LS-Reader Tutorial

Introduction

This document describes the application functions interface of LS-Reader using C++, Python and C.

The LS-Reader is designed to read LS-DYNA results and it supports C, C++ and Python languages. It supports both Windows(vs2010, vs2015, vs2017, vs2019) and Linux(GCC>=4.1.2). Because of the simplicity of the LS-Reader, using the libraries is very convenient.

C++

D3plotReader

API Functions

```
class D3plotReader
{
public:
      D3plotReader(const char* name, bool force_to_single = false);
```

- Purpose: Constructor.
- Input: name - d3plot name, the second one is reserved.
- Return: D3plotReader object.

```
      bool GetData(D3P_DataType type, char* value, const
      D3P_Parameter& param = D3P_Parameter());
```

- Purpose: Get the value of the special data variable.
- Input: type - enum the data variables's name in d3plot.

  value - store the return value.

  param - structure of description which is the advance setting for getting special data in d3plot.
- Return: bool

```
};
```

D3P_Parameter

parameter to call D3plotReader::GetData*, only specific those member variables you are interested, otherwise, ignore this.

```
struct D3P_Parameter

{


        int ist;

        int ipt;

        int ipart;

        int i_rigid_wall;

        int ides;

        int ihv;

        int index_multisolver;

        int id_var_multisolver;

        const char* var_name;

};
```

1. **ist**: Specify the state number, starting from 0, as follows:

```
D3P_Parameter param;

param.ist = 167;

param.ipt = 2;

dr.GetData(D3P_SHELL_STRESS, (char*)shell_stress, param);
```

2. **ipt**: Specify the integration point, ranging in [0, MAXINT), as follows:

```
D3P_Parameter param;

param.ipt = 0;

...
```

3. **ipart**: Specify the index of part, starting from 0, as follows:

```
D3P_Parameter param;

param.ipart = 0;

...
```

4. **i_rigid_wall**: Specify the index of rigid wall, starting from 0, as follows:

```
D3P_Parameter param;

param.i_rigid_wall = 0;

...
```

5. **ides**: Specify the index of the des data, starting from 0, as follows:

```
D3P_Parameter param;

param.ides = 0;

...
```

6. **ihv**: Specify the index of history variables, starting from 0, as follows:

```
D3P_Parameter param;

param.ihv = 0;

...
```

7. **index_multisolver**: Specify the index of the multisolver domain, start from 0 and default is 0 also:

```
D3P_Parameter param;

param.index_multisolver = 0;

...
```

8. **id_var_multisolver**: Specify the index of the multisolver var, start from 0 and default is 0 also:

```
D3P_Parameter param;

param.id_var_multisolver = 0;

...
```

9. **var_name**: Specify name of output variables, currently used by DES and CPM data, default is empty

```
D3P_Parameter param;

param.var_name = "Start N";

...
```

DataType

```
struct D3P_Vector
{
	float v[3];

	float X() const { return v[0]; }
	float Y() const { return v[1]; }
	float Z() const { return v[2]; }

	bool operator==(const D3P_Vector& src);
};

struct D3P_VectorDouble
{
	double v[3];

	double X() const { return v[0]; }
	double Y() const { return v[1]; }
	double Z() const { return v[2]; }

	bool operator==(const D3P_VectorDouble& src);
};

struct D3P_Tensor
{
	float t[6];

	float X() const { return t[0]; }
	float Y() const { return t[1]; }
	float Z() const { return t[2]; }
```

```
        float XY() const { return t[3]; }
        float YZ() const { return t[4]; }
        float ZX() const { return t[5]; }

        bool operator==(const D3P_Tensor& src);
};

struct D3P_Solid
{
        int conn[10];
        int mat;

        int Mat() const { return mat; }
        int Node(int index) const { return conn[index]; }

        bool operator==(const D3P_Solid& src);
};

struct D3P_Tshell
{
        int conn[10];
        int mat;
        int Mat() const { return mat; }
        int Node(int index) const { return conn[index]; }

        bool operator==(const D3P_Tshell& src);
};

struct D3P_Beam
{
        int conn[2];
```

```
	int third;
	int w_int;
	int h_int;
	int mat;
	int Mat() const { return mat; }
	int Node(int index) const { return conn[index]; }

	bool operator==(const D3P_Beam& src);
};

struct D3P_Shell
{
	int conn[4];
	int mat;
	int Mat() const { return mat; }
	int Node(int index) const { return conn[index]; }

	bool operator==(const D3P_Shell& src);
};

struct D3P_Sph
{
	int id;
	unsigned int mat;
	int Mat() const { return mat; }
	int Id() const { return id; }

	bool operator==(const D3P_Sph& src);
};

struct D3P_VAR
```

```
{
    int type;
    char name[8];
       std::string Name() const { return name; }
       int Type() const { return type; }

       bool operator==(const D3P_VAR& src);
};

struct D3P_DES
{
       int id;
       int mat;
       float radius;
       float mass;
       float inertia;
       int Id() const { return id; }
       int Mat() const { return mat; }
       float Radius() const { return radius; }
       float Mass() const { return mass; }
       float Inertia() const { return inertia; }

       bool operator==(const D3P_DES& src);
};

struct D3P_AIRBAG_INFO
{
       int bagid;
       int startn;
       int npart;
       int ngas;
```

```
        int nchamber;
} ;
```

name	conversion	length	parameters
D3P_NUM_STATES	int	1	ignore
D3P_TIMES	float	D3P_NUM_STATES	ignore
D3P_TITLE	char		ignore
Global			
D3P_GLOBAL_KINETIC_ENERGY	float	1	ist
D3P_GLOBAL_INTERNAL_ENERGY	float	1	ist
D3P_GLOBAL_TOTAL_ENERGY	float	1	ist
D3P_GLOBAL_VELOCITY	D3P_Vector	1	ist
Part			
D3P_NUM_PARTS	int	1	ignore
D3P_PART_IDS	int	D3P_NUM_PARTS	ignore
D3P_PART_NAME	char	80	ipart
D3P_PART_INTERNAL_ENERGY	float	1	ist, ipart
D3P_PART_KINETIC_ENERGY	float	1	ist, ipart
D3P_PART_VELOCITY	D3P_Vector	1	ist, ipart
D3P_PART_MASS	float	1	ist, ipart
D3P_PART_HOURGLASS	float	1	ist, ipart
RIGID WALL			
D3P_NUM_RIGID_WALL	int	1	ignore
D3P_RIGID_WALL_FORCE	float	1	ist, i_rigid_wall
D3P_RIGID_WALL_POSITION	D3P_Vector	1	ist, i_rigid_wall
NODE			
D3P_NUM_NODES	int	1	ignore
D3P_NODE_INITIAL_COORDINATES	D3P_Vector	D3P_NUM_NODES	ignore
D3P_NODE_IDS	int	D3P_NUM_NODES	ignore
D3P_NODE_TEMPERATURE	float	D3P_NUM_NODES	ist
D3P_NODE_COORDINATES	D3P_Vector	D3P_NUM_NODES	ist
D3P_NODE_VELOCITIES	D3P_Vector	D3P_NUM_NODES	ist
D3P_NODE_ACCELERATIONS	D3P_Vector	D3P_NUM_NODES	ist

D3P_NODE_COORDINATES_DOUBLE	D3P_VectorDouble	D3P_NUM_NODES	ist
D3P_NODE_VELOCITIES_DOUBLE	D3P_VectorDouble	D3P_NUM_NODES	ist
D3P_NODE_ACCELERATIONS_DOUBLE	D3P_VectorDouble	D3P_NUM_NODES	ist
SOLID			
D3P_NUM_SOLID	int	1	ignore
D3P_SOLID_MAXINT	int	1	ignore
D3P_SOLID_CONNECTIVITY_MAT	D3P_Solid	D3P_NUM_SOLID	ignore
D3P_SOLID_IDS	int	D3P_NUM_SOLID	ignore
D3P_SOLID_STRESS	D3P_Tensor	D3P_NUM_SOLID	ist, ipt if necessary
D3P_SOLID_EFFECTIVE_PLASTIC_STRAIN	float	D3P_NUM_SOLID	ist, ipt if necessary
D3P_SOLID_STRAIN	D3P_Tensor	D3P_NUM_SOLID	ist, ipt if necessary
D3P_SOLID_HISTORY_VAR	float	D3P_NUM_SOLID	ist, ipt, ihv
TSHELL			
D3P_NUM_TSHELL	int	1	ignore
D3P_TSHELL_MAXINT	int	1	ignore
D3P_TSHELL_CONNECTIVITY_MAT	D3P_Tshell	D3P_NUM_TSHELL	ignore
D3P_TSHELL_IDS	int	D3P_NUM_TSHELL	ignore
D3P_TSHELL_STRESS	D3P_Tensor	D3P_NUM_TSHELL	ist, ipt
D3P_TSHELL_EFFECTIVE_PLASTIC_STRAIN	float	D3P_NUM_TSHELL	ist, ipt
D3P_TSHELL_STRAIN	D3P_Tensor	D3P_NUM_TSHELL	ist, ipt ipt=0:mean 1:inner 2:outer
D3P_TSHELL_HISTORY_VAR	float	D3P_NUM_TSHELL	ist, ipt, ihv
BEAM			

D3P_NUM_BEAM	int	1	ignore
D3P_BEAM_MAXINT	int	1	ignore
D3P_BEAM_CONNECTIVITY_THIRD_MAT	D3P_Beam	D3P_NUM_BEAM	ignore
D3P_BEAM_IDS	int	D3P_NUM_BEAM	ignore
D3P_BEAM_AXIAL_FORCE	float	D3P_NUM_BEAM	ist
D3P_BEAM_S_SHEAR_RESULTANT	float	D3P_NUM_BEAM	ist
D3P_BEAM_T_SHEAR_RESULTANT	float	D3P_NUM_BEAM	ist
D3P_BEAM_S_BENDING_MOMENT	float	D3P_NUM_BEAM	ist
D3P_BEAM_T_BENDING_MOMENT	float	D3P_NUM_BEAM	ist
D3P_BEAM_TORSIONAL_RESULTANT	float	D3P_NUM_BEAM	ist
D3P_BEAM_RS_SHEAR_STRESS	float	D3P_NUM_BEAM	ist, ipt
D3P_BEAM_TR_SHEAR_STRESS	float	D3P_NUM_BEAM	ist, ipt
D3P_BEAM_AXIAL_STRESS	float	D3P_NUM_BEAM	ist, ipt
D3P_BEAM_AXIAL_PLASTIC_STRAIN	float	D3P_NUM_BEAM	ist, ipt
D3P_BEAM_AXIAL_STRAIN	float	D3P_NUM_BEAM	ist, ipt
D3P_BEAM_HISTORY_VAR	float	D3P_NUM_BEAM	ist, ipt, ihv
SHELL			
D3P_NUM_SHELL	int	1	ignore
D3P_SHELL_MAXINT	int	1	ignore
D3P_SHELL_CONNECTIVITY_MAT	D3P_Shell	D3P_NUM_SHELL	ignore
D3P_SHELL_IDS	int	D3P_NUM_SHELL	ignore
D3P_SHELL_STRESS	D3P_Tensor	D3P_NUM_SHELL	ist, ipt
D3P_SHELL_EFFECTIVE_PLASTIC_STRAIN	float	D3P_NUM_SHELL	ist, ipt
D3P_SHELL_STRAIN	D3P_Tensor	D3P_NUM_SHELL	ist, ipt ipt=0:mean 1:inner 2:outer
D3P_SHELL_HISTORY_VAR	float	D3P_NUM_SHELL	ist, ipt, ihv

