IRATE Documentation

Release 0.2dev

Erik Tollerud, Shea Garrison-Kimmel, Miguel Rocha

Apr 16, 2017

Contents

The *IRATE* (IRvine Astrophysical simulaTion structurE) package contain tools for accessing and validating HDF5 files following the IRATE format, a format for astropysical n-body and hydrodynamical simulations.

Contents:

CHAPTER 1

Getting and Installing IRATE

Getting IRATE

You can get the latest version of the IRATE format from bitbucket at the IRATE-format project page. You'll need to install mercurial and just run the command at the top of the project page that begins with `hg clone`. Once a release is ready, it will appear on PyPI

Installing

Of course, you must have python installed before you can do anything with IRATE, although most operating systems other than Windows have it installed automatically.

Once you have a copy of the IRATE source code, you will need to be sure you install the HDF5 libraries (libhdf5) which can be found on the HDF5 web site, although it's usually much easier to just install using a package manager if your OS has one. IRATE requires at least v1.8 of the HDF5 format.

Additionally, if you intend to convert Tipsy binary format files to IRATE, you will need to have <Cython *http://cython.org/*>_ installed before you install IRATE.

Two other packages are needed by IRATE - h5py, and numpy. If you so desire, you can install these yourself beforehand, although the install process for IRATE should download and install it for you

Once you have the requirements satisfied, execute the following command in the IRATE source code directory:

```
python setup.py install
```

Note that on most unix-like systems, you will have to run this command as root, or (e.g. Ubuntu) include the command `sudo` in front of it. This should install IRATE, and you can go ahead and use any of the scripts or library components.

CHAPTER 2

IRATE Format Specification

This document describes the specific required structure of the IRATE format for format version 0.

All IRATE files are HDF5 files, and hence usually have either an `.h5` or `.hdf5` extension.

IRATE File Format

The main data file for an IRATE format is referred to as simply an "IRATE file". These files may store any number of the actual outputs of a simulation (typically meaning multiple snapshots), associated halo and/or galaxy catalogs and merger trees, and any other data that might be associated with such a simulation (e.g. black hole catalogs).

Note: An IRATE file is intended to hold at most *one* simulation. Multiple simulations should be stored as multiple separate IRATE files. A single simulation *can*, however, be spread over multiple files - see the *irate.core.scatter_files* and *irate.core.gather_files* functions for examples. In that case, the root file that contains the main heirarchy is the "IRATE file" and the others are ancillary files.

To conform to the IRATE standard, such a file must satisfy the following conditions:

- The root of the file must have an integer attribute named 'IRATEVersion' that specifies the version of the IRATE format that the file obeys. The format version for this documentation is 0. The format version for the currently-installed IRATE tools can always be accessed as an integer via `irate.formatversion`.
- At the root of the file, there must be a `Group` 'Cosmology'. This `Group` must have the following HDF5 attributes to specify the cosmology that defines the data:
  - 'HubbleParam'
  - 'OmegaMatter'
  - 'OmegaLambda'
  - 'OmegaBaryon'
  - 'PowerSpectrumIndex'

  - 'sigma_8'

  Furthermore, if the cosmology used has an accepted name (e.g. WMAP-7), it is strongly recommended that the `Group` have an additional attribute, 'Name', for human readability; such an attribute, however, is not required.

  Some cosmologies may include additional parameters, in which case such parameters can be included as attributes of the 'Cosmology' group, or as datasets if such information can only be stored in array form. The naming conventions used above are recommended for custom parameters.

  For non-cosmological simulations, the 'Cosmology' `Group` must still be present; however, to signify that the cosmology is unimportant, all of the attributes should be set to zero and the 'Name' attribute should be set to 'Non-Cosmological.'

- The root of the file must also contain a `Group` named 'SimulationProperties'. Various properties of the simulation, such as the box size and assorted flags, should be provided in this `Group`. If it's possible, they should be given as attributes; however, it is accepted in the format that this group contain datasets as well.
- Also at the root of the file, there may be any number of Groups with names of the form 'Snapshot#####', where the # is typically a number identifying the output in the context of the simulation, padded to be five digits long (e.g. Snapshot 35 would be saved under /Snapshot00035). Each Snapshot `Group` should have an attribute named 'ScaleFactor', but if there's neither particle nor grid data contained within the the snapshot, it's not required. It must contain only other Groups, which may be 'ParticleData' or 'GridData' (whose individual requirements are discussed in *Particle Data* and *Grid Data*, repectively), along with any number of halo or galaxy catalogs (described below in *Halo Catalogs* and *Galaxy Catalogs*).

Todo

Developers, Should redshift be required? It's not provided by halo catalogs usually, so we'd be requiring users to manually type it in.)

Todo

Developers, Is requiring that the simulation groups be called "Snapshot#" too restrictive? Should some other naming convention be required, instead? Or just say any groups not explicitly called for here will be treated as snapshots regardless of their names (that's in conflict with the second bullet point below)?

- The root of the file may (but is not required to) contain a 'MergerTrees' `Group`, which holds information about the merger trees in the simulation. If present, this group must obey the format specified in *Merger Trees*.
- The root of the file may also contain any other Groups that are desired, but their form is not specified in the format. Additionally, it is strongly recommended that they follow the same conventions with regards to units and naming structure that are laid out elsewhere in this documentation.

Todo

Developers, do we want to allow this, or should there be nothing else allowed at the root level?

- There must not be spaces in any group names so as not to confuse some HDF5 tools that don't play well with spaces.

Note: All group and attribute names are *case-sensitive*.

Unit Information

For all datasets that have units associated with them, those units should be stored either in the individual datasets as attributes, or as attributes of the `Group` that contains the datasets. In either case, it should be presented in both human readable and in the form of a conversion factor to CGS units. If a dataset does not have units, it will be assumed to be dimensionless.

Todo

Developers, how do you like this method of including units sound? Its based on Andrew's and the yt/GDF format scheme...

If the units are attached directly to the `Dataset` that they relate to, they must be named 'unitname' and 'unitcgs'; if they are instead attached to a `Group` above them, the names should be prepended with the exact name of the `Dataset` that they relate to; e.g. the units for the `Dataset` 'R200b' would be named 'R200bunitname' and 'R200bunitcgs', if they are attributes to the group that contains that `Dataset`.

The 'unitname' attribute should be a string defining the unit, e.g. 'kpc/h'. The unitcgs attribute must be a three element array, where the stored values are, in order, the numerical conversion factor to CGS, the value of the exponent on the Hubble Parameter that the conversion factor should be multiplied by, and lastly the value of the exponent on the scale factor that the conversion factor should be multipled by.

For example, if 'unitname' is 'comoving Mpc/h', 'unitcgs' should be an array containing [3.0857e24, -1, 1].

Note that the core library provides utilities for accessing units - see *`irate.core.get_units()`*, *`irate.core.set_units()`*, and *`irate.core.get_cgs_factor()`*.

Other Metadata

Other metadata associated with individual datasets should be included in the same fashion as units. That is, they should either be attributes directly attached to the dataset with the metadata field name, or they can be attributes of groups further up the hierarchy, following the simple naming convention *datasetnamemetadataname*. The core library provides utilities for accessing or setting metadata in *`irate.core.get_metadata()`* and *`irate.core.set_metadata()`*.

Particle Data

The ParticleData `Group`, if it exists, must contain at least one group, of which the most common are 'Dark', 'Gas', and 'Star'; these contain the data for dark matter, stars, and gas, respectively. Users are free to use other names for particle blocks, e.g. if the users want to separate high resolution from low resolution particles, but any `Group` containing dark matter particles must have a (case-sensitive) name that begins with 'Dark' (e.g. 'Dark_HighRes'), any `Group` containing gas particles must have a name that begins with 'Gas', and any `Group` containing star particles must have a name beginning with 'Star'. Users are free to store other particle types in IRATE files; it is strongly recommended that they follow the same convention laid out here (e.g. 'BlackHole'). Tools that read in IRATE files, such as halo finders, will assume the type of particle based on the group name.

Any groups within /Snapshot#/ParticleData/ may contain only data sets. For particle data, the following `Dataset` objects must be present in each group that exists, even if they have 0 particles:

- 'Position' (N x d)
- 'Velocity' (N x d)
- 'Mass' (N)
- 'ID' (N)

where *d* is the dimensionality (presumably pretty much always 3) and *N* is the total number of particles. Additional data sets (e.g. 'Metallicity','Entropy', 'Density', etc.) may be present, but the above 4 are the minimum required. Any other data sets are encouraged to either be shape *N* for scalar data, or *N* x *d* for vector data.

Grid Data

The grid data specification has not yet been defined.

Halo Catalogs

Halo catalogs are stored as a `Group` that must have names that begin with the phrase 'HaloCatalog', For example, both 'HaloCatalog_AHF1' and 'HaloCatalog_Rockstar' are valid names; 'AHFCatalog' and 'Catalog_Rockstar', however, are not.

Any halo catalogs that are contained within a Snapshot `Group` should have, as attributes, any parameters that are relevant to the halo finder, such as FOF linking lengths, overdensity criterion, or the code used to produce that catalog (though the former may be obvious from the name of the group).

Any halo catalogs must contain a `Dataset` with the Name 'Center' that has shape N x d, where N is the number of halos in thecatalog, and d is the dimensionality (typically 3). All other datasets in the catalog should have a matching first dimension, and should be in the same order. That is, the ith entry in 'Center' should correspond to the same halo as the ith entry in any of the other datasets.

