Impact Users Manual

Table of Contents

Impact Users Manual

This chapter is intended for the user more than the programmer. Reference for the Fembic indata program is also included

Installation

Impact is a Java program which means that there is no compilation of sourcecode or similar to be done. However, there are some programs you need to install to be able to run Impact and to see the results.

Start by downloading the program files of impact from http://sourceforge.net/projects/impact

The file is a .zip file and must be untarred using the command `tar -xvf filename.zip` if you are running Linux. For Windows users the Winzip program will handle the expansion.

You will now have a directory for impact (highly originally called impact) where some files and directories will reside. They should be:

- README
- jama (dir) − The directory containing classes for matrix algebra
- run (dir) − All the source code for Impact
- Impact.gid (dir) − Configuration files for the GID pre/post processor
- manual_Users (dir) − Users manual
- manual_Programmers (dir) − Programmers manual and program installation
- examples (dir) − All the example problems for Impact (.in)

Prerequisites

To get Impact working you need:

1. A Java engine
2. A Pre− and Postprocessor (optional but recommended)

Java_engine

A good Java engine is the Sun version which can be found at http://java.sun.com/j2se/1.4/download.html. You can take either the Runtime environment or the Software Development Kit.

There are several alternative Java engines. IBM has one which is very fast and is also recommended.

After installation, you can run the solver by going to the Impact directory (cd Impact) and then writing `java run.Impact examples/xxxxx.in` where xxxxx is the name of your indatafile that you want to run with the solver.

Impact will create two outdatafiles:

- xxxxx.in.flavia.res

- xxxxx.in.flavia.msh

These files can be read by the GID pre−and postprocessor.

Processor

The recommended pre and postprocessor is GID. It is freely available as a limited edition. You will need to download a version later than 6.2.0d since Impact uses features that are currently being implemented. You can download GID from http://gid.cimne.upc.es

This is how you should set up and use GID for Preprocessing:

1. Run the installation file for GID and install the program.
2. If you haven't installed Impact, proceed to do this.
3. Look in the GID directory for a subdirectory called problemtypes and go there
4. Make a new subdirectory called Impact
5. Now copy the directory Impact.gid from where you installed Impact, making sure all files come with it
6. The directory structure should now be GiD/problemtypes/Impact/Impact.gid/some files
7. If you now start GiD, you should find Impact as an option under the DATA menu.
8. GiD can now export indata files to Impact via the File−>Export−>CalculationFile menu

Impact also has a built in pre− and postprocessor which is under development. They are accessable from the ImpactGUI.

Solving problems with impact

The solution process is made in three stages:

1. Creation of a model using a pre−processor or direct writing of the Fembic indata file
2. Solution using the Impact program
3. Presentation of the results using a post−processor and the result files from the solution

It is simplest to run Impact and the built in pre− and postprocessors from the GUI. To do that, just run the ImpactGUI.bat file in this directory if you are a Windows user or make the ImpactGUI.sh runnable (chmod 777 ImpactGUI.sh) and run that with ./ImpactGUI if you are a Linux/Unix/Mac user. Alternatively, just write bash ImpactGUI.sh to start.

Creating the indata file

The model can be created by the help of a pre−processor. A good one is called GID and is freely available for small models. it can be downloaded from here in versions for both unix and windows. Be sure to get a version higher than 6.2.0 due to advanced features used by Impact.

After installation of GID, you should copy the directory Impact/Impact and all its contents into the GID subdirectory called GID/problemfiles. This will install an extra option on the GID preprocessor menu and configure GID for Impact file format.

```
cp −r homedirectory/Impact/Impact /homes/myuser/GID7_0/problemtypes/
```

Before starting to create models with GID, you should set GID in the Impact mode so that your model will be tailored for Impact. This is done by selecting Data −> Problem type −> Impact −> Impact. This will give you a range of options under the Data menu which you can use to set material, boundary conditions etc.

After the model is created, it should be exported to a file. This is preferably done using the File −> Export −> Calculation file feature. It is of course also possible to write the complete file by hand using a favourite ASCII editor. The syntax of a Fembic file is explained in a later chapter.

Summary of how you should use GID for Pre−processing (creating models for Impact)

1. Start by selecting Impact as your solver by Data−>Problemtype−>Impact−>Impact
2. Fill in the problem datas under Data−>Problem Data−>...
3. Create a model and mesh it (read the GiD manual for how to do this)
4. Set materials on all elements using Data−>Materials
5. Set boundary conditions on the nodes using Data−>Conditions
6. You can now export the indata file via Files−>Export−>Calculation file

Solving

The solution of the problem is initiated from the GUI or by writing **java run.Impact *file*** at the command prompt, where ***file*** is the name of the indata file. In the case of a Fembic file, make sure it ends with .in because otherwise Impact will not recognise the format. It is also important that you are placed in the impact directory at the time of execution. A java engine must also be installed on your system before execution.

If you are running some of the example problems supplied, you need to add the path to the examples directory. The syntax then becomes: **java run.Impact *examples/file*** where file applies as above.

If all goes well, you should now see the indata file being parsed by impact and the solution process initiated. Each time results are written, a notice will be written to the screen and you will see that execution is in progress. A solution can take considerable time, so be patient.

Reading the results

The results are printed to the flavia.res and flavia.msh files. They will end up in the same directory as your sourcefile. These are tailor made for the GID and the built in post processor.

Start by firing up GID and switch to post processing mode. Next read in the result file flavia.res. The mesh (flavia.msh) file will be read automatically. You should now see the model on the screen.

Press ctrl−d to set the timestep for deformation. Go from the top of the menu, starting by selecting deformation and then time analysis. Select timestep 0, magnification factor 1.0 and then press apply.

Next press ctrl−v and select the results , time analysis and contour fill. Finally, select gausspointstress and apply.

Finally, press ctrl−m. You should now see the results as an animation. There are plenty of ways to view your results, but I refer to the GID users manual for that.

Summary of how you should use GiD for Post−processing (looking at the results)

1. Fire up GID and switch to post−processing mode.
2. Open the xxxxx.in.flavia.res file. If all goes well, you should be able to see your model.
3. Press ctrl−d to set the timestep for deformation.
4. Go from the top of the menu, starting by selecting deformation and then time analysis.
5. Select timestep 0, magnification factor 1.0 and then press apply.
6. Next press ctrl−v and select the results , time analysis and contour fill.
7. Finally, select gausspointstress and apply.
8. Next press ctrl−m to get a nice animation!

