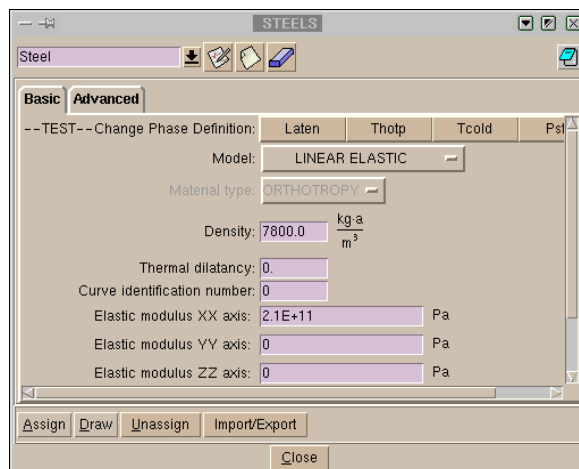
....

TITLE: Advanced

...

Advanced properties

....

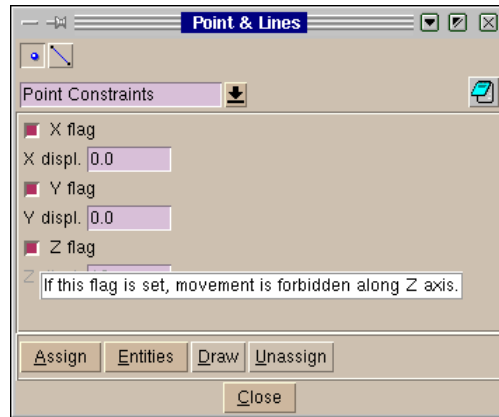


- **Help:** With the **Help** field it is possible to assign a description to the data property preceding it. In this way you can inspect the meaning of the property through the help context function by holding the cursor over the property or by right-clicking on it.

QUESTION: X_flag#CB#(1,0)

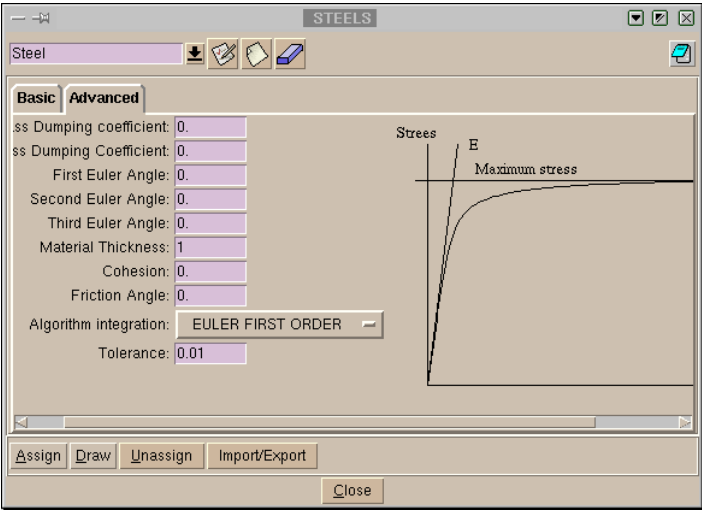
VALUE: 1

HELP: If this flag is set, movement is forbidden along the Z-axis.



- **Image:** The **Image** field is useful for inserting descriptive pictures in the data window. The value of this field is the file name of the picture relating to the problem type location.

IMAGE: omega3.gif



Data window with an image

- **Unit field:** With this feature it is possible to define and work with properties that have units. **Gid** is responsible for the conversion between units of the same magnitude

....

QUESTION: Elastic modulus XX axis:#UNITS#

VALUE: 2.1E+11Pa

...

Elastic modulus XX axis:	2.1E+11	Pa
Elastic modulus YY axis:	0	kg
Elastic modulus ZZ axis:	0	m·s ⁻²
Poisson's ratio XY:	0.3	Pa
Poisson's ratio XZ:	0.	kp
Poisson's ratio YZ:	0.	cm ²

Draw Unassign Import/Export

Data property with units.

- **Dependencies:** Depending on the value, we can define some behavior associated with the property. For each value we can have a list of actions. The syntax is as follows:

```
DEPENDENCIES: (<V1>, [TITLESTATE, <Title>, <State>], <A1>, <P1>, <NV1>,
..., <An>, <Pn>, <NVn>) ... (<Vm>, <Am>, <Pm>, <NVm>, ...)
```

where:

- <Vi>** is the value that triggers the actions. A special value is **#DEFAULT#**, which refers to all the values not listed.
- [TITLESTATE, <Title>, <State>]** this argument is optional. **Titlestate** should be used to show or hide book labels. Many **Titlestate** entries can be given. **<Title>** is the title defined for a book (TITLE: Title). **<State>** is the visualization mode: **normal** or **hidden**.
- <Ai>** is the action and can have one of these values: **SET**, **HIDE**, **RESTORE**. All these actions change the value of the property with the following differences: **SET** disables the property, **HIDE** hides the property and **RESTORE** brings the property to the enabled state.
- <Pi>** is the name of the property to be modified.
- <NVi>** is the new value of **<Pi>**. A special value is **#CURRENT#**, which refers to the current value of **<Pi>**.

Here is an example:

```
...
TITLE: General
QUESTION: Type_of_Analysis:#CB#(FILLING,SOLIDIFICATION)
VALUE: SOLIDIFICATION
DEPENDENCIES: (FILLING,TITLESTATE,Filling-
Strategy,normal,RESTORE,
Filling_Analysis,GRAVITY,HIDE,Solidification_Analysis,#CURRENT#)
DEPENDENCIES: (SOLIDIFICATION,TITLESTATE,Filling-
Strategy,hidden,HIDE,
Filling_Analysis,#CURRENT#,RESTORE,Solidification_Analysis,#CURR
ENT#)
TITLE: Filling-Strategy
QUESTION: Filling_Analysis:#CB#(GRAVITY,LOW-PRESSURE,FLOW-RATE)
VALUE: GRAVITY
QUESTION: Solidification_Analysis:#CB#(THERMAL,THERMO-
MECHANICAL)
VALUE: THERMAL
```

...

- **State:** Defines the state of a field; this state can be: disabled, enabled or hidden. Here is an example:

```
...
QUESTION: Elastic modulus XX axis
VALUE: 2.1E+11
STATE: HIDDEN
...
```

- **#MAT#('BookName'):** Defines the field as a material, to be selected from the list of materials in the book 'BookName'. Here is an example:

```
QUESTION:Composition_Material#MAT#(BaseMat)
VALUE:AISI_4340_STEEL
```

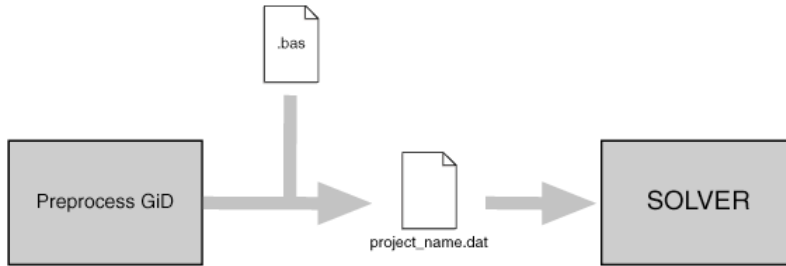
Template File

Once you have generated the mesh, and assigned the conditions and the materials properties, as well as the general problem and intervals data for the solver, it is necessary to produce the data input files to be processed by that program.

To manage this reading, GiD is able to interpret a file called `problem_type_name.bas` (where `problem_type_name` is the name of the working directory of the problem type without the `.bas` extension).

This file (template file) describes the format and structure of the required data input file for the solver that is used for a particular case. This file must remain in the `problem_type_name.gid` directory, as well as the other files already described - `problem_type_name.cnd/.mat/.prb` as well as `problem_type_name.sim` and `***.geo`, if desired.

In the case that more than one data input file is needed, GiD allows the creation of more files by means of additional `***.bas` files (note that while `problem_type_name.bas` creates a data input file named `project_name.dat`, successive `***.bas` files - where `***` can be any name - create files with the names `project_name-1.dat`, `project_name-2.dat`, and so on). The new files follow the same rules as the ones explained next for `problem_type_name.bas` files.



These files work as an interface from GiD's standard results to the specific data input for any individual solver module. This means that the process of running the analysis (see [CALCULATE](#)) simply forms another step that can be completed within the system.

In the event of an error in the preparation of the data input files, the programmer has only to fix the corresponding `problem_type_name.bas` or `***.bas` file and rerun the example, without needing to leave GiD, recompile or reassign any data or re-mesh.

This facility is due to the structure of the template files. They are a group of macros (like an ordinary programming language) that can be read, without the need of a compiler, every time the corresponding analysis file is to be written. This ensures a fast way to debug mistakes.

General description

All the rules that apply to `file_name.bas` files are also valid for other files with the `.bas` extension. Thus, everything in this section will refer explicitly to the file `file_name.bas`. Any information written to this file, apart from the commands given, is reproduced exactly in the output file (the data input file for the numerical solver). The commands are words that begin with the character `*` (if you want to write an asterisk in the file you should write `**`). The commands are inserted among the text to be literally translated. Every one of these commands returns one (see [Single value return commands](#)) or multiple (see [Multiple values return commands](#)) values obtained from the preprocessing component. Other commands mimic the traditional structures to do loops or conditionals (see [Specific commands](#)). It is also possible to create variables to manage some data. Comparing it to a classic programming language, the main differences will be the following:

- The text is reproduced literally, without printing instructions, as it is write-oriented.
- There are no indices in the loops. When the program begins a loop, it already knows the number of iterations to perform. Furthermore, the inner variables of the loop change their values automatically. All the commands can be divided into three types:
 - Commands that return one single value. This value can be an integer, a real number or a string. The value depends on certain values that are available to the command and on the position of the command within the loop or after

setting some other parameters. These commands can be inserted within the text and write their value where it corresponds. They can also appear inside an expression, which would be the example of the conditionals. For this example, you can specify the type of the variable, integer or real, except when using `strcmp` or `strcasecmp`. If these commands are within an expression, no `*` should precede the command.

- Commands that return more than one value. Their use is similar to that of the previously indicated commands, except for the fact that they cannot be used in other expressions. They can return different values, one after the other, depending on some values of the project.
- Commands that perform loops or conditionals, create new variables, or define some specifications. The latter includes conditions or types of element chosen and also serves to prevent line-feeding. These commands must start at the beginning of the line and nothing will be written into the calculations file. After the command, in the same line, there can be other commands or words to complement the definitions, so, at the end of a loop or conditional, after the command you can write what loop or conditional was finished.

The arguments that appear in a command are written immediately after it and inside parenthesis. If there is more than one, they will be separated by commas. The parentheses might be inserted without any argument inside, which is useful for writing something just after the command without inserting any additional spaces. The arguments can be real numbers or integers, meaning the word `REAL` or the word `INT` (both in upper- or lowercase) that the value to which it points has to be considered as real or integer, respectively. Other types of arguments are sometimes allowed, like the type of element, described by its name, in the command `*set elem`, or a chain of characters inserted between double quotes `"..."` for the C-instructions `strcmp` and `strcasecmp`. It is also sometimes possible to write the name of the field instead of its ordering number.

Example:

Below is an example of what a `.bas` file can be. There are two commands (`*nelem` and `*npoin`) which return the total number of elements and nodes of a project.

```
%%% Problem Size %%%
Number of Elements & Nodes:
*nelem *npoin
```

This `.bas` file will be converted into a `project_name.dat` file by GiD. The contents of this file could be something like this:

```
%%% Problem Size %%%
Number of Elements & Nodes:
```

5379 4678

(5379 being the number of elements of the project, and 4678 the number of nodes).

Commands used in the .bas file

Single value return commands

When writing a command, it is generally not case-sensitive (unless explicitly mentioned), and even a mixture of uppercase and lowercase will not affect the results.

- *npoin, *ndime, *nnode, *nelem, *nmats.** These return, respectively, the number of points, the dimensions of the project being considered, the number of nodes of the element with the highest number, the number of elements and the number of materials. All of them are considered as integers and do not carry arguments (see `*format, *intformat`), except `*nelem`, which can bring different types of elements. These elements are: `OnlyPoints`, `Linear`, `Triangle`, `Quadrilateral`, `Tetrahedra`, `Hexahedra`, `Prism` depending on the number of edges the element has, and `All`, which comprises all the possible types. The command `*nmats` returns the number of materials effectively assigned to an entity, not all the defined ones.
- *GenData.** This must carry an argument of integer type that specifies the number of the field to be printed. This number is the order of the field inside the general data list. This must be one of the values that are fixed for the whole problem, independently of the interval (see [Problem and intervals data file \(.prb\)](#)). The name of the field, or an abbreviation of it, can also be the argument instead. The arguments `REAL` or `INT`, to express the type of number for the field, are also available (see `*format, *intformat, *realformat, *if`). If they are not specified, the program will print a character string. It is mandatory to write one of them within an expression, except for `strcmp` and `strcasecmp`. The numeration must start with the number 1.
- *IntvData.** The only difference between this and the previous command is that the field must be one of those fields varying with the interval (see [Problem and intervals data file \(.prb\)](#)). This command must be within a loop over intervals (see `*loop`) and the program will automatically update the suitable value for each iteration.
- *MatProp.** This is the same as the previous command except that it must be within a loop over the materials (see `*loop`). It returns the property whose field number or name is defined by its argument. It is recommended to use names instead of field numbers.

Note: If the argument is 0, it returns the material's name.

Caution: If there are materials with different numbers of fields, you must ensure not to print non-existent fields using conditionals.

- ***ElemsMatProp.** This is the same as `Matprop` but uses the material of the current element. It must be within a loop over the elements (see `*loop`). It returns the property whose field number or name is defined by its argument. It is recommended to use names instead of field numbers.

Example:

```
*loop elements
  *elemsnum *elemsmat *elemsmatprop(young)
*end elements
```

- ***Cond.** The same remarks apply here, although now you have to notify with the command `*set` (see `*set`) which is the condition being processed. It can be within a loop (see `*loop`) over the different intervals should the conditions vary for each interval.
- ***ElemsNum:** This returns the element's number.
- ***NodesNum:** This returns the node's number.
- ***MatNum:** This returns the material's number.
- ***ElemsMat:** This returns the number of the material assigned to the element.

All of these commands must be within a proper loop (see `*loop`) and change automatically for each iteration. They are considered as integers and cannot carry any argument. The number of materials will be reordered numerically, beginning with the number 1 and increasing up to the number of materials assigned to any entity.

- ***LayerNum:** This returns the layer's number.
- ***LayerName:** This returns the layer's name.
- ***LayerColorRGB:** This returns the layer's color in RGB (three integer numbers between 0 and 256). If parameter (1), (2) or (3) is specified, the command returns only the value of one color. RED is 1, GREEN is 2 and BLUE is 3.

The commands ***LayerName**, ***LayerNum** and ***LayerColorRGB** must be inside a loop over layers; you cannot use these commands in a loop over nodes or elements.

Example:

```
*loop layers
*LayerName *LayerColorRGB
*Operation(LayerColorRGB(1)/255.0)
*Operation(LayerColorRGB(2)/255.0)
*Operation(LayerColorRGB(3)/255.0)
*end layers
```

- ***NodesLayerNum:** This returns the layer's number.
- ***NodesLayerName:** This returns the layer's name.
(These must be used in a loop over nodes.)
- ***ElmsLayerNum:** This returns the layer's number.
- ***ElmsLayerName:** This returns the layer's name.
(These must be used in a loop over elements.)
- ***CondNumEntities.** You must have previously selected a condition (see `*set cond`). This returns the number of entities that have a condition assigned over them.
- ***LayerNumEntities.** You must have previously selected a layer (see `*set layer`). This returns the number of entities that are inside this layer.
- ***LoopVar.** This command must be inside a loop and it returns, as an integer, what is considered to be the internal variable of the loop. This variable takes the value 1 in the first iteration and increases by one unit for each new iteration. The parameters `elems`, `nodes`, `materials`, `intervals`, used as an argument for the corresponding loop, allows the program to know which one is being processed. Otherwise, if there are nested loops, the program takes the value of the inner loop.
- ***Operation.** This returns the result of an arithmetical expression what should be written inside parentheses immediately after the command. This operation must be defined in C-format and can contain any of the commands that return one single value. You can force an integer or a real number to be returned by means of the parameters `INT` or `REAL`. Otherwise, GiD returns the type according to the result.

The valid C-functions that can be used are:

- `+, -, *, /, (,), =, <, >, !, &, |`, numbers and variables
- `sin`
- `cos`

- o tan
- o asin
- o acos
- o atan
- o atan2
- o exp
- o fabs
- o abs
- o pow
- o sqrt
- o strcmp
- o strcasecmp

The following are valid examples of operations:

```
*operation(4*elemsnum+1)
*operation(8*(loopvar-1)+1)
```

Note: There cannot be blank spaces between the commands and the parentheses that include the parameters.

Note: Commands inside `*operation` do not need `''` at the beginning.

- ***LocalAxesNum.** This returns the identification name of the local axes system, either when the loop is over the nodes or when it is over the elements, under a referenced condition.
- ***nlocalaxes.** This returns the number of the defined local axes system.
- ***IsQuadratic.** This returns the value 1 when the elements are quadratic or 0 when they are not.
- ***Time.** This returns the number of seconds elapsed since midnight.
- ***Clock.** This returns the number of clock ticks (aprox. milliseconds) of elapsed processor time.

Example:

```
*set var t0=clock
*loop nodes
  *nodescoord
*end nodes
*set var t1=clock
elapsed time=*operation((t1-t0)/1000.0) seconds
```

- ***Units('magnitude')**. This returns the current unit name for the selected magnitude (the current unit is the unit shown inside the unit window).

Example:

```
*Units(LENGTH)
```

- ***BasicUnit('magnitude')**. This returns the basic unit name for the selected magnitude (the basic unit is the unit defined as { Basic } in the *.uni file).

Example:

```
*BasicUnit(LENGTH)
```

- ***FactorUnit('unit')**. This returns the numeric factor to convert a magnitude from the selected unit to the basic unit.

Example:

```
*FactorUnit(PRESSURE)
```

Multiple values return commands

These commands return more than one value in a prescribed order, writing them one after the other. All of them except `LocalAxesDef` are able to return one single value when a numerical argument giving the order of the value is added to the command. In this way, these commands can appear within an expression. Neither `LocalAxesDef` nor the rest of the commands without the numerical argument can be used inside expressions. Below, a list of the commands with the appropriate description is displayed.

- ***NodesCoord**. This command writes the node's coordinates. It must be inside a loop (see `*loop`) over the nodes or elements. The coordinates are considered as real numbers (see `*realformat` and `*format`). It will write two or three coordinates according to the number of dimensions the problem has (see `*Ndime`).

If `*NodesCoord` receives an integer argument (from 1 to 3) inside a loop of nodes, this argument indicates which coordinate must be written: x, y or z. Inside a loop of nodes:

`*NodesCoord` writes 2 or 3 coordinates depending on the number of dimensions.

`*NodesCoord(1)` writes the X-coordinate of the actual node of the loop.

`*NodesCoord(2)` writes the Y-coordinate of the actual node of the loop.

`*NodesCoord(3)` writes the Z-coordinate of the actual node of the loop.

If the argument `real` is given, the coordinates will be treated as real numbers.

Example: using `*NodesCoord` inside a loop of nodes

```
Coordinates:
Node          X                      Y
*loop nodes
*format "%5i%14.5e%14.5e"
*NodesNum *NodesCoord(1,real) *NodesCoord(2,real)
*end nodes
```

This command effects a rundown of all the nodes in the mesh, listing their identifiers and coordinates (**x** and **y**).

The contents of the `project_name.dat` file could be something like this:

```
Coordinates:
Node          X                      Y
    1  -1.28571e+001  -1.92931e+000
    2  -1.15611e+001  -2.13549e+000
    3  -1.26436e+001  -5.44919e-001
    4  -1.06161e+001  -1.08545e+000
    5  -1.12029e+001   9.22373e-002
    ...
```

`*NodesCoord` can also be used inside a loop of elements. In this case, it needs an additional argument that gives the local number of the node inside the element. After this argument it is also possible to give which coordinate has to be written: x, y or z.

Inside a loop of elements:

`*NodesCoord(4)` writes the coordinates of the 4th node of the current loop element.

`*NodesCoord(5,1)` writes the X-coordinate of the 5th node of the current loop element.

`*NodesCoord(5,2)` writes the Y-coordinate of the 5th node of the current loop element.

`*NodesCoord(5,3)` writes the Z-coordinate of the 5th node of the current loop element.

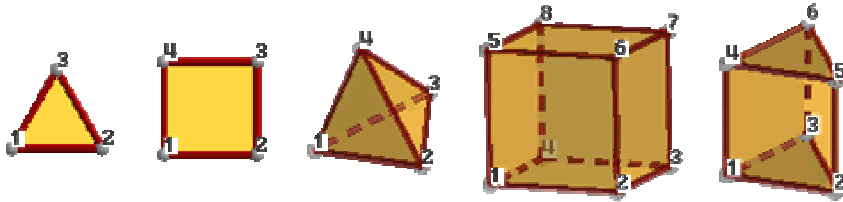
- ***ElemsConec.** This command writes the element's connectivities, i.e. the list of the nodes that belong to the element, displaying the direction for each case (anti-clockwise direction in 2D, and depending on the standards in 3D). For shells, the direction must be defined. However, this command accepts the argument `swap` and this implies that the ordering of the nodes in quadratic elements will be consecutive instead of hierarchical. The connectivities are considered as integers (see `*intformat` and `*format`). If `*ElemsConec` receives an integer argument (beginning with 1), this argument indicates which element connectivity must be written:

```
*loop elems
  all connectivities: *elemsconec
  first connectivity *elemsconec(1)
*end elems
```

Note: In the first versions of GiD, the optional parameter of this last command was `invert` instead of `swap`, as it is now. It was changed for technical reasons. If you have an old `.bas` file prior to this specification, which contains this command in its previous form, when you try to export the calculation file, you will be warned about this change of use. Be aware that the output file will not be created as you expect.

- ***GlobalNodes.** This command returns the nodes that belong to an element's face where a condition has been defined (in the loop over the elements). The direction for this is the same as for that of the element's connectivities. The returned values are considered as integers (see `*intformat` and `*format`). If `*GlobalNodes` receives an integer argument (beginning with 1), this argument indicates which face connectivity must be written.

So, the local numeration of the faces is:



Triangle: 1-2 2-3 3-1

Quadrilateral: 1-2 2-3 3-4 4-1

Tetrahedra: 1-2-3 2-4-3 3-4-1 4-2-1

Hexahedra: 1-2-3-4 1-4-8-5 1-5-6-2 2-6-7-3 3-7-8-4 5-8-7-6

Prism: 1-2-3 1-4-5-2 2-5-6-3 3-6-4-1 4-5-6

- ***LocalNodes.** The only difference between this and the previous one is that the returned value is the local node's numbering for the corresponding element (between 1 and `nnode`).
- ***ElemsNnode.** This command returns the number of nodes of the current element (only valid inside a loop over elements).

Note: This command is only available in GiD version 6.2.0b or later.

Example:

```
*loop elems
  *ElemsNnode
*end elems
```

- ***ElemsNnodeCurt.** This command returns the number of vertex nodes of the current element (only valid inside a loop over elements). For example, for a quadrilateral of 4, 8 or 9 nodes, it returns the value 4.

Note: This command is only available in GiD version 7.2 or later.

- ***ElemsNnodeFace.** This command returns the number of face nodes of the current element face (only valid inside a loop over elements `onlyincond`, with a previous `*set cond of a condition` defined over face elements).

Note: This command is only available in GiD version 7.5.2b or later.

Example:

```
*loop elems
  *ElemsNnodeFace
```

```
*end elems
```

- ***ElmsNNodeFaceCurt.** This command returns the short (corner nodes only) number of face nodes of the current element face (valid only inside a loop over elements onlyincond, with a previous `*set cond` of a condition defined over face elements).
Note: This command is only available in GiD version 7.5.2b or later.

Example:

```
*loop elems
*ElmsNnodeFaceCurt
*end elems
```

- ***ElmsType:** This returns the current element type as a integer value: 1=Linear, 2=Triangle, 3=Quadrilateral, 4=Tetrahedra, 5=Hexahedra, 6=Prism, 7=OnlyPoints (only valid inside a loop over elements).
Note: This command is only available in GiD version 7.2.2b or later.
- ***ElmsTypeName:** This returns the current element type as a string value: `Linear`, `Triangle`, `Quadrilateral`, `Tetrahedra`, `Hexahedra`, `Prism`, `OnlyPoints` (only valid inside a loop over elements).
Note: This command is only available in GiD version 7.2.2b or later.
- ***LocalAxesDef.** This command returns the nine numbers that define the transformation matrix of a vector from the local axes system to the global one.

Example:

```
*loop localaxes
*format "%10.4lg %10.4lg %10.4lg"
x'=*LocalAxesDef(1) *LocalAxesDef(4) *LocalAxesDef(7)
*format "%10.4lg %10.4lg %10.4lg"
y'=*LocalAxesDef(2) *LocalAxesDef(5) *LocalAxesDef(8)
*format "%10.4lg %10.4lg %10.4lg"
z'=*LocalAxesDef(3) *LocalAxesDef(6) *LocalAxesDef(9)
*end localaxes
```

- ***LocalAxesDef(EulerAngles).** This is as the last command, only with the `EulerAngles` option. It returns three numbers that are the 3 Euler angles (radians) that define a local axes system (ϕ, θ, ψ)

$$\begin{pmatrix} \cos\psi \cos\phi - \sin\psi \cos\theta \sin\phi & \cos\psi \sin\phi + \sin\psi \cos\theta \cos\phi & \sin\psi \sin\theta \\ -\sin\psi \cos\phi - \cos\psi \cos\theta \sin\phi & -\sin\psi \sin\phi + \cos\psi \cos\theta \cos\phi & \cos\psi \sin\theta \\ \sin\theta \sin\phi & -\sin\theta \cos\phi & \cos\theta \end{pmatrix} \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}$$

Rotation of a vector expressed in terms of euler angles

- ***LocalAxesDefCenter.** This command returns the origin of coordinates of the local axes as defined by the user. The "Automatic" local axes do not have a center, so the point (0,0,0) is returned. The index of the coordinate (from 1 to 3) can optionally be given to `LocalAxesDefCenter` to get the x, y or z value.

Example:

```
*LocalAxesDefCenter
*LocalAxesDefCenter(1) *LocalAxesDefCenter(2)
*LocalAxesDefCenter(3)
```

Specific commands

- ***** To avoid line-feeding you need to write `*\` so that the line currently being used continues on the following line of the file filename.bas.
- ***#** If this is placed at the beginning of the line, it is considered as a comment and therefore is not written.
- ****** In order for an asterisk symbol to appear in the text, two asterisks `**` must be written.
- ***Include.** The include command allows you to include the contents of a slave file inside a master .bas file, setting a relative path from the Problem Type directory to this secondary file.

Example:

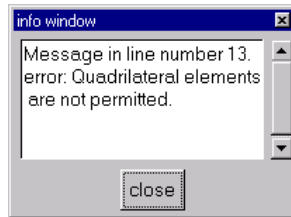
```
*include includes\execntrlmi.h
```

Note: The `*.bas` extension cannot be used for the slave file to avoid multiple output files.

- ***MessageBox.** This command stops the execution of the .bas file and prints a message in a window; this command should only be used when a fatal error occurs.

