Python-IHM Documentation

Benjamin Webb

Nov 20, 2020

Contents

This is a Python package to assist in handling mmCIF files compliant with the integrative/hybrid modeling (IHM) extension.

The documentation below documents the library API. For complete worked examples, see the examples directory at GitHub or real systems deposited using the library, such as Nup133.

CHAPTER 1

Contents

1.1 Introduction

This package provides a mechanism to describe an integrative modeling application with a set of Python objects. This includes

- the data used for the modeling, such as previous computional models from comparative or integrative modeling, and experimental datasets from X-ray crystallography, mass spectrometry, electron microscopy;
- the protocol used to generate models, such as molecular dynamics, clustering, and rescoring;
- the actual coordinates of output models, which may be multi-scale (including both atomic coordinates and more coarse-grained representations), multi-state (multiple conformations and/or compositions of the system needed to explain the input data), or ordered (such as different points in a chemical reaction);
- grouping of multiple models into ensembles or clusters;
- validation of models, for example by scoring against data not used in the modeling itself.

Once created, this set of Python objects can be written to an mmCIF file that is compliant with the IHM extension to the PDBx/mmCIF dictionary, suitable for deposition in the PDB-Dev repository. The files are best viewed in a viewer that supports IHM mmCIF, such as UCSF ChimeraX, although they may be partially viewable in regular PDBx mmCIF viewers (likely only the atomic coordinates will be visible).

The Python package can be used standalone, but is primarily intended for use within modeling software such as IMP, or HADDOCK. For example, IMP provides a class which uses this library to convert an IMP::pmi modeling protocol into an mmCIF file.

1.2 Usage

Usage of the library for output consists of first creating a hierarchy of Python objects that together describe the system, and then dumping that hierarchy to an mmCIF file.

For a complete worked example, see the simple docking example.

The top level of the hierarchy in IHM is the *ihm.System*. All other objects are referenced from a System object.

1.2.1 Datasets

Any data used anywhere in the modeling (including in validation) can be referenced with an *ihm.dataset.Dataset*. For example, electron microscopy data is referenced with *ihm.dataset.EMDensityDataset* and small angle scattering data with *ihm.dataset.SASDataset*.

A dataset uses an *ihm.location.Location* object to describe where it is stored. Typically this is an *ihm.location.DatabaseLocation* for something that's deposited in a experiment-specific database such as PDB, EMDB, PRIDE, or EMPIAR, or *ihm.location.InputFileLocation* for something that's stored as a simple file, either on the local disk or at a location described with a DOI such as Zenodo or a publication's supplementary information. See the locations example for more examples.

1.2.2 System architecture

The architecture of the system is described with a number of classes:

- *ihm.Entity* describes each unique sequence.
- *ihm.AsymUnit* describes each asymmetric unit (chain) in the system. For example, a homodimer would consist of two asymmetric units, both pointing to the same entity, while a heterodimer contains two entities. It is also possible for an entity to exist with no asymmetric units pointing to it - this typically corresponds to something seen in an experiment (such as a cross-linking study) which was not modeled. Note that the IHM extension currently contains no support for symmetry, so two chains that are symmetrically related should each be represented as an "asymmetric" unit.
- *ihm.Assembly* groups asymmetric units and/or entities, or parts of them. Assemblies are used to describe which parts of the system correspond to each input source of data, or that were modeled.
- *ihm.representation.Representation* describes how each part of the system was represented in the modeling, for example *as atoms* or *as coarse-grained spheres*.

1.2.3 Restraints and sampling

Restraints, that score or otherwise fit the computational model against the input data, can be created as *ihm.restraint.Restraint* objects. These generally take as input a *Dataset* pointing to the input data, and an *Assembly* describing which part of the model the data corresponds to. For example, there are restraints for *3D EM* and *small angle scattering*.

ihm.protocol.Protocol objects describe how models were generated from the input data. A protocol can consist of *multiple steps*, such as molecular dynamics or Monte Carlo, followed by one or more analyses, such as clustering, filtering, rescoring, or validation, described by *ihm.analysis.Analysis* objects. These objects generally take an *Assembly* to indicate what part of the system was considered and a *group of datasets* to show which data guided the modeling or analysis.

1.2.4 Model coordinates

ihm.model.Model objects give the actual coordinates of the final generated models. These point to the *Assembly* of what was modeled, the *Protocol* describing how the modeling was done, and the *Representation* showing how the model was represented.