Alternatively, the built in postprocessor is directly accessable from the GUI. Just open the resultfile and select the time you want from the left column and you should see the model. Rotation, moving and zooming is done by holding down any of the mouse buttons while moving the mouse.

Contact handling in Impact

Contacts in impact are handled by two element types:

- Contact_Triangle (CT)
- Contact_Line (CL)

The CT is used to sense contact between nodes and surfaces and the CL senses contact against other CL elements. Together, these two elements can be used to enable contact detection for most cases and models. Both of them are classified as elements which means that they can directly be part of a model mesh as all elements. The user can for example model a wall or a complex rigid contact surface with them.

Since they only have the sole purpose of sensing contact, the have no stiffness at all. This means that if they are used on their own in the model, the nodes connecting them should be fixed by constraints to prevent them from drifting when in contact. It also means that the user can use them in combination with ordinary elements to provide contact sensing where this is not default.

One example where this is useful is when a body has been meshed using solid elements, for example an engine block in a car. This body can then be "dressed" on the outside with a second mesh of contact elements to provide the contact sensitivity against other elements in the car. Any contact sensing inside the engine block is not needed and valuable calculation time can then be saved with this approach.

Some elements have contact sensing as default. Examples of these are:

- Shell_C0_3
- Shell_BT_4
- Rod_2
- Beam_2

When any of these elements is created, one or several contact elements are created by default. These are embedded inside the element and share the element nodes. The rod and beam elements use the Contact_Line element to sense contact. The Shell elements use the Contact_Triangle element to sense contact against the surface and optionally Contact_Line elements at the edges to sense contact against other edges.

The contact elements drain quite a bit of computing resources and as the number of elements increase, so does the amount of computing power since the increase is more than linear. Therefore, some of the elements have options to reduce the contact resolution. This means that the contact sensing will be less accurate during large

deformation of the elements, but the solution will run faster. For this reason, contact sensing has also not been implemented in the solid elements since the user can best minimise the amount of calculations needed, by distributing the contact element where they are needed.

The details of how contact sensing is implemented is explained in the programming manual.

Fembic Indata Format

The fembic indata format is the default indata format for Impact. It is designed to be simple to read and understand, and is written in free format which means that you can type as you wish and do not have to put data at specific locations in the file. A file which is written in Fembic should have a name which ends with **.in** and be in ASCII format.

Comments may be included on any line but must be preceded by a # character. Every text written behind this sign will be ignored and the parser will continue on the next line instead.

The General syntax for this manual is that letters in **bold** are required input for the command. Input in normal writing are optional.

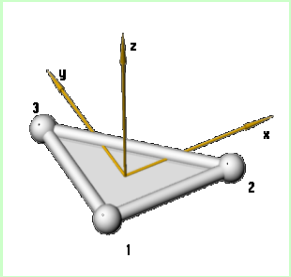
Block Structure

A fembic file is structured in blocks. Each block starts with a keyword, followed by data related to that block. The blocks can come in any order and may be repeated. A block must start on a new line using any of the keywords. Each keyword and the related data will now be described.

Commands

Command	**Elements**
Block	Elements
Description	The elements that make up the finite element model are defined within this block. Only one type of elements can be defined within each block.
Syntax	**Elements of Type *eltype***
Options	*eltype* Any type of element: Rod_2, Beam_2, Solid_Iso_6, Shell_BT_4, Contact_Triangle
Example	Elements of Type Shell_BT_4
See also	Rod_2, Beam, Solid_Iso_6, Shell_BT_4, Shell_C0_3

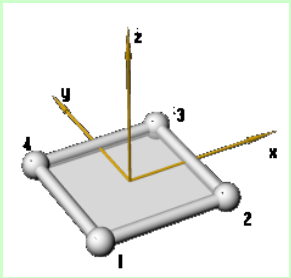
Command	**Beam_2**	
Block	Elements	
Description	This is a simple Beam element which means that it will transfer moment and also take node rotation into account. The cross section is assumed solid circular.	
Syntax	***nr* nodes = [*node1*,*node2*] D=*diameter* material= *elmaterial***	
Options	*nr*	The element number. Must be a unique number in the model, i.e. another element cannot have the same number.
	node1,2,..	The number of the first node etc.
	diameter	the cross section diameter of the Beam. The cross section is solid circular.
	elmaterial	The name of the material that the Beam element uses. This name must be defined under the material block.
Example	1 nodes = [23,24] D = 4.73 material = steel	
See also	Elements, Materials, Nodes	

Command	**Contact_Triangle**
Block	Elements
Description	This is a 3 node contact element. It has no other purpose than detecting if other nodes are about to penetrate it's surface. If this is the case, the element will repel the node with a resulting reaction force onto itself. This element is useful to model contact surfaces. It is also used in most of the other finite elements to handle contact detection.
Syntax	***nr* nodes = [*node1*,*node2*,*node3*] T = *thickness* factor = *factor* friction = *friction***
Options	*nr* — The element number. Must be a unique number in the model, i.e. another element cannot have the same number. *node1,2,..* —  Nodes are defined in a counter clockwise direction as shown in the figure. Results from the element can be local stresses and strains. The picture shows the local axes where the x−axis direction are primarily defined by node 1 and 2. The z−axis is normal to the shell surface. The local y−axis is defined to be octagonal to the x−axis and z−axis. *thickness* — is the thickness of the contact element. The thickness is assumed to be constant over the element width. Nodes outside the thickness are not assumed to be in contact. The contact zone extends to half the thickness on each side of the element. *factor* — The repelling force to be used when a node penetrates. The force will increase linearly as the node intrudes further. *friction* — The friction coefficient to be used. Usually between 0.2 and 0.8 depending on material and condition. Only useful if contact is enabled. If not set, friction is disabled.
Example	1 nodes = [113,118,110] t = 1.0 factor = 100 friction = 0.2
See also	Elements, Nodes, Shell_C0_3, Shell_BT_4, Contact_Line

