

Data format for visualization/analysis of collisional N -body simulation Ver 0.0 — Aug 4 2011

Kob2011 collaboration

August 4, 2011

Contents

1	Introduction	2
2	Basic idea	2
2.1	Some basics of YAML	2
2.2	Particle Stream Data Format	4
3	Some remarks	5
3.1	File format	5
4	Repository	5
5	Sample codes	6

1 Introduction

- need for common data format
- need to be able to reproduce complete trajectory of particles of interests
- machine independence
- performance
- parallel processing

2 Basic idea

We need to store the data sufficient to reconstruct the orbits of individual particles. Conceptually, what we need is a stream of phase-space information of particles, such as:

```
particle_id, time, mass, x, y, z, vx, vy, vz, ...  
particle_id, time, mass, x, y, z, vx, vy, vz, ...  
particle_id, time, mass, x, y, z, vx, vy, vz, ...
```

However, the data format must be flexible enough to be able to include more information such as

- radius, and other info related to stellar evolution
- merger history
- whatever else you can think of

One way to construct such a flexible data format is to use self-describing data format, such as XML or YAML. For simplicity, we adopt YAML here.

2.1 Some basics of YAML

The following is a simple example of YAML data which can be directly converted to/from an Ruby object:

```
--- !!Particle  
id: 0  
x:  
  - 0.1
```

```

    - 0.2
    - 0.3
v:
  - -1
  - -2
  - -3
m: 1.0

```

If this text is in a file “test.yaml”, one can read in it by

```

class Particle
  def taguri
    return 'x-private:Particle'
  end
end
require 'yaml'
YAML.add_private_type('Particle') do |type, val|
  YAML.object_maker(Particle, val)
end
obj= YAML.load_file('test.yaml')

```

The content of the object is now:

```

p obj
-> #<Particle:0xb7d151c0 @v=[-1, -2, -3], @id=0,
    @x=[0.1, 0.2, 0.3], @m=1.0>

```

As one can see, in this way we created a object of class Particle, with index, position, velocity, and mass given in the text. In the above example, the line

```

--- !!Particle

```

indicates that the following text should generate a ruby object with name “Particle”.

```

id: 0

```

defines a member variable with name “id”, and value 0 for this particular instance

```

x:
  - 0.1
  - 0.2
  - 0.3

```

means the member variable “x” is an array with three elements. The first “-” means this line is a data for an array. By default, numbers without “.” are regarded as integers, and with “.” floating point. Note that indentation has meaning here and “-” must be indented the same level or deeper than “x” and should be aligned. I guess I do not need to explain “v” and “m”

2.2 Particle Stream Data Format

With the minimal description of YAML in the previous section, now we can define the generic data format (not yet for binaries, though): The data format is the stream of YAML representation of particle object, like

```
--- !!Particle
id: 0
t: 0
x:
  - 0.1
  - 0.2
  - 0.3
v:
  - -1
  - -2
  - -3
m: 1.0
--- !!Particle
id: 1
t: 0
x:
  - 0.2
  - 0.3
  - 0.4
v:
  - 0
  - 0
  - 0
m: 1.0
....
```

We need to have some naming conversion, so that different programs can understand at least the basic common part of data. So we have

name	description
id	index
m	mass
t	time
dt	timestep
x	position, array with three elements
v	velocity, array with three elements
p	potential
a	acceleration, array with three elements
j	jerk, array with three elements
s	snap, array with three elements
c	crackle, array with three elements
pop	pop, array with three elements

We call this format with basic naming convention PSDF (Particle Stream Data Format).

We call a YAML document which corresponds to a single particle object as “world-point”, since it is a point on the worldline associated with a particle. It is translated to an object of Particle class, to call it particle is a bit confusing.

3 Some remarks

3.1 File format

PSDS does not define the on-memory data description. It does not define how the individual YAML documents are stored.

There are variety of ways to store worldpoints in file(s). In one extreme, all world-points are stored in a single file, in the order defined by time sequence. In the opposite extreme, one worldpoint is stored in one file. We could think of anything in between, such as single file for one particle or single file for some period in time.

In order to do some parallel I/O, using multiple files may be more natural. We do not yet specify any particular form of parallel access, though.

4 Repository

`git@github.com:jmakino/Particle-Stream-Data-Format.git`

you can download it by:

```
mkdir foo
cd foo
git clone git://github.com/jmakino/Particle-Stream-Data-Format.git
```

To update:

```
git pull
```

5 Sample codes

Also in the github.

psdf.rb:

```
require 'yaml'
class Particle
  def taguri
    return 'x-private:Particle'
  end
end
YAML.add_private_type('Particle') do |type, val|
  YAML.object_maker(Particle, val)
end
```

This is current minimal “library”, which defines particle class and “Particle” tag.

writetest.rb:

```
require "psdf.rb"
class Particle
  attr_accessor :id, :x
  def initialize
    @id=0
    @t=0
    @x=[0,1,2]
  end
end
(0..10).each{|id|
  obj=Particle.new
  obj.id =id;
  obj.x[0]=id*0.1
  print YAML.dump(obj)
}
```

This one generate 10 worldpoints.

readtest.rb:

```
require "psdf.rb"
a = []
while s = gets("--- ")
  print s
  print "\n\nend of one gets\n\n"
  print s, "\n"
  obj = YAML.load(s)
  a.push obj if obj
end
p a
```

This one reads in 10 particles and store everything in a single array.