Astrophysix

Release 0.4.1

Damien CHAPON

Dec 08, 2020

DOCUMENTATION

CHAPTER

ONE

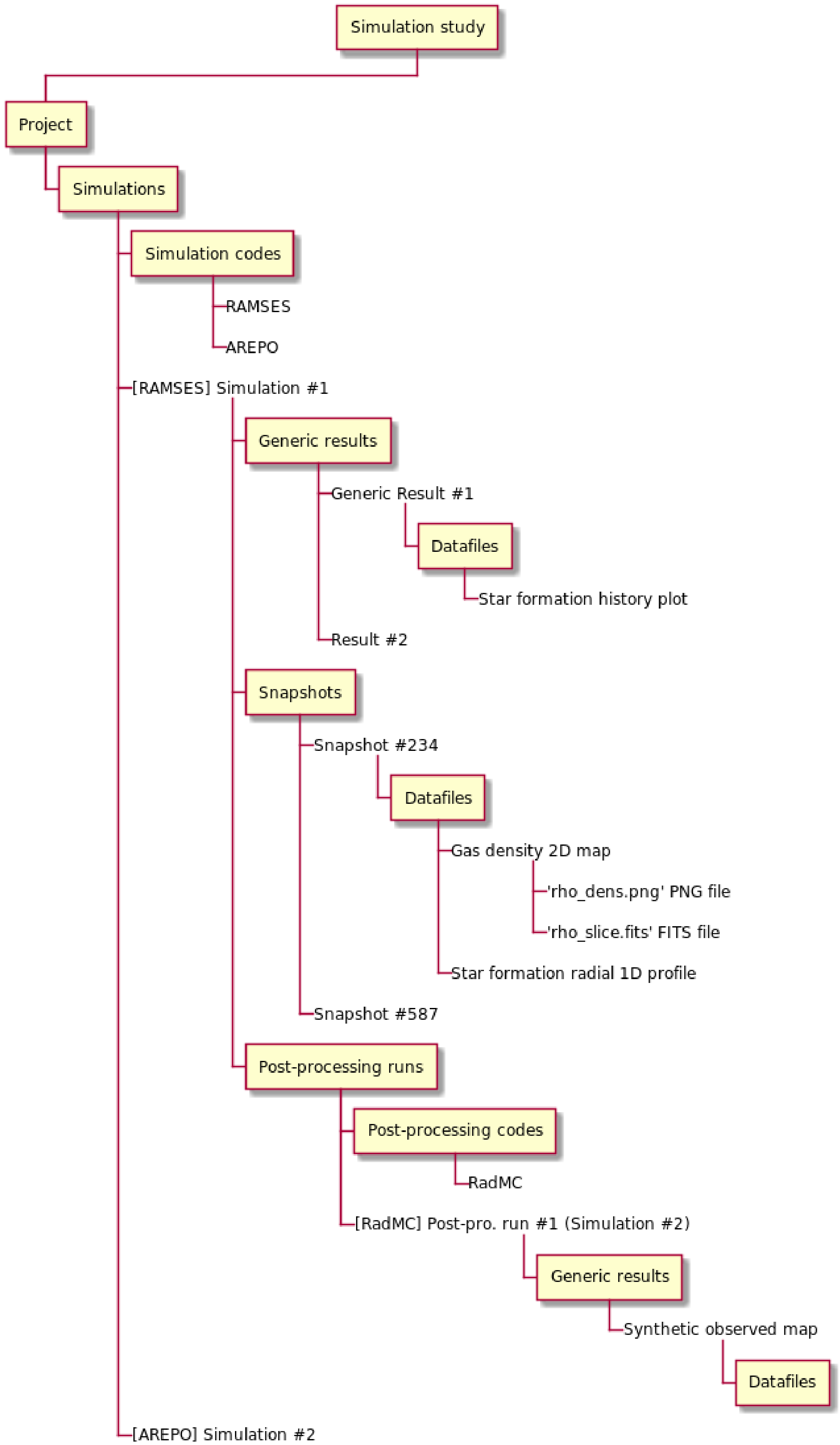
INTRODUCTION

The purpose of the astrophysix Python package is to provide computational astrophysicists a generic tool to document their numerical projects :

- the astrophysix.simdm package follows the *Simulation Datamodel* (SimDM documentation) standard documented by the International Virtual Observatory Alliance.

- the astrophysix.units package provides a more generic-purpose physical quantity and units management tool. The most common physical constants used in astrophysics are defined in this module.

With this package, users can create a *SimulationStudy* in which a single numerical project can be fully documented and all the reduced datasets (PNG plots, FITS files, tarballs, etc.) can be attached. It follows the hierarchical structure :


Simulation study
Project
Simulations
Simulation codes
RAMSES
AREPO
[RAMSES] Simulation #1
Generic results
Generic Result #1
Datafiles
Star formation history plot
Result #2
Snapshots
Snapshot #234
Datafiles
Gas density 2D map
'rho_dens.png' PNG file
'rho_slice.fits' FITS file
Star formation radial 1D profile
Snapshot #587
Post-processing runs
Post-processing codes
RadMC
[RadMC] Post-pro. run #1 (Simulation #2)
Generic results
Synthetic observed map
Datafiles
[AREPO] Simulation #2

CHAPTER

TWO

GALACTICA DATABASE INTEGRATION

These studies can be saved in persistent and portable HDF5 files and then distributed to other members of the scientific collaboration to be updated. *SimulationStudy* HDF5 files can be uploaded with a single-click on the Galactica simulation database to automatically deploy web pages for your astrophysical simulation project.

SimulationStudy HDF5 files can be uploaded on the Galactica simulation database several times in order to :

- Create new project web pages on the web application (creation),
- Update pages for an existing project (update).

> **Warning:** When you upload a *SimulationStudy* HDF5 file, the Galactica server will **NEVER** take the responsibility of deleting any entry from the database related to an item that is missing in your *SimulationStudy* HDF5 file. In other words, deleting an object from your local *SimulationStudy*, saving the study into a HDF5 file and uploading the file on Galactica won't delete the object from the database (only the *creation* and *update* behaviours are enabled).
>
> If you wish to remove items from the web application, you **MUST** do so by hand (it must be an explicit user action) in the Galactica administration interface.

2.1 Installation

astrophysix can be installed in you local Python environment (virtualenv, conda, . . .) with **pip**.

2.1.1 Latest stable releases

Using **pip**, you can easily download and install the latest version of the astrophysix package directly from the Python Package Index (PyPI):

```
> pip install astrophysix
Collecting astrophysix
    Downloading astrophysix-0.4.1-py2.py3-none-any.whl (101 kB)
Collecting h5py>=2.10.0
    Using cached h5py-2.10.0-cp36-cp36m-manylinux1_x86_64.whl (2.9 MB)
Collecting Pillow>=6.2.1
    Using cached Pillow-7.2.0-cp36-cp36m-manylinux1_x86_64.whl (2.2 MB)
Collecting numpy>=1.16.4
    Using cached numpy-1.19.2-cp36-cp36m-manylinux2010_x86_64.whl (14.5 MB)
Collecting future>=0.17.1
    Using cached future-0.18.2.tar.gz (829 kB)
Collecting six
```

(continues on next page)

(continued from previous page)

```
    Using cached six-1.15.0-py2.py3-none-any.whl (10 kB)
Installing collected packages: numpy, six, h5py, Pillow, future, astrophysix
    Running setup.py install for future ... done
Successfully installed Pillow-7.2.0 astrophysix-0.4.1 future-0.18.2 h5py-2.10.0 numpy-
↪1.19.2 six-1.15.0
```

2.1.2 Unstable releases

If you wish to use a specific beta release of `astrophysix`, you can select which version to install :

```
> pip install astrophysix==0.4.0rc1
Collecting astrophysix
    Downloading astrophysix-0.4.0rc1-py2.py3-none-any.whl (101 kB)
Collecting h5py>=2.10.0
    Using cached h5py-2.10.0-cp36-cp36m-manylinux1_x86_64.whl (2.9 MB)
Collecting Pillow>=6.2.1
    Using cached Pillow-7.2.0-cp36-cp36m-manylinux1_x86_64.whl (2.2 MB)
Collecting numpy>=1.16.4
    Using cached numpy-1.19.2-cp36-cp36m-manylinux2010_x86_64.whl (14.5 MB)
Collecting future>=0.17.1
    Using cached future-0.18.2.tar.gz (829 kB)
Collecting six
    Using cached six-1.15.0-py2.py3-none-any.whl (10 kB)
Installing collected packages: numpy, six, h5py, Pillow, future, astrophysix
    Running setup.py install for future ... done
Successfully installed Pillow-7.2.0 astrophysix-0.4.0rc1 future-0.18.2 h5py-2.10.0␣
↪numpy-1.19.2 six-1.15.0
```

2.2 Quick-start guide

This quick-start guide will walk you into the main steps to document your numerical project. If you want to skip the tour and directly see an example, see *Full example script*.

2.2.1 Project and study

Creation and persistency

To create a `Project` and save it into a `SimulationStudy` HDF5 file, you may run this minimal Python script :

```
>>> from astrophysix.simdm import SimulationStudy, Project, ProjectCategory
>>> proj = Project(category=ProjectCategory.Cosmology, project_title="Extreme␣
↪Horizons cosmology project")
>>> study = SimulationStudy(project=proj)
>>> study.save_HDF5("./EH_study.h5")
```

Loading a study

To read the *SimulationStudy* from your HDF5 file, update it and save the updated study back in its original file :

```
>>> from astrophysix.simdm import SimulationStudy
>>> study = SimualationStudy.load_HDF5("./EH_study.h5")
>>> proj = study.project
>>> proj.short_description = "This is a short description (one-liner) of my project."
>>> # Saves your updated study back in the same file './EH_study.h5'
>>> study.save_HDF5()
```

Study information

A *SimulationStudy* object contains details on when it has been created (*creation_time* property) and last modified (*last_modification_time* property) :

```
>>> study.creation_time
datetime.datetime(2020, 9, 4, 14, 05, 21, 84601, tzinfo=datetime.timezone.utc)
>>> study.last_modification_time
datetime.datetime(2020, 9, 18, 15, 12, 27, 14512, tzinfo=datetime.timezone.utc)
```

Full project initialization example

To initialize a *Project*, you only need to set *category* and *project_title* properties. Here is a more complete example of a *Project* initialization with all optional attributes :

```
>>> proj = Project(category=ProjectCategory.StarFormation, alias="FRIG",
...                project_title="Self-regulated magnetised ISM modelisation",
...                short_description="Short description of my 'FRIG' project",
...                general_description="""This is a pretty long description for my␣
↪project spanning over multiple lines
...                if necessary""",
...                data_description="The data available in this project...",
...                directory_path="/path/to/project/data/")
```

> **Warning:** Setting the *alias* property is necessary only if you wish to upload your study on the Galactica simulation database. See *Why an alias ?* and *How can I check validity for Galactica ?*

See also:

- *SimulationStudy*, *Project* and *ProjectCategory* API references.

2.2.2 Simulation codes and runs

To add a simulation run into your project, the definition of a *SimulationCode* is mandatory. Once defined, you can create a *Simulation* based on that simulation code:

```
>>> from astrophysix.simdm.protocol import SimulationCode
>>> from astrophysix.simdm.experiment import Simulation
>>> ramses = SimulationCode(name="Ramses 3.1 (MHD)", code_name="RAMSES")
>>> simu = Simulation(simu_code=ramses, name="Hydro run full resolution")
```

To add the simulation to the project, use the *Project.simulations* property (*ObjectList*):

```
>>> proj.simulations.add(simu)
>>> proj.simulations
Simulation list :
+---+---------------------------+---------------------------------------+
| # |           Index           |                 Item                  |
+---+---------------------------+---------------------------------------+
| 0 | Hydro run full resolution | 'Hydro run full resolution' simulation |
+---+---------------------------+---------------------------------------+
| 1 | Hydro run full resolution | 'MHD run full resolution' simulation   |
+---+---------------------------+---------------------------------------+
```

See also:

- *Simulation*, *SimulationCode* API references,
- *Protocols* detailed section.
- *Experiments* detailed section.
- *ObjectList* API reference.

2.2.3 Post-processing runs

Optionally, you can add *PostProcessingRun* into a *Simulation* using the *Simulation.post_processing_runs* property (*ObjectList*). To create a *PostProcessingRun*, you must first define a *PostProcessingCode* :

```
from astrophysix.simdm.protocol import PostProcessingCode
from astrophysix.simdm.experiment import PostProcessingCodeRun

radmc = PostProcessingCode(name="RADMC-3D", code_name="RADMC-3D", code_version="2.0")
pprun = PostProcessingCodeRun(ppcode=radmc, name="Synthetic observable creation of␣
↪pre-stellar cores")

# Add post-pro run into the simulation
simu.post_processing_runs.add(pprun)
```

See also:

- *PostProcessingRun*, *PostProcessingCode* API references,
- *Protocols* detailed section.
- *Experiments* detailed section.
- *ObjectList* API reference.

2.2.4 Results and snapshots

Experiment-wide

You can add results into any simulation or post-processing run. If it is an experiment-wide result, create a *GenericResult* and use the *Simulation.generic_results* or *PostProcessingRun.generic_results* property (*ObjectList*) to add it into your run :

```
from astrophysix.simdm.results import GenericResult

res1 = GenericResult(name="Star formation history", directory_path="/my/path/to/result
↪",
                     description="""This is the star formation history during the 2
                                    Myr of the galaxy major merger""")
simu.generic_results.add(res1)
```

Time-specific

Otherwise, if it is time-specific result, create a *Snapshot* and use the *Simulation.snapshots* or *PostProcessingRun.snapshots* property (*ObjectList*) to add it into your run :

```
from astrophysix.simdm.results import Snapshot
from astrophysix import units as U

sn34 = Snapshot(name="Third pericenter", time=(254.7, U.Myr),
                directory_path="/my/path/to/simu/outputs/output_00034")
simu.snapshots.add(sn34)
```

See also:

- *GenericResult*, *Snapshot* API references,
- *Experiments* detailed section.
- *Results and associated datafiles* detailed section.
- *ObjectList* API reference.

2.2.5 Datafiles

You can add `Datafile` objects into any *Snapshot* or *GenericResult* :

```
>>> from astrophysix.simdm.datafiles import Datafile
>>>
>>> # Add a datafile into a snapshot
>>> imf_df = Datafile(name="Initial mass function",
...                   description="This is my IMF plot detailed description...")
>>> sn34.datafiles.add(imf_df)
```

And then embed files from your local filesystem into the `Datafile` (the file raw byte array will be imported along with the original file *name* and *last modification date*):

```
>>> from astrophysix.simdm.datafiles import image
>>> from astrophysix.utils.file import FileType
>>>
>>> # Attach files to datafile
```

(continues on next page)

(continued from previous page)

```
>>> imf_df[FileType.PNG_FILE] = os.path.join("/data", "io", "datafiles", "plot_image_
↪IMF.png")
>>> jpg_fpath = os.path.join("/data", "io", "datafiles", "plot_with_legend.jpg")
>>> imf_df[FileType.JPEG_FILE] = image.JpegImageFile.load_file(jpg_fpath)
>>> imf_df[FileType.HDF5_FILE] = os.path.join("/data", "io", "HDF5", "stars.h5")
```

Anyone can reopen your `SimulationStudy` HDF5 file, read the attached files and re-export them on their local filesystem to retrieve a carbon copy of your original file:

```
>>> imf_df[FileType.JPEG_FILE].save_to_disk("/home/user/Desktop/export_plot.jpg")
File '/home/user/Desktop/export_plot.jpg' saved
```

See also:

- `Datafile` API references,
- *Results and associated datafiles* detailed section.