Example:

```
*MessageBox error: Quadrilateral elements are not permitted.
```



This window appears if the command `MessageBox` is executed

- ***WarningBox.** This is the same as `MessageBox`, but the execution is not stopped.

Example:

```
WarningBox Warning: Exist Bad elements. A STL file is a
collection of triangles bounding a volume.
```

The following commands must be written at the beginning of a line and the rest of the line will serve as their modifiers. No additional text should be written.

- ***loop, *end, *break.** These are declared for the use of loops. A loop begins with a line that starts with `*loop` (none of these commands is case-sensitive) and contains another word to express the variable of the loop. There are some lines in the middle that will be repeated depending on the values of the variable, and whose parameters will keep on changing throughout the iterations if necessary. Finally, a loop will end with a line that finishes with `*end`. After `*end`, you may write any kind of comments in the same line. The command `*break` inside a `*loop` or `*for` block, will finish the execution of the loop and will continue after the `*end` line.

The variables that are available for `*loop` are the following:

- **elems, nodes, materials, layers, intervals, localaxes.** These commands mean, respectively, that the loop will iterate over the elements, nodes, materials, layers, intervals or local axes systems. The loops can be nested among them. The loop over the materials will iterate only over the effectively assigned materials to an entity, in spite of the fact that more materials have been defined. The number of the materials will begin with the number 1. If a command that depends on the loop is located outside it, the number will also take by default the value 1.

With the command ***NotUsed**, it is possible to make a loop over those materials that are defined but not used.

Example:

```
*loop materials *NotUsed
```

After the command `*loop`, if the variable is `elems` or `nodes`, you can include one of the modifiers: `*all`, `*OnlyInCond` or `*OnlyInLayer`. The first one signifies that the iteration is going to be performed over all the entities; the `*OnlyInCond` modifier implies that the iteration will only take place over the entities that satisfy the relevant condition. This condition must have been previously defined with `*set cond`.

`*OnlyInLayer` implies that the iteration will only take place over the entities that are in the specified layer; layers must be specified with the command `*set Layer`. By default, it is assumed that the iteration will affect all the entities.

Example 1:

```
*set elems(all)
*loop nodes
*format "%5i%14.5e%14.5e"
*NodesNum *NodesCoord(1,real) *NodesCoord(2,real)
*end nodes
```

This command carries out a rundown of all the nodes of the mesh, listing their identifiers and coordinates (X- and Y-coordinates).

Example 2:

```
*Set Cond Point-Weight *nodes
*loop nodes *OnlyInCond
*NodesNum      *cond(1)
*end
```

This carries out a rundown of all the nodes assigned the condition "Point-Weight" and provides a list of their identifiers and the first "weight" field of the condition in each case.

Example 3:

```
*Loop Elems
*ElemsNum
*ElemsLayerNum
*End Elems
```

This carries out a rundown of all the elements and provides a list of their identifier and the identifier of the layer to which they belong.

Example 4:

```
*Loop Layers
*LayerNum
*LayerName
*End Layers
```

This carries out a rundown of all the layers and for each layer it lists its identifier and name.

- ***if, *else, *elseif, *endif.** These commands create the conditionals. The format is a line which begins with ***if** followed by an expression between parenthesis. This expression will be written in C-language syntax, value return commands will not begin with *****, and its variables must be defined as integers or real numbers (see ***format**, ***intformat**, ***realformat**), with the exception of **strcmp** and **strcasecmp**. It can include relational as well as arithmetic operators inside the expressions.

The following are valid examples of the use of the conditionals:

```
*if((fabs(loopvar)/4)<1.e+2)
*if((p3<p2)||p4)
*if((strcasecmp(cond(1),"XLoad")==0)&&(cond(2)!=0))
```

The first example is a numerical example where the condition is satisfied for the values of the loop under 400, while the other two are logical operators; in the first of these two, the condition is satisfied when $p3 < p2$ or $p4$ is not equal to 0, and in the second, when the first field of the condition is called **XLoad** (with this particular writing) and the second is not null.

If the checked condition is true, GiD will write all the lines until it finds the corresponding ***else**, ***elseif** or ***endif** (***end** is equivalent to ***endif** after ***if**). ***else** or ***elseif** are optional and require the writing of all the lines until the corresponding ***endif**, but only when the condition given by ***if** is false. If either ***else** or ***elseif** is present, it must be written between ***if** and ***endif**. The conditionals can be nested among them.

The behavior of ***elseif** is identical to the behavior of ***else** with the addition of a new condition:

```
*if(GenData(31,int)==1)
... (1)
```

```

*elseif (GenData (31,int)==2)
... (2)
*else
... (3)
*endif

```

In the previous example, the body of the first condition (written as 1) will be written to the data file if `GenData (31,int)` is 1; the body of the second condition (written as 2) will be written to the data file if `GenData (31,int)` is 2; and if neither of these is true, the body of the third condition (written as 3) will be written to the data file.

Note: A conditional can also be written in the middle of a line. To do this, begin another line and write the conditional by means of the command `*\`.

- ***for, *end, *break.** The syntax of this command is equivalent to `*for` in C-language.

```

*for (varname=expr.1;varname<=expr.2;varname=varname+1)
*end for

```

The meaning of this statement is the execution of a controlled loop, since `varname` is equal to `expr.1` until it is equal to `expr.2`, with the value increasing by 1 for each step. `varname` is any name and `expr.1` and `expr.2` are arithmetical expressions or numbers whose only restrictions are to express the range of the loop.

The command `*break` inside a `*loop` or `*for` block, will finish the execution of the loop and will continue after the `*end` line.

Example:

```

*for (i=1;i<=5;i=i+1)
variable i=*i
*end for

```

- ***set.** This command has the following purposes:
 - `*set cond:` To set a condition.
 - `*set Layer "layer name" *elems|nodes:` To set a layer.
 - `*set elems:` To indicate the elements.
 - `*set var:` To indicate the variables to use.

It is not necessary to write these commands in lowercase, so `*Set` will also be valid in all the examples.

***set cond.:** In the case of the conditions, GiD allows the combination of a group of them with the use of `*add cond`. When a specific condition is about to be used, it must first be defined, and then this definition will be used until another is defined. If this feature is performed inside a loop over intervals, the corresponding entities will be chosen. Otherwise, the entities will be those referred to in the first interval.

It is done in this way because when you indicate to the program that a condition is going to be used, GiD creates a table that lets you know the number of entities over which this condition has been applied. It is necessary to specify whether the condition takes place over the `*nodes`, over the `*elems` or over `*layers` to create the table.

So, a first example to check the nodes where displacement constraints exist could be:

```
*Set Cond Volu-Cstrt *nodes
*Add Cond Surf-Cstrt *nodes
*Add Cond Line-Cstrt *nodes
*Add Cond Poin-Cstrt *nodes
```

These let you apply the conditions directly over any geometric entity.

```
*Set Layer "layer name" *elems|nodes
*Add Layer "layer name"
*Remove Layer "layer name"
```

This command sets a group of nodes. In the following loops over nodes or elements with the modifier `*OnlyInLayer`, the iterations will only take place over the nodes or elements of that group.

Example 1:

```
*set Layer example_layer_1 *elems
*loop elems *OnlyInLayer
    N°:*ElemsNum Name of Layer:*ElemsLayerName N° of Layer
:*ElemsLayerNum
*end elems
```

Example 2:

```
*loop layers
*set Layer *LayerName *elems
```

```

*loop elems *OnlyInLayer
    N°:*ElemsNum Name of Layer:*ElemsLayerName N° of Layer
:*ElemsLayerNum
*end elems
*end layers

```

In this example the command `*LayerName` is used to get the layer name.

There are some modifiers available to point out particular specifications of the conditions.

If the command `*CanRepeat` is added after `*nodes` or `*elems` in `*Set cond`, one entity can appear several times in the entities list. If the command `*NoCanRepeat` is used, entities will appear only once in the list. By default, `*CanRepeat` is off except where one condition has the `*CanRepeat` flag already set.

A typical case where you would not use `*CanRepeat` might be:

```
*Set Cond Line-Constraints *nodes
```

In this case, when two lines share one endpoint, instead of two nodes in the list, only one is written.

A typical situation where you would use `*CanRepeat` might be:

```
*Set Cond Line-Pressure *elems *CanRepeat
```

In this case, if one triangle of a quadrilateral has more than one face in the marked boundary then we want this element to appear several times in the elements list, once for each face.

Other modifiers are used to inform the program that there are nodes or elements that can satisfy a condition more than once (for instance, a node that belongs to a certain number of lines with different prescribed movements) and that have to appear unrepeatd in the data input file, or, in the opposite case, that have to appear only if they satisfy more than one condition. These requirements are achieved with the commands `*or(i,type)` and `*and(i,type)`, respectively, after the input of the condition, where `i` is the number of the condition to be considered and `type` is the type of the variable (integer or real).

For the previous example there can be nodes or elements in the intersection of two lines or maybe belonging to different entities where the same condition had been applied. To avoid the repetition of these nodes or elements, GiD has the modifier `*or`, and in the case where two or more different values were applied over a node or element, GiD only would consider one, this value being different from zero. The reason for this can be easily understood by looking at the following example. Considering the previous commands transformed as:

```
*Set Cond Volu-Cstrt *nodes *or(1,int) *or(2,int)
*Add Cond Surf-Cstrt *nodes *or(1,int) *or(2,int)
*Add Cond Line-Cstrt *nodes *or(1,int) *or(2,int)
*Add Cond Poin-Cstrt *nodes *or(1,int) *or(2,int)
```

where `*or(1,int)` means the assignment of that node to the considered ones satisfying the condition if the integer value of the first condition's field is not equal to zero, and `*or(2,int)` means the same assignment if the integer value of the second condition's field is not equal to zero. Let us imagine that a zero in the first field implies a restricted movement in the direction of the X-axis and a zero in the second field implies a restricted movement in the direction of the Y-axis. If a point belongs to an entity whose movement in the direction of the X-axis is constrained, but whose movement in the direction of the Y-axis is released and at the same time to an entity whose movement in the direction of the Y-axis is constrained, but whose movement in the direction of the X-axis is released, GiD will join both conditions at that point, appearing as a fixed point in both directions and as a node satisfying the four expressed conditions that would be counted only once.

The same considerations explained for adding conditions through the use of `*add cond` apply to elements, the only difference being that the command is `*add elems`. Moreover, it can sometimes be useful to remove sets of elements from the ones assigned to the specific conditions. This can be done with the command `*remove elems`. So, for instance, GiD allows combinations of the type:

```
*Set Cond Dummy *elems
*Set elems(All)
*Remove elems(Linear)
```

To indicate that all dummy elements apart from the linear ones will be considered, as well as:

```
*Set Cond Dummy *elems
*Set elems(Hexahedra)
```



```
*Add elems(Tetrahedra)
*Add elems(Quadrilateral)
*Add elems(Triangle)
```

The format for `*set var` differs from the syntax for the other two `*set` commands. Its syntax is as follows:

```
*Set var varname = expression
```

where `varname` is any name and `expression` is any arithmetical expression, number or command, where the latter must be written without `*` and must be defined as `Int` or `Real`.

A Tcl procedure can also be called, but it must return a numerical result. The following are valid examples for these assignments:

```
*Set var k01=cond(1,real)
*Set var k02=2
*Set var S1=CondNumEntities
*Set var p1=elemsnum()
*Set var b=operation(p1*2)
*tcl(proc MultiplyByTwo { x } { return [expr {$x*2}] })*\
  *Set var a=tcl(MultiplyByTwo *p1)
```

- ***intformat, *realformat, *format.** These commands explain how the output of different mathematical expressions will be written to the analysis file. The use of this command consists of a line which begins with the corresponding version, `*intformat`, `*realformat` or `*format` (again, these are not case-sensitive), and continues with the desired writing format, expressed in C-language syntax argument, between double quotes ("...").

The integer definition of `*intformat` and the real number definition of `*realformat` remain unchanged until another definition is provided via `*intformat` and `*realformat`, respectively. The argument of these two commands is composed of a unique field. This is the reason why the `*intformat` and `*realformat` commands are usually invoked in the initial stages of the `.bas` file, to set the format configuration of the integer or real numbers to be output during the rest of the process.

The `*format` command can include several field definitions in its argument, mixing integer and real definitions, but it will only affect the line that follows the command's

instance one. Hence, the `*format` command is typically used when outputting a listing, to set a temporary configuration.

In the paragraphs that follow, there is an explanation of the C format specification, which refers to the field specifications to be included in the arguments of these commands. Keep in mind that the type of argument that the `*format` command expects may be composed of several fields, and the `*intformat` and `*realformat` commands' arguments are composed of an unique field, declared as integer and real, respectively, all inside double quotes:

A format specification, which consists of optional and required fields, has the following form: `%[flags][width][.precision]type`. The start of a field is signaled by the percentage symbol (%). Each field specification is composed of: some flags, the minimum width, a separator point, the level of precision of the field, and a letter which specifies the type of the data to be represented. The field type is the only one required.

The most common flags are:

- To left align the result
- + To prefix the numerical output with a sign (+ or -)
- # To force the real output value to contain a decimal point.

The most usual representations are integers and floats. For integers the letters `d` and `i` are available, which force the data to be read as signed decimal integers, and `u` for unsigned decimal integers.

For floating point representation, there are the letters `e`, `f` and `g`, these being followed by a decimal point to separate the minimum width of the number from the figure giving the level of precision. The number of digits after the decimal point depends on the requested level of precision.

Note: The standard width specification never causes a value to be truncated. A special command exists in GiD: `*SetFormatForceWidth`, which enables this truncation to a prescribed number of digits.

For string representations, the letter `s` must be used. Characters are printed until the precision value is reached.

The following are valid examples of the use of **format**:

```
*Intformat "%5i"
```

With this sentence, usually located at the start of the file, the output of an integer quantity is forced to be right aligned on the fifth column of the text format on the right side. If the number of digits exceeds five, the representation of the number is not truncated.

```
*Realformat "%10.3e"
```

This sentence, which is also frequently located in the first lines of the template file, sets the output format for the real numbers as exponential with a minimum of ten digits, and three digits after the decimal point.

```
*format "%10i%10.3e%10i%15.6e"
```

This complex command will specify a multiple assignment of formats to some output columns. These columns are generated with the line command that will follow the format line. The subsequent lines will not use this format, and will follow the general settings of the template file or the general formats: `*IntFormat`, `*RealFormat`.

- ***SetFormatForceWidth**, ***SetFormatStandard** The default width specification of a "C/C+" format, never causes a value to be truncated.

`*SetFormatForceWidth` is a special command that allows a figure to be truncated if the number of characters to print exceeds the specified width.

`*SetFormatStandard` changes to the default state, with truncation disabled.

For example:

```
*SetFormatForceWidth
*set var num=-31415.16789
*format "%8.3f"
*num
*SetFormatStandard
*format "%8.3f"
*num
```

Output:

```
-31415.1
-31415.168
```

The first number is truncated to 8 digits, but the second number, printed with "C" standard, has 3 numbers after the decimal point, but more than 8 digits.

- ***Tcl** This command allows information to be printed using the Tcl extension language. The argument of this command must be a valid Tcl command or expression which must return the string that will be printed. Typically, the Tcl command is defined in the Tcl file (.tcl , see [TCL/TK EXTENSION](#) for details).

Example: In this example the ***Tcl** command is used to call a **Tcl** function defined in the `problem_type.tcl` file. That function can receive a variable value as its argument with ***variable**. It is also possible to assign the returned value to a variable, but the Tcl procedure must return a numerical value.

In the .bas file:

```
*set var num=1
*tcl(WriteSurfaceInfo *num)
*set var num2=tcl(MultiplyByTwo *num)
```

In the .tcl file:

```
proc WriteSurfaceInfo { num } {
    return [GiD_Info list_entities surfaces $num]
}

proc MultiplyByTwo { x } {
    return [expr {$x*2}]
}
```

Detailed example - Template file creation

Below is an example of how to create a Template file, step by step.

Note that this is a real file and as such has been written to be compatible with a particular solver program. This means that some or all of the commands used will be non-standard or incompatible with the solver that another user may be using.

The solver for which this example is written treats a line inside the calculation input file as a comment if it is prefixed by a `$` sign. In the case of other solvers, another convention may apply.

Of course, the real aim of this example is familiarize you with the commands GiD uses. What follows is the universal method of accessing GiD's internal database, and then outputting the desired data to the solver.

It is assumed that files with the `.bas` extension will be created inside the working directory where the problem type file is located. The filename must be `problem_type_name.bas` for the first file and any other name for the additional `.bas` files. Each `.bas` file will be read by GiD and translated to a `.dat` file.

It is very important to remark that any word in the `.bas` file having no meaning as a GiD compilation command or not belonging to any command instructions (parameters), will be written verbatim to the output file.

First, we create the header that the solver needs in this particular case.

It consists of the name of the solver application and a brief description of its behavior.

```
$-----
CALSEF: PROGRAM FOR STRUCTURAL ANALYSIS
```

What follows is a commented line with the `ECHO ON` command. This, when uncommented, is useful if you want to monitor the progress of the calculation. While this particular command may not be compatible with your solver, a similar one may exist.

```
$-----
$ ECHO ON
```

The next line specifies the type of calculation and the materials involved in the calculation; this is not a GiD related command either.

```
$-----
ESTATICO-LINEAL, EN SOLIDOS
```

As you can see, a commented line with dashes is used to separate the different parts of the file, thus improving the readability of the text.

The next stage involves the initialization of some variables. The solver needs this to start the calculation process.

The following assignments take the first (parameter (1)) and second (parameter (2)) fields in the general problem, as the number of problems and the title of the problem.

The actual position of a field is determined by checking its order in the problem file, so this process requires you to be precise.

Assignment of the first (1) field of the Problem data file, with the command `*GenData(1):`

```
$-----
$  NUMBER OF PROBLEMS: NPROB = *GenData(1)
$-----
```

Assignment of the second (2) field assignment, `*GenData(2):`

```
$  TITLE OF THE PROBLEM: TITULO= *GenData(2)
$-----
```

The next instruction states the field where the starting time is saved. In this case, it is at the 10th position of the general problem data file, but we will use another feature of the `*GenData` command, the parameter of the command will be the name of the field.

This method is preferable because if the list is shifted due to a field being added or subtracted, you will not lose the actual position. This command accepts an abbreviation, as long as there is no conflict with any other field name.

```
$-----
$  TIME OF START: TIME= *GenData(Starting_time)
$-----
```

Here comes the initialization of some general variables relevant to the project in question - the number of points, the number of elements or the number of materials.

The first line is a description of the section.

```
$  DIMENSIONS OF THE PROBLEM:
```

The next line introduces the assignments.

```
DIMENSIONS :
```

This is followed by another line which features the three variables to be assigned. `NPNOD` gets, from the `*npoin` function, the number of nodes for the model; `NELEM` gets, from `*nelem`, either the total number of elements in the model or the number of elements for every kind of element; and `NMATS` is initialized with the number of materials:

```
NPNOD= *npoin,    NELEM= *nelem,    NMATS= *nmats,    \
```

In the next line, `NNODE` gets the maximum number of nodes per element and `NDIME` gets the variable `*ndime`. This variable must be a number that specifies whether all the nodes are on the plane whose Z values are equal to 0 (`NDIME=2`), or if they are not (`NDIME=3`):

```
NNODE= *nnode,    NDIME= *ndime,                                \
```

The following lines take data from the general data fields in the problem file. `NCARG` gets the number of charge cases, `NGDLN` the number of degrees of freedom, `NPROP` the properties number, and `NGAUSS` the gauss number; `NTIPO` is assigned dynamically:

```
NLOAD= *GenData(Load_Cases), *\
```

You could use `NGDLN= *GenData(Degrees_Freedom), *\` but because the length of the argument will exceed one line, we have abbreviated its parameter (there is no conflict with other question names in this problem file) to simplify the command.

```
NGDLN= *GenData(Degrees_Fre), *\
NPROP= *GenData(Properties_Nbr), \
NGAUS= *GenData(Gauss_Nbr) , NTIPO= *\
```

Note that the last assignment is ended with the specific command `*\` to avoid line feeding. This lets you include a conditional assignment of this variable, depending on the data in the General data problem.

Within the conditional a C format-like `strcmp` instruction is used. This instruction compares the two strings passed as a parameter, and returns an integer number which expresses the relationship between the two strings. If the result of this operation is equal to 0, the two strings are identical; if it is a positive integer, the first argument is greater than the second, and if it is a negative integer, the first argument is smaller than the second.

The script checks what problem type is declared in the general data file, and then it assigns the coded number for this type to the `NTIPO` variable:

```

*if(strcmp(GenData(Problem_Type),"Plane-stress")==0)
1 *\
*elseif(strcmp(GenData(Problem_Type),"Plane-strain")==0)
2 *\
*elseif(strcmp(GenData(Problem_Type),"Revol-Solid")==0)
3 *\
*elseif(strcmp(GenData(Problem_Type),"Solid")==0)
4 *\
*elseif(strcmp(GenData(Problem_Type),"Plates")==0)
5 *\
*elseif(strcmp(GenData(Problem_Type),"Revol-Shell")==0)
6 *\
*endif

```

You have to cover all the cases within the if sentences or end the commands with an `elseif` you do not want unpredictable results, like the next line raised to the place where the parameter will have to be:

```

$ Default Value:
*else
0*\
*endif

```

In our case this last rule has not been followed, though this can sometimes be useful, for example when the problem file has been modified or created by another user and the new specification may differ from the one we expect.

The next assignment is formed by a string compare conditional, to inform the solver about a configuration setting.

First is the output of the variable to be assigned.

```

, IWRT= *\

```

Then there is a conditional where the string contained in the value of the `Result_File` field is compared with the string `'Yes'`. If the result is 0, then the two strings are the same, while the output result 1 is used to declare a boolean `TRUE`.

```

*if(strcmp(GenData(Result_File),"Yes")==0)

```



```
1 ,*\
```

Then we compare the same value string with the string 'No', to check the complementary option. If we find that the strings match, then we output a 0.

```
*elseif(strcmp(GenData(Result_File),"No")==0)
0 ,*\
*endif
```

The second to last assignment is a simple output of the solver field contents to the INDSO variable:

```
INDSO= *GenData(Solver) , *\<
```

The last assignment is a little more complicated. It requires the creation of some internal values, with the aid of the `*set cond` command.

The first step is to set the conditions so we can access its parameters. This setting may serve for several loops or instructions, as long as the parameters needed for the other blocks of instructions are the same.

This line sets the condition Point-Constraints as an active condition. The `*nodes` modifier means that the condition will be listed over nodes. The `*or(...)` modifiers are necessary when an entity shares some conditions because it belongs to two or more elements.

As an example, take a node which is part of two lines, and each of these lines has a different condition assigned to it. This node, a common point of the two lines, will have these two conditions in its list of properties. So declaring the `*or` modifiers, GiD will decide which condition to use, from the list of conditions of the entity.

A first instruction will be as follows, where the parameters of the `*or` commands are an integer - (1, and (3, in this example - and the specification `int`, which forces GiD to read the condition whose number position is the integer.

In our case, we find that the first (1) field of the condition file is the X-constraint, and the third (3) is the Y-constraint:

GiD still has no support for substituting the condition's position in the file by its corresponding label, in contrast to case for the fields in the problem data file, for which it is possible.

```
*Set Cond Surface-Constraints *nodes *or(1,int) *or(3,int)
```

Now we want to complete the setting of the loop, with the addition of new conditions.

```
*Add Cond Line-Constraints *nodes *or(1,int) *or(3,int)
*Add Cond Point-Constraints *nodes *or(1,int) *or(3,int)
```

Observe the order in which the conditions have been included: firstly, the surface constraints with the `*Set Cond` command, since it is the initial sentence; then the pair of `*Add Cond` sentences, the line constraints; and finally, the point constraints sentence. This logical hierarchy forces the points to be the most important items.

Last of all, we set a variable with the number of entities assigned to this particular condition.

Note that the execution of this instruction is only possible if a condition has been set previously.

```
NPRES= *CondNumEntities
```

To end this section, we put a separator in the output file:

```
$-----
```

Thus, after the initialization of these variables, this part of the file ends up as:

```
$ DIMENSIONS OF THE PROBLEM:
DIMENSIONES :
  NPNOD= *npoin,  NELEM= *nelem,  NMATS= *nmats,      \
  NNODE= *nnode,  NDIME= *ndime,                    \
  NCARG= *GenData(Charge_Cases), * \
  NGDLN= *GenData(Degrees_Fre), * \
  NPROP= *GenData(Properties_Nbr), \
  NGAUS= *GenData(Gauss_Nbr) , NTIPO= * \
  if(strcmp(GenData(Problem_Type),"Tens-Plana")==0)
1 * \
  elseif(strcmp(GenData(Problem_Type),"Def-Plana")==0)
2 * \
  elseif(strcmp(GenData(Problem_Type),"Sol-Revol")==0)
3 * \
  elseif(strcmp(GenData(Problem_Type),"Sol-Tridim")==0)
4 * \
```

```

*elseif(strcmp(GenData(Problem_Type),"Placas")==0)
5 *\
*elseif(strcmp(GenData(Problem_Type),"Laminas-Rev")==0)
6 *\
*endif
, IWRIT= *\
*if(strcmp(GenData(Result_File),"Yes")==0)
1 ,\
*elseif(strcmp(GenData(Result_File),"No")==0)
0 ,\
*endif
    INDSO= *GenData(Solver) , *\
*Set Cond Surface-Constraints *nodes *or(1,int) *or(3,int)
*Add Cond Line-Constraints *nodes *or(1,int) *or(3,int)
*Add Cond Point-Constraints *nodes *or(1,int) *or(3,int)
NPRES=*CondNumEntities
$-----

```

After creating or reading our model, and once the mesh has been generated and the conditions applied, we can export the file (`project_name.dat`) and send it to the solver.

The command to create the `.dat` file can be found on the `File→Export→Calculation File GiD` menu. It is also possible to use the keyboard shortcut `Ctrl-x Ctrl-c`.

These would be the contents of the `project_name.dat` file:

```

$-----
CASESF: PROGRAM FOR STRUCTURAL ANALYSIS
$-----
$ECHO ON
$-----
LINEAR-STATIC, SOLIDS
$-----
$NUMBER OF PROBLEMS:
NPROB = 1
$-----
$ PROBLEM TITLE
TITLE= Title_name
$-----

```

```
$DIMENSIONS OF THE PROBLEM
```

```
DIMENSIONS :
```

```
  NPNOD=   116 ,  NELEM=  176 ,  NMATS=   0 ,      \
  NNODE=    3 ,  NDIME=   2 ,                      \
  NCARG=    1 ,  NGDLN=   1 ,  NPROP=   5 ,      \
  NGAUS=    1 ,  NTIPO=   1 ,  IWRT=    1 ,      \
  INDSO=   10 ,  NPRES=   0
```

```
$-----
```

This is where the calculation input begins.

A) Elements, materials and connectivities listing

Now we want to output the desired results to the output file. The first line should be a title or a label as this lets the solver know where a loop section begins and ends. The end of this block of instructions will be signalled by the line `END_GEOMETRY`.

```
GEOMETRY
```

The next two of lines give the user information about what types of commands follow.

