Introduction

1 Introduction

GiDPost 2.7

gidpost is a set of functions (library) for writing postprocess results for GiD in ASCII or binary compressed format.

This description asumes that the readear is familiar with the postprocess terminology.

For futher details please check the online help available in GiD (Postprocess data files chapter).

The library was implemented taking into account two of the must widely used development environments: C/C++ and FORTRAN.

Here we are going to describe how to compile and use the library. At the end the reference of the library functions can be found.

1.1 Changes

From version 2.6 to 2.7

- Added support for MeshGroups (BeginMeshGroup(), End..., BeginOnMeshGroup(), End...) in HDF5 as used in GiD, useful for dynamic meshes, refinement, multi-stage simulation, etc.
- All non implemented functions for HDF5 now return -1, i.e. all GiD_fXXX() functions.

From version 2.5 to 2.6

- Added support for ComplexMatrix Result types.
- More exhaustive example and some corrections.

From version 2.4 to 2.5

- Default format to print real numbers changed from "%g" to "%.9g" increasing the amount of significant digits to be printed in ASCII.
- New funcion GiD_PostSetFormatReal to allow change the default format of real numbers.

From version 2.3 to 2.4

- Allow write OnNurbsSurface results for isogeometrical analysis

Changes

From version 2.1 to 2.3

- Allow use HD5 from FORTRAN interface
- Fixed bug with mesh group

From version 2.0 to 2.1

- Allow write complex scalar and complex vector results
- Write units of mesh or results
- Enhanced FORTRAN 77 and 90 interface to be more compatible.

From version 1.7.2 to 2.0

- New set of functions "GiD_fxxx" that allow specify the file to write, in case of have multiple files of mesh or results.
- Library rewritten in C avoiding C++ to be more compatible linking with other languages.
- Optional define of HDF5 to enable write postprocess files using the HDF5 library

Old changes

2008/09/30

* Updated cfortran to release 4.4

2008/02/25

* changing to release 1.8

* M build/Jamroot, source/gidpost.cpp source/gidpostInt.h: removed C++ dependencies.

2008/01/29

* source/gidpost.cpp:

- GiD_OpenPostMeshFile should not write a header. Thanks to Janosch Stascheit <janosch.stascheit@rub.de>

- invalid access in GiD_BeginMeshGroup: should ensure that a valid mesh file object is available so etMeshFile() should be called

before writting. Thanks to Janosch Stascheit <janosch.stascheit@rub.de> for providing the fix.

2008/01/18

* source/gidpost.cpp: GiD_BeginOnMeshGroup should be "OnGroup \"%s\"" instead of "Group \"%s\"". Thanks to Janosch Stascheit

<janosch.stascheit@rub.de>

Changes

2007/06/13

* source/gidpost.cpp: GiD_WritePlainDefMatrix can be written in binary files.

2007/06/11

* GiD_Sphere and GiD_Circle

2007/06/11

* source/gidpost.cc: Write2DMatrix is now available on binary format, Z component is set to 0.

2007/01/23:

* source/gidpostfor.c: bug reported by a user: GiD_Begin3DMatResult must be ncomp = SetupComponents(6, not 4 Factoring fortran declaration in gidpostfor.h

Defining symbols for both gcc and icc.

2006/06/25

* doc/*:

* sources/gidpost.cpp,gidpost.h: included pyramid element.

2006/06/25

* tag 1.7: rel_1_7

* examples/testpost.c: fixed a bug, AsciiZipped must has the mesh

file in ascii format.

2006/05/25

* gidpost.h: dllexport.

2006/05/18

* all: GiD_BeginMeshColor, GiD_BeginMeshGroup, GiD_EndMeshGroup,

GiD_BeginOnMeshGroup, GiD_EndOnMeshGroup

2005/10/24

* doc:

* gidpost.h : documented GiD_ResultDescriptionDim

2005/10/21

* gidpost* : added support to new type specification for result

groups. The type could be Type:dim, where Type is one valid result

type and dim is a number giving a valid dimension. Updated fortran

Changes

interface for recently added functions.

2005/10/07

* all: new version 1.60. "Result Groups" can now be written. The macro USE_CONST can be used to enable (if defined) or desable (if not defined) the use of const char* arguments.

2005/09/22

* all: ha sido un error cambiar de 'char *' a 'const char *'

2005/09/22

* gidpost.cpp: flag_begin_values = 0; must be set in GiD_BeginResultGroup.

2005/09/21

* gidpost.h:
* gidpost.cpp: values_location_{min,max} not needed, CBufferValue
* gidpostInt.cpp: now controls the types of results written within
* gidpostInt.h: a ResultGroup.

2005/09/16

* gidpost.cpp: in GiD_WriteVectorModule, when writing if we are inside a ResultGroup block the module value is ignored.
* gidpostInt.cpp: in CBufferValues::WriteValues, buffer overflow is checked after checking for change in location.

2005/09/15

* gidpost.cpp:
* gidpostInt.cpp:
* gidpostInt.h: fixed a bug when writing results in a "Result Group", vectors can be 3 or 4 component long.

* examples/testpost.c : bug when passing location id in result group sample code.

2005/06/27

* source/gidpost.{h,cpp} including Prism element
* doc/gidpost.{html,subst} including Prism element documentation.

Changes

* all: change to version 1.52

2005/05/09

* all: constification, version change from 1.5 to 1.51

2005/01/04

* source/gidpostInt.cpp: fixed a bug when writing 3D vectors.

2005/01/07

* source/gidpost.cpp added const char and a filter to remove
double quotes from names which can cause problems within gid.

2003/07/29

* doc/gidpost.html : commented the GiD_ResultUnit
* source/gidpost.h : interface because it is not
* source/gidpostfor.c: supported yet inside GiD.

2003/07/28

* doc/gidpost.html: updated documentation for release 1.5
* binary/gidpost.lib: binary release for windows

2003/07/15

* gidpost.h:
* gidpost.cpp:
* gidpostfor.c: new function 'GiD_WriteCoordinates2D' (to write
coordinates in 2D but only in ASCII format, the function can be
used in binary but the library provide the 'z' cordinate a zero.

2003/07/15

* gidpost.h:
* gidpost.cpp: removed 'char * UnitName' argument , new functions:
GiD_BeginResultHeader, GiD_ResultRange, GiD_ResultComponents,
GiD_ResultUnit, GiD_BeginResultGroup, GiD_ResultDescription,
GiD_ResultValues, GiD_FlushPostFile. Validation in debug mode.