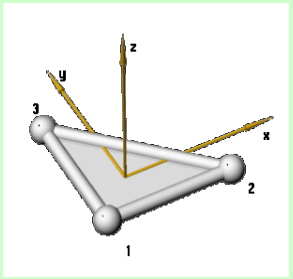
Command	**Contact_Line**	
Block	Elements	
Description	This element is a two node contact element of a line segment. The element will provide contact sensitivity within the diameter of the line. Also ends are detected within the radius. Contact is sensed against nodes and other contact_line elements.	
Syntax	***nr*** **nodes = [*node1*,*node2*] D=*diameter*** factor = *c_factor* contact = *c_type*	
Options	*nr*	The element number. Must be a unique number in the model, i.e. another element cannot have the same number.
	node1,2,..	The number of the first node etc.
	diameter	the cross section diameter of the element. The cross section is solid circular.
	c_factor	The contact factor. This is the reaction force at full penetration of contact node. If nothing is specified, default is 10.
	c_type	Contact type. Can be OFF to disable contact. Default contact type is BASIC which means that contact sensing is enabled.
Example	1 nodes = [23,24] D = 4.73	
See also	Elements, Nodes, Shell_C0_3, Shell_BT_4, Contact_triangle	

Command	**Rod_2**	
Block	Elements	
Description	This element is a two node element of a Rod. The cross section will shrink as the element extends which is important as the rod leaves the elastic state and becomes plastic.	
Syntax	***nr* nodes = [*node1*,*node2*] D=*diameter* material= *elmaterial*** factor = *c_factor* contact = *c_type*	
Options	*nr*	The element number. Must be a unique number in the model, i.e. another element cannot have the same number.
	node1,2,..	The number of the first node etc.
	diameter	the cross section diameter of the rod. The cross section is solid circular.
	elmaterial	The name of the material that the rod element uses. This name must be defined under the material block.
	c_factor	The contact factor. This is the reaction force at full penetration of contact node. If nothing is specified, default is the same as for the contact_line element.
	c_type	Contact type. Can be OFF to disable contact. Default contact type is BASIC which means that a Contact_Line element will represent the rod.
Example	1 nodes = [23,24] D = 4.73 material = steel	
See also	Elements, Materials, Nodes	

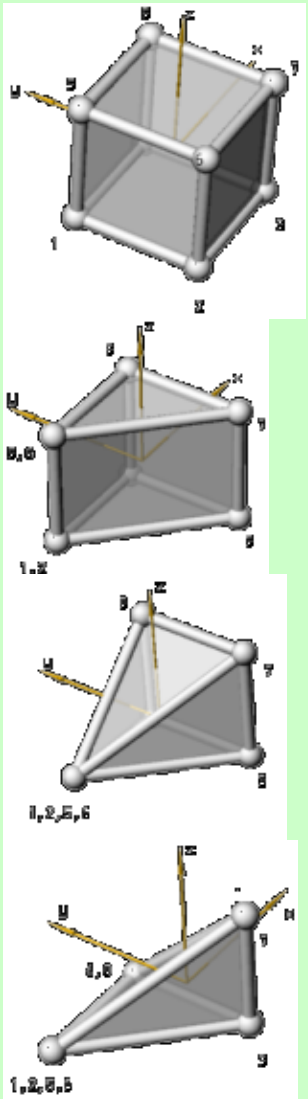
Command	**Beam_Spring_2**	
Block	Elements	
Description	This element is a two node beam spring element. Since it is a spring, both the stiffness and damping can be defined in six directions. The element relies on a local coordinate system which is set up along the element. Therefore, this spring cannot be used if the nodes are at identical position, i.e. the element length is 0. The coordinate system is constantly updated as the element moves. Note that time step issues for this element often be related to the fact that inertia has not been defined on *both* the connecting nodes.	
Syntax	***nr*** **nodes = [*node1*,*node2*,*node3*] material= *elmaterial*** D = *diameter* factor = *c_factor* contact = *c_type*	
Options	*nr*	The element number. Must be a unique number in the model, i.e. another element cannot have the same number.
	node1,2,3	Node 1 and 2 are the start and end nodes respectively. The third node is a control node which defines the plane for the local y−axis of the element. The local x−axis runs from node 1 to node 2. This axis, together with the control node sets up a plane wherein the y−axis will be. The Y−axis is always perpendicular to the x−axis. The local z−axis is then perpendicular to the plane.
	elmaterial	The name of the material that the spring element uses. This material must be of type spring and be defined in the material block. The material defines all the stiffness attributes for the element.
	diameter	The cross section diameter of the spring. This is only used for the contact search. Any node within this diameter will be concidered in contact. This does not have to be defined if contact is not enabled.
	c_factor	The contact factor. This is the reaction force at full penetration of contact node. If nothing is specified, default is the same as for the contact_line element. This does not have to be defined if contact is not enabled.
	c_type	Contact type. Can be set to BASIC to enable contact. Contact is disabled by default for this element.
Example	1 nodes = [23,24,27] material = attrib1	
See also	Elements, Materials, Nodes	

Command	**Shell_BT_4**
Block	Elements
Description	This is a 4 node shell element based on the classical Belytchko−Tsai formulation. It is a very robust design which has become the workhorse in explicit finite element simulations. The element has only one integration point which means that the result are only calculated in one point which is situated in the middle of the element. The advantage is that the element is very fast. The drawback is that it is sensitive to hourglassing. To prevent this, there is an additional compensation built in to the formulation, called hourglass control.
Syntax	***nr*** **nodes = [*node1*,*node2*,*node3*,*node4*] T = *thickness* material = *elmaterial*** NIP = *noip* PIP = *nopip* SHEAR_FACTOR = *shearfactor* HOURGLASS = *hglass* MHC = *mhc* OOPHC = *oophc* RHC = *rhc* LOAD = *loadname* FACTOR = *c_factor* CONTACT = *c_type* FRICTION = *friction* THINNING = *thinning*
Options	*nr* — The element number. Must be a unique number in the model, i.e. another element cannot have the same number. *node1,2,..* —  Nodes are defined in a counter clockwise direction as shown in the figure. Results from the element can be local stresses and strains. The picture shows the local axes where the x−axis direction are primarily defined by node 1 and 2. The z−axis is normal to the shell surface. The local y−axis is defined primarily by node 1 and 4. *thickness* — is the thickness of the shell. The thickness is assumed to be constant over the element width. *elmaterial* — The name of the material that the shell element uses. This name must be defined under the material block. *noip* — the number of integration points through the thickness of the element. All from 1 up to 5 integration points are possible. A minimum of three integration points are recommended. *nopip* — the number of the integration point which results will be printed in the result file. Can be anything from 1 up to NIP. If nothing is specified, it will be the middle point in the shell thickness, for example if NIP = 5, PIP will be equal to 3. *shearfactor* — the arbitrary parameter used to enforce the Kirchhoff normality condition as the shell become thin. Default value is 1.0. *hglass* — a switch to enable or disable hourglass control on the element. It can be either ON or OFF. Default is ON. *mhc*