D3P_SHELL_MX	float	D3P_NUM_SHELL	ist
D3P_SHELL_MY	float	D3P_NUM_SHELL	ist
D3P_SHELL_MXY	float	D3P_NUM_SHELL	ist
D3P_SHELL_QX	float	D3P_NUM_SHELL	ist
D3P_SHELL_QY	float	D3P_NUM_SHELL	ist
D3P_SHELL_NX	float	D3P_NUM_SHELL	ist
D3P_SHELL_NY	float	D3P_NUM_SHELL	ist
D3P_SHELL_NXY	float	D3P_NUM_SHELL	ist
DELETION			
D3P_HAS_DELETION	bool	1	ist
D3P_SOLID_DELETION	float	D3P_NUM_SOLID	ist
D3P_TSHELL_DELETION	float	D3P_NUM_TSHELL	ist
D3P_SHELL_DELETION	float	D3P_NUM_SHELL	ist
D3P_BEAM_DELETION	float	D3P_NUM_BEAM	ist
SPH			
D3P_NUM_SPH	int	1	ignore
D3P_SPH_NODE_MAT	D3P_Sph	D3P_NUM_SPH	ignore
D3P_SPH_RADIUS	float	D3P_NUM_SPH	ist
D3P_SPH_PRESSURE	float	D3P_NUM_SPH	ist
D3P_SPH_STRESS	D3P_Tensor	D3P_NUM_SPH	ist
D3P_SPH_PLASTIC_STRAIN	float	D3P_NUM_SPH	ist
D3P_SPH_DENSITY	float	D3P_NUM_SPH	ist
D3P_SPH_INTERNAL_ENERGY	float	D3P_NUM_SPH	ist
D3P_SPH_NUMBER_OF_PARTICLE_NEIGHBORS	int	D3P_NUM_SPH	ist
D3P_SPH_STRAIN	D3P_Tensor	D3P_NUM_SPH	ist
D3P_SPH_MASS	float	D3P_NUM_SPH	ist
DES			
D3P_HAS_DES_DATA	bool	1	ignore

D3P_NUM_DES_DATA	int	1	ignore
D3P_NUM_DES_PART_IN_GEOM	int	1	ides if necessary
D3P_NUM_DES_ELEM_IN_GEOM	int	1	ides if necessary
D3P_NUM_DES_PART_IN_STATE	int	1	ides if necessary
D3P_NUM_DES_ELEM_IN_STATE	int	1	ides if necessary
D3P_NUM_DES_PART_VAR_IN_GEOM	int	1	ides if necessary
D3P_DES_PART_VAR_LIST_IN_GEOM	D3P_VAR	D3P_NUM_DES_PART_VAR_IN_GEOM	ides if necessary
D3P_NUM_DES_ELEM_VAR_IN_GEOM	int	1	ides if necessary
D3P_DES_ELEM_VAR_LIST_IN_GEOM	D3P_VAR	D3P_NUM_DES_ELEM_VAR_IN_GEOM	ides if necessary
D3P_NUM_DES_PART_VAR_IN_STATE	int	1	ides if necessary
D3P_DES_PART_VAR_LIST_IN_STATE	D3P_VAR	D3P_NUM_DES_PART_VAR_IN_STATE	ides if necessary
D3P_NUM_DES_ELEM_VAR_IN_STATE	int	1	ides if necessary
D3P_DES_ELEM_VAR_LIST_IN_STATE	D3P_VAR	D3P_NUM_DES_ELEM_VAR_IN_STATE	ides if necessary
D3P_DES_NODAL_MAT_RADIUS_MASS_INERTIA	D3P_DES	D3P_NUM_DES_ELEM_IN_GEOM	ides if necessary
D3P_DES_DATA_IN_STATE	int/float/vector/tensor..depends	D3P_NUM_DES_ELEM_IN_STATE	var_name, ist, ides if necessary
CPM			
D3P_HAS_CPM_DATA	bool	1	ignore
D3P_CPM_NUM_AIRBAGS	int	1	ignore
D3P_CPM_NUM_PARTICLES	int	1	ignore
D3P_CPM_NUM_GEOM_VAR	int	1	ignore

D3P_CPM_GEOM_VAR_LIST	D3P_VAR	D3P_CPM_NUM_GEOM_VAR	ignore
D3P_CPM_GEOM_DATA	D3P_VAR	D3P_CPM_NUM_GEOM_VAR	ignore
D3P_CPM_NUM_STATE_VAR	int	1	ignore
D3P_CPM_STATE_VAR_LIST	D3P_VAR	D3P_CPM_NUM_STATE_VAR	ignore
D3P_CPM_STATE_DATA	int/float...depends	D3P_CPM_NUM_PARTICLES	var_name, ist
D3P_CPM_NUM_STATE_GEOM_VAR	int	1	ignore
D3P_CPM_STATE_GEOM_VAR_LIST	D3P_VAR	D3P_CPM_NUM_STATE_GEOM_VAR	ignore
D3P_CPM_STATE_GEOM_DATA	int/float...depends	D3P_CPM_NUM_AIRBAGS	var_name, ist
Multisolver			
D3P_HAS_MS_DATA	bool	1	ignore
D3P_MS_NUM_DOMAINS	int	1	ignore
D3P_MS_DOMAIN_ID	int	1	index_multisolver
D3P_MS_DOMAIN_NAME	char	80	index_multisolver
D3P_MS_DOMAIN_VAR_NUM	int	1	index_multisolver
D3P_MS_DOMAIN_VARS_LIST	int	D3P_MS_DOMAIN_VAR_NUM	index_multisolver
D3P_MS_VAR_NAME	char	80	id_var_multisolver
D3P_MS_VAR_IS_VECTOR	bool	1	id_var_multisolver
D3P_MS_VAR_IS_SCALAR	bool	1	id_var_multisolver
D3P_MS_VAR_IS_TENSOR	bool	1	id_var_multisolver
D3P_MS_DOMAIN_VAR_LENGTH	int	1	ist, index_multisol

			ver
D3P_MS_DOMAIN_IS_SOLID	bool	1	ist, index_multisolver
D3P_MS_DOMAIN_IS_SHELL	bool	1	ist, index_multisolver
D3P_MS_DOMAIN_IS_BEAM	bool	1	ist, index_multisolver
D3P_MS_DOMAIN_ELEM_NUM_IN_STATE	int	1	ist, index_multisolver
D3P_MS_SOLID_CONNECTIVITY_MAT_IN_STATE	D3P_Solid	D3P_MS_DOMAIN_ELEM_NUM_IN_STATE	ist, index_multisolver
D3P_MS_SHELL_CONNECTIVITY_MAT_IN_STATE	D3P_Shell	D3P_MS_DOMAIN_ELEM_NUM_IN_STATE	ist, index_multisolver
D3P_MS_BEAM_CONNECTIVITY_MAT_IN_STATE	D3P_Beam	D3P_MS_DOMAIN_ELEM_NUM_IN_STATE	ist, index_multisolver
D3P_MS_DOMAIN_NODE_NUM_IN_STATE	int	1	ist, index_multisolver
D3P_MS_DOMAIN_COORD_IN_STATE	D3P_Vector	D3P_MS_DOMAIN_NODE_NUM_IN_STATE	ist, index_multisolver
D3P_MS_DOMAIN_DATA_IN_STATE	float or D3P_Vector or D3P_Tensor	D3P_MS_DOMAIN_VAR_LENGTH	ist, index_multisolver,id_var_multisolver
D3P_MS_DOMAIN_DATA_IS_ON_STRUCTURE_ELEMENT	bool	1	index_multisolver
D3P_MS_DOMAIN_DATA_IS_ON_MS_NODE	bool	1	index_multisolver
D3P_MS_DOMAIN_DATA_IS_ON_MS_ELEMENT,	bool	1	index_multisolver

D3P_MS_DOMAIN_IS_FOLLOW_SURFACE_METHOD	bool	1	index_multisolver
D3P_MS_DOMAIN_NODE_NUM_ONSURFACE_IN_STATE	int	1	ist, index_multisolver
D3P_MS_DOMAIN_SURFACE_IDS_IN_STATE	int	D3P_MS_DOMAIN_NODE_NUM_ONSURFACE_IN_STATE	ist, index_multisolver

How to use

Sample1.cpp

Purpose: obtain resultant displacement for all the nodes and find maximum value.

ist: last.

```
#include "../config.h"          /* define DATA_PATH_1 "d3plot/path" */
#include "../d3plotreader.h"

#include <string.h>
#include <math.h>
#include <iostream>

using namespace std;

int main(){
    string d3plot = DATA_PATH_1;
    D3plotReader dr(d3plot.c_str());

    int num_nodes=0;
    dr.GetData(D3P_NUM_NODES, (char*)&num_nodes);

    int num_states=0;
    dr.GetData(D3P_NUM_STATES, (char*)&num_states);

    D3P_Parameter param;
    param.ist = num_states-1;

    D3P_Vector* node_ini_coor = new D3P_Vector[num_nodes];
    dr.GetData(D3P_NODE_INITIAL_COORDINATES, (char*)node_ini_coor);
    D3P_Vector* node_coor = new D3P_Vector[num_nodes];
    dr.GetData(D3P_NODE_COORDINATES, (char*)node_coor, param);
```

```
    /* obtain resultant displacement for all nodes
       and find maximum value.
    */
    float* node_res_disp = new float[num_nodes];
    float disp_x, disp_y, disp_z, tmp;
    for(int i=0; i<num_nodes; i++){
            disp_x = node_coor[i].X()-node_ini_coor[i].X();
            disp_y = node_coor[i].Y()-node_ini_coor[i].Y();
            disp_z = node_coor[i].Z()-node_ini_coor[i].Z();
            tmp = pow(disp_x, 2) + pow(disp_y, 2) + pow(disp_z, 2);
            node_res_disp[i] = pow(tmp, 0.5);
    }

    float max = node_res_disp[0];
    int index=0;
    for(int i=0; i<num_nodes; i++){
        if(max < node_res_disp[i]){
            max = node_res_disp[i];
            index = i;
        }
    }

    cout<<"Maximum resultant displacement of nodes is: "<<max
        <<", index is "<<index<<endl;
};
```

Sample2.cpp

Purpose: extract Variable data for Multisolver.