* gidpostInt.h:
* gidpostInt.cpp: new member
CPostFile::WriteValues(int id, int n, double * buffer), new class

Changes

CBufferValues to write the values in a result group, validation in debug mode.

* gidpostfor.c: updated fortran interface for the new functionality.

* testpost.c:

* testpostfor.f: added test code for the new functionality.

2 Compiling

All platforms:

Required third part libraries:

- **zlib:** it is necessary previously obtain or compile the zlib compression library (www.zlib.net)
- **HDF5**: this library is optional www.hdfgroup.org. Required only to write postprocess files with this format (files opened with GiD_PostHDF5=3 flag)

To install the required packages in Linux just do the following **as root**:

Ubuntu (Debian and the like):

$ **apt-get install zlib1g-dev libhdf5-serial-dev**

Scientific Linux (Redhat and the like):

$ **yum install zlib-devel.x86_64 hdf5-devel.x86_64**

Note:

Bear in mind that there are several flavours of hdf5 development packages and that you should choose the right for you.

For instance, in Ubuntu these are the HDF5 development packages available:

libhdf5-mpi-dev

libhdf5-mpich-dev

libhdf5-openmpi-dev

libhdf5-serial-dev

And in Scientific Linux the HDF5 development packages available:

hdf5-devel.x86_64

hdf5-mpich-devel.x86_64

hdf5-openmpi.x86_64

Compiling

--> Remeber to change the suffix x86_64 to i686 for the 32bit versions in Scientific Linux.

Required third part tool:

- **CMake:** The CMake tool (www.cmake.org) is used to have a cross-platform build system. Must download and install this program to compile gidpost.lib (gidpost.a on Linux).

To use the graphical interface of cmake use **cmake-gui** or **ccmake** .

Note: For MS Visual Studio a project obtained from CMake is also included in the distribution, to avoid the requirement of install CMake.

Note: Users that don't want to use CMake could also manually create off course its own makefile for Unix/Linux or its own compiler project.

All the source code to generate the library is included, it can be compiled in other platforms.

The third party code, "cfortran.h", is also provided as a link between C and FORTRAN. It has its own distribution policy. Please, read the file "cfortran.doc" about the licence terms of this code.

Precompiled version:

The 'binaries' folder store precompiled release versions of the library for Windows and Linux (x32 and x64 platforms), to avoid the requirement of compile them.

CMake use: (to compile the gidpost library)

Command line:

```
$ <uncompress the package>
$ mkdir build
$ cd build
$ cmake -DHDF5=true -DGIDPOST_USE_SYSTEM_ZLIB=true <path-to-source-files>
$ make
```

For the graphical interface of cmake use **cmake-gui** or **ccmake** .

e.g. for Windows, to compile with MicroSoft Visual Studio 2005, fill in

Where is the source code: C:/gid project/gidpost

Where to build the binaries: C:/gid project/gidpost/win/vs2005

and press "Configure"

select generate a project of "Visual Studio 8 2005", "use default native compilers"

then some errors appear (lines in red)

select GIDPOST_USE_SYSTEM_ZLIB and press "Configure"

then must specify the path to the zlib include and the zlib library

(zlib must be compiled previously using its own mechanism, or get it precompiled)

To add HDF5 extra features the HDF5 library must be compiled using its own mechanism, or get it precompiled

and then set in Cmake the values pointing to its include directory and library

press Configure, and then in 'Ungrouped Entries' appear HDF5, select it to be used.

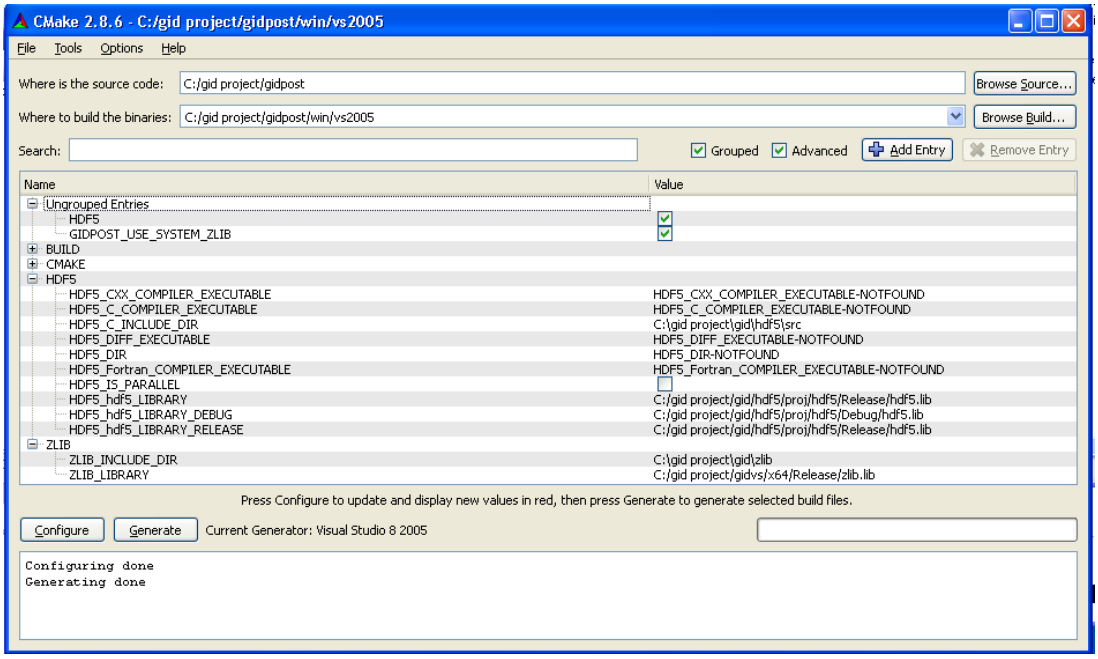
and press Configure once more

Finally in the CMake tree, for Visual Studio is preferred instead /MD the flag /MT (to avoid dependencies of MSCVRTxxx.dll)

and instead /MDd /MTd for the debug version

Press Configure, and then "Generate", to create the VS project, then the project is created

C:\gid project\gidpost\win\vs2005\gidpost.snl and can be opened with Visual Studio



3 Using the library

- **C / C++ language:**

Include the file header file gidpost.h to use the libray gidpost

```
#include "gidpost.h"
```

Using the library

Small examples:

- testpost.c, is provided to show the use of the library from C/C++.
- testpost_fd.c that use the *f functions where the file handler is explicitly specified (to allow multiple files)
- IGA_test_1.c, to show the use of OnNurbsSurface isogeometric results

- **FORTRAN language:**

A small example, called testpostfor.f, is provided to show the use of the library with FORTRAN 77

the same case is implemented with FORTRAN 90 in the example testpost.f90, using the gidpost.F90 interface module

Note: The gidpost fortran 90 module use the new ISO_C_BINDING intrinsic module defined in the Fortran 2003 standard.