2.2.6 Full example script

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# This file is part of the 'astrophysix' Python package.
#
# Copyright © Commissariat a l'Energie Atomique et aux Energies Alternatives (CEA)
#
#  FREE SOFTWARE LICENCING
#  -----------------------
# This software is governed by the CeCILL license under French law and abiding by the␣
↪rules of distribution of free
# software. You can use, modify and/or redistribute the software under the terms of␣
↪the CeCILL license as circulated by
# CEA, CNRS and INRIA at the following URL: "http://www.cecill.info". As a␣
↪counterpart to the access to the source code
# and rights to copy, modify and redistribute granted by the license, users are␣
↪provided only with a limited warranty
# and the software's author, the holder of the economic rights, and the successive␣
↪licensors have only limited
# liability. In this respect, the user's attention is drawn to the risks associated␣
↪with loading, using, modifying
# and/or developing or reproducing the software by the user in light of its specific␣
↪status of free software, that may
# mean that it is complicated to manipulate, and that also therefore means that it is␣
↪reserved for developers and
# experienced professionals having in-depth computer knowledge. Users are therefore␣
↪encouraged to load and test the
# software's suitability as regards their requirements in conditions enabling the␣
↪security of their systems and/or data
# to be ensured and, more generally, to use and operate it in the same conditions as␣
↪regards security. The fact that
# you are presently reading this means that you have had knowledge of the CeCILL␣
↪license and that you accept its terms.
#
#
# COMMERCIAL SOFTWARE LICENCING
```

(continues on next page)

(continued from previous page)

```
# ------------------------------
# You can obtain this software from CEA under other licencing terms for commercial␣
↪purposes. For this you will need to
# negotiate a specific contract with a legal representative of CEA.
#
from __future__ import print_function, unicode_literals
import os
import numpy as N

from astrophysix.simdm import SimulationStudy, Project, ProjectCategory
from astrophysix.simdm.experiment import Simulation, AppliedAlgorithm,␣
↪ParameterSetting, ParameterVisibility,\
    ResolvedPhysicalProcess
from astrophysix.simdm.protocol import SimulationCode, AlgoType, Algorithm,␣
↪InputParameter, PhysicalProcess, Physics
from astrophysix.simdm.results import GenericResult, Snapshot
from astrophysix.simdm.datafiles import Datafile, PlotType, PlotInfo
from astrophysix.utils.file import FileType
from astrophysix import units as U

# ----------------------------------------------- Project creation ------------------
↪-------------------------------- #
# Available project categories are :
# - ProjectCategory.SolarMHD
# - ProjectCategory.PlanetaryAtmospheres
# - ProjectCategory.StarPlanetInteractions
# - ProjectCategory.StarFormation
# - ProjectCategory.Supernovae
# - ProjectCategory.GalaxyFormation
# - ProjectCategory.GalaxyMergers
# - ProjectCategory.Cosmology
proj = Project(category=ProjectCategory.StarFormation, project_title="Frig",
               alias="FRIG", short_description="Short description of my 'FRIG' project
↪",
               general_description="""This is a pretty long description for my project
↪""",
               data_description="The data available in this project...", directory_
↪path="/path/to/project/data/")
print(proj)  # "[Star formation] 'Frig' project"
# ----------------------------------------------------------------------------------
↪-------------------------------- #


# --------------------------------------- Simulation code definition -----------------
↪-------------------------------- #
ramses = SimulationCode(name="Ramses 3 (MHD)", code_name="Ramses", code_version="3.10.
↪1", alias="RAMSES_3",
                        url="https://www.ics.uzh.ch/~teyssier/ramses/RAMSES.html",
                        description="This is a fair description of the Ramses code")
# => Add algorithms : available algorithm types are :
# - AlgoType.AdaptiveMeshRefinement
# - AlgoType.VoronoiMovingMesh
# - AlgoType.SmoothParticleHydrodynamics
# - AlgoType.Godunov
# - AlgoType.PoissonMultigrid
# - AlgoType.PoissonConjugateGradient
# - AlgoType.ParticleMesh
```

(continues on next page)

(continued from previous page)

```
# - AlgoType.FriendOfFriend
# - AlgoType.HLLCRiemann
# - AlgoType.RayTracer
amr = ramses.algorithms.add(Algorithm(algo_type=AlgoType.AdaptiveMeshRefinement,␣
↪description="AMR descr"))
ramses.algorithms.add(Algorithm(algo_type=AlgoType.Godunov, description="Godunov␣
↪scheme"))
ramses.algorithms.add(Algorithm(algo_type=AlgoType.HLLCRiemann, description="HLLC␣
↪Riemann solver"))
ramses.algorithms.add(Algorithm(algo_type=AlgoType.PoissonMultigrid, description=
↪"Multigrid Poisson solver"))
ramses.algorithms.add(Algorithm(algo_type=AlgoType.ParticleMesh, description="PM␣
↪solver"))

# => Add input parameters
ramses.input_parameters.add(InputParameter(key="levelmin", name="Lmin",
                                           description="min. level of AMR refinement
↪"))
lmax = ramses.input_parameters.add(InputParameter(key="levelmax", name="Lmax",
                                                  description="max. level of AMR␣
↪refinement"))

# => Add physical processes : available physics are :
# - Physics.SelfGravity
# - Physics.Hydrodynamics
# - Physics.MHD
# - Physics.StarFormation
# - Physics.SupernovaeFeedback
# - Physics.AGNFeedback
# - Physics.MolecularCooling
ramses.physical_processes.add(PhysicalProcess(physics=Physics.StarFormation,␣
↪description="descr sf"))
ramses.physical_processes.add(PhysicalProcess(physics=Physics.Hydrodynamics,␣
↪description="descr hydro"))
grav = ramses.physical_processes.add(PhysicalProcess(physics=Physics.SelfGravity,␣
↪description="descr self G"))
ramses.physical_processes.add(PhysicalProcess(physics=Physics.SupernovaeFeedback,␣
↪description="SN feedback"))
# -------------------------------------------------------------------------------------
↪-------------------------------- #


# -------------------------------------------- Simulation setup ----------------------
↪-------------------------------- #
simu = Simulation(simu_code=ramses, name="My most important simulation", alias="SIMU_1
↪", description="Simu description",
                  execution_time="2020-03-01 18:45:30", directory_path="/path/to/my/
↪project/simulation/data/")
proj.simulations.add(simu)

# Add applied algorithms implementation details. Warning : corresponding algorithms␣
↪must have been added in the 'ramses'
# simulation code.
simu.applied_algorithms.add(AppliedAlgorithm(algorithm=amr, details="My AMR␣
↪implementation [Teyssier 2002]"))
simu.applied_algorithms.add(AppliedAlgorithm(algorithm=ramses.algorithms[AlgoType.
↪HLLCRiemann.name],
```

(continues on next page)

(continued from previous page)

```
                                                  details="My Riemann solver␣
↪implementation [Teyssier 2002]"))

# Add parameter setting. Warning : corresponding input parameter must have been added␣
↪in the 'ramses' simulation code.
# Available parameter visibility options are :
# - ParameterVisibility.NOT_DISPLAYED
# - ParameterVisibility.ADVANCED_DISPLAY
# - ParameterVisibility.BASIC_DISPLAY
simu.parameter_settings.add(ParameterSetting(input_param=ramses.input_parameters["Lmin
↪"], value=8,
                                             visibility=ParameterVisibility.BASIC_
↪DISPLAY))
simu.parameter_settings.add(ParameterSetting(input_param=lmax, value=12,
                                             visibility=ParameterVisibility.BASIC_
↪DISPLAY))

# Add resolved physical process implementation details. Warning : corresponding␣
↪physical process must have been added to
# the 'ramses' simulation code
simu.resolved_physics.add(ResolvedPhysicalProcess(physics=ramses.physical_
↪processes[Physics.StarFormation.name],
                                                  details="Star formation specific␣
↪implementation"))
simu.resolved_physics.add(ResolvedPhysicalProcess(physics=grav, details="self-gravity␣
↪specific implementation"))
# -------------------------------------------------------------------------------------
↪-------------------------------- #


# -------------------------------------- Simulation generic result and snapshots -----
↪-------------------------------- #
# Generic result
gres = GenericResult(name="Key result 1 !", description="My description", directory_
↪path="/my/path/to/result")
simu.generic_results.add(gres)

# Simulation snapshot
# In one-line
sn = simu.snapshots.add(Snapshot(name="My best snapshot !", description="My first␣
↪snapshot description",
                                 time=(125, U.kyr), physical_size=(250.0, U.kpc),␣
↪directory_path="/path/to/snapshot1",
                                 data_reference="OUTPUT_00056"))
# Or create snapshot, then add it to the simulation
sn2 = Snapshot(name="My second best snapshot !", description="My second snapshot␣
↪description", time=(0.26, U.Myr),
               physical_size=(0.25, U.Mpc), directory_path="/path/to/snapshot2", data_
↪reference="OUTPUT_00158")
simu.snapshots.add(sn2)
# -------------------------------------------------------------------------------------
↪-------------------------------- #


# ---------------------------------------------------- Result datafiles --------------
↪-------------------------------- #
# Datafile creation
```

(continues on next page)

(continued from previous page)

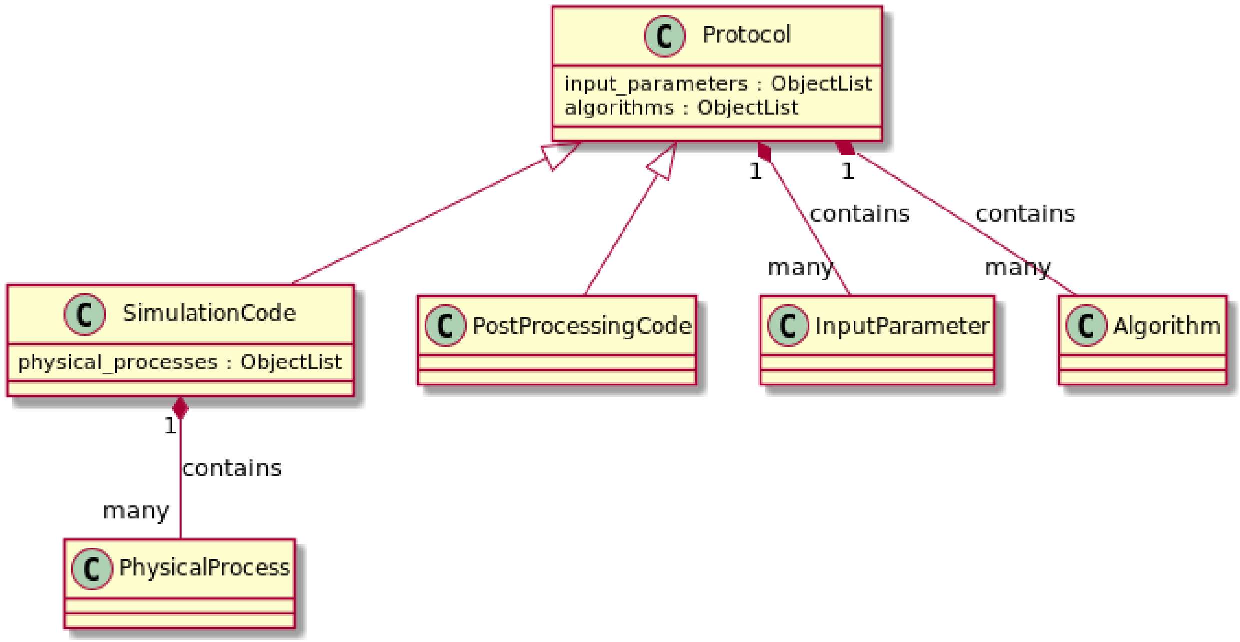
```
imf_df = sn.datafiles.add(Datafile(name="Initial mass function plot",
                                   description="This is my plot detailed description
↪"))

# Add attached files to a datafile (1 per file type). Available file types are :
# - FileType.HDF5_FILE
# - FileType.PNG_FILE
# - FileType.JPEG_FILE
# - FileType.FITS_FILE
# - FileType.TARGZ_FILE
# - FileType.PICKLE_FILE
# - FileType.JSON_FILE
# - FileType.CSV_FILE
# - FileType.ASCII_FILE
imf_df[FileType.PNG_FILE] = os.path.join("/data", "io", "datafiles", "plot_image_IMF.
↪png")
imf_df[FileType.JPEG_FILE] = os.path.join("/data", "io", "datafiles", "plot_with_
↪legend.jpg")
imf_df[FileType.FITS_FILE] = os.path.join("/data", "io", "datafiles", "cassiopea_A_0.
↪5-1.5keV.fits")
imf_df[FileType.TARGZ_FILE] = os.path.join("/data", "io", "datafiles", "archive.tar.gz
↪")
imf_df[FileType.JSON_FILE] = os.path.join("/data", "io", "datafiles", "test_header_
↪249.json")
imf_df[FileType.ASCII_FILE] = os.path.join("/data", "io", "datafiles", "abstract.txt")
imf_df[FileType.HDF5_FILE] = os.path.join("/data", "io", "HDF5", "study.h5")
imf_df[FileType.PICKLE_FILE] = os.path.join("/data", "io", "datafiles", "dict_saved.
↪pkl")

# Datafile plot information (for plot future updates and online interactive␣
↪visualisation on Galactica web pages).
# Available plot types are :
# - LINE_PLOT
# - SCATTER_PLOT
# - HISTOGRAM
# - HISTOGRAM_2D
# - IMAGE
# - MAP_2D
imf_df.plot_info = PlotInfo(plot_type=PlotType.LINE_PLOT, xaxis_values=N.array([10.0,␣
↪20.0, 30.0, 40.0, 50.0]),
                            yaxis_values=N.array([1.256, 2.456, 3.921, 4.327, 5.159]),
↪ xaxis_log_scale=False,
                            yaxis_log_scale=False, xaxis_label="Mass", yaxis_label=
↪"Probability", xaxis_unit=U.Msun,
                            plot_title="Initial mass function", yaxis_unit=U.Mpc)
# ------------------------------------------------------------------------------------
↪-------------------------------- #


# Save study in HDF5 file
study = SimulationStudy(project=proj)
study.save_HDF5("./frig_study.h5")

# Eventually reload it from HDF5 file to edit its content
# study = SimulationStudy.load_HDF5("./frig_study.h5")
```

2.3 Protocols

Numerical codes used to produce simulation data or post-process them are called *Protocols* in the `Simulation Datamodel` vocabulary. They can be of two types :

- *SimulationCode* to run simulations,
- *PostProcessingCode* to post-process simulation data.

Any *Protocol* contains a set of *Algorithm* and a set of *InputParameter*. In addition, a *SimulationCode* contains a set of *PhysicalProcess* :



2.3.1 Simu/Post-pro codes

To initialize a *SimulationCode*, you only need to set *name* and *code_name* properties. Here is a more complete example of a *SimulationCode* initialization with all optional attributes :

```
>>> from astrophysix.simdm.protocol import SimulationCode
>>> ramses = SimulationCode(name="Ramses 3 (MHD)", code_name="RAMSES", code_version=
↪"3.10.1",
...                          alias="RAMSES_3", url="https://www.ics.uzh.ch/~teyssier/
↪ramses/RAMSES.html",
...                          description="This is a fair description of the Ramses code
↪")
```

The same applies to the initialization of a *PostProcessingCode*.

Warning: Setting the `SimulationCode.alias` and `PostProcessingCode.alias` properties is necessary only if you wish to upload your study on the Galactica simulation database. See *Why an alias ?* and *How can I check validity for Galactica ?*

See also:

- *SimulationCode* API reference,
- *PostProcessingCode* API reference,
- *Experiments* section.

2.3.2 Input parameters

To add *InputParameter* objects into any *Protocol*, use :

- *SimulationCode.input_parameters* for a *SimulationCode*,
- *PostProcessingCode.input_parameters* for a *PostProcessingCode*.

```
>>> from astrophysix.simdm.protocol import InputParameter
>>> # One-liner
>>> lmin = ramses.input_parameters.add(InputParameter(key="levelmin", name="Lmin",
...                                                    description="min. level of AMR␣
↪refinement"))
# Or
>>> lmax = InputParameter(key="levelmax", name="Lmax", description="max. level of AMR␣
↪refinement")
>>> ramses.input_parameters.add(lmax)
```

To initialize an *InputParameter*, only the *InputParameter.name* property must be set :

```
>>> # Input parameters should be initialised with at least a 'name' attribute.
>>> rho_min = InputParameter()
AttributeError : Input parameter 'name' attribute is not defined (mandatory).
```

See also:

InputParameter API reference.

2.3.3 Algorithms

To add *Algorithm* objects into any *Protocol*, use :

- *SimulationCode.algorithms* for a *SimulationCode*,
- *PostProcessingCode.algorithms* for a *PostProcessingCode*.

```
>>> from astrophysix.simdm.protocol import Algorithm, AlgoType
>>> # One-liner
>>> voronoi = arepo.algorithms.add(Algorithm(algo_type=AlgoType.VoronoiMovingMesh,
...                                          description="Moving mesh based on a␣
↪Voronoi-Delaunay tesselation."))
# Or
>>> voronoi = Algorithm(algo_type=AlgoType.VoronoiMovingMesh,
...                     description="Moving mesh based on a Voronoi-Delaunay␣
↪tesselation.")
>>> arepo.algorithms.add(voronoi)
```

To initialize an *Algorithm*, only the *Algorithm.algo_type* property must be set :

```
>>> # Algorithm should be initialised with at least an 'algo_type' attribute.
>>> algo = Algorithm()
AttributeError : Algorithm 'algo_type' attribute is not defined (mandatory).
```

Available algorithm types are :

- *AlgoType.AdaptiveMeshRefinement*
- *AlgoType.SmoothParticleHydrodynamics*
- *AlgoType.VoronoiMovingMesh*
- *AlgoType.Godunov*
- *AlgoType.PoissonMultigrid*
- *AlgoType.PoissonConjugateGradient*
- *AlgoType.ParticleMesh*
- *AlgoType.FriendOfFriend*
- *AlgoType.HLLCRiemann*
- *AlgoType.RayTracer*

See also:

Algorithm and *AlgoType* API references.

2.3.4 Physical processes

Note: For *SimulationCode* only.

To add *PhysicalProcess* objects into a *SimulationCode*, use the *SimulationCode.physical_processes* property.