		the Membrane Hourglass Control factor. This factor is multiplied with the calculated hourglass forces acting in the shell membrane plane. Default is 0.1.
	oophc	the Out Of Plane Hourglass Control factor. This factor is multiplied with the calculated hourglass forces acting out of the membrane plane, causing ex. twist of the element. Default is 0.1.
	rhc	the Rotational Hourglass Control factor. This factor is multiplied with the calculated hourglass moments. Default is 0.1.
	loadname	Name of a load defined under the load block. Pressure onto the shell element is defined this way.
	c_factor	The contact factor. This is the reaction force at full penetration of contact node.
	c_type	Contact type. Can be OFF to disable contact. Default contact type is BASIC which means that two Contact_Triangle elements will represent the surface. This works fine for small deformations of the element. ADVANCED will use four Contact_Triangle elements and be able to handle self contact within the element itself, but at a cost of calculation time. EDGE enables edge contact sensitivity along the edges of the element together with the standard surface contact sensitivity. ADVANCED_EDGE does the latter but together with the advanced contact surface model which uses four Contact_Triangle elements.
	friction	The friction coefficient to be used. Usually between 0.2 and 0.8 depending on material and condition. Only useful if contact is enabled. If not set, friction is disabled.
	thinning	Determines if the shell should reduce thickness at large strains. Useful in pressing simulations. Default is ON. It can be disabled by setting equal to OFF.
Example	1 nodes = [113,118,110,106] nip = 5 t = 1.0 material = steel load = pres	
See also	Elements, Materials, Nodes, Shell_C0_3	

Command	**Shell_C0_3**
Block	Elements
Description	This is a 3 node shell element based on the classical C0 formulation by Belytchko et. al. It complement the BT_4 shell and is common in explicit finite element simulations. The element has only one integration point which means that the result are only calculated in one point which is situated in the middle of the element. The advantage is that the element is very fast. Unlike the BT_4 element, this element does not need hourglass control. Triangulars are however by it's nature stiffer than the BT_4 element which means it should be used with care.
Syntax	***nr* [*node1*,*node2*,*node3*] T = *thickness* material = *elmaterial*** NIP = *noip* PIP = *nopip* LOAD = *loadname* FACTOR = *c_factor* CONTACT = *c_type* FRICTION = *friction* THINNING = *thinning*
Options	*nr* — The element number. Must be a unique number in the model, i.e. another element cannot have the same number. *node1,2,..* —  Nodes are defined in a counter clockwise direction as shown in the figure. Results from the element can be local stresses and strains. The picture shows the local axes where the x−axis direction are primarily defined by node 1 and 2. The z−axis is normal to the shell surface. The local y−axis is defined to be octagonal to the x−axis and z−axis. *thickness* — is the thickness of the shell. The thickness is assumed to be constant over the element width. *elmaterial* — The name of the material that the shell element uses. This name must be defined under the material block. *noip* — the number of integration points through the thickness of the element. All from 1 up to 5 integration points are possible. A minimum of three integration points are recommended. *nopip* — the number of the integration point which results will be printed in the result file. Can be anyting from 1 up to NIP. If nothing is specified, it will be the middle point in the shell thickness, for example if NIP = 5, PIP will be equal to 3. *loadname* — Name of a load defined under the load block. Pressure onto the shell element is defined this way. *c_factor* — The contact factor. This is the reaction force at full penetration of contact node. *c_type* — Parameter to set contact sensitivity type. Can be set to OFF to disable contact. If unspecified, contact is enabled for surface only. If set to EDGE, the surface contact will be

		complemented with edge contact sensitivity.
	friction	The friction coefficient to be used. Usually between 0.2 and 0.8 depending on material and condition. Only useful if contact is enabled. If not set, friction is disabled.
	thinning	Determines if the shell should reduce thickness at large strains. Useful in pressing simulations. Default is ON. It can be disabled by setting equal to OFF.
Example	1 nodes = [113,118,110] nip = 5 t = 1.0 material = steel load = pres	
See also	Elements, Materials, Nodes, Shell_BT_4	

Command	**Solid_Iso_6**
Block	Elements
Description	This is a simple isoparametric solid element with eight integration points as showed in the figure. The element is available with either one integration point which is then situated in the middle of the element, or eight integration points. The benefit of having eight points are the the element will be more stable since hourglass modes cannot occur. At the time of writing, there is no hourglass control algorithm implemented which enables a stable use of one integration point which means that this configuration is currently not recommended. The results from the element are stresses and strains in global directions.
Syntax	***nr* [*node1*,*node2*,*node3*,*node4*, *node5*,*node6*,*node7*,*node8*] material= *elmaterial*** NIP=*noip*
Options	*nr* The element number. Must be a unique number in the model, i.e. another element cannot have the same number. *node1,2,..*  Nodes are defined as shown in the figure where the local axes are shown as well. Results from the element can be local stresses and strains. Other types of solid elements can be created by collapsing the element edges. This is achieved by assigning the same node number to the original nodes on the cube element. For example, the wedge would be achieved by assigning node nr 1 on position 1 and 2 for the solid element. In general, the collapse of elements are not as good as rewriting them from scratch. They are however possible to use, but will not give optimal results, so be aware of this when using them.

	noip	the number of integration points in the element; can either be 1 or 8. This will also change the result files since results are calculated in each integration point and the data from each point will then be printed.
	elmaterial	The name of the material that the rod element uses. This name must be defined under the material block.
Example	1 nodes = [23,24,34,42,65,76,89,33] material = steel nip = 8 2 nodes = [23,23,34,42,65,65,89,33] material = steel nip = 8	
See also	Elements, Materials, Nodes	

Command	**Nodes**
Block	Nodes
Description	The node block starts with the keyword **nodes** on a single line. The following lines should then specify the nodes with one node per line. Impact is designed for three dimensional space problems which means that for each node, all three space coordinates must be defined at all times. If a two dimensional problem is to be solved, each node must be constrained from movement in the third dimension.
Syntax	**Nodes**
Options	–
Example	Nodes
See also	Node