(Tested in intel fortran >= 10.1, Sun/Oracle studio >= 12.1, and GNU gfortran >= XX)

Link libraries: gidpost.lib zlib.lib hdf5.lib (Linker - Input - Additional Dependencies)

and remember set the path to these libraries (e.g. Linker - General - Additional Library Directories - ..\..\binaries\win\x32)

Note: the hdf5.lib library is only required to write files with HDF5 format.

Note: For Linux platforms the extension of libraries is .a instead of .lib

4 Functions references

4.1 Library use

Before call other library functions the GiD_PostInit funcion must be called (and GiD_PostDone when finishing its use)

There are two collection of functions: multi-file and single-file

- GiD_f* functions with the 'f' letter, and an extra attribute for the file handler (new interface from library version 2.0)
- GiD_* functions without 'f' letter. There is an implicit single file (old interface)

The HDF5 format (GiD_fOpenPostResultFile -pag. 18-) is only available with old single-file style, the multiple-file interface will be implemented in future versions.

4.1.1 GiD_PostInit

This procedure must be invoked once before use other commands

It initialize some internal structures. Must be called from the main thread of the program.

GiD_PostDone

4.1.2 GiD_PostDone

This procedure must be invoked once at the end.

It release the internal structures initialized by GiD_PostInit -pag. 9-. Must be called from the main thread of the program.

4.1.3 GiD_PostSetFormatReal

Functions references>Mesh file functions>GiD_fOpenPostMeshFile

int GiD_PostSetFormatReal(GP_CONST char * format)

Description: overwrite the default format to print real numbers in ASCII strings (by default, this is "%.9g")

Parameters:

char* format

a valid C/C++ format to print a real number

Example:

C/C++

GiD_PostSetFormatReal("%15.5f");

4.2 Mesh file functions

The mesh file is not necessary if the postprocess mesh is the same used in GiD preprocess.

4.2.1 GiD_fOpenPostMeshFile

GiD_FILE GiD_fOpenPostMeshFile(const char * FileName,GiD_PostMode Mode);

Description: Open a new post mesh file, and return its file handler.

Parameters:

char* FileName

name of the mesh file (*.post.msh)

GiD_PostMode Mode

GiD_PostAscii=0 for ascii output

GiD_PostAsciiZipped=1 for compressed ascii output

GiD_PostBinary=2 for compressed binary output

GiD_PostHDF5=3 for HDF5 output (the library must be compiled with HDF5 support)

GiD_fOpenPostMeshFile

GiD_FILE and unsigned int that identify the channel.

Example:

C/C++

fd=GiD_fOpenPostMeshFile("testpost.post.msh", GiD_PostAscii);

FORTRAN

fd=GID_FOPENPOSTMESHFILE('testpost.post.msh',0)

4.2.2 GiD_fClosePostMeshFile

int GiD_fClosePostMeshFile(GiD_FILE fd);

Description: Close the post mesh file

Parameters:

GiD_FILE fd, the handler returned by GiD_fOpenPostMeshFile

Example:

C/C++

GiD_fClosePostMeshFile(fd);

FORTRAN

CALL GID_FCLOSEPOSTMESHFILE(fd)

4.2.3 GiD_fBeginMeshGroup

int GiD_fBeginMeshGroup(GiD_FILE fd, const char* Name);

Description: This function open a group of mesh. This enable specifying multiples meshes withing the group.

Parameters:

GiD_FILE fd the file descriptor

char* Name

Name of the group. This name can be used later when givin the set of results that apply to this group, see GiD_fBeginOnMeshGroup.

Example:

C/C++

GiD_fBeginMeshGroup

GiD_fBeginMeshGroup(fd,"steps 1, 2, 3 and 4");

FORTRAN

CALL GID_FBEGINMESHGROUP(fd,"steps 1, 2, 3 and 4")

4.2.4 GiD_fEndMeshGroup

int GiD_fEndMeshGroup(GiD_FILE fd);

Description: This function close the previously opened group of mesh. See GiD_fBeginMeshGroup.

Parameters:

GiD_FILE fd the file descriptor

Example:

C/C++

GiD_fEndMeshGroup(fd);

FORTRAN

CALL GID_FENDMESHGROUP(fd)

4.2.5 GiD_fBeginMesh

int GiD_fBeginMesh(GiD_FILE fd,const char* MeshName,GiD_Dimension Dim,GiD_ElementType EType,int NNode);

int GiD_fBeginMeshColor(GiD_FILE fd, const char* MeshName,GiD_Dimension Dim, GiD_ElementType EType,int NNode,double Red, double Green, double Blue);

Description: Begin a new mesh. After that you should write the nodes and the elements. When writing the elements it is assumed a connectivity of size given in the paramenter NNode. The second function allows to specify a color for the mesh where each component take values on the interval [0,1].

Parameters:

GiD_FILE fd the file descriptor

char* MeshName

Name of the mesh

GiD_Dimension Dim

GiD_fBeginMesh

GiD_2D=2 for a 2D mesh (is assumed coordinates z=0)

GiD_3D=3 for a 3D mesh

GiD_ElementType EType

GiD_Point=1 for a 1 noded-element

GiD_Linear=2 for a line element

(the number of nodes can be 2 for a linear case or 3 for the quadratic case)

GiD_Triangle=3 for a triangle element

(the number of nodes can be 3 for a linear case or 6 for the quadratic case)

GiD_Quadrilateral=4 for a quadrilateral element

(the number of nodes can be 4 for a linear case and 8 or 9 for the quadratic case)

GiD_Tetrahedra=5 for a tetrahedral element

(the number of nodes can be 4 for a linear case or 10 for the quadratic case)

GiD_Hexahedra=6 for a hexahedral element

(the number of nodes can be 8 for a linear case and 20 or 27 for thequadratic case)

GiD_Prism=7 for a Prism element

(the number of nodes can be 6 for the linear case or 15 for the quadratic case)

GiD_Pyramid=8 for a Pyramid element

(the number of nodes can be 5 for the linear case or 13 for the quadratic case)

GiD_Sphere=9 for a sphere element with 1 node and a radius

GiD_Circle=10 for circle element, with 1 node, radius and a plane normal

int NNode

Number of nodes of this type of element. The element type an nnods must be constant for all mesh elements, but it is valid to define more that one mesh.