ist: 2

```
#include "../config.h"          /* define DATA_PATH_3 "d3plot/path" */
#include "../d3plotreader.h"

#include <string.h>
#include <math.h>
#include <iostream>

using namespace std;

int main() {
	string d3plot = DATA_PATH_3;
	D3plotReader dr_ms(d3plot.c_str());
	bool has_ms_data = false;
	dr_ms.GetData(D3P_HAS_MS_DATA, (char*)&has_ms_data);
	if(!has_ms_data)
	{
		cout << "No Multisolver Data" << endl;
		return 0;
	}

	int num_ms_datasets = 0;
	dr_ms.GetData(D3P_MS_NUM_DOMAINS, (char*)&num_ms_datasets);

	D3P_Parameter para;
	int* domain_var_ids;
	for (int dataset = 0; dataset < num_ms_datasets; dataset++)
	{
```

```
para.index_multisolver = dataset;


int domain_var_num = 0;
dr_ms.GetData(
	D3P_MS_DOMAIN_ID, (char*)& domain_var_num, para
);

domain_var_ids = new int[domain_var_num];
dr_ms.GetData(
	D3P_MS_DOMAIN_VARS_LIST, (char*)domain_var_ids, para
);

para.ist = 2;
for (int var = 0; var < domain_var_num; var++)
{
	para.id_var_multisolver = domain_var_ids[var];

	int sizevar = 0;
	dr_ms.GetData(
		D3P_MS_DOMAIN_VAR_LENGTH, (char*)& sizevar, para
	);

	bool is_scalar = false;
	bool is_vector = false;
	bool is_tensor = false;
	dr_ms.GetData(
		D3P_MS_VAR_IS_SCALAR, (char*)& is_scalar, para
	);
	dr_ms.GetData(
		D3P_MS_VAR_IS_VECTOR, (char*)& is_vector, para
	);
```

```
dr_ms.GetData(
	D3P_MS_VAR_IS_TENSOR, (char*)& is_tensor, para
);

if (is_scalar)
{
	float *value = new float[sizevar];
	dr_ms.GetData(
		D3P_MS_DOMAIN_DATA_IN_STATE, (char*)value, para
	);
	cout << "Value type: scalar, value[0]=" << value[0]
		 << endl;
	delete[] value;
}
if (is_vector)
{
	float maxvalue = -1.0e20;
	D3P_Vector *value = new D3P_Vector[sizevar];
	dr_ms.GetData(
		D3P_MS_DOMAIN_DATA_IN_STATE, (char*)value, para
	);
	cout << "Value type: vector, value[0].X()="
		 << value[0].X() << endl;
	delete[] value;
}
if (is_tensor)
{
	D3P_Tensor *value = new D3P_Tensor[sizevar];
	dr_ms.GetData(
		D3P_MS_DOMAIN_DATA_IN_STATE, (char*)value, para
	);
```

```
                        cout << "Value type: tensor, value[0].X()"
                                << value[0].X() << endl;
                        delete[] value;
                    }
                }
            }
            delete[] domain_var_ids;
};
```

BinoutReader

API Functions

```
class BinoutReader
{
public:
```

```
BinoutReader(const char* filename);
```

- ❖ Purpose: Constructor.
- ❖ Input: filename- binout name.
- ❖ Return: BinoutReader object.

```
bool SetBranch(const std::string& branch);
```

- ❖ Purpose: Set current branch.
- ❖ Input: branch: The name of the branch to set.
- ❖ Return: True.

```
bool GetBranch(std::vector<std::string>& branches);
```

- ❖ Purpose: Get branches.
- ❖ Input: branches: store return value.
- ❖ Return: True.

```
bool SetId(const std::string& id);
```

- ❖ Purpose: Set current id.
- ❖ Input: id: The id to set.
- ❖ Return: True.

```
bool SetId(unsigned int id);
```

❖ Purpose: Set current id.

❖ Input: id: The id to set.

❖ Return: True.

```
bool SetId(unsigned int id, bool master);
```

❖ Purpose: Set current id.

❖ Input: id: The id to set.

master: choose master or slave.

❖ Return: True.

```
bool GetId(std::vector<unsigned int>& ids);
```

❖ Purpose: Get ids.

❖ Input: ids: store the return values.

❖ Return: True.

```
bool SetComponent(const std::string& comp);
```

❖ Purpose: Set current component.

❖ Input: branch: The name of the component to set.

❖ Return: True.

```
bool GetComponent(std::vector<std::string>& comps);
```

❖ Purpose: Get components.

❖ Input: comps: store the return values.

❖ Return: True.

```
bool GetXArray(std::vector<double>& x_array);
```

- ❖ Purpose: Get the array of X direction.
- ❖ Input: x_array: store the return value.
- ❖ Return: True.

```
bool GetYArray(std::vector<double>& y_array);
```

- ❖ Purpose: Get the array of Y direction.
- ❖ Input: y_array: store the return values.
- ❖ Return: True.

```
};
```

How to use

Sample1.cpp

Purpose: obtain branches and component, and get x_array, y_array.

Branch: nodout.

Component: x_acceleration.

Id: 1787

Ouput: nodoutPy.dat

```
#include <string.h>
#include <iostream>
#include <vector>

#include "binoutreader.h"
using namespace std;

int main()
{
	//======================BinoutReader=====================

	string binout_file = DATA_PATH_BINOUT;
	bool ret = BinoutReader::IsValid(binout_file);

	BinoutReader br(binout_file.c_str());

	std::vector<std::string> branches;
	br.GetBranch(branches);
	cout << "Branches: " << endl;
	for (size_t i = 0; i < branches.size(); i++)
	{
		cout << branches[i] <<",";
	}
```

```
        cout << "\n" << endl;


        br.SetBranch("nodout");
        br.SetId(1787);
        br.SetComponent("x_acceleration");


        std::vector<double> x_array;
        std::vector<double> y_array;
        br.GetXArray(x_array);
        br.GetYArray(y_array);


        cout << "Branch=\"nodout\", id=\"1787\", component=\"x_acceleration\"" <<
endl;
        cout << "x_array,    y_array:" << endl;
        for (size_t i = 0; i < 20; i++)
        {
               cout << x_array[i] << ",    " << y_array[i] << endl;
        }
        cout << "..." << endl;
};
```

Python

D3plotReader

API Functions(*Recommended*)

```
class D3plotReader():
    def __init__(self, path):
        pass
```

❖ Purpose: Constructor.

❖ Input: path: d3plot name.

❖ Return: D3plotReader object.

Example: `dr = D3plotReader("d3plot/file/path")`

```
    def get_data(self, type, param):
        pass
```

❖ Purpose: Extract data.

❖ Input: type: type - enum the data variables' name in d3plot.

param: structure of description which is the advance setting for getting special data in d3plot.

❖ Return: data.

Example:

```
dr = D3plotReader("d3plot/file/path")
p = D3P_Parameter()
p.ist = 11
p.ipt = 0
shell_stress = dr.get_data(DataType.D3P_SHELL_STRESS, p)
```

Or

```
dr = D3plotReader("d3plot/file/path")
```

```
shell_stress = dr.get_data(
DataType.D3P_SHELL_STRESS, ist=11, ipt=0
)
```

API Functions *(Deprecated)*

```
class D3plotReader():
    def __init__(self, path):
        pass
```

❖ Purpose: Constructor.

❖ Input: path: d3plot name.

❖ Return: D3plotReader object.

Example:`dr = D3plotReader("d3plot/file/path")`

```
    def GetDataInt(self, type, param):
        pass
```

Deprecated. Use the get_data(...) instead.

❖ Purpose: Get an integer value.

❖ Input: type - enum the data variables' name in d3plot.

Param - structure of description which is the advance setting for getting special data in d3plot.

❖ Return: int

```
    def GetDataFloat(self, type, param):
        pass
```

Deprecated. Use the get_data(...) instead.

❖ Purpose: Get a float value.

❖ Input: type - enum the data variables' name in d3plot.

param - structure of description which is the advance setting for getting special data in d3plot.

❖ Return: float.

```
def GetDataString(self, type, param):
    pass
```

Deprecated. Use the get_data(...) instead.

❖ Purpose: Get a string value.

❖ Input: type - enum the data variables' name in d3plot.

param - structure of description which is the advance setting for getting special data in d3plot.

❖ Return: string.

```
def GetDataIntArray(self, type, param):
    pass
```

Deprecated. Use the get_data(...) instead.

❖ Purpose: Get a int array.

❖ Input: type - enum the data variables' name in d3plot.

param - structure of description which is the advance setting for getting special data in d3plot.

❖ Return: int array.

```
def GetDataFloatArray(self, type, param):
    pass
```

Deprecated. Use the get_data(...) instead.

❖ Purpose: Get a float array.

❖ Input: type - enum the data variables' name in d3plot.

param - structure of description which is the advance setting for getting special data in d3plot.

❖ Return: float array.

```
def GetDataVectorArray(self, type, param):
    pass
```

Deprecated. Use the get_data(...) instead.

❖ Purpose: Get a vector array.

❖ Input: type - enum the data variables' name in d3plot.

param - structure of description which is the advance setting for getting special data in d3plot.

❖ Return: vector array.

```
def GetDataTensorArray(self, type, param):
    pass
```

Deprecated. Use the get_data(...) instead.

❖ Purpose: Get a tensor array.

❖ Input: type - enum the data variables' name in d3plot.

param - structure of description which is the advance setting for getting special data in d3plot.

❖ Return: tensor array.

```
def GetDataSolidArray(self):
    pass
```

Deprecated. Use the get_data(...) instead.

❖ Purpose: Get solid elements array.

❖ Return: solid elements array.

```
def GetDataTshellArray(self):
    pass
```

Deprecated. Use the get_data(...) instead.

❖ Purpose: Get tshell elements array.

❖ Return: tshell elements array.

```
def GetDataBeamArray(self):
    pass
```

Deprecated. Use the get_data(...) instead.

❖ Purpose: Get beam elements array.

❖ Return: beam elements array.

```
def GetDataShellArray(self):
    pass
```

Deprecated. Use the get_data(...) instead.

❖ Purpose: Get shell elements array.

❖ Return: shell elements array.

```
def GetDataSphArray(self):
    pass
```

Deprecated. Use the get_data(...) instead.

❖ Purpose: Get sph elements array.

❖ Return: sph elements array.

D3P_Parameter

parameter to call D3plotReader::get_data*, only specific those member variables you are interested, otherwise, ignore this.

```
class D3P_Parameter:

    def __init__(self):

        self.ist = -1

        self.ipt = -1

        self.ipart = -1

        self.i_rigid_wall = -1

        self.ides = -1

        self.ihv = -1

        self.index_multisolver = -1

        self.id_var_multisolver = -1

        self.var_name = ""
```

1. **ist**: Specify the state number, starting from 0, as follows:

   ```
   shell_thickness = dr.get_data(DataType.D3P_SHELL_THICKNESS, ist=11)

   Or

   p = D3P_Parameter()

   p.ist = 11

   shell_thickness = dr.get_data(DataType.D3P_SHELL_THICKNESS, p)
   ```

2. **ipt**: Specify the integration point, ranging in [0, MAXINT), as follows:

   ```
   shell_stress = dr.get_data(DataType.D3P_SHELL_STRESS, ist=11, ipt=0)

   Or

   p = D3P_Parameter()

   p.ist = 11

   p.ipt = 0
   ```

```
shell_stress = dr.get_data(DataType.D3P_SHELL_STRESS, p)
```

3. **ipart**: Specify the index of part, starting from 0, as follows:

```
part_name = dr.get_data(DataType.D3P_PART_NAME, ipart=0)
Or
p = D3P_Parameter()
p.ipart = 0
part_name = dr.get_data(DataType.D3P_PART_NAME, p)
```

4. **i_rigid_wall**: Specify the index of rigid wall, starting from 0, as follows:

```
r_wall_f = dr.get_data(
      DataType.D3P_RIGID_WALL_FORCE, ist=11, i_rigid_wall=0
)
Or
p = D3P_Parameter()
p.ist = 11
p.i_rigid_wall = 0
r_wall_f = dr.get_data(DataType.D3P_RIGID_WALL_FORCE, p)
```