```
>>> from astrophysix.simdm.protocol import PhysicalProcess, Physics
>>> # One-liner
>>> sf = ramses.physical_processes.add(PhysicalProcess(physics=Physics.StarFormation,
...                                                    description="Conversion of gas␣
↪into massive star particles."))
# Or
>>> sf = PhysicalProcess(physics=Physics.StarFormation, description="Conversion of␣
↪gas into massive star particles.")
>>> ramses.physical_processes.add(sf)
```

To initialize a *PhysicalProcess*, only the *PhysicalProcess.physics* property must be set :

```
>>> # PhysicalProcess should be initialised with at least a 'physics' attribute.
>>> process = PhysicalProcess()
AttributeError : PhysicalProcess 'physics' attribute is not defined (mandatory).
```

Available physics are :

- *Physics.SelfGravity*
- *Physics.Hydrodynamics*
- *Physics.MHD*
- *Physics.StarFormation*
- *Physics.SupernovaeFeedback*
- *Physics.AGNFeedback*

- *Physics.MolecularCooling*

See also:

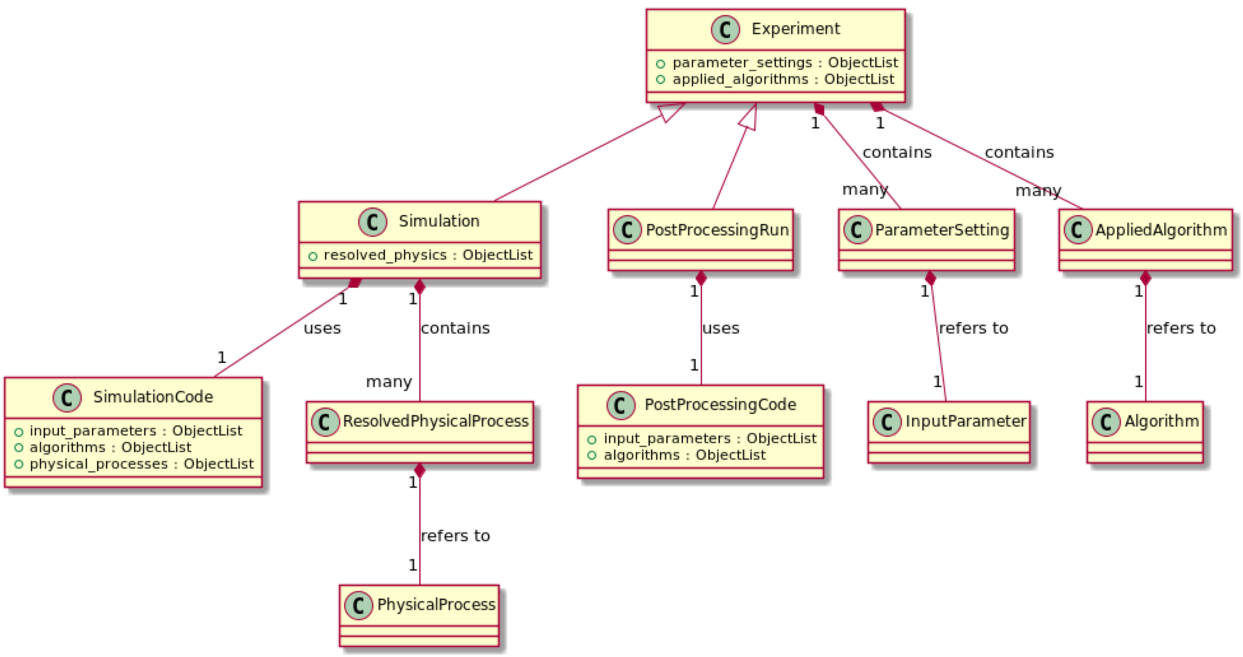
PhysicalProcess and *Physics* API references.

2.4 Experiments

In the `Simulation Datamodel` vocabulary, a numerical *Experiment* produces or post-processes scientific data. It can be of two types :

- *Simulation* (using `SimulationCode` as *Protocols*),
- *PostProcessingRun* (using `PostProcessingCode` as *Protocols*).

Any *Experiment* contains a set of *AppliedAlgorithm* and a set of *ParameterSetting*. In addition, a *Simulation* contains a set of *ResolvedPhysicalProcess* :



A strict binding is enforced between :

- an *Experiment*'s *ParameterSetting* and its *Protocol*'s *InputParameter*,
- an *Experiment*'s *AppliedAlgorithm* and its *Protocol*'s *Algorithm*,
- a *Simulation*'s *ResolvedPhysicalProcess* and its `SimulationCode`'s *PhysicalProcess*.

For more details, see *Strict protocol/experiment bindings*.

2.4.1 Simulation

To define a `Simulation`, only two attributes are mandatory :

- **name** : a non-empty string value,
- **simu_code** : a `SimulationCode` instance, (used to initialize the `Simulation.simulation_code` property).

a `Simulation` has an `execution_time` property that can be set to any string-formatted datetime following the `%Y-%m-%d %H:%M:%S` format.

Here is a more complete example of a `Simulation` initialization with all optional attributes :

```
>>> from astrophysix.simdm.protocol import SimulationCode
>>> from astrophysix.simdm.experiment import Simulation
>>>
>>> enzo = SimulationCode(name="", code_name="ENZO", code_version="2.6", alias="ENZO_
↪26,
...                       url="https://enzo-project.org/",
...                       description="This is a fair description of the ENZO AMR code
↪")
>>> simu = Simulation(simu_code=enzo, name="Cosmological simulation",
...                   alias="SIMU_A", description="Simu description",
...                   execution_time="2020-09-15 16:25:12",
...                   directory_path="/path/to/my/project/simulation/data/")
```

Warning: Setting the `Simulation.alias` property is necessary only if you wish to upload your study on the Galactica simulation database. See *Why an alias ?* and *How can I check validity for Galactica ?*

2.4.2 Post-processing run

Once a `Simulation` has been defined, you can add a `PostProcessingRun` into it, to initialize one, only two attributes are mandatory :

- **name** : a non-empty string value,
- **ppcode** : a `PostProcessingCode` instance, (used to initialize the `PostProcessingRun.postpro_code` property).

Here is a more complete example of how to initialize a `PostProcessingRun` with all optional attributes and add it into a `Simulation` :

```
>>> from astrophysix.simdm.protocol import PostProcessingCode
>>> from astrophysix.simdm.experiment import PostProcessingRun
>>>
>>> hop = PostProcessingCode(name="Hop", code_name="HOP")
>>> pprun = PostProcessingRun(name="Overdense structure detection", ppcode=hop,
...                           alias="HOP_DETECTION"
...                           description="This is the description of my HOP post-
↪processing " \
...                                       "run to detect overdense gas structures in
↪" \
...                                       "the star-forming ISM.",
...                           directory_path="/path/to/my/hop_catalogs")
>>> simu.post_processing_runs.add(pprun)
```

Warning: Setting the *PostProcessingRun.alias* property is necessary only if you wish to upload your study on the Galactica simulation database. See *Why an alias ?* and *How can I check validity for Galactica ?*

2.4.3 Parameter settings

To define the value you used for an input parameter of your code in a given simulation (or post-processing) run, you can define a *ParameterSetting*. To do so, you must :

- make a reference to the associated code *InputParameter* : **input_param** attribute,
- give a value (float, int, string, bool) : **value** attribute,
- Optionally, you can set a **visibility** flag : *ParameterVisibility* (default to *BASIC_DISPLAY*), only used by the Galactica web app., for display purposes.

Available parameter setting *visibility* options are :

- *ParameterVisibility.NOT_DISPLAYED*,
- *ParameterVisibility.ADVANCED_DISPLAY*,
- *ParameterVisibility.BASIC_DISPLAY*.

Finally, use the *parameter_settings* property to add it into your run. Here is an example :

```
>>> from astrophysix.simdm.experiment import ParameterSetting
>>>
>>> set_levelmin = ParameterSetting(input_param=ramses.input_parameters['levelmin'],
...                                 value=8,
...                                 visibility=ParameterVisibility.ADVANCED_DISPLAY)
>>> simu.parameter_settings.add(set_levelmin)
>>> set_levelmax = simu.parameter_settings.add(ParameterSetting(input_param=lmax,
...                                                        value=12))
```

Warning: A *ParameterSetting* is strictly bound to its *Experiment*'s *Protocol*'s *InputParameter* instance, see *Strict protocol/experiment bindings* for details.

2.4.4 Applied algorithms

To define which algorithms were enabled in a given simulation (or post-processing) run and what were their implementation details, you can define a *AppliedAlgorithm*. To do so, you must :

- make a reference to the associated code *Algorithm* : **algorithm** attribute,
- optionally provide an implementation details (string) : **details** attribute.

Finally, use the *applied_algorithms* property to add it into your run. Here is an example :

```
>>> from astrophysix.simdm.experiment import AppliedAlgorithm
>>>
>>> app_amr = AppliedAlgorithm(algorithm=ramses.algorithms[AlgoType.
↪AdaptiveMeshRefinement.name],
...                           details="Fully threaded tree AMR implementation␣
↪[Teyssier 2002].")
>>> ramses_simu.applied_algorithms.add(app_amr)
```

Warning: A `AppliedAlgorithm` is strictly bound to its *Experiment*'s *Protocol*'s `Algorithm` instance, see *Strict protocol/experiment bindings* for details.

2.4.5 Resolved physical processes (for Simulation only)

To define which physical processes were resolved in a given simulation run and what were their implementation details, you can define a `ResolvedPhysicalProcess`. To do so, you must :

- make a reference to the associated `SimulationCode`'s `PhysicalProcess` : **physics** attribute,
- optionally provide an implementation details (`string`) : **details** attribute.

Finally, use the `resolved_physics` property to add it into your run. Here is an example :

```
>>> from astrophysix.simdm.experiment import ResolvedPhysicalProcess
>>>
>>> res_sf = ResolvedPhysicalProcess(physics=ramses.physical_processes[Physics.
↪StarFormation.name],
...                                  details="Star formation specific implementation
↪")
>>> simu.resolved_physics.add(res_sf)
>>> res_sgrav = ResolvedPhysicalProcess(physics=ramses.physical_processes[Physics.
↪SelfGravity.name],
                                        details="Self-gravity implementation (gas +␣
↪particles)"))
>>> simu.resolved_physics.add(res_sgrav)
```

Warning: A `ResolvedPhysicalProcess` is strictly bound to its `Simulation`'s `SimulationCode`'s `PhysicalProcess` instance, see *Strict protocol/experiment bindings* for details.

2.4.6 Strict protocol/experiment bindings

When you manipulate *Experiment* class objects (`Simulation` or `PostProcessingRun`) and *Protocol* class objects (`SimulationCode` or `PostProcessingCode`) and when you link objects together, `astrophysix` makes sure for you that your entire study hierarchical structure remains consistent at all times :

- upon object creation,
- upon object addition into another object,
- upon object deletion from another object.

Upon object creation

You are free to create any kind of `astrophysix` object, even those *linked* to another object :

```
>>> from astrophysix.simdm.protocol import InputParameter
>>> from astrophysix.simdm.experiment import ParameterSetting
>>>
>>> lmin = InputParameter(key="levelmin", name="Lmin", description="min. level of AMR␣
↪refinement"))
>>> set_levelmin = ParameterSetting(input_param=lmin, value=9))
```

There is no risk of breaking your study hierarchical structure consistency.

Upon object addition

To avoid *dangling* references into the study hierarchical structure, `astrophysix` will prevent you from :

- Adding a `ParameterSetting` object into the `parameter_settings` list of an *Experiment* if its associated `InputParameter` does not **already** belong to the *Experiment*'s *Protocol*'s `input_parameters` list,
- Adding a `AppliedAlgorithm` object into the `applied_algorithms` list of an *Experiment* if its associated `Algorithm` does not **already** belong to the *Experiment*'s *Protocol*'s `algorithms` list,
- Adding a `ResolvedPhysicalProcess` object into the `resolved_physics` list of a `Simulation` if its associated `PhysicalProcess` does not **already** belong to the `Simulation`'s `SimulationCode`'s `physical_processes` list.

I know it is a bit convoluted, let's see an example :

```
>>> from astrophysix.simdm.protocol import SimulationCode, InputParameter, Algorithm,␣
↪\
                                             PhysicalProcess, AlgoType, Physics
>>> from astrophysix.simdm.experiment import Simulation, ParameterSetting, \
                                               AppliedAlgorithm, ResolvedPhysicalProcess
>>>
>>> amr_code = SimulationCode(name="My AMR code", code_name="Proto_AMR")
>>> simu = Simulation(simu_code=amr_code, name="My test run")

>>> # ------------------- Input parameters ------------------------------
>>> lmin = InputParameter(key="levelmin", name="Lmin")
>>> set_levelmin = ParameterSetting(input_param=lmin, value=9)
>>> simu.parameter_settings.add(set_levelmin) # => Error
AttributeError: Simulation '[Lmin = 9] parameter setting' does not refer
to one of the input parameters of '[My AMR code] simulation code'.
>>>
>>> # Add first the input parameter into the code,
>>> amr_code.input_parameters.add(lmin)
>>> # THEN add the parameter setting into the simulation.
>>> simu.parameter_settings.add(set_levelmin) # => Ok
>>> # ------------------------------------------------------------------
>>>
>>> # ------------------- Applied algorithms ----------------------------
>>> amr_algo = Algorithm(algo_type=AlgoType.AdaptiveMeshRefinement)
>>> app_amr = AppliedAlgorithm(algorithm=amr_algo)
>>> simu.applied_algorithms.add(app_amr) # Error
AttributeError: Simulation '[Adaptive mesh refinement] applied algorithm'
does not refer to one of the algorithms of '[My AMR code] simulation code'.
>>>
>>> # Add first the algorithm into the code
>>> amr_code.algorithms.add(amr_algo)
>>> # THEN add the applied algorithm  into the simulation
>>> simu.applied_algorithms.add(app_amr)
>>> # ------------------------------------------------------------------
>>>
>>> # ---------------- Resolved physical processes ----------------------
>>> sf_process = PhysicalProcess(physics=Physics.StarFormation)
>>> res_sf = ResolvedPhysicalProcess(physics=sf_process)
>>> simu.resolved_physics.add(res_sf) # Error
```

(continues on next page)

(continued from previous page)

```
AttributeError: Simulation '[Star formation] resolved physical process'
does not refer to one of the physical processes of '[My AMR code] simulation code'.
>>>
>>> # Add first the physical process into the code
>>> amr_code.physical_processes.add(sf_process)
>>> # THEN add the resolved physical process into the simulation
>>> simu.resolved_physics.add(res_sf)
>>> # ------------------------------------------------------------------
```

Upon object deletion

To avoid missing references into the study hierarchical structure, `astrophysix` will also prevent you from :

- Deleting a `InputParameter` object from a *Protocol*'s `input_parameters` list if any *Experiment* associated to that *Protocol* contains a `ParameterSetting` that refers to the input parameter to be deleted,
- Deleting a `Algorithm` object from a *Protocol*'s `algorithms` list if any *Experiment*'s associated to that *Protocol* contains a `AppliedAlgorithm` that refers to the algorithm to be deleted,
- Deleting a `PhysicalProcess` object from a `SimulationCode`'s `physical_processes` list if any `Simulation` associated to that `SimulationCode` contains a `ResolvedPhysicalProcess` that refers to the physical process to be deleted.

I know it is a bit convoluted, let's see an example :