Firstly, a title for the first subsection, `ELEMENTAL CONNECTIVITIES`:

```
$ ELEMENTAL CONNECTIVITIES
```

followed by a header that precedes the output list:

```
$ ELEM. MATER.      CONNECTIVITIES
```

The next part of the code concerns the elements of the model with the inclusion of the `*loop` instruction, followed in this case by the `elems` argument.

```
*loop elems
```

For each element in the model, GiD will output: its element number, by the action of the `*elemsnum` command, the material assigned to this element, using the `*elemsmat` command, and the connectivities associated to the element, with the `*elemsConec` command:

```
*elemsnum *elemsmat *elemsConec
*end elems
```

You can use the `swap` parameter if you are working with quadratic elements and if the listing mode of the nodes is non-hierarchical (by default, corner nodes are listed first and mid nodes afterwards):

```
*elemsnum *elemsmat *elemsConec (swap)
*end elems
```

B) Formatted nodes and coordinates listing

As with the previous section, this block of code begins with a title for the subsection:

```
$ NODAL COORDINATES
```

followed by the header of the output list:

```
$  NODE  COORD.-X  COORD.-Y  COORD.-Z
```

Now GiD will trace all the nodes of the model:

```
*loop nodes
```

For each node in the model, GiD will generate and output its number, using `*NodesNum`, and its coordinates, using `*NodesCoord`.

The command executed before the output `*format` will force the resulting output to follow the guidelines of the specified formatting.

In this example below, the `*format` command gets a string parameter with a set of codes: `%6i` specifies that the first word in the list is coded as an integer and is printed six points from the left; the other three codes, all `%15.5f`, order the printing of a real number, represented in a floating point format, with a distance of 15 spaces between columns (the number will be shifted to have the last digit in the 15th position of the column) and the fractional part of the number will be represented with five digits.

Note that this is a C language format command.

```
*format "%6i%15.5f%15.5f%15.5f"
*NodesNum *NodesCoord
*end nodes
```

At the end of the section the end marker is added, which in this solver example is as follows:

```
END_GEOMETRY
```

The full set of commands to make this part of the output is shown in the following lines.

```
GEOMETRY
$ ELEMENT CONNECTIVITIES
$ ELEM. MATER.    CONNECTIVITIES
*loop elems
  *elemsnum *elemsmat *elemsConec
*end elems
$ NODAL COORDINATES
$  NODE  COORD.-X  COORD.-Y  COORD.-Z
*loop nodes
*format "%6i%15.5f%15.5f%15.5f"
  *NodesNum *NodesCoord
*end
END_GEOMETRY
```

The result of the compilation is output to a file (`project_name.dat`) to be processed by the solver program.

The first part of the section:

```
$-----
GEOMETRY
$ ELEMENT CONNECTIVITIES
$ ELEM. MATER.    CONNECTIVITIES
    1      1      73      89      83
    2      1      39      57      52
    3      1      17      27      26
    4      5       1       3       5
    5      5       3      10       8
    6      2      57      73      67
    .      .       .       .       .
    .      .       .       .       .
    .      .       .       .       .
  176      5      41      38      24
```

And the second part of the section:

```
$ NODAL COORDINATES
$  NODE  COORD.-X  COORD.-Y  COORD.-Z
      1      5.55102      5.51020
      2      5.55102      5.51020
      3      4.60204      5.82993
      4      4.60204      5.82993
      5      4.88435      4.73016
      6      4.88435      4.73016
      .      .          .
      .      .          .
      .      .          .
    116     -5.11565      3.79592
END_GEOMETRY
```

If the solver module you are using needs a list of the nodes that have been assigned a condition, for example, a neighborhood condition, you have to provide it as is explained in the next example.

C) Nodes listing declaration

First, we set the necessary conditions, as was done in the previous section.

```
*Set Cond Surface-Constraints *nodes *or(1,int) *or(3,int)
*Add Cond Line-Constraints *nodes *or(1,int) *or(3,int)
*Add Cond Point-Constraints *nodes *or(1,int) *or(3,int)
NPRES=*CondNumEntities
```

After the data initialization and declarations, the solver requires a list of nodes with boundary conditions and the fields that have been assigned.

In this example, all the selected nodes will be output and the 3 conditions will also be printed. The columns will be output with no apparent format.

Once again, the code begins with a subsection header for the solver program and a commentary line for the user:

```
BOUNDARY CONDITIONS
$ RESTRICTED NODES
```

Then comes the first line of the output list, the header:

```
$ NODE CODE          PRESCRIPTED VALUES
```

The next part the loop instruction, in this case over nodes, and with the specification argument `*OnlyInCond`, to iterate only over the entities that have the condition assigned. This is the condition that has been set on the previous lines.

```
*loop nodes *OnlyInCond
```

The next line is the format command, followed by the lines with the commands to fill the fields of the list.

```
*format "%5i%1i%1i%f%f"
*NodesNum *cond(1,int) *cond(3,int)  *\  

```

The `*format` command influence also includes the following two if sentences. If the degrees of freedom field contains an integer equal or greater than 3, the number of properties will be output.

```
*if (GenData (Degrees_Freedom_Nodes,int) >=3)
*cond(5,int) *\  
*endif
```

And if the value of the same field is equal to 5 the output will be a pair of zeros.

```
*if (GenData (Degrees_Free,int) ==5)
0 0  *\  
*endif
```

The next line outputs the values contained in the second and fourth fields, both real numbers.

```
*cond(2,real) *cond(4,real) *\  

```

In a similar manner to the previous if sentences, here are some lines of code which will output the sixth condition field value if the number of degrees of freedom is equal or greater than three, and will output a pair of zeros if it is equal to five.

```
*if (GenData (Degrees_Free,int) >=3)
```



```

*cond(6,real) *\
*endif
*if(GenData(Degrees_Free,int)==5)
0.0 0.0 *\
*endif

```

Finally, to end the section, the `*end` command closes the previous `*loop`. The last line is the label of the end of the section.

```

*end
END_BOUNDARY CONDITIONS
$-----

```

The full set of commands included in this section are as follows:

```

BOUNDARY CONDITIONS
$ RESTRICTED NODES
$ NODE CODE          PRESCRIPTED VALUES
*loop nodes *OnlyInCond
*format "%5i%1i%1i%f%f"
      *NodesNum *cond(1,int) *cond(3,int) *\
*if(GenData(Degrees_Free,int)>=3)
*cond(5,int) *\
*endif
*if(GenData(Degrees_Free,int)==5)
0 0 *\
*endif
*cond(2,real) *cond(4,real) *\
*if(GenData(Degrees_Free,int)>=3)
*cond(6,real) *\
*endif
*if(GenData(Degrees_Free,int)==5)
0.0 0.0 *\
*endif
*end
END_BOUNDARY CONDITIONS
$-----

```

The next section deals with outputting a materials listing.

D) Materials listing declaration

As before, the first lines must be the title of the section and a commentary:

```
MATERIAL PROPERTIES
$ MATERIAL PROPERTIES FOR MULTILAMINATE
```

Next there is the loop sentence, this time concerning materials:

```
*loop materials
```

Then comes the line where the number of the material and its different properties are output:

```
*matnum() *MatProp(1) *MatProp(2) *MatProp(3) *MatProp(4)
```

Finally, the end of the section is signalled:

```
*end materials
END_MATERIAL PROPERTIES
$-----
```

The full set of commands is as follows:

```
MATERIAL PROPERTIES
$ MATERIAL PROPERTIES FOR MULTILAMINATE
*loop materials
    *matnum() *MatProp(1) *MatProp(2) *MatProp(3) *MatProp(4)
*end materials
END_MATERIAL PROPERTIES
$-----
```

The next section deals with generating an elements listing.

E) Elements listing declaration

First, we set the loop to the interval of the data.

```
*loop intervals
```

The next couple of lines indicate the starting of one section and the title of the example, taken from the first field in the interval data with an abbreviation on the label. They are followed by a comment explaining the type of data we are using.

```
LOADS
    TITLE: *IntvData(Charge_case)
$ LOAD TYPE
```

We begin by setting the condition as before. If one condition is assigned twice or more to the same element without including the `*CanRepeat` parameter in the `*Set Cond`, the condition will appear once; if the `*CanRepeat` parameter is present then the number of conditions that will appear is the number of times it was assigned to the condition.

```
*Set Cond Face-Load *elems *CanRepeat
```

Then, a condition checks if any element exists in the condition.

```
*if(CondNumEntities(int)>0)
```

Next is a title for the next section, followed by a comment for the user.

```
DISTRIBUTED ON FACES
$ LOADS DISTRIBUTED ON ELEMENT FACES
```

We assign the number of nodes to a variable.

```
$ NUMBER OF NODES BY FACE NODGE = 2
$ LOADED FACES AND FORCE VALUES
*loop elems *OnlyInCond
    ELEMENT=*elemsnum(), CONNECTIV *globalnodes
    *cond(1) *cond(1) *cond(2) *cond(2)
*end elems
END_DISTRIBUTED ON FACES
*endif
```

The final section deals with outputting a list of the nodes and their conditions.

F) Nodes and its conditions listing declaration

As for previous sections, the first thing to do set the conditions.

```
*Set Cond Point-Load *nodes
```

As in the previous section, the next loop will only be executed if there is a condition in the selection.

```
*if(CondNumEntities(int)>0)
```

Here begins the loop over the nodes.

```
PUNCTUAL ON NODES
```

```
*loop nodes *OnlyInCond
  *NodesNum *cond(1) *cond(2) *\
```

The next `*if` sentences determine the output writing of the end of the line.

```
*if(GenData(Degrees_Free,int)>=3)
*cond(3) *\
*endif
*if(GenData(Degrees_Free,int)==5)
0 0 *\
*endif
*end nodes
```

To end the section, once again you have to include the end label and the closing `*endif`.

```
END_PUNCTUAL ON NODES
*endif
```

Finally, a message is written if the value of the second field in the interval data section inside the problem file is equal to `'si'` (yes).

```
*if(strcasecmp(IntvData(2),"Si")==0)
SELF_WEIGHT
*endif
```

To signal the end of this part of the forces section, the following line is entered.

```
END_LOADS
```

Before the end of the section it remains to tell the solver what the postprocess file will be. This information is gathered from the `*IntvData` command. The argument that this command receives (3) specifies that the name of the file is in the third field of the loop iteration of the interval.

```
$-----
$POSTPROCESS FILE FEMV = *IntvData(3)
```

To end the forces interval loop the `*end` command is entered.

```
$-----
*end nodes
```

Finally, the complete file is ended with the sentence required by the solver.

```
END_CASEF $-----
```

The preceding section is compiled completely into the following lines:

```
*Set Cond Point-Load *nodes
*if(CondNumEntities(int)>0)
PUNCTUAL ON NODES
*loop nodes *OnlyInCond
    *NodesNum *cond(1) *cond(2) *\
*if(GenData(Degrees_Free,int)>=3)
*cond(3) *\
*endif
*if(GenData(Degrees_Free,int)==5)
0 0 *\
*endif
*end
END_PUNCTUAL ON NODES
*endif
*if(strcasecmp(IntvData(2),"Si")==0)
SELF_WEIGHT
```

```

*endif
END_LOADS
$-----
$POSTPROCESS FILE
FEMV = *IntvData(3)
$-----
*end nodes
END_CALSEF
$-----

```

This is the end of the template file example.

Executing an external program

Once all the problem type files are finished (.cnd, .mat, .prb, .sim, .bas files), you can run the solver. You may wish to run it directly from inside GiD.

To do so, it is necessary to create the file `problem_type_name.bat` in the Problem Type directory. This must be a shell script that can contain any type of information and that will be different for every operating system. When you select the `Caluculate` option in GiD Preprocess this shell script is executed (see [CALCULATE](#)).

Because the .bat file will be different depending on the operating system, it is possible to create two files: one for Windows and another for Unix/Linux. The Windows file has to be called: `problem_type_name.win.bat`; the Unix/Linux file has to be called: `problem_type_name.unix.bat`.

If GiD finds a .win.bat or .unix.bat file, the file `problem_type_name.bat` will be ignored.

If a .bat file exists in the problem type directory when choosing `Start` in the calculations window, GiD will automatically write the analysis file inside the example directory assigning the name `project_name.dat` to this file (if there are more files, the names `project_name-1.dat` ... are used). Next, this shell script will be executed. GiD will assign three arguments to this script:

- **1st argument:** `project_name` (name of the current project);
- **2nd argument:** `c:\a\b\c\project_name.gid` (path of the current project);

- **3rd argument:** `c:\a\b\c\problem_type_name.gid` (path of the problem type selected);

Among other utilities, this script can move or rename files and execute the process until it finishes.

Note1: This file must have the executable flag set (see the UNIX command `chmod`) in UNIX systems.

Note2: GiD sets as the current directory the model directory (example: `c:\examples\test1.gid`) just before executing the `.bat` file. Therefore, the lines (`cd $directory`) are not necessary in the scripts.

Commands accepted by the GiD command.exe

The keywords are as follows:

- `%`
- `Shift`
- `Rem`
- `Chdir (Cd)`
- `Del (Delete, Erase)`
- `Copy`
- `Rename (Ren, Move)`
- `Mkdir (Md)`
- `Set`
- `Echo (@echo)`
- `If`
- `Call`
- `Goto`
- `:`
- `Type`

Unknown instructions will be executed as from an external file.

Not all the possible parameters and modifiers available in the operating system are implemented in the GiD executable command.exe.

Note: At the moment, `command.exe` is only used in Windows operating systems as an alternative to `command.com` or `cmd.exe`. With the **GiD** `command.exe` some of the disadvantages of Windows can be avoided (the limited length of parameters, temporary use of letters of virtual units that sometimes cannot be eliminated, fleeting appearance of the console window, etc).

If GiD finds the file `command.exe` located next to `gid.exe`, it will be used to interpret the `*.bat` file of the problem type; if the file `command.exe` cannot be found, the `*.bat` file will be interpreted by the windows `command.com`.

If conflicts appear by the use of some instruction still not implemented in the **GiD** `command.exe`, it is possible to rename the `command.exe` file, so that GiD does not find it, and the operating system `command.com` is used.

%

Returns the value of a variable.

`%number`

`%name%`

Parameters:

number

The number is the position (from 0 to 9) of one of the parameters which the `*.bat` file receives.

name

The name of an environment variable. That variable has to be declared with the instruction "set".

Note: GiD sends three parameters to the `*.bat` file: `%1`, `%2`, `%3`

`%1` is the name of the current project (`project_name`)

`%2` is the path of the current project (`c:\a\b\c\project_name.gid`)

`%3` is path of the problem type (`c:\a\b\c\problem_type_name.gid`)

For example, if GiD is installed in `c:\gidwin`, the "problemtypes" name is `cmass2d.gid` and the project is `test.gid`, located in `c:\temp` (the project is a directory called `c:\temp\test.gid` with some files inside), parameters will have the following values:

`%1 test`

`%2 c:\temp\test.gid`

`%3 c:\gidwin\problemtypes\cmass2d.gid`

Note: It is possible that the file and directory names of these parameters are in the short mode Windows format. So, parameter %3 would be: c:\GIDWIN\PROBLE~\CMAS2D.GID.

Examples:

```
echo %1 > %2\%1.txt
echo %TEMP% >> %1.txt
```

Shift

The shift command changes the values of parameters %0 to %9 copying each parameter in the previous one. That is to say, value %1 is copied to %0, value %2 is copied to %1, etc.

Shift

Parameter:

None.

Note: The shift command can be used to create a batch program that accepts more than 10 parameters. If it specifies more than 10 parameters in the command line, those that appear after tenth (%9) will move to parameter %9 one by one.

Rem

Rem is used to include comments in a *.bat file or in a configuration file.

```
rem [comment]
```

Parameter:

comment

Any character string.

Note: Some comments are GiD commands.

Chdir (Cd)

Changes to a different directory.

```
chdir [drive:path] [...] -or- cd [drive:path] [...]
```

Parameters:*[drive:path]*

Disk and path of the new directory.

[..]

Goes back one directory. For example if you are within the `C:\WINDOWS\COMMAND>` directory this would take you to `C:\WINDOWS>`.

Note: When GiD calls the `*.bat` file, the path of the project is the current path, so it is not necessary to use `cd %2` at the beginning of the `*.bat` file.

Examples:

```
chdir e:\tmp cd ..
```

Delete (Del, Erase)

Command used to delete files permanently from the computer.

```
delete[drive:][path] fileName [fileName]
```

Parameters:

[drive:][path] fileName [fileName] Parameters that specify the location and the name of the file that has to be erased from disk. Several file names can be given.

Note: Files will be eliminated although they have the hidden or read only flag. Use of wildcards is not allowed. For example `del *.*` is not valid. File names must be separated by spaces and if the path contains blank spaces, the path should be inside inverted commas (the short path without spaces can also be used).

Examples:

```
delete %2\%1\file.cal  
del C:\tmp\fa.dat C:\tmp\fb.dat  
del "C:\Program files\test 4.txt"
```

Copy

Copies one or more files to another location.

```
copy source [+ source [+ ...]] destination
```

Parameters:

source Specifies the file or files to be copied.

destination Specifies the filename for the new file(s).

To append files, specify a single file for destination, but multiple files for source (using the `file1 + file2 + file3 format`).

Note: If the destination file already exists, it will be overwritten without prompting whether or not you wish to overwrite it.

File names must be separated by spaces. If the destination only contains the path but not the filename, the new name will be the same as the source filename.

Examples:

```
copy f1.txt f2.txt
```

```
copy f1.txt c:\tmp
```

```
rem if directory c:\tmp exists, c:\tmp\f1.txt will be created, if it  
does not exist, file c:\tmp will be created.
```

```
copy a.txt + b.txt + c.txt abc.txt
```

Rename (Ren, Move)

Used to rename files and directories from the original name to a new name.

```
rename [drive:][path] fileName1 fileName2
```

Parameter:

[drive:][path] fileName1 Specifies the path and the name of the file which is to be renamed.

fileName2 Specifies the new name file.

Note: If the destination file already exists, it will be overwritten without prompting whether or not you wish to overwrite it. Wildcards are not accepted (*,?), so only one file can be renamed every time. Note that you cannot specify a new drive for your destination. A directory can be renamed in the same way as if it was a file.

Examples:

```
Rename fa.txt fa.dat
```

```
Rename "c:\Program Files\fa.txt" c:\tmp\fa.txt
```

```
Rename c:\test.gid c:\test2.gid
```

Mkdir (md)

Allows you to create your own directories.

```
mkdir[drive:]pathmd [drive:]path
```

Parameter:

drive: Specifies the drive where the new directory has to be created.

path Specifies the name and location of the new directory. The maximum length of the path is limited by the file system.

Note: mkdir can be used to create a new path with many new directories.

Examples:

```
mkdir e:\tmp2  
mkdir d1\d2\d3
```

Set

Displays, sets, or removes Windows environment variables.

```
set variable=[string]
```

Parameters:

variable Specifies the environment-variable name.

string Specifies a series of characters to assign to the variable.

Note: The set command creates variables which can be used in the same way as the variables %0 to %9. Variables %0 to %9 can be assigned to new variables using the set command.

To get the value of a variable, the variable has to be written inside two % symbols. For example, if the environment-variable name is V1, its value is %V1%. Variable names are not case-sensitive.

Examples:

```
set basename = %1  
set v1 = my text
```

Echo (@echo)

Displays messages.

```
echo [message]
```

Parameters:

message Specifies the text that will be displayed in the screen.

Note: The message will not be visible because the console is not visible, since GiD hides it. Therefore, this command is only useful if the output is redirected to a file (using > or >>). The symbol > sends the text to a new file, and the symbol >> sends the text to a file if it already exists. The if and echo commands can be used in the same command line.

Examples:

```
Echo %1 > out.txt
```

```
Echo %path% >> out.txt
```

```
If Not Exist %2\%1.flavia.res Echo "Program failed" >> %2\%1.err
```

If

Executes a conditional sentence. If the specified condition is true, the command which follows the condition will be executed; if the condition is false, the next line is ignored.

- *if[not] exist fileName command*
- *if[not] string1==string2 command*
- *if[not] errorlevel number command*

Parameters:

not Specifies that the command has to be executed only if the condition is false.

exist file Returns true if file exists.

command Is the command that has to be executed if the condition returns true.

string1==string2 Returns true if *string1* and *string2* are equal. It is possible to compare two strings, or variables (for example, %1).

errorlevel number Returns true if the last program executed returned a code equal to or bigger than the number specified.

Note: Exist can also be used to check if a directory exists.

Examples:

```
If exist sphere.igs echo File exists >> out.txt
```

```
if not exist test.gid echo Dir does not exist >> out.txt
if %1 == test echo Equal %1 >> out.txt
```

Call

Executes a new program.

```
call[drive:][path] file [parameters]
```

Parameter:

[drive:][path] file Specifies the location and the name of the program that has to be executed.

parameters Parameters required by the program executed.

Note: The program can be *.bat file, a *.exe file or a *.com file. If the program does a recursive call, some condition has to be imposed to avoid an endless curl.

Examples:

```
call test.bat %1
call gid.exe -n -PostResultsToBinary %1.flavia.res %1.flavia.bin
```

Goto

The execution jumps to a line identified by a label.

```
goto label
```

Parameter:

label It specifies a line of the *.bat file where the execution will continue. That label must exist when the Goto command is executed. A label is a line of the *.bat file and starts with (:).

Goto is often used with the command if, in order to execute conditional operations. The Goto command can be used with the label :EOF to make the execution jump to the end of the *.bat file and finish.

Note: The label can have more than eight characters and there cannot be spaces between them. The label name is not case-sensitive.

Example:

```
goto :EOF
if exist %1.err goto end
```

```
...  
:end
```

```
:
```

Declares a label.

```
:labelName
```

Parameter:

labelName A string which identifies a line of the file, so that the `Goto` command can jump there. The label line will not be executed.

Note: The label can have more than eight characters and there cannot be spaces between them. The label name is not case-sensitive.

Examples:

```
:my_label  
:end
```

Type

Displays the contents of text files.

```
type[drive:][path] fileName
```

Parameters:

[*drive:*][*path*] *fileName* Specifies the location and the name of the file to be displayed. If the file name contains blank spaces it should be inside inverted commas ("*file_name*").

Note: The text will not be visible because the console is not visible, since GiD hides it. Therefore, this command is only useful if the output is redirected to a file (using `>` or `>>`). The symbol `>` sends the text to a new file, and the symbol `>>` sends the text to a file if it already exists. It is recommended to use the copy command instead of type.

In general, the `type` command should not be used with binary files.

Examples:

```
type %2\%1.dat > %2\%1.txt
```

Showing feedback when running the solver

The information about what is displayed when `Output view:` is pressed is also given here. To determine what will be shown, the script must include a comment line in the following form:

For Windows :

```
rem    OutputFile: %1.log
```

For Linux/Unix:

```
#    OutputFile: $1.log
```

where `$1.log` means to display in that window a file whose name is: `project_name.log`. The name can also be an absolute name like `output.dat`. If this line is omitted, when you press `Output view:`, nothing will be displayed.

Managing errors

A line of code like

```
rem ErrorFile: %1.err    (for Windows)
or
# ErrorFile: $1.err      (for Linux/UNIX)
```

included in the `.bat` file means that the given filename is the error file. At the end of the execution of the `.bat` file, if the errorfile does not exist or is zero, execution is considered to be successful. If not, an error window appears and the contents of the error file are considered to be the error message. If this line exists, GiD will delete this file just before calculating to avoid errors with previous calculations.

A comment line like

```
rem WarningFile: %1.err
or
# WarningFile: $1.err
```

included in the `.bat` file means that the given filename is the warning file. This file stores the warnings that may appear during the execution of the `.bat` file.

Examples

Here are two examples of easy scripts to do. One of them is for Unix/Linux machines and the other one is for MS-Dos or Windows.

- **UNIX/Linux example:**

```
#!/bin/csh set basename = $1
set directory = $2
set ProblemDirectory = $3
#   OutputFile: $1.log           IT IS USED BY GiD
#   ErrorFile: $1.err           IT IS USED BY GiD
rm -f $basename.flavia.res
$ProblemDirectory/myprogram $basename
mv $basename.post $basename.flavia.res
```

- **MS-DOS/Windows example:**

```
rem basename=%1                JUST INFORMATIVE
rem directory=%2               JUST INFORMATIVE
rem ProblemDirectory=%3       JUST INFORMATIVE
rem   OutputFile: %1.log      IT IS USED BY GiD
rem   ErrorFile: %1.err      IT IS USED BY GiD
del %1.flavia.res%3\myprogram %1
move %1.post %1.flavia.res
```

POSTPROCESS DATA FILES

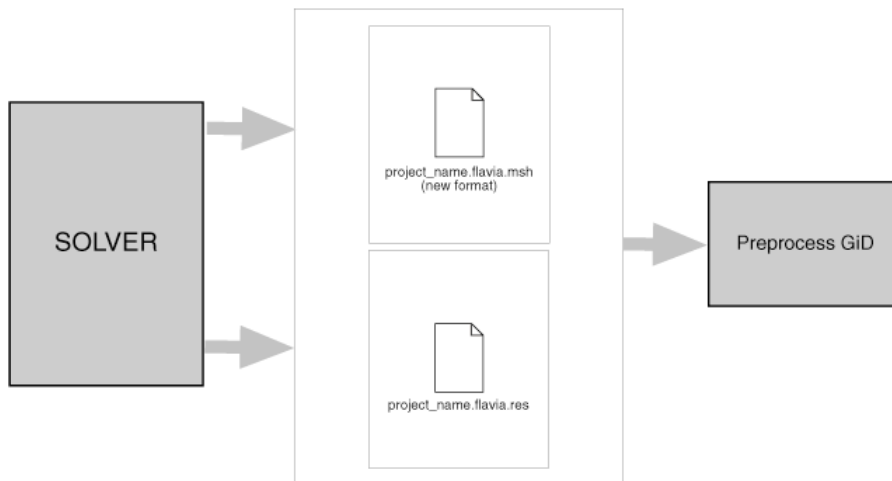
In GiD Postprocess you can study the results obtained from a solver program. The solver and GiD Postprocess communicate through the transfer of files. The solver program has to write the results to a file that must have the extension `.post.res`, or the old `.flavia.res`, and its name must be the project name.

The solver program can also send the postprocess mesh to GiD (though this is not mandatory), where it should have the extension `.post.msh`, or the old version `.flavia.msh`. If this mesh is not provided by the solver program, GiD uses the preprocess mesh in Postprocess.

The extensions `.msh` and `.res` are also allowed, but only files with the extensions `.post.res` or `.flavia.res` – and potentially `.post.msh` or `.flavia.msh` – will automatically be read by GiD when postprocessing the GiD project.

Postprocessing data files are ASCII files, and can be separated into two categories:

- Mesh Data File: `project_name.post.msh` (or `project_name.flavia.msh`) for volume and surface (3D or 2D) mesh information, and
- Results Data File: `project_name.post.res` (or `project_name.flavia.res`) for results information.
-



Note: `ProjectName.post.msh`, or the old `ProjectName.flavia.msh`, handles meshes of different element types: points, lines, triangles, quadrilaterals, tetrahedra and hexahedra. The old format, which only handles one type of element per file, is still supported inside GiD (see [Old postprocess mesh format](#)).

If a project is loaded into GiD, when changing to GiD Postprocess it will look for `ProjectName.post.res`, or the old `ProjectName.flavia.res`. If a mesh information file with the name `ProjectName.post.msh`, or the old `ProjectName.flavia.msh` is present, it will also be read, regardless of the information available from GiD Preprocess.

- **ProjectName.post.msh** (or the old **ProjectName.flavia.msh**): This file should contain nodal coordinates of the mesh and its nodal connectivities as well as the material of each element. At the moment, only one set of nodal coordinates can be entered. Different kinds of elements can be used but separated into different sets. If no material is supplied, GiD takes the material number to be equal to zero.
- **ProjectName.post.res** (or the old **ProjectName.flavia.res**): This second file must contain the nodal or gaussian variables. GiD lets you define as many nodal variables as desired, as well as several steps and analysis cases (limited only by the memory of the machine). The definitions of the Gauss points and the results defined on these points should also be written in this file.