Command	**Node**
Block	Nodes
Description	This command defines a node. Impact is designed for three dimensional space problems which means that for each node, all three space coordinates must be defined at all times. If a two dimensional problem is to be solved, each node must be constrained from movement in the third dimension.
Syntax	***nr* X = *xcoord* Y = *ycoord* Z = *zcoord*** constraint = *cname* loads = *lname* M = *mass* Ixx = *x_inertia* Iyy = *y_inertia* Izz = *z_inertia* Ixy = *xy_inertia* Iyz = *yz_inertia* Ixz = *xz_inertia*
Options	*xcoord, ycoord, zcoord* — the space coordinates for the node in respective direction. The numbers can contain decimals. *cname* — the name of the constraint set that the node should obey. The constraint set must be defined elsewhere in the file under the constraint block. Only one constraint set can be applied on each node. This is an optional parameter and does not have to be defined if the node is free. *lname* — the name of the load set that the node should use. The load set must be defined elsewhere in the file under the load block. Only one load set can be applied to each node. This is an optional parameter and does not have to be defined if there is no load on the node. *mass* — the weight of the concentrated mass applied to the node. The mass is applied to all spatial directions. *x_inertia* — The inertia around global x−axis applied to the node. *y_inertia* — The inertia around global y−axis applied to the node. *z_inertia* — The inertia around global z−axis applied to the node. *xy_inertia* — The xy inertia component. The yx component is assumed equal. *yz_inertia* — The yz inertia component. The zy component is assumed equal. *xz_inertia* — The xz inertia component. The zx component is assumed equal.
Example	Nodes
See also	Node

Command	Constraints
Block	Constraints
Description	Under this block heading, the constraint sets are defined. Each set must be defined on a single line.
Syntax	**Constraints of type *ctype***
Options	*ctype* Any type of constraint: Boundary_Condition, Rigid_Body
Example	Constraints of type Boundary_Condition
See also	Constraint, Node, Load

Command	Boundary_Condition
Block	Constraints
Description	Defines a boundary condition constraint. A constraint controls movement for the nodes in different directions by setting the acceleration and velocity for the node. Any given combination can be set. There is no need to define all the variables. If none is set, the default value is that the node will be uncontrolled in that direction.
Syntax	***name*** ax = *value* ay = *value* az = *value* vx = *value* vy = *value* vz = *value* arx = *value* ary = *value* arz = *value* vrx = *value* vry = *value* vrz = *value* axis = [*node1,node2,node3*] update = *upd*
Options	*name* Name of the constraint. Must be unique. *value* Value of the constraint. Can be either a simple number (constant) or alternatively a variable over time defined as [t1,y1,t2,y2,...,tn,yn] where y1 is the value at time t1 and so on. At this stage, y1 can also be **off** which means that the constraint will not be effective from this time forward until a new value is set. *node1,node2,node3* These nodes set up the local coordinate system for the boundary condition. If these are specified, the values specified in the constraint will be assumed to be relating to this local coordinate system. The local x−axis of the system runs from node1 to node2. Local z−axis is then normal to the plane defined by this x−axis and a vector from node1 to node3. Finally, the y−axis is normal to the x− and z−axis. *upd* The update option is connected to the axis option. If the local coordinate system is defined and update is set to ON, the nodes defining the coordinate system will be continuosly scanned and the system updated. This means the system can rotate over time.
Example	exampleconstraint ax = [0,0,1,1.5,5,off,6,3,100,3] ay = 3.0 az = 0.0
See also	Node, Load

Command	**Rigid_Body**
Block	Constraints
Description	Defines a rigid body constraint. The nodes referring to this constraint are all considered part of a single rigid body. They are all connected to the master node. If specified, the master node can automatically be placed in the centre of gravity for the body. The movement of the body will be controlled from the master node, on which an ordinary boundary condition or load can be placed. The master node will automatically be given the mass and inertia for the rigid body, based on the slave nodes mass, inertia and position.
Syntax	***name*** **master_node =** ***nnum*** update_position = *updt*
Options	*name* — Name of the constraint. Must be unique. *nnum* — Node number of the master node. The node will be moved automatically to the centre of gravity of the rigid body before the solution starts. *updt* — If this is set to ON, the master node will automatically be moved to the centre of mass for the rigid body before the solution starts.
Example	rb1 master_node = 25
See also	Boundary_Condition, Node

Command	**Loads**
Block	Loads
Description	Under this block heading, the load sets are defined. Each set must be defined on a single line.
Syntax	***Loads***
Options	–
Example	Loads
See also	Load, Node, Constraint

Command	**Load**	
Block	Loads	
Description	The loads block is initiated by the heading above on a single line. The loads themselves follows, defined one per line. A load set is applied on nodes or some elements. It consists of concentrated forces in any direction defined by their x,y and z components. Accelerations, such as gravity can also be defined here. Pressure can also be defined.	
Syntax	***name*** fx = *value* fy = *value* fz = *value* mx = *value* my = *value* mz = *value* ax = *acc* ay = *acc* az = *acc* arx = *acc* ary = *acc* arz = *acc* p = *pressure*	
Options	*name*	Name of the load. Must be unique.
	value	Value of the load. Can be either a simple number (constant) or alternatively a variable over time defined as [t1,y1,t2,y2,...,tn,yn] where y1 is the value at time t1 and so on. At this stage, y1 can also be **off** which means that the load will not be effective from this time forward until a new value is set.
	acc	Value of the acceleration. Accelerations are added to the load on a node which makes this the way to simulate gravity. Acceleration can be either a simple number (constant) or alternatively a variable over time defined as [t1,y1,t2,y2,...,tn,yn] where y1 is the acceleration at time t1 and so on. At this stage, y1 can also be **off** which means that the acceleration will not be effective from this time forward until a new value is set.
	pressure	Value of the pressure. Can be either a simple number (constant) or alternatively a variable over time defined as [t1,y1,t2,y2,...,tn,yn] where y1 is the pressure at time t1 and so on. At this stage, y1 can also be **off** which means that the pressure will not be effective from this time forward until a new value is set.
Example	exampleload ax = [0,0,1,1.5,5,off,6,3,100,3] p = 3.0	
See also	Node, Load	