Models can be grouped together for any purpose using the *ihm.model.ModelGroup* class. If a given group describes an ensemble of models, the *ihm.model.Ensemble* class allows for additional information on the ensemble

to be provided, such as *localization densities* of parts of the system and precision. Due to size, generally only representative models of an ensemble are deposited in mmCIF, but the *Ensemble* class allows the full ensemble to be referred to, for example in a more compact binary format (e.g. DCD) deposited at a given DOI. Groups of models can also be shown as corresponding to different states of the system using the *ihm.model.State* class.

1.2.5 Metadata

Metadata can also be added to the system, such as

- *ihm.Citation*: publication(s) that describe this modeling or the methods used in it.
- *ihm.Software*: software packages used to process the experimental data, generate intermediate inputs, do the modeling itself, and/or process the output.
- *ihm.Grant*: funding support for the modeling.
- *ihm.reference.UniProtSequence*: information on a sequence used in modeling, in UniProt.

1.2.6 Residue numbering

The library keeps track of several numbering schemes to reflect the reality of the data used in modeling:

- *Internal numbering*. Residues are always numbered sequentially starting at 1 in an *Entity*. All references to residues or residue ranges in the library use this numbering.
- *Author-provided numbering*. If a different numbering scheme is used by the authors, for example to correspond to the numbering of the original sequence that is modeled, this can be given as an author-provided numbering for one or more asymmetric units. See the `auth_seq_id_map` parameter to *AsymUnit*. (The mapping between author-provided and internal numbering is given in the `pdbx_poly_seq_scheme` table in the mmCIF file.)
- *Starting model numbering*. If the initial state of the modeling is given by one or more PDB files, the numbering of residues in those files may not line up with the internal numbering. In this case an offset from starting model numbering to internal numbering can be provided - see the `offset` parameter to *StartingModel*.

1.2.7 Output

Once the hierarchy of classes is complete, it can be freely inspected or modified. All the classes are simple lightweight Python objects, generally with the relevant data available as member variables. For example, modeling packages such as IMP will typically generate an IHM hierarchy from their own internal data models, but in many cases some information relevant to IHM (such as the *associated publication*) cannot be determined automatically and can be filled in by adding more objects to the hierarchy.

The complete hierarchy can be written out to an mmCIF or BinaryCIF file using the *ihm.dumper.write()* function.

1.2.8 Input

Hierarchies of IHM classes can also be read from mmCIF or BinaryCIF files. This is done using the *ihm.reader.read()* function, which returns a list of *ihm.System* objects.

1.3 Provenance

The IHM dictionary is designed to capture all aspects of integrative modeling, from the original deposited experimental data to the final validated models. This allows for maximum reproducibility and resuability. However, many modeling packages are only concerned with the conversion of their own inputs to output models (for example, a model of a complex may be generated by docking comparative models guided by some experimental data of the entire complex). If only this last step of the procedure is captured in the output mmCIF file (in this case, without any information on how the comparative models were themselves obtained) the chain is broken and the outputs cannot be reproduced.

One solution to this problem is to diligently ensure that every input to the modeling has been deposited in an appropriate database and always refer to inputs using *ihm.location.DatabaseLocation*. In cases where this is not possible, the library provides some metadata parsers in the *ihm.metadata* module. These will make a best effort to extract any metadata from files available on the local hard drive to better describe their provenance. For example, if the file contains headers or other information that shows that it is merely a copy of a file deposited in an official database, the metadata parsers will return a suitable *DatabaseLocation* for the dataset. Other information, such as the software used to generate the file, may be available in the metadata.

For more details, see *ihm.metadata.MRCParser* for electron microscopy density maps (MRC files) or *ihm.metadata.PDBParser* for coordinate files in PDB format.

1.4 Design principles

1.4.1 Lightweight

The classes in this package are designed to be lightweight, taking up as little memory as possible. For example, individual atoms are *not* stored in Python classes, and are only requested when needed. This is because the library is designed to work with an existing modeling package, which likely already stores data on the system in its own files or data structures, such that duplicating this information would be very inefficient.

1.4.2 Mutable

All classes are designed to be *mutable*; that is, their contents can be changed after creation. For example, protein chains can be added to or removed from an existing *ihm.Assembly* object, or the amino acid sequence of an *ihm.Entity* can be extended. This because some of the modeling packages which use these classes build up their own data model in a similar way.

1.4.3 Types rather than enums

Where the underlying IHM mmCIF dictionary uses an enumeration, generally this corresponds to separate sibling classes in this package. For example, two datasets which differ only in their `data_type` in the dictionary (such as a electron microscopy density map and small angle scattering data) are represented with two classes in this package: *ihm.dataset.EMDensityDataset* and *ihm.dataset.SASDataset*. This cleanly enforces the allowed types in the most Pythonic manner.