The files are created and read in the order that corresponds to the natural way of solving a finite element problem: mesh, surface definition and conditions and finally, evaluation of the results. The format of the read statements is normally free, i.e. it is necessary only to separate them by spaces.

Thus, files can be modified with any format, leaving spaces between each field, and the results can also be written with as many decimal places as desired. Should there be an error, the program warns the user about the type of mistake found.

GiD reads all the information directly from the preprocessing files whenever possible in order to gain efficiency.

Postprocess mesh format: `ProjectName.post.msh`, `ProjectName.flavia.msh`

Note: This postprocess mesh format requires GiD version 6.0 or higher.

Comments are allowed and should begin with a '#'. Blank lines are also allowed.

To enter the mesh names and result names in another encoding, just write # encoding followed by your encoding type, for example:

```
# encoding utf-8
```

Inside this file one or more meshes can be defined, and each of them should:

- Begin with a header that follows this model:

```
MESH "mesh_name" dimension my_dimension Elemtype my_type Nnode  
my_number
```

where

- MESH, dimension, elemtype, nnode: are not case-sensitive;
- "mesh_name": is an optional name for the mesh;
- my_dimension: is 2 or 3 according to the geometric dimension of the mesh;
- my_type: describes the element type of this MESH. It should be one of the following: Point, Linear, Triangle, Quadrilateral, Tetrahedra, Hexahedra, Prism;
- my_number: the number of nodes of my_type element:

Point connectivity:



- ♦ Point: 1 node,

Line connectivities:



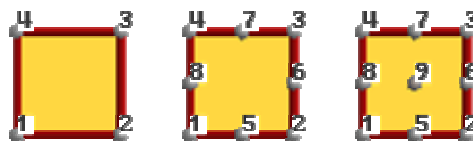
- ♦ Linear: 2 or 3 nodes,

Triangle connectivities:



- ♦ Triangle: 3 or 6 nodes,

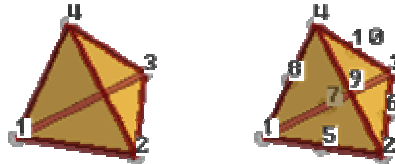
Quadrilateral connectivities:



- ♦ Quadrilateral: 4, 8 or 9 nodes,

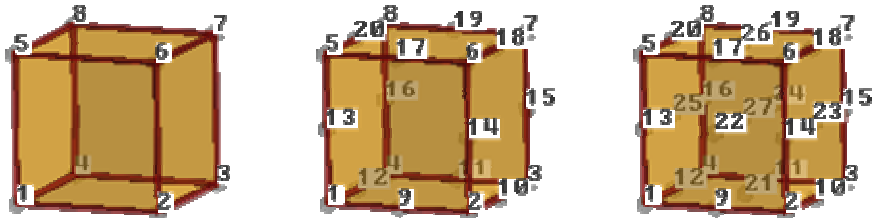
- Tetrahedron: 4 or 10 nodes,

Tetrahedron connectivities:



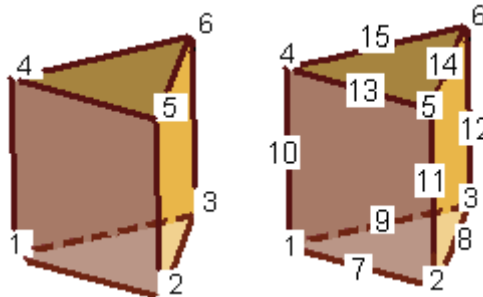
- Hexahedron: 8, 20 or 27 nodes.

Hexahedron connectivities:



- Prism: 6 or 15 nodes,

Prism connectivities:



Note: For elements of order greater than linear, the connectivities must be written in hierarchical order, i.e. the vertex nodes first, then the middle ones.

- Have an optional line describing its color with `# color R G B`, where R, G and B are the Red, Green and Blue components of the color written in integer format between 0 and 255, or in floating (real) format between 0.0 and 1.0. (Note that if 1 is found in the line it will be understood as integer, and so 1 above 255, rather than floating, and so 1 above 1.0)

`# color 127 127 0`

In this way different colors can be specified for several meshes, taking into account that the # color line must be between the MESH line and the Coordinates line.

- Be followed by the coordinates:

```
coordinates
1    0.0    1.0
3.0  . . .1000
-2.5    9.3    21.8
end coordinates
```

where

- the keywords `coordinates` and `end coordinates` are not case-sensitive;
- between these keywords there should be the nodal coordinates of all the meshes or the current one.

Note: If each MESH specifies its own coordinates, the node number should be unique, for instance, if MESH "mesh one" uses nodes 1..100, and MESH "other mesh" uses 50 nodes, they should be numbered from 101 up.

- Be followed by the elements connectivity

```
elements
#el_num node_1 node_2 node_3 material
1      1      2      3      215
. . .
1000    32    48    23    215
end elements
```

where

- the keywords `elements` and `end elements` are not case-sensitive;
- between these keywords there should be the nodal connectivities for the `my_type` elements.

Note: On elements of order greater than linear, the connectivities must be written in hierarchical order, i.e. the vertex nodes first, then the middle ones;

- there is optionally a material number.

Mesh example

This example clarifies the description:

```
#mesh of a table
MESH "board" dimension 3 ElemType Triangle Nnode 3
# color 127 127 0
Coordinates
# node number    coordinate_x    coordinate_y    coordinate_z
    1             -5             3             -3
    2             -5             3             0
    3             -5             0             0
    4             -2             2             0
    5             -1.66667        3             0
    6             -5             -3            -3
    7             -2             -2             0
    8              0              0             0
    9             -5             -3             0
   10              1.66667        3             0
   11             -1.66667       -3             0
   12              2              2             0
   13              2             -2             0
   14              1.66667       -3             0
   15              5              3            -3
   16              5              3             0
   17              5              0             0
   18              5             -3            -3
   19              5             -3             0
end coordinates

#we put both material in the same MESH,
#but they could be separated into two MESH

Elements
# element  node_1    node_2    node_3    material_number
    5         19       17       13         3
    6          3        9        7         3
    7          2        3        4         3
```

```

      8      17      16      12      3
      9      12      16      10      3
     10      12      10       4      3
     11       7       9      11      3
     12       7      11      13      3
     13       2       4       5      3
     14       5       4      10      3
     15      19      13      14      3
     16      14      13      11      3
     17       3       7       4      3
     18      17      12      13      3
     19      13      12       8      4
     20      13       8       7      4
     21       7       8       4      4
     22       4       8      12      4
end elements

MESH      dimension 3 ElemType Linear  Nnode 2
Coordinates
#no coordinates then they are already in the first MESH
end coordinates

Elements
# element  node_1  node_2 material_number
      1       9       6       5
      2      19      18       5
      3      16      15       5
      4       2       1       5
end elements

```

Postprocess results format: ProjectName.post.res, ProjectName.flavia.res

Note: The new postprocess results format requires GiD version 6.1.4b or higher.

Note: Code developers can download the GiDpost tool from the GiD web page; this is a C/C++/Fortran library for creating postprocess files for GiD in both ASCII and compressed binary format.

This is the ASCII format description:

The first line of the files with results written in this new postprocess format should be:

```
GiD Post Results File 1.0
```

Comment lines are allowed and should begin with a '#'. Blank lines are also allowed.

Results files can also be included with the keyword `include`, for instance:

```
include "My Other Results File"
```

This is useful, for instance, for sharing several `GaussPoints` definitions and `ResultRangeTable` among different analyses.

This keyword '`include`' should be outside the **Blocks** of information.

There are several types of **Blocks** of information, all of them identified by a keyword:

- `GaussPoints`: Information about gauss points: name, number of gauss points, natural coordinates, etc.;
- `ResultRangesTable`: Information for the result visualization type **Contour Ranges**: name, range limits and range names;
- `Result`: Information about a result: name, analysis, analysis/time step, type of result, location, values;
- `ResultGroup`: several results grouped in one block. These results share the same analysis, time step, and location (nodes or gauss points).

Gauss Points

If Gauss points are to be included, they must be defined before the `Result` which uses them. Each Gauss points block is defined between the lines `GaussPoints` and `End GaussPoints`.

The structure is as follows, and should:

- Begin with a header that follows this model:

```
GaussPoints "gauss_points_name" Elementype my_type "mesh_name"
```

where

- GaussPoints, elementype: are not case-sensitive;
- "gauss_points_name": is a name for the gauss points set, which will be used as reference by the results that are located on these gauss points;
- my_type: describes which element type these gauss points are for, i.e. Point, Linear, Triangle, Quadrilateral, Tetrahedra or Hexahedra;
- "mesh_name": is an optional field. If this field is missing, the gauss points are defined for all the elements of type my_type. If a mesh name is given, the gauss points are only defined for this mesh.

- Be followed by the gauss points properties:

```
Number of Gauss Points: number_gauss_points_per_element
```

```
Nodes included
```

```
Nodes not included
```

```
Natural Coordinates: Internal
```

```
Natural Coordinates: Given
```

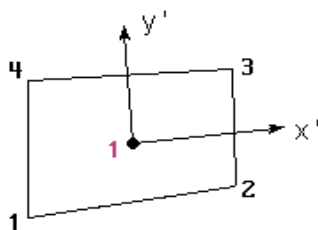
```
natural_coordinates_for_gauss_point_1 . . .
```

```
natural_coordinates_for_gauss_point_n
```

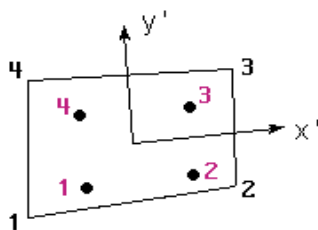
where

- Number of Gauss Points: number_gauss_points_per_element: is not case-sensitive and is followed by the number of gauss points per element that defines this set. If Natural Coordinates: is set to Internal, number_gauss_points_per_element should be one of:
 - ◆ 1, 3, 6 for Triangles;
 - ◆ 1, 4, 9 for Quadrilaterals;
 - ◆ 1, 4, 10 for Tetrahedra;
 - ◆ 1, 8, 27 for Hexahedra;
 - ◆ 1, 6 for Prisms; and
 - ◆ 1, ... n points equally spaced over lines.

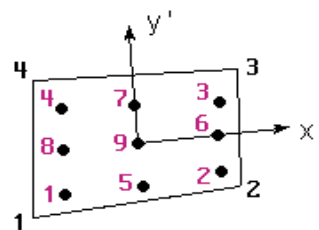
For triangles and quadrilaterals the order of the gauss points with **Internal** natural coordinates will be thus:



Internal coordinates:
 $(0, 0)$

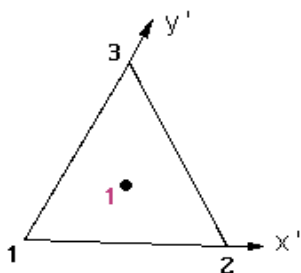


Internal coordinates:
 $a=0.57735027$
 $(-a, -a)$ $(a, -a)$
 (a, a) $(-a, a)$

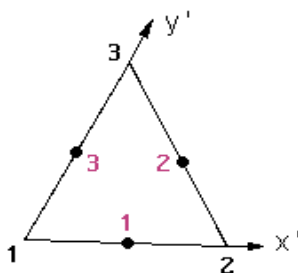


Internal coordinates:
 $a=0.77459667$
 $(-a, -a)$ $(a, -a)$ (a, a)
 $(-a, a)$ $(0, -a)$ $(a, 0)$
 $(0, a)$ $(-a, 0)$ $(0, 0)$

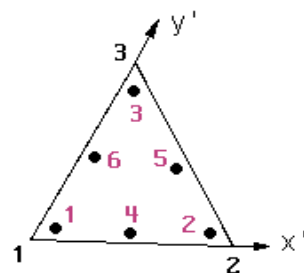
Gauss Points positions of the quadrature of Gauss-Legendre Quadrilaterals



Internal coordinates:
 $a=1/3$
 (a, a)



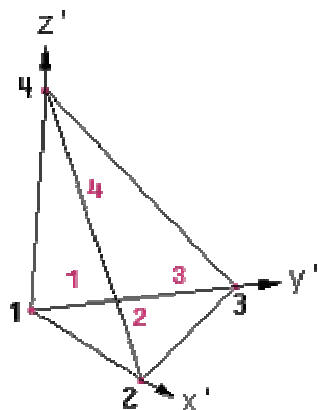
Internal coordinates:
 $a=1/2$
 $(a, 0)$ (a, a) $(0, a)$



Internal coordinates:
 $a=0.09157621$ $b=0.81684757$
 $c=0.44594849$ $d=0.10810301$
 (a, a) (b, a) (a, b)
 (c, d) (c, c) (d, c)

Gauss Points positions of the quadrature of Gauss for Triangles

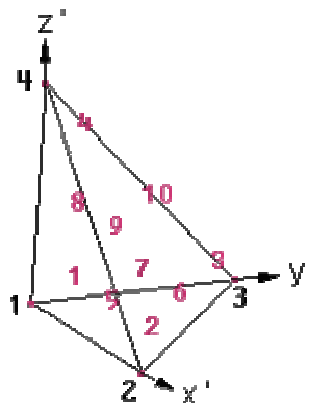
For tetrahedra the order of the **Internal** Gauss Points is thus:



Internal coordinates:

$a=0.58541020$, $b=0.13819660$

$(b, b, b) (a, b, b) (b, a, b) (b, b, a)$



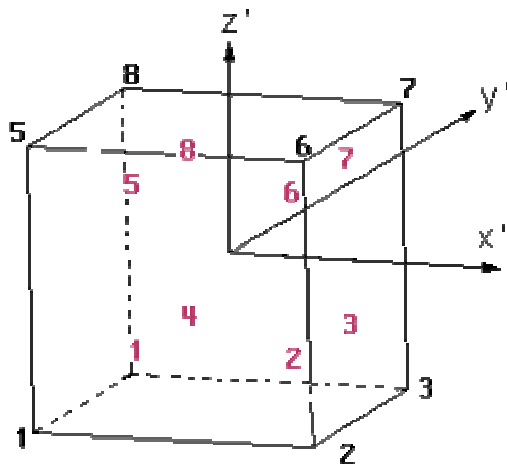
Internal coordinates:

$a=0.10810301$, $b=0.44594849$, $c=0.81684757$

$(a, a, a) (c, a, a) (a, c, a) (a, a, c) (b, a, a)$

$(b, b, a) (a, b, a) (a, a, b) (b, a, b) (a, b, b)$

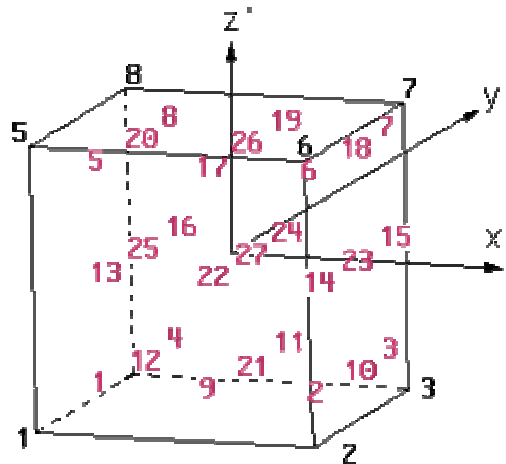
For hexahedra the order of the **Internal** Gauss Points is thus:



Internal coordinates:

$a=0.57735027$

$(-a, -a, -a) (a, -a, -a) (a, a, -a) (-a, a, -a)$



Internal coordinates:

$a=0.77459667$

$(-a, -a, -a) (a, -a, -a) (a, a, -a) (-a, a, -a)$

$(-a, -a, a) (a, -a, a) (a, a, a) (-a, a, a)$

$(-a, -a, a) (a, -a, a) (a, a, a) (-a, a, a)$

$(0, -a, -a) (a, 0, -a) (0, a, -a) (-a, 0, -a)$

$(-a, -a, 0) (a, -a, 0) (a, a, 0) (-a, a, 0)$

$(0, -a, a)$

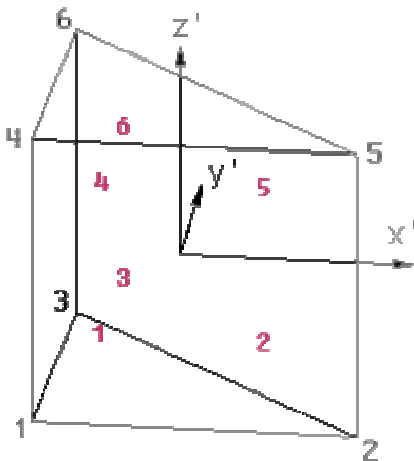
$(a, 0, a) (0, a, a) (-a, 0, a) (0, 0, -a)$

$(0, -a, 0) (a, 0, 0) (0, a, 0) (-a, 0, 0)$

$(0, 0, a)$

$(0, 0, 0)$

For prisms the order of the **Internal** Gauss Points is thus:



Internal coordinates:

$a=0.16666666$, $b=0.66666666$

$c=0.21132486$, $d=0.78867513$

$(a, a, b) (b, a, c) (a, b, c)$

$(a, a, d) (b, a, d) (a, b, d)$

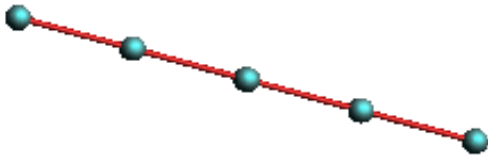
The **given** natural coordinates for Gauss Points should range:

- Between **0.0** and **1.0** for Triangles, Tetrahedra and Prisms, and
- Between **-1.0** and **1.0** for Quadrilaterals and Hexahedra.

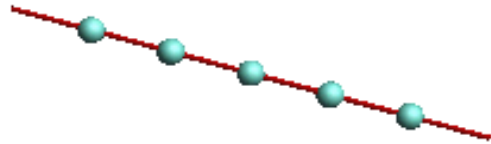
Note: If the natural coordinates used are the internal ones, almost all the Results visualization possibilities will have some limitations for tetrahedra and hexahedra with more than one gauss points. If the natural coordinates are given, these limitations are

extended to those elements with `number_gauss_points_per_element` not included in the list written above.

- `Nodes Included / Nodes not Included`: are not case-sensitive, and are only necessary for gauss points on Linear elements which indicate whether or not the end nodes of the Linear element are included in the `number_gauss_points_per_element` count.



Nodes included



Nodes not included

Note: By now, Natural Coordinates for linear elements cannot be "Given"

- `Natural Coordinates: Internal / Natural Coordinates: Given`: are not case-sensitive, and indicate whether the natural coordinates are calculated internally by GiD, or are given in the following lines. The natural coordinates should be written for each line and gauss point.
- End with this tail:

```
End GaussPoints
```

where

- `End GaussPoints`: is not case-sensitive.

Here is an example of results on Gauss Points:

```
GaussPoints "Board gauss internal" ElemType Triangle "board"
  Number Of Gauss Points: 3
  Natural Coordinates: internal
end gausspoints
```

Result Range Table

If a Result Range Table is to be included, it must be defined before the `Result` which uses it.

Each Result Range Table is defined between the lines `ResultRangesTable` and `End ResultRangesTable`.

The structure is as follows and should:

- Begin with a header that follows this model:

```
ResultRangesTable "ResultsRangeTableName"
```

where

- `ResultRangesTable`: is not case-sensitive;
- `"ResultsRangeTableName"`: is a name for the Result Ranges Table, which will be used as a reference by the results that use this Result Ranges Table.

- Be followed by a list of Ranges, each of them defined as follows:

```
Min_Value - Max_Value: "Range Name"
```

where

- `Min_value`: is the minimum value of the range, and may be void if the `Max_value` is given. If void, the minimum value of the result will be used;
- `Max_value`: is the maximum value of the range, and may be void if the `Min_value` is given. If void, the maximum value of the result will be used;
- `"Range Name"`: is the name of the range which will appear on legends and labels.

- End with this tail:

```
End ResultRangesTable
```

where

- `End ResultRangesTable`: is not case-sensitive.

Here are several examples of results range tables:

- Ranges defined for the whole result:

```
ResultRangesTable "My table"
# all the ranges are min <= res < max except
# the last range is min <= res <= max
    - 0.3: "Less"
    0.3 - 0.7: "Normal"
    0.7 -      : "Too much"
End ResultRangesTable
```

- Just a couple of ranges:

```
ResultRangesTable "My table"
    0.3 - 0.7: "Normal"
    0.7 - 0.9: "Too much"
End ResultRangesTable
```

- Or using the maximum of the result:

```
ResultRangesTable "My table"
    0.3 - 0.7: "Normal"
    0.7 -      : "Too much"
End ResultRangesTable
```

Result block

Each **Result** block is identified by a **Result** header, followed by several optional properties: component names, ranges table, and the result values, defined by the lines **Values** and **End Values**.

The structure is as follows and should:

- Begin with a header that follows this model:

```
Result "result name" "analysis name" step_value my_result_type
my_location "location name"
```


where

- o `Result`: is not case-sensitive;
- o `"result name"`: is a name for the `Result`, which will be used for menus;
- o `"analysis name"`: is the name of the analysis of this `Result`, which will be used for menus;
- o `step_value`: is the value of the step inside the analysis `"analysis name"`;
- o `my_type`: describes the type of the `Result`. It should be one of the following: `Scalar`, `Vector`, `Matrix`, `PlainDeformationMatrix`, `MainMatrix`, `LocalAxes`;
- o `my_location`: is where the `Result` is located. It should be one of the following: `OnNodes`, `OnGaussPoints`. If the `Result` is `OnGaussPoints`, a `"location name"` should be entered;
- o `"location name"`: is the name of the Gauss Points on which the `Result` is defined.

- Be followed (optionally) by `result` properties:

ResultRangesTable "Name of a result ranges table" **ComponentNames** "Name of Component 1", "Name of Component 2"

where

- o `ResultRangesTable`: is not case-sensitive;
- o `"Name of a result ranges table"`: is the name of the previously defined `Result Ranges Table`, which will be used if the `Contour Ranges` result visualization is chosen (see [Result Range Table](#));
- o `ComponentNames`: is not case-sensitive;
- o `"Name of Component 1", "Name of Component 2"`: are the names of the components of the results which will be used in GiD. The number of `Component Names` are:
 - ♦ One for a `Scalar Result`
 - ♦ Three for a `Vector Result`
 - ♦ Six for a `Matrix Result`
 - ♦ Four for a `PlainDeformationMatrix Result`
 - ♦ Six for a `MainMatrix Result`
 - ♦ Three for a `LocalAxes Result`

- End with the result values:

Values

```
node_or_elem_number component_1_value component_2_value
. . . node_or_elem_number component_1_value component_2_value
End Values
```

where

- Values: is not case-sensitive, and indicates the beginning of the results values section;
- The lines
 - ◆ node_or_elem_number component_1_value
component_2_value
 - ◆ . . .
 - ◆
 - ◆ node_or_elem_number component_1_value
component_2_value

are the values of the result at the related 'node_or_elem_number'.

The number of results values are limited thus:

- ◆ If the Result is located OnNodes, they are limited to the number of nodes defined in `ProjectName.flavia.msh`.
- ◆ If the Result is located OnGaussPoints "My GP", and if the Gauss Points "My GP" are defined for the mesh "My mesh", the limit is the number of gauss points in "My GP" multiplied by the number of elements of the mesh "My mesh".

For results in gauss points, each element must have 'ngauss' lines of results.

For example, if the number of gauss points is 3, then for an element, 3 lines of gauss point result must appear.

```
Values
1 1.155
   2.9
   3.955
End Values
```

Holes are allowed in any result. The element nodes with no result defined will not be drawn, i.e. they will appear transparent.

The number of components for each Result Value are:

- ◆ for Scalar results: one component
result_number_i scalar_value

- ♦ for Vector results: three components, with an optional fourth component for signed modules


```
result_number_i x_value y_value z_value
result_number_i x_value y_value z_value
signed_module_value
```
 - ♦ for Matrix results: three components (2D models) or six components (3D models)


```
2D: result_number_i Sxx_value Syy_value Sxy_value
3D: result_number_i Sxx_value Syy_value Szz_value
Sxy_value Syz_value Sxz_value
```
 - ♦ for PlainDeformationMatrix results: four components


```
result_number_i Sxx_value Syy_value Sxy_value
Szz_value
```
 - ♦ for MainMatrix results: twelve components


```
result_number_i Si_value Sii_value Siii_value
Vix_value Viy_value Viz_value Viix_value Viyy_value
Viiz_value Viiiix_value Viiiy_value Viiiiz_value
```
 - ♦ for LocalAxes results: three components describing the Euler angles


```
result_number_i euler_ang_1_value
euler_ang_2_value euler_ang_3_value
```
- End Values: is not case-sensitive, and indicates the end of the results values section.

Note: For `Matrix` and `PlainDeformationMatrix` results, the `Si`, `Sii` and `Siii` components are calculated by GiD, which represents the eigen values & vectors of the matrix results, and which are ordered according to the eigen value.

Result group

`Results` can be grouped into one block. These results belong to the same time step of the same analysis and are located in the same place. So all the results in the group are nodal results or are defined over the same set of gauss points.

Each `Result` group is identified by a `ResultGroup` header, followed by the results descriptions and its optional properties - such as components names and ranges tables, and the results values - all between the lines `Values` and `End Values`.

The structure is as follows and should:

- Begin with a header that follows this model

```
ResultGroup "analysis name" step_value my_location "location
name"
```

where

- **ResultGroup**: is not case-sensitive;
 - "analysis name": is the name of the analysis of this **ResultGroup**, which will be used for menus;
 - **step_value**: is the value of the step inside the analysis "analysis name";
 - **my_location**: is where the **ResultGroup** is located. It should be one of the following: **OnNodes**, **OnGaussPoints**. If the **ResultGroup** is **OnGaussPoints**, a "location name" should be entered;
 - "location name": is the name of the Gauss Points on which the **ResultGroup** is defined.
- Be followed by at least one of the results descriptions of the group

```
ResultDescription "result name"
my_result_type[:components_number]
ResultRangesTable "Name of a result ranges table"
ComponentNames "Name of Component 1", "Name of Component 2"
```

where

- **ResultDescription**: is not case-sensitive;
- "result name": is a name for the **Result**, which will be used for menus;
- **my_type**: describes the type of the **Result**. It should be one of the following: **Scalar**, **Vector**, **Matrix**, **PlainDeformationMatrix**, **MainMatrix**, or **LocalAxes**. The number of components for each type is as follows:
 - ♦ One for a **Scalar**: the_scalar_value
 - ♦ Three for a **Vector**: X, Y and Z
 - ♦ Six for a **Matrix**: Sxx, Syy, Szz, Sxy, Syz and Sxz
 - ♦ Four for a **PlainDeformationMatrix**: Sxx_value, Syy, Sxy and Szz
 - ♦ Twelve for a **MainMatrix**: Si, Sii, Siii, ViX, ViY, ViZ, ViiX, ViiY, ViiZ, ViiiX, ViiiY and ViiiZ
 - ♦ Three for a **LocalAxes**: euler_ang_1, euler_ang_2 and euler_ang_3

Following the description of the type of the result, an optional modifier can be appended to specify the number of components separated by a colon. It only makes sense to indicate the number of components on vectors and matrices:

- o Vector:2, Vector:3 or Vector:4, which specify:
 - ♦ Vector:2: X and Y
 - ♦ Vector:3: X, Y and Z
 - ♦ Vector:4: X, Y, Z and |Vector|

The default (Vector) is 3 components per vector.