```
>>> from astrophysix.simdm.protocol import SimulationCode, InputParameter, Algorithm,␣
↪\
                                           PhysicalProcess, AlgoType, Physics
>>> from astrophysix.simdm.experiment import Simulation, ParameterSetting, \
                                             AppliedAlgorithm, ResolvedPhysicalProcess
>>> amr_code = SimulationCode(name="My AMR code", code_name="Proto_AMR")
>>> simu = Simulation(simu_code=amr_code, name="My test run")
>>>
>>> # ------------------- Input parameters ------------------------------
>>> lmin = InputParameter(key="levelmin", name="Lmin")
>>> amr_code.input_parameters.add(lmin)
>>> set_levelmin = ParameterSetting(input_param=lmin, value=9)
>>> simu.parameter_settings.add(set_levelmin)
>>> del amr_code.input_parameters[lmin]
AttributeError: '[levelmin] 'Lmin' input parameter' cannot be deleted, the following␣
↪items depend
on it (try to delete them first) : ['My test run' simulation [Lmin = 9] parameter␣
↪setting].
>>>
>>> # Delete the parameter setting from the simulation first,
>>> del simu.parameter_settings[set_levelmin]
>>> # THEN delete the input parameter from the code.
>>> del amr_code.input_parameters[lmin] # => Ok
>>> # ------------------------------------------------------------------
>>>
>>> # ------------------- Applied algorithms ----------------------------
>>> amr_algo = Algortihm(algo_type=AlgoType.AdaptiveMeshRefinement)
>>> amr_code.algorithms.add(amr_algo)
>>> app_amr = AppliedAlgorithm(algorithm=amr_algo)
>>> simu.applied_algorithms.add(app_amr)
>>> del amr_code.algorithms[amr_algo]
```

(continues on next page)

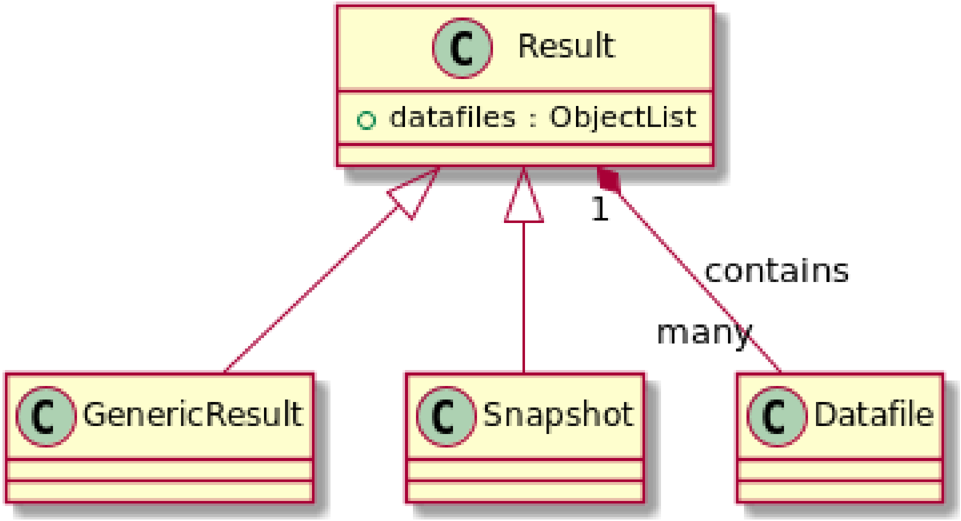
(continued from previous page)

```
AttributeError: ''Adaptive mesh refinement' algorithm' cannot be deleted, the␣
↪following items depend
on it (try to delete them first) : ['My test run' simulation [Adaptive mesh␣
↪refinement] applied algorithm].
>>>
>>> # Delete the applied algorithm from the simulation first,
>>> del simu.applied_algorithms[app_amr]
>>> # THEN delete the algorithms from the code
>>> del amr_code.algorithms[amr_algo] # => Ok
>>>
>>> # -------------------------------------------------------------------
>>>
>>> # ---------------- Resolved physical processes ----------------------
>>> sf_process = PhysicalProcess(physics=Physics.StarFormation)
>>> amr_code.physical_processes.add(sf_process)
>>> res_sf = ResolvedPhysicalProcess(physics=sf_process)
>>> simu.resolved_physics.add(res_sf)
>>> del amr_code.physical_processes[sf_process]
AttributeError: ''Star formation' physical process' cannot be deleted, the following␣
↪items depend
on it (try to delete them first) : ['My test run' simulation [Star formation]␣
↪resolved physical process].
>>>
>>> # Delete the resolved physical process from the simulation first
>>> del simu.resolved_physics[res_sf]
>>> # THEN delete the physical process from the code
>>> del amr_code.physical_processes[sf_process] # => Ok
>>> # -------------------------------------------------------------------
```

2.5 Results and associated datafiles

To any *Experiment* (`Simulation` or `PostProcessingRun`), you can attach results of your scientific analysis. There are two kinds of result available in `astrophysix`:

- `GenericResult` : result **not strictly related to a particular instant** in the dynamical evolution of your numerical experiment (e.g. star formation history, solar activity cycles, planetary orbital decay, etc.),
- `Snapshot` : result **corresponding to a specific moment** during the numerical experiment (galactic pericentric passage, solar activity peak, star formation burst, etc.).



2.5.1 Generic result

Here is a full example on how to create a `GenericResult` object with all optional parameters and add it into an *Experiment*, using the `Simulation.generic_results` or `PostProcessingRun.generic_results` property :

```
>>> from astrophysix.simdm.results import GenericResult
>>>
>>> res1 = GenericResult(name="Star formation history",
...                      directory_path="/my/path/to/result",
...                      description="""This is the star formation history during
...                                  the 2 Myr of the galaxy major merger""")
>>> simu.generic_results.add(res1)
```

2.5.2 Snapshot

A `Snapshot` derives from a `GenericResult` object but with additional optional properties :

- `time`,
- `physical_size`,
- `data_reference`.

Here is a full example on how to create a `Snapshot` object and add it into any *Experiment*, using `Simulation.snapshots` or `PostProcessingRun.snapshots` property :

```
>>> from astrophysix.simdm.results import Snapshot
>>> from astrophysix import units as U

>>> sn34 = Snapshot(name="Third pericenter",
...                 time=(254.7, U.Myr),
...                 physical_size=(400.0, U,kpc),
...                 decription="""This snapshot corresponds to the third pericentric
...                            of the galactic major merger simulation, occurring
...                            around $t\simeq255 \; \\textrm{Myr}$.""",
```

(continues on next page)

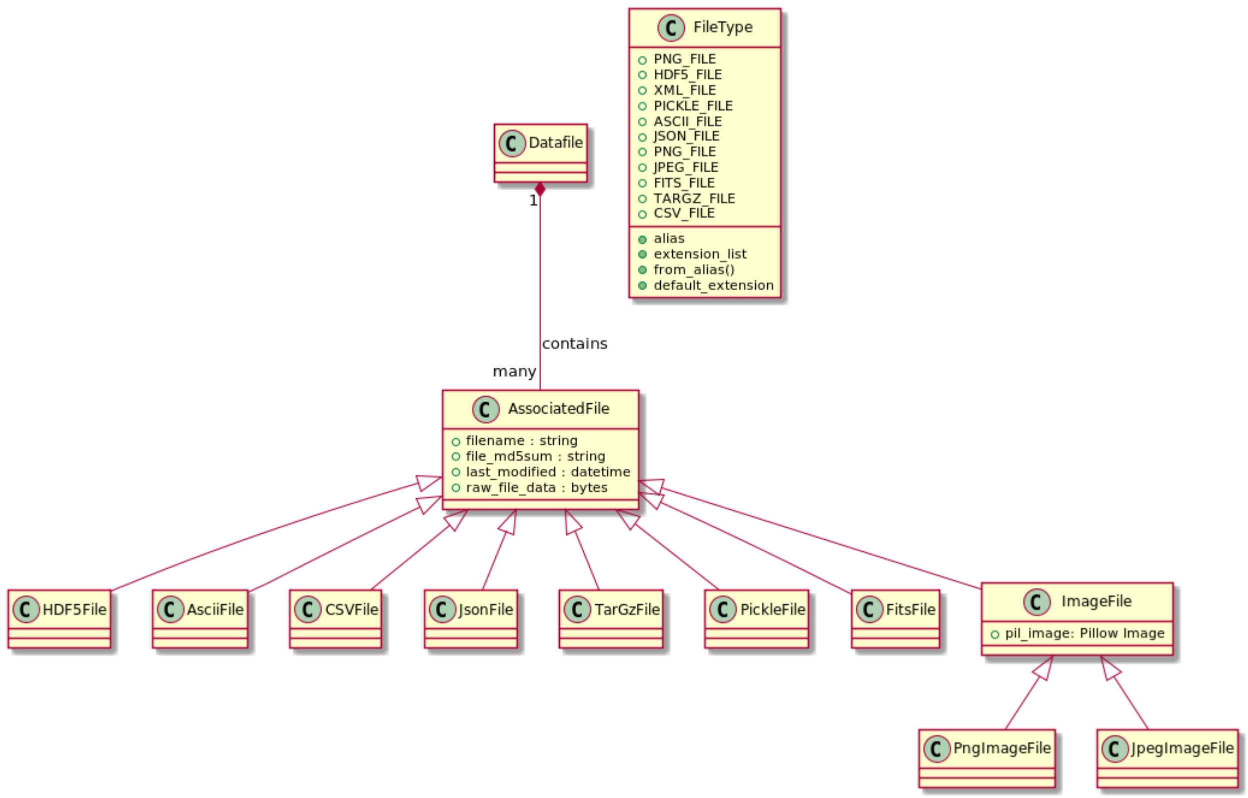
(continued from previous page)

```
...                    data_reference="34;Big_endian",
...                    directory_path="/my/path/to/galactic_merger/simu/outputs/output_
↪00034")
>>> simu.snapshots.add(sn34)
```

Note: The *Snapshot.data_reference* property is only used by the Galactica web application to provide a reference on your raw simulation data files to the on-demand post-processing services (see Terminus documentation.

2.5.3 Datafiles

One of the most important feature implemented in the `astrophysix` package is the possibility to insert documents into a *SimulationStudy* and to describe each one of them with meta-information.



To do so, you must create a *Datafile* (the *name* attribute is the only one mandatory) and then add it into your *Snapshot* (or *GenericResult*) using the *Snapshot.datafiles* (or *GenericResult.datafiles*) property :

```
>>> from astrophysix.simdm.datafiles import Datafile, PlotType, PlotInfo, image, file
>>> from astrophysix.utils.file import FileType, FileUtil
>>>
>>> imf_df = Datafile(name="Initial mass function",
...                   description="This is my IMF plot detailed description...")
>>> snapshot_100.datafiles.add(imf_df)
```

Attached files

Once created, a single *Datafile* can contain different files, but **at most one per** *FileType*. The available *FileType* values are :

- *FileType.HDF5_FILE*
- *FileType.PNG_FILE*
- *FileType.JPEG_FILE*
- *FileType.FITS_FILE*
- *FileType.TARGZ_FILE*
- *FileType.PICKLE_FILE*
- *FileType.JSON_FILE*
- *FileType.CSV_FILE*
- *FileType.ASCII_FILE*
- *FileType.XML_FILE*

To add files from your filesystem in a *Datafile*, you can do it in 2 steps (create first a *AssociatedFile* and then put it in the *Datafile*):

```
>>> import os
>>> from astrophysix.simdm.datafiles import image, file
>>>
>>> # JPEG image
>>> jpeg_filepath = os.path.join("/data", "path", "to", "my", "plots", "IMF_plot.jpg")
>>> jpeg_image_file = image.JpegImageFile.load_file(jpeg_filepath)
>>> imf_df[FileType.JPEG_FILE] = jpeg_image_file
>>>
>>> # HDF5 file
>>> hdf5_filepath = os.path.join("/data", "path", "to", "raw", "datasets", "all_stars.
↪h5")
>>> hdf5_file = file.HDF5File.load_file(hdf5_filepath)
>>> imf_df[FileType.HDF5_FILE] = hdf5_file
```

or if you are in a hurry, you can do it in a single one :

```
>>> import os
>>> from astrophysix.simdm.datafiles import image, file
>>>
>>> imf_df[FileType.JPEG_FILE] = os.path.join("/data", "path", "plots", "IMF_plot.jpg
↪")
>>> imf_df[FileType.HDF5_FILE] = os.path.join("/data", "path", "datasets", "all_stars.
↪h5")
```

To delete a file from a *Datafile* use the `del` Python operator:

```
>>> del imf_df[FileType.HDF_FILE]
```

The *AssociatedFile* contains your file raw byte array and has information on the original file (filename, last modification time). It can be used to re-export your file from your *SimulationStudy* to save it on your local filesystem (it even preserves the *last modification time* of the original file):

```
>>> jpeg_image_file = imf_df[FileType.JPEG_FILE]
>>> jpeg_image_file.last_modified
datetime.datetime(2020, 9, 22, 10, 42, 18, tzinfo=datetime.timezone.utc)
>>> saved_path = os.path.join("/home", "user", "Desktop", jpeg_image_file.filename)
>>> jpeg_image_file.save_to_disk(saved_path)
File '/home/user/Desktop/IMF_plot.jpg' saved
>>>
>>> import filecmp
>>> # Is the file saved identical to the original one ?
>>> filecmp.cmp(saved_path, jpeg_filepath, shallow=False)
True
>>>
>>> from astrophysix.utils.file import FileUtil
>>> from astrophysix.utils import DatetimeUtil
>>> # Was the file 'last modification time' preserved ?
>>> last_mod_tms = FileUtil.last_modification_timestamp(fpath)
>>> last_mod_dt = DatetimeUtil.utc_from_timestamp(last_mod_tms)
>>> last_mod_dt == jpeg_image_file.last_modified
True
```

Note: Since you can embed all your reduced data files into a *SimulationStudy*, you can safely remove your datafiles from your local filesystem and use the *SimulationStudy* HDF5 file as a self-contained, portable filesystem that you can exchange with your scientific collaborators.

Plot information

A *Datafile* can also have additional meta-information on a scientific plot for which you may already have attached PNG files (or JPEG, etc.). This meta-information can be used by other users to reproduce your plot or by the Galactica web application to display an interactive version of your plot online.

```
>>> from astrophysix.simdm.datafiles import PlotType, PlotInfo
>>> from astrophysix import units as U
>>>
>>> imf_df.plot_info = PlotInfo(plot_type=PlotType.LINE_PLOT, title="My plot title",
...                             xaxis_values=N.array([10.0, 20.0, , 22.0, 24.2, 30.
↪0]),
...                             yaxis_values=N.array([1.2, 35.2, 5.2, 21.2, 14.9]),
...                             xaxis_log_scale=False, yaxis_log_scale=True,
...                             xlabel="x-axis label", ylabel="y-axis label",
...                             xaxis_unit="Myr", yaxis_unit=U.kpc)
```

2.6 Physical units and constants module

The astrophysix package also provides a *Unit* helper class to handle physical constants and units. In addition, a large set of *physical quantities* and *constants* are defined in this module.

See also:

- The *Unit* API reference.

2.6.1 Basic use cases

Unit information

You can have access to unit parameters with the *name*, *description*, *coeff*, *dimensions*, *latex* and *physical_type* properties :

```
>>> from astrophysix import units as U
>>> mass_unit = U.Msun
>>> mass_unit.name
Msun
>>> mass_unit.dimensions
array([1, 0, 0, 0, 0, 0, 0, 0], dtype=int32)
>>> mass_unit.coeff
1.9889e+30
>>> mass_unit.description
'Solar mass'
>>> mass_unit.latex
'\\textrm{M}_{\\odot}'
>>> mass_unit.physical_type
'mass
```

A summary of any *Unit* can be displayed using the *Unit.info* method :

```
>>> from astrophysix import units as U
>>> U.ly.info()
Unit : ly
---------
Light year

Value
-----
9460730472580800.0 m

Equivalent units
----------------
 * m             :      1.057e-16 ly
 * um            :      1.057e-22 ly
 * mm            :      1.057e-19 ly
 * cm            :      1.057e-18 ly
 * nm            :      1.057e-25 ly
 * km            :      1.057e-13 ly
 * Angstrom      :      1.057e-26 ly
 * au            :    1.58125e-05 ly
 * pc            :        3.26156 ly
 * kpc           :        3261.56 ly
 * Mpc           :    3.26156e+06 ly
 * Gpc           :    3.26156e+09 ly
 * Rsun          :    7.35153e-08 ly
```

Unit retrieval

The *built-in units and constants* defined in the `astrophysix` package are directly accessible as variables of the package :

```
>>> from astrophysix import units as U
>>> U.Msun.description
'Solar mass'
>>> U.kHz.description
'kilo-Hertz : frequency unit'
```

For custom units retrieval, see *Custom Units*

Unit operations

You can create composite units by multiplying or dividing `Unit` objects by `float` values or other `Unit` objects. You can also raise `Unit` objects to a given (integer) power :

```
>>> from astrophysix import units as U
>>> u = U.km/U.s
>>> print(u)
(1000 m.s^-1)

>>> joule = kg*(m/s)**2
>>> joule == U.J
True

>>> my_p = 250.0 * J * m**-3
>>> my_p.physical_type
'pressure'
```

Custom Units

If you want to create your own units and use them elsewhere in your code, you can create a `Unit` instance that will be included in the `astrophysix` unit registry, use `Unit.create_unit` method to add a new unit, and `Unit.from_name` to retrieve it :