Command	**Materials**	
Block	Materials	
Description	A specific material is defined by setting the parameters of a specific material law. There are several material laws available, depending on material choice. There are laws that are suitable for metals and other more suitable for foams, which may behave differently. The material law is then assigned to one or several elements in the element definition.	
Syntax	**Materials of type *mtype***	
Options	*mtype*	The name of a specific material law. After each block heading, the specific materials are listed with the parameters defined. One material per line.
Example	Materials of Type Elastic	
See also	Elastic, Elastoplastic	

Command	**Elastic**	
Block	Materials	
Description	This is a simple elastic material law.	
Syntax	***name* E = *yvalue* RHO = *dvalue* NU = *nuvalue*** FAILURE_STRAIN = *fstrain* FAILURE_STESS = *fstress*	
Options	***name***	Name of the material. Must be unique.
	yvalue	Young's modulus for the material.
	dvalue	The density of the material.
	nuvalue	Poisson's constant of the material.
	fstrain	The strain at which the material fractures. If an element reaches this strain, it will be removed from the simulation.
	nuvalue	The stress at which the material fractures. If an element reaches this stress, it will be removed from the simulation.
Example	steel E = 210 D = 0.0000078 NU = 0.3	
See also	Elements, Materials, Elastoplastic	

Command	**Elastoplastic**	
Block	Materials	
Description	This is an isoparametric elasto−plastic material law. The elastic Young's modulus defines the stress−strain relation up to the yield stress. Above yield stress, there are several options. The plastic behaviour can be described by the plastic modulus (EP) which defines a linear relation between the stress and effective plastic strain. This relation can also be a curve, defined by a range of stress/strain coordinates. The EP in this case has no function and can be omitted. Finally, the relation can also be dependent on the strain rate in which several stress/strain curves are defined together with a parameter setting the velocity for which each curve is representative. The stress for a certain effective strain value is detemined as a linear interpolation from these curves.	
Syntax	***name*** **E =** ***yvalue*** **RHO =** ***dvalue*** **NU =** ***nuvalue*** **YIELD_STRESS =** ***svalue*** EP = *fvalue* Y1,Y2.. Y9 = *svalue* V1,V2..V9 = *vvalues* FAILURE_STRAIN = *fstrain* FAILURE_STESS = *fstress*	
Options	***name***	Name of the material. Must be unique.
	yvalue	Young's modulus for the material.
	dvalue	The density of the material.
	nuvalue	Poisson's constant of the material.
	svalue	The yield stress of the material. If this is a single number, the EP variable must be set in order to define a linear plastic relation. The second option is to define the yield stress as a range of strain/stress coordinate pairs. Example is [eps0,stress0,eps1,stress2,....,epsn,stressn]. Remember that the strain is effective plastic strain which is equal to zero at initial yield. When the Y1,2... parameters are used, this stress/strain curve is defined for a certain strain rate, defined in the corresponding vvalue
	fvalue	is the plastic modulus or tangent modulus in the plastic region. If a curve is defined for the yield stress, this parameter is not needed.
	vvalue	is the strain rate for which the Yx curve is defined. The V1 value defines the strain rate for Y1 and so on. The stress/strain curve for a zero velocity is defined in the ordinary YIELD_STRESS parameter.
	fstrain	The strain at which the material fractures. If an element reaches this strain, it will be removed from the simulation.
	nuvalue	The stress at which the material fractures. If an element reaches this stress, it will be removed from the simulation.
Example	epsteel E = 210 RHO = 0.0000078 NU = 0.3 YIELD_STRESS = 0.180 EP = 0.1 steel2 E = 210 RHO = 0.0000078 NU = 0.3 YIELD_STRESS = [0,0.180,0.3,0.220,2.0,0.250]	

	v_steel E = 210 RHO = 0.0000078 NU = 0.3 YIELD_STRESS = [0,0.180,0.3,0.220] V1 = 0.2 Y1 = [0,0.200,0.3,0.240]
See also	Elements, Materials, Elastic

Command	**Spring**	
Block	Materials	
Description	This is a dummy material which defines all the spring stiffnesses and damping for a spring element. It cannot be used together with any other element types. Stiffness and damping can be defined as a function or constant for all directions.	
Syntax	***name*** KX = *kxvalue* KY = *kyvalue* KZ = *kzvalue* KRX = *krxvalue* KRY = *kryvalue* KRZ = *krzvalue* CX = *cxvalue* CY = *cyvalue* CZ = *czvalue* CRX = *crxvalue* CRY = *cryvalue* CRZ = *crzvalue*	
Options	***name***	Name of the material. Must be unique.
	kxvalue	The stiffness along the local x−axis. Can be defined as a constant or a function of the local x−displacement. If a function is wanted, the syntax shuld be kx = [d0,K0,d1,K1,...,dN,KN]. By default, this stiffness is assumed 0.
	kyvalue	The stiffness along the local y−axis. Can be defined as a constant or a function of the local y−displacement. By default, this stiffness is assumed to be the same as for KX.
	kzvalue	The stiffness along the local z−axis. Can be defined as a constant or a function of the local z−displacement. By default, this stiffness is assumed to be the same as for KX.
	krxvalue	The stiffness around the local x−axis. Can be defined as a constant or a function of the local x−rotation. By default, this stiffness is assumed 0.
	kryvalue	The stiffness around the local y−axis. Can be defined as a constant or a function of the local y−rotation. By default, this stiffness is assumed to be the same as for KRX.
	krzvalue	The stiffness around the local z−axis. Can be defined as a constant or a function of the local z−rotation. By default, this stiffness is assumed to be the same as for KRX.
	cxvalue	The damping along the local x−axis. Can be defined as a constant or a function of the local x−displacement. If a function is wanted, the syntax shuld be cx = [d0,C0,d1,C1,...,dN,CN]. By default, this damping is assumed 0.
	cyvalue	The damping along the local y−axis. Can be defined as a constant or a function of the local y−displacement. By default, this damping is assumed to be the same as for CX.
	czvalue	The damping along the local z−axis. Can be defined as a constant or a function of the local z−displacement. By default, this damping is assumed to be the same as for CX.
	crxvalue	The damping around the local x−axis. Can be defined as a constant or a function of the local x−rotation. By default, this

		damping is assumed 0.
	cryvalue	The damping around the local y−axis. Can be defined as a constant or a function of the local y−rotation. By default, this damping is assumed to be the same as for CRX.
	crzvalue	The stiffness around the local z−axis. Can be defined as a constant or a function of the local z−rotation. By default, this damping is assumed to be the same as for CRX.
Example	attrib KX = 10 CX = [0,0,1,20,2,off,3,30,45,0]	
See also	Elements, Materials, Elastoplastic	