- o Matrix:3 or Matrix:6, which specify:
 - ♦ Matrix:3: Sxx, Syy and Sxy
 - ♦ Matrix:6: Sxx, Syy, Szz, Sxy, Syz and Sxz

The default (Matrix) is 6 components for matrices.

Here are some examples:

```
ResultDescription "Displacements" Vector:2
```

```
ResultDescription "2D matrix" Matrix:3
```

```
ResultDescription "LineDiagramVector" Vector:4
```

and where (optional properties)

- o ResultRangesTable "Name of a result ranges table": is not case-sensitive, and is followed by the name of the previously defined Result Ranges Table which will be used if the Contour Ranges result visualization is chosen (see [Result Range Table](#));
 - o ComponentNames "Name of Component 1", "Name of Component 2": is not case-sensitive, and is followed by the names of the components of the results which will be used in GiD. The number of Component Names are:
 - ♦ One for a Scalar Result
 - ♦ Three for a Vector Result
 - ♦ Six for a Matrix Result
 - ♦ Four for a PlainDeformationMatrix Result
 - ♦ Six for a MainMatrix Result
 - ♦ Three for a LocalAxes Result
- End with the results values:

Values

```
location_1 result_1_component_1_value result_1_component_2_value
result_1_component_3_value result_2_component_2_value
result_2_component_2_value result_2_component_3_value
. . .
```

```
location_n result_1_component_1_value result_1_component_2_value
result_1_component_3_value result_2_component_2_value
result_2_component_2_value result_2_component_3_value
End Values
```

where

- Values: is not case-sensitive, and indicates the beginning of the results values section;
- The lines
 - ◆ location_1 result_1_component_1_value
result_1_component_2_value
result_1_component_3_value
result_2_component_2_value
result_2_component_2_value
result_2_component_3_value
 - ◆ . . .
 - ◆ location_n result_1_component_1_value
result_1_component_2_value
result_1_component_3_value
result_2_component_2_value
result_2_component_2_value
result_2_component_3_value

are the values of the various results described with `ResultDescription` for each location. All the results values for the location 'i' should be written in the same line 'i'.

The number of results values are limited thus:

- ◆ If the `Result` is located `OnNodes`, they are limited to the number of nodes defined in `ProjectName.post.msh`, or the old `ProjectName.flavia.msh`.
- ◆ If the `Result` is located `OnGaussPoints "My GP"`, and if the Gauss Points "My GP" are defined for the mesh "My mesh", the limit is the number of gauss points in "My GP" multiplied by the number of elements of the mesh "My mesh".

Holes are allowed. The element nodes with no result defined will not be drawn, i.e. they will appear transparent.

The number of components for each `ResultDescription` are:

- ◆ for Scalar results: one component `result_number_i scalar_value`
- ◆ for Vector results: three components `result_number_i x_value y_value z_value`
- ◆ for Matrix results: six components (3D models)3D: `result_number_i Sxx_value Syy_value Szz_value Sxy_value Syz_value Sxz_value`
- ◆ for `PlainDeformationMatrix` results: four components `result_number_i Sxx_value Syy_value Sxy_value Szz_value`

- ♦ for MainMatrix results: twelve components result_number_i Si_value Sii_value Siii_value Vix_value Viy_value Viz_value Viix_value Viyy_value Viiz_value Viiix_value Viiiy_value Viiiz_value
 - ♦ for LocalAxes results: three components describing the Euler angles result_number_i euler_ang_1_value euler_ang_2_value euler_ang_3_value
- End Values: is not case-sensitive, and indicates the end of the results group values section.

Note: Vectors in a ResultGroup always have three components.

Note: Matrices in a ResultGroup always have six components.

Note: All the results of one node or gauss point should be written on the same line.

Note: For Matrix and PlainDeformationMatrix results, the Si, Sii and Siii components are calculated by GiD, which represents the eigen values & vectors of the matrix results, and which are ordered according to the eigen value.

Nodal ResultGroup example:

ResultGroup "Load Analysis" 1 OnNodes

ResultDescription "Ranges test" Scalar

ResultRangesTable "My table"

ResultDescription "Scalar test" Scalar

ResultRangesTable "Pressure"

ResultDescription "Displacements" Vector

ComponentNames "X-Displ", "Y-Displ" "Z-Displ"

ResultDescription "Nodal Stresses" Matrix

ComponentNames "Sx", "Sy", "Sz", "Sxy", "Syz", "Sxz"

Values

```

  1  0.0      0.000E+00 0.000E+00  0.000E+00  0.0  0.550E+00  0.972E-01
-0.154E+00  0.0  0.0  0.0
  2  6.4e-01  0.208E-04 0.208E-04 -0.191E-04  0.0  0.506E+00  0.338E-01
-0.105E+00  0.0  0.0  0.0
  3  0.0      0.355E-04 0.355E-04 -0.376E-04  0.0  0.377E+00  0.441E-02
-0.547E-01  0.0  0.0  0.0
...
```

```

    115 7.8e-01 0.427E-04 0.427E-04 -0.175E-03 0.0 0.156E-01 -0.158E-01
-0.300E-01 0.0 0.0 0.0
    116 7.4e-01 0.243E-04 0.243E-04 -0.189E-03 0.0 0.216E-02 -0.968E-02
-0.231E-01 0.0 0.0 0.0

```

End Values

Gauss Points ResultGroup example:

GaussPoints "My Gauss" ElemType Triangle "2D Beam"

Number Of Gauss Points: 3

Natural Coordinates: Internal

End gausspoints

ResultGroup "Load Analysis" 1 OnGaussPoints "My Gauss"

ResultDescription "Gauss test" Scalar

ResultDescription "Vector Gauss" Vector

ResultDescription "Gauss Points Stresses" PlainDeformationMatrix

Values

```

    1 1.05    1 0                                0.0    -19.4607 -1.15932 -1.43171
-6.18601
    2.1      0 1                                0.0    -19.4607 -1.15932 -1.43171
-6.18601
    3.15     1 1                                0.0    -19.4607 -1.15932 -1.43171
-6.18601
    2 1.2     0 0                                0.0    -20.6207 0.596461 5.04752
-6.00727
    2.25     0 0                                0.0    -20.6207 0.596461 5.04752
-6.00727
    3.3      2.0855e-05 -1.9174e-05 0.0    -20.6207 0.596461 5.04752
-6.00727
    3 1.35   2.0855e-05 -1.9174e-05 0.0    -16.0982 -1.25991 2.15101
-5.20742
    2.4      2.0855e-05 -1.9174e-05 0.0    -16.0982 -1.25991 2.15101
-5.20742
    3.45     2.0855e-05 -1.9174e-05 0.0    -16.0982 -1.25991 2.15101
-5.20742
...

```



```

191 29.55  4.2781e-05 -0.00017594  0.0      -0.468376 12.1979 0.610867
3.51885
    30.6   4.2781e-05 -0.00017594  0.0      -0.468376 12.1979 0.610867
3.51885
    31.65  4.2781e-05 -0.00017594  0.0      -0.468376 12.1979 0.610867
3.51885
192 29.7   4.2781e-05 -0.00017594  0.0      0.747727 11.0624 1.13201
3.54303
    30.75  4.2781e-05 -0.00017594  0.0      0.747727 11.0624 1.13201
3.54303
    31.8   2.4357e-05 -0.00018974  0.0      0.747727 11.0624 1.13201
3.54303

```

End Values

Results example

Here is an example of results for the table in the previous example (see [Mesh example](#)):

GiD Post Results File 1.0

```
GaussPoints "Board gauss internal" ElemType Triangle "board"
```

```
  Number Of Gauss Points: 3
```

```
  Natural Coordinates: internal
```

```
end gausspoints
```

```
GaussPoints "Board gauss given" ElemType Triangle "board"
```

```
  Number Of Gauss Points: 3
```

```
  Natural Coordinates: Given
```

```
    0.2 0.2
```

```
    0.6 0.2
```

```
    0.2 0.6
```

```
End gausspoints
```

```
GaussPoints "Board elements" ElemType Triangle "board"
```

```
  Number Of Gauss Points: 1
```

```
  Natural Coordinates: internal
```

```
end gausspoints
```

```
GaussPoints "Legs gauss points" ElemType Linear
  Number Of Gauss Points: 5
  Nodes included
  Natural Coordinates: Internal
End Gausspoints
```

```
ResultRangesTable "My table"
# el ultimo rango es min <= res <= max
  - 0.3: "Less"
  0.3 - 0.9: "Normal"
  0.9 - 1.2: "Too much"
End ResultRangesTable
```

```
Result "Gauss element" "Load Analysis" 1 Scalar OnGaussPoints "Board
elements"
```

```
Values
  5      0.00000E+00
  6      0.20855E-04
  7      0.35517E-04
  8      0.46098E-04
  9      0.54377E-04
 10      0.60728E-04
 11      0.65328E-04
 12      0.68332E-04
 13      0.69931E-04
 14      0.70425E-04
 15      0.70452E-04
 16      0.51224E-04
 17      0.32917E-04
 18      0.15190E-04
 19      -0.32415E-05
 20      -0.22903E-04
 21      -0.22919E-04
 22      -0.22283E-04
```

```
End Values
```

```
Result "Displacements" "Load Analysis" 1 Vector OnNodes
ResultRangesTable "My table"
ComponentNames "X-Displ", "Y-Displ", "Z-Displ"
```

Values

1	0.0	0.0	0.0
2	-0.1	0.1	0.5
3	0.0	0.0	0.8
4	-0.04	0.04	1.0
5	-0.05	0.05	0.7
6	0.0	0.0	0.0
7	-0.04	-0.04	1.0
8	0.0	0.0	1.2
9	-0.1	-0.1	0.5
10	0.05	0.05	0.7
11	-0.05	-0.05	0.7
12	0.04	0.04	1.0
13	0.04	-0.04	1.0
14	0.05	-0.05	0.7
15	0.0	0.0	0.0
16	0.1	0.1	0.5
17	0.0	0.0	0.8
18	0.0	0.0	0.0
19	0.1	-0.1	0.5

End Values

Result "Gauss displacements" "Load Analysis" 1 Vector OnGaussPoints
"Board gauss given"

Values

5	0.1	-0.1	0.5
	0.0	0.0	0.8
	0.04	-0.04	1.0
6	0.0	0.0	0.8
	-0.1	-0.1	0.5
	-0.04	-0.04	1.0
7	-0.1	0.1	0.5
	0.0	0.0	0.8
	-0.04	0.04	1.0
8	0.0	0.0	0.8
	0.1	0.1	0.5
	0.04	0.04	1.0
9	0.04	0.04	1.0
	0.1	0.1	0.5

	0.05	0.05	0.7
10	0.04	0.04	1.0
	0.05	0.05	0.7
	-0.04	0.04	1.0
11	-0.04	-0.04	1.0
	-0.1	-0.1	0.5
	-0.05	-0.05	0.7
12	-0.04	-0.04	1.0
	-0.05	-0.05	0.7
	0.04	-0.04	1.0
13	-0.1	0.1	0.5
	-0.04	0.04	1.0
	-0.05	0.05	0.7
14	-0.05	0.05	0.7
	-0.04	0.04	1.0
	0.05	0.05	0.7
15	0.1	-0.1	0.5
	0.04	-0.04	1.0
	0.05	-0.05	0.7
16	0.05	-0.05	0.7
	0.04	-0.04	1.0
	-0.05	-0.05	0.7
17	0.0	0.0	0.8
	-0.04	-0.04	1.0
	-0.04	0.04	1.0
18	0.0	0.0	0.8
	0.04	0.04	1.0
	0.04	-0.04	1.0
19	0.04	-0.04	1.0
	0.04	0.04	1.0
	0.0	0.0	1.2
20	0.04	-0.04	1.0
	0.0	0.0	1.2
	-0.04	-0.04	1.0
21	-0.04	-0.04	1.0
	0.0	0.0	1.2
	-0.04	0.04	1.0
22	-0.04	0.04	1.0
	0.0	0.0	1.2

```

        0.04  0.04  1.0
End Values

Result "Legs gauss displacements" "Load Analysis" 1 Vector
OnGaussPoints "Legs gauss points"
Values
  1  -0.1  -0.1  0.5
     -0.2  -0.2  0.375
     -0.05 -0.05  0.25
     0.2   0.2   0.125
     0.0   0.0   0.0
  2   0.1  -0.1  0.5
     0.2  -0.2  0.375
     0.05 -0.05  0.25
     -0.2  0.2   0.125
     0.0   0.0   0.0
  3   0.1   0.1  0.5
     0.2   0.2  0.375
     0.05  0.05  0.25
     -0.2  -0.2  0.125
     0.0   0.0   0.0
  4  -0.1   0.1  0.5
     -0.2   0.2  0.375
     -0.05  0.05  0.25
     0.2   -0.2  0.125
     0.0   0.0   0.0
End Values

```

Re-meshing and adaptivity

If the same meshes are used for all the analyses, the following section can be skipped.

A new concept has been introduced in Postprocess: *Group*, which allows the postprocessing of problems which require re-meshing or adaptive meshes.

Meshes that belong to a group should be defined between the following highlighted commands

```

Group "group name"
MESH "mesh_name" dimension ...
...
end elements

MESH "another_mesh" ...
...
end elements
end group

```

Results which refer to one of the groups should be written between these highlighted commands

```

OnGroup "group name"
Result "result name"
...
end values
...
end ongroup

```

Note: GiD versions 7.7.3b and later only allow one group at a time, i.e. only one group can be defined across several steps of the analysis. Care should be taken so that groups do not overlap inside the same step/analysis.

For instance, an analysis which is 10 steps long:

- For steps 1, 2, 3 and 4: an 'environment' mesh of 10000 elements and a 'body' mesh of 10000 elements are used

```

MESH "environment"
... Coordinates
...
10000 ...
end elements
MESH "body" ...
...
20000 ...
end elements

```

and its results

GiD Post Results File 1.0

```
...
Results "result 1" "time" 1.0 ...
...
Results "result 1" "time" 2.0 ...
...
Results "result 1" "time" 3.0 ...
...
Results "result 1" "time" 4.0 ...
...
end values
```

- For steps 5, 6, 7 and 8: with some refinement, the 'environment' mesh now being used has 15000 elements and the 'body' mesh needs 20000 elements

```
MESH "environment"
...
Coordinates
...
15000 ...
end elements
MESH "body" ...
...
35000 ...
end elements
```

and its results are

```
GiD Post Results File 1.0
...
Results "result 1" "time" 5.0 ...
...
Results "result 1" "time" 6.0 ...
...
Results "result 1" "time" 7.0 ...
...
Results "result 1" "time" 8.0 ...
...
end values
```

- For steps 9 and 10: the last meshes to be used are of 20000 and 40000 elements, respectively

```
MESH "environment" ...
Coordinates
...
20000 ...
end elements
MESH "body" ...
...
60000 ...
end elements
```

and its results are

```
GiD Post Results File 1.0
...
Results "result 1" "time" 9.0 ...
...
Results "result 1" "time" 10.0 ...
...
end values
```

There are two ways to postprocess this:

- store the information in three pairs (or three binary files), thus:
 - ◆ steps_1_2_3_4.post.msh and steps_1_2_3_4.post.msh (or steps_1_2_3_4.post.bin)
 - ◆ steps_5_6_7_8.post.msh and steps_5_6_7_8.post.msh (or steps_5_6_7_8.post.bin)
 - ◆ steps_9_10.post.msh and steps_9_10.post.msh (or steps_9_10.post.bin)

and use the `Open multiple` option (see [Files menu](#)) to selected the six (or three) files; or

- write them in only two files (one in binary) by using the **Group** concept:
 - ◆ all_analysis.post.msh (note the group – end group pairs)

```
Group "steps 1, 2, 3 and 4"
```



```
MESH "environment" ...
```

```
...
```

```
MESH "body" ...
```

```
...
```

```
end group
```

```
Group "steps 5, 6, 7 and 8"
```

```
MESH "environment" ...
```

```
...
```

```
MESH "body" ...
```

```
...
```

```
end group
```

```
Group "steps 9 and 10"
```

```
MESH "environment" ...
```

```
...
```

```
MESH "body" ...
```

```
...
```

```
end group
```

```
and
```

- ♦ `all_analysis.post.res` (note the ongroup – end ongroup pairs)

```
GiD Post Results File 1.0
```

```
OnGroup "steps 1, 2, 3 and 4"
```

```
...
```

```
Results "result 1" "time" 1.0 ...
```

```
...
```

```
Results "result 1" "time" 2.0 ...
```

```
...
```

```
Results "result 1" "time" 3.0 ...
```

```
...
```

```
Results "result 1" "time" 4.0 ...
```

```
...
```

```
end ongroup
```

```
OnGroup "steps 5, 6, 7 and 8"
```

```
...
```

```
Results "result 1" "time" 5.0 ...
```

```
...
```

```
Results "result 1" "time" 6.0 ...
```

```

...
Results "result 1" "time" 7.0 ...
...
Results "result 1" "time" 8.0 ...
...
end ongroup

OnGroup "steps 9 and 10"
...
Results "result 1" "time" 9.0 ...
...
Results "result 1" "time" 10.0 ...
...
end ongroup

```

and use the normal `Open` option.

Old postprocess results format

This file is a complete list of the dumped results, where each result will be organized as follows:

Set 1: Header. Results description

The total number of lines in this set is 1, composed of 1 character string, 1 integer, 1 real, 1 optional character string, which depends on the first integer, plus 3 integers:

```
descr_menu load_type step_val [load_desc] data_typedata_loc desc_comp
["gauss_points_name"]
```

where:

- `descr_menu` = results title that will appear on the menus (maximum 15 characters without any blank spaces).
- `load_type` = type of analysis carried out to obtain this result:
 - 1 - time analysis (Time Step).
 - 2 - load analysis (Load Step).
 - 3 - frequency analysis (Frequency).
 - 4 - user defined analysis (User Step).
- `step_val` = number of steps inside the analysis.

- `load_desc` = description, without any blank spaces, of the analysis that will appear on the menus. This field must only be specified when the analysis is defined by the user (`load_type` = 4).
- `data_type` = kind of results:
 - 1 - scalar.
 - 2 - vector.
 - 3 - matrix.
 - 4 - 2D plane deformation matrix
 - 5 - Main stresses (3 modules and 3 vectors)
 - 6 - Euler angles (for local axes)
- `data_loc` = position of the data: 1 - on the nodes; 2 - on the Gauss points.
- `desc_comp` = specification of the existence of a description of each component that will be displayed as a menu's button:
 - 0 - no description (inside GiD, the program itself creates the description for the corresponding components).
 - 1 - there will be a description, without any blank spaces, of the components with one component per line.
- "`gauss_points_name`": optional field that specifies the set of gauss points to be used (new gauss point format see [Gauss Points](#)). If not specified the general gauss points definition will be used (old format).

Set 2: Description of the components

The description of each one of the result's components, without any blank spaces, should be entered here if needed, one per line. The number of lines will be as follows:

- One line if it is a scalar.
- Three lines if it is vector.
- Six lines if it is a matrix.
- Four lines if it is a 2D plane deformation matrix.
- Six lines if it is Main Stresses.
- Three lines if it is a Euler angles result.

This description will appear in different menus to select the variable to be displayed at each stage.

Note: GiD also supports 2D results types, so description components can be two for vectors, and three or four for matrix and plane strain analysis, respectively.

Set 3: Results

The total number of lines in this set is the total number of points if `data_loc = 1` or the total number of elements multiplied by the number of Gauss points per element if `data_loc = 2`. The definition of the results is itemized below.

- **Scalar:** Each line is composed of one integer plus one real number:

```
i result[i]
```

where:

- `i` = node or Gauss point number.
- `result[i]` = value of the result on the node or Gauss point number `i`.

- **Vector:** Each line is composed of 1 integer plus 3 real numbers:

```
i result_x[i] result_y[i] result_z[i] result_m[i]
```

where:

- `i` = node or Gauss point number.
- `result_x[i]` = value of the x_component of the result on the node or Gauss point number `i`.
- `result_y[i]` = value of the y_component of the result on the node or Gauss point number `i`.
- `result_z[i]` = value of the x_component of the result on the node or Gauss point number `i`. Optional if a 2D result type is specified. Should be specified if `result_m[i]` is given.
- `result_m[i]` = value of the signed module of the vector (to allow negative values for the vector diagram result view). This component is optional; if not specified, GiD calculates the module of the entered vector. If it is defined, however, `result_z[i]` should be defined too.

- **Matrix:** Each line is composed of 1 integer plus 6 real numbers:

```
i result_Sxx[i] result_Syy[i] result_Szz[i] result_Sxy[i]
result_Syz[i] result_Sxz[i]
```

where:

- `i` = node or Gauss point number.

- `result_Sxx[i]` = value of the `xx`_component of the result on the node or Gauss point number `i`.
 - `result_Syy[i]` = value of the `yy`_component of the result on the node or Gauss point number `i`.
 - `result_Szz[i]` = value of the `zz`_component of the result on the node or Gauss point number `i`. Optional if a 2D result type is specified that is not a plane deformation matrix.
 - `result_Sxy[i]` = value of the `xy`_component of the result on the node or Gauss point number `i`.
 - `result_Syz[i]` = value of the `yz`_component of the result on the node or Gauss point number `i`. Optional if a 2D result type is specified.
 - `result_Sxz[i]` = value of the `xz`_component of the result on the node or Gauss point number `i`. Optional if a 2D result type is specified.
- **Main Stresses:** Another way to give Stresses to GiD is by entering modules and vectors of these main stresses, so each line is composed of 1 integer plus 12 real numbers:

```
i result_Si[i] result_Sii[i] result_Siii[i] result_Vi_x[i]
result_Vi_y[i] result_Vi_z[i] result_Vii_x[i] result_Vii_y[i]
result_Vii_z[i] result_Viii_x[i] result_Viii_y[i]
result_Viii_z[i]
```

where:

- `i` = node or Gauss point number.
- `result_Si[i]` = value of the `Si`_module of the result on the node or Gauss point number `i`.
- `result_Sii[i]` = value of the `Sii`_module of the result on the node or Gauss point number `i`.
- `result_Siii[i]` = value of the `Siii`_module of the result on the node or Gauss point number `i`. Optional if a 2D result type is specified.
- `result_Vi_x[i]` = value of the `X`_component of the vector `Si` on the node or Gauss point number `i`.
- `result_Vi_y[i]` = value of the `Y`_component of the vector `Si` on the node or Gauss point number `i`.
- `result_Vi_z[i]` = value of the `Z`_component of the vector `Si` on the node or Gauss point number `i`. Optional if a 2D result type is specified.
- `result_Vii_x[i]` = value of the `X`_component of the vector `Sii` on the node or Gauss point number `i`.

- `result_Vii_y[i]` = value of the Y_component of the vector Sii on the node or Gauss point number `i`.
 - `result_Vii_z[i]` = value of the Z_component of the vector Sii on the node or Gauss point number `i`. Optional if a 2D result type is specified.
 - `result_Viii_x[i]` = value of the X_component of the vector Siii on the node or Gauss point number `i`.
 - `result_Viii_y[i]` = value of the Y_component of the vector Siii on the node or Gauss point number `i`.
 - `result_Viii_z[i]` = value of the Z_component of the vector Siii on the node or Gauss point number `i`. Optional if a 2D result type is specified.
- **Local Axes:** Local Axes are entered using the Euler angles that define them, so each line is composed of 1 integer plus 3 real numbers:

```
i euler_ang_1[i] euler_ang_2[i] euler_ang_3[i]
```

where:

- `i` = node or Gauss point number.
- `euler_ang_1[i]` = value of the 1st angle of Euler of the local axis on the node or Gauss point number `i`.
- `euler_ang_2[i]` = value of the 2nd angle of Euler of the local axis on the node or Gauss point number `i`.
- `euler_ang_3[i]` = value of the 3rd angle of Euler of the local axis on the node or Gauss point number `i`.

Note: For `Matrix` and `PlainDeformationMatrix` results, the `Si`, `Sii` and `Siii` components are calculated by GiD, which represents the eigen values and vectors of the matrix results, and which are ordered according to the eigen value.

Results on GaussPoints: When defining results on Gauss Points using the new Gauss points format, i.e. giving a "`gauss_points_name`" to the Result's Header description, the results should be given on a per element basis specifying the element number only once.

For example, assuming a three gauss point set named "`GaussTriang`" has been defined over triangles, and there are only two triangles, then a supposed '`Displacement`' result will look like this:

```
GaussDISPLAC.    2    1    2    2    0 "GaussTriang"
      5      0.1  -0.1   0.5
              0.0   0.0   0.8
```

```

        0.04 -0.04  1.0
6      0.0   0.0   0.8
      -0.1  -0.1   0.5
      -0.04 -0.04  1.0

```

Gauss Points (Old format)

Note: Here is a description of the **old Gauss Points file format** for the old results file format. However, the **new Gauss Points file format** (see [Gauss Points](#)) is also compatible with the old results format.

Gauss Points: For the Gauss points to be included in the results, they must be treated as if they were a type of result, but:

- they must be inserted at the beginning of the file; and
- the header structure is the same as that of the results files, but the meaning changes.

Note: GiD can only support Gauss Points on Lines, Triangles and Quadrilaterals, as well as one Gauss point for Tetrahedra and Hexahedra, at the same time.

Set 1: Header. Gauss points

The total number of lines in this set is also 1, but now it is always composed of one character string, one integer, one real number plus three integers:

```
descr_menu load_type step_val data_type data_loc desc_comp
```

where:

- `descr_menu` will not be used.
- `load_type` = 0, to indicate that they are Gauss points.
- `step_val` = number of Gauss points per element:
 - 1, 3, 6 for Triangles;
 - 1, 4, 9 for quadrilaterals;
 - 1, 4, 10 for Tetrahedra;
 - 1, 8, 27 for hexahedra; and
 - 1, ... points equally spaced over lines.

Note: This must be constant for the whole geometry.

Note: Tetrahedra with 4 or 10 Gauss Points and Hexahedra with 8 or 27 Gauss Points are not functional and are still under development.

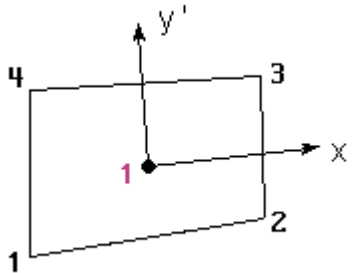
- `data_type` = this field indicates whether the Natural coordinates for the Gauss points are the ones described below this header or are the ones defined inside GiD.

- 0 - the Natural Coordinates for the Gauss points will be the ones which are described below. For Triangles and Tetrahedra they should be between 0.0 and 1.0, and for Quadrilaterals and Hexahedra should be between -1.0 and 1.0. For instance, the Natural Coordinates of three Gauss Points on Triangles will be:

`Coords_P_Gauss 0 3 0 0 0 1 0.5 0.0 2 0.5 0.5 3 0.0 0.5`

These are also the ones that GiD uses internally to calculate Gauss Points for Triangles with three Gauss Points, when this field is set to 1.

- 1 - the program must calculate the Gauss Points and they will be these ones:



Gauss Points positions of the quadrature of Gauss-Legendre for Triangles and Quadrilaterals

This field has no relevance for lines, and should be set to 1.

- `data_loc` = this option indicates whether or not the nodes are included inside the number of points over lines.
 - 1 - nodes are not included in the points count for lines, so points are placed at a distance from the nodes $i / (n_points + 1)$ with $i = 1..n_points$ and $n_points \geq 1$.
 - 2 - nodes are included in the points count for lines, so points are placed at a distance from the nodes $(i - 1) / (n_points - 1)$ with $i = 1..n_points$ and $n_points \geq 2$.