If the index of the most bound particle of each halo is included in the halo catalog, it should be stored in a `Dataset` named 'MostBoundParticleID'.

If particle data is included with the halo catalog, it must be saved in a `Group` inside the halo catalog with the name 'HaloParticleData'. This group must contain at least two datasets. The first of these should be named 'HaloParticleIDs', while the second should be named 'ParticlePerHalo'.

'HaloParticleIDs' should contain integer particle IDs in order such that all particles in the first halo come first, followed by those in the second halo, and so on. Here, halo order is the same as the order of the halos in the 'Center' dataset. Note that the number of elements of this dataset is not neccesarily the same as the number of total particles, because some particles may be members of multiple halos, in which case they appear on 'HaloParticleIDs' more than once.

The 'ParticlePerHalo' `Dataset`, on the other hand, must be of a length matching the first dimension of of the 'Center' dataset, and should give the (integer) number of particles in each halo. The sum of all of the values in this dataset must match the size of the 'HaloParticleIDs' dataset. This allows 'HaloParticleIDs' and 'ParticlesPerHalo' to provide all the information needed determine which particles are in which halos.

Many users will find it convenient to store the type of particle as well. This should be saved in a third `Dataset` named 'HaloParticleTypes', but this dataset is not required by the format. If it is present, it should be of the same size as 'HaloParticleIDs'.

Galaxy Catalogs

Galaxy catalogs are stored as a `Group` that must have names that begin with the phrase 'GalaxyCatalog', For example, both 'GalaxyCatalog_Galacticus' and 'GalaxyCatalog_LGalaxies' are valid names; 'GalacticusCatalog' and 'Catalog_LGalaxies', however, are not.

Any galaxy catalogs that are contained within a Snapshot `Group` should have, as attributes, any parameters that are relevant to the galaxy formation code, such as input parameter values, or the version of the code used to produce that catalog.

Any galaxy catalogs must contain two `Dataset` s with the names 'HaloID' and 'HaloSnapshot', that have shape N, where N is the number of galaxies in the catalog. All other datasets in the catalog should have a matching first dimension, and should be in the same order. That is, the ith entry in 'HaloID' should correspond to the same halo as the ith entry in any of the other datasets. The 'HaloID' dataset should give the ID of the halo in which the galaxy is located, while 'HaloSnapshot' should give the corresponding snapshot number at which that halo exists. (Galaxies may be located in halos which exist at an earlier snapshot if, for example, the halo can no longer be found at the current snapshot, but the galaxy formation code determines that the galaxy itself has not yet merged.) If corresponding halos are not present in the file these two `Dataset` s should have all values set to `-1`.

Merger Trees

Merger trees are stored as a `Group` that must have names that begin with the phrase 'MergerTrees'. For example, both 'MergerTrees_yt' and 'MergerTrees_Millennium' are valid names; 'ytTrees', however, is not.

Merger tree groups should have, as attributes, any parameters that are relevant to the merger tree builder, such as the name of the code used to build the trees.

Any merger tree groups must contain a `Dataset` with the name 'HaloID' that has shape N, where N is the total number of halos in all trees. The dataset should give the integer index of a halo in a 'HaloCatalog' `Group`. Also required are `Dataset` s with the names 'HaloSnapshot', 'DescendentID' and 'DescendentSnapshot' which must have the same shape, N, and should be in the same order. That is, the ith entry in 'HaloID' should correspond to the same halo as the ith entry in 'HaloSnapshot', 'DescendentID' and 'DescendentSnapshot'. The 'HaloSnapshot' `Dataset` must give the index of the snapshot to which this halo belongs. The 'DescendentID' and 'DescendentSnapshot' `Dataset` s must give the index and snapshot of the halo into which this halo descends. For halos with no descendent (e.g. the root halo of a tree), values of -1 should be used.

In addition, the MergerTrees `Group` must contain a `Dataset` name 'HalosPerTree' must be of a length equal to the total number of trees present in group, and should give the (integer) number of halos in each tree. The sum of all of the values in this dataset must match the size of the 'HaloID' `Dataset`. 'HalosPerTree' provides all the information needed determine which halos are in which trees. Optionally, a `Dataset` named 'TreeID', which should have the same length as 'HalosPerTree' may be present, and should give a unique identifying index for each tree.