Command	**Trackers**	
Block	Trackers	
Description	The trackers are used to track result data from a solution. There are several different trackers, each specially tailored for different results.	
Syntax	**Trackers of Type *ttype***	
Options	*ttype*	Any type of tracker: Nodeforce, Sectionforce, etc
Example	Trackers of Type Nodeforce	
See also	Nodeforce, Sectionforce	

Command	**Nodeforce**	
Block	Trackers	
Description	This tracker reads the forces from one or several nodes and plots the result into a file. The file is currently readable by the GID pre/postprocessor but the tracker can also print in a different fileformat. This is controlled by the Trackwriter command. Target is a value set by the user. If this value is reached during simulation, a file will be written (with extension .target). This is useful when debugging new versions of Impact.	
Syntax	***nr* nodes = [*tnode***,tnode,...,tnode**] DIRECTION = *dir* FILENAME = *fname*** TARGET = [*ttime,timetol,tvalue,valuetol*]	
Options	*nr*	The tracker number. Must be a unique number in the model, i.e. another nodetracker cannot have the same number.
	tnode	The number of the node to track forces from.
	dir	The direction of the force to track. Can be either 'X', 'Y' or 'Z'. It is also possible to select components thereof by adding a − or +, i.e. 'X+' will plot the component acting in the positive X direction. If only X is used, the sum of the positive and negative component will be plotted.
	filename	The name of the file of which the nodetracker should write. Must be a unique name for each nodetracker.
	ttime	The time where the target is to be checked.
	ttol	The tolerance for the target time
	tvalue	The target value.
	valuetol	The tolerance for the target value
Example	1 nodes = [23] direction = x+ filename = nodeforce.trk	
See also	Trackwriter, Sectionforce, Trackers	

Command	**Nodemoment**	
Block	Trackers	
Description	This tracker reads the moments from one or several nodes and plots the result into a file. The file is currently readable by the GID pre/postprocessor but the tracker can also print in a different fileformat. This is controlled by the Trackwriter command. Target is a value set by the user. If this value is reached during simulation, a file will be written (with extension .target). This is useful when debugging new versions of Impact.	
Syntax	***nr* nodes = [*tnode***,tnode,...,tnode**] DIRECTION = *dir* FILENAME = *fname*** TARGET = [*ttime,timetol,tvalue,valuetol*]	
Options	*nr*	The tracker number. Must be a unique number in the model, i.e. another nodetracker cannot have the same number.
	tnode	The number of the node to track moments from.
	dir	The direction of the moment to track. Can be either 'X', 'Y' or 'Z'. It is also possible to select components thereof by adding a − or +, i.e. 'X+' will plot the component acting in the positive X rotation direction. If only X is used, the sum of the positive and negative component will be plotted.
	filename	The name of the file of which the nodetracker should write. Must be a unique name for each nodetracker.
	ttime	The time where the target is to be checked.
	ttol	The tolerance for the target time
	tvalue	The target value.
	valuetol	The tolerance for the target value
Example	1 nodes = [23] direction = x+ filename = nodemoment.trk	
See also	Trackwriter, Nodeforce, Sectionforce, Trackers	

Command	**NodeDisplacement**	
Block	Trackers	
Description	This tracker reads the displacement of a single node and plots the result into a file. The file is currently readable by the GID pre/postprocessor but the tracker can also print in a different fileformat. This is controlled by the Trackwriter command. Target is a value set by the user. If this value is reached during simulation, a file will be written (with extension .target). This is useful when debugging new versions of Impact.	
Syntax	***nr* node = [*tnode*] DIRECTION = *dir* FILENAME = *fname*** TARGET = [*ttime,timetol,tvalue,valuetol*]	
Options	*nr*	The tracker number. Must be a unique number in the model, i.e. another nodetracker cannot have the same number.
	tnode	The number of the node to track displacement from.
	dir	The direction of the displacement to track. Can be either 'X', 'Y' or 'Z'
	filename	The name of the file of which the nodetracker should write. Must be a unique name for each nodetracker.
	ttime	The time where the target is to be checked.
	ttol	The tolerance for the target time
	tvalue	The target value.
	valuetol	The tolerance for the target value
Example	1 node = [23] direction = z filename = nodedisp.trk	
See also	Trackwriter, Sectionforce, Trackers	

Command	**NodeAcceleration**	
Block	Trackers	
Description	This tracker reads the acceleration of a single node and plots the result into a file. The file is currently readable by the GID pre/postprocessor but the tracker can also print in a different fileformat. This is controlled by the Trackwriter command. Target is a value set by the user. If this value is reached during simulation, a file will be written (with extension .target). This is useful when debugging new versions of Impact.	
Syntax	***nr* node = [*tnode*] DIRECTION = *dir* FILENAME = *fname*** TARGET = [*ttime,timetol,tvalue,valuetol*]	
Options	*nr*	The tracker number. Must be a unique number in the model, i.e. another nodetracker cannot have the same number.
	tnode	The number of the node to track acceleration from.
	dir	The direction of the acceleration to track. Can be either 'X', 'Y' or 'Z'
	filename	The name of the file of which the nodetracker should write. Must be a unique name for each nodetracker.
	ttime	The time where the target is to be checked.
	ttol	The tolerance for the target time
	tvalue	The target value.
	valuetol	The tolerance for the target value
Example	1 node = [23] direction = z filename = nodeacc.trk	
See also	Trackwriter, Sectionforce, Trackers	

Command	**Sectionforce**	
Block	Trackers	
Description	This tracker collects the nodal forces from a range of nodes. The first three of the nodes is the basis of a plane of which a normal axis is calculated. The force from each node is calculated in this direction, summarised and then plotted. A minimum of three nodes must be specified. This tracker is suitable for measuring the load through a cross section of a member or a beam.	
Syntax	***nr* nodes = [*node1*,*node2*,*node3*,*nodeN*] direction = *dir* filename = *fname***	
Options	*nr*	The tracker number. Must be a unique number in the model, i.e. another sectionforcetracker cannot have the same number.
	node1,2,..	The number of the first node etc. The first three nodes are mandatory. The rest are optional.
	dir	The direction of the forces to be collected. Can only be equal to "negative". When set, all the forces acting in the opposite direction to the section normal will be summed. If this parameter is not specified at all, the forces acting in the same direction as the section normal will be summed (default).
	filename	The filename of the result file of which the tracker should print the results. Must be unique for each tracker.
Example	1 nodes = [23,24,12,34,15] filename = sectionforce_1.trk	
See also	Nodeforce, Trackwriter, Trackers	