Example:

C/C++

```
GiD_fBeginMesh(fd,"TestMsh",GiD_2D,GiD_Triangle,3);
```

FORTRAN

```
CALL GID_FBEGINMESH(fd,'quadmesh',2,4,4)
```

4.2.6 GiD_fMeshUnit

int GiD_fMeshUnit(GiD_FILE fd, const char* UnitName);

Description: Define the unit string associated to the mesh coordinates

GiD_fMeshUnit

Parameters:

GiD_FILE fd the file descriptor

char* UnitName ,the string of the unit (length magnitude)

Example:

C/C++

GiD_fBeginMesh(fd,"TestMsh",GiD_2D,GiD_Triangle,3);

GiD_fMeshUnit(fd,"m");

FORTRAN

CHARACTER*10 UNITNAME

UNITNAME = 'm'

GID_FMESHUNIT(fd,UNITNAME)

4.2.7 GiD_fEndMesh

int GiD_fEndMesh(GiD_FILE fd);

Description: End the current mesh.

Parameters:

GiD_FILE fd the file descriptor

Example:

C/C++

GiD_FEndMesh(fd);

FORTRAN

CALL GID_ENDMESH(fd)

4.2.8 GiD_fBeginCoordinates

int GiD_fBeginCoordinates(GiD_FILE fd);

Description: Start a coordinate block in the current mesh. All nodes

coordinates must be written between a to this function and GiD_fEndCoordinates().

Parameters:

GiD_FILE fd the file descriptor

Example:

GiD_fBeginCoordinates

C/C++

GiD_fBeginCoordinates(fd);

FORTRAN

CALL GID_FBEGINCOORDINATES(fd)

4.2.9 GiD_fEndCoordinates

int GiD_fEndCoordinates(GiD_FILE fd);

Description: Close the current coordinate block.

Parameters:

GiD_FILE fd the file descriptor

Example:

C/C++

GiD_fEndCoordinates(fd);

FORTRAN

CALL GID_FENDCOORDINATES(fd)

4.2.10 GiD_fWriteCoordinates

int GiD_fWriteCoordinates(GiD_FILE fd, int id, double x, double y, double z);

int GiD_fWriteCoordinates2D(GiD_FILE fd, int id, double x, double y);

Description: Write a coordinate member at the current Coordinates Block.

If the mesh dimension is 2D then you can use GiD_fWriteCoordinates2D

Parameters:

GiD_FILE fd the file descriptor

int id

Node number identifier (starting from 1, is recommended to avoid jumps in the numeration)

double x,double y,double z

Cartesian coordinates

Example:

C/C++

GiD_fWriteCoordinates

int id=1;

double x=3.5,y=1.5e-2,z=0;

GiD_fWriteCoordinates(fd,id,x,y,z);

FORTRAN

REAL*8 rx, ry

INTEGER*4 idx

idx=1

rx=3.5

ry=-4.67

CALL GID_FWRITECOORDINATES(fd,idx,rx,ry,0.0)

4.2.11 GiD_fBeginElements

int GiD_fBeginElements(GiD_FILE fd);

Description: Start a elements block in the current mesh.

Parameters:

GiD_FILE fd the file descriptor

Example:

C/C++

GiD_fBeginElements(fd);

FORTRAN

CALL GID_FBEGINELEMENTS(fd)

4.2.12 GiD_fEndElements

int GiD_fEndElements(fd);

Description: Close the current elements block.

Parameters:

GiD_FILE fd the file descriptor

Example:

C/C++

GiD_fEndElements(fd);

FORTRAN

GiD_fEndElements

CALL GID_FENDELEMENTS(fd)

4.2.13 GiD_fWriteElement

int GiD_fWriteElement(GiD_FILE fd, int id, int nid[]);

int GiD_fWriteSphere(GiD_FILE fd, int id, int nid, double r);

int GiD_fWriteCircle(GiD_FILE fd, int id, int nid, double r,double nx, double ny, double nz);

Description: Write an element member at the current Elements Block.

Parameters:

GiD_FILE fd the file descriptor

int id Element number identifier

int nid[] connectivities of the element, the vector dimension must be equal to the NNode parameter given in the previous call to GiD_fBeginMesh

r the radius (sphere or circle element only)

nx,ny,nz unitary normal of the circle plane (circle element only)

Example:

C/C++

int id=2;

int nid[3];

nid[0]=4; nid[1]=7; nid[2]=3;

GiD_fWriteElementMat(fd,id,nid);

FORTRAN

INTEGER *4 nid(1:3)

INTEGER*4 idx

idx=2

nid(1)=4

nid(2)=7

nid(3)=3

CALL GID_FWRITEELEMENT(fd,idx,nid)

4.2.14 GiD_fWriteElementMat

int GiD_fWriteElementMat(GiD_FILE fd, int id, int nid[]);

int GiD_fWriteSphereMat(GiD_FILE fd, int id, int nid, double r, int mat);

int GiD_fWriteCircleMat(GiD_FILE fd, int id, int nid, double r, double nx, double ny, double nz, int

mat);

Description: Similar to GiD_WriteElement, but the last aditional integer on nid correspond to a material number.

4.3 Results file functions

4.3.1 GiD_fOpenPostResultFile

GiD_FILE GiD_fOpenPostResultFile(const char * FileName, GiD_PostMode Mode);

Description: Open a new post result file and return its file handler, to allow subsequent call to functions write the information into the desired file. If there is no mesh file opened then the output of the mesh is writen into this file also.

Parameters:

char* FileName

name of the mesh file (*.post.res)

GiD_PostMode Mode

GiD_PostAscii=0 for ascii output

GiD_PostAsciiZipped=1 for compressed ascii output

GiD_PostBinary=2 for compressed binary output

GiD_PostHDF5=3 for HDF5 output (the library must be compiled with HDF5 support)

Note: In binary output, the mesh must be inside the same file as the results, you should not call GiD_fOpenPostMeshFile and GiD_fClosePostMeshFile in that case.