Examples

Here we provide the structure of a sample IRATE Format file in the form output by the `h5dump` utility (included in libhdf5 library). Note that the 'Halo', 'Bulge', and 'Disk' groups are not actually a part of the specification, but are examples of possible ways one might wish to sub-divide the particle data. Also note that a typical IRATE file will contain many more datasets, particularly in the catalogs, which have been removed from here for the sake of brevity:

```
HDF5 "SampleIRATEfile.hdf5" {
FILE_CONTENTS {
 group      /                        (Contains attribute defining the version of the␣
↪IRATE format that this file conforms to)
 group      /Cosmology               (Contains attributes defining the cosmology of␣
↪the simulation)
 group      /SimulationProperties    (Contains attributes defining non-cosmological␣
↪properties of the simulation)
 group      /Snapshot00144           (Contains attributes defining redshift, scale␣
↪factor, or both)
 group      /Snapshot00144/HaloCatalog_AHF      (Should contain attributes defining␣
↪the parameters of the halo finding)
 dataset    /Snapshot00144/HaloCatalog_AHF/Center    (Contains attributes with unit␣
↪information)
 dataset    /Snapshot00144/HaloCatalog_AHF/Ekin      (Contains attributes with unit␣
↪information)
 dataset    /Snapshot00144/HaloCatalog_AHF/Epot      (Contains attributes with unit␣
↪information)
```

```
group      /Snapshot00144/HaloCatalog_AHF/HaloParticleData
ext link   /Snapshot00144/HaloCatalog_AHF/HaloParticleData/HaloParticleTypes ->␣
↪SampleIRATEfile-00144particles.hdf5 /HaloParticleTypes
ext link   /Snapshot00144/HaloCatalog_AHF/HaloParticleData/HaloParticleIDs ->␣
↪SampleIRATEfile-00144particles.hdf5 /HaloParticleIDs
ext link   /Snapshot00144/HaloCatalog_AHF/HaloParticleData/ParticlesPerHalo ->␣
↪SampleIRATEfile-00144particles.hdf5 /ParticlesPerHalo
dataset    /Snapshot00144/HaloCatalog_AHF/L        (Contains attributes with unit␣
↪information)
dataset    /Snapshot00144/HaloCatalog_AHF/Mvir     (Contains attributes with unit␣
↪information)
dataset    /Snapshot00144/HaloCatalog_AHF/Phi      (Contains attributes with unit␣
↪information)
group      /Snapshot00144/HaloCatalog_AHF/RadialProfiles
dataset    /Snapshot00144/HaloCatalog_AHF/RadialProfiles/L         (Contains␣
↪attributes with unit information)
dataset    /Snapshot00144/HaloCatalog_AHF/RadialProfiles/M_in_r    (Contains␣
↪attributes with unit information)
dataset    /Snapshot00144/HaloCatalog_AHF/RadialProfiles/dens      (Contains␣
↪attributes with unit information)
dataset    /Snapshot00144/HaloCatalog_AHF/RadialProfiles/npart
dataset    /Snapshot00144/HaloCatalog_AHF/RadialProfiles/r         (Contains␣
↪attributes with unit information)
dataset    /Snapshot00144/HaloCatalog_AHF/RadialProfiles/vcirc     (Contains␣
↪attributes with unit information)
dataset    /Snapshot00144/HaloCatalog_AHF/Rmax         (Contains attributes with␣
↪unit information)
dataset    /Snapshot00144/HaloCatalog_AHF/Rvir         (Contains attributes with␣
↪unit information)
dataset    /Snapshot00144/HaloCatalog_AHF/Velocity     (Contains attributes with␣
↪unit information)
dataset    /Snapshot00144/HaloCatalog_AHF/Vmax         (Contains attributes with␣
↪unit information)
dataset    /Snapshot00144/HaloCatalog_AHF/fMhires
dataset    /Snapshot00144/HaloCatalog_AHF/lambda
dataset    /Snapshot00144/HaloCatalog_AHF/nbins
dataset    /Snapshot00144/HaloCatalog_AHF/npart
group      /Snapshot00144/HaloCatalog_Rockstar     (Should contain attributes␣
↪defining the parameters of the halo finding)
dataset    /Snapshot00144/HaloCatalog_Rockstar/Center      (Contains attributes with␣
↪unit information)
dataset    /Snapshot00144/HaloCatalog_Rockstar/M200b       (Contains attributes with␣
↪unit information)
dataset    /Snapshot00144/HaloCatalog_Rockstar/R200b       (Contains attributes with␣
↪unit information)
dataset    /Snapshot00144/HaloCatalog_Rockstar/Rmax        (Contains attributes with␣
↪unit information)
dataset    /Snapshot00144/HaloCatalog_Rockstar/Spin
dataset    /Snapshot00144/HaloCatalog_Rockstar/Velocity    (Contains attributes with␣
↪unit information)
dataset    /Snapshot00144/HaloCatalog_Rockstar/Vmax        (Contains attributes with␣
↪unit information)
dataset    /Snapshot00144/HaloCatalog_Rockstar/npart
group      /Snapshot00144/ParticleData                     (Contains attributes with␣
↪unit information for all datasets within it)
group      /Snapshot00144/ParticleData/Dark_Bulge
dataset    /Snapshot00144/ParticleData/Dark_Bulge/ID
dataset    /Snapshot00144/ParticleData/Dark_Bulge/Mass
dataset    /Snapshot00144/ParticleData/Dark_Bulge/Position
dataset    /Snapshot00144/ParticleData/Dark_Bulge/Velocity
group      /Snapshot00144/ParticleData/Dark_Disk
dataset    /Snapshot00144/ParticleData/Dark_Disk/ID
dataset    /Snapshot00144/ParticleData/Dark_Disk/Mass
dataset    /Snapshot00144/ParticleData/Dark_Disk/Position
dataset    /Snapshot00144/ParticleData/Dark_Disk/Velocity
group      /Snapshot00144/ParticleData/Dark_Halo
dataset    /Snapshot00144/ParticleData/Dark_Halo/ID
dataset    /Snapshot00144/ParticleData/Dark_Halo/Mass
dataset    /Snapshot00144/ParticleData/Dark_Halo/Position
dataset    /Snapshot00144/ParticleData/Dark_Halo/Velocity
group      /Snapshot00153                      (Contains attributes defining␣
↪redshift, scale factor, or both)
group      /Snapshot00153/HaloCatalog_AHF      (Should contain attributes defining␣
↪the parameters of the halo finding)
dataset    /Snapshot00153/HaloCatalog_AHF/Center       (Contains attributes with␣
↪unit information)
group      /Snapshot00153/HaloCatalog_AHF/HaloParticleData
ext link   /Snapshot00153/HaloCatalog_AHF/HaloParticleData/HaloParticleTypes ->␣
↪SampleIRATEfile-00153particles.hdf5 /HaloParticleTypes
ext link   /Snapshot00153/HaloCatalog_AHF/HaloParticleData/HaloParticleIDs ->␣
↪SampleIRATEfile-00153particles.hdf5 /HaloParticleIDs
ext link   /Snapshot00153/HaloCatalog_AHF/HaloParticleData/ParticlesPerHalo ->␣
↪SampleIRATEfile-00153particles.hdf5 /ParticlesPerHalo
dataset    /Snapshot00153/HaloCatalog_AHF/L            (Contains attributes with␣
↪unit information)
dataset    /Snapshot00153/HaloCatalog_AHF/Mvir         (Contains attributes with␣
↪unit information)
group      /Snapshot00153/HaloCatalog_AHF/RadialProfiles
dataset    /Snapshot00153/HaloCatalog_AHF/RadialProfiles/M_in_r    (Contains␣
↪attributes with unit information)
dataset    /Snapshot00153/HaloCatalog_AHF/RadialProfiles/r         (Contains␣
↪attributes with unit information)
dataset    /Snapshot00153/HaloCatalog_AHF/RadialProfiles/vcirc     (Contains␣
↪attributes with unit information)
dataset    /Snapshot00153/HaloCatalog_AHF/Rmax         (Contains attributes with␣
↪unit information)
dataset    /Snapshot00153/HaloCatalog_AHF/Rvir         (Contains attributes with␣
↪unit information)
dataset    /Snapshot00153/HaloCatalog_AHF/Velocity     (Contains attributes with␣
↪unit information)
dataset    /Snapshot00153/HaloCatalog_AHF/Vmax         (Contains attributes with␣
↪unit information)
dataset    /Snapshot00153/HaloCatalog_AHF/nbins
dataset    /Snapshot00153/HaloCatalog_AHF/npart
group      /Snapshot00153/HaloCatalog_Rockstar     (Should contain attributes␣
↪defining the parameters of the halo finding)
dataset    /Snapshot00153/HaloCatalog_Rockstar/Center          (Contains attributes␣
↪with unit information)
dataset    /Snapshot00153/HaloCatalog_Rockstar/M200b           (Contains attributes␣
↪with unit information)
dataset    /Snapshot00153/HaloCatalog_Rockstar/Mbound200b      (Contains attributes␣
↪with unit information)
dataset    /Snapshot00153/HaloCatalog_Rockstar/R200b           (Contains attributes␣
↪with unit information)
dataset    /Snapshot00153/HaloCatalog_Rockstar/Rmax            (Contains attributes␣
↪with unit information)
```

```
 dataset    /Snapshot00153/HaloCatalog_Rockstar/Velocity        (Contains attributes␣
↪with unit information)
 dataset    /Snapshot00153/HaloCatalog_Rockstar/Vmax            (Contains attributes␣
↪with unit information)
 dataset    /Snapshot00153/HaloCatalog_Rockstar/npart           (Contains attributes␣
↪with unit information)
 group      /Snapshot00153/ParticleData             (Contains attributes with unit␣
↪information for all datasets within it)
 group      /Snapshot00153/ParticleData/Dark_Bulge
 dataset    /Snapshot00153/ParticleData/Dark_Bulge/ID
 dataset    /Snapshot00153/ParticleData/Dark_Bulge/Mass
 dataset    /Snapshot00153/ParticleData/Dark_Bulge/Position
 dataset    /Snapshot00153/ParticleData/Dark_Bulge/Velocity
 group      /Snapshot00153/ParticleData/Dark_Disk
 dataset    /Snapshot00153/ParticleData/Dark_Disk/ID
 dataset    /Snapshot00153/ParticleData/Dark_Disk/Mass
 dataset    /Snapshot00153/ParticleData/Dark_Disk/Position
 dataset    /Snapshot00153/ParticleData/Dark_Disk/Velocity
 group      /Snapshot00153/ParticleData/Dark_Halo
 dataset    /Snapshot00153/ParticleData/Dark_Halo/ID
 dataset    /Snapshot00153/ParticleData/Dark_Halo/Mass
 dataset    /Snapshot00153/ParticleData/Dark_Halo/Position
 dataset    /Snapshot00153/ParticleData/Dark_Halo/Velocity
 }
}
```

In addition, we include here an example of a file with merger trees:

```
HDF5 "treesIRATE.hdf5" {
FILE_CONTENTS {
 group      /
 group      /Cosmology
 group      /MergerTrees
 dataset    /MergerTrees/DescendentID
 dataset    /MergerTrees/DescendentSnapshot
 dataset    /MergerTrees/HaloID
 dataset    /MergerTrees/HaloSnapshot
 dataset    /MergerTrees/HalosPerTree
 dataset    /MergerTrees/HostID
 dataset    /MergerTrees/TreeID
 group      /SimulationProperties
 group      /Snapshot00016
 group      /Snapshot00016/HaloCatalog
 dataset    /Snapshot00016/HaloCatalog/AngularMomentum
 dataset    /Snapshot00016/HaloCatalog/Center
 dataset    /Snapshot00016/HaloCatalog/HalfMassRadius
 dataset    /Snapshot00016/HaloCatalog/Index
 dataset    /Snapshot00016/HaloCatalog/Mass
 dataset    /Snapshot00016/HaloCatalog/MostBoundParticleID
 dataset    /Snapshot00016/HaloCatalog/Velocity
 group      /Snapshot00016/ParticleData
 group      /Snapshot00016/ParticleData/Dark
 dataset    /Snapshot00016/ParticleData/Dark/ID
 dataset    /Snapshot00016/ParticleData/Dark/Mass
 dataset    /Snapshot00016/ParticleData/Dark/Position
 dataset    /Snapshot00016/ParticleData/Dark/Velocity
 }
}
```

CHAPTER 3

Command Line Scripts and Conversion Tools

One of the main intents of the IRATE format is to unify the variety of formats currently used for simulations into a standard format that many tools can access efficiently. Hence, the IRATE package provides a number of conversion tools from various formats

IRATE Utilities

Validate IRATE File Structure

The `iratevalidate` script tests whether or not a file conforms to the IRATE format. It can also be used to show the general structure of a supplied file (or files).

Options:

`-v, --verbose`
Prints detailed messages during validation

`-i, --immediate`
Causes the script to immediately exit on first validaton error instead of reporting all errors.

`-c, --print-cosmology`
Print cosmology information from the supplied IRATE file(s) in addition to validating.

`-p, --print-structure`
Print group and dataset structure information for the supplied IRATE file(s) in addition to validating.

`-t, --type`
The type of IRATE file to assume for validation - used to load custom validators.

`-l, --list-types`
Lists all valid types and immediately exits.

`-s, --skips`
Skips the validation stage.

Gather IRATE File

The `irategather` script takes an IRATE file (or really any HDF5 file) and combines any externally linked datasets or groups into a single monolithic file. See *`irate.core.gather_files()`* for additional information.

Options:

`-o, --output`
File to output gathered file to. If not given, the input file will be overwritten.

`-d, --delete-gathered`
Delete all files that were combined to make the output file after gathering is complete.

`-v, --validate`
Validate before gathering. If validation fails, gathering will not occur.

Format Converters

Tipsy Binary -> IRATE

The `tipsy2irate` script converts files in the tipsy binary format to IRATE. Use it as `tipsy2irate [options] infile [outfile]`. If *outfile* is not specified, it will be the same as *infile* with '.h5' added.

Options:

add options

Gadget Binary -> IRATE

The `gadget2irate` script converts files in the Gadget format to IRATE. Use it as `gadget2irate input-file output-file [options]`.

Options:

`-i, --inits`
Specifies that the input file is an initial conditions file, in which case gas densities and smoothing lengths are skipped.

`-p, --potential`
Specifies that the gravitational potential is contained in the GADGET file, which is controlled via the Makefile of GADGET2. Has no effect if –type2 is enabled.

`-a, --acceleration`
Specifies that the acceleration is contained in the GADGET file, which is controlled via the Makefile of GADGET2. No effect if –type2 is enabled.

