`SofteningBndryMaxPhys` `0`

This specifies the maximum physical softening of the sixth particle group.

6 File formats of GADGET-2: Snapshots & Initial conditions

The primary result of a simulation with GADGET-2 are *snapshots*, which are simply dumps of the state of the system at certain times. GADGET-2 supports parallel output by distributing a snapshot into several files, each written by a group of processors. This procedure allows an easier handling of very large simulations; instead of having to deal with one file of size of a few GB, say, it is much easier to have several files with a smaller size of a few hundred MB instead. Also, the time spent for I/O in the simulation code can be reduced if several files are written in parallel.

Each particle dump consists of k files, where k is given by `NumFilesPerSnapshot`. For $k > 1$, the filenames are of the form `snapshot_XXX.Y`, where `XXX` stands for the number of the dump, `Y` for the number of the file within the dump. Say we work on dump 7 with $k = 16$ files, then the filenames are `snapshot_007.0` to `snapshot_007.15`. For $k = 1$, the filenames will just have the form `snapshot_XXX`. The base "`snapshot`" can be changed by setting `SnapshotFileBase` to another value.

Each of the individual files of a given set of snapshot files contains a variable number of particles. However, the files all have the same basic format (this is the case for all three fileformats supported by GADGET-2), and all of them are in binary. A binary representation of the particle data is our preferred choice, because it allows *much faster* I/O than ASCII files. In addition, the resulting files are *much smaller*, and the data is stored *loss-less*.

6.1 Structure of the default snapshot file format

In the default file format of GADGET-2 (selected as fileformat `1` for `SnapFormat`), the data is organised in blocks, each containing a certain information about the particles. For example, there is a block for the coordinates, and one for the temperatures, etc. A list of the blocks defined in the public version of GADGET-2 is given in Table 2. The first block has a special role, it is a header which contains global information about the particle set, for example the number of particles of each type, the number of files used for this snapshot set, etc.

Within each block, the particles are ordered according to the particle type, i.e. gas particles will come first (type 0), then 'halo' particles (type 1), followed by 'disk' particles (type 2), and so on. The correspondence between type number and symbolic type name is given in Table 3. However, it is important to realize that the detailed sequence of particles within the blocks will still change from snapshot to snapshot. Also, a given particle may not always be stored in the snapshot file with the same number among the files belonging to one set of snapshot files. This is because particles may move around from one processor to another during the coarse of a parallel simulation. In order to trace a particle between different outputs, one therefore has to resort to the particle IDs, which can be used to label particles uniquely.

To allow an easy access of the data also in Fortran[2], the blocks are stored using the 'unformatted binary' convention of most Fortran implementations. In it, the data of each read or write statement is bracketed by block-size fields, which give the length of the data block in bytes. These block fields (which are two 4-byte integers, one stored before the data block and one after) can be useful also in C-code to check the consistency of the file structure, to make sure that one reads at the right place, and to skip individual blocks quickly by using the length information of the block size fields to fast forward in the file without actually having to read the data. The latter makes it possible to efficiently read only certain blocks, say just temperatures and densities, but no coordinates and velocities.

Assuming that variables have been allocated/declared appropriately, a possible read-statement of some of these blocks in Fortran could then for example take the form:

```
read (1) npart, massarr, a, redshift, flagsfr, flagfeedb, nall
read (1) pos
read (1)
read (1) id
read (1) masses
```

In this example, the block containing the velocities, and several fields in the header, would be skipped. Further read-statements may follow to read additional blocks if present. In the file `read_snapshot.f`, you can find a simple example of a more complete read-in routine. There you will also find a few lines that automatically generate the filenames, and further hints how to read in only certain parts of the data.

When you use C, the block-size fields need to be read or skipped explicitly, which is quite simple to do. This is demonstrated in the file `read_snapshot.c`. Note that you need to generate the appropriate block field when you write snapshots in C, for example when generating initial conditions for GADGET-2. Unlike GADGET-1, GADGET-2 will check explicitly that the block-size fields contain the correct values and refuse to start if not. In the `Analysis` directory, you can also find an example for a read-in routine in IDL.

The sequence of blocks in a file is given in Table 2. Note however that not all of the blocks need to be present, depending on the type of the simulation, and the makefile options of the code. For example, if there are no gas particles in the snapshot file, the blocks for internal energy or density will be absent. The presence of the mass block depends on whether or not particle masses are defined to be constant for certain particle types by means of the `Massarr` table in the file header. If a non-zero mass is defined there for a certain particle type, particles of this type will not be listed with individual masses in the mass block. If such fixed particle masses are defined for all types that are present in the snapshot file, the mass block will be absent.

The detailed structure of the fields of the file header is given in Table 4. Note that the header is filled to a total length of 256 bytes with unused entries, leaving space for extensions that can be easily ignored by legacy code. Also note that a number of the flags defined in the header are

[2] This should not be misunderstood as an encouragement to use Fortran.

