# Data format for visualization/analysis of collisional $N$-body simulation

Ver 0.0 — Aug 4 2011
Ver 0.1 — Aug 17 2011

Kob2011 collaboration

September 8, 2011

## Contents

# 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 one 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 a data in YAML format.

```
--- !!Particle
id: 0
r:
 - 0.1
 - 0.2
```

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

In the above example, the line

```
--- !!Particle
```

Is the header, which indicates that it describes the data of an object of type `Particle`.

```
id: 0
```

defines a field with name "id", and value 0.

```
r:
 - 0.1
 - 0.2
 - 0.3
```

means the field "r" 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 "r" and should be aligned. Here, "v" and "m" are similar.

# 3 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
r:
 - 0.1
 - 0.2
```

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

## 3.1 Formalities

For a data format to be understandable by a computer program, there need to be some convension. We start with the list of reserved words for the names:

| name | description |
|---|---|
| id | index (can be arbitrary text) |
| m | mass |
| t | time |
| t_max | max time to which this record is valid |
| r | position, array with three elements |
| v | velocity, array with three elements |
| pot | potential |
| acc | acceleration, array with three elements |
| jerk | jerk, array with three elements |
| snap | snap, array with three elements |
| crackle | crackle, array with three elements |
| pop | pop, array with three elements |

We require that time, position, velocity and higher derivatives are consistent (that if position is given in parsec and time in year, velocity must be in parsec/year, not km/s, for example). Also, we expect the users of this format to follow the

convension of using these names to descrive these physical quantities. Here, `t_max` is rather special, in that it gives the maximum possible time that this record is used to predict the orbit of this particle.

Any of these fields are not required. One can make a record without velocity, mass, time or whatsoever. If a program needs the positions of particles but the data file does not give them, the program should raise error message. We will give example APIs in several languages in the appendix.

In the a data file, if the record of one particle is not produced after certain time, the analysis/visualization program regard that it somehow vanished after time `t_max`.

We considered the possibility of defining a special record for creation and destruction of a particle, but decided against that to keep the parser as simple as possible to write.

We call this format with basic naming convention PSDF (Particle Stream Data Format). If one wants to add his/her own data, one possibility is to use a name like "NBODY6_predicted_position". We do not enforce the use of keyword "X_", but are not against it.

We call a YAML document which corresponds to a single particle object as "worldpoint", since it is a point on the worldline associated with a particle.

# 4 Some remarks

## 4.1 File format

PSDF 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 worldpoints are stored in a single big file, in the order defined by the 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. We might provide some examples, though.

# 5 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
```

# 6 Sample codes

Also in the github.

## 6.1 Base library

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.

## 6.2 Simple program to generate a psdf file

writetest.rb:

```
require "psdf.rb"
class Particle
 attr_accessor :id, :r
 def initialize
   @id=0
   @t=0
   @r=[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.

## 6.3 Simple program to read a psdf file

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.

## 6.4 An ACS Body method to write a psdf worldpoint

acs_psdf.rb:

```
require "psdf.rb"
class Particle
 attr_accessor :id, :r, :v, :a, :p, :j, :m, :t, :dt
end

class Body
 def to_psdf
   obj=Particle.new
   obj.id = @body_id
   obj.t  = @time
   obj.r = Array[*@pos]
```

```
   obj.v = Array[*@vel]
   obj.a = Array[*@acc]
   obj.j = Array[*@jerk]
   obj
 end
 def psdf_output
   print YAML.dump(self.to_psdf)
 end
end
```

In the ACS Body class, position, velocity and other vector data are of class `Vector`, not `Array`. So you need to convert them to `Array`.

## 6.5 An OpenGL animation program

The file testplot.rb gives a simple example. It it intended to be a skeleton code, with minimal UI and other stuff. It takes an input file with name "testin", in which a sequence of psdf world points is stored (can come from shared-timestep or individual-timestep code).

This program requires ruby-opengl library. To install this (if not already installed) try:

```
  sudo gem install opengl
```

On my machine with CentOS 5.5, it failed with:

```
sudo gem install ruby-opengl
Building native extensions.  This could take a while...
ERROR:  Error installing ruby-opengl:
        ERROR: Failed to build gem native extension.

/usr/bin/ruby -rubygems /usr/lib/ruby/gems/1.8/gems/rake-0.8.7/bin/rake RUBYARCHDIR=/usr
/usr/bin/ruby mkrf_conf.rb
(in /usr/lib/ruby/gems/1.8/gems/ruby-opengl-0.60.1)
rake
gcc  -fPIC -g -O2    -Wall -DRUBY_VERSION=187  -I/usr/include -I/usr/lib/ruby/1.8/x86_64
(in /usr/lib/ruby/gems/1.8/gems/ruby-opengl-0.60.1/ext/gl)
gl-enums.c:6  include :
../common/common.h:45:21: error: GL/glut.h:
                                                rake aborted!
Command failed with status (1): [gcc  -fPIC -g -O2    -Wall -DRUBY_VERSION=...]
/usr/lib/ruby/gems/1.8/gems/ruby-opengl-0.60.1/ext/gl/Rakefile:29
```

```
(See full trace by running task with --trace)
rake aborted!
Command failed with status (1): [rake...]

(See full trace by running task with --trace)


Gem files will remain installed in /usr/lib/ruby/gems/1.8/gems/ruby-opengl-0.60.1 for in
Results logged to /usr/lib/ruby/gems/1.8/gems/ruby-opengl-0.60.1/gem_make.out
```

In this case, you need to install GLUT library, by

```
sudo yum install freeglut   freeglut-devel
```