1.4.4 Hierarchy of classes

The underlying IHM mmCIF dictionary is essentially structured as a set of rows in database tables, with IDs acting as keys or pointers into other tables. This is naturally represented in Python as a hierarchy of classes, with members pointing to other objects as appropriate. IDs are not used to look up other objects, and are only used internally to populate the tables. For example, to group multiple models together, the dictionary assigns all of the models the

same model_group_id while in the Python package the *ihm.model.Model* objects are placed into a *ihm.model.ModelGroup* object, which acts like a simple Python list.

The table-based representation of the dictionary does allow for objects to exist that are not referenced by other objects, unlike the Python-based hierarchy. Such 'orphan' objects can be referenced from orphan lists in the top-level *ihm.System* if necessary.

1.4.5 Equal versus identical objects

Since the Python objects are mutable, can be constructed iteratively by a modeling package, and live in a hierarchy, it can sometimes turn out that two Python objects while not identical (they point to different locations in memory) are equal (their contents are the same). For example, the two *ihm.Assembly* objects, one of proteins A, B, and C, and the other of A, C, and B, are not identical (they are different objects) but are equal (the order of the proteins does not matter). The library will attempt to detect such objects and consolidate them on output, describing both of them in the mmCIF file with the same ID, to avoid meaningless duplication of rows in the output tables. This removes some of the burden from the author of the modeling package, which may not care about such a distinction.

1.4.6 mmCIF backend

The classes in this package roughly correspond to categories in the underlying IHM mmCIF dictionary. This allows for simple output of mmCIF formatted files, but also allows for the potential future support for other file formats that support the dictionary or a subset of it, such as MMTF.

1.5 Change history

1.5.1 0.18 - 2020-11-06

- Update to match latest FLR dictionary.
- Add a simple utility (util/make-mmcif.py) to make a minimal compliant IHM mmCIF file, given an mmCIF file (potentially just coordinates) as input.
- Bugfix: the full residue range spanned by a starting model is now reported, rather than just the subset that is mapped to one or more templates (#55).
- Bugfix: handle TrEMBL UniProt sequences (#57).

1.5.2 0.17 - 2020-07-10

- Convenience classes are added to describe hydrogen/deuterium exchange data (*ihm.dataset.HDXDataset*) and datasets stored in the PDB-Dev database (*ihm.location.PDBDevLocation*).
- Multiple *ihm.restraint.CrossLinkPseudoSite* objects can now be assigned to a given *ihm.restraint.CrossLink*.
- Bugfix: the *ihm.dataset.Dataset* base class now has a type of "Other" rather than "unspecified" to conform with the latest IHM dictionary.

1.5.3 0.16 - 2020-05-29

- *ihm.reader.read()* no longer discards models read from non-IHM mmCIF files; they are instead placed in their own *ihm.model.ModelGroup*.
- Bugfix: both the pure Python and C-accelerated mmCIF readers are now more robust, able to handle files in binary mode (e.g. from opening a URL) and in Unicode (mmCIF files are supposed to be ASCII but python-ihm should handle any encoding Python supports).

1.5.4 0.15 - 2020-04-14

- *ihm.dataset.Dataset* objects that derive from another dataset can now record any transformation involved; see *ihm.dataset.TransformedDataset*.
- *ihm.metadata.PDBParser* now extracts basic metadata from PDB files generated by SWISS-MODEL.
- An *ihm.Entity* can now be linked to one or more reference databases (e.g. UniProt). See the classes in the *ihm.reference* module.

1.5.5 0.14 - 2020-02-26

- A cross-link can now use pseudo sites to represent one or both ends of the link. The new *ihm.restraint.CrossLinkPseudoSite* object is used when the end of the cross-link is not represented in the model but its position is known (e.g. it may have been approximated given the position of nearby residues).
- *ihm.restraint.PseudoSiteFeature* now references an underlying *ihm.restraint.PseudoSite*, allowing a single pseudo site to be shared between a feature and a cross-link if desired.
- *ihm.model.Ensemble* now supports describing subsamples from which the ensemble was constructed; see *ihm.model.Subsample*.
- Bugfix: *ihm.Citation.from_pubmed_id()* now works correctly when the journal volume or page range are empty, or the page "range" is just a single page.

1.5.6 0.13 - 2019-11-14

- *ihm.reader.read()* has a new optional `reject_old_file` argument. If set, it will raise an exception if asked to read a file that conforms to too old a version of the IHM extension dictionary.
- Definitions for the DHSO and BMSO cross-linkers are now provided in the *ihm.cross_linkers* module.