#	Block ID	HDF5 identifier	Block content
1	`HEAD`		Header
2	`POS`	`Coordinates`	Positions
3	`VEL`	`Velocities`	Velocities
4	`ID`	`ParticleIDs`	Particle ID's
5	`MASS`	`Masses`	Masses (only for particle types with variable masses)
6	`U`	`InternalEnergy`	Internal energy per unit mass (only SPH particles)
7	`RHO`	`Density`	Density (only SPH particles)
8	`HSML`	`SmoothingLength`	SPH smoothing length h (only SPH particles)
9	`POT`	`Potential`	Gravitational potential of particles (only when enabled in makefile)
10	`ACCE`	`Acceleration`	Acceleration of particles (only when enabled in makefile)
11	`ENDT`	`RateOfChangeOfEntropy`	Rate of change of entropic function of SPH particles (only when enabled in makefile)
12	`TSTP`	`TimeStep`	Timestep of particles (only when enabled in makefile)

Table 2: Output blocks defined in GADGET-2's file formats.

Type	Symbolic Type Name
0	Gas
1	Halo
2	Disk
3	Bulge
4	Stars
5	Bndry

Table 3: Symbolic type names and their corresponding type numbers.

not used by the public version of GADGET-2. The data type of each field in the header is either `double` (corresponding to `real*8` in Fortran), or `int` (corresponding to `int*4` in Fortran), with the exception of the particle numbers, which are `unsigned int`.

The format, units, and variable types of the individual blocks in the snapshot files are as follows:

1. **Header:**
   `struct io_header;`
   The individual fields of the file header are defined in Table 4.

2. **Particle positions:**
   `float pos[N][3];` *`real*4 pos(3,N)`*
   Comoving coordinates in internal length units (corresponds to $h^{-1}\mathrm{kpc}$ if the above choice for the system of units is adopted). `N=sum(io_header.Npart)` is the total number of particles in the file. Note: The particles are ordered in the file according to their type, so it is guaranteed that particles of type 0 come before those of type 1, and so on. However, within a type, the sequence of particles can change from dump to dump.

3. **Particle velocities:**
   `float vel[N][3];` *`real*4 vel(3,N)`*
   Particle velocities u in internal velocity units (corresponds to km/sec if the default choice for the system of units is adopted). Peculiar velocities v are obtained by multiplying u with $\sqrt{a}$, i.e. $v = u\sqrt{a}$. For non-cosmological integrations, $u = v$ are just ordinary velocities.

4. **Particle identifications:**
   `unsigned int id[N];` `int*4 id(N)`
   Particle number for unique identification. The order of particles may change between different output dumps, but the IDs can be easily used to bring the particles back into the original order for every dump.

5. **Variable particle masses:**
   `float masses[Nm];` `real*4 masses(Nm)`
   Particle masses for those particle types that have variable particle masses (i.e. zero entries in `massarr`). `Nm` is the sum of those `npart` entries that have vanishing `massarr`. If `Nm=0`, this block is not present at all.

6. **Internal energy:**
   `float u[Ngas];` `real*4 u(Ngas)`
   Internal energy per unit mass for the `Ngas=npart(0)` gas particles in the file. The block is only present for `Ngas`>0. Units are again in internal code units, i.e. for the above system of units, `u` is given in $(\mathrm{km/sec})^2$.

7. **Density:**
   `float rho[Ngas];` `real*4 rho(Ngas)`

Header Field	Type	HDF5 name	Comment
`Npart[6]`	`uint`	NumPart_ThisFile	The number of particles of each type in the present file.
`Massarr[6]`	`double`	MassTable	The mass of each particle type. If set to 0 for a type which is present, individual particle masses from the `mass` block are read instead.
`Time`	`double`	Time	Time of output, or expansion factor for cosmological simulations.
`Redshift`	`double`	Redshift	$z = 1/a - 1$ (only set for cosmological integrations).
`FlagSfr`	`int`	Flag_Sfr	Flag for star formation (unused in the public version of GADGET-2).
`FlagFeedback`	`int`	Flag_Feedback	Flag for feedback (unused).
`Nall[6]`	`int`	NumPart_Total	Total number of particles of each type in the simulation.
`FlagCooling`	`int`	Flag_Cooling	Flag for cooling (unused).
`NumFiles`	`int`	NumFilesPerSnapshot	Number of files in each snapshot.
`BoxSize`	`double`	BoxSize	Gives the box size if periodic boundary conditions are used.
`Omega0`	`double`	Omega0	Matter density at $z = 0$ in units of the critical density (only relevant for cosmological integrations).
`OmegaLambda`	`double`	OmegaLambda	Vacuum energy density at $z = 0$ in units of the critical density.
`HubbleParam`	`double`	HubbleParam	Gives 'h', the Hubble constant in units of $100\,\mathrm{km\,s^{-1}Mpc^{-1}}$ (for cosmological integrations).
`FlagAge`	`int`	Flag_StellarAge	Creation times of stars (unused).
`FlagMetals`	`int`	Flag_Metals	Flag for metallicity values (unused).
`NallHW[6]`	`int`	NumPart_Total_HW	For simulations that use more than 2^{32} particles, these fields hold the most significant word of 64-bit total particle numbers. Otherwise zero.
`flag_entr_ics`	`int`	Flag_Entropy_ICs	Flags that *initial conditions* contain entropy instead of thermal energy in the `u` block.
`unused`			Currently unused space which fills the header to a **total length of 256 bytes**, leaving room for future additions.