5. **ides**: Specify the index of the des data, starting from 0, as follows:

```
num_des = dr.get_data(DataType.D3P_NUM_DES_PART_IN_GEOM, ides=0)
Or
p = D3P_Parameter()
p.ides = 0
num_des = dr.get_data(DataType.D3P_NUM_DES_PART_IN_GEOM, p)
```

6. **ihv**: Specify the index of history variables, starting from 0, as follows:

```
solid_hsvar = dr.get_data(
      DataType.D3P_SOLID_HISTORY_VAR, ist=11, ipt=0, ihv=5
)
Or
```

```
p = D3P_Parameter()

p.ist = 11

p.ipt = 0

p.ihv = 5

solid_hsvar = dr.get_data(DataType.D3P_SOLID_HISTORY_VAR, p)
```

7. **index_multisolver**: Specify the index of the multisolver domain, start from 0 and default is 0 also:

```
ms_id = dr.get_data(DataType.D3P_MS_DOMAIN_ID, index_multisolver=0)

Or

p = D3P_Parameter()

p.index_multisolver = 0

ms_id = dr.get_data(DataType.D3P_MS_DOMAIN_ID, p)
```

8. **id_var_multisolver**: Specify the index of the multisolver var, start from 0 and default is 0 also:

```
ms_varn = dr.get_data(DataType.D3P_MS_VAR_NAME, id_var_multisolver=0)

Or

p = D3P_Parameter()

p.id_var_multisolver = 0

ms_varn = dr.get_data(DataType.D3P_MS_VAR_NAME, p)
```

9. **var_name**: Specify name of output variables, currently used by DES and CPM data, default is empty

```
cpm_geodt = dr.get_data(DataType.D3P_CPM_GEOM_DATA, var_name='cpm1')

Or

p = D3P_Parameter()

p.var_name = 'cpm1'

cpm_geodt = dr.get_data(DataType.D3P_CPM_GEOM_DATA, p)
```

DataType

```
class D3P_Vector():
    def x(self):
        pass
    def y(self):
        pass
    def z(self):
        pass


class D3P_VectorDouble():
    def x(self):
        pass
    def y(self):
        pass
    def z(self):
        pass


class D3P_Tensor():
    def x(self):
        pass
    def y(self):
        pass
    def z(self):
        pass
    def xy(self):
        pass
```

```
    def yz(self):
        pass
    def zx(self):
        pass

class D3P_Solid():
    def node(self, index):
        pass
    def mat(self):
        pass

class D3P_Tshell():
    def node(self, index):
        pass
    def mat(self):
        pass

class D3P_Beam():
    def node(self, index):
        pass
    def mat(self):
        pass

class D3P_Shell():
    def node(self, index):
        pass
    def mat(self):
```

```
        pass


class D3P_Sph():
    def id(self):
        pass
    def mat(self):
        pass


class D3P_Var():
    def type(self):
        pass
    def name(self):
        pass


class D3P_Des():
    def id(self):
        pass
    def mat(self):
        pass
    def radius(self):
        pass
    def mass(self):
        pass
    def inertia(self):
        pass
```

---

name	conversion	length	parameters
D3P_NUM_STATES	int	1	ignore
D3P_TIMES	float	D3P_NUM_STATES	ignore
D3P_TITLE	Char		ignore
Global			
D3P_GLOBAL_KINETIC_ENERGY	float	1	ist
D3P_GLOBAL_INTERNAL_ENERGY	float	1	ist
D3P_GLOBAL_TOTAL_ENERGY	float	1	ist
D3P_GLOBAL_VELOCITY	D3P_Vector	1	ist
Part			
D3P_NUM_PARTS	int	1	ignore
D3P_PART_IDS	int	D3P_NUM_PARTS	ignore
D3P_PART_NAME	char	80	ipart
D3P_PART_INTERNAL_ENERGY	float	1	ist, ipart
D3P_PART_KINETIC_ENERGY	float	1	ist, ipart
D3P_PART_VELOCITY	D3P_Vector	1	ist, ipart
D3P_PART_MASS	float	1	ist, ipart
D3P_PART_HOURGLASS	float	1	ist, ipart
RIGID WALL			
D3P_NUM_RIGID_WALL	int	1	ignore
D3P_RIGID_WALL_FORCE	float	1	ist, i_rigid_wall
D3P_RIGID_WALL_POSITION	D3P_Vector	1	ist, i_rigid_wall
NODE			
D3P_NUM_NODES	int	1	ignore
D3P_NODE_INITIAL_COORDINATES	D3P_Vector	D3P_NUM_NODES	ignore
D3P_NODE_IDS	int	D3P_NUM_NODES	ignore
D3P_NODE_TEMPERATURE	float	D3P_NUM_NODES	ist
D3P_NODE_COORDINATES	D3P_Vector	D3P_NUM_NODES	ist
D3P_NODE_VELOCITIES	D3P_Vector	D3P_NUM_NODES	ist
D3P_NODE_ACCELERATIONS	D3P_Vector	D3P_NUM_NODES	ist
D3P_NODE_COORDINATES_DOUBLE	D3P_VectorDouble	D3P_NUM_NODES	ist

D3P_NODE_VELOCITIES_DOUBLE	D3P_VectorDouble	D3P_NUM_NODES	ist
D3P_NODE_ACCELERATIONS_DOUBLE	D3P_VectorDouble	D3P_NUM_NODES	ist
SOLID			
D3P_NUM_SOLID	int	1	ignore
D3P_SOLID_MAXINT	int	1	ignore
D3P_SOLID_CONNECTIVITY_MAT	D3P_Solid	D3P_NUM_SOLID	ignore
D3P_SOLID_IDS	int	D3P_NUM_SOLID	ignore
D3P_SOLID_STRESS	D3P_Tensor	D3P_NUM_SOLID	ist, ipt if necessary
D3P_SOLID_EFFECTIVE_PLASTIC_STRAIN	float	D3P_NUM_SOLID	ist, ipt if necessary
D3P_SOLID_STRAIN	D3P_Tensor	D3P_NUM_SOLID	ist, ipt if necessary
D3P_SOLID_HISTORY_VAR	float	D3P_NUM_SOLID	ist, ipt, ihv
TSHELL			
D3P_NUM_TSHELL	int	1	ignore
D3P_TSHELL_MAXINT	int	1	ignore
D3P_TSHELL_CONNECTIVITY_MAT	D3P_Tshell	D3P_NUM_TSHELL	ignore
D3P_TSHELL_IDS	int	D3P_NUM_TSHELL	ignore
D3P_TSHELL_STRESS	D3P_Tensor	D3P_NUM_TSHELL	ist, ipt
D3P_TSHELL_EFFECTIVE_PLASTIC_STRAIN	float	D3P_NUM_TSHELL	ist, ipt
D3P_TSHELL_STRAIN	D3P_Tensor	D3P_NUM_TSHELL	ist, ipt
D3P_TSHELL_HISTORY_VAR	float	D3P_NUM_TSHELL	ist, ipt, ihv
BEAM			
D3P_NUM_BEAM	int	1	ignore
D3P_BEAM_MAXINT	int	1	ignore

D3P_BEAM_CONNECTIVITY_THIRD_MAT	D3P_Beam	D3P_NUM_BEAM	ignore
D3P_BEAM_IDS	int	D3P_NUM_BEAM	ignore
D3P_BEAM_AXIAL_FORCE	float	D3P_NUM_BEAM	ist
D3P_BEAM_S_SHEAR_RESULTANT	float	D3P_NUM_BEAM	ist
D3P_BEAM_T_SHEAR_RESULTANT	float	D3P_NUM_BEAM	ist
D3P_BEAM_S_BENDING_MOMENT	float	D3P_NUM_BEAM	ist
D3P_BEAM_T_BENDING_MOMENT	float	D3P_NUM_BEAM	ist
D3P_BEAM_TORSIONAL_RESULTANT	float	D3P_NUM_BEAM	ist
D3P_BEAM_RS_SHEAR_STRESS	float	D3P_NUM_BEAM	ist, ipt
D3P_BEAM_TR_SHEAR_STRESS	float	D3P_NUM_BEAM	ist, ipt
D3P_BEAM_AXIAL_STRESS	float	D3P_NUM_BEAM	ist, ipt
D3P_BEAM_AXIAL_PLASTIC_STRAIN	float	D3P_NUM_BEAM	ist, ipt
D3P_BEAM_AXIAL_STRAIN	float	D3P_NUM_BEAM	ist, ipt
D3P_BEAM_HISTORY_VAR	float	D3P_NUM_BEAM	ist, ipt, ihv
SHELL			
D3P_NUM_SHELL	int	1	ignore
D3P_SHELL_MAXINT	int	1	ignore
D3P_SHELL_CONNECTIVITY_MAT	D3P_Shell	D3P_NUM_SHELL	ignore
D3P_SHELL_IDS	int	D3P_NUM_SHELL	ignore
D3P_SHELL_STRESS	D3P_Tensor	D3P_NUM_SHELL	ist, ipt
D3P_SHELL_EFFECTIVE_PLASTIC_STRAIN	float	D3P_NUM_SHELL	ist, ipt
D3P_SHELL_STRAIN	D3P_Tensor	D3P_NUM_SHELL	ist, ipt
D3P_SHELL_HISTORY_VAR	float	D3P_NUM_SHELL	ist, ipt, ihv
D3P_SHELL_MX	float	D3P_NUM_SHELL	ist
D3P_SHELL_MY	float	D3P_NUM_SHELL	ist
D3P_SHELL_MXY	float	D3P_NUM_SHELL	ist
D3P_SHELL_QX	float	D3P_NUM_SHELL	ist

D3P_SHELL_QY	float	D3P_NUM_SHELL	ist
D3P_SHELL_NX	float	D3P_NUM_SHELL	ist
D3P_SHELL_NY	float	D3P_NUM_SHELL	ist
D3P_SHELL_NXY	float	D3P_NUM_SHELL	ist
DELETION			
D3P_HAS_DELETION	bool	1	ist
D3P_SOLID_DELETION	float	D3P_NUM_SOLID	ist
D3P_TSHELL_DELETION	float	D3P_NUM_TSHELL	ist
D3P_SHELL_DELETION	float	D3P_NUM_SHELL	ist
D3P_BEAM_DELETION	float	D3P_NUM_BEAM	ist
SPH			
D3P_NUM_SPH	int	1	ignore
D3P_SPH_NODE_MAT	D3P_Sph	D3P_NUM_SPH	ignore
D3P_SPH_RADIUS	float	D3P_NUM_SPH	ist
D3P_SPH_PRESSURE	float	D3P_NUM_SPH	ist
D3P_SPH_STRESS	D3P_Tensor	D3P_NUM_SPH	ist
D3P_SPH_PLASTIC_STRAIN	float	D3P_NUM_SPH	ist
D3P_SPH_DENSITY	float	D3P_NUM_SPH	ist
D3P_SPH_INTERNAL_ENERGY	float	D3P_NUM_SPH	ist
D3P_SPH_NUMBER_OF_PARTICLE_NEIGHBORS	int	D3P_NUM_SPH	ist
D3P_SPH_STRAIN	D3P_Tensor	D3P_NUM_SPH	ist
D3P_SPH_MASS	float	D3P_NUM_SPH	ist
DES			
D3P_HAS_DES_DATA	bool	1	ignore
D3P_NUM_DES_DATA	int	1	ignore
D3P_NUM_DES_PART_IN_GEOM	int	1	ides if necessary
D3P_NUM_DES_ELEM_IN_GEOM	int	1	ides if necessary