Command	**Energy**
Block	Trackers
Description	This tracker reads the energy from the model and plots it. There are several different energy types that can be plotted, but only one per tracker
Syntax	***nr* TYPE = *ttype* FILENAME = *fname*** TARGET = [*ttime,timetol,tvalue,valuetol*]
Options	*nr* — The tracker number. Must be a unique number in the model, i.e. another nodetracker cannot have the same number. *ttype* — The type of energy to plot. Can be one of • contact − for contact energy. • external − for external applied energy. • internal − for internal absorbed energy. • hourglass − for hourglass energy used to stabilize some elements. *filename* — The name of the file of which the energytracker should write. Must be a unique name for each energytracker. *ttime* — The time where the target is to be checked. *ttol* — The tolerance for the target time *tvalue* — The target value. *valuetol* — The tolerance for the target value
Example	1 type = external filename = energy_external.trk
See also	Trackwriter, Sectionforce, Trackers

Command	**NodeDistance**	
Block	Trackers	
Description	This tracker calculates the distance between two nodes and plots it as a function of time into a selected file. The distance is the shortest space distance.	
Syntax	***nr*** **node = [*node1*,*node2*] FILENAME = *fname*** TARGET = [*ttime,timetol,tvalue,valuetol*]	
Options	*nr*	The tracker number. Must be a unique number in the model, i.e. another NodeDistance tracker cannot have the same number.
	node1,node2	The number of the nodes to track distance between.
	filename	The name of the file of which the tracker should write. Must be a unique name for each tracker.
	ttime	The time where the target is to be checked.
	ttol	The tolerance for the target time
	tvalue	The target value.
	valuetol	The tolerance for the target value
Example	1 node = [23,15] filename = nodedist.trk	
See also	Trackwriter, Sectionforce, Trackers	

Command	**RodForce**	
Block	Trackers	
Description	This tracker reads the local force from a given Rod_2 element. The force is then plotted into a file as a function of time. Note that the force is always local and not plotted in any global direction.	
Syntax	***nr*** **element = [*telem*] FILENAME = *fname*** TARGET = [*ttime,timetol,tvalue,valuetol*]	
Options	*nr*	The tracker number. Must be a unique number in the model, i.e. another RodForce tracker cannot have the same number.
	telem	The number of the Rod_2 element to track force from.
	filename	The name of the file of which the tracker should write. Must be a unique name for each tracker.
	ttime	The time where the target is to be checked.
	ttol	The tolerance for the target time
	tvalue	The target value.
	valuetol	The tolerance for the target value
Example	1 element = [23] filename = rodforce.trk	
See also	Trackwriter, Sectionforce, Trackers	

Command	BeamSpring	
Block	Trackers	
Description	This tracker reads the local force from a given Beam_Spring_2 element. The force or moment is then plotted into a file as a function of time. Note that the force or moment is always local and not plotted in any global direction.	
Syntax	***nr* element = [*telem*] FILENAME = *fname*** COMPONENT = *comp* TARGET = [*ttime,timetol,tvalue,valuetol*]	
Options	*nr*	The tracker number. Must be a unique number in the model, i.e. another BeamSpring tracker cannot have the same number.
	telem	The number of the Beam_Spring_2 element to track force or moment from.
	filename	The name of the file of which the tracker should write. Must be a unique name for each tracker.
	comp	Sets which component to track. Can be one of: FX, FY, FZ, MX, MY, MZ. If nothing is set here, the default is to track the FX component.
	ttime	The time where the target is to be checked.
	ttol	The tolerance for the target time
	tvalue	The target value.
	valuetol	The tolerance for the target value
Example	1 element = [23] filename = beamspring_mz.trk component = mz	
See also	Trackwriter, Sectionforce, Trackers, Beam_Spring_2	

Command	Controls
Block	Controls
Description	The control block is initiated with the word **control** on a single line. There can only be one control block in any give indata file. All commands designed to control the solution process is defined here. One command with it's associated parameters is defined on each line.
Syntax	**Controls**
Options	
Example	Controls
See also	Run, Print

Command	**Run**
Block	Controls
Description	This command controls the solution process. The starting time and the end time are mandatory while control of the step size is optional. if this is left blank, impact will choose the most optimal step size for each step during the solution process.
Syntax	**Run from *svalue* to *evalue*** step *stpvalue*
Options	***svalue*** Start time for the solution ***evalue*** End time for the solution. *stpvalue* Stepsize for the solution. Specifying this value disables autostep.
Example	Run from 0.0 to 1.2 step 0.0001
See also	Controls, Print

Command	**Print**
Block	Controls
Description	This command controls the print process during the solution. The command can also be repeated with the tracker word if the printing interval is to be set specifically for the trackers. Impact will print the results with the interval specified. Depending on how the element perform it's internal calculations, the output may be local or global stresses and strains. Displacements of nodes are also printed so that mesh deformation can be followed. If no specific interval will be set for the trackers, they will print at the same time as the general printing occurs.
Syntax	**print** tracker **every *value* step**
Options	***value*** Step time for printing.
Example	Print every 0.01 step
See also	Controls, Run

Command	For	
Block	Controls	
Description	This command selects which Writer and Trackwriter to use. This selection determines the output format for the result files and thereby the selection of postprocessor. The default is that both the resultfiles and tracker files are printed in GID format. This command is entirely optional and is often left out.	
Syntax	For *writertype* use *selected_type*	
Options	*writertype*	Choice of writer type to specify. Can be either Writer or TrackWriter.
	selected_type	The selected type. For writers this can currently only be GIDWriter. Future extensions include Dynawriter and Radiosswriter For Trackwriters this can currently only be GIDTrackWriter. Future extensions include DynaTrackwriter and RadiossTrackwriter
Example	For Writer use GIDWriter	
See also	Trackers, Elements	

Other Indata and Outdata Formats

Impact is designed to handle other indata and outdata formats. At the time of writing, there are no additional formats supported but the process of extending Impact is documented in the programmers manual for those who would like to.