`-s, --entropy`
Specifies that the rate of change of entropy of gas particles is included in the GADGET file, which is controlled via the Makefile of GADGET2. Has no effect if –type2 is enabled.

`-t, --timestep`
Specifies that the timestep of each particle is contained in the GADGET file, which is controlled via the Makefile of GADGET2. Has no effect if –type2 is enabled.

`--t0name, --gasname`
Name of the group that the particles in the gas group of the GADGET file is given, under the specified tree.

--t1name, --haloname
Name of the group that the particles in the halo group of the GADGET file is given, under the specified tree.

--t2name, --diskname
Name of the group that the particles in the disk group of the GADGET file is given, under the specified tree.

--t3name, --bulgename
Name of the group that the particles in the bulge group of the GADGET file is given, under the specified tree.

--t4name, --starname
Name of the group that the particles in the star group of the GADGET file is given, under the specified tree.

--t5name, --bndryname
Name of the group that the particles in the boundary group of the GADGET file is given, under the specified tree.

--s8, --sigma8
Specify the cosmology parameter sigma8 for inclusion in the Cosmology group

--ns, --n_s
Specify the Power Spectrum Index for inclusion in the Cosmology group

--snap, --snapshot
Explicitely identify the snapshot number that data will be saved under in the IRATE file, rather than using the GADGET convention

-L
Human readable string the specifies the units of length in the file

-v
Human readable string the specifies the units of velocity in the file

-M
Human readable string the specifies the units of mass in the file

--_unit
Factor to convert units for the property given by the underscore to CGS

--_hfact
The exponent of the reduced Hubble Parameter as it appears in the units for the quantity given by the underscore

--_afact
The exponent of the scale factor as it appears in the units for the quantity given by the underscore

EnBiD -> IRATE

The `enbid2irate` script adds the data from files produced by EnBiD in the GADGET format to existing IRATE files. Use it as `enbid2irate enbid-file irate-file [output-file] [options]`. If *outfile* is not specified, it defaults to irate-file with '-wenbid.hdf5' added. enbid-file should be a binary file as produced by EnBiD, and irate-file should be an existing IRATE file. All data is saved under /Analysis

Warning: This script hasn't been updated to conform to the new IRATE standard, and, as such, should probably not be used yet.

Options:

--t0tree, --gastree
Tree (either gas, dark, or star) that the particles in the gas group of the GADGET file will be saved under.

--t1tree, --halotree
Tree (either gas, dark, or star) that the particles in the halo group of the GADGET file will be saved under.

--t2tree, --disktree
Tree (either gas, dark, or star) that the particles in the disk group of the GADGET file will be saved under.

--t3tree, --bulgetree
Tree (either gas, dark, or star) that the particles in the bulge group of the GADGET file will be saved under.

--t4tree, --startree
Tree (either gas, dark, or star) that the particles in the star group of the GADGET file will be saved under.

--t5tree, --bndrytree
Tree (either gas, dark, or star) that the particles in the boundary group of the GADGET file will be saved under.

--t0name, --gasname
Name of the group that the particles in the gas group of the GADGET file is given, under the specified tree.

--t1name, --haloname
Name of the group that the particles in the halo group of the GADGET file is given, under the specified tree.

--t2name, --diskname
Name of the group that the particles in the disk group of the GADGET file is given, under the specified tree.

--t3name, --bulgename
Name of the group that the particles in the bulge group of the GADGET file is given, under the specified tree.

--t4name, --starname
Name of the group that the particles in the star group of the GADGET file is given, under the specified tree.

--t5name, --bndryname
Name of the group that the particles in the boundary group of the GADGET file is given, under the specified tree.

Gadget Binary + EnBiD -> IRATE

The `gb2-enbid2irate` script combines the gb2irate and enbid2irate scripts into one step. Use it as `gb-enbid2irate gadget-file enbid-file [output-file] [options]`. If *outfile* isn't specified, it defaults to gadget-file with '-enbid-irate.hdf5' added. Takes the same options as gb2irate.

Warning: This script hasn't been updated to conform to the new IRATE standard, and, as such, should probably not be used yet.

AHF ASCII -> IRATE

The `ahf2irate` script converts files in the AHF ASCII format to IRATE. Use it as `ahf2irate input-file-base output-file snapshot-number [options]`. The input file base should be provided as `*.AHF_`; i.e. everything up to the "halos" or "particles" or "profiles" part of the filename.

Options:

--name
The name that will be used for the group that the halo catalog is saved in.

--particles
Include particle data from input-file-base + 'particles'. If given, the data will be saved in a second HDF5 file that will be linked to the main IRATE catalog file.

--profiles
Include radial profiles from input-file-base + 'profiles'. Data will be saved under /Catalog/RadialProfiles/

-p
Include both particles and profiles data; i.e. enables both –particles and –particles.

--nogas
Enable if AHF was not compiled with -DGAS_PARTICLES or code will fail in trying to combine spatial datasets related to gas and star particles.

--paramfile=`<parameter file>`
Specify an AHF parameter file to be saved as attributes to the group that contains the halo catalog.

-s, --size
Maximum size that particle data will be in memory before writing to a file, in GB. Note that this only has an effect if using HDF5 1.8; otherwise, all the particles will be read before anything is written.

--hdf5-16
Force the usage of HDF5 1.6 API, even if 1.8 is found. At present, the benefits that 1.8 provides aren't working, so this is always enabled.

--pos
Human readable string that identifies the units used for position

--vel
Human readable string that identifies the units used for velocity

--mass
Human readable string that identifies the units used for mass

--rad
Human readable string that identifies the units used for radius

--energy
Human readable string that identifies the units used for energy

--phi
Human readable string that identifies the units used for Phi0

--ang
Human readable string that identifies the units used for angular momentum

--___unit
Factor to convert the units for the quantity given by the underscore to CGS

--___hfact
The exponent of the reduced Hubble Parameter as it appears in the units for the quantity given by the underscore

--___afact
The exponent of the scale factor as it appears in the units for the quantity given by the underscore

AHF Particles -> IRATE

The `ahfparticles2irate` script reads an AHF _particles file, saves it to an HDF5 file, then links the resulting datasets to an existing IRATE catalog file. Use it as `ahfparticles2irate particle-file irate-file output-file snapshot-number [options]`. If *output-file* isn't specified, it defaults to irate-file + '-{snapshot number}particles.hdf5'.

Options:

-o, --overwrite
Overwrite the existing output file, if it exists.

-s, --size
Maximum size that particle data will be in memory before writing to a file, in GB. Note that this only has an effect if using HDF5 1.8; otherwise, all the particles will be read before anything is written.

--hdf5-16
Force the usage of HDF5 1.6 API, even if 1.8 is found. At present, the benefits that 1.8 provides aren't working, so this is always enabled.

--name
The name of the group that the halo catalog is saved under. Must match what is already in the file.

Rockstar -> IRATE

The `rockstar2irate` script converts Rockstar halo catalogs to IRATE catalog files. Use it as `rockstar2irate input-file output-file snapshot-number [format] [options]`. If *output-file* isn't specified, it defaults to input-file with '-irate.hdf5' added.

Options:

-b
Specifies that the input file is a Rockstar binary file.

-a
Specifies that the input file is a Rockstar ASCII file.

-n, --name
Specify the name of the group that the halo catalog is to be saved under.

--ns, --n_s
Specify the Power Spectrum Index for inclusion in the Cosmology group

--s8, --sigma8
Specify the cosmological parameter sigma8 for inclusion in the Cosmology group.

CHAPTER 4

Custom Validators/Extending the IRATE Format

The IRATE format includes functionality for providing custom validators to allow more specific checking of a file than just for conformance to the IRATE format (outlined in *IRATE Format Specification*). This is done by subclassing the `irate.validate.Validator` class to make objects that test whether or not a given file conforms to the extended format.

The validator structure is contained within `irate.validate`. The most straightforward way to provide a set of validators for a given file format (called "validator types" in the IRATE documentation and scripts) is to use the custom type directory. This directory is typically $HOME/.irate, but for the exact directory for your platform and machine, call the `irate.validate.find_custom_validator_dir()` function, which will return the directory to place validator type scripts.

Any file that ends in '.py' in the directory will be executed and searched for `Validator` subclasses. Those classes will be automatically added to a type taken from the name of the file (e.g. if the script is `myformat.py`, the validator type will be *myformat*). Normally, you will want to include all the default validators (these ensure the IRATE format is being followed), but if not, add a global variable to the file named *includedefaults* and set it to False.

Once this file has been created (or given to you by someone else and placed in the correct directory), the function `irate.validate.activate_validator_type()` can be called to activator that validator type, or the `iratevalidate` script can be used with the `-t` option (see *Command Line Scripts and Conversion Tools*).

Writing Validator Subclasses

Writing validators is quite straightforward and the full reference is in the docstrings for the `irate.validate.Validator` class. A basic validator simply looks like:

```
from irate.validate import Validator)

class MyGroupNameValidator(Validator):

    groupname = 'MyGroupName'

    def validate(self,grp):
        if 'SpecialData' not in grp:
            self.invalid('SpecialData dataset not present')
```

Validating will then fail for any group that has "MyGroupName" (case-sensitive) as part of its name that does not have a "SpecialData" element (i.e. Group or Dataset). The validator will automatically call the appropriate validator on child groups - all that's necessary is to check if the particular group being validated here meets whatever standards are set for the group. If the group fails, a call to `self.invalid('a message here about the problem')` will properly report the format error.