This field has no relevance for triangles, quadrilaterals, tetrahedra and hexahedra.

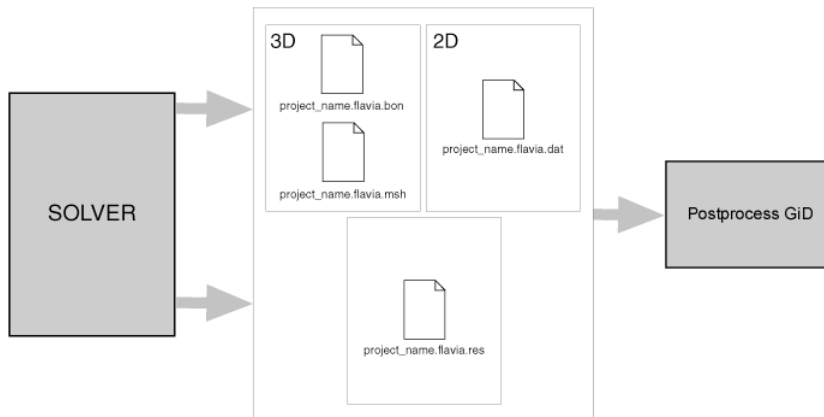
- `desc_comp` does not matter, but it must be specified.

Old postprocess mesh format

The old postprocess mesh format is still compatible with this version of GiD. The files containing the postprocess mesh (in the old file format) can be separated into two categories:

- **3D Data Files:** `ProjectName.post.msh`, or the old `ProjectName.flavia.msh`, for volume mesh information and `ProjectName.post.bon`, or the old `ProjectName.flavia.bon`, for surface mesh information; and
- **2D Data Files:** `ProjectName.post.dat`, or the old `ProjectName.flavia.dat`, for 2D mesh information.

Postprocessing data files are ASCII files and must be in a specific format, which is explained below. Each mesh information file can only handle one type of element.



- **ProjectName.flavia.msh:** The first file, which is named `ProjectName.flavia.msh`, should contain the information relating to the 3D volume mesh. It contains the nodal coordinates of the 3D mesh, its nodal connectivities and the material of each element. The nodal coordinates must include those on the surface mesh. If no material is supplied, GiD takes the material number to be equal to zero.
- **ProjectName.flavia.bon:** The second file, which is named `ProjectName.flavia.bon`, should contain the information about 3D surface sets. It can be used to represent boundary conditions of the volumetric mesh and additional surfaces (for instance, sheets, beams and shells). At the very least, all the mesh points supplied in `ProjectName.flavia.msh` should be present in `ProjectName.flavia.bon` at the beginning of the file.
- **ProjectName.flavia.dat:** This file contains information about 2D meshes and can only be used if neither of the two above files is used. It should specify the nodal coordinates of a mesh, its connectivities (elements) and, if desired, its material number (if not

specified, GiD takes this to be 0). The files are created and read in the order that corresponds to the natural way of solving a finite element problem: mesh, surface definition and conditions and finally, evaluation of the nodal results. The format of the read statements is normally free, i.e. it is only necessary to separate them by spaces. Thus, you can modify the files with any format, leaving spaces between each field and can also write out the results with as many decimal places as desired. If there is an error, the program warns you about the type of mistake found. Whenever possible, GiD reads all the information directly from the preprocessing files in order to gain efficiency.

Old file format: **ProjectName.flavia.msh**

Set 1: Header

This set contains six lines which are included so that information about the project can be included. They can be left blank, but it is suggested to use them for the project name and current version, as well as any extra comments, e.g. the type of project, the kinds of equations used, the conditions and materials involved, etc.

Note: As the seventh line of the file (i.e. Set 2) contains a series of numbers, it is advisable to use the sixth line to explain what these figures represent.

Set 2: General mesh data

The total number of lines in this set is 1, composed of 3 integers and an optional 4th integer:

```
n_3D_mesh_elements n_3D_mesh_points n_element_type [ last_node]
```

where:

- `n_3D_mesh_elements` = number of mesh elements,
- `n_3D_mesh_points` = number of mesh points,
- `n_element_type` = type of elements,
- `last_node` = number of the last node and required if nodes are not between 1 and `n_3D_mesh_points`.

The third parameter is used by the program to recognize what kind of finite element is being used. To do this GiD considers the following finite element types:

- number 1 corresponds to a hexahedron with eight nodes,
- number 3 corresponds to a tetrahedron with four nodes.

Set 3: Free line for any use

This is a free line, which can be used to write anything, though most modules inside GiD write the word `'Coordinates'` here to indicate the meaning of the following lines.

Set 4: Coordinates

The total number of lines in this set is `n_3D_mesh_points`, one for each nodal point, composed of 1 integer plus 3 real numbers:

```
i x_coord[i] y_coord[i] z_coord[i]
```

where:

- `i` = node number.
- `x_coord[i]` = x_coordinate of the node number `i`.
- `y_coord[i]` = y_coordinate of the node number `i`.
- `z_coord[i]` = z_coordinate of the node number `i`.

All the points of the meshes in the domain have to appear in this file.

Set 5: Free line for any use

This is a free line, which can be used to write anything, though most modules inside GiD write the word 'Connectives' here to indicate the meaning of the following lines.

Set 6: Connectivities

The total number of lines in this set is `n_3D_mesh_elements`, composed of 1 integer plus `n_nodes/element` integers and 1 further optional integer:

```
j node[j][1] node[j][2] ... node[j][n_nodes/element] mat[j]
```

where:

- `j` = element number.
- `node[j][1]` = node number 1 for the element number `j`.
- `node[j][2]` = node number 2 for the element number `j`...
- `node[j][n_nodes/element]` = last node number for the element number `j`.
- `mat[j]` = material index of the element number `j`.

The nodal connections must follow some specifications, so, for each tetrahedral element with four nodes, the rule is that the first three nodes that form a triangular face must be sorted so as to to define a normal which points towards the semi space containing the fourth node.

The vector `mat[j]` holds the material index of the element number `j`.

Old file format: ProjectName.flavia.bon

Set 1: Header

This set contains six lines which are included so that information about the project can be included. They can be left blank, but it is suggested to use them for the project name and current version, as well as any extra comments, e.g. the type of project, the kinds of equations used, the conditions and materials involved, etc.

Note: As the seventh line of the file (i.e. Set 2) contains a series of numbers, it is advisable to use the sixth line to explain what these figures represent.

Set 2: General boundary data

The total number of lines in this set is 1, composed of 3 integers and an optional 4th integer:

```
n_bound_elements n_bound_points n_element_type  [ last_node]
```

where:

- `n_bound_elements` = number of boundary elements,
- `n_bound_points` = number of boundary points,
- `n_element_type` = type of elements,
- `last_node` = number of the last node and required if nodes are not between 1 and `n_bound_points`.

For the third parameter, GiD considers the following finite element types:

- number 7 corresponds to a triangle with three nodes,
- number 9 corresponds to a quadrilateral with four nodes,
- number 11 corresponds to a line with two nodes.

Set 3: Free line for any use

This is a free line, which can be used to write anything, though most modules inside GiD write the word 'Coordinates' here to indicate the meaning of the following lines.

Set 4: Coordinates

The total number of lines in this set is `n_bound_points`, one for each nodal point, composed of 1 integer plus 3 real numbers:

```
i x_coord[i] y_coord[i] z_coord[i]
```

where:

- `i` = node number,

- `x_coord[i]` = `x_coordinate` of the node number `i`,
- `y_coord[i]` = `y_coordinate` of the node number `i`,
- `z_coord[i]` = `z_coordinate` of the node number `i`.

All the points of the domain have to appear in this file, which includes all the mesh points introduced in `ProjectName.flavia.msh` at the beginning. Once all the volumetric mesh has been entered, it is possible to add surfaces that belong to a boundary of the domain but which do not belong to a volumetric mesh and for this reason only appear in `ProjectName.flavia.bon`, and not in `ProjectName.flavia.msh`.

Set 5: Free line for any use

This is a free line, which can be used to write anything, though most modules inside GiD write the word 'Connectivities' here to indicate the meaning of the following lines.

Set 6: Connectivities

The total number of lines in this set is `n_bound_elements`, composed of 1 integer plus `n_nodes/element` integers and 2 optional integers more:

```
j node[j][1] node[j][2] ... node[j][n_nodes/element] set[j]
```

where:

- `j` = element number.
- `node[j][1]` = node number 1 for the element number `j`.
- `node[j][2]` = node number 2 for the element number `j`...
- `node[j][n_nodes/element]` = last node number for the element number `j`.
- `set[j]` = number of set to which the element number `j` belongs.

The vector `set[j]` lets you distinguish between groups of elements in different sets. It applies, for instance, in the case of defining the different conditions that the element fulfills.

Old file format: `ProjectName.flavia.dat`

Set 1: Header

This set contains six lines which are included so that information about the project can be included. They can be left blank, but it is suggested to use them for the project name and current version, as well as any extra comments, e.g. the type of project, the kinds of equations used, the conditions and materials involved, etc.

Note: As the seventh line of the file (i.e. Set 2) contains a series of numbers, it is advisable to use the sixth line to explain what these figures represent.

Set 2: General mesh data

The total number of lines in this set is 1, composed of 3 integers and an optional 4th integer:

```
n_2D_mesh_elements n_2D_mesh_points n_element_type [ last_node]
```

where:

- `n_2D_mesh_elements` = number of 2D mesh elements,
- `n_2D_mesh_points` = number of 2D points,
- `n_element_type` = type of elements,
- `last_node` = number of the last node and required if nodes are not between 1 and `n_2D_mesh_points`.

The third parameter is used by the program to recognize what kind of finite element is being used. To do this GiD considers the number of nodes that the finite element type uses. So,

- number 2 corresponds to a line with two nodes,
- number 3 corresponds to a triangle with three nodes,
- number 4 corresponds to a quadrilateral with four nodes,
- number 6 corresponds to a triangle with six nodes,
- number 8 corresponds to a quadrilateral with eight nodes,
- number 9 corresponds to a quadrilateral with nine nodes.

Set 3: Free line for any use

This is a free line, which can be used to write anything, though most modules inside GiD write the word '`Coordinates`' here to indicate the meaning of the following lines.

Set 4: Coordinates

The total number of lines in this set is `n_2D_mesh_points`, one for each nodal point, composed of 1 integer plus 3 real numbers:

```
i x_coord[i] y_coord[i]
```

where:

- `i` = node number.
- `x_coord[i]` = x_coordinate of the node number `i`.
- `y_coord[i]` = y_coordinate of the node number `i`.

Set 5: Free line for any use

This is a free line, which can be used to write anything, though most modules inside GiD write the word 'Connectivities' here to indicate the meaning of the following lines.

Set 6: Connectivities

The total number of lines in this set is `n_2D_mesh_elements`, composed of 1 integer plus `n_nodes/element` integers and a further 2 optional integers:

```
j node[j][1] node[j][2] ... node[j][n_nodes/element] set[j]
```

where:

- `j` = element number.
- `node[j][1]` = node number 1 for the element number `j`.
- `node[j][2]` = node number 2 for the element number `j`...
- `node[j][n_nodes/element]` = last node number for the element number `j`.
- `set[j]` = number of set to which the element number `j` belongs.

The vector `set[j]` lets you distinguish between groups of elements in different sets. It applies, for instance, in the case of defining the different conditions that the element fulfills.

Note: The numeration of quadratic elements is linear and not hierarchical, i.e. nodes should be specified counterclockwise, without skipping internal nodes.

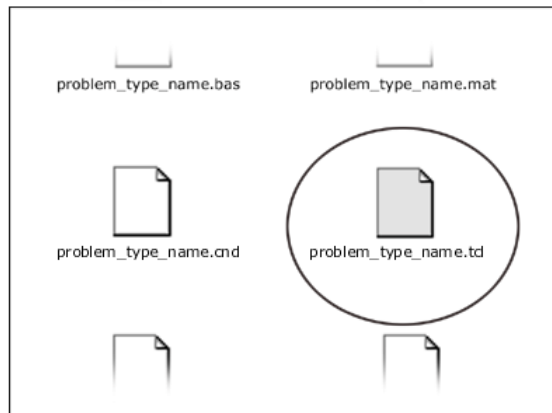
TCL/TK EXTENSION

This chapter looks at the advanced features of GiD in terms of expandability and total control. Using the Tcl/Tk extension you can create script files to automatize any process created with GiD. With this language new windows and functionalities can be added to the program.

For more information about the Tcl/Tk programming language, visit <http://www.scriptics.com>.

If you are going to use a Tcl file, it must be located in the Problem Type directory and be called `problem_type_name.tcl`.

Problem Type directory



Event procedures

The structure of `problem_type_name.tcl` can optionally implement some of these Tcl prototype procedures (and other user-defined procedures). The procedures listed below are automatically called by GiD. Their syntax corresponds to standard Tcl/Tk language:

```
proc InitGIDProject { dir } {  
    ...body...  
}  
proc InitGIDPostProcess {} {  
    ...body...
```



```
}
proc EndGIDProject {} {
    ...body...
}
proc EndGidPostprocess {} {
    ...body...
}
proc AfterOpenFile { filename format error } {
    ...body...
}
proc LoadGIDProject { filesdpd } {
    ...body...
}
proc SaveGIDProject { filesdpd } {
    ...body...
}
proc LoadResultsGIDPostProcess { file } {
    ...body...
}
proc BeforeMeshGeneration { elementsizesize } {
    ...body...
}
proc AfterMeshGeneration { fail } {
    ...body...
}
proc SelectGIDBatFile { dir basename } {
    ...body...
    set value ...
    return $value
}
proc BeforeRunCalculation { batfilename basename dir problemtypedir
gidexe args } {
    ...body...
}
proc AfterRunCalculation { basename dir problemtypedir where error
errorfilename } {
    ...body...
}
proc ChangedLanguage { language } {
```

```

    ...body...
}
proc BeforeWriteCalcFileGIDProject { file } {
    ...body...
    set value ...
    return $value
}
proc AfterWriteCalcFileGIDProject { file error } {
    ...body...
    set value ...
    return $value
}
proc AfterTransformProblemType { file oldproblemtypetype newproblemtypetype }
{
    ...body...
}

proc LoadFileInGidUnknowExtension { filename } {
    ...body...
}

```

- **InitGIDProject:** will be called when the problem type is selected. It receives the dir argument, which is the absolute path to the `problem_type_name.gid` directory, which can be useful inside the routine to locate some alternative files.
- **InitGIDPostProcess:** will be called when postprocessing starts. It has no arguments.
- **EndGIDProject:** will be called when this project is about to be closed. It has no arguments.
- **EndGIDPostProcess:** will be called when you leave Postprocess and open Preprocess. It has no arguments.
- **AfterOpenFile:** will be called after a geometry or mesh file is read inside GiD. It receives as arguments:
 - `filename`: the full name of the file that has been read;
 - `format`: ACIS_FORMAT, CGNS_FORMAT, DXF_FORMAT, GID_BATCH_FORMAT, GID_GEOMETRY_FORMAT, GID_MESH_FORMAT, IGES_FORMAT, NASTRAN_FORMAT, PARASOLID_FORMAT, RHINO_FORMAT, SHAPEFILE_FORMAT, STL_FORMAT, VDA_FORMAT, VRML_FORMAT or 3DSTUDIO_FORMAT;
 - `error`: boolean 0 or 1 to indicate an error when reading.
- **LoadGIDProject:** will be called when a GiD project or problem type is loaded. It receives the argument `filesdpd`, which is the path of the file which is being opened, but

with the extension `.spd` (**s**pecific **p**roblemtype **d**ata). This path can be useful if you want to write specific information about the problem type in a new file.

- **SaveGIDProject**: will be called when the currently open file is saved to disk. It receives the argument `filesdpd`, which is the path of the file being saved, but with the extension `.spd` (**s**pecific **p**roblemtype **d**ata). This path can be useful if you want to write specific information about the problem type in a new file.
- **LoadResultsGIDPostProcess**: will be called when a results file is opened in GiD Postprocess. It receives one argument, the name of the file being opened without its extension.
- **BeforeMeshGeneration**: will be called before the mesh generation. It receives the mesh size desired by the user as the `elementsiz` argument. This event can typically be used to assign some condition automatically.
- **AfterMeshGeneration**: will be called after the mesh generation. It receives as its fail argument a true value if the mesh is not created.
- **SelectGIDBatFile**: must be used to switch the default batch file for special cases. This procedure must return as a value the alternative pathname of the batch file. For example it is used as a trick to select a different analysis from a list of batch calculation files.
- **BeforeRunCalculation**: will be called before running the analysis. It receives several arguments:
 - `batfilename`: the name of the batch file to be run (see [Executing an external program](#));
 - `basename`: the short name model;
 - `dir`: the full path to the model directory;
 - `problemtypedir`: the full path to the Problem Types directory;
 - `gidexe`: the full path to `gid`;
 - `args`: an optional list with other arguments.
- **AfterRunCalculation**: will be called just after the analysis finishes. It receives as arguments:
 - `basename`: the short name model;
 - `dir`: the full path to the model directory;
 - `problemtypedir`: the full path to the Problem Types directory;
 - `where`: must be local or remote (remote if it was run in a server machine, using `ProcServer`);
 - `error`: returns 1 if an calculation error was detected;
 - `errorfilename`: an error filename with some error explanation, or nothing if everything was OK.
- **ChangedLanguage**: will be called when you change the current language. The argument is the new language (`en`, `es`, ...). It is used, for example, to update problem type customized menus, etc.

- **BeforeWriteCalcFileGIDProject:** will be called just before writing the calculation file. It is useful for validating some parameters. If it returns `-cancel-` as a value then nothing will be written.
 - `file`: the name of the output calculation file.
- **AfterWriteCalcFileGIDProject:** will be called just after writing the calculation file and before the calculation process. It is useful for renaming files, or cancelling the analysis. If it returns `-cancel-` as a value then the calculation is not invoked.
 - `file`: the name of the output calculation file
 - `error`: an error code if there is some problem writing the output calculation file.
- **AfterTransformProblemType:** will be called just after transforming a model from a problem type to a new problem type version:
 - `file`: the name of the model to be transformed;
 - `oldproblemtyp`: the name of the previous problem type;
 - `newproblemtyp`: the name of the problem type to be transformed.
- **LoadFileInGidUnknowExtension:** will be called when you drop a file with an unknown extension, then the problem type can try to read it.
 - `filename`: the name of dropped file.

Note: To use Tcl to improve the capabilities of writing the calculations file, it is possible to use the command `*tcl` in the template file (`.bas` file); see [Specific commands](#) for details.

Control functions

GiD offers the following Tcl functions:

- **GiD_Process** *command_1 command_2 ...*: used to execute GiD commands;
- **GiD_Info** *option*: used to obtain information about the current GiD project.

Process function

GiD_Process *command_1 command_2 ...*

This is a simple function, but a very powerful one. It is used to enter commands directly inside the central event manager. The commands have the same form as those typed in the command line within GiD.

You have to enter exactly the same sequence as you would do interactively, including the escape sequences (using the word `escape`) and selecting the menus and operations used.

You can obtain the exact commands that GiD needs by checking the **Right buttons** menu (Utilities→Tools→Toolbars). It is also possible to save a batch file (Utilities→Preferences) where the commands used during the GiD session can be checked.

Here is a simple example to create one line:

```
GiD_Process Mescape Geometry Create Line 0,0,0 10,0,0 escape
```

Note: Mescape is a multiple 'escape' command, to go to the top of the commands tree.

Info function

GiD_Info *option*

This function provides any information about GiD, the current data or the state of any task inside the application. Depending on the arguments introduced after the `GiD_Info` command, GiD will output different information:

- **GiD_Info materials**

This command returns a list of the materials in the project.

These options are also available:

- *["material_name"]*: If a material name is given, its properties are returned. It is also possible to add the option *[OTHERFIELDS]* to get the fields of that material, or the option *[BOOK]* to get the book of that material.
- *[BOOKS]*: If this option is given, a list of the material books in the project is returned.

Examples:

```
in: GiD_Info materials
```

```
out: "Air Steel Aluminium Concrete Water Sand"
```

```
in: GiD_Info materials Steel
```

```
out: "1 Density 7850"
```

- **GiD_Info conditions ovpnt | ovline | ovsurf | ovvol**

This command returns a list of the conditions in the project. One of the arguments **ovpnt**, **ovline**, **ovsurf**, **ovvol** must be given to indicate the type of condition required, respectively, conditions over points, lines, surfaces or volumes.

Instead of **ovpnt**, **ovline**, **ovsurf**, **ovvol**, the following options are also available:

- *[-interval "intv"] [-localaxes | -localaxesmat | -localaxescenter | -localaxesmatcenter] "condition_name" [geometry | mesh]:* if a condition name is given, the command returns the properties of that condition. It is also possible to add the options *geometry* or *mesh*, and all geometry or mesh entities that have this condition assigned will be returned. If *-interval "intv"* is set, then the conditions on this interval ("intv"=1,...n) are returned instead of those on the current interval. If *-localaxes* is set, then the three numbers that are the three Euler angles that define a local axes system are also returned (only for conditions with local axes, see [Local axes](#)). Selecting *-localaxesmat*, the nine numbers that define the transformation matrix of a vector from the local axes system to the global one are returned. If *-localaxescenter* is set, then the three Euler angles and the local axis center are also returned. Selecting *-localaxesmatcenter* returns the nine matrix numbers and the center. Adding the number id of an entity (*["entity_id"]*) after the options *mesh* or *geometry*, the command returns the value of the condition assigned to that entity. Other options available if the condition name is given are *[OTHERFIELD]*, to get the fields of that condition, and *[BOOK]*, to get the book of the condition.
- *[BOOKS]:* If this option is given, a list of the condition books of the project is returned.

Examples:

```
in: GiD_Info conditions ovpt
out: "Point-Weight Point-Load"
```

```
in: GiD_Info conditions Point-Weight
out: "ovpt 1 Weight 0"
```

```
in: GiD_Info conditions Point-Weight geometry
out: "E 1 - 2334 , E 2 - 2334 , E 3 - 343"
```

```
in: GiD_Info Conditions -localaxes Concrete_rec_section mesh 2
out: {E 2 - {4.7123889803846897 1.5707963267948966 0.0} N-m 0.3
0.3 HA-25}
```

- **GiD_Info layers**

This command returns a list of the layers in the project. These options are also available:

- *["layer_name"]:* If a layer name is given, the command returns the properties of that layer.
- *[-on]:* Returns a list of the visible layers.

- *[-off]*: Returns a list of the hidden layers.
- *[-hasbacklayers]*: Returns 1 if the project has entities inside back layers.
GiD_Infoback_layers returns a list with the back layers
Example:
in: GiD_Info back_layers
out: Layer2_*back*
- *[-bbox[-use geometry|mesh]]: layer_name_1 layer_name_2 ...]*: Returns two coordinates (x1,y1,z1,x2,y2,z2) which define the bounding box of the entities that belong to the list of layers. If the option *[-use geometry|mesh]* is used, the command returns the bounding box of the **geometry** or the bounding box of the **mesh**. If the list of layers is empty, the maximum bounding box is returned.
- *[-entities]*: One of the following arguments must be given:
nodes,elements,points,lines,surfaces or *volumes*. A layer name must also be given. The command will return the nodes, elements, points, lines surfaces or volumes of that layer.

Examples:

```
in: GiD_Info layers
out: "layer1 layer2 layer_aux"
```

```
in: GiD_Info layers -on
out: "layer1 layer2"
```

```
in: GiD_Info layers -entities lines layer2
out: "6 7 8 9"
```

- **GiD_Info gendata**

This command returns the information entered in the **Problem Data** window (see [Problem and intervals data file \(.prb\)](#)).

The following options are available:

- *[OTHERFIELDS]*: It is possible to add this option to get the additional fields from the Problem Data window.
- *[BOOKS]*: If this option is given, a list of the Problem Data books in the project is returned. Example: in: GiD_Info gendata out: "2
Unit_System#CB#(SI,CGS,User) SI Title M_title"

- **GiD_Info intvdata**

This command returns a list of the interval data in the project (see [Problem and intervals data file \(.prb\)](#)). The following options are available:

- *-interval <number>*: To get data from an interval different from the number 0 (default).
- *[OTHERFIELDS]*: It is possible to add this option to get the additional fields from the Interval Data window.
- *[BOOKS]*: If this option is given, a list of the Interval Data books in the project is returned.
- *[NUM]*: If this option is given, a list of two natural numbers is returned. The first element of the list is the current interval and the second element is the total number of intervals.

- **GiD_Info Project <item>?**

This command returns information about the project. More precisely, it returns a list with:

- Problem type name.
- Current model name.
- 'There are changes' flag.
- Current layer to use.
- Active part (GEOMETRYUSE, MESHUSE, POSTUSE or GRAFUSE).
- Quadratic problem flag.
- Drawing type (normal, polygons, render, postprocess).
- NOPOST or YESPOST.
- Debug or nodebug.
- GiD temporary directory.
- Must regenerate the mesh flag (0 or 1).
- Last element size used for meshing (NONE if there is no mesh).
- BackgroundFilename is the name of a background mesh file to assign mesh sizes. It is possible to ask for a single item only rather than the whole list, with <item> equal to:

ProblemType|ModelName|AreChanges|LayerToUse|ViewMode|Quadratic|RenderMode|ExistPost|Debug|TmpDirectory|MustReMesh|LastElementSize|BackgroundFilename

Examples:

in: GiD_Info Project

out: "cmas2d e:\models\car_model 1 layer3 MESHUSE 0 normal YESPOST
nodebug C:\TEMP\gid2 0 1.4"

in: GiD_Info Project ModelName

out: "e:\models\car_model"

- **GiD_Info Geometry**

This command gives the user information about the geometry in the project. For each entity, there are two possibilities:

- *[NumPoints]*: Returns the total number of points in the model.
- *[NumLines]*: Returns the total number of lines.
- *[NumSurfaces]*: Returns the total number of surfaces.
- *[NumVolumes]*: Returns the total number of volumes.
- *[NumDimensions]*: Returns the total number of dimensions.
- *[MaxNumPoints]*: Returns the maximum point number used (can be higher than NumPoints).
- *[MaxNumLines]*: Returns the maximum line number used.
- *[MaxNumSurfaces]*: Returns the maximum surface number used.
- *[MaxNumVolumes]*: Returns the maximum volume number used.
- *[MaxNumDimensions]*: Returns the maximum dimension number used.

- **GiD_Info Mesh**

This command gives the user information about the selected mesh in the project. It returns a 1 followed by a list with all types of element used in the mesh.