D3P_NUM_DES_PART_IN_STATE	int	1	ides if necessary
D3P_NUM_DES_ELEM_IN_STATE	int	1	ides if necessary
D3P_NUM_DES_PART_VAR_IN_GEOM	int	1	ides if necessary
D3P_DES_PART_VAR_LIST_IN_GEOM	D3P_Var	D3P_NUM_DES_PART_VAR_IN_GEOM	ides if necessary
D3P_NUM_DES_ELEM_VAR_IN_GEOM	int	1	ides if necessary
D3P_DES_ELEM_VAR_LIST_IN_GEOM	D3P_Var	D3P_NUM_DES_ELEM_VAR_IN_GEOM	ides if necessary
D3P_NUM_DES_PART_VAR_IN_STATE	int	1	ides if necessary
D3P_DES_PART_VAR_LIST_IN_STATE	D3P_Var	D3P_NUM_DES_PART_VAR_IN_STATE	ides if necessary
D3P_NUM_DES_ELEM_VAR_IN_STATE	int	1	ides if necessary
D3P_DES_ELEM_VAR_LIST_IN_STATE	D3P_Var	D3P_NUM_DES_ELEM_VAR_IN_STATE	ides if necessary
D3P_DES_NODAL_MAT_RADIUS_MASS_INERTIA	D3P_Des	D3P_NUM_DES_ELEM_IN_GEOM	ides if necessary
D3P_DES_DATA_IN_STATE	int/float/vector/tensor..depends	D3P_NUM_DES_ELEM_IN_STATE	var_name, ist, ides if necessary
CPM			
D3P_HAS_CPM_DATA	bool	1	ignore
D3P_CPM_NUM_AIRBAGS	int	1	ignore
D3P_CPM_NUM_PARTICLES	int	1	ignore
D3P_CPM_NUM_GEOM_VAR	int	1	ignore
D3P_CPM_GEOM_VAR_LIST	D3P_Var	D3P_CPM_NUM_GEOM_VAR	ignore
D3P_CPM_GEOM_DATA	D3P_Var	D3P_CPM_NUM_GEOM_VAR	ignore

D3P_CPM_NUM_STATE_VAR	int	1	ignore
D3P_CPM_STATE_VAR_LIST	D3P_Var	D3P_CPM_NUM_STATE_VAR	ignore
D3P_CPM_STATE_DATA	int/float...depends	D3P_CPM_NUM_PARTICLES	var_name, ist
D3P_CPM_NUM_STATE_GEOM_VAR	int	1	ignore
D3P_CPM_STATE_GEOM_VAR_LIST	D3P_Var	D3P_CPM_NUM_STATE_GEOM_VAR	ignore
D3P_CPM_STATE_GEOM_DATA	int/float...depends	D3P_CPM_NUM_AIRBAGS	var_name, ist
Multisolver			
D3P_HAS_MS_DATA	bool	1	ignore
D3P_MS_NUM_DOMAINS	int	1	ignore
D3P_MS_DOMAIN_ID	int	1	index_multisolver
D3P_MS_DOMAIN_NAME	char	80	index_multisolver
D3P_MS_DOMAIN_VAR_NUM	int	1	index_multisolver
D3P_MS_DOMAIN_VARS_LIST	int	D3P_MS_DOMAIN_VAR_NUM	index_multisolver
D3P_MS_VAR_NAME	char	80	id_var_multisolver
D3P_MS_VAR_IS_VECTOR	bool	1	id_var_multisolver
D3P_MS_VAR_IS_SCALAR	bool	1	id_var_multisolver
D3P_MS_VAR_IS_TENSOR	bool	1	id_var_multisolver
D3P_MS_DOMAIN_VAR_LENGTH	int	1	ist, index_multisolver
D3P_MS_DOMAIN_IS_SOLID	bool	1	ist, index_multisolver

D3P_MS_DOMAIN_IS_SHELL	bool	1	ist, index_multi solver
D3P_MS_DOMAIN_IS_BEAM	bool	1	ist, index_multi solver
D3P_MS_DOMAIN_ELEM_NUM_IN_STATE	int	1	ist, index_multi solver
D3P_MS_SOLID_CONNECTIVITY_MAT_IN_STATE	D3P_Solid	D3P_MS_DOMAIN_ELEM_NUM_IN_STATE	ist, index_multi solver
D3P_MS_SHELL_CONNECTIVITY_MAT_IN_STATE	D3P_Shell	D3P_MS_DOMAIN_ELEM_NUM_IN_STATE	ist, index_multi solver
D3P_MS_BEAM_CONNECTIVITY_MAT_IN_STATE	D3P_Beam	D3P_MS_DOMAIN_ELEM_NUM_IN_STATE	ist, index_multi solver
D3P_MS_DOMAIN_NODE_NUM_IN_STATE	int	1	ist, index_multi solver
D3P_MS_DOMAIN_COORD_IN_STATE	D3P_Vector	D3P_MS_DOMAIN_NODE_NUM_IN_STATE	ist, index_multi solver
D3P_MS_DOMAIN_DATA_IN_STATE	float or D3P_Vector or D3P_Tensor	D3P_MS_DOMAIN_VAR_LENGTH	ist, index_multi solver,id_var_multisolver
D3P_MS_DOMAIN_DATA_IS_ON_STRUCTURE_ELEMENT	bool	1	index_multi solver
D3P_MS_DOMAIN_DATA_IS_ON_MS_NODE	bool	1	index_multi solver
D3P_MS_DOMAIN_DATA_IS_ON_MS_ELEMENT,	bool	1	index_multi solver
D3P_MS_DOMAIN_IS_FOLLOW_SURFACE_METHOD	bool	1	index_multi solver
D3P_MS_DOMAIN_NODE_NUM_ONSURFACE_IN_STATE	int	1	ist, index_multi

			solver
D3P_MS_DOMAIN_SURFACE_IDS_IN_STATE	int	D3P_MS_DOMAIN_NODE_NUM_ONSURFACE_IN_STATE	ist, index_multi solver

How to use

Sample1.py

Purpose: obtain resultant displacement for all the nodes and find maximum value.

3D scatterplot(x=shell_nodes_x, y=shell_nodes_y, z=shell_nodes_z, c=resultant displacement of shell nodes)

ist: last.

```
from lsreader import D3plotReader, DataType as dt
import os
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
from math import pow


d3plot = os.path.join(os.getcwd(), 'd3plot')
dr = D3plotReader(d3plot)


num_states = dr.get_data(dt.D3P_NUM_STATES)
nodes_init_coor = dr.get_data(
    dt.D3P_NODE_INITIAL_COORDINATES, ist=num_states-1
)
nodes_coor = dr.get_data(dt.D3P_NODE_COORDINATES, ist=num_states-1)


# obtain resultant displacement for all nodes and find maximum
nodes_res_disp = []
for i in range(nodes_coor.__len__()):
    disp_x = nodes_coor[i].x() - nodes_init_coor[i].x()
    disp_y = nodes_coor[i].y() - nodes_init_coor[i].y()
    disp_z = nodes_coor[i].z() - nodes_init_coor[i].z()
```

```
    tmp = pow(disp_x, 2) + pow(disp_y, 2) + pow(disp_z, 2)
    nodes_res_disp.append(pow(tmp, 0.5))

print(
"""
Maximum resultant displacement of nodes is: {0}, index is: {1}
""".format(
    max(nodes_res_disp), nodes_res_disp.index(max(nodes_res_disp))
    )
)

# nodes coordinates of shell elements when ist=last
shells = dr.get_data(dt.D3P_SHELL_CONNECTIVITY_MAT)
nodes_shell = []
for shell in shells:
    nodes_shell.append(shell.node(0))
    nodes_shell.append(shell.node(1))
    nodes_shell.append(shell.node(2))
    nodes_shell.append(shell.node(3))
nodes_shell = list(set(nodes_shell))
nodes_shell.sort()
nodes_x, nodes_y, nodes_z, res = [], [], [], []
for node_shell in nodes_shell:
    nodes_x.append(nodes_coor[node_shell-1].x())
    nodes_y.append(nodes_coor[node_shell-1].y())
    nodes_z.append(nodes_coor[node_shell-1].z())
    res.append(nodes_res_disp[node_shell-1])
```

```
# plotting
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1, projection='3d')
scat = ax.scatter3D(
    nodes_x, nodes_y, nodes_z, c=res, s=15,
)
fig.colorbar(scat, label='Resultant Displacement')
ax.set_zlim3d(-50, 50)
plt.show()
```

Sample2.py

Purpose: extract Variable data for Multisolver.

State: 2

```
import lsreader

from lsreader import D3plotReader

from lsreader import DataType as dt

from lsreader import D3P_Parameter as dp

import os


d3plot = os.path.join(os.getcwd(), 'd3plot')

dr = D3plotReader(d3plot)


has_ms_data = dr.get_data(dt.D3P_HAS_MS_DATA)

if not has_ms_data:

    print("No Multisolver Data")

num_ms_datasets = dr.get_data(dt.D3P_MS_NUM_DOMAINS)

for dataset in range(num_ms_datasets):

    domain_var_ids = dr.get_data(dt.D3P_MS_DOMAIN_VARS_LIST, index_multiso
lver=dataset)


    for var in range(domain_var_ids.__len__()):

        sizevar = dr.get_data(dt.D3P_MS_DOMAIN_VAR_LENGTH, index_multisolv
er=dataset, ist=2)

        is_scalar = dr.get_data(dt.D3P_MS_VAR_IS_SCALAR, id_var_multisolve
r=domain_var_ids[var])

        is_vector = dr.get_data(dt.D3P_MS_VAR_IS_VECTOR, id_var_multisolve
r=domain_var_ids[var])
```

```
        is_tensor = dr.get_data(dt.D3P_MS_VAR_IS_TENSOR, id_var_multisolve
r=domain_var_ids[var])

        p = dp()

        p.ist=2

        p.index_multisolver = dataset

        p.id_var_multisolver = domain_var_ids[var]

        if is_scalar:

            svalue = dr.get_data(dt.D3P_MS_DOMAIN_DATA_IN_STATE, p)

            print("Value type: scalar, value[0]={}".format(svalue[0]))

        if is_vector:

            vvalue = dr.get_data(dt.D3P_MS_DOMAIN_DATA_IN_STATE, p)

            print(

                   "Value type: vector, value[0].X()={}"

                   .format(vvalue[0].x())

             )

        if is_tensor:

            tvalue = dr.get_data(dt.D3P_MS_DOMAIN_DATA_IN_STATE, p)

            print(

                   "Value type: tensor, value[0].X()={}"

                   .format(tvalue[0].x())

             )
```

BinoutReader

API Functions

```
class BinoutReader():
    def __init__(self, path):
        pass
```

❖ Purpose: Constructor.

❖ Input: path: binout name.

❖ Return: BinoutReader object.

Example: `br = BinoutReader("binout/file/path")`

```
@staticmethod
def is_valid(path):
    pass
```

❖ Purpose: Check if the path is correct

❖ Input: path: binout name(full path).

❖ Return: True or False.

```
@staticmethod
def write(path, x_array, y_array):
    pass
```

❖ Purpose: Output the x_array and y_array to path.

❖ Input: path: binout name(full path).

x_array: The array of X direction.

y_array: The array of Y direction.

❖ Return: True.

```
def get_branch(self):
    pass
```

❖ Purpose: Get branches.

❖ Input: void.