```
>>> from astrophysix import units as U
>>> U.km_s == U.km/U.s # Soft equality => same coefficient and same dimensions
True
>>> U.km_s.identical(U.km/U.s)  # Strict equality => they do not share the same names/
↪LaTEX formulae, descriptions
False
# Create a custom Solar mass per square kiloparsec surface density unit
>>> u = U.Unit.create_unit(name="Msun_pc2", base_unit=U.Msun/U.pc**2, descr="Solar␣
↪mass per square parsec",
                           latex="\\textrm{M}_{\\odot}\\cdot\\textrm{pc}^{-2}")

>>> u.identical(U.Msun/U.pc**2)
False

>>> # Later in your Python script ...
>>> surf_dens_unit = U.Unit.from_name("Msun_pc2")
surf_dens_unit == U.Msun/U.pc**2
True
```

(continues on next page)

(continued from previous page)

```
>>>surf_dens_unit.description
"Solar mass per square parsec"
```

Unit search

You can browse the units available in the astrophysix unit registry using the *Unit.equivalent_unit_list* method, the *Unit.appropriate_unit* method or the *Unit.iterate_units* iterator :

```
>>> # Equivalent units
>>> U.km.equivalent_unit_list()
[m : (1 m),
 um : (1e-06 m),
 mm : (0.001 m),
 cm : (0.01 m),
 nm : (1e-09 m),
 Angstrom : (1e-10 m),
 au : (1.49598e+11 m),
 pc : (3.08568e+16 m),
 kpc : (3.08568e+19 m),
 Mpc : (3.08568e+22 m),
 Gpc : (3.08568e+25 m),
 Rsun : (6.95508e+08 m),
 ly : (9.46073e+15 m)]

>>> # Most appropriate unit
>>> u = 0.3 * U.pc
>>> f, best_unit = u.appropriate_unit()
>>> print("{f:g} {bu:s}".format(f=f, bu=best_unit.name))
0.978469 ly

>>> # Unit iterator
>>> for u in Unit.iterate_units(phys_type="time"):
        print(u.name)
s
min
hour
day
sid_day
year
kyr
Myr
Gyr
```

Unit conversion

Unit conversion can be done with the *Unit.express* method :

```
>>> from astrophysix import units as U
>>> # Basic unit conversion
>>> l = 100.0 * U.kpc
>>> t = 320.0 U. Myr
>>> v = l/t
>>> print("v = {c:g} km/s".format(c=v.express(U.km_s)))
v = 305.56 km/s
```

2.6.2 Built-in quantities and constants

Base units

Name	Description
A	Ampere : electric intensity base unit
K	Kelvin : base temperature unit
cd	Candela: base luminous intensity unit
kg	Kilogram : base mass unit
m	Meter : base length unit
mol	mole: amount of a chemical substance base unit
none	Unscaled dimensionless unit
rad	radian: angular measurement (ratio lengh / radius of an arc)
s	Second : base time unit

See also:

Wikipedia : SI base unit

Constants and common units

Name	Value	Decomposition in base units	Description
Angstrom	1e-10	m	Angstrom: 10**-10 m
C	1	s.A	Coulomb
F	1	kg^-1.m^-2.s^4.A^2	Farad
G	6.67428e-11	kg^-1.m^3.s^-2	Graviational constant
GHz	1e+09	s^-1	giga-Hertz : frequency unit
Gauss	0.0001	kg.s^-2.A^-1	Gauss
Gpc	3.08568e+25	m	Gigaparsec
Gyr	3.15576e+16	s	Gigayear : trillion years
H	2.26855e-18	s^-1	Hubble's constant
H_cc	2.18421e-21	kg.m^-3	Atoms per cubic centimeter
Henry	1	kg.m^2.s^-2.A^-2	Henry
Hz	1	s^-1	Hertz : frequency unit
J	1	kg.m^2.s^-2	Joule : (SI) energy unit
Jy	1e-26	kg.s^-2	Jansky
Lsun	3.846e+26	kg.m^2.s^-3	Solar luminosity
MHz	1e+06	s^-1	mega-Hertz : frequency unit
Mearth	5.9722e+24	kg	Earth mass
Mpc	3.08568e+22	m	Megaparsec
Msun	1.9889e+30	kg	Solar mass
Myr	3.15576e+13	s	Megayear : million years
N	1	kg.m.s^-2	Newton : (SI) force unit
Ohm	1	kg.m^2.s^-3.A^-2	Ohm
Pa	1	kg.m^-1.s^-2	Pascal: (SI) pressure unit
Rsun	6.95508e+08	m	Solar radius
S	1	kg^-1.m^-2.s^3.A^2	Siemens
T	1	kg.s^-2.A^-1	Tesla
V	1	kg.m^2.s^-3.A^-1	Volt
W	1	kg.m^2.s^-3	Watt

continues on next page

Table 1 – continued from previous page

Name	Value	Decomposition in base units	Description
a_r	7.56577e-16	kg.m^-1.s^-2.K^-4	Radiation constant
arcmin	0.000290888	rad	arc minute: 1/60 of a hour angle
arcsec	4.84814e-06	rad	arc second: 1/60 of an arcminute
atm	101325	kg.m^-1.s^-2	atm: atmospheric pressure (101 3525 Pa)
au	1.49598e+11	m	Astronomical unit
bar	100000	kg.m^-1.s^-2	Bar
barn	1e-28	m^2	barn: surface unit used in HEP
barye	0.1	kg.m^-1.s^-2	Barye: (CGS) pressure unit
c	2.99792e+08	m.s^-1	Speed of light in vacuum
cm	0.01	m	Centimeter
cm3	1e-06	m^3	Cubic centimeter
day	86400	s	Day
deg	0.0174533	rad	degree: angular measurement corresponding to 1/360 of a full rotation
dyne	1e-05	kg.m.s^-2	dyne : (CGS) force unit
e	1.60218e-19	s.A	e : elementary electric charge carried by a proton
eV	1.60218e-19	kg.m^2.s^-2	electron-Volt
erg	1e-07	kg.m^2.s^-2	erg : (CGS) energy unit
g	0.001	kg	Gram
g_cc	1000	kg.m^-3	Gram per cubic centimeter
h	6.62607e-34	kg.m^2.s^-1	Planck Constant
hPa	100	kg.m^-1.s^-2	Hectopascal
hbar	1.05457e-34	kg.m^2.s^-1	Reduced Planck constant
hour	3600	s	Hour
hourangle	0.261799	rad	hour angle: angular measurement with 24 in a full circle
kB	1.38065e-23	kg.m^2.s^-2.K^-1	Boltzmann constant
kHz	1000	s^-1	kilo-Hertz : frequency unit
kPa	1000	kg.m^-1.s^-2	Kilopascal
km	1000	m	Kilometer
km_s	1000	m.s^-1	kilometers per second
kpc	3.08568e+19	m	Kiloparsec
kpc3	2.938e+49	m^3	Cubic kiloparsec
kyr	3.15576e+10	s	kyr : millenium
lm	1	rad^2.cd	Lumen
lx	1	m^-2.rad^2.cd	Lux
ly	9.46073e+15	m	Light year
m3	1	m^3	Cubic meter
mGauss	1e-07	kg.s^-2.A^-1	Milligauss
mH	1.66e-27	kg	Hydrogen atomic mass
m_s	1	m.s^-1	Meters per second
min	60	s	Minute
mm	0.001	m	Millimeter
nm	1e-09	m	Nanometer
pc	3.08568e+16	m	Parsec
pc3	2.938e+49	m^3	Cubic parsec
percent	0.01		One hundredth of unity
rhoc	9.2039e-27	kg.m^-3	Friedmann's universe critical density
sid_day	86164.1	s	Sidereal day : Earth full rotation time
sigmaSB	5.6704e-08	kg.s^-3.K^-4	Stefan-Boltzmann constant
sr	1	rad^2	Steradian: solid angle (SI) unit

continues on next page

Table 1 – continued from previous page

Name	Value	Decomposition in base units	Description
t	1000	kg	Metric ton
uGauss	1e-10	kg.s^-2.A^-1	Microgauss
um	1e-06	m	Micron
year	3.15576e+07	s	Year

Physical quantities

Quantity	Decomposition in base units
acceleration	m.s^-2
amount of substance	mol
angle	rad
angular acceleration	s^-2.rad
angular momentum	kg.m^2.s^-1
angular velocity	s^-1.rad
area	m^2
dimensionless	
dynamic viscosity	kg.m^-1.s^-1
electric capacitance	kg^-1.m^-2.s^4.A^2
electric charge	s.A
electric charge density	m^-3.s.A
electric conductance	kg^-1.m^-2.s^3.A^2
electric conductivity	kg^-1.m^-3.s^3.A^2
electric current	A
electric current density	m^-2.A
electric dipole moment	m.s.A
electric field strength	kg.m.s^-3.A^-1
electric flux density	m^-2.s.A
electric potential	kg.m^2.s^-3.A^-1
electric resistance	kg.m^2.s^-3.A^-2
electric resistivity	kg.m^3.s^-3.A^-2
energy	kg.m^2.s^-2
energy flux density	kg.s^-3
entropy	kg.m^2.s^-2.K^-1
force	kg.m.s^-2
frequency	s^-1
inductance	kg.m^2.s^-2.A^-2
kinematic viscosity	m^2.s^-1
length	m
linear density	kg.m^-1
luminance	m^-2.cd
luminous emittence	m^-2.rad^2.cd
luminous flux	rad^2.cd
luminous intensity	cd
magnetic field strength	m^-1.A
magnetic flux	kg.m^2.s^-2.A^-1
magnetic flux density	kg.s^-2.A^-1

continues on next page

Table 2 – continued from previous page

Quantity	Decomposition in base units
magnetic permeability	kg.m.s^-2.A^-2
mass	kg
molar volume	m^-3.mol
moment of inertia	kg.m^2
momentum/impulse	kg.m.s^-1
permittivity	kg^-1.m^-3.s^4.A^2
power	kg.m^2.s^-3
pressure	kg.m^-1.s^-2
radiant intensity	kg.m^2.s^-3.rad^-2
solid angle	rad^2
specific energy	m^2.s^-2
specific volume	kg^-1.m^3
spectral flux density	kg.s^-2
surface density	kg.m^-2
temperature	K
thermal conductivity	kg.m.s^-3.K^-1
time	s
velocity	m.s^-1
volume	m^3
volume density	kg.m^-3
wavenumber	m^-1

2.7 Frequently asked questions

2.7.1 Why an alias ?

the *alias* property in the `astrophysix` package is an optional parameter only mandatory within `SimulationStudy` HDF5 files that are meant to be uploaded on the Galactica simulation database. This optional property can be found in various classes :

- `Project`,
- `SimulationCode`,
- `PostProcessingCode`,
- `Simulation`,
- `PostProcessingRun`.

These *alias* properties are used to reference in a unique way the projects, protocols and experiments displayed on the web pages and appear in the URL of your web browser when you wish to visit the page of a given project or simulation. For example, this is the URL of a *ORION_FIL_MHD* simulation of the *ORION* project in the *STAR_FORM* (`ProjectCategory.StarFormation`) project category :

- http://www.galactica-simulations.eu/db/STAR_FORM/ORION/ORION_FIL_MHD/

2.7.2 How can I check validity for Galactica ?

TODO

2.7.3 How to delete object from lists ?

Let us assume that you wish to remove a *Snapshot* from a *Simulation*. Then you can use the standard `del` python operator to remove it :

```
>>> simu.snapshots
Snapshot list :
+---+---------------------------+--------------------------------------+
| # |           Index           |                 Item                 |
+---+---------------------------+--------------------------------------+
| 0 | My best snapshot !        | 'My best snapshot !' snapshot        |
+---+---------------------------+--------------------------------------+
| 1 | My second best snapshot ! | 'My second best snapshot !' snapshot |
+---+---------------------------+--------------------------------------+
>>> del simu.snapshots[1]
>>> simu.snapshots
Snapshot list :
+---+---------------------------+--------------------------------------+
| # |           Index           |                 Item                 |
+---+---------------------------+--------------------------------------+
| 0 | My best snapshot !        | 'My best snapshot !' snapshot        |
+---+---------------------------+--------------------------------------+
```

See also:

ObjectList example in the API reference.

2.7.4 How to delete files from a Datafile ?

To remove a file from a *Datafile*, you can use the standard `del` python operator:

```
>>> from astrophysix.utils.file import FileType
>>>
>>> power_spectrum_datafile.display_files()
[Power scpectrum] datafile. Attached files :
+-----------+-----------------------------+
| File type |          Filename           |
+-----------+-----------------------------+
| PNG       | spectrum_1.png              |
+-----------+-----------------------------+
| JPEG      | spectrum_with_overlays.jpg  |
+-----------+-----------------------------+
>>>
>>> del power_spectrum_datafile[FileType.PNG_FILE]
>>> power_spectrum_datafile.display_files()
[Power scpectrum] datafile. Attached files :
+-----------+-----------------------------+
| File type |          Filename           |
+-----------+-----------------------------+
| JPEG      | spectrum_with_overlays.jpg  |
+-----------+-----------------------------+
```

See also:

- `Datafile` example in the API reference,
- *Attached files* detailed section.

2.8 Changelog

2.8.1 0.4.1

- Set `InputParameter.name` property as index in *Protocol* input parameter list. Set `InputParameter.key` as optional (Galactica simulation database compatibility).

2.8.2 0.4.0

Implemented `Simulation Datamodel` classes :

- `Project` and `SimulationStudy`,
- *Protocols* (`SimulationCode` and `PostProcessingCode`),
- *Experiments* (`Simulation` and `PostProcessingRun`),
- *Results* (`GenericResult` and `Snapshot`),
- `Datafile`.

2.9 Study and projects

2.9.1 SimulationStudy

class astrophysix.simdm.**SimulationStudy**(*project=None*)

HDF5 simulation study file for Project tree structure persistency

Parameters **project** (`Project`) – study main project

property creation_time

Study creation date/time (`datetime.datetime`).

property last_modification_time

Study last modification date/time (`datetime.datetime`).

classmethod load_HDF5(*study_file_path*)

Loads a new or existing SimulationStudy from a HDF5 (*.h5) file

Parameters **study_file_path** (`string`) – SimulationStudy HDF5 (existing) file path

Returns **study** – Study loaded from HDF5 file.

Return type `SimulationStudy`

property project

Study main project

save_HDF5(*study_fname=None*, *dry_run=False*, *callback=None*, *galactica_checks=False*)

Save the SimulationStudy into a HDF5 (*.h5) file

Parameters

- **study_fname** (string) – Simulation study HDF5 filename.
- **dry_run** (bool) – perform a dry run ? Default False.
- **callback** (callable) – method to execute upon saving each item of the study.
- **galactica_checks** (bool) – Perform Galactica database validity checks and display warning in case of invalid content for upload on Galactica. Default False (quiet mode).

property study_filepath
Simulation study HDF5 file path

property uid
Study UUID

2.9.2 Project and ProjectCategory

class astrophysix.simdm.**ProjectCategory**(*value*)
Project category enum

Example

```
>>> cat = ProjectCategory.PlanetaryAtmospheres
>>> cat.verbose_name
"Planetary atmospheres"
```

Cosmology = ('COSMOLOGY', 'Cosmology')

GalaxyFormation = ('GAL_FORMATION', 'Galaxy formation')

GalaxyMergers = ('GAL_MERGERS', 'Galaxy mergers')

PlanetaryAtmospheres = ('PLANET_ATMO', 'Planetary atmospheres')

SolarMHD = ('SOLAR_MHD', 'Solar Magnetohydrodynamics')

StarFormation = ('STAR_FORM', 'Star formation')

StarPlanetInteractions = ('STAR_PLANET_INT', 'Star-planet interactions')

Supernovae = ('SUPERNOVAE', 'Supernovae')

property alias
Project category alias

classmethod from_alias(*alias*)

Parameters **alias** (string) – project category alias

Returns **c** – Project category matching the requested alias.

Return type *ProjectCategory*

Raises **ValueError** – if requested alias does not match any project category.

Example