1.5.7 0.12 - 2019-10-16

- *ihm.restraint.ResidueFeature* objects can now act on one or more *Residue* objects, which act equivalently to 1-residue ranges (*AsymUnitRange* or *EntityRange*).
- The new *ihm.dataset.GeneticInteractionsDataset* class and the `mic_value` argument to *ihm.restraint.DerivedDistanceRestraint* can be used to represent restraints from genetic interactions, such as point-mutant epistatic miniarray profile (pE-MAP) data.

1.5.8 0.11 - 2019-09-05

- *ihm.Assembly* objects can now only contain *AsymUnit* and *AsymUnitRange* objects (not *Entity* or *EntityRange*).
- Bugfix: ensembles that don't reference a *ihm.model.ModelGroup* no longer cause the reader to create bogus empty model groups.

1.5.9 0.10 - 2019-07-09

- Features (*ihm.restraint.AtomFeature*, *ihm.restraint.ResidueFeature*, and *ihm.restraint.NonPolyFeature*), which previously could select part or all of an *ihm.AsymUnit*, can now also select parts of an *Entity*. A restraint acting on an entity-feature is assumed to apply to all instances of that entity.

1.5.10 0.9 - 2019-05-31

- Add support for the latest version of the IHM dictionary.

1.5.11 0.8 - 2019-05-28

- *ihm.reader.read()* can now be asked to warn if it encounters categories or keywords in the mmCIF or BinaryCIF file that it doesn't know about (and will ignore).
- Predicted contacts (*ihm.restraint.PredictedContactRestraint*) are now supported.
- *ihm.reader.read()* will now read starting model coordinates and sequence difference information into the *ihm.startmodel.StartingModel* class. Applications that don't require coordinates can instruct the reader to ignore them with the new *read_starting_model_coord* flag.
- The new *ihm.flr* module allows for information from Fluorescence / FRET experiments to be stored. This follows the definitions in the FLR dictionary.

1.5.12 0.7 - 2019-04-24

- Authors of the mmCIF file itself (*_audit_author* category) can now be set by manipulating *ihm.System.authors*. (If this list is empty on output, the set of all citation authors is used instead, as before.)
- Any grants that supported the modeling can now be listed in *ihm.System.grants*.
- A copy of SWIG is no longer needed to install releases of python-ihm via *pip* as pre-generated SWIG outputs are included in the PyPI package. SWIG is still needed to build directly from source code though.

1.5.13 0.6 - 2019-03-22

- *Entity* now takes an optional *ihm.source.Source* object to describe the method by which the sample for the entity was produced. *ihm.metadata.PDBParser* will also extract this information from input PDB files.
- *ihm.reader.read()* and *ihm.dumper.write()* now support reading or writing additional user-defined mmCIF categories.

1.5.14 0.5 - 2019-01-17

- *ihm.restraint.CrossLinkRestraint* now takes an *ihm.ChemDescriptor* object rather than the name of the cross-linker used. This allows the use of novel cross-linkers (beyond those currently listed in a fixed enumeration in the IHM dictionary). *ihm.ChemDescriptor* allows for the chemical structure of the cross-linker to be uniquely specified, as a SMILES or INCHI string. The *ihm.cross_linkers* module provides chemical descriptors for some commonly-used cross-linkers.
- Pseudo sites are now supported. *ihm.restraint.PseudoSiteFeature* allows points or spheres with arbitrary coordinates to be designated as features, which can then be used in *ihm.restraint.DerivedDistanceRestraint*.

1.5.15 0.4 - 2018-12-17

- Certain restraints can now be grouped using the *ihm.restraint.RestraintGroup* class. Due to limitations of the underlying dictionary, this only works for some restraint types (currently only *ihm.restraint.DerivedDistanceRestraint*) and all restraints in the group must be of the same type.
- Bugfix: the model's representation (see *ihm.representation*) need not be a strict subset of the model's *ihm.Assembly*. However, any *ihm.model.Atom* or *ihm.model.Sphere* objects must be covered by both the representation and the model's *ihm.Assembly*.
- Bugfix: the reader no longer fails to read files that contain _entity.formula_weight.