❖ Return: The array of branches.

```
def set_branch(self, branch):
    pass
```

❖ Purpose: Set current branch.

❖ Input: branch: The name of the branch to set.

❖ Return: True.

```
def set_id(self, id, master):
    pass
```

❖ Purpose: Set current id.

❖ Input: id: The id to set. It can be string or integer.

master: choose master or slave. It can be ignored.

❖ Return: True.

```
def get_id(self):
    pass
```

❖ Purpose: Get ids.

❖ Input: void.

❖ Return: The array of ids.

```
def set_component(component):
    pass
```

❖ Purpose: Set current component.

❖ Input: branch: The name of the component to set.

❖ Return: True.

```
def get_component():
    pass
```

❖ Purpose: Get components.

❖ Input: void.

❖ Return: The array of components.

```
def get_x_array():
    pass
```

❖ Purpose: Get the array of X direction.

❖ Input: void.

❖ Return: The array of X direction.

```
def get_y_array():
    pass
```

❖ Purpose: Get the array of Y direction.

❖ Input: void.

❖ Return: The array of Y direction.

How to use

Sample1.py

Purpose: obtain branches and component, and get x_array, y_array.

Branch: nodout.

Component: x_acceleration.

Id: 1787

Ouput: nodoutPy.dat

```
br = BinoutReader(data_path)


res = BinoutReader.is_valid(data_path)
print(res)


branches = br.get_branch()
for branch in branches:
    print(branch, end=',')


br.set_branch('nodout')
br.set_id(1787)
br.set_component('x_acceleration')
x_array = br.get_x_array()
y_array = br.get_y_array()
out_path = os.path.join(cwd, 'nodoutPy.dat')
BinoutReader.write(out_path, x_array, y_array)
```

C

D3plotReader

API Functions

```
char* D3P_Open (const char* filename);
```

- ❖ Purpose: Open the d3plot file.
- ❖ Input: filename - d3plot name.
- ❖ Return: The pointer to the d3plot.

```
int   D3P_Read (char* handle, enum D3P_DataType type, char* value,
_D3P_Parameter param);
```

- ❖ Purpose: Get value of the special data varialbe.
- ❖ Input: handle - the pointer to the d3plot

  type - enum the data variables's name in d3plot.

  value - store the return value.

  param - structure of description which is the advance setting for getting special data in d3plot.
- ❖ Return: void.

```
void  D3P_Close(char* handle);
```

- ❖ Purpose: Close the d3plot file.
- ❖ Input: handle - the pointer to the d3plot
- ❖ Return: void

_D3P _Parameter

```
typedef struct _D3P_Parameter_
{
	int ist;
	int ipt;
	int ipart;
	int i_rigid_wall;
	int ides;
	int ihv;
	int index_multisolver;
	int id_var_multisolver;
	const char* var_name;
} _D3P_Parameter;
```

1. **ist**: Specify the state number, starting from 0, as follows:

```
_D3P_Parameter dp;
dp.ist = 11;
dp.ipt = 0;
D3P_Read(handle, D3P_BEAM_AXIAL_STRESS, (char*)beam_axial_stress, dp);
```

2. **ipt**: Specify the integration point, ranging in [0, MAXINT), as follows:

```
_D3P_Parameter dp;
dp.ipt = 0;
...
```

3. **ipart**: Specify the index of part, starting from 0, as follows:

```
_D3P_Parameter dp;
dp.ipart = 0;
...
```

4. **i_rigid_wall**: Specify the index of rigid wall, starting from 0, as follows:

```
_D3P_Parameter dp;
```

```
dp.i_rigid_wall = 0;

...
```

5. **ides**: Specify the index of the des data, starting from 0, as follows:

```
_D3P_Parameter dp;

dp.ides = 0;

...
```

6. **ihv**: Specify the index of history variables, starting from 0, as follows:

```
_D3P_Parameter dp;

dp.ihv = 0;

...
```

7. **index_multisolver**: Specify the index of the multisolver domain, start from 0 and default is 0 also:

```
_D3P_Parameter dp;

dp.index_multisolver = 0;

...
```

8. **id_var_multisolver**: Specify the index of the multisolver var, start from 0 and default is 0 also:

```
_D3P_Parameter dp;

dp.id_var_multisolver = 0;

...
```

9. **var_name**: Specify name of output variables, currently used by DES and CPM data, default is empty

```
_D3P_Parameter dp;

dp.var_name = "";

...
```

DataType

```
typedef struct _D3P_Vector_
{
	float v[3];
} _D3P_Vector;

typedef struct D3P_VectorDouble_
{
	double v[3];
} _D3P_VectorDouble;

typedef struct _D3P_Tensor_
{
	float t[6];
} _D3P_Tensor;

typedef struct _D3P_Solid_
{
	int conn[10];
	int mat;
} _D3P_Solid;

typedef struct _D3P_Tshell_
{
	int conn[10];
	int mat;
} _D3P_Tshell;

typedef struct _D3P_Beam_
{
```

```
        int conn[2];
        int third;
        int w_int;
        int h_int;
        int mat;
} _D3P_Beam;

typedef struct _D3P_Shell_
{
        int conn[4];
        int mat;
} _D3P_Shell;

typedef struct _D3P_Sph
{
        int id;
        unsigned int mat;
} _D3P_Sph;

typedef struct _D3P_Var_
{
    int type;
    char name[8];
} _D3P_Var;

typedef struct _D3P_Des_
{
        int id;
        int mat;
        float radius;
        float mass;
```

```
        float inertia;
} _D3P_Des;

typedef struct _D3P_AirbagInfo_
{
        int bagid;
        int startn;
        int npart;
        int ngas;
        int nchamber;
} _D3P_AirbagInfo;
```

---

name	conversion	length	parameters
D3P_NUM_STATES	int	1	ignore
D3P_TIMES	float	D3P_NUM_STATES	ignore
D3P_TITLE	Char		ignore
Global			
D3P_GLOBAL_KINETIC_ENERGY	float	1	ist
D3P_GLOBAL_INTERNAL_ENERGY	float	1	ist
D3P_GLOBAL_TOTAL_ENERGY	float	1	ist
D3P_GLOBAL_VELOCITY	_D3P_Vector	1	ist
Part			
D3P_NUM_PARTS	int	1	ignore
D3P_PART_IDS	int	D3P_NUM_PARTS	ignore
D3P_PART_NAME	char	80	ipart
D3P_PART_INTERNAL_ENERGY	float	1	ist, ipart
D3P_PART_KINETIC_ENERGY	float	1	ist, ipart
D3P_PART_VELOCITY	_D3P_Vector	1	ist, ipart
D3P_PART_MASS	float	1	ist, ipart
D3P_PART_HOURGLASS	float	1	ist, ipart
RIGID WALL			
D3P_NUM_RIGID_WALL	int	1	ignore
D3P_RIGID_WALL_FORCE	float	1	ist, i_rigid_wall
D3P_RIGID_WALL_POSITION	_D3P_Vector	1	ist,

			i_rigid_wall
NODE			
D3P_NUM_NODES	int	1	ignore
D3P_NODE_INITIAL_COORDINATES	_D3P_Vector	D3P_NUM_NODES	ignore
D3P_NODE_IDS	int	D3P_NUM_NODES	ignore
D3P_NODE_TEMPERATURE	float	D3P_NUM_NODES	ist
D3P_NODE_COORDINATES	_D3P_Vector	D3P_NUM_NODES	ist
D3P_NODE_VELOCITIES	_D3P_Vector	D3P_NUM_NODES	ist
D3P_NODE_ACCELERATIONS	_D3P_Vector	D3P_NUM_NODES	ist
D3P_NODE_COORDINATES_DOUBLE	_D3P_VectorDouble	D3P_NUM_NODES	ist
D3P_NODE_VELOCITIES_DOUBLE	_D3P_VectorDouble	D3P_NUM_NODES	ist
D3P_NODE_ACCELERATIONS_DOUBLE	_D3P_VectorDouble	D3P_NUM_NODES	ist
SOLID			
D3P_NUM_SOLID	int	1	ignore
D3P_SOLID_MAXINT	int	1	ignore
D3P_SOLID_CONNECTIVITY_MAT	_D3P_Solid	D3P_NUM_SOLID	ignore
D3P_SOLID_IDS	int	D3P_NUM_SOLID	ignore
D3P_SOLID_STRESS	_D3P_Tensor	D3P_NUM_SOLID	ist, ipt if necessary
D3P_SOLID_EFFECTIVE_PLASTIC_STRAIN	float	D3P_NUM_SOLID	ist, ipt if necessary
D3P_SOLID_STRAIN	_D3P_Tensor	D3P_NUM_SOLID	ist, ipt if necessary
D3P_SOLID_HISTORY_VAR	float	D3P_NUM_SOLID	ist, ipt, ihv
TSHELL			
D3P_NUM_TSHELL	int	1	ignore
D3P_TSHELL_MAXINT	int	1	ignore
D3P_TSHELL_CONNECTIVITY_MAT	_D3P_Tshell	D3P_NUM_TSHELL	ignore
D3P_TSHELL_IDS	int	D3P_NUM_TSHELL	ignore

D3P_TSHELL_STRESS	_D3P_Tensor	D3P_NUM_TSHELL	ist, ipt
D3P_TSHELL_EFFECTIVE_PLASTIC_STRAIN	float	D3P_NUM_TSHELL	ist, ipt
D3P_TSHELL_STRAIN	_D3P_Tensor	D3P_NUM_TSHELL	ist, ipt
D3P_TSHELL_HISTORY_VAR	float	D3P_NUM_TSHELL	ist, ipt, ihv
BEAM			
D3P_NUM_BEAM	int	1	ignore
D3P_BEAM_MAXINT	int	1	ignore
D3P_BEAM_CONNECTIVITY_THIRD_MAT	_D3P_Beam	D3P_NUM_BEAM	ignore
D3P_BEAM_IDS	int	D3P_NUM_BEAM	ignore
D3P_BEAM_AXIAL_FORCE	float	D3P_NUM_BEAM	ist
D3P_BEAM_S_SHEAR_RESULTANT	float	D3P_NUM_BEAM	ist
D3P_BEAM_T_SHEAR_RESULTANT	float	D3P_NUM_BEAM	ist
D3P_BEAM_S_BENDING_MOMENT	float	D3P_NUM_BEAM	ist
D3P_BEAM_T_BENDING_MOMENT	float	D3P_NUM_BEAM	ist
D3P_BEAM_TORSIONAL_RESULTANT	float	D3P_NUM_BEAM	ist
D3P_BEAM_RS_SHEAR_STRESS	float	D3P_NUM_BEAM	ist, ipt
D3P_BEAM_TR_SHEAR_STRESS	float	D3P_NUM_BEAM	ist, ipt
D3P_BEAM_AXIAL_STRESS	float	D3P_NUM_BEAM	ist, ipt
D3P_BEAM_AXIAL_PLASTIC_STRAIN	float	D3P_NUM_BEAM	ist, ipt
D3P_BEAM_AXIAL_STRAIN	float	D3P_NUM_BEAM	ist, ipt
D3P_BEAM_HISTORY_VAR	float	D3P_NUM_BEAM	ist, ipt, ihv
SHELL			
D3P_NUM_SHELL	int	1	ignore
D3P_SHELL_MAXINT	int	1	ignore
D3P_SHELL_CONNECTIVITY_MAT	_D3P_Shell	D3P_NUM_SHELL	ignore
D3P_SHELL_IDS	int	D3P_NUM_SHELL	ignore
D3P_SHELL_STRESS	_D3P_Tensor	D3P_NUM_SHELL	ist, ipt