An additional useful utility is the *`irate.validate.Validator.check_units()`* method. This can be called as `self.check_units(grp['somedatasetname')` to check that a dataset has units. Alternatively, it can be called as `self.check_units(grp['otherdataset'],'unitname)` to check that a particular dataset has units with a particular name. Note that this should be used instead of directly checking attributes on the dataset because the IRATE format supports storing units on groups above the dataset in the HDF5 heirarchy (see *Unit Information* for details on how units are stored in the IRATE format).

CHAPTER 5

Tips for Using IRATE and writing I/O modules

An extremely useful tool for parallel simulations is HDF5's external links. These allow an HDF5 file to be scattered over many logical files while still looking to someone accessing the file as though it were all one file. A typical use case might be a simulation where multiple nodes of a cluster are processing a single snapshot simultaneously. For the sake of example, imagine a simulation where a single snapshot is processed by two seperate nodes that each create hdf5 files "node1.h5" and "node2.h5", that contain datasets containing dark matter particles following the format standard specified in *Particle Data*. An IRATE file might be generated to house these datasets like so:

```
import h5py

#assume simulation.h5 already has the Cosmology and SimulationProperties groups

f = h5py.File('simulation.h5')

snap1 = f.create_group('Snapshot0001')
snap1data = snap1.create_group('ParticleData')

snap1data['Dark_node1'] = h5py.ExternalLink('node1.h5','/')
snap1data['Dark_node2'] = h5py.ExternalLink('node2.h5','/')
```

The file 'simulation.h5' can now be loaded and manipulated just like any IRATE file, and as long as the "node1.h5" and "node2.h5" files are kept in the same directory, everything will work fine.

CHAPTER 6

Reference Guide

This page documents the public function and class library for the IRATE format tool package. These modules can be used by other programs or users to simplify various processing tasks relating to the IRATE format. The following modules are present:

- *`core`*

  General tools of use with the IRATE format, such as creating standard sections or quickly extracting particles. Note that these functions should be imported directly from the *`irate`* namespace.

- *`validate`*

  Code for testing if a file obeys the IRATE format and tools for extending this process (see *Custom Validators/Extending the IRATE Format*)

- *`ahf`*
- *`gadget`*
- `enbid`
- *`rockstar`*
- `tipsy`

  These modules all contain functions used by the *Command Line Scripts and Conversion Tools* to convert files in other formats from or to IRATE. This documentation is provided to allow other programs to easily script such conversions without using the command line tools.

Not that the base *`irate`* module contains most of the *`irate.core`* functionality, as well as `__version__`, which contains a string with the version of the IRATE tools, and `formatversion`, which contains an integer specifying the format version.

irate.core Module

The *`irate.core`* module contains the core utilities used by IRATE. These include a variety of convinience functions for working with IRATE format files. While they are technically part of the *`irate.core`* module, they are all also included in the base irate namespace, so the best way to import these is to do:

```
from irate import create_irate
```

rather than importing from *irate.core*.

`irate.core.`**`add_cosmology`**(*iratefile*, *omegaM=None*, *omegaL=None*, *h=None*, *s8=None*, *ns=None*, *omegaB=None*, *cosmoname=None*, *update=True*, *verbose=False*)

Creates or updates a cosmology section of an IRATE file.

This function creates an IRATE file or opens it if it already exists, and either adds the given cosmology-related values to it, checks that they match with what's in the existing file, or updates the cosmology with the given values.

Parameters

- **iratefile** – The name of the file to open or a `h5py.File` object.
- **omegaM** (*float*) – The present day matter density or None to skip this parameter.
- **omegaL** (*float*) – The present day dark energy density or None to skip this parameter.
- **h** (*float*) – The reduced Hubble Parameter $h = H_0/100$ or None to skip this parameter.
- **s8** (*float*) – σ_8, the amplitude of the power spectrum at 8 Mpc/h, or None to skip this parameter.
- **ns** (*float*) – The index of the primordial power spectrum or None to skip this parameter.
- **omegaB** (*float*) – The present day baryon density or None to skip this parameter
- **cosmoname** (*str*) – The name of the cosmology as a string or None to skip this value. This attribute is optional in the IRATE format.
- **update** (*bool*) – If an existing file is given and this is True, the parameters will be updated with those given to this function. Otherwise, the file will be *checked* to ensure the file matches. If this is False and the file does not exist, an exception will be raised.
- **verbose** (*bool*) – If True, informational messages are printed as the function does various actions.

Raises

- **TypeError** – If the *iratefile* input is invalid
- **ValueError** – If update is False and the file's values don't match those passed into the function, or the file does not exist.
- **KeyError** – If update is False and there are values passed that aren't in the file.

`irate.core.`**`add_standard_cosmology`**(*iratefile*, *cname*, *update=True*, *verbose=False*)

Adds or updates a cosmology from the name of a standard cosmology.

The *cname* parameter gives the name of the cosmology to be used from the list below, while all other parameters are the same as for the *add_cosmology()* function.

- 'WMAP7'

  LCDM Cosmological parameters for the 7-year WMAP data (Komatsu et al. 2011). For details, see http://lambda.gsfc.nasa.gov/product/map/dr4/params/lcdm_sz_lens_wmap7.cfm

- 'WMAP5'

  LCDM Cosmological parameters for the 5-year WMAP data (Komatsu et al. 2009). For details, see http://lambda.gsfc.nasa.gov/product/map/dr3/params/lcdm_sz_lens_wmap7.cfm

- 'WMAP3'

  LCDM Cosmological parameters for the 3-year WMAP data (Spergel et al. 2007). For details, see http://lambda.gsfc.nasa.gov/product/map/dr2/params/lcdm_sz_lens_wmap7.cfm

- 'WMAP1'

  LCDM Cosmological parameters for the 1-year WMAP data. For details, see Spergel et al. 2003.

`irate.core.`**`get_irate_particle_nums`** (*fn*, *validate=True*)

Convinience function to determine the number of particles of each type in an IRATE format file.

Parameters

- **`fn`** (*`str`*) – The filename of an IRATE formatted file or a `h5py.File` object.
- **`validate`** (*`bool`*) – If True, the file will be validate as IRATE format. If 'strict', strict validation will be used. Otherwise no validation will be performed.

Returns a 3-namedtuple (ndark,nstar,ngas)

`irate.core.`**`get_irate_catalog_halo_nums`** (*fn*, *validate=True*)

Convinience function to determine the number of entries in an IRATE halo catalog file.

Parameters

- **`fn`** (*`str`*) – The filename of an IRATE halo catalog file or a `h5py.File` object.
- **`validate`** (*`bool`*) – If True, the file will be validate as IRATE halo catalog format.

Returns an integer with the total number of halos

`irate.core.`**`create_irate`** (*fn*, *dark*, *star*, *gas*, *headername*, *headerdict=None*, *compression=None*)

A Convinience function to create an IRATE format file data as numpy arrays. This will overwrite a currently existing file.

Parameters

- **`fn`** (*`str`*) – The file name to use for the new file
- **`dark`** – The data for the dark matter particles. Should be a dict with keys 'Position', 'Velocity', 'Mass', (and possibly others) mapping to `numpy.ndarray` arrays. Alternatively, it can be a structured array (i.e. dtype with names and formats) with the names of the dtype giving the names of the resulting datasets.
- **`star`** – The data for the star particles. See *dark* for format.
- **`gas`** – The data for the gas particles. See *dark* for format.
- **`headername`** (*`str`*) – The group name to use for the header entry or None to create no header.
- **`headerdict`** – A dictionary mapping strings to attribute values. These will be used for the attributes of the header entry. If None, an attribute will be added to indicate header content is missing.
- **`compression`** – The type of compression to use for the datasets. Can be any of the compression that h5py supports on your system ('gzip','lzf', or 'szip') or None to do no compression.

Returns The `h5py.File` of the newly-created file (still open).

Raises

- ***`IRATEFormatError`*** – If any of the data arrays are missing the necessary fields.
- **`ValueError`** – If the input data is invalid.

`irate.core.`**`get_all_particles`**(*fn*, *ptype*, *dataname*, *subsample=None*, *validate=True*)

Convinience function to build an array with *all* the particles of a requested type or types (i.e. all subgroups, if present, will be visited).

Parameters

- **`fn`** (*`str`*) – The filename of an IRATE formatted file or a `h5py.File` object.
- **`ptype`** (*`str`*) – The particle type: 'Dark','Star', 'Gas', a sequence of a mix of those three, or None for all particles of all types.
- **`dataname`** (*`str`*) – The dataset name to extract (e.g. 'Position', 'Mass')
- **`subsample`** (*`int`*) – Gives the stride of the data - e.g. each array will be spliced to only return every *subsample* data point. If None, all the data will be returned.
- **`validate`** (*`bool`*) – If True, the file will be validate as IRATE format. If 'strict', strict validation will be used. Otherwise no validation will be performed.

Returns A 3-tuple (data,grpidx,grpmap) where *data* is the data arrays, *grpidx* is an int array with the same first dimension as *data*, and *grpmap* is a dictionary mapping the values in *grpidx* to group names.

Raises **`ValueError`** – If a group does not have the requested *dataname*.

`irate.core.`**`scatter_files`**(*infile*, *outfile=None*, *splitgroups=False*)

Splits a single IRATE file into separate files for each dataset.

Parameters

- **`infile`** – A file name or an *h5py.File* object to be scattered.
- **`outfile`** – The filename or an *h5py.File* object to use as the new base IRATE format file or None to overwrite the old name. Other output files will be placed in the same directory as this file.
- **`splitgroups`** (*`bool`*) – If True, also split groups into separate files.