```
>>> c = ProjectCategory.from_alias("STAR_FORM")
>>> c.verbose_name
"Star formation"
>>> c2 = ProjectCategory.from_alias("MY_UNKNOWN_CATEGORY")
ValuerError: No ProjectCategory defined with the alias 'MY_UNKNOWN_CATEGORY'.
```

property verbose_name
Project category verbose name

class astrophysix.simdm.**Project**(*Simulation data model*)

Parameters

- **category** (*ProjectCategory* or string) – project category or project category alias (mandatory)
- **project_title** (string) – project title (mandatory)
- **alias** (string) – Project alias (if defined, 16 max characters is recommended)
- **short_description** (string) – project short description
- **general_description** (string) – (long) project description
- **data_description** (string) – available data description in the project
- **directory_path** (string) – project directory path

__eq__(*other*)
Project comparison method

other: ***Project*** project to compare to.

__unicode__()
String representation of the instance

property alias

property category

property data_description
Data description available in this project

property directory_path
Project data directory path

galactica_validity_check(***kwargs*)
Perform validity checks on this instance and eventually log warning messages.

Parameters kwargs (*dict*) – keyword arguments (optional)

property general_description
General description of the project

property project_title

property short_description
Short description of the project

property simulations
Project *Simulation* list (*ObjectList*)

2.10 Protocols

2.10.1 Simulation and post-processing codes

`Protocol` is the generic term for a software tool used to conduct a numerical `Experiment`. There are two different types of protocols :

- *SimulationCode*,
- *PostProcessingCode*.

class astrophysix.simdm.protocol.**SimulationCode**(***kwargs*)
Simulation code

Parameters

- **name** (string) – name (mandatory)
- **code_name** (string) – base code name (mandatory)
- **alias** (string) – code alias
- **url** (string) – reference URL
- **code_version** (string) – code version
- **description** (string) – code description

__eq__(*other*)
SimulationCode comparison method

other: SimulationCode simulation code to compare to

__ne__(*other*)
Not an implied relationship between "rich comparison" equality methods in Python 2.X but only in Python 3.X see https://docs.python.org/2.7/reference/datamodel.html#object.__ne__

other: other instance to compare to

__unicode__()
String representation of the instance

property algorithms
Protocol *Algorithm* list (*ObjectList*)

property alias
Protocol alias

property code_name
Protocol code name

property code_version
Protocol code version

property description
Protocol description

galactica_validity_check(***kwargs*)
Perform validity checks on this instance and eventually log warning messages.

Parameters kwargs (dict) – keyword arguments (optional)

property input_parameters
Protocol *InputParameter* list (*ObjectList*)

property name
Protocol name

property physical_processes
Simulation code *PhysicalProcess* list (*ObjectList*)

property uid

property url
Protocol code url

class astrophysix.simdm.protocol.**PostProcessingCode**(***kwargs*)
Post-processing code

Parameters

- **name** (string) – name (mandatory)
- **code_name** (*:obj:`string*) – base code name (mandatory)
- **alias** (string) – code alias
- **url** (string) – reference URL
- **code_version** (string) – code version
- **description** (string) – code description

__eq__(*other*)
Protocol comparison method

other: Protocol Protocol to compare to

__ne__(*other*)
Not an implied relationship between "rich comparison" equality methods in Python 2.X but only in Python 3.X see https://docs.python.org/2.7/reference/datamodel.html#object.__ne__

other: other instance to compare to

__unicode__()
String representation of the instance

property algorithms
Protocol *Algorithm* list (*ObjectList*)

property alias
Protocol alias

property code_name
Protocol code name

property code_version
Protocol code version

property description
Protocol description

galactica_validity_check(***kwargs*)
Perform validity checks on this instance and eventually log warning messages.

Parameters kwargs (dict) – keyword arguments (optional)

property input_parameters
Protocol *InputParameter* list (*ObjectList*)

property name
Protocol name

property uid

property url
Protocol code url

2.10.2 Input parameters

class astrophysix.simdm.protocol.**InputParameter**(***kwargs*)
Protocol input parameter

Parameters

- **name** (string) – input parameter name (mandatory)
- **key** (string) – input parameter configuration key
- **description** (string) – input parameter description

__eq__(*other*)
InputParameter comparison method

other: *InputParameter* input parameter to compare to

__ne__(*other*)
Not an implied relationship between "rich comparison" equality methods in Python 2.X but only in Python 3.X see https://docs.python.org/2.7/reference/datamodel.html#object.__ne__

other: other instance to compare to

__unicode__()
String representation of the instance

property description
Input parameter description

galactica_validity_check(***kwargs*)
Perform validity checks on this instance and eventually log warning messages.

Parameters **kwargs** (dict) – keyword arguments (optional)

property key
Input parameter configuration key

property name
Input parameter name

property uid

2.10.3 Algorithms

Algorithm type

class astrophysix.simdm.protocol.**AlgoType**(*value*)
Algorithm type enum

Example

```
>>> t = AlgoType.PoissonMultigrid
>>> t.name
"Multigrid Poisson solver"
```

AdaptiveMeshRefinement = ('AMR', 'Adaptive mesh refinement')

FriendOfFriend = ('FOF', 'Friend-of-friend')

Godunov = ('Godunov', 'Godunov scheme')

HLLCRiemann = ('HLLC', 'Harten-Lax-van Leer-Contact Riemann solver')

ParticleMesh = ('PM', 'Particle-mesh solver')

PoissonConjugateGradient = ('Poisson_CG', 'Conjugate Gradient Poisson solver')

PoissonMultigrid = ('Poisson_MG', 'Multigrid Poisson solver')

RayTracer = ('ray_tracer', 'Ray-tracer')

SmoothParticleHydrodynamics = ('SPH', 'Smooth particle hydrodynamics')

VoronoiMovingMesh = ('Voronoi_MM', 'Voronoi tesselation-based moving mesh')

classmethod from_key(*key*)

Parameters **key** (string) – algorithm type key

Returns **t** – Algorithm type matching the requested key.

Return type *AlgoType*

Raises **ValueError** – if requested key does not match any algorithm type.

Example

```
>>> t = AlgoType.from_key("FOF")
>>> t.name
"Friend-of-friend"
>>> t2 = AlgoType.from_key("MY_UNKNOWN_ALGO_TYPE")
ValuerError: No AlgoType defined with the key 'MY_UNKNOWN_ALGO_TYPE'.
```

property key
Algorithm type indexing key

Algorithm

class astrophysix.simdm.protocol.**Algorithm**(***kwargs*)
Protocol algorithm

Parameters

- **algo_type** (*AlgoType* or string) – AlgoType enum value or AlgoType valid key (mandatory).
- **description** (string) – algorithm description

__eq__(*other*)
Algorithm comparison method

other: ***Algorithm*** algorithm to compare to

__ne__(*other*)
Not an implied relationship between "rich comparison" equality methods in Python 2.X but only in Python 3.X see https://docs.python.org/2.7/reference/datamodel.html#object.__ne__

other: other instance to compare to

__unicode__()
String representation of the instance

property algo_type
Algorithm type (*AlgoType*)

property description
Algorithm description

galactica_validity_check(***kwargs*)
Perform validity checks on this instance and eventually log warning messages.

Parameters **kwargs** (dict) – keyword arguments (optional)

property name
Algorithm name

property uid

2.10.4 Physical processes

Physics

class astrophysix.simdm.protocol.**Physics**(*value*)
Physics enum

Example

```
>>> ph = Physics.MHD
>>> ph.name
"Magnetohydrodynamics"
```

AGNFeedback = ('AGN_feedback', 'AGN feedback')

Hydrodynamics = ('hydro', 'Hydrodynamics')

MHD = ('mhd', 'Magnetohydrodynamics')

MolecularCooling = ('mol_cooling', 'Molecular cooling')

SelfGravity = ('self_gravity', 'Self-gravity')

StarFormation = ('star_form', 'Star formation')

SupernovaeFeedback = ('sn_feedback', 'Supernovae feedback')

classmethod from_key(*key*)

Parameters **key** (string) – physics key

Returns **t** – Physics matching the requested key.

Return type *Physics*

Raises **ValueError** – if requested key does not match any physics.

Example

```
>>> ph = Physics.from_key("star_from")
>>> ph.name
"Star formation"
>>> ph2 = Physics.from_key("MY_UNKNOWN_PHYSICS")
ValuerError: No Physics defined with the key 'MY_UNKNOWN_PHYSICS'.
```

property key
Physics indexing key

Physical process

class astrophysix.simdm.protocol.**PhysicalProcess**(***kwargs*)
Simulation code physical process

Parameters

- **physics** (*Physics* or string) – Physics enum value or Physics valid key. (mandatory)
- **description** (string) – physics description

__eq__(*other*)
PhysicalProcess comparison method

other: PhysicalProcess physical process to compare to

__ne__(*other*)
Not an implied relationship between "rich comparison" equality methods in Python 2.X but only in Python 3.X see https://docs.python.org/2.7/reference/datamodel.html#object.__ne__

other: other instance to compare to

__unicode__()
String representation of the instance

property description
Physical process description

galactica_validity_check(***kwargs*)
Perform validity checks on this instance and eventually log warning messages.

Parameters **kwargs** (dict) – keyword arguments (optional)

property name
Physical process name

property physics
Prrocess type (*Physics*)

property uid

2.11 Experiments

2.11.1 Simulation and post-processing runs

Numerical `Experiments` can be of two different types:

- *Simulation*,
- *PostProcessingRun*.

class astrophysix.simdm.experiment.**Simulation**(*Simulation data model*)

Parameters

- **name** (string) – Simulation name (mandatory)
- **simu_code** (*SimulationCode*) – Simulation code used for this simulation (mandatory)
- **alias** (string) – Simulation alias (if defined, 16 max characters is recommended)
- **description** (string) – Long simulation description
- **directory_path** (string) – Simulation data directory path
- **execution_time** (string) – Simulation execution time in the format '%Y-%m-%d %H:%M:%S'

EXETIME_FORMAT = '%Y-%m-%d %H:%M:%S'

__eq__(*other*)
Simulation comparison method

other: ***Simulation*** simulation to compare to

__ne__(*other*)
Not an implied relationship between "rich comparison" equality methods in Python 2.X but only in Python 3.X see https://docs.python.org/2.7/reference/datamodel.html#object.__ne__

other: other instance to compare to

__unicode__()
String representation of the instance

property alias
Experiment alias. Can be edited.

property applied_algorithms
Experiment applied algorithm list (*ObjectList*)

property description
Experiment description. Can be edited.

property directory_path
Experiment data directory path. Can be edited.

property execution_time
Simulation execution date/time. Can be edited.

Example

```
>>> simu = Simulation(simu_code=gadget4, name="Maxi Cosmic", execution_time=
↪"2020-09-10 14:25:48")
>>> simu.execution_time = '2020-09-28 18:45:24'
```

property execution_time_as_utc_datetime
UTC execution time of the simulation (timezone aware)

galactica_validity_check(***kwargs*)
Perform validity checks on this instance and eventually log warning messages.

Parameters **kwargs** (*dict*) – keyword arguments (optional)

property generic_results
Experiment generic result list (*ObjectList*)

property name
Experiment name. Can be edited.

property parameter_settings
Experiment parameter setting list (*ObjectList*)

property post_processing_runs
Simulation associated post-processing run list (*ObjectList*)

property resolved_physics
Simulation resolved physical process list (*ObjectList*).

property simulation_code
SimulationCode used to run this simulation. Cannot be changed after simulation initialisation.

property snapshots
Experiment snapshot list (*ObjectList*)

property uid

class astrophysix.simdm.experiment.**PostProcessingRun**(**args*, ***kwargs*)
Post-processing run (Simulation data model)

Parameters

- **name** (string) – post-processing run name (mandatory)
- **ppcode** (*PostProcessingCode*) – post-processing code used for this post-processing run (mandatory)
- **alias** (string) – Post-processing run alias (if defined, 16 max characters is recommended)
- **description** (string) – Long post-processing run description

__eq__(*other*)
PostProcessingRun comparison method

other: PostProcessingRun post-processing run to compare to

__ne__(*other*)
Not an implied relationship between "rich comparison" equality methods in Python 2.X but only in Python 3.X see https://docs.python.org/2.7/reference/datamodel.html#object.__ne__

other: other instance to compare to

__unicode__()
String representation of the instance

property alias
Experiment alias. Can be edited.

property applied_algorithms
Experiment applied algorithm list (*ObjectList*)

property description
Experiment description. Can be edited.

property directory_path
Experiment data directory path. Can be edited.

galactica_validity_check(***kwargs*)
Perform validity checks on this instance and eventually log warning messages.

Parameters **kwargs** (*dict*) – keyword arguments (optional)

property generic_results
Experiment generic result list (*ObjectList*)

property name
Experiment name. Can be edited.

property parameter_settings
Experiment parameter setting list (*ObjectList*)

property postpro_code
PostProcessingCode used to run this post-processing run. Cannot be changed after post-processing run initialisation.

property snapshots
Experiment snapshot list (*ObjectList*)

property uid

2.11.2 Parameter settings

Parameter visibility flag

class astrophysix.simdm.experiment.**ParameterVisibility**(*value*)
Parameter setting visibility flag (enum)

Example

```
>>> vis = ParameterVisibility.BASIC_DISPLAY
>>> vis.display_name
"Basic display"
```

ADVANCED_DISPLAY = ('advanced', 'Advanced display')

BASIC_DISPLAY = ('basic', 'Basic display')

NOT_DISPLAYED = ('not_displayed', 'Not displayed')

property display_name
Parameter visibility display name

classmethod from_key(*key*)

Parameters **key** (string) – parameter visibility flag key

Returns **t** – Parameter vsibility flag matching the requested key.

Return type *ParameterVisibility*

Raises **ValueError** – if requested key does not match any parameter visibility.

Example

```
>>> vis = ParameterVisibility.from_key("advanced")
>>> vis.display_name
"Advanced display"
>>> vis2 = ParameterVisibility.from_key("MY_UNKNOWN_FLAG")
ValuerError: No ParameterVisibility defined with the key 'MY_UNKNOWN_FLAG'.
```

property key

Parameter visibility flag key

Parameter setting

class astrophysix.simdm.experiment.**ParameterSetting**(***kwargs*)

Experiment input parameter setting class

Parameters

- **input_param** (*InputParameter*) – protocol input parameter (mandatory)
- **value** (float or int or string or bool) – numeric/string/boolean value of the input parameter (mandatory)
- **unit** (Unit or string) – parameter value unit (or unit key string)
- **visibility** (*ParameterVisibility*) – Parameter setting visibility (for display use only). Default *BASIC_DISPLAY*

__eq__(*other*)

ParameterSetting comparison method

other: ***ParameterSetting*** parameter setting to compare to

__ne__(*other*)

Not an implied relationship between "rich comparison" equality methods in Python 2.X but only in Python 3.X see https://docs.python.org/2.7/reference/datamodel.html#object.__ne__

other: other instance to compare to

__unicode__()

String representation of the instance

galactica_validity_check(***kwargs*)

Perform validity checks on this instance and eventually log warning messages.

Parameters **kwargs** (dict) – keyword arguments (optional)

property input_parameter

Experiment protocol's *InputParameter*. Cannot be edited after parameter setting initialisation.

property parameter_key

Experiment protocol's *InputParameter* key

property uid

property unit

Parameter value unit (`Unit`). Can be edited.

Example

```
>>> from astrophysix import units as U
>>> psetting = ParameterSetting(input_param=inpp, value=0.5, unit=U.pc)
>>> psetting.unit = U.kpc
>>> psetting.unit = "Mpc"
```

property value

Parameter value.

Can be set to a `bool`, `string`, `int` or `float` value. When set, *`value_type`* is also set accordingly.

Example

```
>>> psetting = ParameterSetting(input_param=inpp, value=0.5)
>>> type(psetting.value) is float and psetting.value == 0.5 and psetting.
↪value_type == DataType.REAL
True
>>> psetting.value = "true"
>>> type(psetting.value) is bool and psetting.value is True and psetting.
↪value_type == DataType.BOOLEAN
True
>>> psetting.value = "false"
>>> type(psetting.value) is bool and psetting.value is False and psetting.
↪value_type == DataType.BOOLEAN
True
>>> psetting.value = "banana"
>>> type(psetting.value) is str and psetting.value == "banana" and psetting.
↪value_type == DataType.STRING
True
>>> psetting.value = 4.256
>>> type(psetting.value) is float and psetting.value == 4.256 and psetting.
↪value_type == DataType.REAL
True
>>> psetting.value = 58.0
>>> type(psetting.value) is int and psetting.value == 58 and psetting.value_
↪type == DataType.INTEGER
True
>>> psetting.value = "3.584e2"
>>> type(psetting.value) is float and psetting.value == 358.4 and psetting.
↪value_type == DataType.REAL
True
>>> psetting.value = "-254"
>>> type(psetting.value) is int and psetting.value == -254 and psetting.value_
↪type == DataType.INTEGER
True
```

property value_type

Parameter value type (*`DataType`*)

property visibility

Parameter setting visibility flag (*`ParameterVisibility`*). Can be edited.

Example