D3P_SHELL_EFFECTIVE_PLASTIC_STRAIN	float	D3P_NUM_SHELL	ist, ipt
D3P_SHELL_STRAIN	_D3P_Tensor	D3P_NUM_SHELL	ist, ipt
D3P_SHELL_HISTORY_VAR	float	D3P_NUM_SHELL	ist, ipt, ihv
D3P_SHELL_MX	float	D3P_NUM_SHELL	ist
D3P_SHELL_MY	float	D3P_NUM_SHELL	ist
D3P_SHELL_MXY	float	D3P_NUM_SHELL	ist
D3P_SHELL_QX	float	D3P_NUM_SHELL	ist
D3P_SHELL_QY	float	D3P_NUM_SHELL	ist
D3P_SHELL_NX	float	D3P_NUM_SHELL	ist
D3P_SHELL_NY	float	D3P_NUM_SHELL	ist
D3P_SHELL_NXY	float	D3P_NUM_SHELL	ist
DELETION			
D3P_HAS_DELETION	bool	1	ist
D3P_SOLID_DELETION	float	D3P_NUM_SOLID	ist
D3P_TSHELL_DELETION	float	D3P_NUM_TSHELL	ist
D3P_SHELL_DELETION	float	D3P_NUM_SHELL	ist
D3P_BEAM_DELETION	float	D3P_NUM_BEAM	ist
SPH			
D3P_NUM_SPH	int	1	ignore
D3P_SPH_NODE_MAT	_D3P_Sph	D3P_NUM_SPH	ignore
D3P_SPH_RADIUS	float	D3P_NUM_SPH	ist
D3P_SPH_PRESSURE	float	D3P_NUM_SPH	ist
D3P_SPH_STRESS	_D3P_Tensor	D3P_NUM_SPH	ist
D3P_SPH_PLASTIC_STRAIN	float	D3P_NUM_SPH	ist
D3P_SPH_DENSITY	float	D3P_NUM_SPH	ist
D3P_SPH_INTERNAL_ENERGY	float	D3P_NUM_SPH	ist
D3P_SPH_NUMBER_OF_PARTICLE_NEIGHBORS	int	D3P_NUM_SPH	ist
D3P_SPH_STRAIN	_D3P_Tensor	D3P_NUM_SPH	ist

D3P_SPH_MASS	float	D3P_NUM_SPH	ist
DES			
D3P_HAS_DES_DATA	bool	1	ignore
D3P_NUM_DES_DATA	int	1	ignore
D3P_NUM_DES_PART_IN_GEOM	int	1	ides if necessary
D3P_NUM_DES_ELEM_IN_GEOM	int	1	ides if necessary
D3P_NUM_DES_PART_IN_STATE	int	1	ides if necessary
D3P_NUM_DES_ELEM_IN_STATE	int	1	ides if necessary
D3P_NUM_DES_PART_VAR_IN_GEOM	int	1	ides if necessary
D3P_DES_PART_VAR_LIST_IN_GEOM	_D3P_Var	D3P_NUM_DES_PART_VAR_IN_GEOM	ides if necessary
D3P_NUM_DES_ELEM_VAR_IN_GEOM	int	1	ides if necessary
D3P_DES_ELEM_VAR_LIST_IN_GEOM	_D3P_Var	D3P_NUM_DES_ELEM_VAR_IN_GEOM	ides if necessary
D3P_NUM_DES_PART_VAR_IN_STATE	int	1	ides if necessary
D3P_DES_PART_VAR_LIST_IN_STATE	_D3P_Var	D3P_NUM_DES_PART_VAR_IN_STATE	ides if necessary
D3P_NUM_DES_ELEM_VAR_IN_STATE	int	1	ides if necessary
D3P_DES_ELEM_VAR_LIST_IN_STATE	_D3P_Var	D3P_NUM_DES_ELEM_VAR_IN_STATE	ides if necessary
D3P_DES_NODAL_MAT_RADIUS_MASS_INERTIA	_D3P_Des	D3P_NUM_DES_ELEM_IN_GEOM	ides if necessary
D3P_DES_DATA_IN_STATE	int/float/vector/tensor..depends	D3P_NUM_DES_ELEM_IN_STATE	var_name, ist, ides if necessary
CPM			
D3P_HAS_CPM_DATA	bool	1	ignore

D3P_CPM_NUM_AIRBAGS	int	1	ignore
D3P_CPM_NUM_PARTICLES	int	1	ignore
D3P_CPM_NUM_GEOM_VAR	int	1	ignore
D3P_CPM_GEOM_VAR_LIST	_D3P_Var	D3P_CPM_NUM_GEOM_VAR	ignore
D3P_CPM_GEOM_DATA	_D3P_Var	D3P_CPM_NUM_GEOM_VAR	ignore
D3P_CPM_NUM_STATE_VAR	int	1	ignore
D3P_CPM_STATE_VAR_LIST	_D3P_Var	D3P_CPM_NUM_STATE_VAR	ignore
D3P_CPM_STATE_DATA	int/float...depends	D3P_CPM_NUM_PARTICLES	var_name, ist
D3P_CPM_NUM_STATE_GEOM_VAR	int	1	ignore
D3P_CPM_STATE_GEOM_VAR_LIST	_D3P_Var	D3P_CPM_NUM_STATE_GEOM_VAR	ignore
D3P_CPM_STATE_GEOM_DATA	int/float...depends	D3P_CPM_NUM_AIRBAGS	var_name, ist
Multisolver			
D3P_HAS_MS_DATA	bool	1	ignore
D3P_MS_NUM_DOMAINS	int	1	ignore
D3P_MS_DOMAIN_ID	int	1	index_multisolver
D3P_MS_DOMAIN_NAME	char	80	index_multisolver
D3P_MS_DOMAIN_VAR_NUM	int	1	index_multisolver
D3P_MS_DOMAIN_VARS_LIST	int	D3P_MS_DOMAIN_VAR_NUM	index_multisolver
D3P_MS_VAR_NAME	char	80	id_var_multisolver
D3P_MS_VAR_IS_VECTOR	bool	1	id_var_multisolver
D3P_MS_VAR_IS_SCALAR	bool	1	id_var_multisolver

D3P_MS_VAR_IS_TENSOR	bool	1	id_var_multisolver
D3P_MS_DOMAIN_VAR_LENGTH	int	1	ist, index_multisolver
D3P_MS_DOMAIN_IS_SOLID	bool	1	ist, index_multisolver
D3P_MS_DOMAIN_IS_SHELL	bool	1	ist, index_multisolver
D3P_MS_DOMAIN_IS_BEAM	bool	1	ist, index_multisolver
D3P_MS_DOMAIN_ELEM_NUM_IN_STATE	int	1	ist, index_multisolver
D3P_MS_SOLID_CONNECTIVITY_MAT_IN_STATE	_D3P_Solid	D3P_MS_DOMAIN_ELEM_NUM_IN_STATE	ist, index_multisolver
D3P_MS_SHELL_CONNECTIVITY_MAT_IN_STATE	_D3P_Shell	D3P_MS_DOMAIN_ELEM_NUM_IN_STATE	ist, index_multisolver
D3P_MS_BEAM_CONNECTIVITY_MAT_IN_STATE	_D3P_Beam	D3P_MS_DOMAIN_ELEM_NUM_IN_STATE	ist, index_multisolver
D3P_MS_DOMAIN_NODE_NUM_IN_STATE	int	1	ist, index_multisolver
D3P_MS_DOMAIN_COORD_IN_STATE	D3P_Vector	D3P_MS_DOMAIN_NODE_NUM_IN_STATE	ist, index_multisolver
D3P_MS_DOMAIN_DATA_IN_STATE	float or D3P_Vector or D3P_Tensor	D3P_MS_DOMAIN_VAR_LENGTH	ist, index_multisolver,id_var_multisolver
D3P_MS_DOMAIN_DATA_IS_ON_STRUCTURE_ELEMENT	bool	1	index_multisolver
D3P_MS_DOMAIN_DATA_IS_ON_MS_NODE	bool	1	index_multisolver

D3P_MS_DOMAIN_DATA_IS_ON_MS_ELEMENT,	bool	1	index_multisolver
D3P_MS_DOMAIN_IS_FOLLOW_SURFACE_METHOD	bool	1	index_multisolver
D3P_MS_DOMAIN_NODE_NUM_ONSURFACE_IN_STATE	int	1	ist, index_multisolver
D3P_MS_DOMAIN_SURFACE_IDS_IN_STATE	int	D3P_MS_DOMAIN_NODE_NUM_ONSURFACE_IN_STATE	ist, index_multisolver

How to use

Sample1.c

Purpose: obtain resultant displacement for all the nodes and find maximum value.

3D scatterplot(x=shell_nodes_x, y=shell_nodes_y, z=shell_nodes_z, c=resultant displacement of shell nodes)

ist: last.

```c
#include "../config.h"      /* define DATA_PATH_1 "d3plot/path" */
#include "../d3plotreaderwrapperc.h"

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main() {
	char d3plot[1024] = { 0 };
	char* handle;
	int i;

	int num_nodes = 0;
	int num_states = 0;
	int index = 0;
	_D3P_Vector* node_ini_coor = NULL;
	_D3P_Vector* node_coor = NULL;
	float* node_res_disp = NULL;
	float disp_x, disp_y, disp_z, tmp, max;
	_D3P_Parameter param;
	param.ides = -1;
	param.id_var_multisolver = -1;
	param.ihv = -1;
```

```
param.index_multisolver = -1;
param.ipart = -1;
param.ipt = -1;
param.ist = -1;
param.i_rigid_wall = -1;
param.var_name = "";

strcpy(d3plot, DATA_PATH_1);
handle = D3P_Open(d3plot);

D3P_Read(handle, D3P_NUM_NODES, (char*)& num_nodes, param);
D3P_Read(handle, D3P_NUM_STATES, (char*)& num_states, param);



node_ini_coor = (
	(_D3P_Vector*)malloc(num_nodes * sizeof(_D3P_Vector))
	);
node_coor = (_D3P_Vector*)malloc(num_nodes * sizeof(_D3P_Vector));
D3P_Read(
	handle, D3P_NODE_INITIAL_COORDINATES,
	(char*)node_ini_coor, param
);
param.ist = num_states - 1;
D3P_Read(handle, D3P_NODE_COORDINATES, (char*)node_coor, param);

D3P_Close(handle);

/*
	obtain resultant displacement for all nodes
	and find maximum value.
*/
```

```
	node_res_disp = (float*)malloc(num_nodes * sizeof(float));
	for (i = 0; i < num_nodes; i++) {
		disp_x = node_coor[i].v[0] - node_ini_coor[i].v[0];
		disp_y = node_coor[i].v[1] - node_ini_coor[i].v[1];
		disp_z = node_coor[i].v[2] - node_ini_coor[i].v[2];
		tmp = pow(disp_x, 2) + pow(disp_y, 2) + pow(disp_z, 2);
		node_res_disp[i] = pow(tmp, 0.5);
	}

	max = node_res_disp[0];
	for (i = 0; i < num_nodes; i++) {
		if (max < node_res_disp[i]) {
			max = node_res_disp[i];
			index = i;
		}
	}

	printf(
		"Maximum resultant displacement of nodes is: %f,"
		"index is %d", max, index
	);

	free(node_ini_coor);
	free(node_coor);
	free(node_res_disp);
};
```

Sample2.c

Purpose: extract Variable data for Multisolver.