Returns A list of filenames that were created with the base name as the first.

`irate.core.`**`gather_files`**(*infile*, *outfile=None*)

Combine all external file that are part of a master IRATE file into a single monolithic file.

Note: This function can be driven from the command line via the `irategather` script. See *Command Line Scripts and Conversion Tools* for details.

Parameters

- **`infile`** – A file name or an *h5py.File* object to be gathered.
- **`outfile`** – The filename or an *h5py.File* object to use as the new IRATE format file or None to overwrite the old name.

Returns (outfilename,filelist) where *outfilename* is the name of the output file, and *filelist* is a list of all the files gathered together to create this file.

`irate.core.`**`get_metadata`**(*dataset*, *metadataname*)

Gets a metadata value from an IRATE file dataset.

Parameters

- **`dataset`** – An `Dataset` object in an IRATE file.

- **`metadataname`** (*`str`*) – The name of the meteadata field requested.

Returns The value of the metadata requested, or None if the metadata with the requested name does not exist.

Raises **`TypeError`** – If the *dataset* is not a dataset

`irate.core.`**`set_metadata`**(*dataset*, *metadataname*, *value*, *group=None*)

Sets a metadata value for an IRATE file dataset.

Parameters

- **`dataset`** – An `Dataset` object in an IRATE file.
- **`metadataname`** (*`str`*) – The name of the meteadata field requested.
- **`value`** – The name of the meteadata field requested.
- **`group`** – The group this metadata should be set for, or None.

Note: Any other datasets with the same as this one below this group in the heirarchy will be assigned the same units (unless that dataset overrides them itself).

Raises

- **`TypeError`** – If *dataset* is not a dataset.
- **`ValueError`** – If *group* is not a parent of *dataset*.

Note: A `MetadataHiddenWarning` will be issued as a warning if the metadata for this dataset has already been set somewhere lower down in the heirarchy than the requested *group*. The result of this is that this function will not alter the resulting units for the *dataset* because it will be overshadowed by the metadata setting further down. (see `warnings` for an explanation of warnings)

`irate.core.`**`get_units`**(*dataset*)

Gets the units for a dataset in an IRATE file.

Because units are typically expressed in factors of the hubble constant and may or may not be comoving or otherwise varying with scale factor, two additional factors are needed beyond the raw unit conversion. To use these together, the appropriate factor to multiple the dataset by to get to physical units for an assumed hubble parameter *h* and scale factor *a* is: *tocgs'*h^'hubbleexponent'*a^'scalefactorexponent* . Hence, for example, comoving Mpc/h would have *tocgs* =3.0857e24 , *hubbleexponent* =-1, and *scalefactorexponent* =1 .

Parameters **`dataset`** – A `Dataset` object in an IRATE file

Returns *name,tocgs,hubbleexponent,scalefactorexponent*. *name* is the name of the unit for this dataset, *tocgs* is a factor to multiply the dataset by to get cgs units, *hubleexponent* is the exponent of the reduced hubble parameter (h=H0/100) for this unit, and *scalefactorexponent* is the exponent describing how this unit varies with the scale factor. Alternatively, if no unit is found, `None` is returned.

Note: I f you are not working with a file where you know for sure what the units are, is important to check whether or not the unit is None, because the IRATE format does not require units for all datasets, as some quantities are dimensionless. The easiest way to do this is:

```
res = get_units(mydataset)
if res is None:
    ... do something if it is unitless ...
else:
    nm,tocgs,hexp,sfexp = res
    ... do something with the units ...
```

Raises **`TypeError`** – If *dataset* is not a dataset

`irate.core.`**`set_units`**(*dataset*, *unitname*, *tocgs*, *hubbleexponent*, *scalefactorexponent*, *unitgroup=None*)

Sets the units of a dataset to the provided values.

Because units are typically expressed in factors of the hubble constant and may or may not be comoving or otherwise varying with scale factor, two additional factors are needed beyond the raw unit conversion. To use these together, the appropriate factor to multiple the dataset by to get to physical units for an assumed hubble parameter *h* and scale factor *a* is: *tocgs‘*h^‘hubbleexponent‘*a^‘scalefactorexponent* . Hence, for example, comoving Mpc/h would have *tocgs* =3.0857e24 , *hubbleexponent* =-1, and *scalefactorexponent* =1 .

Parameters

- **`dataset`** – The dataset for which to set the units.
- **`unitname`** (*`str`*) – A human-readable name of the unit.
- **`tocgs`** (*`str`*) – A numerical factor to multiply the dataset by to convert to cgs units.
- **`hubbleexponent`** (*`str`*) – The exponent of the hubble parameter for this unit.
- **`scalefactorexponent`** (*`str`*) – The exponent of the scale factor for this unit.

Raises **`TypeError`** – If an input is not the correct type

`irate.core.`**`get_cgs_factor`**(*dataset*, *h=0.7*, *a=1*)

Retrieves the factor to multiply by the dataset to convert to physical CGS units at a given scale factor and hubble constant.

Parameters

- **`dataset`** – An `Dataset` for which to read the units.
- **`h`** – Reduced hubble parameter to assume for the conversion.
- **`a`** – Scale factor to assume for the conversion = 1/(z+1).

Raises

- **`TypeError`** – If *dataset* is not a dataset
- **`ValueError`** – If the dataset is dimensionless

irate.validate Module

The *`irate.validate`* module contains classes and functions for testing if a file conforms to the IRATE specification. It also contains tools for easily extending the validator to allow 3rd parties to provide validator modules for their formats.

See *Custom Validators/Extending the IRATE Format* for more details on how to write custom extensions to the IRATE format.

exception `irate.validate.`**`IRATEFormatError`**(*msg*, *h5grpname=None*, *validator=None*)
This exception is raised if a file is loaded that does not conform to the IRATE Format.

class `irate.validate.`**`RootValidator`**(*parentvalidator*, *printmsg=None*, *immediatefail=None*)
Validates the root level of an IRATE file.

class `irate.validate.`**`Validator`**(*parentvalidator*, *printmsg=None*, *immediatefail=None*)
The base class for classes that are intended to validate IRATE format files.

To implement a validator, a subclass *must* override `validate()`, which will be valled to validate a `Group`. If not, an object of the subclass will not be able to be created, as this is an abstract class. See the `Validator.validate` docstring for the correct way to report format errors when they are encountered.

A subclass should also define a *groupname* attribute at the class level - that name will be used to identify groups that are to be assigned to that validator.

`check_units`(*dataset*, *unitname=None*, *invalidate=True*)
Check the units on a dataset

Parameters

- **`dataset`** – The `Dataset` to check for units.
- **`unitname`** – The name the unit should have, or None to just check if any units are present for this dataset.
- **`invalidate`** (*`bool`*) – If True, `invalid()` will be called if the check fails with a message indicating a unit check failed.

Returns True if the unit check succeeded, False if it failed because the name of the units does not match *unitname*, or None if it failed because no units are present.

Note: If the *dataset* provided is not a dataset, this method will call `invalid()` with a message indicating that this occured.

`validate`(*grp*)
Subclasses should override this - it is called to validate a particular `h5py.Group` that is matched to this validator.

If a problem is encountered, the validator should call `Validator.invalid()` with a message describing the problem. If a dataset on the validated group should have units, the `check_units()` method should be called on the dataset to check for units (or a specific unit name, if desired)

After this method finishes, all subgroups will also be validated - to skip validation for some subgroups, set the *skipsubgroups* attribute of this object to either a list of names to skip, or `True` to not validate any subgroups.

Thus, a simple example might be:

```python
def validate(self,grp):
    if 'somegroup' not in grp:
        self.invalid('somegroup missing!')
    self.skipsubgroups = ['ignorethisgroupname']
    self.check_units(grp['datasetwunits'])
    self.check_units(grp['distance'],'kpc')
```

`irate.validate.`**`activate_validator_type`**(*tnames*, *defaultlast=True*)
Sets the active list of validators to those for the requested type.

Parameters

- **tnames** – The name of the type (group of validators) to activate or a list of types to activate. Note that the order of types determines the order in which validators are checked for matching a particular group in an IRATE file.
- **defaultlast** – If True, when a list of types is given and the default validators are present, the default validators will be set to run only after all other validators have been run.

Raises `KeyError` – If *tname* is not a validator type name.

`irate.validate.`**`add_validator_type`**`(`*tname*, *validators*, *includedefaults=True*`)`

Adds a set of validator classes as a type.

Parameters

- **tname** – The name for this type (group of validators).
- **validators** – A sequence of subclasses of `Validator` to use as validators for this type.
- **includedefaults** (`bool`) – If True, the default validators are added after the supplied validators. This is usually what you want because otherwise there's no guarantee the file will actually follow the IRATE format.

Raises `TypeError` – If something in *validators* is not a `Validator` subclass.

`irate.validate.`**`find_custom_validator_dir`**`()`

Identifies and returns the directory that stores custom validators.

Returns The directory where validator files are supposed to live

Raises `OSError` – If no suitable directory can be found.

`irate.validate.`**`find_validator`**`(`*groupname*`)`

Locates the appropriate validator based on the given group name.

Parameters **groupname** – The name of the group to match against a `Validator` *groupname*.

Returns The class matched to the provided group name or False if the search failed.

`irate.validate.`**`get_validator_types`**`(`*tname=None*`)`

Returns a list of all the validator types available, or all the validator classes associated with a given validator type.

Parameters **tname** – If None, returns a list of validator types, or if a validator type, returns a list of `Validator` subclasses for that type.