Example:

C/C++

```
fd=GiD_fOpenPostResultFile( "testpost.post.bin",GiD_PostBinary);
```

FORTRAN

```
fd=GID_OPENPOSTRESULTFILE('testfortran.post.res',2)
```

4.3.2 GiD_fClosePostResultFile

int GiD_fClosePostResultFile(GiD_FILE fd);

Description: Close the post results file

Parameters:

GiD_fClosePostResultFile

GiD_FILE fd the file descriptor

Example:

C/C++

```
GiD_fClosePostResultFile(fd);
```

FORTRAN

```
CALL GID_CLOSEPOSTRESULTFILE(fd)
```

4.3.3 GiD_fBeginGaussPoint

```
int GiD_fBeginGaussPoint(GiD_FILE fd, const char * name, GiD_ElementType EType,
                         const char * MeshName, int GP_number, int NodesIncluded, int InternalCoord);
```

Description: Begin Gauss Points definition. The gauss point definition should have a name wich may be referenced in futher results blocks. The gauss points could be internal (InternalCoord=1) or given (InternalCoord=0). If the gauss points are given then the list of its natural coordinates should be written using the function GiD_fWriteGaussPoint2D or GiD_fWriteGaussPoint3D depending on the dimension of the element type.

Parameters:

GiD_FILE fd the file descriptor

char* name

Name to reference this gauss points definition

GiD_ElementType EType

GiD_Point=1 for a 1 noded-element

GiD_Linear=2 for a line element

GiD_Triangle=3 for a triangle element

GiD_Quadrilateral=4 for a quadrilateral element

GiD_Tetrahedra=5 for a tetrahedral element

GiD_Hexahedra=6 for a hexahedral element

GiD_Prism=7 for a prism element

GiD_Pyramid=8 for a pyramid element

GiD_Sphere=9 for sphere element

GiD_Circle=10 for circle element

char* MeshName

GiD_fBeginGaussPoint

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.

int GP_number

number of gauss points per element. The GiD internal accepted number should be:

1, 3, 6 for Triangles;

1, 4, 9 for quadrilaterals;

1, 4 for Tetrahedras;

1, 8, 27 for hexahedras;

1 or 6 for prism;

1 or 5 for pyramid and

1, ... n points equally spaced over lines.

int NodesIncluded

Can be 0 for nodes not included or 1 for included. Only used for gauss points on Linear elements which indicate whether the end nodes of the Linear element are included in the number of gauss points per element count or not.

int InternalCoord

Can be 0 for given coordinates or 1 for internal GiD gauss points location.

Example:

C/C++

```
GiD_fBeginGaussPoint(fd,"GPtria",GiD_Triangle,NULL,1,0,1);
```

FORTRAN

```
CHARACTER*4 NULL
NULL = CHAR(0)//CHAR(0)//CHAR(0)//CHAR(0)
CALL GID_FBEGINGAUSSPOINT(fd,'GPtria',3,NULL,1,0,1);
```

4.3.4 GiD_fEndGaussPoint

int GiD_fEndGaussPoint(GiD_FILE fd);

Description: End current Gauss Points definition

GiD_fEndGaussPoint

Parameters:

GiD_FILE fd the file descriptor

Example:

C/C++

GiD_fEndGaussPoint(fd);

FORTRAN

CALL GID_FENDGAUSSPOINT(fd)

4.3.5 GiD_fWriteGaussPoint

int GiD_fWriteGaussPoint3D(GiD_FILE fd, double x, double y, double z);

int GiD_fWriteGaussPoint2D(GiD_FILE fd, double x, double y);

Description: Write internal gauss point local coordinates (only required if InternalCoord=0)

Parameters:

GiD_FILE fd the file descriptor

double x,double y,double z

Cartesian gauss points local coordinates

Example:

C/C++

double x=3.5;

double y=-7;

GiD_fWriteGaussPoint2D(fd,double x,double y);

FORTRAN

REAL*8 x, y

rx=3.5

ry=-7

CALL GID_FWRITEGAUSSPOINT2D(fd,x,y)

4.3.6 GiD_fBeginRangeTable

int GiD_fBeginRangeTable(GiD_FILE fd, const char * name);

Description: Begin a Range Table definition. With a range table you can

group the result values into intervals and label each interval with a name.

GiD_fBeginRangeTable

Inside GiD this can be visualized with a contour range.

Parameters:

GiD_FILE fd the file descriptor

char* name

name identifier

Example:

C/C++

GiD_fBeginRangeTable(fd,"table1");

FORTRAN

CALL GID_FBEGINRANGETABLE(fd,'table1')

4.3.7 GiD_fEndRangeTable

int GiD_fEndRangeTable(GiD_FILE fd);

Description: End a Range Table definition.

Parameters:

GiD_FILE fd the file descriptor

Example:

C/C++

GiD_fEndRangeTable(fd);

FORTRAN

CALL GID_FENDRANGETABLE(fd)

4.3.8 GiD_fWriteRange

int GiD_fWriteRange(GiD_FILE fd, double min, double max, const char * name);

int GiD_fWriteMinRange(GiD_FILE fd, double max, const char * name);

int GiD_fWriteMaxRange(GiD_FILE fd, double min, const char * name);

Description: Write Range functions. Must be between GiD_fBeginRangeTable and GiD_fEndRangeTable.

fWriteMinRange : write a range with an implicit minimum value, the minimum absolute in the result set.

GiD_fWriteRange

fWriteRange : write an explicit range.

fWritemaxRange: write a range with an implicit maximum value,

the maximum absolute in the result set.

Parameters:

GiD_FILE fd the file descriptor

double min,double max

Values to define a interval

char* name

string asociated to be showed for this interval

Example:

C/C++

```
GiD_fWriteRange(fd,0.0,100.0,"Normal");
```

FORTRAN

```
CALL GID_FWRITERANGE(fd,0.0,100.0,'Normal')
```

4.3.9 GiD_fBeginResultHeader

```
int GiD_fBeginResultHeader(GiD_FILE fd, const char * Result,
                           const char * Analysis,
                           double step,
                           GiD_ResultType Type, GiD_ResultLocation Where,
                           const char * GaussPointsName);
```

Description:

Begin Result Block. This function open a result block. Only the result, analysis, location and location name are provided in the function call.