1.5.16 0.3 - 2018-11-21

- The library now includes basic support for nonpolymers and water molecules. In addition to the previous support for polymers (amino or nucleic acid chains), *ihm.Entity* objects can now comprise ligands, water molecules, and user-defined chemical components.
- The library can now read mmCIF dictionaries and validate mmCIF or BinaryCIF files against them. See *ihm.dictionary*.
- Any *ihm.model.Atom* or *ihm.model.Sphere* objects are now checked against the model's representation (see *ihm.representation*); for example, an *ihm.model.Atom* must correspond to an *ihm.representation.AtomicSegment*. The representation in turn must be a subset of the model's *ihm.Assembly*.
- More examples are now provided, of creating and using non-standard residue types (chemical components); representing nonpolymers; and using the C mmCIF parser in other C programs.

1.5.17 0.2 - 2018-09-06

- This release should fix installation of the package using pip: *pip install ihm* should now work correctly.

1.5.18 0.1 - 2018-09-06

- First stable release. This provides largely complete support for the current version of the wwPDB IHM mmCIF extension dictionary, and will read and write mmCIF and BinaryCIF files that are compliant with the PDBx and IHM dictionaries.

API Reference:

1.6 The ihm Python module

Representation of an IHM mmCIF file as a set of Python classes.

Generally class names correspond to mmCIF table names and class attributes to mmCIF attributes (with prefixes like *pdbx_* stripped). For example, the data item _entity.details is found in the *Entity* class, as the *details* member.

Ordinals and IDs are generally not used in this representation (instead, pointers to objects are used).

ihm.**unknown = ?**
A value that isn't known. Note that this is distinct from a value that is deliberately omitted, which is represented by Python None.

class ihm.**System**(*title=None*, *id='model'*)
Top-level class representing a complete modeled system.

Parameters

- **title** (*str*) – Title (longer text description) of the system.
- **id** (*str*) – Unique identifier for this system in the mmCIF file.

asym_units = None
All asymmetric units used in the system. See *AsymUnit*.

authors = None
List of all authors of this system, as a list of strings (last name followed by initials, e.g. "Smith AJ"). When writing out a file, if this is list is empty, the set of all citation authors (see Citation.authors) is used instead.

citations = None
List of all citations. See *Citation*.

comments = None
List of plain text comments. These will be added to the top of the mmCIF file.

complete_assembly = None
The assembly of the entire system. By convention this is always the first assembly in the mmCIF file (assembly_id=1). Note that currently this isn't filled in on output until dumper.write() is called. See *Assembly*.

ensembles = None
All ensembles. See *Ensemble*.

entities = None
All entities used in the system. See *Entity*.

flr_data = None
Contains the fluorescence (FLR) part. See *FLRData*.

grants = None
List of all grants that supported this work. See *Grant*.

locations = None
Locations of all extra resources. See *Location*.

ordered_processes = None
All ordered processes. See *OrderedProcess*.

orphan_assemblies = None
All orphaned assemblies in the system. See *Assembly*. This can be used to keep track of all assemblies that are not otherwise used - normally one is assigned to a *Model*, *ihm.protocol.Step*, or *Restraint*.

orphan_chem_descriptors = None
All orphaned chemical descriptors in the system. See *ChemDescriptor*. This can be used to track descriptors that are not otherwise used - normally one is assigned to a *ihm.restraint.CrossLinkRestraint*.

orphan_dataset_groups = None
All orphaned groups of datasets. This can be used to keep track of all dataset groups that are not otherwise used - normally a group is assigned to a *Protocol*. See *DatasetGroup*.

orphan_datasets = None
All orphaned datasets. This can be used to keep track of all datasets that are not otherwise used - normally a dataset is assigned to a *DatasetGroup*, *StartingModel*, *Restraint*, *Template*, or as the parent of another *Dataset*. See *Dataset*.

orphan_features = None
All orphaned features. This can be used to keep track of all features that are not otherwise used - normally a feature is assigned to a *GeometricRestraint*. See *Feature*.

orphan_geometric_objects = None
All orphaned geometric objects. This can be used to keep track of all objects that are not otherwise used - normally an object is assigned to a *GeometricRestraint*. See *GeometricObject*.

orphan_protocols = None
All orphaned modeling protocols. This can be used to keep track of all protocols that are not otherwise used - normally a protocol is assigned to a *Model*. See *Protocol*.

orphan_pseudo_sites = None
All orphaned pseudo sites. This can be used to keep track of all pseudo sites that are not otherwise used - normally a site is used in a *PseudoSiteFeature* or a *CrossLinkPseudoSite*.

orphan_representations = None
All orphaned representations of the system. This can be used to keep track of all representations that are not otherwise used - normally one is assigned to a *Model*. See *Representation*.

orphan_starting_models = None
All orphaned starting models for the system. This can be used to keep track of all starting models that are not otherwise used - normally one is assigned to an *ihm.representation.Segment*. See *StartingModel*.

restraint_groups = None
All restraint groups. See *RestraintGroup*.

restraints = None
All restraints on the system. See *Restraint*.

software = None
List of all software used in the modeling. See *Software*.

state_groups = None
All state groups (collections of models). See *StateGroup*.

update_locations_in_repositories(*repos*)
Update all `Location` objects in the system that lie within a checked-out `Repository` to point to that repository.