Returns A list of types if *tname* is None or a list of `Validator` subclasses

Raises `KeyError` – If *tname* is not a validator type name.

`irate.validate.`**`register_validator`**`(`*valcls*, *front=True*`)`

Registers a `Validator` class for use by the `find_validator()` function.

Parameters

- **valcls** – A class that is a subclass of `Validator`. The matching group names will be inferred from its *groupname* attribute.
- **front** – If True, the given validator will be registered in front of others - e.g. a group that matches its name will not move on to the others. Otherwise it will be placed in the back.

`irate.validate.`**`remove_validator_type`**`(`*tname*`)`

Removes a set of type validators. :param tname: The name of the type to have its validators removed.

Raises `KeyError` – If *tname* is not a validator type name.

`irate.validate.`**`validate_file`**(*fnorfile*, *verbose=False*, *immediatefail=False*)

Tests whether or not the file conforms to the IRATE format.

Parameters

- **`fnorfile`** – The filename or `h5py.File` to test.
- **`verbose`** (*`bool`*) – If True, informational messages will be printed to stdout, and if False no messages will be printed. Can also be a callable *f(str)* that will be called each time a message is to be printed.
- **`immediatefail`** (*`bool`*) – If True, an exception is raise when the first invalid aspect of the file is encountered. Otherwise, the function completes and returns all errors.

Returns A list of errors encountered (or an empty list if valid).

Raises ***`IRATEFormatError`*** – If any part of the file does not conform to the standard and *immediatefail* is True.

irate.ahf Module

`irate.ahf.`**`ahf_halos`**(*fname, outname, snapnum, name='HaloCatalog_AHF', posname='comoving Mpc/h', posunit=[3.08568025e+24, -1, 1], velname='km/s', velunit=[100000.0, 0, 0], mname='M_sun/h', munit=[1.98892e+33, -1, 0], radname='comoving kpc/h', radunit=[3.08568025e+21, -1, 1], enname='M_sun/h*(km/s)^2', enunit=[1.98892e+43, -1, 0], phiname='(km/s)^2', phiunit=[10000000000.0, 0, 0]*)

Reads an AHF_halos file, then collimates it such that a list of column headers and a list of arrays, each of which corresponds to a column. All entries in the columns will be in the same order–i.e. the ith entry in column x corresponds to the same halo as the ith entry in column y.

Parameters

- **`fname`** – The file to be read; should be in ASCII format with column headers preceeded by a #
- **`outname`** – The IRATE format file to either be created or to have data added to it, if it already exists.
- **`snapnum`** (*`int`*) – Specifies the snapshot number that the data will be saved under in the IRATE file; that is, the data ends up in the group "Snapshot"+snapnum
- **`name`** – Specifies the name of the group that holds the datasets. Data is saved under "Snapshot"+snapnum+"/"+name
- **`_____name`** – A human readable string that identifies the units used for the property specified by ____ (position, velocity, mass, radius, energy, and phi0)
- **`_____units`** – A three element array that gives, in order, the conversion factor between the units used for the property given by * to CGS, the exponent on the reduced Hubble Parameter that appears in that unit, and the exponent on the scale factor that appears in that unit.

Returns An array that contains the number of bins used for each halo in the radial profiles file, for the purpose of reading the .AHF_profiles file.

`irate.ahf.`**`ahf_param`**(*pfile*, *outname*, *snapnum*, *name='HaloCatalog_AHF'*)

Reads an AHF .parameter file and adds it to an existing IRATE file as attributes to the /CatalogHeader/AHF group.

Parameters

- **pfile** – The parameter file to be read. Should be standard AHF ASCII parameter file format.
- **outname** – The IRATE file to store the parameters in.

`irate.ahf.`**`ahf_particles`** (*fname*, *outname*, *snapnum*, *pfilename=None*, *name='HaloCatalog_AHF'*)

Read a .AHF_particles file and save an array with all the particles in it in order, along with a list of the number of particles in each halo to an IRATE file. If there's a particle identifier, also save an array with the particle type.

uses external progrms to re-format the data to make the read quicker.

Parameters

- **fname** – The .AHF_particles file to be read. Must be in ASCII format
- **outname** – The IRATE file that the data is linked to
- **pfilename** – The HDF5 file that the data is saved in
- **snapnum** – The snapshot number that identifies the group that the halo catalog belongs to in the IRATE file
- **name** – The name of the halo catalog in the IRATE file that the particles correlate to.

`irate.ahf.`**`ahf_profiles`** (*fname, nhalos, nbins, outname, snapnum, name='HaloCatalog_AHF', radname='comoving kpc/h', radunits=[3.08568025e+21, -1, 1], mname='M_sun/h', munits=[1.98892e+33, -1, 0], velname='km/s', velunits=[100000.0, 0, 0], angname='(M_sun/h)*(Mpc/h)*(km/s)', angunits=[6.137171162830001e+62, -2, 1], enname='M_sun/h*(km/s)^2', enunits=[1.98892e+43, -1, 0]*)

Read an ASCII .AHF_profiles file, collimate it, and return a list of data and a list of column headers. The list of data is a list over columns, then each halo has it's own list within the columns.

Parameters

- **fname** – The ASCII format .AHF_profiles file to be read.
- **nhalos** (*int*) – The number of halos contained in the file
- **nbins** – An array or list that tells you the number of radial bins for each halo
- **outname** – The existing IRATE file to save radial profile data to
- **snapnum** (*int*) – The number of the snapshot that the halo catalog that this data belongs to is under in the existing IRATE file
- **name** – The name of the group that contains the halo catalog
- **_____name** – A human readable string that gives the units for the given property: radius, mass, velocity, angular momentum, or energy
- **_____units** – A three element array, the first of which tells the conversion factor between the units of * and CGS, the second of which gives the exponent on h as it appears in those units, and the third gives the exponent on a as it appears in those units.

`irate.ahf.`**`awked_ahf_particles`** (*fname*, *outname*, *snapnum*, *filename=None*, *name='HaloCatalog_AHF'*)

Read a .AHF_particles file that has been run through the awk script that accompanies IRATE, which is faster, and save an array with all the particles in it in order, along with a list of the number of particles in each halo to an IRATE file. If there's a particle identifier, the array is Nx2 with the identifier in the second column.

Parameters

- **fname** – The .AHF_particles file to be read. Must be in ASCII format

- **outname** – The IRATE file that the data is linked to
- **pfilename** – The HDF5 file that the data is saved in
- **snapnum** – The snapshot number that identifies the group that the halo catalog belongs to in the IRATE file
- **name** – The name of the halo catalog in the IRATE file that the particles correlate to.

`irate.ahf.`**`read_and_write_particles`**(*infile*, *outfile*, *maxsize*)

The HDF5 v 1.8 method of reading and writing particles should allow me to use a lot less memory by periodically writing to the file (since v 1.8 allows for appending to datasets). Note that this will probably be slower than the 1.6 method, at least for small files, but if memory is a problem, this is probably the way to go.

Warning: THIS DOESN'T WORK RIGHT NOW. I can't get the appending to work, so if you desperately want to use this function, you'll have to fix it. In fact, I should probably just delete it, but I always hate doing that Furthermore, since it isn't working, it hasn't been updated to work adhere to the new IRATE format, so this function absolutely does NOT work.

Parameters

- **infile** – The AHF_particles file to read. Must be in ASCII format.
- **outfile** – The IRATE catalog file to save the particle data to.
- **maxsize** – The maximum amount of memory that can be used at any given time, in GB.

irate.gadget Module

`irate.gadget.`**`gadget_hdf5_to_irate`**(*inname, outname, snapnum, t0_name='Gas', t1_name='Dark_Halo', t2_name='Dark_Disk', t3_name='Dark_Bulge', t4_name='Star', t5_name='Dark_Boundary', s8=None, ns=None, omegaB=None, lenient=False, lname='comoving Mpc/h', lunits=[3.08568025e+24, -1, 1], vname='(km/s)*sqrt(a)', vunits=[100000.0, 0, 0.5], mname='1e10 M_sun/h', munits=[1.98892e+43, -1, 0]*)

Transforms Gadget type 3 (HDF5) snapshots into a format that meets the IRATE specifications. This will automatically check for Makefile enabled blocks.

Parameters

- **inname** – Gadget type 3 snapshot to convert to IRATE
- **outname** – IRATE file to output
- **snapnum** (*int*) – The number to add to 'Snapshot' that becomes the name of the group that particle data is added to.
- **#name** – Name of group that particles of type # are given
- **s8** (*float*) – sigma_8, for the purposes of adding it to the Cosmology group
- **ns** (*float*) – n_s, for the purposes of adding it to the Cosmology group
- **omegaB** (*float*) – omegaB, for the purposes of adding it to the Cosmology group
- **lenient** (*bool*) – allow for discrepancies in SimulationProperties. discrepancies will be printed, but won't halt the conversion. old values will be retained.

- **___name** – A human readable string that defines the units for either length (l), velocity (v), or mass (m)
- **___units** – A three-element array. The first entry defines the conversion factor to convert either length (l), velocity (v), or mass (m) to CGS units. The second element is the exponent on the reduced Hubble Parameter that appears in that unit, and the third is the exponent on the scale factor that appears in that unit.