The other result attributes as range table or components names are provided in a separated function calls. See GiD_fResultComponents -pag. 27-, GiD_fResultRange -pag. 27- and GiD_fResultUnit -pag. 29-.

Before set the results GiD_fResultValues -pag. 31- must be called

Parameters:

GiD_FILE fd the file descriptor

char* Result

a name for the Result, which will be used for menus.

GiD_fBeginResultHeader

char* Analysis

the name of the analysis of this Result, which will be used for menus.

double step

the value of the time step inside the analysis "analysis name".

(for multiple steps results)

GiD_ResultType Type

The type of defined result:

GiD_Scalar = 0,

GiD_Vector,

GiD_Matrix,

GiD_PlainDeformationMatrix,

GiD_MainMatrix,

GiD_LocalAxes,

GiD_ComplexScalar,

GiD_ComplexVector,

GiD_ComplexMatrix

GiD_ResultLocation Where

The location of the results

GiD_OnNodes=0

GiD_OnGaussPoints=1

char* GaussPointsName

If Where is GiD_OnGaussPoints a "location name" (predefined in GiD_BeginGaussPoint) should be entered.

Example:

C++:

```
GiD_fBeginResultHeader(fd,"Result","Static",1.0d0,GiD_Scalar,GiD_OnNodes,NULL);
GiD_fResultUnit(fd,'m/s');
GiD_fResultValues(fd)
for(int i=1;i<10;i++){
  GiD_fWriteScalar(fd,i,value[i])
}
GiD_fEndResult(fd)
```

4.3.10 GiD_fBeginResult

Note: these functions are deprecated because can't provide the result unit, to specify unit fBeginResultHeader must be used.

```
int GiD_fBeginResult(GiD_FILE fd, const char * Result,
                     const char * Analysis,
                     double step,
                     GiD_ResultType Type, GiD_ResultLocation Where,
                     const char * GaussPointsName,
                     const char * RangeTable,
                     int compc, const char * compv[]);

int GiD_fBeginScalarResult(GiD_FILE fd,char* Result,char* Analysis,double step,
      GiD_ResultLocation Where,
char* GaussPointsName,
char* RangeTable,char* Comp);

int GiD_fBeginVectorResult(GiD_FILE fd,char* Result,char* Analysis,double step,
       GiD_ResultLocation Where,
       char* GaussPointsName,char* RangeTable,
       char* Comp1,char* Comp2,char* Comp3,char* Comp4);

int GiD_fBegin2DMatResult(GiD_FILE fd,char* Result,char* Analysis,double step,
      GiD_ResultLocation Where,
      char* GaussPointsName,char* RangeTable,
      char* Comp1,char* Comp2,char* Comp3);

int GiD_fBegin3DMatResult(GiD_FILE fd,char* Result,char* Analysis,double step,
      GiD_ResultLocation Where,
      char* GaussPointsName,char* RangeTable,
      char* Comp1,char* Comp2,char* Comp3,
      char* Comp4,char* Comp5,char* Comp6);

int GiD_fBeginPDMMatResult(GiD_FILE fd,char* Result,char* Analysis,double step,
       GiD_ResultLocation Where,
       char* GaussPointsName,char* RangeTable,
```

```
      char* Comp1,char* Comp2,char* Comp3,
      char* Comp4);

int GiD_fBeginMainMatResult(GiD_FILE fd,char* Result,char* Analysis,double step,
       GiD_ResultLocation Where,
       char* GaussPointsName,char* RangeTable,
       char* Comp1,char* Comp2,char* Comp3,
       char* Comp4,char* Comp5,char* Comp6,
       char* Comp7,char* Comp8,char* Comp9,
       char* Comp10,char* Comp11,char* Comp12);

int GiD_fBeginLAResult(GiD_FILE fd,char* Result,char* Analysis,double step,
          GiD_ResultLocation Where, char* GaussPointsName,char* RangeTable,
char* Comp1,char* Comp2,char* Comp3);

int GiD_fBeginComplexScalarResult(GiD_FILE fd,char* Result,char* Analysis,double step,
      GiD_ResultLocation Where,
      char* GaussPointsName,char* RangeTable,
       GP_CONST char * Re, GP_CONST char * Im);

int   GiD_fBeginComplexVectorResult(GiD_FILE fd,  GP_CONST char *  Result,  GP_CONST char *
Analysis, double step,
    GiD_ResultLocation Where,
    GP_CONST char * GaussPointsName, GP_CONST char * RangeTable,
    GP_CONST char * Rex, GP_CONST char * Imx, GP_CONST char * Rey,
    GP_CONST char * Imy, GP_CONST char * Rez, GP_CONST char * Imz);
```

Description: Begin Result Block. This function open a result block.

These functions are deprecated because can't provide the result unit, to specify unit fBeginResultHeader must be used.

Parameters:

See GiD_fBeginResultHeader -pag. 23-

char* RangeTable

A valid Range table name or NULL

GiD_fBeginResult

int compc

The number of component names or 0

char* compv[]

array of 'compc' strings to be used as component names

Example:

C/C++

GiD_fBeginResult(fd,"Result","Static",1.0d0,GiD_Scalar,GiD_OnNodes,NULL,NULL,0,NULL);

FORTRAN

CHARACTER*4 NULL

NULL = CHAR(0)//CHAR(0)//CHAR(0)//CHAR(0)

CALL GID_FBEGINSCALARRESULT(fd,'Result','Analy.',1.0d0,0,NULL,NULL,NULL)

4.3.11 GiD_fResultRange

int GiD_fResultRange(GiD_FILE fd, const char * RangeTable);

Description:

Define the components names associated to the current result, either a single result block or the current result defined in a result group.