This is intended for the use case where the current working directory is a checkout of a repository which is archived somewhere with a DOI. Locations can then be simply constructed pointing to local files, and retroactively updated with this method to point to the DOI if appropriate.

For each Location, if it points to a local file that is below the *root* of one of the *repos*, update it to point to that repository. If is under multiple roots, pick the one that gives the shortest path. For example, if run

in a subdirectory *foo* of a repository archived as *repo.zip*, the local path *simple.pdb* will be updated to be *repo-top/foo/simple.pdb* in *repo.zip*:

```
l = ihm.location.InputFileLocation("simple.pdb")
system.locations.append(l)

r = ihm.location.Repository(doi='1.2.3.4',
          url='https://example.com/repo.zip',)
          top_directory="repo-top", root="..")
system.update_locations_in_repositories([r])
```

class ihm.**Software**(*name*, *classification*, *description*, *location*, *type='program'*, *version=None*)

Software used as part of the modeling protocol.

Parameters

- **name** (*str*) – The name of the software.
- **classification** (*str*) – The major function of the sofware, for example 'model building', 'sample preparation', 'data collection'.
- **description** (*str*) – A longer text description of the software.
- **location** (*str*) – Place where the software can be found (e.g. URL).
- **type** (*str*) – Type of software (program/package/library/other).
- **version** (*str*) – The version used.

Generally these objects are added to *System.software* or passed to *ihm.startmodel.StartingModel*, *ihm.protocol.Step*, *ihm.analysis.Step*, or ihm.restraint.PredictedContactResstraint objects.

class ihm.**Citation**(*pmid*, *title*, *journal*, *volume*, *page_range*, *year*, *authors*, *doi*)

A publication that describes the modeling.

Generally citations are added to *System.citations* or passed to *ihm.restraint.EM3DRestraint* objects.

Parameters

- **pmid** (*str*) – The PubMed ID.
- **title** (*str*) – Full title of the publication.
- **journal** (*str*) – Abbreviated journal name.
- **volume** (*int*) – Journal volume number.
- **page_range** – The page (int) or page range (as a 2-element int tuple).
- **year** (*int*) – Year of publication.
- **authors** – All authors in order, as a list of strings (last name followed by initials, e.g. "Smith AJ").
- **doi** (*str*) – Digital Object Identifier of the publication.

classmethod from_pubmed_id(*pubmed_id*)

Create a Citation from just a PubMed ID. This is done by querying NCBI's web API, so requires network access.

Parameters **pubmed_id** (*int*) – The PubMed identifier.

Returns A new Citation for the given identifier.

Return type *Citation*

class ihm.**Grant**(*funding_organization*, *country*, *grant_number*)
Information on funding support for the modeling. See *System.grants*.

Parameters

- **funding_organization** (*str*) – The name of the organization providing the funding, e.g. "National Institutes of Health".
- **country** (*str*) – The country that hosts the funding organization, e.g. "United States".
- **grant_number** (*str*) – Identifying information for the grant, e.g. "1R01GM072999-01".

class ihm.**ChemComp**(*id*, *code*, *code_canonical*, *name=None*, *formula=None*)
A chemical component from which *Entity* objects are constructed. Usually these are amino acids (see *LPeptideChemComp*) or nucleic acids (see *DNAChemComp* and *RNAChemComp*).

For standard amino and nucleic acids, it is generally easier to use a *Alphabet* and refer to the components with their one-letter (amino acids, RNA) or two-letter (DNA) codes.

Parameters

- **id** (*str*) – A globally unique identifier for this component (usually three letters).
- **code** (*str*) – A shorter identifier (usually one letter) that only needs to be unique in the entity.
- **code_canonical** (*str*) – Canonical version of *code* (which need not be unique).
- **name** (*str*) – A longer human-readable name for the component.
- **formula** (*str*) – The chemical formula. This is a space-separated list of the element symbols in the component, each followed by an optional count (if omitted, 1 is assumed). The formula is terminated with the formal charge (if not zero). The element list should be sorted alphabetically, unless carbon is present, in which case C and H precede the rest of the elements. For example, water would be "H2 O" and arginine (with +1 formal charge) "C6 H15 N4 O2 1".