# refers to the same names as Gadget2:

- 0 = gas particles
- 1 = halo particles
- 2 = disk particles
- 3 = bulge particles
- 4 = star particles
- 5 = bndry particles

`irate.gadget.`**`gadget_multihdf5_to_irate`**(*inbase, outname, snapnum, t0_name='Gas', t1_name='Dark_Halo', t2_name='Dark_Disk', t3_name='Dark_Bulge', t4_name='Star', t5_name='Dark_Boundary', s8=None, ns=None, omegaB=None, lenient=False, lname='comoving Mpc/h', lunits=[3.08568025e+24, -1, 1], vname='(km/s)*sqrt(a)', vunits=[100000.0, 0, 0.5], mname='1e10 M_sun/h', munits=[1.98892e+43, -1, 0]*)

Transform a Gadget type 3 (HDF5) snapshot that is split into multiple blocks into a format that meets the IRATE specifications. This will automatically check for Makefile enabled blocks.

Parameters

- **inbase** – Wildcard basename of a Gadget type 3 snapshot with multiple blocks to convert to IRATE
- **outname** – IRATE file to output
- **snapnum** (*int*) – The number to add to 'Snapshot' that becomes the name of the group that particle data is added to.
- **#name** – Name of group that particles of type # are given
- **s8** (*float*) – sigma_8, for the purposes of adding it to the Cosmology group
- **ns** (*float*) – n_s, for the purposes of adding it to the Cosmology group
- **omegaB** (*float*) – omegaB, for the purposes of adding it to the Cosmology group
- **lenient** (*bool*) – allow for discrepancies in SimulationProperties. discrepancies will be printed, but won't halt the conversion. old values will be retained.
- **___name** – A human readable string that defines the units for either length (l), velocity (v), or mass (m)
- **___units** – A three-element array. The first entry defines the conversion factor to convert either length (l), velocity (v), or mass (m) to CGS units. The second element is the exponent on the reduced Hubble Parameter that appears in that unit, and the third is the exponent on the scale factor that appears in that unit.

# refers to the same names as Gadget2:

- 0 = gas particles
- 1 = halo particles
- 2 = disk particles
- 3 = bulge particles
- 4 = star particles
- 5 = bndry particles

`irate.gadget.`**`gbin2_to_irate`**(*inname, outname, snapnum, t0_name='Gas', t1_name='Dark_Halo', t2_name='Dark_Disk', t3_name='Dark_Bulge', t4_name='Star', t5_name='Dark_Boundary', s8=None, ns=None, omegaB=None, fprec=4, zfields=3, lenient=False, lname='comoving Mpc/h', lunits=[3.08568025e+24, -1, 1], vname='(km/s)*sqrt(a)', vunits=[100000.0, 0, 0.5], mname='1e10 M_sun/h', munits=[1.98892e+43, -1, 0]*)

Reads a GADGET type 2 snapshot file block by block (e.g. coordinate block for gas particles), writes the block to an IRATE formate HDF5 file, and then deletes that block from memory. If a given block identifier isn't recognized, one of two things will happen: either the script will figure out which particles that data belongs to and add it automatically with the dataset named according to the label used, or if it's not obvious what particles the data belongs to, the user will be given the option to skip it or to exit entirely.

Parameters

- **`inname`** – The name of the gadget binary file to be read
- **`outname`** – The name of the IRATE file to be written,
- **`snapnum`** (*`int`*) – The number to add to 'Snapshot' that becomes the name of the group that particle data is added to.
- **`t#_name`** – Determines the name of the group that contains the data from #
- **`s8`** (*`float`*) – sigma_8, for the purposes of adding it to the Cosmology group
- **`ns`** (*`float`*) – n_s, for the purposes of adding it to the Cosmology group
- **`omegaB`** (*`float`*) – omegaB, for the purposes of adding it to the Cosmology group
- **`fprec`** – Precision of the file. Single (4) or double (8). Assumed single precision. Warning! Right now changing this parameter to double prec (8) will probably not work but I hope it is a good start point.
- **`zfields`** – Fields in the Z block for each particle.
- **`lenient`** (*`bool`*) – allow for discrepancies in SimulationProperties. discrepancies will be printed, but won't halt the conversion. old values will be retained.
- **`___name`** – A human readable string that defines the units for either length (l), velocity (v), or mass (m)
- **`___units`** – A three-element array. The first entry defines the conversion factor to convert either length (l), velocity (v), or mass (m) to CGS units. The second element is the exponent on the reduced Hubble Parameter that appears in that unit, and the third is the exponent on the scale factor that appears in that unit.

# refers to the same names as Gadget2:

- 0 = gas particles
- 1 = halo particles

- 2 = disk particles
- 3 = bulge particles
- 4 = star particles
- 5 = bndry particles

`irate.gadget.`**`gbin_to_irate`**(*inname, outname, snapnum, potential=False, accel=False, entropy=False, timestep=False, t0_name='Gas', t1_name='Dark_Halo', t2_name='Dark_Disk', t3_name='Dark_Bulge', t4_name='Star', t5_name='Dark_Boundary', ics=False, s8=None, ns=None, omegaB=None, lenient=False, lname='comoving Mpc/h', lunits=[3.08568025e+24, -1, 1], vname='(km/s)*sqrt(a)', vunits=[100000.0, 0, 0.5], mname='1e10 M_sun/h', munits=[1.98892e+43, -1, 0]*)

Reads a GADGET format file block by block (e.g. coordinate block for gas particles), writes the block to an IRATE formate HDF5 file, and then deletes that block from memory.

param inname The name of the gadget binary file to be read

param outname The name of the IRATE file to be written

param int snapnum The number to add to 'Snapshot' that becomes the name of the group that particle data is added to.

param bool potential True to read the Makefile enabled potential block

param bool accel True to read the Makefile enabled acceleration block

param bool entropy True to read the Makefile enabled dA/dt block

param bool timestep True to read the Makefile enabled timestep block

param t#_name Determines the name of the group that contains the data from #

param bool ics True if the file being converted is an initial conditions file, in which case the gas density and smoothing length blocks won't be looked for.

param float s8 sigma_8, for the purposes of adding it to the Cosmology group

param float ns n_s, for the purposes of adding it to the Cosmology group

param bool lenient allow for discrepancies in SimulationProperties. discrepancies will be printed, but won't halt the conversion. old values will be retained.

param ___name A human readable string that defines the units for either length (l), velocity (v), or mass (m)

param ___units A three-element array. The first entry defines the conversion factor to convert either length (l), velocity (v), or mass (m) to CGS units. The second element is the exponent on the reduced Hubble Parameter that appears in that unit, and the third is the exponent on the scale factor that appears in that unit.

# refers to the same names as Gadget2:

- 0 = gas particles
- 1 = halo particles
- 2 = disk particles
- 3 = bulge particles
- 4 = star particles

- 5 = bndry particles

`irate.gadget.`**`irate_to_gbin`**(*inname*, *outname*, *snapnum*)

Converts a single snapshot in an IRATE format file into a SnapFormat = 1 Gadget binary file. If there are groups with "Star" in the name, they're assumed to contain star particles and will be placed in Gadget group 4; likewise, if there are groups with "Gas" in the name, their particles will be placed in Gadget group 0. Dark matter particles with the lowest mass will be plced in group 1 and all other particles will be placed in Gadget group 5.

Parameters

- **`inname`** – The input IRATE format file.
- **`outname`** – The name of the output Gadget file
- **`snapnum`** (*`int`*) – The integer identifying where the data is stored; that is, particle data is expected to be under '/Snapshot{snapnum}/ParticleData.

`irate.gadget.`**`read_gbin`**(*fname*, *potential*, *accel*, *entropy*, *timestep*)

Reads a GADGET format file and returns the data in lists, one for each type of particle that contains arrays for each bit of data about each particle.

Warning: Currently unused in the IRATE suite, and definitely not maintained either. Use at your own risk.

Parameters

- **`fname`** – The name of the gadget binary file to be read
- **`potential`** (*`bool`*) – True to read the Makefile enabled potential block
- **`accel`** (*`bool`*) – True to read the Makefile enabled acceleration block
- **`entropy`** (*`bool`*) – True to read the Makefile enabled dA/dt block
- **`timestep`** (*`bool`*) – True to read the Makefile enabled timestep block

irate.enbid Module

irate.rockstar Module

`irate.rockstar.`**`rockstar_ascii`**(*fname*, *outname*, *snapnum*, *name='HaloCatalog_Rockstar'*, *s8=None*, *ns=None*, *omegaB=None*)

Reads a Rockstar ASCII file and collimates it, then saves the data in an IRATE format file (creating one if it doesn't exist).

Parameters

- **`fname`** – The input ASCII file to read and collimate.
- **`outname`** – The IRATE file to save the data in
- **`snapnum`** – The integer snapshot number, which the data will be saved under in the IRATE file.
- **`name`** – The name of the group that will contain the datasets that hold the data
- **`s8`** – sigma_8, for inclusion in the cosmology group. May be None.
- **`ns`** – Power Spectrum Index, for inclusion in the cosmology group. May be None.

irate.rockstar.**rockstar_param**(*fname*, *outname*, *snapnum*, *name='HaloCatalog_Rockstar'*)

This functions read in a Rockstar .cfg file, parses it, and then saves it's contents as attributes to the group that the halo catalog is expected to be saved in; that is /Snapshot{snapnum}/{name} in {outname}.

Parameters

- **fname** – The ASCII rockstar.cfg file to open and read.
- **outname** – The existing IRATE file to add the attributes to
- **snapnum** – Identifies the snapshot number that the halo catalog that rockstar.cfg belongs to.
- **name** – The name of the group that holds the halo catalog that rockstar.cfg belongs to.

irate.tipsy Module

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index