Parameters:

GiD_FILE fd the file descriptor

char * RangeTable --> name of the range table previously defined

Example:

C/C++

GiD_fResultRange(fd,"MyTable");

FORTRAN

GID_FRESULTRANGE(fd,'MyTable');

4.3.12 GiD_fResultComponents

int GiD_fResultComponents(GiD_FILE fd, int compc, const char * compv[]);

Description: Define the components names associated to the current result, either a single result block or the current result defined in a result group. In FORTRAN you should call a diferent function of each result type:

GiD_fResultComponents

GiD_fScalarComp(GiD_FILE fd, STRING Comp1) --> for scalar result

GiD_fVectorComp(GiD_FILE fd, STRING Comp1, STRING Comp2, STRING Comp3, STRING Comp4) --> for vector result type

GiD_f2DMatrixComp(GiD_FILE fd, STRING Sxx, STRING Syy, STRING Sxy) --> for matrix result type

GiD_f3DMatrixComp(GiD_FILE fd, STRING Sxx, STRING Syy, STRING Szz, STRING Sxy, STRING Syz, STRING Sxz) --> for matrix result type

GiD_fPDMComp(GiD_FILE fd, STRING Sxx, STRING Syy, STRING Sxy, STRING Szz); --> for plain deformation matrix result type

GiD_fMainMatrixComp(GiD_FILE fd, STRING Si, STRING Sii, STRING Siii, STRING Vix, STRING Viy, STRING Viz,

STRING Viix, STRING Viiy, STRING Viiz, STRING Viiix, STRING Viiiy, STRING Viiiz) --> for main matrix result type

GiD_fLAComponents(GiD_FILE fd, STRING axes_1, STRING axes_2, STRING axes_3) --> for local axis result type

GiD_fComplexScalarComp(GiD_FILE fd, STRING Re,STRING Im)

GiD_fComplexVectorComp(GiD_FILE fd, STRING Rex,STRING Imx,STRING Rey,STRING Imy,STRING Rez,STRING Imz)

Parameters:

GiD_FILE fd the file descriptor

int compc --> number of components names to write.

char * compv[] --> array of names.

Example:

C/C++

char cnames[] = {"X", "Y", "Z", "Mod"};

Gid_fResultComponents(fd,3, cnames);

FORTRAN

CHARACTER*10 XN, YN, ZN, MN

XN = 'X'

YN = 'Y'

ZN = 'Z'

MN = 'Mod'

CALL GiD_fVectorComp(fd,XN, YN, ZN, MN)

4.3.13 GiD_fResultUnit

int GiD_fResultUnit(GiD_FILE fd, CONST char * UnitName);

Description: Define the unit string associated to the current result,
either a single result block or the current result defined in a result group

Parameters:

See GiD_fBeginResult.

Example:

C/C++

GiD_fResultUnit(fd,"m");

FORTRAN

CHARACTER*10 UNITNAME

UNITNAME = 'm'

GID_FRESULTUNIT(fd,UNITNAME)

4.3.14 GiD_fBeginResultGroup

int GiD_fBeginResultGroup(GiD_FILE fd, const char * Analysis, double step,
GiD_ResultLocation Where,
const char * GaussPointsName);

Description: Begin a result group. All grouped in the same analysis and
step. See GiD online help on this topic.

Parameters:

See GiD_fBeginResult.

Example:

C/C++

GiD_fBeginResultGroup(fd,"Analysis", 1.0d0, GiD_OnNodes, NULL);

GiD_fResultDescription(fd,"EscalarNodos", GiD_Scalar);

GiD_fResultDescription(fd,"VectorNodos", GiD_Vector);

GiD_fBeginResultGroup

```
GiD_fResultDescription(fd,"Matrix", GiD_Matrix);
GiD_fResultDescription(fd,"Local Axes", GiD_LocalAxes);
GiD_fResultDescription(fd,"A complex Scalar", GiD_ComplexScalar);
GiD_fResultDescription(fd,"A complex Vector", GiD_ComplexVector);
GiD_fResultDescription(fd,"A complex Matrix", GiD_ComplexMatrix);
for ( i = 0; i < 9; i++ ) {
  GiD_fWriteScalar(fd,nodes[i].id, Random());
  GiD_fWriteVector(fd,nodes[i].id, Random(), Random(), Random());
  GiD_fWrite3DMatrix(fd,nodes[i].id, Random(), Random(), Random(), Random(), Random(),
Random());
  GiD_fWriteComplexScalar(fd,nodes[i].id, Random(), Random());
  GiD_fWriteComplexVector(fd,nodes[i].id, Random(), Random(), Random(), Random(),
Random(), Random());
  GiD_fWrite3DMatrix(fd,nodes[i].id, Random(), Random(), Random(), Random(), Random(),
Random(),
        Random(), Random(), Random(), Random(), Random(), Random());
}
GiD_EndResult(fd);
```

```
FORTRAN
CALL GID_FBEGINRESULTGROUP(fd,"Analysis", 1.0d0, 0, NULL)
CALL GID_FRESULTDESCRIPTION(fd,"EscalarNodos", 0)
CALL GID_FRESULTDESCRIPTION(fd,"VectorNodos",1)
CALL GID_FRESULTDESCRIPTION(fd,"Matrix",2)
CALL GID_FRESULTDESCRIPTION(fd,"Local Axes",5)
do idx=1,9
value = idx*1.5;
CALL GID_FWRITESCALAR(fd,nodes[i].id, value)
CALL GID_FWRITEVECTOR(fd,nodes[i].id, value, value*2, value*3)
CALL GID_FWRITE3DMATRIX(fd,i+1,value,value*2,value*3,value*4,value*7,value*1.1)
CALL GID_FWRITELOCALAXES(fd,i+1,value,value*3,value*5)
end do
GID_ENDRESULT(fd);
```

4.3.15 GiD_fResultDescription

GiD_fResultDescription

int GiD_fResultDescription(GiD_FILE fd, const char * Result, GiD_ResultType Type);

int GiD_fResultDescriptionDim(GiD_FILE fd, const char * Result, GiD_ResultType Type, size_t dim);

Description: Define a result member of a result group given the name and result type. The second prototype enable us to specify the dimension of the result types. Most of the types do not allow more than one dimension. Bellow if the set of valid dimensions for the argument dim given the value of Type:

- Scalar : 1 (GiD_fWriteScalar)
- Vector : 2 (GiD_fWrite2DVector), 3(GiD_fWriteVector) or 4 (GiD_fWriteVectorModule)
- Matrix : 3 (GiD_fWrite2DMatrix) or 6 (GiD_fWrite3DMatrix)
- PlainDeformationMatrix : 4 (GiD_fWritePlainDefMatrix)
- MainMatrix : 12 (GiD_fWriteMainMatrix)
- LocalAxes : 3 (GiD_fWriteLocalAxes)

Parameters:

See GiD_fBeginResult.

Example:

C/C++

See GiD_fBeginResultGroup

FORTRAN

See GiD_fBeginResultGroup

4.3.16 GiD_fResultValues

int GiD_fResultValues(GiD_FILE fd);

Description: This function is not needed anymore it is just maintained for backward compatibility.