For example, glycine would have `id='GLY', code='G', code_canonical='G'` while selenomethionine would use `id='MSE', code='MSE', code_canonical='M'`, guanosine (RNA) `id='G', code='G', code_canonical='G'`, and deoxyguanosine (DNA) `id='DG', code='DG', code_canonical='G'`.

formula_weight
Formula weight (dalton). This is calculated automatically from the chemical formula and known atomic masses.

class ihm.**PeptideChemComp**(*id*, *code*, *code_canonical*, *name=None*, *formula=None*)
A single peptide component. Usually *LPeptideChemComp* is used instead (except for glycine) to specify chirality. See *ChemComp* for a description of the parameters.

class ihm.**LPeptideChemComp**(*id*, *code*, *code_canonical*, *name=None*, *formula=None*)
A single peptide component with (normal) L- chirality. See *ChemComp* for a description of the parameters.

class ihm.**DPeptideChemComp**(*id*, *code*, *code_canonical*, *name=None*, *formula=None*)
A single peptide component with (unusual) D- chirality. See *ChemComp* for a description of the parameters.

class ihm.**RNAChemComp**(*id*, *code*, *code_canonical*, *name=None*, *formula=None*)
A single RNA component. See *ChemComp* for a description of the parameters.

class ihm.**DNAChemComp**(*id*, *code*, *code_canonical*, *name=None*, *formula=None*)
A single DNA component. See *ChemComp* for a description of the parameters.

class ihm.**NonPolymerChemComp**(*id*, *name=None*, *formula=None*)
A non-polymer chemical component, such as a ligand (for crystal waters, use *WaterChemComp*).

Parameters

- **id** (*str*) – A globally unique identifier for this component.
- **name** (*str*) – A longer human-readable name for the component.
- **formula** (*str*) – The chemical formula. See *ChemComp* for more details.

class ihm.**WaterChemComp**
The chemical component for crystal water.

class ihm.**Alphabet**
A mapping from codes (usually one-letter, or two-letter for DNA) to chemical components. These classes can be used to construct sequences of components when creating an *Entity*. They can also be used like a Python dict to get standard components, e.g.:

```
a = ihm.LPeptideAlphabet()
met = a['M']
gly = a['G']
```

See *LPeptideAlphabet*, *RNAAlphabet*, *DNAAlphabet*.

class ihm.**LPeptideAlphabet**
A mapping from one-letter amino acid codes (e.g. H, M) to L-amino acids (as *LPeptideChemComp* objects, except for achiral glycine which maps to *PeptideChemComp*). Some other common modified residues are also included (e.g. MSE). For these their full name rather than a one-letter code is used.

class ihm.**DPeptideAlphabet**
A mapping from D-amino acid codes (e.g. DHI, MED) to D-amino acids (as *DPeptideChemComp* objects, except for achiral glycine which maps to *PeptideChemComp*). See *LPeptideAlphabet* for more details.

class ihm.**RNAAlphabet**
A mapping from one-letter nucleic acid codes (e.g. A) to RNA (as *RNAChemComp* objects).

class ihm.**DNAAlphabet**
A mapping from two-letter nucleic acid codes (e.g. DA) to DNA (as *DNAChemComp* objects).

class ihm.**Entity**(*sequence*, *alphabet=<class 'ihm.LPeptideAlphabet'>*, *description=None*, *details=None*, *source=None*, *references=[]*)
Represent a CIF entity (with a unique sequence)

Parameters

- **sequence** (*sequence*) – The primary sequence, as a sequence of *ChemComp* objects, and/or codes looked up in *alphabet*.
- **alphabet** (*Alphabet*) – The mapping from code to chemical components to use (it is not necessary to instantiate this class).
- **description** (*str*) – A short text name for the sequence.
- **details** (*str*) – Longer text describing the sequence.
- **source** (*ihm.source.Source*) – The method by which the sample for this entity was produced.
- **references** (sequence of *ihm.reference.Reference* objects) – Information about this entity stored in external databases (for example the sequence in UniProt)