Table 4: Fields in the file header of snapshot files. In the native file formats of GADGET-2, these fields are the elements of a structure of total length of 256 bytes. In the HDF5 file format, each field is stored as an attribute to a `Header` data group.

Comoving density of SPH particles. Units are again in internal code units, i.e. for the above system of units, `rho` is given in $10^{10}\,h^{-1}\mathrm{M}_\odot/(h^{-1}\mathrm{kpc})^3$.

8. **Smoothing length:**
   `float hsml[Ngas];` `real*4 hsml(Ngas)`
   Smoothing length of the SPH particles. Given in comoving coordinates in internal length units.

9. **Gravitational potential:**
   `float pot[N];` `real*4 pot(N)`
   Gravitational potential for particles. This block will only be present if it is explicitly enabled in the makefile.

10. **Accelerations:**
    `float acc[N][3];` `real*4 acc(3,N)`
    Accelerations of particles. This block will only be present if it is explicitly enabled in the makefile.

11. **Rate of entropy production:**
    `float dAdt[Ngas];` `real*4 dAdt(Ngas)`
    The rate of change of the entropic function of each gas particles. For adiabatic simulations, the entropy can only increase due to the implemented viscosity. This block will only be present if it is explicitly enabled in the makefile.

12. **Timesteps of particles:**
    `float dt[N];` `real*4 dt(N)`
    Individual particle timesteps of particles. For cosmological simulations, the values stored here are $\Delta\ln(a)$, i.e. intervals of the natural logarithm of the expansion factor. This block will only be present if it is explicitly enabled in the makefile.

6.2 A variant of the default file format

Whether or not a certain block is present in GADGET-2's default fileformat depends on the type of simulation, and the makefile options. This makes it difficult to write reasonably general I/O routines for analysis software, capable of dealing with output produced for a variety of makefile settings. For example, if one wants to analyse the (optional) rate of entropy production of SPH particles, then the location of the block in the file changes if, for example, the output of the gravitational potential is enabled or not. This problem becomes particularly acute in the full physics version of GADGET-2, where already around 40 I/O blocks are defined.

To remedy this problem, a variant of the default fileformat was implemented in GADGET-2 (based on a suggestion by Klaus Dolag), which can be selected by setting `SnapFormat=2`. Its only difference is that each block is preceeded by a small additional block which contains an identifier in the form of a 4-character string (filled with spaces where appropriate). This identifier is contained in the second column of Table 2. Using it, one can write more flexible

I/O routines and analysis software that quickly fast forwards to blocks of interest in a simple way, without having to know where it is expected in the snapshot file.

6.3 The HDF file format

If the hierarchical data format is selected as format for snapshot files or initial conditions files, GADGET-2 accesses files with low level HDF5 routines. Advantages of HDF5 lie in its portability (e.g. automatic endianness conversion) and generality (which however results in somewhat more complicated read and write statements), and in the availability of a number of tools to manipulate or display HDF5 files. A wealth of information about HDF5 can be found on the website of the project.

GADGET-2 stores the snapshot file data in several data groups, which are organised as follows:

```
/Header
/Type0/Positions
/Type0/Velocities
/Type0/IDs
...
/Type1/Positions
/Type1/Velocities
...
```

The `Header` group only contains attributes that hold the fields of the file header in the standard file format of GADGET-2. The names of these attributes are listed in Table 4. For each particle type present in the snapshot file, a separate data group is defined, named `Type0`, `Type1`, and so on. The blocks of the standard file format are then stored as datasets within these groups. The names of these data sets are listed in Table 2.

6.4 Format of initial conditions

The possible file formats for initial conditions are the same as those for snapshot files, and are selected with the `ICFormat` parameter. However, only the blocks up to and including the gas temperature (if gas particles are included) need to be present; gas densities, SPH smoothing lengths and all further blocks need not be provided.

In preparing initial conditions for simulations with gas particles, the temperature block can be filled with zero values, in which case the initial gas temperature is set in the parameterfile with the `InitGasTemp` parameter. However, even when this is done, the temperature block must still be present.

Note that the field `Time` in the header will be ignored when GADGET-2 is reading an initial conditions file. Instead, you have to set the time of the start of the simulation with the `TimeBegin` option in the parameterfile.