```
>>> psetting = ParameterSetting(input_param=inpp, value=0.5,␣
↪visibility=ParameterVisibility.ADVANCED_DISPLAY)
>>> psetting.visibility = ParameterVisibility.BASIC_DISPLAY
>>> psetting.visibility = "not_displayed"
```

2.11.3 Applied algorithms

class astrophysix.simdm.experiment.**AppliedAlgorithm**(***kwargs*)

Experiment applied algorithm class

Parameters

- **algorithm** (*Algorithm*) – protocol algorithm (mandatory).
- **details** (string) – implementation details.

__eq__(*other*)

AppliedAlgorithm comparison method

other: ***AppliedAlgorithm*** applied algorithm to compare to

__ne__(*other*)

Not an implied relationship between "rich comparison" equality methods in Python 2.X but only in Python 3.X see https://docs.python.org/2.7/reference/datamodel.html#object.__ne__

other: other instance to compare to

__unicode__()

String representation of the instance

property algo_name

Algorithm name. Cannot be edited.

property algorithm

Experiment protocol's *Algorithm*. Cannot be edited after applied algorithm initialisation.

galactica_validity_check(***kwargs*)

Perform validity checks on this instance and eventually log warning messages.

Parameters kwargs (dict) – keyword arguments (optional)

property implementation_details

Applied algorithm implementation details (string). Can be edited.

property uid

2.11.4 Resolved physical processes

class astrophysix.simdm.experiment.**ResolvedPhysicalProcess**(***kwargs*)

Simulation resolved physical process class

Parameters

- **physics** (*PhysicalProcess*) – simulation code's *PhysicalProcess* instance (mandatory)
- **details** (string) – resolved physical process implementation details

__eq__(*other*)
ResolvedPhysicalProcess comparison method

other: ***ResolvedPhysicalProcess*** resolved physical process to compare to

__ne__(*other*)
Not an implied relationship between "rich comparison" equality methods in Python 2.X but only in Python 3.X see https://docs.python.org/2.7/reference/datamodel.html#object.__ne__

other: other instance to compare to

__unicode__()
String representation of the instance

galactica_validity_check(***kwargs*)
Perform validity checks on this instance and eventually log warning messages.

Parameters **kwargs** (`dict`) – keyword arguments (optional)

property implementation_details
Resolved physical process implementation details. Editable.

property physical_process
Simulation code's *PhysicalProcess*. Cannot be edited after instance initialisation

property process_name
Simulation code's *PhysicalProcess* name. Cannot be edited.

property uid

2.12 Results

2.12.1 Generic results and snapshots

class astrophysix.simdm.results.generic.**GenericResult**(***kwargs*)
Experiment generic result class

Parameters

- **name** (`string`) – result name (mandatory)
- **description** (`string`) – result description
- **directory_path** (`string`) – result data directory path

__eq__(*other*)
GenericResult comparison method

other: ***GenericResult*** generic result to compare to

__ne__(*other*)
Not an implied relationship between "rich comparison" equality methods in Python 2.X but only in Python 3.X see https://docs.python.org/2.7/reference/datamodel.html#object.__ne__

other: other instance to compare to

__unicode__()
String representation of the instance

property datafiles
Result *Datafile* list (*ObjectList*)

property description
Result description. Can be set to any `string` value.

property directory_path
Result directory.path. Can be set to any `string` value.

galactica_validity_check(***kwargs*)
Perform validity checks on this instance and eventually log warning messages.

Parameters **kwargs** (`dict`) – keyword arguments (optional)

property name
Result name. Can be set to a non-empty `string` value.

property uid

class astrophysix.simdm.results.snapshot.**Snapshot**(***kwargs*)
Experiment snapshot class (Simulation data model)

Parameters

- **name** (`string`) – snapshot name (mandatory)
- **description** (`string`) – snapshot description
- **directory_path** (`string`) – snapshot directory path
- **time** ((`float`, `Unit`) `tuple`) – snapshot time info (value, unit) tuple
- **physical_size** ((`float`, `Unit`) `tuple`) – snapshot physical size info (value, unit) tuple
- **data_reference** (`string`) – snapshot data reference (e.g. data directory name, snapshot number) string

__eq__(*other*)
Snapshot comparison method

other: ***Snapshot*** snapshot to compare to

__ne__(*other*)
Not an implied relationship between "rich comparison" equality methods in Python 2.X but only in Python 3.X see https://docs.python.org/2.7/reference/datamodel.html#object.__ne__

other: other instance to compare to

__unicode__()
String representation of the instance

property data_reference
Snapshot data reference (e.g. data directory name, snapshot number). Can be set to any `string` value.

property datafiles
Result *Datafile* list (*ObjectList*)

property description
Result description. Can be set to any `string` value.

property directory_path
Result directory.path. Can be set to any `string` value.

galactica_validity_check(***kwargs*)
Perform validity checks on this instance and eventually log warning messages.

Parameters **kwargs** (`dict`) – keyword arguments (optional)

property name
Result name. Can be set to a non-empty `string` value.

property physical_size
Snapshot physical size info (value, unit) tuple . Can be set to a `float` value (unitless) or a (`float`, `Unit`) tuple.

Example

```
>>> sn = Snapshot(name="My super snapshot")
>>> sn.physical_size = "0.256"
>>> sn.physical_size = ("0.24", U.pc)
>>> sn.physical_size = ("0.45", "kpc")
>>> sn.physical_size[1] == U.kpc
True
>>> sn.physical_size = 4.46
>>> sn.physical_size = (7.89e2, "Mpc")
>>> sn.physical_size[1] == U.Mpc
True
>>> sn.physical_size = (78.54, U.ly)
```

property time
Snapshot time info (value, unit) tuple . Can be set to a `float` value (unitless) or a (`float`, `Unit`) tuple.

Example

```
>>> sn = Snapshot(name="My super snapshot")
>>> sn.time = "0.256"
>>> sn.time[1] == U.none
True
>>> sn.time = ("0.24", U.year)
>>> sn.time = ("0.45", "Myr")
>>> sn.time[1] == U.Myr
True
>>> sn.time = (7.89e2, "Gyr")
>>> sn.time = (78.54, U.min)
```

property uid

2.12.2 Datafiles

Datafile and AssociatedFile

class astrophysix.simdm.datafiles.**Datafile**(***kwargs*)
Datafile class

Parameters

- **name** (`string`) – datafile name (mandatory)
- **description** (`string`) – datafile description

Example

```
>>> from astrophysix.utils import FileType
>>> from astrophysix.simdm.datafiles import JpegImageFile
>>> df = Datafile(name="Pre-stellar cores mass spectrum")
>>> df[FileType.PNG_FILE] = "/data/SIMUS/result_spectrum/mass_spectrum.png"
>>> df[FileType.FITS_FILE] = "/data/SIMUS/result_spectrum/pre-stellar-core-mass-
↪hist.fits"
>>> df[FileType.PNG_FILE] = JpegImageFile.load_file("/data/SIMUS/result_spectrum/
↪hist.jpg")
ValueError: Datafile associated file type mismatch : expected PngImageFile object␣
↪but JpegImageFile was provided.
>>> df[FileType.PNG_FILE] = "/data/SIMUS/result_spectrum/hist.jpg"
AttributeError: Invalid filename for a PNG file (/data/SIMUS/result_spectrum/hist.
↪jpg).
>>> # Removing a file
>>> del df[FileType.FITS_FILE]
```

__delitem__(*ftype*)
Remove associated file given its file type.

Parameters **item** (*FileType*) –

Raises **KeyError** – if the search index type is not a *FileType* instance or if there is no associated file with the required file type.

__eq__(*other*)
Datafile comparison method

Parameters **other** (*Datafile*) – datafile to compare to:

__getitem__(*ftype*)
Get an associated file from the data file, given its file type.

Parameters **ftype** (*FileType*) – Associated file type

Returns **f** – datafile associated file for the required file type.

Return type *AssociatedFile*

Raises **KeyError** – if the search index type is not a *FileType* instance or if there is no associated file with the required file type.

__ne__(*other*)
Not an implied relationship between "rich comparison" equality methods in Python 2.X but only in Python 3.X see https://docs.python.org/2.7/reference/datamodel.html#object.__ne__

other: other instance to compare to

__setitem__(*filetype*, *ass_file*)
Set an associated file with a given file type into the data file.

Parameters

- **filetype** (*FileType*) – Associated file type
- **ass_file** (string or *AssociatedFile*) – Associated file path or instance

__unicode__()
String representation of the data file instance

property description
Datafile description. Can be set to any string value.

display_files()

Show tabulated view of associated files

Example

```
>>> df.display_files()
[My best datafile] datafile. Attached files :
+-----------+-----------------------------+
| File type |          Filename           |
+-----------+-----------------------------+
| PNG       | CEA.png                     |
+-----------+-----------------------------+
| JPEG      | irfu_simple.jpg             |
+-----------+-----------------------------+
| FITS      | cassiopea_A_0.5-1.5keV.fits |
+-----------+-----------------------------+
| TARGZ     | archive.tar.gz              |
+-----------+-----------------------------+
| JSON      | test_header_249.json        |
+-----------+-----------------------------+
| ASCII     | abstract.txt                |
+-----------+-----------------------------+
| HDF5      | study.h5                    |
+-----------+-----------------------------+
| PICKLE    | dict_saved.pkl              |
+-----------+-----------------------------+
```

galactica_validity_check(***kwargs*)

Perform validity checks on this instance and eventually log warning messages.

Parameters **kwargs** (`dict`) – keyword arguments (optional)

property name

Datafile name. Can be set to a non-empty `string` value.

property plot_info

Datafile plot information. Can be set to a *`PlotInfo`* instance.

property uid

class astrophysix.simdm.datafiles.file.**AssociatedFile**(***kwargs*)

class astrophysix.simdm.datafiles.file.**FitsFile**(***kwargs*)

Bases: *`astrophysix.simdm.datafiles.file.AssociatedFile`*

Datafile associated *`FITS_FILE`* file class.

property filename

Gets associated file name. Cannot be edited.

property last_modified

Returns file last modification time. Cannot be edited.

classmethod load_file(*filepath*)

Loads an *`AssociatedFile`* object from a filepath

Parameters **filepath** (`string`) – path of the file to load.

Returns **f** – Loaded associatedfile

Return type *`AssociatedFile`* instance

property raw_file_data
File binary raw data. Cannot be edited.

save_to_disk(*filepath=None*)
Save associated file to an external file on the local filesystem

Parameters **filepath** (`string`) – external file path

class astrophysix.simdm.datafiles.file.**PickleFile**(***kwargs*)
Bases: *astrophysix.simdm.datafiles.file.AssociatedFile*

Datafile associated *PICKLE_FILE* file class.

property filename
Gets associated file name. Cannot be edited.

property last_modified
Returns file last modification time. Cannot be edited.

classmethod load_file(*filepath*)
Loads an *AssociatedFile* object from a filepath

Parameters **filepath** (`string`) – path of the file to load.

Returns **f** – Loaded associatedfile

Return type *AssociatedFile* instance

property raw_file_data
File binary raw data. Cannot be edited.

save_to_disk(*filepath=None*)
Save associated file to an external file on the local filesystem

Parameters **filepath** (`string`) – external file path

class astrophysix.simdm.datafiles.file.**AsciiFile**(***kwargs*)
Bases: *astrophysix.simdm.datafiles.file.AssociatedFile*

Datafile associated *ASCII_FILE* file class.

property filename
Gets associated file name. Cannot be edited.

property last_modified
Returns file last modification time. Cannot be edited.

classmethod load_file(*filepath*)
Loads an *AssociatedFile* object from a filepath

Parameters **filepath** (`string`) – path of the file to load.

Returns **f** – Loaded associatedfile

Return type *AssociatedFile* instance

property raw_file_data
File binary raw data. Cannot be edited.

save_to_disk(*filepath=None*)
Save associated file to an external file on the local filesystem

Parameters **filepath** (`string`) – external file path

class astrophysix.simdm.datafiles.file.**HDF5File**(***kwargs*)
Bases: *astrophysix.simdm.datafiles.file.AssociatedFile*

Datafile associated *HDF5_FILE* file class.

property filename
Gets associated file name. Cannot be edited.

property last_modified
Returns file last modification time. Cannot be edited.

classmethod load_file(*filepath*)
Loads an *AssociatedFile* object from a filepath

Parameters filepath (string) – path of the file to load.

Returns f – Loaded associatedfile

Return type *AssociatedFile* instance

property raw_file_data
File binary raw data. Cannot be edited.

save_to_disk(*filepath=None*)
Save associated file to an external file on the local filesystem

Parameters filepath (string) – external file path

class astrophysix.simdm.datafiles.file.**JsonFile**(***kwargs*)
Bases: *astrophysix.simdm.datafiles.file.AssociatedFile*

Datafile associated *JSON_FILE* file class.

property filename
Gets associated file name. Cannot be edited.

property last_modified
Returns file last modification time. Cannot be edited.

classmethod load_file(*filepath*)
Loads an *AssociatedFile* object from a filepath

Parameters filepath (string) – path of the file to load.

Returns f – Loaded associatedfile

Return type *AssociatedFile* instance

property raw_file_data
File binary raw data. Cannot be edited.

save_to_disk(*filepath=None*)
Save associated file to an external file on the local filesystem

Parameters filepath (string) – external file path

class astrophysix.simdm.datafiles.file.**CSVFile**(***kwargs*)
Bases: *astrophysix.simdm.datafiles.file.AssociatedFile*

Datafile associated *CSV_FILE* file class.

property filename
Gets associated file name. Cannot be edited.

property last_modified
Returns file last modification time. Cannot be edited.

classmethod load_file(*filepath*)
Loads an `AssociatedFile` object from a filepath

Parameters **filepath** (`string`) – path of the file to load.

Returns **f** – Loaded associatedfile

Return type `AssociatedFile` instance

property raw_file_data
File binary raw data. Cannot be edited.

save_to_disk(*filepath=None*)
Save associated file to an external file on the local filesystem

Parameters **filepath** (`string`) – external file path

class astrophysix.simdm.datafiles.file.**TarGzFile**(***kwargs*)
Bases: *astrophysix.simdm.datafiles.file.AssociatedFile*

Datafile associated *TARGZ_FILE* file class.

property filename
Gets associated file name. Cannot be edited.

property last_modified
Returns file last modification time. Cannot be edited.

classmethod load_file(*filepath*)
Loads an `AssociatedFile` object from a filepath

Parameters **filepath** (`string`) – path of the file to load.

Returns **f** – Loaded associatedfile

Return type `AssociatedFile` instance

property raw_file_data
File binary raw data. Cannot be edited.

save_to_disk(*filepath=None*)
Save associated file to an external file on the local filesystem

Parameters **filepath** (`string`) – external file path

class astrophysix.simdm.datafiles.image.**PngImageFile**(***kwargs*)
Bases: `astrophysix.simdm.datafiles.image.ImageFile`

Datafile associated *PNG_FILE* image file class.

property filename
Gets associated file name. Cannot be edited.

property last_modified
Returns file last modification time. Cannot be edited.

classmethod load_file(*filepath*)
Loads an `AssociatedFile` object from a filepath

Parameters **filepath** (`string`) – path of the file to load.

Returns **f** – Loaded associatedfile

Return type `AssociatedFile` instance

property pil_image
Pillow image (JPEG/PNG) image property getter. Implements lazy I/O.

property raw_file_data
File binary raw data. Cannot be edited.

save_to_disk (*filepath=None*)
Save associated file to an external file on the local filesystem

Parameters **filepath** (string) – external file path

class astrophysix.simdm.datafiles.image.**JpegImageFile** (***kwargs*)
Bases: astrophysix.simdm.datafiles.image.ImageFile

Datafile associated *JPEG_FILE* image file class.

property filename
Gets associated file name. Cannot be edited.

property last_modified
Returns file last modification time. Cannot be edited.

classmethod load_file (*filepath*)
Loads an *AssociatedFile* object from a filepath

Parameters **filepath** (string) – path of the file to load.

Returns **f** – Loaded associatedfile

Return type *AssociatedFile* instance

property pil_image
Pillow image (JPEG/PNG) image property getter. Implements lazy I/O.

property raw_file_data
File binary raw data. Cannot be edited.

save_to_disk (*filepath=None*)
Save associated file to an external file on the local filesystem

Parameters **filepath** (string) – external file path

Plot information

class astrophysix.simdm.datafiles.plot.**PlotType** (*value*)
Plot type enum

Example

```
>>> pt = PlotType.HISTOGRAM_2D
>>> pt.alias
"2d_hist"
>>> pt.display_name
"2D histogram"
>>> pt.ndimensions
2
```

HISTOGRAM = ('hist', 'Histogram', 1, 1)

HISTOGRAM_2D = ('2d_hist', '2D histogram', 2, 1)

IMAGE = ('img', 'Image', 2, 1)

LINE_PLOT = ('line', 'Line plot', 1, 0)

MAP_2D = ('2d_map', '2D map', 2, 1)

SCATTER_PLOT = ('scatter', 'Scatter plot', 1, 0)

property alias
Plot type alias

property axis_size_offset

property display_name
Plot type verbose name

classmethod from_alias (*alias*)
Find a PlotType according to its alias

Parameters **alias** (string) – required plot type alias

Returns **ft** – Plot type matching the requested alias.

Return type *PlotType*

Raises **ValueError** – if requested alias does not match any plot type.

Example