- *[NumElements [Elemtype] [nnode]]*: returns the number of elements of the mesh. *Elemtype* can be: *Linear/Triangle/Quadrilateral/Tetrahedra/Hexahedra/Prism/Any*. *nnode* is the number of nodes of an element.
- *[NumNodes]*: Returns the total number of nodes of the mesh.
- *[MaxNumElements]*: Returns the maximum element number.
- *[MaxNumNodes]*: Returns the maximum node number.
- *[Elements Elemtype[First_idLast_id]]*: Returns a list with the element number, the connectivities and the material number, from 'first_id' to 'last_id', if they are specified. **Elemtype** can be: **Linear/Triangle/Quadrilateral/Tetrahedra/Hexahedra/Prism/Any**.
- *[Nodes[First_idLast_id]]*: Returns a list with the node number and x y z coordinates, from 'first_id' to 'last_id', if they are specified.
- *[-sublist]* : Returns each result item as a Tcl list (enclosed in braces)

Examples:

in: GiD_Info Mesh

out: "1 Tetrahedra Triangle"

in: GiD_Info Mesh MaxNumNodes

out: "1623"

- **GiD_Info Coordinates point_id/node_id [geometry|mesh]**
This command returns the coordinates (x,y,z) of a given point or node.
- **GiD_Info variables "variable_name"**
This command returns the value of the variable indicated. GiD variables can be found in the **Right buttons** menu (see [Tools](#)), under the option Utilities -> Variables.
- **GiD_Info localaxes ?<name>? ?-localaxesmat?info localaxes**
Returns a list with all the user defined local axes.
info localaxes <name> returns the parameters (three euler angles and the center) that define the local axes called <name>.
info localaxes <name> -localaxesmat instead of returning the three euler angles, it returns the nine numbers that define the transformation matrix of a vector from the local axes system to the global one.
- **GiD_Info ortholimits**
This command returns a list of the limits of the geometry in the project.
- **GiD_Info perspectivefactor**
This command returns which perspective factor is currently being used in the project.
- **GiD_Info graphcenter**
This command returns the coordinates (x,y,z) of the center of rotation.
- **GiD_Info MeshQuality**
This command returns a list of numbers. These numbers are the **Y** relative values of the graph shown in the option **Meshing**→**Mesh quality** (see [Mesh quality](#)).

This command has the following arguments:

- *MinAngle/MaxAngle/ElemSize/ElemMinEdge/ElemMaxEdge/ElemShapeQuality*: quality criterion.
 - *Linear/Triangle/Tetrahedra/QuadrilateralHexahedra/Prism*: type of element.
 - *"min_degrees"*: minimum number of degrees accepted.
 - *"max_degrees"*: maximum number of degrees accepted.
 - *"num_divisions"*: number of divisions. Example: in: GiD_Info MeshQuality MinAngle Triangle 20 60 4 out: "13 34 23 0"
-
- **GiD_Info postprocess arewein**
This command returns YES if the user is in the GiD postprocess, and NO, if not.
 - **GiD_Info postprocess get**
This command returns information about the GiD postprocess. The following options are available:
 - **all_volumes**: Returns a list of all volumes.
 - **all_surfaces**: Returns a list of all surfaces.
 - **all_cuts**: Returns a list of all cuts.

- **all_graphs**: Returns a list of all graphs.
- **all_volumes_colors**: Returns a list of the volume colors used in the project. Colors are represented in RGB hexadecimal format. Example: #000000 would be black, and #FFFFFF would be white.
- **all_surfaces_colors**: Returns a list of the surface colors used in the project. Colors are represented in RGB hexadecimal format. Example: #000000 would be black, and #FFFFFF would be white.
- **all_cuts_colors** : Returns a list of the cut colors used in the project. Colors are represented in RGB hexadecimal format. Example: #000000 would be black, and #FFFFFF would be white.
- **cur_volumes**: Returns a list of the visible volumes.
- **cur_surfaces**: Returns a list of the visible surfaces.
- **cur_cuts**: Returns a list of the visible cuts.
- **all_display_styles**: Returns a list of all types of display styles available.
- **cur_display_style**: Returns the current display style.
- **all_display_renders**: Returns a list of all types of rendering available.
- **cur_display_render**: Returns the current rendering method.
- **all_display_culling**: Returns a list of all types of culling available.
- **cur_display_culling**: Returns the current culling visualization.
- **cur_display_transparence**: Returns Opaque or Transparent depending on the current transparency. Transparency is chosen by the user in the **Select & Display Style** window.
- **cur_display_body_type**: Returns Massive if the option **Massive** is selected in the **Select & Display Style** window. It returns Hollow if that option is not activated.
- **all_analysis**: Returns a list of all analyses in the project.
- **all_steps "analysis_name"**: Returns the number of steps of "analysis_name".
- **cur_analysis**: Returns the name of the current analysis.
- **cur_step**: Returns the current step.
- **all_results_views**: Returns all available result views.
- **cur_results_view**: Returns the current result view.
- **cur_results_list**: This option has one more argument: the kind of result visualization must be given. The available kinds of result visualization are given by the option **all_results_views**. The command returns a list of all the results that can be represented with that visualization in the current step of the current analysis.
- **results_list**: This option has three arguments: the first argument is the kind of result visualization. The available kinds of result visualization are given by the option **all_results_views**. The second argument is the analysis name. The

third argument is the step number. The command returns a list of all the results that can be represented with that visualization in the given step.

- **cur_result**: Returns the current selected result. The kind of result is selected by the user in the **View results** window.
- **cur_components_list "result_name"**: Returns a list of all the components of the result "result_name".
- **cur_component**: Returns the current component of the current result.
- **main_geom_state**: Returns whether the main geometry is Deformed or Original
- **main_geom_all_deform**: Returns a list of all the deformation variables (vectors) of the main geometry.
- **main_geom_cur_deform**: Returns the current deformation variable (vectors) of the main geometry.
- **main_geom_cur_step**: Returns the main geometry current step.
- **main_geom_cur_anal**: Returns the main geometry current analysis.
- **main_geom_cur_factor**: Returns the main geometry current deformation factor.
- **show_geom_state**: Returns whether the reference geometry is Deformed or Original.
- **show_geom_cur_deform**: Returns the current deformation variable (vectors) of the reference geometry.
- **show_geom_cur_analysis**: Returns the reference geometry current analysis.
- **show_geom_cur_step**: Returns the reference geometry current step.
- **iso_all_display_styles**: Returns a list of all available display styles for isosurfaces.
- **iso_cur_display_style**: Returns the current display style for isosurfaces.
- **iso_all_display_renders**: Returns a list of all types of rendering available for isosurfaces.
- **iso_cur_display_render**: Returns the current rendering method for isosurfaces.
- **iso_cur_display_transparence**: Returns **Opaque** or **Transparent** depending on the current transparency of isosurfaces.
- **contour_limits**: Returns the minimum and maximum value of the contour limits. Before each value, the word STD appears if the contour limit value is the default value, and USER if it is defined by the user.
- **animationformat**: Returns the default animation format.
- **cur_show_conditions**: Returns the option selected in the **Conditions** combo box of the **Select & Display Style** window. (Possible values: Geometry Mesh None)

- **all_show_conditions**: Returns all the options available in the **Conditions** combo box of the **Select & Display Style** window. (Geometry Mesh None)
- **cur_contour_limits**: Returns the minimum and maximum value of the current value.
- **current_color_scale**: Returns a list of the colors used for the color scale; the first element of the list is the number of colors. Each color is represented in RGB hexadecimal format. Example: #000000 would be black, and #FFFFFF would be white.
- **GiD_Info AutomaticTolerance**
This command returns the value of the **Import Tolerance** used in the project. This value is defined in the **Preferences** window under **Import**.
- **GiD_Info RGBDefaultBackground**
This command returns the default background color in RGB. The format is three 8 bit numbers separated by #. Example: 255#255#255 would be white.
- **GiD_Info list_entities status**
This command returns a list with general information about the current GiD project.

Example:

in: `GiD_Info list_entities status`

out:

Project name: UNNAMED

Problem type: UNKNOWN

Changes to save(0/1): 1

Necessary to mesh again (0/1): 1

Using LAYER: NONE

Interval 1 of 1 intervals

Degree of elements is: Normal

Using now mode(geometry/mesh): geometry

number of points: 6

number of points with 2 higher entities: 6

number of points with 0 conditions: 6

number of lines: 6

number of lines with 1 higher entities: 6

number of lines with 0 conditions: 6

number of surfaces: 1

```
number of surfaces with 0 higher entities: 1
```

```
number of surfaces with 0 conditions: 1
```

```
number of volumes: 0
```

```
number of nodes: 8
```

```
number of nodes with 0 conditions: 8
```

```
number of Triangle elements: 6
```

```
number of elements with 0 conditions: 6
```

```
Total number of elements: 6
```

```
Last size used for meshing: 10
```

```
Internal information:
```

```
Total MeshingData:0 Active: 0 0%
```

- **GiD_Info list_entities**

This command returns information about entities.

It has the following arguments:

- **Points/Lines/Surfaces/Volumes/Nodes/Elements/Results:** Type of entity or Results. **Note:** If the user is postprocessing in GiD, the information returned by **Nodes/Elements** concerns the nodes and elements in postprocess, including its results information. To access preprocess information about the preprocess mesh, the following entity keywords should be used: **PreNodes/PreElements**.
- **entity_id:** The number of an entity. It is also possible to enter a list of entities (e.g.: 2 3 6 45), a range of entities (e.g.: entities from 3 to 45, would be 3:45) or a layer (e.g.: layer:layer_name).

Using "list_entities Results" you must also specify <analysis_name> <step> <result_name> <indexes> With the option **-more**, more information is returned about the entity. The **-more** option used with lines returns the length of the line, its radius (arcs), and the list of surfaces which are higher entities of that line; used with elements it returns the type of element, its number of nodes and its volume.

Example 1:

```
in: GiD_Info list_entities Points 2 1
```

```
out:
```

```
POINT
```

```
Num: 2 HigherEntity: 1 conditions: 0 material: 0
```

```
LAYER: car_lines
```

```

Coord: -11.767595 -2.403779 0.000000
END POINT
POINT
Num: 1 HigherEntity: 1 conditions: 0 material: 0
LAYER: car_lines
Coord: -13.514935 2.563781 0.000000
END POINT

```

Example 2:

```

in: GiD_Info list_entities lines layer:car_lines
out:
STLINE
Num: 1 HigherEntity: 0 conditions: 0 material: 0
LAYER: car_lines
Points: 1 2
END STLINE
STLINE
Num: 13 HigherEntity: 0 conditions: 0 material: 0
LAYER: car_lines
Points: 13 14
END STLINE

```

Example 3 (using -more):

```

in: GiD_Info list_entities -more Lines 2
out:
STLINE
Num: 2 HigherEntity: 2 conditions: 0 material: 0
LAYER: Layer0
Points: 2 3
END STLINE
LINE (more)
Length=3.1848 Radius=100000
Higher entities surfaces: 1 3
END LINE

```

- **GiD_Info parametric**

This command returns geometric information (coordinates, derivatives, etc.) about parametric lines or surfaces. For lines it has the following syntax:

```
GiD_Info parametric line entity_id
coord|deriv_t|deriv_tt|t_fromcoord t|x y z
```

And for surfaces:

```
GiD_Info parametric surface entity_id
coord|deriv_u|deriv_v|deriv_uu|deriv_vv|deriv_uv|normal|uv_fromc
oord u v|x y z
```

The result for each argument is:

- **line|surface**: Type of entity.
- **entity_id**: The number of an entity.
- **coord**: 3D coordinate of the point with parameter t (line) or u,v (surface).
- **deriv_t**: First curve derivative at parameter t.
- **deriv_tt**: Second curve derivative at parameter t.
- **t_fromcoord**: t parameter for a 3D point.
- **deriv_u,deriv_v**: First partial u or v surface derivatives.
- **deriv_uu,deriv_vv,deriv_uv**: Second surface partial derivatives.
- **normal**: Unitary surface normal at u,v parameters.
- **uv_fromcoord**: u,v space parameters for a 3D point.

Note: The vector derivatives are not normalized.

Example:

```
in: GiD_Info parametric line 26 deriv_t 0.25
out: 8.060864 -1.463980 0.000000
```

- **GiD_Info check**

This command returns some specialized entities check.

For lines it has the following syntax:

```
GiD_Info check line <entity_id> isclosed
```

For surfaces:

```
GiD_Info check surface <entity_id>
isclosed|isdegeneratedboundary|selfintersection
```


And for volumes:

```
GiD_Info check volume <entity_id> orientation|boundaryclose
```

The result for each argument is:

- **line|surface|volume**: Type of entity.
- **<entity_id>**: The number of an entity.
- **isclosed**: For lines: 1 if start point is equal to end point, 0 otherwise. For surfaces: A surface is closed if its coordinate curves (of the full underlying surface) with parameter 0 and 1 are equal. It returns a bit encoded combination of closed directions: 0 if it is not closed, 1 if it is closed in u, 2 if it is closed in v, 3 if it is closed in both u and v directions.
- **isdegeneratedboundary**: A surface is degenerated if some boundary in parameter space (south, east, north or west) becomes a point in 3d space. It returns a bit encoded combination of degenerated boundaries, for example: 0 if it is not degenerated, $9=2^0+2^3$ if south and west boundaries are degenerated.
- **selfintersection**: Intersections check between surface boundary lines. It returns a list of detected intersections. Each item contains the two line numbers and their parameter values.
- **orientation**: For volumes, it returns a two-item list. The first item is the number of bad oriented volume surfaces, and the second item is a list of these surfaces' numbers.
- **boundaryclose**: For volumes, a boundary is topologically closed if each line is shared by two volume surfaces. It returns 0 if it is not closed and must be corrected, or 1 if it is closed.

Example:

```
in: GiD_Info check volume 5 orientation
```

```
out: 2 {4 38}
```

- **GiD_Info ListMassProperties**

This command returns properties of the selected entities. It returns the **length** if entities are lines, **area** if surfaces, **volume** if volumes, or the **center of gravity** if entities are nodes or elements. It has the following arguments:

- **Points/Lines/Surfaces/Volumes/Nodes/Elements**: Type of entity.
- **entity_id**: The number of an entity. It is also possible to enter a list of entities (e.g.: 2 3 6 45) or a range of entities (e.g.: entities from 3 to 45, would be 3:45).

Example:

```

in: GiD_Info ListMassProperties Lines 13 15
out:
  LINES
  n. Length
  13 9.876855
  15 9.913899
  Selected 2 figures

```

```

Total Length=19.790754

```

- **GiD_Info problemtypepath**
This command returns the absolute path to the current problem type.
- **GiD_Info GiDVersion**
This command returns the GiD version number. For example 8.0

Special functions

Some special commands exist to control the redraw and wait state of GiD:

.central.s disable graphics 'value' The value 0/1 Enable/Disable Graphics (GiD does not redraw)

Example: to disable the redraw:

```
.central.s disable graphics 1
```

.central.s disable graphinput 'value' The value 0/1 Enable/Disable GraphInput (enable or disable peripherals: mouse, keyboard, ...)

Example: to disable the peripherals input:

```
.central.s disable graphinput 1
```

.central.s disable windows 'value' The value 0/1 Enable/Disable Windows (GiD displays, or not, windows which require interaction with the user)

Example: to disable the interaction windows:

```
.central.s disable windows 1
```

.central.s waitstate 'value' The value 0/1 Enable/Disable the Wait state (GiD displays a hourglass cursor in wait state)

Example: to set the state to wait:

```
.central.s waitstate 1
```

Usually these command are used jointly:

Example:

```
#deactivate redraws, etc wit a widget named $w
$w conf -cursor watch .central.s waitstate 1
update
.central.s disable graphics 1
.central.s disable windows 1
.central.s disable graphinput 1
...
#reactivate all and redraw
.central.s disable graphics 0
.central.s disable windows 0
.central.s disable graphinput 0
GiD_Redraw
$w conf -cursor ""
.central.s waitstate 0
```

Note: It is recommended for a Tcl developer to use the more 'user-friendly' procedures defined inside the file 'dev_kit.tcl' (located in the \scripts directory). For example, to disable and enable redraws, you can use:

```
::GidUtils::DisableGraphics
::GidUtils::EnableGraphics
```

Other GiD-Tcl special commands exist to manage materials, conditions, intervals or create nodes and elements directly, as follows:

GiD_CreateData create|delete material ?<basename>? <name> ?<values>?

To create or delete materials:

- **<basename>** this only applies to the **create** operation, and is the base material from which the new material is derived;
- **<name>** is the name of material itself;
- **<values>** is a list of all field values for the new material.

Example:

```
GiD_CreateData create material Steel Aluminium {3.5 4 0.2}
GiD_CreateData delete material Aluminium
```

GiD_AssignData material|condition <name> <over> ?<values>? <entities>

To assign materials or conditions over entities:

- **<name>** is the name of the material or condition;
- **<over>** must be: points, lines, surfaces, volumes, layers, nodes, elements, body_elements, or face_elements (elements is equivalent to body_elements);
- **<newvalues>** is only required for conditions. If it is set to "..." then the default values are used;
- **<entities>** a list of entities (it is valid to use ranges as a:b ; if <over> is face_elements then you must specify a list of "entity numface" instead just "entity").

Example:

```
GiD_AssignData materials Steel Surface {1 5}
GiD_AssignData condition Point-Load Nodes {3.5 2.1 8.0} {4 8}
GiD_AssignData condition Face-Load face_elements {3.5 2.1 8.0} {15 1
18 1 20 2}
```

GiD_ModifyData materials|intvdata|gendata ?<name>? <values>

To change all field values of materials, interval data or general data:

- **<name>** is the material name or interval number;
- **<values>** is a list of all the new field values for the material, interval data or general data.

Example:

```
GiD_ModifyData materials Steel {2.1e6 0.3 7800}
GiD_ModifyData intvdata 1 ...
GiD_ModifyData gendata ...
```

GiD_AccessValue set|get materials|conditions|intvdata|gendata ?<name>? <question> ?<attribute>? <value>

To change only some field values of materials, interval data or general data:

- **<name>** is the material, condition name or interval number (not necessary for gendata);
- **<question>** is a field name;
- **<attribute>** is the attribute name to be changed (STATE, HELP, etc.) instead of the field value;
- **<value>** is the new field or attribute value.

Example:

```
GiD_AccessValue set gendat Solver Direct
```

GiD_IntervalData <mode> <number>[?copyconditions?

To create, delete or set interval data;

- **<mode>** must be 'create', 'delete' or 'set';
- **<number>** is the interval number (integer >=1). **Create** returns the number of the newly created interval and can optionally use 'copyconditions' to copy to the new interval the conditions of the current one.

For **set** mode, if <number> is not supplied, the current interval number is returned.

Example:

```
set current [GiD_IntervalData set]
GiD_IntervalData set 2
set newnum [GiD_IntervalData create]
set newnum [GiD_IntervalData create copyconditions]
```

GiD_LocalAxes <mode> <name> ?<type>? <Cx Cy Cz> <PAxex PAxey PAxex> <PPlanex PPlaney PPlanez>?

To create, delete or modify local axes:

- **<mode>**: must be one of "create|delete|edit|exists", which correspond to the operations: create, delete, edit or exists;
- **<name>**: is the name of local axes to be created or deleted;
- **<type>**: must be one of "rectangular|cylindrical|spherical C_XZ_Z|C_XY_X". Currently, GiD only supports rectangular axes. **C_XZ_Z** is an optional word to specify one point over the XZ plane and another over the Z axis (default). **C_XY_X** is an optional word to specify one point over the XY plane and another over the X axis;
- **<Cx Cy Cz>** is a Tcl list with the real coordinates of the local axes origin;
- **<PAxex PAxey PAxex>** is a Tcl list with the coordinates of a point located over the Z' local axis (where Z' is positive). The coordinates must be separated by a space. If the z coordinate is missing, it is set to z=0.0;
- **<PPlanex PPlaney PPlanez>** is a Tcl list with the coordinates of a point located over the Z'X'half-plane (where X' is positive).

For the 'exists' operation, if only the <name> field is specified, 1 is returned if this name exists, and 0 if it does not. If the other values are also specified, <name> is ignored.

The value returned is:

- 1 if the global axes match;
- 2 if the automatic local axes match;
- 3 if the automatic alternative local axes match;
- 0 if it does not match with any axes;
- <n> if the user-defined number <n> (n>0) local axes match.

Example:

```
GiD_LocalAxes create "axes_1" rectangular C_XY_X {0 0 0} {0 1 0}
{1 0 0}
GiD_LocalAxes delete axes_1
GiD_LocalAxes exists axes_1
GiD_LocalAxes exists "" rectangular C_XY_X {0 0 0} {0 1 0} {1 0
0}
```

this last sample returns -1 (equivalent to global axis)

GiD_Geometry create|delete|get|list point|line|surface|volume <num>|append <data>

To create, delete, get data or list the identifiers of geometric entities:

- o **<num>|append:** <num> is the entity identifier (integer > 0). You can use the word 'append' to set a new number automatically.
- o **<data>:** is all the geometric definition data (**create**) or a selection specification (**delete, get or list**):

create: to make new geometric entities

- o GiD_Geometry create volume <num>|append layer numsurfaces {surface1 verso1} ...
- o GiD_Geometry create surface <num>|append nurbsurface layer numlines u_degree v_degree numpoints_u numpoints_v istrimmed isrational {line1 verso1} ... {point1_x point1_y point1_z ?point1_w?} ... knot_u_1 ... knot_v_1 ...
- o GiD_Geometry create line <num>|append nurbsline layer inipoint endpoint degree numpoints isrational {point1_x point1_y point1_z ?point1_w?} ... knot_1 ...
- o GiD_Geometry create line <num>|append stline layer inipoint endpoint
- o GiD_Geometry create point <num>|append layer point_x point_y point_z

delete: to erase model entities

- o GiD_Geometry delete point|line|surface|volume <args> with <args>: num numa:numb numa:layer:layer_name

get: to obtain all the geometrical data to define a single entity

- o GiD_Geometry get point|line|surface|volume <args> with <args>: num

list: to get a list of entity identifiers of a range or inside some layer

- o GiD_Geometry list point|line|surface|volume <args> with <args>: num
numa:numb numa: layer:layer_name

Examples:

```
GiD_Geometry create surface 1 nurbssurface Layer0 4 1 1 2 2 0 0
{1 1} {4 1} {3 1} {2 1} \
    {0.17799 6.860841 0.0} {-8.43042200 6.86084199 0.0}
    {0.17799400 0.938510 0.0} \
    {-8.43042 0.938510 0.0} 0.0 0.0 1.0 1.0 0.0 0.0 1.0 1.0
GiD_Geometry list points 1: layer:layer_name
```

GiD_Mesh create|delete|edit node|element <num>|append <elemtype> <nnode> <N1 ... Nnnode> ?<matname>? | <x y z>

To create or delete mesh nodes or elements:

- o **<num>|append:** <num> is the identifier (integer > 0) for the node or element. You can use the word 'append' to set a new number automatically. The number of the created, deleted or modified entity is returned as the result. When deleting, it is possible to use a list of entities;
- o **<elemtype>:** must be one of "onlypoints|linear|triangle|quadrilateral|tetrahedra|hexahedra|prism";
- o **<nnode>** is the number of nodes an element has;
- o **<N1 ... Nnnode>** is a Tcl list with the element connectivities;
- o **<matname>** is the optional element material name;
- o **<x y z>** are the node coordinates. If the z coordinate is missing, it is set to z=0.0.

Examples:

```
GiD_Mesh create node append {1.5 3.4e2 6.0}
GiD_Mesh create element 58 triangle 3 {7 15 2} steel
GiD_Mesh delete element {58 60}
```

GiD_Result create|delete|get|get_nodes <data>

To create, delete or get postprocess results:

- o GiD_Result create {Result header} { result_id scalar|vector|matrix_values} {...}
{...} : these creation parameters are the same as for the postprocess results format (see [Postprocess results format: ProjectName.post.res](#), [ProjectName.flavia.res](#));

- GiD_Result delete {Result_name result_analysis step_value} : deletes one result;
- GiD_Result get [-max | -min | -compmax | -compmin | -info] {Result_name result_analysis step_value} : retrieves the results value list of the specified result; or if one of the **-max**, **-min**, **-compmax**, **-compmin**, or **-info** flags was specified: the minimum/maximum value of the result, every minimum/maximum of the components of the result, or the header information of the result (with type and location) is retrieved, respectively;
- GiD_Result get_nodes.

Examples:

```
GiD_Result create {Result "Res Nodal 1" "Load analysis" 4 Scalar
OnNodes} {1 2} {2 2} {113 2} {3 5} {112 4}
GiD_Result create {Result "Res Gauss 1" "Load analysis" 4 Scalar
OnGaussPoints "My Gauss"} {165 2} {2} {3} {164 5} {4} {3}
GiD_Result delete {"Res Nodal 1" "Load analysis" 4}
```

GiD_ModifiedFileFlag set|get ?<value>?

There is a GiD internal flag to indicate that the model has changed, and must be saved before exit.

With this command it is possible to set or get this flag value:

- **<value>** is only required for **set**: must be 0 (false), or 1 (true).

Example:

```
ModifiedFileFlag set 1
ModifiedFileFlag get
```

GiD_MustRemeshFlag set|get ?<value>?

There is a GiD internal flag to indicate that the geometry, conditions, etc. have changed, and that the mesh must be re-generated before calculations are performed.

With this command it is possible to set or get this flag value:

- **<value>** is only required for **set**: must be 0 (false), or 1 (true).

Example:

```
GID_MustRemeshFlag set 1
GID_MustRemeshFlag get
```

GiD_SetModelName <name>

To change the current model name.

GiD_Set <varname> ?<value>?

This command is used to set or get GiD variables. GiD variables can be found through the **Right buttons** menu under the option Utilities -> Variables:

- **<varname>** is the name of the variable;
- **<value>** if this is omitted, the current variable value is returned (analogous with 'GiD_Info variables <varname>').

Example:

```
GiD_Set CreateAlwaysNewPoint 1
```

drawopengl

It is possible to use OpenGL commands directly from GiD-Tcl by using the command "drawopengl draw". For example, for C/C++ use:

```
glBegin(GL_LINES);
glVertex(x1,y1,z1);
glVertex(x2,y2,z2);
glEnd();
```

for GiD-Tcl use:

```
drawopengl draw -begin lines
drawopengl draw -vertex [list $x1 $y1 $z1]
drawopengl draw -vertex [list $x2 $y2 $z2]
drawopengl draw -end
```

The standard syntax must be changed according to these rules: - OpenGL constants: "GL" prefix and underscore character '_' must be removed; the command must be written in lowercase.

Example:

```
GL_COLOR_MATERIAL -> colormaterial
```

- OpenGL functions: "GL" prefix must be removed and the command written in lowercase. Pass parameters as list, without using parentheses ()

Example:

```
glBegin(GL_LINES) → glBegin lines
```

The subcommand "drawopengl draw" provides access to standard OpenGL commands, but other "drawopengl" special GiD subcommands also exist:

- **register <tlfunc>** Register a Tcl procedure to be invoked automatically when redrawing the scene. It returns a handle to unregister.

Example:

```
proc MyRedrawProcedure { } { ...body... }
set id [drawopengl register MyRedrawProcedure]
```

- **unregister <handle>** Unregister a procedure previously registered with **register**.

Example:

```
drawopengl unregister $id
```

- **registercondition <tlfunc> <cond>** Register a Tcl procedure to be invoked automatically when redrawing the specified condition. It returns a handle to unregister.
- **unregistercondition <cond>** Unregister a procedure previously registered with **registercondition**.
- **draw <-cmd args -cmd args>** This is the most important subcommand, it calls standard OpenGL commands. See the list of supported OpenGL functions.
- **drawtext <text>** Draw a text more easily than using standard OpenGL commands (draw in the current 2D location, see rasterpos OpenGL command).

Example:

```
drawopengl draw -rasterpos [list $x $y]
drawopengl drawtext "hello world"
```

- **drawentity ?-mode normal|filled? point|line|surface|volume|node|element|dimension <id list>** To draw an internal GiD preprocess entity.

Example:

```
drawentity -mode filled surface "1 5 6"
```

- **project <x y z>** Given three world coordinates, this returns the corresponding three window coordinates.
- **unproject <x y z>** Given three window coordinates, this returns the corresponding three world coordinates.
- **doscrczoffset <boolean>** Special trick to avoid the lines on surfaces hidden by the surfaces.

