CATima library manual {#mainpage}

Compiling and Instalation

See README.md for details.

Units

The following units are used for input and outputs:

- projectile energy - MeV/u
- density - g/cm^3
- material thickness - g/cm^2
- material length - cm
- angle - rad
- time of flight - ns

Projectile

The **Projectile** class is used to store projectile data. Each projectile must provide A,Z combination, additionally charge state can be set as well. The example of projectile definition:

```
catima::Projectile p1(12,6); //12C projectile
catima::Projectile p2(12,6,5); //12C(5+) projectile
```

to set the energy of the projectile in MeV/u units:

```
p1.T = 1000.0;
p2(1000.0);
```

Material Definition

The material is defined and stored in **Material** class. The class stores the atom constituents, thickness and density of the material etc.

There are 2 ways to define materials: specifying all elements in constructor or using add_element(double, int, double) function. The example of water definition:

```
catima::Material water1({
        {1,1,2}, // 2 atoms of 1H
        {16,8,1} // 1 atom of 16O
        });
water1.density(1.0);
water1.thickness(2.0);

catima::Material water2;
water2.add_element(1,1,2);
water2.add_element(16,8,1);
water2.density(1.0).thickness(2.0);
```

If mass number is equal to 0, the mass number of the element is taken as atomic weight of the element with provided Z Other methods which can be used for Material:

```
double den = water2.density(); //den equals 1.0
double th = water2.thickness(); //th equals 2.0
double molar_mass = water2.M(); // get molar mass of
water
int ncomp = water2.ncomponents(); // ncomp equals 2
```

predefined materials

If the library is compiled with predefined materials database, the Material can be retrieved from the database as:

```
using namespace catimal
Material water = get_material(material::WATER);
Material graphite = get_material(6);
```

currently all meterial up to Z=98 are predefined plus compounds from MOCADI:

```
        enum material{
            PLASTIC = 201,
            AIR = 202,
            CH2,
            LH2,
            LD2,
            WATER,
            DIAMOND,
            GLASS,
            ALMG3,
            ARCO2_30,
            CF4,
            ISOBUTANE,
            KAPTON,
            MYLAR,
            NAF,
            P10,
            POLYOLEFIN,
            CMO2,
            SUPRASIL,
            HAVAR
        };
```

Calculation

To calculate all observable following function can be used:

```
Result calculate(Projectile &p, const Material &t,
double T, Config c)
Result calculate(Projectile &p, const Material &t,
Config c)
```

Both function returns structure ***Result*** which contains all calculated variables.

```
    struct Result{
        double Ein=0.0;
        double Eout=0.0;
        double Eloss = 0.0;
        double range=0.0;
        double dEdxi=0.0;
        double dEdxo=0.0;
        double sigma_E=0.0;
        double sigma_a=0.0;
        double sigma_r=0.0;
        double tof=0.0;
    };
```

If one is interested only in one of the variable, the following function can be used:

```
double dedx(Projectile &p, double T, const Material
&t, Config c=default_config);
double domega2dx(Projectile &p, double T, const
Material &t, Config c=default_config);
double range(Projectile &p, double T, const Material
&t, Config c=default_config);
double range_straggling(Projectile &p, double T,
```

```
const Material &t, Config c=default_config);
double angular_straggling(Projectile &p, double T,
const Material &t, Config c=default_config);
\end{verbatim}
```

Example calculation:

```
...
double T=1000;
auto result = catima::calculate(p1(1000),water1);
cout<<"T "<<T<<", dEdx = "<<result.dEdxi<<"
MeV/g/cm2"<<", range = "<<result.range<<" g/cm2"
<<endl;
```

Multilayer Material

The layers of **Materials** are stored in **catima::Layers** class.

There are following ways to define Layers from catima::Material classes:

```
catima::Material graphite({12,6,1});
catima::Material nitrogen({14,7,1});
...
catima::Layers matter1;
matter1.add(graphite);
matter1.add(nitrogen);
matter1.add(graphite);
cout<<"number of layers = "<<matter1.num()<<"\n"; //
3
```

Layers can be copied from existing Layes:

```
catima::Layers matter2;
matter2 = matter1; //matter2 contain 3 layers
matter2.add(nitrogen);
matter2.add(graphite); //matter2 contains 5 layers
now
```

Layers can be created as a combination of another Layers:

```
catima::Layers matter3 = matter1 + matter2;
```

Config

The calculation configuration is set via catima::Config class. The dafault configuration is predefined in **catima::default_config** variable. This **default_config** is supplied as default argument to functions like catima::calculate. If custom config is needed another configuration can be provided.

the structure Config is defined as:

```
    struct Config{
        char z_effective=z_eff_type::atima;
        char skip=skip_none;
        char dedx = 0;
    };
```

effective charge calculation###

the following effective charge calculations are buit in:

```
    enum z_eff_type:char {
```

```
        none = 0,
        atima = 1,
        pierce_blann = 1,
        anthony_landorf = 2,
        hubert = 3
    };
```

- z_eff_type::none - the provided Projectile Q is used as a charge
- z_eff_type::pierce_blann - Pierce Blann formula, using function: z_eff_Pierce_Blann()
- z_eff_type::anthony_landorf - function: z_eff_Anthony_Landorf()
- z_eff_type::huber - function: z_eff_Huber()

All available switches are defined in **config.h** file.

Using the library

the include direcotry and LD_LIBRARY_PATH must be properly adjusted. The app must be linked against catima library. For example check examples directory and makefile inside to see how to link.

All functions and classes are inside **catima namespace**

Normally including main file is enough:

```
#include "catima/catima.h"
```

Using with C

the C wrapper is provided in cwapper.h, this file can be included in C app. The C app must be then linked against catima library. It provides only basic interface.