State: 2

```c
#include "../config.h"      /* define DATA_PATH_3 "d3plot/path" */
#include "../d3plotreaderwrapperc.h"

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <stdbool.h>

int main() {
        char d3plot[1024] = { 0 };
        char* handle_ms = NULL;
        bool has_ms_data = false;
        int num_ms_datasets = 0;
        int* domain_var_ids = NULL;
        int dataset = 0;
        int domain_var_num = 0;
        int var = 0;
        int sizevar = 0;
        int i;
        bool is_scalar = false;
        bool is_vector = false;
        bool is_tensor = false;
        float* fvalue = NULL;
        float maxvalue = -1.0e20;
        _D3P_Vector* vvalue = NULL;
```

```
_D3P_Tensor* tvalue = NULL;
_D3P_Parameter dp;
dp.ides = -1;
dp.id_var_multisolver = -1;
dp.ihv = -1;
dp.index_multisolver = -1;
dp.ipart = -1;
dp.ipt = -1;
dp.ist = -1;
dp.i_rigid_wall = -1;
dp.var_name = "";

strcpy(d3plot, DATA_PATH_3);
handle_ms = D3P_Open(d3plot);

D3P_Read(handle_ms, D3P_HAS_MS_DATA, (char*)& has_ms_data, dp);
if (!has_ms_data)
{
	printf("No Multisolver Data\n");
}

D3P_Read(
	handle_ms, D3P_MS_NUM_DOMAINS, (char*)& num_ms_datasets, dp
);

for (dataset; dataset < num_ms_datasets; dataset++)
{
	dp.index_multisolver = dataset;
	D3P_Read(
		handle_ms, D3P_MS_DOMAIN_ID, (char*)& domain_var_num, dp
	);
```

```
domain_var_ids = (int*)malloc(domain_var_num * sizeof(int));
D3P_Read(
        handle_ms, D3P_MS_DOMAIN_VARS_LIST,
        (char*)domain_var_ids, dp
);

dp.ist = 2;
var = 0;
for (var; var < domain_var_num; var++)
{
        dp.id_var_multisolver = domain_var_ids[var];

        sizevar = 0;
        D3P_Read(
                handle_ms, D3P_MS_DOMAIN_VAR_LENGTH,
                (char*)& sizevar, dp
        );

        D3P_Read(
                handle_ms, D3P_MS_VAR_IS_SCALAR,
                (char*)& is_scalar, dp
        );
        D3P_Read(
                handle_ms, D3P_MS_VAR_IS_VECTOR,
                (char*)& is_vector, dp
        );
        D3P_Read(
                handle_ms, D3P_MS_VAR_IS_TENSOR,
                (char*)& is_tensor, dp
        );
```

```
if (is_scalar)
{
        fvalue = (float*)malloc(sizevar * sizeof(float));
        D3P_Read(
                handle_ms, D3P_MS_DOMAIN_DATA_IN_STATE,
                (char*)fvalue, dp
        );
        printf("Value type: scalar, value[0]=%f\n",
                fvalue[0]
        );
        free(fvalue);
        fvalue = NULL;
}
if (is_vector)
{
        vvalue = (_D3P_Vector*)malloc(
                sizevar * sizeof(_D3P_Vector)
        );
        D3P_Read(
                handle_ms, D3P_MS_DOMAIN_DATA_IN_STATE,
                (char*)vvalue, dp
        );
        printf("Value type: vector, value[0].X()=%f\n",
                vvalue[0].v[0]
        );
        free(vvalue);
        vvalue = NULL;
}
if (is_tensor)
{
        tvalue = (_D3P_Tensor*)malloc(
```

```
                                sizevar * sizeof(_D3P_Tensor)
                        );
                        D3P_Read(
                                handle_ms, D3P_MS_DOMAIN_DATA_IN_STATE,
                                (char*)tvalue, dp
                        );
                        printf("Value type: tensor, value[0].X()=%f\n",
                                tvalue[0].t[0]
                        );
                        free(tvalue);
                        tvalue = NULL;
                }
        }
    }
    free(domain_var_ids);
    domain_var_ids = NULL;

    D3P_Close(handle_ms);
}
```

BinoutReader

API Functions

```
char*  Binout_Open(const char* filename);
```

- ❖ Purpose: Open the binout file.
- ❖ Input: filename - binout name.
- ❖ Return: The pointer to the binout.

```
int    Binout_GetNumberOfBranch(char* handle, size_t* max);
```

- ❖ Purpose: Get number and maximum string size of branches.
- ❖ Input: handle - the pointer to the binout.
  max - store the return maximum string size value.
- ❖ Return: The number of strings.

```
int    Binout_GetBranch(char* handle, char* branches[]);
```

- ❖ Purpose: Get branches.
- ❖ Input: handle - the pointer to the binout.
  branches- store the return branches value.
- ❖ Return: 1 or 0.

```
int    Binout_GetNumberOfComponent(char* handle, size_t* max);
```

- ❖ Purpose: Get number and maximum string size of components.
- ❖ Input: handle - the pointer to the binout.
  max - store the return maximum string size value.
- ❖ Return: The number of strings.

```
int    Binout_GetComponent(char* handle, char* components[]);
```

- ❖ Purpose: Get components.

❖ Input: handle - the pointer to the binout.

branches- store the return components value.

❖ Return: 1 or 0.

```
int     Binout_GetNumberOfXArray(char* handle);
int     Binout_GetNumberOfYArray(char* handle);
```

❖ Purpose: Get number of array(x, y).

❖ Input: handle - the pointer to the binout.

max - store the return number value.

❖ Return: The number of array.

```
int     Binout_GetXArray(char* handle, double* xArray);
int     Binout_GetYArray(char* handle, double* yArray);
```

❖ Purpose: Get array(x, y).

❖ Input: handle - the pointer to the binout.

branches- store the return array value(x, y).

❖ Return: 1 or 0.

```
int     Binout_GetNumberOfId(char* handle);
```

❖ Purpose: Get number of ids.

❖ Input: handle - the pointer to the binout.

max - store the return number value.

❖ Return: The number of ids.

```
int     Binout_GetId(char* handle, unsigned int* id);
```

❖ Purpose: Get array of ids.

- Input: handle - the pointer to the binout.

  branches- store the return array value.
- Return: 1 or 0.

```
int    Binout_SetBranch(const char* handle, const char* branch);
```

- Purpose: Set current branch.
- Input: handle - the pointer to the binout

  branch: The name of the branch to set.
- Return: 1 or 0.

```
int    Binout_SetStrId(const char* handle, const char* id);
```

- Purpose: Set current id.
- Input: handle - the pointer to the binout

  id: The id to set. It is string.
- Return: 1 or 0.

```
int    Binout_SetId(const char* handle, unsigned int id);
```

- Purpose: Set current id.
- Input: handle - the pointer to the binout

  id: The id to set. It is integer.
- Return: 1 or 0.

```
int    Binout_SetIdMasterSlave(const char* handle, unsigned int
id, int master);
```

- Purpose: Set current id.
- Input: handle - the pointer to the binout.

  id: The id to set. It is integer.

master: Choose master(1) or slave(0).

❖ Return: 1 or 0.

```
int    Binout_SetComponent(const char* handle, const char* comp);
```

❖ Purpose: Set current component.

❖ Input: handle - the pointer to the binout.

comp: The component to set.

❖ Return: 1 or 0.

```
int    Binout_IsValid(const char* filename);
```

❖ Purpose: Check if the path is correct

❖ Input: filename: binout name(full path).

❖ Return: 1or 0.

```
int    Binout_Write(const char* filename, double* x_array,
double* y_array, int size);
```

❖ Purpose: Output the x_array and y_array to path.

❖ Input: filename: binout name(full path).

x_array: The array of X direction.

y_array: The array of Y direction.

❖ Return: 1.

```
void   Binout_Close(char* handle);
```

❖ Purpose: Close the binout file.

❖ Input: handle - the pointer to the binout.

❖ Return: void.

How to use

Sample1.c

Purpose: obtain branches and component, and get x_array, y_array.

Branch: nodout.

Component: x_acceleration.

Id: 1787

Ouput: nodoutC.dat

```
int main()
{
	char binout_files[1024];
	char out_path[1024];
	char* handle_binout = NULL;
	char** branches = NULL;
	size_t max_branch = 0;
	unsigned int num_branch = 0;
	unsigned int num_id = 0;
	unsigned int num_array = 0;
	double* x_array = NULL;
	double* y_array = NULL;
	unsigned int* ids = NULL;
	//=====================BinoutReader=====================
	strcpy(binout_files, DATA_PATH_BINOUT);
	strcpy(out_path, OUTPUT_PATH_C);

	printf("%d\n", Binout_IsValid(binout_files));

	handle_binout = Binout_Open(binout_files);

	if (!handle_binout)
		goto cleanup_binout;
```

```
num_branch = Binout_GetSizeOfBranch(handle_binout, &max_branch);
branches = (char**)calloc(num_branch, sizeof(char*));
if (!branches)
	goto cleanup_binout;
for (i = 0; i < num_branch; i++)
{
	branches[i] = (char*)malloc(max_branch);
	if (branches[i])
		memset(branches[i], 0, max_branch);
	else
		goto cleanup_binout;
}
if (!Binout_GetBranch(handle_binout, branches))
	goto cleanup_binout;
printf("Branches: \n");
for (i = 0; i < num_branch; i++)
{
	printf("%s,", branches[i]);
}
printf("\n");

Binout_SetBranch(handle_binout, "nodout");
num_id = Binout_GetSizeOfId(handle_binout);

ids = (unsigned int*)malloc(sizeof(unsigned int) * num_id);
if (!ids)
	goto cleanup_binout;
memset(ids, 0, sizeof(unsigned int) * num_id);

if (!Binout_GetId(handle_binout, ids))
```

```
            goto cleanup_binout;
        printf("ids:\n");
        for (i = 0; i < num_id; i++)
        {
            printf("%d,", ids[i]);
        }
        printf("\n");

        Binout_SetId(handle_binout, 1787);
        Binout_SetComponent(handle_binout, "x_acceleration");

        num_array = Binout_GetSizeOfXArray(handle_binout);
        x_array = (double*)malloc(sizeof(double) * num_array);
        y_array = (double*)malloc(sizeof(double) * num_array);
        if (!x_array || !y_array)
            goto cleanup_binout;
        memset(x_array, 0, sizeof(double) * num_array);
        memset(y_array, 0, sizeof(double) * num_array);

        Binout_GetXArray(handle_binout, x_array);
        Binout_GetYArray(handle_binout, y_array);

        Binout_Write(out_path, x_array, y_array, num_array);

cleanup_binout:
        if (handle_binout)
        {
            Binout_Close(handle_binout);
            handle_binout = NULL;
        }
        if (branches)
```

```
        {
                for (i = 0; i < num_branch; i++)
                {
                        if (branches[i])
                        {
                                free(branches[i]);
                                branches[i] = NULL;
                        }
                }
                free(branches);
                branches = NULL;
        }
        if (x_array)
        {
                free(x_array);
                x_array = NULL;
        }
        if (y_array)
        {
                free(y_array);
                y_array = NULL;
        }
};
```