List of supported OpenGL functions:

accum alphafunc begin blendfunc call calllist clear clearaccum
 clearcolor cleardepth clearstencil clipplane color colormask
 colormaterial copypixels cullface deletelists depthfunc
 depthmask dfactorBlendTable disable drawbuffer drawpixels
 edgeflag enable end endlist evalcoord1 evalcoord2 evalmesh1
 evalmesh2 finish flush fog frontface frustum genlists hint
 hintModeTable initnames light lightmodel linestipple linewidth
 loadidentity loadmatrix loadname lookat map1 map2 mapgrid1
 mapgrid2 material matrixmode modeColorMatTable multmatrix
 newlist newListTable normal opStencilTable opStencilTable ortho
 perspective pickmatrix pixeltransfer pixelzoom pointsize
 polygonmode popattrib popmatrix popname pushattrib pushmatrix
 pushname rasterpos readbuffer readpixels rect rendermode rotate
 scale scissor selectbuffer shademodel stencilfunc stencilmask
 stencilop texcoord texenv texgen teximage1d teximage2d
 texparameter translate vertex viewport

List of special non OpenGL standard functions:

getselection

List of supported OpenGL constants:

accum accumbuffer accumbufferbit add alphatest always allattrib
 allattribbits ambient ambientanddiffuse autonormal aux0 aux1
 aux2 aux3 back backleft backright blend bluebias bluescale ccw
 clamp clipplane0 clipplane1 clipplane2 clipplane3 clipplane4
 clipplane5 colorbuffer colorbufferbit colorindex colormaterial
 compile compileandexecute constantattenuation cullface current
 currentbit cw decal decr depthbuffer depthbufferbit depthtest
 diffuse dither dstalpha dstcolor enable enablebit emission equal
 eval evalbit exp exp2 eyelinear eyeplane feedback fill flat fog
 fogbit fogcolor fogdensity fogend fogmode fogstart front
 frontandback frontleft frontright gequal greater greenbias
 greenscale hint hintbit incr invert keep left lequal less light0

```

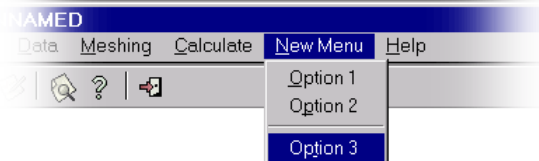
light1 light2 light3 light4 light5 light6 light7 lighting
lightingbit lightmodelambient lightmodellocalviewer
lightmodeltwoside line linebit linear linearattenuation lineloop
lines linesmooth linestipple linestrip list listbit load
map1color4 map1normal map1texturecoord1 map1texturecoord2
map1texturecoord3 map1texturecoord4 map1vertex3 map1vertex4
map2color4 map2normal map2texturecoord1 map2texturecoord2
map2texturecoord3 map2texturecoord4 map2vertex3 map2vertex4
modelview modulate mult nearest never none normalize notequal
objectlinear objectplane one oneminusdstalpha oneminusdstcolor
oneminussrclalpha oneminussrccolor packalignment packlsbfirst
packrowlength packskippixels packskiprows packswapbytes
pixelmode pixelmodebit point pointbit points polygon polygonbit
polygonstipple polygonstipplebit position projection q
quadraticattenuation quads quadstrip r redbias redscale render
repeat replace return right s scissor scissorbit select
shininess smooth specular spheremap spotcutoff spotdirection
spotexponent srclalpha srclalphasaturate srccolor stenciltest
stencilbuffer stencilbufferbit t texture texture1d texture2d
texturebit texturebordercolor textureenv textureenvcolor
textureenvmode texturegenmode texturegens texturegent
texturemagfilter textureminfilter texturewraps texturewrapt
transform transformbit triangles trianglefan trianglestrip
unpackalignment unpacklsbfirst unpackrowlength unpackskippixels
unpackskiprows unpackswapbytes viewport viewportbit zero

```

You can find more information about standard OpenGL functions in a guide to OpenGL.

Managing menus

GiD offers you the opportunity to customize the **pull-down** menus. You can add new menus or to change the existing ones. If you are creating a problem type, these functions should be called from the `InitGIDProject` or `InitGIDPostProcess` functions (see [TCL/TK EXTENSION](#)).



Note: Menus and option menus are identified by their names.

Note: It is not necessary to restore the menus when leaving the problem type, GiD does this automatically.

The Tcl functions are:

- **GiDMenu::Create { menu_name_untranslated prepost {pos -1} {translationfunc _} }**

Creates a new menu. New menus are inserted between the **Calculate** and **Help** menus.

- menu_name_untranslated: text of the new menu (English).
- prepost can have these values:
"PRE" to create the menu only in GiD Preprocess.
"POST" to create the menu only in GiD Postprocess.
"PREPOST" to create the menu in both Pre- and Postprocess.
- pos: optional, index where the new menu will be inserted (by default it is inserted before the 'Help' menu)
- translationfunc: optional, must be _ for GiD strings (default), or = for problemtype strings

- **GiDMenu::Delete { menu_name_untranslated prepost {translationfunc _} }**

Deletes a menu.

- menu_name_untranslated: text of the menu to be deleted (English).
- prepost can have these values:
"PRE" to delete the menu only in GiD Preprocess.
"POST" to delete the menu only in GiD Postprocess.
"PREPOST" to delete the menu in both Pre- and Postprocess.
- translationfunc: optional, must be _ for GiD strings (default), or = for problemtype strings

- **GiDMenu::InsertOption { menu_name_untranslated option_name_untranslated position prepost command {acceler ""} {icon ""} {ins_repl "replace"} {translationfunc _} }**

Creates a new option for a given menu in a given position (positions start at 0, the word 'end' can be used for the last one).

- menu_name_untranslated: text of the menu into which you wish to insert the new option (English), e.g "Utilities"
- option_name_untranslated: name of the new option (English) you want to insert.

The option name, is a menu sublevels sublevels list, like [list "List" "Points"]

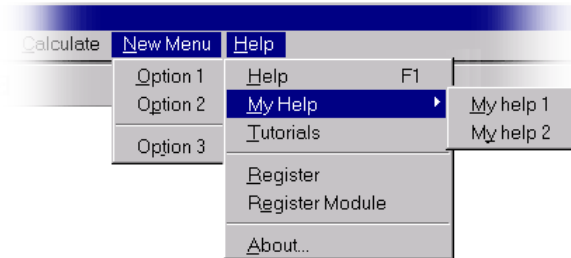
If you wish to insert a separator line in the menu, put "---" as the option_name.

- position: position in the menu where the option is to be inserted. Note that positions start at 0, and separator lines also count.

- `prepost`: this argument can have the following values:
"PRE" to insert the option into GiD Preprocess menus.
"POST" to insert the option into GiD Postprocess menus.
"PREPOST" to insert the option into both Pre- and Postprocess menus.
 - `command`: is the command called when the menu option is selected.
 - `acceler`: optional, key accelerator, like "Ctrl-s"
 - `icon`: optional, name of a 16x16 pixels icon to show in the menu
 - `ins_repl`: optional, if the argument is:
 - ◆ `replace`: (default) the new option replaces the option in the given position
 - ◆ `insert`: the new option is inserted before the given position.
 - ◆ `insertafter`: the new option is inserted after the given position.
 - `translationfunc`: optional, must be `_` for GiD strings (default), or `=` for problemtype strings
- **GiDMenu::RemoveOption {menu_name_untranslated option_name_untranslated prepost {translationfunc _}}**
Removes an option from a given menu.
 - `menu_name_untranslated`: name of the menu (English) which contains the option you want to remove. e.g "Utilities"
 - `option_name_untranslated`: name of the option (English) you want to remove. The option name, is a menu sublevels list, like [list "List" "Points"]
 - `prepost`: this argument can have the following values:
"PRE" to insert the option into GiD Preprocess menus.
"POST" to insert the option into GiD Postprocess menus.
"PREPOST" to insert the option into both Pre- and Postprocess menus.
 - `translationfunc`: optional, must be `_` for GiD strings (default), or `=` for problemtype strings
- **GiDMenu::ModifyOption { menu_name_untranslated option_name_untranslated prepost new_option_name {new_command -default-} {new_acceler -default-} {new_icon -default-} {translationfunc _} }**
Edit an existent option from a given menu.
Some parameters can be ``-default-'` to keep the current value for the command, accelerator, etc
- **GiDMenu::UpdateMenus {}**
Updates changes made on menus. This function must be called when all calls to create, delete or modify menus are made.

Example: creating and modifying menus

In this example we create a new menu called "New Menu" and we modify the GiD **Help** menu:



The code to make these changes would be:

```
GiDMenu::Create "New Menu" "PRE" -1 =
GiDMenu::InsertOption "New Menu" [list "Option 1"] 0 PRE "Command_1"
"" "" replace =
GiDMenu::InsertOption "New Menu" [list "Option 2"] 1 PRE "Command_2"
"" "" replace =
GiDMenu::InsertOption "New Menu" [list "---"] 2 PRE "" "" "" replace =
GiDMenu::InsertOption "New Menu" [list "Option 3"] 3 PRE "Command_3"
"" "" replace =

GiDMenu::InsertOption "Help" [list "My Help"] 1 PRE "" "" "" insert _
GiDMenu::InsertOption "Help" [list "My Help" "My help 1"] 0 PRE
"Command_help1" "" "" replace _
GiDMenu::InsertOption "Help" [list "My Help" "My help 2"] 1 PRE
"Command_help2" "" "" replace _

GiDMenu::RemoveOption "Help" [list "Customization Help"] PRE _
GiDMenu::RemoveOption "Help" [list "What is new"] PRE _
GiDMenu::RemoveOption "Help" [list "FAQ"] PRE _

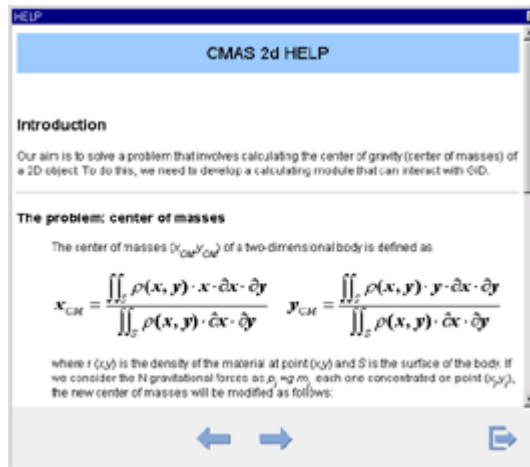
GiDMenu::UpdateMenus
```

HTML support

Problem type developers can take advantage of the internal HTML browser if they wish to provide online help.

HelpWindow

1. Create a directory named `html` inside your Problem Type directory
2. Call **HelpWindow "CUSTOM_HELP" "problem_type_name"**, where `problem_type_name` is the name of your problem type with the `.gid` extension (e.g. `Examples/emas2d.gid`).
3. The function **HelpWindow** opens the file `"index.html"` which must be inside the `html` folder.



It is a good idea to call the function **HelpWindow "CUSTOM_HELP" "problem_type_name"** using the menu functions (see [Managing menus](#)).

Example: Adding a customized HTML help in the Help menu for the CMAS2D problem type:

```
GidMenu::InsertOption "Help" [list "Help CMAS2D"] 0 PREPOST
{HelpWindow "CUSTOM_HELP" "Examples/emas2d.gid"} "" "" insert _
GidMenu::UpdateMenus
```


Note: In order to test this example, the html file `problemtypes/Examples/cmas2d.gid/html/index.html` must be provided.

GiDCustomHelp

With GiD version 7.4 and later, problem type developers can take advantage of the new help format. It is essentially the same html content, but now with an enhanced look and structure. The GiDCustomHelp procedure below is how you can show help using the new format:

```
GiDCustomHelp ?args?
```

where args is a list of pairs option value. The valid options are:

- **-title** : specifies the title of the help window. By default it is "Help on <problem_type_name>".
- **-dir** : gives the path for the help content. If **-dir** is missing it defaults to "<ProblemType dir>/html". Multilingual content could be present; in such a case it is assumed that there is a directory for each language provided. If the current language is not found, language 'en' (for English) is tried. Finally, if 'en' is not found the value provided for **-dir** is assumed as the base directory for the help content.
- **-start** : is a path to an html link (and is relative to the value of **-dir**). For instance:

```
-start html-version  
-start html-tutorials/tutorial_1
```
- **-report** : is a boolean value indicating if the window format is report. If **-report** is 1, no tree is shown and only the content pane is displayed.

Structure of the help content

Assuming that html has been chosen as the base directory for the multilingual help content, the following structure is possible:

```
html  
  \__ en - English content  
  \__ es - Spanish content
```

Each content will probably have a directory structure to organize the information. By default the help system builds a tree resembling the directory structure of the help content. In this way there will be an internal node for each subdirectory, and the html documents will be the terminal nodes of the tree.

You can also provide a `help.conf` configuration file in order to provide more information about the structure of the help. In a help file you can specify a table of contents (`TocPage`), help subdirectories (`HelpDirs`) and an index of topics (`IndexPage`).

HelpDirs

With `HelpDirs` we can specify which of the subdirectories will be internal nodes of the help tree. Moreover, we can specify labels for the nodes and a link to load when a particular node is clicked. The link is relative the node. For instance:

```
HelpDirs {html-version "GiD Help" "intro/intro.html"} \
        {html-customization "GiD Customization"} \
        {html-faq "Frequently Asked Questions"} \
        {html-tutorials "GiD Tutorials" "tutorials_toc.html"} \
        {html_whatsnew "What's New"}
```

TocPage

`TocPage` defines an html page as a table of contents for the current node (current directory). We have considered two ways of specifying a table of contents:

```
<UL> <LI> ... </UL> (default)
<DT> <DL> ... </DT>
```

The first is the one generated by `texinfo`.

For instance:

```
TocPage gid_toc.html
TocPage contents.ht DT
```

IndexPage

If we specify a topic index by `IndexPage`, we can take advantage of the search index. In `IndexPage` we can provide a set of html index pages along with the structure type of the index. The type of the index could be:

```
<DIR> <LI> ... </DIR> (default)
<UL> <LI> ... </UL> (only one level of <UL>)
```

The first is the one generated by `texinfo`.

For instance:

```
IndexPage html-version/gid_18.html html-faq/faq_11.html
```

CUSTOM DATA WINDOWS

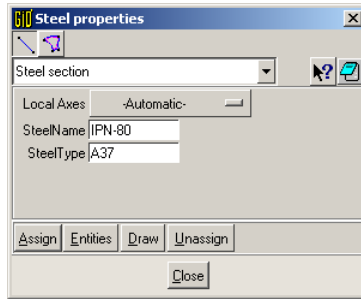
In this section the **Tcl/Tk** (scripted) customization of the look and feel of the data windows is shown. The layout of the properties drawn in the interior of any of the data windows - either Conditions, Materials, Interval Data or Problem Data - can be customized by a feature called **TkWidget**; moreover, the common behaviour of two specific data windows, Conditions and Materials, can be modified by a Tcl procedure provided for that purpose. This common behaviour includes, in the case of Materials for example, assigning/unassigning, drawing, geometry types, where to assign materials, creating/deleting materials, etc.

TkWidget

The problem type developer can change the way a **QUESTION** is displayed and if he wishes he can also change the whole contents of a window, while maintaining the basic behavior of the data set, i.e. in the Condition window: assign, unassign, draw; in the Material window: create material, delete material; and so on.

With the default layout for the data windows, the questions are placed one after another in one column inside a container frame, the **QUESTION**'s label in column zero and the **VALUE** in column one. For an example see the picture below.

```
CONDITION: Steel_section
CONDTYPE: over lines
CONDMESHTYPE: over body elements
QUESTION: Local_Axes#LA# (-Default-, -Automatic-)
VALUE: -Default-
QUESTION: SteelName
VALUE: IPN-80
QUESTION: SteelType
VALUE: A37
END CONDITION
```



Default layout in data windows

The developer can override this behavior using **TKWIDGET**. **TKWIDGET** is defined as an attribute of a **QUESTION** and the value associated with it must be the name of a Tcl procedure, normally implemented in a Tcl file for the problem type. This procedure will take care of drawing the **QUESTION**. A **TKWIDGET** may also draw the entire contents of the window and deal with some events related to the window and its data.

The prototype of a **TKWIDGET** procedure is as follow:

```
proc TKWidgetProc {event args} {
    switch $event {
        INIT {
            ...
        }
        SYNC {
            ...
        }
        DEPEND {
            ...
        }
        CLOSE {
            ...
        }
    }
}
```

The procedure should return:

- an empty string "" meaning that every thing was OK; or

- a two-list element {ERROR-TYPE Description} where ERROR-TYPE could be ERROR or WARNING. ERROR means that something is wrong and the action should be aborted. If ERROR-TYPE is the WARNING then the action is not aborted but Description is shown as a message. In any case, if Description is not empty a message is displayed.

The argument event is the type of event and args is the list of arguments depending on the event type. The possible events are: **INIT**, **SYNC**, **CLOSE** and **DEPEND**. Below is a description of each event.

- **INIT**: this event is triggered when **GiD** needs to display the corresponding **QUESTION** and the list of arguments is {frame row-var GDN STRUCT QUESTION}: frame is the container frame where the widget should be placed; row-var is the name of the variable, used by **GiD**, with the current row in the frame; **GDN** and **STRUCT** are the names of internal variables needed to access the values of the data; **QUESTION** is the QUESTION's name for which the **TKWIDGET** procedure was invoked. Normally the code for this event should initialize some variables and draw the widget.
- **SYNC**: this is triggered when **GiD** requires a synchronization of the data. Normally it involves updating some of the QUESTIONS of the data set. The argument list is {GDN STRUCT QUESTION}.
- **CLOSE**: this is triggered before closing the window, as mentioned this can be canceled if an **ERROR** is returned from the procedure.
- **DEPEND**: this event is triggered when a dependence is executed over the **QUESTION** for which the **TKWIDGET** is defined, ie. that **QUESTION** is an lvalue of the dependence. The list of arguments is {GDN STRUCT QUESTION ACTION value} where **GDN**, **STRUCT** and **QUESTION** are as before, **ACTION** could be **SET**, **HIDE** or **RESTORE** and value is the value assigned in the dependence.

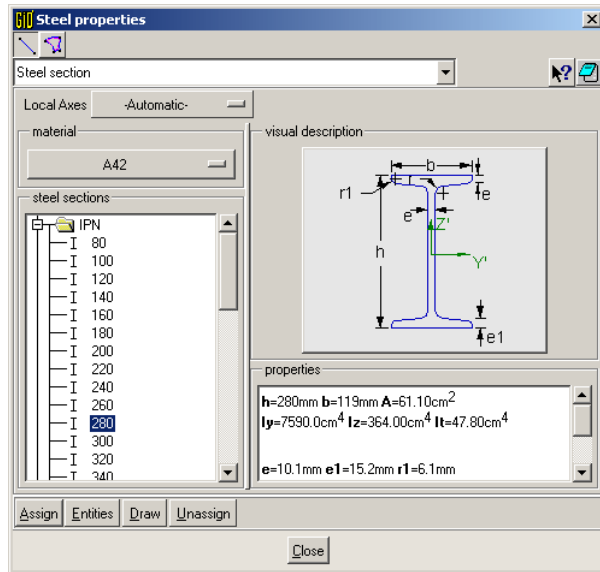
The picture below shows a fragment of the data definition file and the **GUI** obtained. This sample is taken from the problem type RamSeries/rambshell and in this case the **TKWIDGET** is used to create the whole contents of the condition windows. For a full implementation, please download the problem type and check it.

```
CONDITION: Steel_section
CONDTYPE: over lines
CONDMESHTYPE: over body elements
QUESTION: Local_Axes#LA#(-Default-,-Automatic-)
VALUE: -Default-
QUESTION: SteelName
VALUE: -
QUESTION: SteelType
```

```

VALUE: -
TKWIDGET: SteelSections
END CONDITION

```



Customized layout in data windows

Data Windows Behaviour

In this subsection we explain a Tcl procedure used to configure the common behaviour of Materials. We are working on providing a similar functionality for Conditions using the same interface.

`GiD_DataBehaviour` controls properties of data windows for Materials and Conditions (not currently implemented). For Materials we can modify the behaviour of assign, draw, unassign, impexp (import/export), new, modify and delete. We can also specify the entity type list with the assign option through the subcommands `geomlist` and `meshlist`.

The syntax of the procedure is as follows:

```
GiD_DataBehaviour data_class name ?cmd? proplist
```

where

- `data_class` could be "material" if we want to modify the behaviour of a particular material, or "materials" if a whole book is to be modified;
- `name` takes the value of a material's name or a book's name, depending on the value of `data_class`;
- `cmd` can take one of the values: `show`, `hide`, `disable`, `geomlist` and `meshlist`;
- `proplist` is a list of options or entity types. When the value of `cmd` is `show`, `hide` or `disable`, then `proplist` can be a subset of `{assign draw unassign impexp new modify delete}`. If the value of `cmd` is `show` it makes the option visible, if the value is `hide` then the option is not visible, and when the value is `disable` then the option is visible but unavailable. When the value of `cmd` is `geomlist` then `proplist` can take a subset of `{points lines surfaces volumes}` defining the entities that can have the material assigned when in geometry mode; if the value of `cmd` is `meshlist` then `proplist` can take the value `elements`. Bear in mind that only elements can have a material assigned in mesh mode. If `cmd` is not provided, the corresponding state for each of the items provided in `proplist` is obtained as a result.

Example:

```
GiD_DataBehaviour materials Table geomlist {surfaces volumes}
GiD_DataBehaviour materials Solid hide {delete impexp}
```

GiD version

Normally, a problem type requires a minimum version of GiD to run. Because the problem type can be distributed or sold separately from GiD, it is important to check the GiD version before continuing with the execution of the problem type. GiD offers a function, **GiDVersionCmp**, which compares the version of the GiD currently being run with a given version.

GiDVersionCmp { Version }

This returns a negative integer if `Version` is greater than the currently executed GiD version; zero if the two versions are identical; and a positive integer if `Version` is less than the GiD version.

Note: This function will always return the value -1 if the GiD version is previous to 6.1.5.

Example:

```
proc InitGIDProject { dir } {
```



```
global GidPriv
set VersionRequired "8.0"
set comp -1
catch {
    set comp [GiDVersionCmp $VersionRequired]
}
if { $comp < 0 } {
    WarnWin [= "This interface requires GiD %s or later"
$VersionRequired]
}
}
```

Using EXEC in GiD

The Tcl language has the `exec` command used to invoke a subprocess. This command treats its arguments as the specification of one or more subprocesses to execute. It is possible to invoke a subprocess from GiD using this option.

Example: invoking a process in the background

```
exec netscape http://www.gidhome.com &
```

Note: In Windows, instead of `&` it is necessary to put `>& NUL: &` to run the process in the background.

Example:

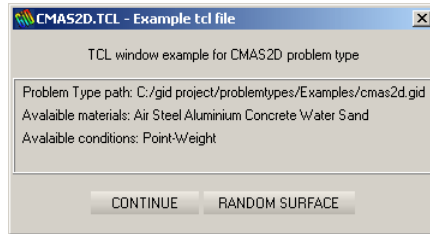
```
exec PROGRAM_NAME >& NUL: &
```

Detailed example - Tcl/Tk extension creation

Here is a step by step example of how to create a Tcl/Tk extension. In this example we will create the file `cmas2d.tcl`, so we will be extending the capabilities of the **cmas2d** problem type. The file `cmas2d.tcl` has to be placed inside the **cdmas2d** Problem Type directory.

Note: The **cmas2d** problem type calculates the center of mass of a 2D surface. This problem type is located inside the Problem Types directory, in the GiD directory.

In this example, the `cmas2d.tcl` file creates a window which appears when the problem type is selected.



Window created in the `cmas2d.tcl` example file

This window gives information about the location, materials and conditions of the problem type. The window has two buttons: **CONTINUE** lets you continue working with the **cmas2d** problem type; **RANDOM SURFACE** creates a random 2D surface in the plane XY.

What follows is the Tcl code for the example. There are three main procedures in the `cmas2d.tcl` file:

- **proc InitGIDProject {dir}**

```
proc InitGIDProject {dir} {
    set materials [GiD_Info materials]
    set conditions [GiD_Info conditions ovpnt]
    CreateWindow $dir $materials $conditions
}
```

This is the main procedure. It is executed when the problem type is selected. It calls the **CreateWindow** procedure.

- **proc CreateWindow {dir mat cond}**

```
proc CreateWindow {dir mat cond} {
    set w .gid.win_example
    InitWindow $w [= "CMAS2D.Tcl - Example tcl file"] ExampleCMAS2D
    "" "" 1
    frame $w.top
    label $w.top.title_text -text [= "Tcl window example for CMAS2D
problem type"]
    frame $w.information -relief ridge -bd 2
```

```

    label $w.information.path -text [= "Problem Type path: %s "
$dir]
    label $w.information.materials -text [= "Available materials:
%s" $mat]
    label $w.information.conditions -text [= "Available
conditions: %s" $cond]
    frame $w.bottom
    button $w.bottom.start -text [= "CONTINUE"] -height 1 -width 14
-command [list destroy $w]
    button $w.bottom.random -text [= "RANDOM SURFACE"] \
        -height 1 -width 20 -command [list CreateRandomSurface
$w]

pack $w.top.title_text -pady 10
pack $w.information.path $w.information.materials \
    $w.information.conditions -side top -anchor w
pack $w.bottom.start $w.bottom.random -side left -anchor
center
pack $w.top
pack $w.information -expand yes -fill both
pack $w.bottom -side top -padx 6 -pady 10 -ipady 2
}

```

This procedure creates the window with information about the path, the materials and the conditions of the project. The window has two buttons: if **CONTINUE** is pressed the window is dismissed; if **RANDOM SURFACE** is pressed, it calls the **CreateRandomSurface** procedure.

- **proc CreateRandomSurface {w}**

```

proc CreateRandomSurface {w} {
    set ret [tk_dialogRAM $w.dialog [= "Warning"] \
        [= "Warning: this will create a nurbs surface in your
current project"] "" 1 [= "Ok"] [= "Cancel"]]
    if {$ret ==0} {
        Create_surface
        destroy $w
    }
}

```

This procedure is called when the **RANDOM SURFACE** button is pressed. Before creating the surface, a dialog box asks you to continue with or cancel the creation of the surface. If the surface is to be created, the **Create_surface** procedure is called. Then, the window is destroyed.

```

proc Create_surface {} {
    set a_x [expr rand()*10]
    set a_y [expr rand()*10]

    set b_x [expr $a_x + rand()*10]
    set b_y [expr $a_y + rand()*10]

    set c_x [expr $b_x + rand()*10]
    set c_y [expr $b_y - rand()*10]

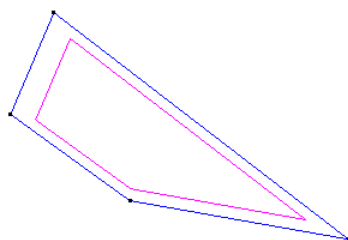
    if {$a_y < $c_y} {
        set d_y [expr $a_y - rand()*10]
        set d_x [expr $a_x + rand()*10]
    } else {
        set d_y [expr $c_y - rand()*10]
        set d_x [expr $c_x - rand()*10]
    }

    GiD_Process escape escape escape escape

    GiD_Process geometry create line \
    $a_x,$a_y,0.000000tol0.176991 \
    $b_x,$b_y,0.000000tol0.176991 \
    $c_x,$c_y,0.000000tol0.176991 \
    $d_x,$d_y,0.000000tol0.176991 \
    close

    GiD_Process escape escape escape escape
    GiD_Process geometry create NurbsSurface Automatic 4 escape
    GiD_Process zoom frame escape escape escape escape
}

```



A 2D surface (a four-sided 2D polygon) is created. The points of this surface are chosen at random.