Parameters:

Example:

4.3.17 GiD_fEndResult

int GiD_fEndResult(GiD_FILE fd);

GiD_fEndResult

Description: End Result Block.

Parameters:

GiD_FILE fd the file descriptor

Example:

C/C++

GiD_fEndResult(fd);

FORTRAN

CALL GID_FENDRESULT(fd)

4.3.18 GiD_fBeginOnMeshGroup

int GiD_fBeginOnMeshGroup(GiD_FILE fd, char * Name);

Description:

Results which are referred to a mesh group (see GiD_fBeginMeshGroup) should be written between a call to this function and GiD_fEndOnMeshGroup.

Parameters:

GiD_FILE fd the file descriptor

char* Name

Name of the mesh group where the results will be specified. This group must be previously defined in a call to GiD_fBeginMeshGroup.

Example:

C/C++

GiD_fBeginOnMeshGroup(fd,"steps 1, 2, 3 and 4");

FORTRAN

CALL GID_FBEGINONMESHGROUP(fd,"steps 1, 2, 3 and 4")

4.3.19 GiD_fEndOnMeshGroup

int GiD_fEndOnMeshGroup(GiD_FILE fd);

Description:

This function close a previously opened block of result over a mesh group. See

GiD_fEndOnMeshGroup

GiD_fBeginOnMeshGroup.

Parameters:

GiD_FILE fd the file descriptor

Example:

C/C++

```
GiD_fEndOnMeshGroup(fd);
```

FORTRAN

```
GID_fENDONMESHGROUP(fd);
```

4.3.20 GiD_fResultUserDefined

int GiD_fResultUserDefined(GiD_FILE fd, GP_CONST char * Name,GP_CONST char * Value)

Description: Write a special 'ResultUserDefined' comment whith a name and a value that GiD will ignore and could be interpreted by a problemtype for special uses.

Parameters:

GiD_FILE fd the file descriptor

Name: an arbitrary string

Value: an arbitrary string

Note: these strings must be ASCII, and avoid use the special character (end of line).

To avoid parsing problems double quotes characters will be replaced by single quote

Example:

C/C++

```
GiD_fResultUserDefined(fd,"ComponentsLocal","Nx Ny Nxy'');
```

4.3.21 GiD_fFlushPostFile

int GiD_fFlushPostFile(GiD_FILE fd);

Description: Flushes all pending output into the postprocess file. This function should be called only when strictly necessary when writing in GiD_PostAsciiZipped or GiD_PostBinary modes because it can degrade compression.

Parameters:

GiD_fFlushPostFile

GiD_FILE fd the file descriptor

Example:

C/C++

```
GiD_fFlushPostFile(fd);
```

FORTRAN

```
CALL GID_FFLUSHPOSTFILE(fd)
```

4.3.22 GiD_fWriteXXX

```
int GiD_fWriteScalar(GiD_FILE fd, int id, double v);
int GiD_fWrite2DVector(GiD_FILE fd, int id, double x, double y);
int GiD_fWriteVector(GiD_FILE fd, int id, double x, double y, double z);
int GiD_fWriteVectorModule(GiD_FILE fd, int id, double x, double y, double z, double mod);
int GiD_fWrite2DMatrix(GiD_FILE fd, int id, double Sxx, double Syy, double Sxy);
int   GiD_fWrite3DMatrix(GiD_FILE fd, int id, double Sxx, double Syy, double Szz, double Sxy, double Syz, double Sxz);
int   GiD_fWritePlainDefMatrix(GiD_FILE fd, int id, double Sxx, double Syy, double Sxy, double Szz);
int GiD_fWriteMainMatrix(GiD_FILE fd, int id,
                         double Si, double Sii, double Siii,
                         double Vix, double Viy, double Viz,
                         double Viix, double Viiy, double Viiz,
                         double Viiix, double Viiiy, double Viiiz);
int GiD_fWriteLocalAxes(GiD_FILE fd, int id, double euler_1, double euler_2, double euler_3);
int GiD_WriteComplexScalar( int id, double complex_real, double complex_imag);
int GiD_fWriteComplexScalar( GiD_FILE fd, int id, double complex_real, double complex_imag);
int GiD_fWrite2DComplexVector( GiD_FILE fd, int id,
                               double x_real, double x_imag,
                               double y_real, double y_imag);
int GiD_fWriteComplexVector( GiD_FILE fd, int id,
                             double x_real, double x_imag,
                             double y_real, double y_imag,
                             double z_real, double z_imag);
int GiD_fWrite2DComplexMatrix(GiD_FILE fd, int id,
                              double Sxx_real, double Syy_real, double Sxy_real,
```

```
                        double Sxx_imag, double Syy_imag, double Sxy_imag);
int GiD_fWrite3DComplexMatrix(GiD_FILE fd, int id,
                        double Sxx_real, double Syy_real, double Szz_real,
                        double Sxy_real, double Syz_real, double Sxz_real,
                        double Sxx_imag, double Syy_imag, double Szz_imag,
                        double Sxy_imag, double Syz_imag, double Sxz_imag);
int GiD_fWriteNurbsSurface(GiD_FILE fd, int id, int n, double * v);
int GiD_fWriteNurbsSurfaceVector(GiD_FILE fd, int id, int n, int num_comp, double * v);
```

Description:

Write result functions. Must be between GiD_fBeginResult and GiD_fEndResult

Parameters:

GiD_FILE fd the file descriptor

For GiD_fWriteNurbsSurface and GiD_fWriteNurbsSurfaceVector:

n: the number of control points

num_comp: number of vector components

v: the results in order like:

R1 = u1, v1

R2 = u1, v2

R3 = u1, v3

...

Rn = u1, vn

Rn+1 = u2,v1

Rn+2= u2,v2

Rn+3=u2,v3

R2n=u2,vn

....

Rk=um,vn

where u and v are nurbs directions.

In case that one result is not defined use GP_UNKNOWN.

Example:

GiD_fWriteXXX

C/C++

GiD_fWriteScalar(fd,3,4.6);

FORTRAN

INTEGER*4 idx

REAL*8 value

idx=3

value=4.6

CALL GID_FWRITESCALAR(fd,idx,value)

5 Download the library

The library can be downloaded at: ftp://www.gidhome.com/pub/Tools/gidpost