```
>>> pt = PlotType.from_alias("hist")
>>> pt.display_name
"Histogram"
>>> pt2 = PlotType.from_alias("MY_UNKNOWN_PLOT_YPE")
ValuerError: No PlotType defined with the alias 'MY_UNKNOWN_PLOT_YPE'.
```

property ndimensions
Plot type number of dimensions

class astrophysix.simdm.datafiles.plot.**PlotInfo** (***kwargs*)
Datafile class (Simulation data model)

Parameters

- **plot_type** (*PlotType* or string) – Plot type or plot type alias (mandatory)
- **xaxis_values** (numpy.ndarray) – x-axis coordinate values numpy 1D array (mandatory).
- **yaxis_values** (numpy.ndarray) – y-axis coordinate numpy 1D array (mandatory).
- **values** (numpy.ndarray) – plot data values numpy array (mandatory for 2D plots).
- **xlabel** (string) – x-axis label
- **ylabel** (string) – y-axis label
- **values_label** (string) – plot values label
- **xaxis_unit** – TODO
- **yaxis_unit** – TODO
- **values_unit** – TODO
- **xaxis_log_scale** (bool) – TODO
- **yaxis_log_scale** (bool) – TODO
- **values_log_scale** (bool) – TODO

- **plot_title** (string) – Plot title.

__eq__(*other_plot_info*)
PlotInfo comparison method

Parameters **other_plot_info** (*PlotInfo*) – plot info object to compare to:

__unicode__()
String representation of the instance

galactica_validity_check(***kwargs*)
Perform validity checks on this instance and eventually log warning messages.

Parameters **kwargs** (*dict*) – keyword arguments (optional)

property plot_type
Returns the plot type (*PlotType*). Cannot be edited.

set_data(*xaxis_values*, *yaxis_values*, *values=None*)
Set plot data arrays.

Parameters

- **xaxis_values** (numpy.ndarray) – x-axis coordinate array
- **yaxis_values** (numpy.ndarray) – TODO
- **values** (numpy.ndarray) – TODO

property title
Plot title. Can be set to any `string` value.

property values
Plot values array. Cannot be edited. Implements lazy I/O.

> **Note:** To edit plot values, see *PlotInfo.set_data()* method.

property values_label
plot values label. Can be set to any `string` value.

property values_log_scale
value log scale boolean flag. Can be edited to any `bool` value.

property values_unit
TODO

property xaxis_log_scale
x-axis log scale boolean flag. Can be edited to any `bool` value.

property xaxis_unit
TODO

property xaxis_values
Plot x-axis coordinate array (`numpy.ndarray`). Cannot be edited. Implements lazy I/O.

> **Note:** To edit plot values, see *PlotInfo.set_data()* method.

property xlabel
x-axis label. Can be set to any `string` value.

property yaxis_log_scale
y-axis log scale boolean flag. Can be edited to any `bool` value.

property yaxis_unit
TODO

property yaxis_values
Plot y-axis coordinate array (`numpy.ndarray`). Cannot be edited. Implements lazy I/O.

Note: To edit plot values, see `PlotInfo.set_data()` method.

property ylabel
y-axis label. Can be set to any `string` value.

2.13 Miscellaneous

2.13.1 Datatype enum

class astrophysix.simdm.utils.**DataType**(*value*)
Value data type enum

Example

```
>>> dt = DataType.INTEGER
>>> dt.name
"Integer number"
```

BOOLEAN = ('bool', 'Boolean')

COMPLEX = ('comp', 'Complex number')

DATETIME = ('time', 'Datetime')

INTEGER = ('int', 'Integer number')

RATIONAL = ('rat', 'Rational number')

REAL = ('real', 'Real number')

STRING = ('str', 'String')

classmethod from_key(*k*)

Parameters **key** (`string`) – data type key

Returns **t** – Physics matching the requested key.

Return type `DataType`

Raises **ValueError** – if requested key does not match any physics.

Example

```
>>> dt = DataType.from_key("rat")
>>> dt.name
"Rational number"
>>> dt2 = DataType.from_key("MY_UNKNOWN_DTYPE")
ValuerError: No DataType defined with the key 'MY_UNKNOWN_DTYPE'.
```

property key
Data type index key

2.13.2 Object lists

class astrophysix.simdm.utils.**ObjectList**(*obj_class*, *index_prop_name*, *validity_check=None*)

Generic object list container class

Parameters

- **obj_class** (type) – base class of the objects that can be added to the list
- **index_prop_name** (string) – object property name used as a list index
- **validity_check** (callable) – method called upon object addition into the list. Default None.

Examples

```
>>> run1 = Simulation(simu_code=arepo, name="Pure-hydro run (isolated galaxy)")
>>> run2 = Simulation(simu_code=arepo, name="MHD run")
>>> run3 = project.simulation.add(Simulation(simu_code=arepo, name="Hydro run␣
↪with BH feedback")
>>> run4 = Simulation(simu_code=arepo, name="MHD run with BH feedback")
>>> project.simulation.add(run1)
>>> project.simulation.add(run2)
>>> project.simulation.add(run3)
>>> project.simulation.add(run4, insert_pos=2)  # Insert at position 2, not␣
↪appendend at the end of the list
>>> len(project.simulations)
4
>>> print(str(project.simulations))
Simulation list :
+---+-----------------------------------+----------------------------------------
↪------+
| # |               Index               |                          Item          ␣
↪       |
+---+-----------------------------------+----------------------------------------
↪------+
| 0 | Pure-hydro run (isolated galaxy)  | 'Pure-hydro run (isolated galaxy)'␣
↪simulation |
+---+-----------------------------------+----------------------------------------
↪------+
| 1 | MHD run                           | 'MHD run' simulation                   ␣
↪       |
+---+-----------------------------------+----------------------------------------
↪------+
```

(continues on next page)

(continued from previous page)

```
| 2 | MHD run with BH feedback           | 'MHD run with BH feedback' simulation ␣
↪      |
+---+-----------------------------------+---------------------------------------
↪------+
| 3 | Hydro run with BH feedback         | 'Hydro run with BH feedback' simulation␣
↪      |
+---+-----------------------------------+---------------------------------------
↪------+
>>> run3 is project.simulations[3]  # Search by item position
True
>>> project.simulations["MHD run"]  # Search by item index value
'MHD run' simulation
>>> del project.simulations[0]
>>> del project.simulations["MHD run"]
>>> del project.simulations[run4]
>>> print(str(project.simulations))
Simulation list :
+---+-----------------------------------+---------------------------------------
↪------+
| # |              Index                |                          Item          ␣
↪      |
+---+-----------------------------------+---------------------------------------
↪------+
| 0 | Hydro run with BH feedback         | 'Hydro run with BH feedback' simulation␣
↪      |
+---+-----------------------------------+---------------------------------------
↪------+
```

__delitem__(*item*)
Delete an object from the list.

Parameters **item** (`object` or `int` or `string`) – instance to delete, object position in the list (`int`) or index property value (`string`) of the object to remove from the list.

__eq__(*other*)
Object list comparison method

Parameters **other** (*ObjectList*) – other object list to compare to

__getitem__(*index*)
Get an object from the list.

Parameters **item** (`int` or `string`) – object position in the list (`int`) or index property value (`string`) of the object to fetch from the list.

Returns **o** – Found object in the list. None if none were found.

Return type `object` of type self.object_class

Raises

- **AttributeError** – if the search index type is neither an `int` nor a `string`.
- **IndexError** – if the `int` search index value is lower than 0 or larger than the length of the list - 1.

__iter__()
Basic object list iterator

__len__()
Size of the object list

__unicode__()
String representation of the instance

add(*obj*, *insert_pos=- 1*)
Adds a instance to the list at a given position

Parameters

- **obj** (`object`) – instance to insert in the list
- **insert_pos** (`int`) – insertion position in the simulation list. Default -1 (last).

find_by_uid(*uid*)
Find an object in the list with a matching UUID

Parameters uid (`UUID` or `string`) – UUID or UUID string representation of the object to search for.

Returns o

Return type Matching object with corresponding UUID,if any. Otherwise returns None

galactica_validity_check(***kwargs*)
Perform validity checks on this instance and eventually log warning messages.

Parameters kwargs (`dict`) – keyword arguments (optional)

property index_attribute_name
Name of the object property used as an index in this object list

property object_class
Type of object that can be added into the list

2.14 Physical quantities/constants/units

class astrophysix.units.unit.**Unit**(*name=''*, *base_unit=None*, *coeff=1.0*, *dims=None*, *descr=None*, *latex=None*)
Dimensional physical unit class

Parameters

- **name** (`string`) – Unit name
- **base_unit** (`Unit` instance) – Composite unit from which this instance should be initialised
- **coeff** (`float`) – dimensionless value of the unit instance.
- **dims** (8-`tuple` of `int`) – dimension of the unit object expressed in the international unit system (`kg`, `m`, `s`, `K`, `A`, `mol`, `rad`, `cd`)
- **descr** (`string` or **None**) – Unit description
- **latex** (`string` or **None**) – Unit displayed name (latex format)

Examples

```
>>> cs_m_s = Unit(name="cs", coeff=340.0, dims=(0, 1, -1, 0, 0, 0, 0, 0), descr=
↪"sound speed unit")
>>> print("sound speed = {v:g} m/h".format(v=cs_m_s.express(km/hour)))
sound speed = 1224 km/h
>>>
>>> dens = Unit(name="Msun/kpc^3", base_unit=Msun/kpc**3, descr="Solar mass per␣
↪cubic kiloparsec",
                latex="{u1:s}.{u2:s}^{{-3}}".format(u1=Msun.latex, u2=kpc.latex))
>>> print(dens)
(6.76957356533e-29 m^-3.kg)
```

UNKNOWN_PHYSICAL_TYPE = 'unknown'

__eq__(*other*)
Checks Unit instance equality

Parameters **other** (`Unit`) – other unit instance to compare to

Returns **e** – True if `Unit.coeff` and `Unit.dimensions` are identical, otherwise False.

Return type `bool`

appropriate_unit(*nearest_log10=1.0*)
Try to find the better suited unit (among available equivalent units to represent this unit).

Parameters **nearest_log10** (`float`) – log of the nearest value to round to. Default 1.0.

Example

```
>>> u = 2426.2 * U.ly
>>> bv, bu = u.appropriate_unit()
>>> print("Appropriate unit : 2426.2 ly = {v:g} {bu:s}".format(v=bv, bu=bu.
↪name))
Appropriate unit : 2426.2 ly = 0.743876 kpc
```

property coeff
Constant value of this unit

classmethod create_unit(*name='', base_unit=None, coeff=1.0, dims=None, descr=None, latex=None*)
Add a new Unit instance to the registry

Parameters

- **name** (`string`) – Unit name
- **base_unit** (`Unit` instance) – Composite unit from which this instance should be initialised
- **coeff** (`float`) – dimensionless value of the unit instance.
- **dims** (8-`tuple` of `int`) – dimension of the unit object expressed in the international unit system (`kg`, `m`, `s`, `K`, `A`, `mol`, `rad`, `cd`)
- **descr** (`string` or **None**) – Unit description
- **latex** (`string` or **None**) – Unit displayed name (latex format)

Raises **ValueError** – If the provided *name* already corresponds to a unit in the registry.

property description
Unit description

property dimensions
Unit dimension array

equivalent_unit_list()
Get the equivalent unit list (with same physical type)

Example

```
>>> print(U.kg.equivalent_unit_list())
[g : (0.001 kg), t : (1000 kg), mH : (1.66e-27 kg), Msun : (1.9889e+30 kg),␣
↪Mearth : (5.9722e+24 kg)]
```

express(*unit*)
Unit conversion method. Gives the conversion factor of this `Unit` expressed into another (dimension-compatible) given `Unit`.

Checks that :

- the **unit** param. is also a `Unit` instance
- the **unit** param. is dimension-compatible.

Parameters **unit** (`Unit`) – unit in which the conversion is made

Returns **fact** – conversion factor of this unit expressed in **unit**

Return type `float`

Examples

- Conversion of a kpc expressed in light-years :

```
>>> factor = kpc.express(ly)
>>> print("1 kpc = {fact:f} ly".format(fact=factor))
1 kpc = 3261.563777 ly
```

- Conversion of $1M_{\odot}$ into kpc/Myr :

```
>>> print(Msun.express(kpc/Myr))
UnitError: Incompatible dimensions between :
- Msun : (1.9889e+30 kg) (type: mass) and
- (977792 m.s^-1) (type: velocity)
```

classmethod from_name(*unit_name*)
Get a `Unit` from its name in the `astrophysix` unit registry.

Parameters **unit_name** (`string`) – name of the unit to search.

Raises **AttributeError** – if *unit_name* attribute does not correspond to any unit in the `astrophysix` unit registry.

identical(*other_unit*)
Strict unit instance comparison method

Parameters **other_unit** (`Unit`) – other unit to compare to.

Returns **e** – True only if *other_unit* is equals to *self* AND has identical name/description/LaTex formula. Otherwise returns False.

Return type `bool`

info()

Print information about this unit. If any, print the name and description of this unit, then print the value of this unit and the list of equivalent unit contained in the built-in unit registry associated with their conversion factor.

Example

```
>>> U.kpc.info()
Unit : kpc
----------
Kiloparsec
Value
-----
3.0856775814671917e+19 m
Equivalent units
----------------
* m              :    3.24078e-20 kpc
* um             :    3.24078e-26 kpc
* mm             :    3.24078e-23 kpc
* cm             :    3.24078e-22 kpc
* nm             :    3.24078e-29 kpc
* km             :    3.24078e-17 kpc
* Angstrom       :    3.24078e-30 kpc
* au             :    4.84814e-09 kpc
* pc             :          0.001 kpc
* Mpc            :           1000 kpc
* Gpc            :          1e+06 kpc
* Rsun           :    2.25399e-11 kpc
* ly             :    0.000306601 kpc
```

is_base_unit()

Checks whether the Unit is a base SI Unit (kg, m, s, K, A, mol, rad, cd).

Returns **b** – True only if unit is a base SI unit(kg, m, s, K, A, mol, rad, cd). Otherwise returns False.

Return type `bool`

classmethod iterate_units(*phys_type=None*)

Unit iterator method. Iterates over all units in the `astrophysix` unit registry.

Parameters **phys_type** (`string`) – Name of the physical quantity type of the units to iterate over. Default **None** (all physical quantities).

Yields **u** (*`Unit`*) – unit of the required physical quantity type, if any given.

property latex

Unit displayed name (LaTex format)

property name

Unit name

property physical_type

Get the unit physical type (dimensioned physical quantity).

Returns **t** – The name of the physical quantity, or *Unit.UNKNOWN_PHYSICAL_TYPE* if the physical quantity is unknown.

Return type string

2.15 Utils

class astrophysix.utils.file.**FileType**(*value*)
File type enum

Example

```
>>> ft = FileType.ASCII_FILE
>>> ft.alias
"ASCII"
>>> ft.extension_list
[".dat", ".DAT", ".txt", ".TXT", ".ini", ".INI"]
```

ASCII_FILE = ('ASCII', ['.dat', '.DAT', '.txt', '.TXT', '.ini', '.INI'])

CSV_FILE = ('CSV', ['.csv', '.CSV'])

FITS_FILE = ('FITS', ['.fits', '.FITS'])

HDF5_FILE = ('HDF5', ['.h5', '.H5', '.hdf5', '.HDF5'])

JPEG_FILE = ('JPEG', ['.jpg', '.jpeg', '.JPG', '.JPEG'])

JSON_FILE = ('JSON', ['.json', '.JSON'])

PICKLE_FILE = ('PICKLE', ['.pkl', '.PKL', '.pickle', '.sav', '.save'])

PNG_FILE = ('PNG', ['.png', '.PNG'])

TARGZ_FILE = ('TARGZ', ['.tar.gz', '.TAR.GZ', '.TAR.gz', '.tar.GZ', '.tgz', '.TGZ'])

XML_FILE = ('XML', ['.xml', '.XML'])

__unicode__()
String representation of the enum value. Returns alias.

property alias
Returns file type alias

property default_extension
Returns the first item in the file type extension list

property extension_list
Returns file type valid extension list

property file_regexp
Returns filename matching regular expression for the current file type

classmethod from_alias(*alias*)
Find a FileType according to its alias

Parameters **alias** (string) – required file type alias

Returns **ft** – File type matching the requested alias.

Return type *FileType*

Raises **`ValueError`** – if requested alias does not match any file type.

Example

```
>>> ft = FileType.from_alias("PNG")
>>> ft.extension_list
[".png", ".PNG"]
>>> ft2 = FileType.from_alias("MY_UNKNOWN_FILETYPE")
ValuerError: No FileType defined with the alias 'MY_UNKNOWN_FILETYPE'.
```

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex

PYTHON MODULE INDEX

a

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

X

Y