The sequence for an entity can be specified explicitly as a list of chemical components, or (more usually) as a list or string of codes, or a mixture of both. For example:

```
# Construct with a string of one-letter amino acid codes
protein = ihm.Entity('AHMD')
# Some less common amino acids (e.g. MSE) have three-letter codes
protein_with_mse = ihm.Entity(['A', 'H', 'MSE', 'D'])

# Can use a non-default alphabet to make DNA or RNA sequences
dna = ihm.Entity(('DA', 'DC'), alphabet=ihm.DNAAlphabet)
rna = ihm.Entity('AC', alphabet=ihm.RNAAlphabet)

# Can pass explicit ChemComp objects by looking them up in Alphabets
dna_al = ihm.DNAAlphabet()
rna_al = ihm.RNAAlphabet()
dna_rna_hybrid = ihm.Entity((dna_al['DG'], rna_al['C']))

# For unusual components (e.g. modified residues or ligands),
# new ChemComp objects can be constructed
psu = ihm.RNAChemComp(id='PSU', code='PSU', code_canonical='U',
                      name="PSEUDOURIDINE-5'-MONOPHOSPHATE",
                      formula='C9 H13 N2 O9 P')
rna_with_psu = ihm.Entity(('A', 'C', psu), alphabet=ihm.RNAAlphabet)
```

For more examples, see the ligands and water example.

All entities should be stored in the top-level System object; see *System.entities*.

formula_weight
Formula weight (dalton). This is calculated automatically from that of the chemical components.

is_polymeric()
Return True iff this entity represents a polymer, such as an amino acid sequence or DNA/RNA chain (and not a ligand or water)

residue(*seq_id*)
Get a *Residue* at the given sequence position

seq_id_range
Sequence range

class ihm.**EntityRange**(*entity*, *seq_id_begin*, *seq_id_end*)
Part of an entity. Usually these objects are created from an *Entity*, e.g. to get a range covering residues 4 through 7 in *entity* use:

```
entity = ihm.Entity(sequence=...)
rng = entity(4,7)
```

class ihm.**AsymUnit**(*entity*, *details=None*, *auth_seq_id_map=0*, *id=None*)
An asymmetric unit, i.e. a unique instance of an Entity that was modeled.

Parameters

- **entity** (*Entity*) – The unique sequence of this asymmetric unit.
- **details** (*str*) – Longer text description of this unit.
- **auth_seq_id_map** – Mapping from internal 1-based consecutive residue numbering (*seq_id*) to "author-provided" numbering (*auth_seq_id*). This can be either be an int offset, in which case `auth_seq_id = seq_id + auth_seq_id_map`, or a mapping type (dict, list, tuple) in which case `auth_seq_id = auth_seq_id_map[seq_id]`. (Note that if a *list* or *tuple* is used, the first element in the list or tuple does **not** correspond

to the first residue and will never be used - since *seq_id* can never be zero.) The default if not specified, or not in the mapping, is for `auth_seq_id == seq_id`.

- **id** (*str*) – User-specified ID (usually a string of one or more upper-case letters, e.g. A, B, C, AA). If not specified, IDs are automatically assigned alphabetically.

See *System.asym_units*.

residue(*seq_id*)
Get a *Residue* at the given sequence position

seq_id_range
Sequence range

class ihm.**AsymUnitRange**(*asym*, *seq_id_begin*, *seq_id_end*)
Part of an asymmetric unit. Usually these objects are created from an *AsymUnit*, e.g. to get a range covering residues 4 through 7 in *asym* use:

```
asym = ihm.AsymUnit(entity)
rng = asym(4,7)
```

class ihm.**Atom**(*residue*, *id*)
A single atom in an entity or asymmetric unit. Usually these objects are created by calling *Residue.atom()*.

class ihm.**Residue**(*seq_id*, *entity=None*, *asym=None*)
A single residue in an entity or asymmetric unit. Usually these objects are created by calling *Entity.residue()* or *AsymUnit.residue()*.

atom(*atom_id*)
Get a *Atom* in this residue with the given name.

auth_seq_id
Author-provided seq_id; only makes sense for asymmetric units

class ihm.**Assembly**(*elements=()*, *name=None*, *description=None*)
A collection of parts of the system that were modeled or probed together.

Parameters

- **elements** (*sequence*) – Initial set of parts of the system.
- **name** (*str*) – Short text name of this assembly.
- **description** (*str*) – Longer text that describes this assembly.

This is implemented as a simple list of asymmetric units (or parts of them), i.e. a list of *AsymUnit* and/or *AsymUnitRange* objects. An Assembly is typically assigned to one or more of

- *Model*
- *ihm.protocol.Step*
- *ihm.analysis.Step*
- *Restraint*

See also *System.complete_assembly* and *System.orphan_assemblies*.

Note that any duplicate assemblies will be pruned on output.

parent = None
Assembly that is the immediate parent in a hierarchy, or *None*

class ihm.**ChemDescriptor**(*auth_name*, *chem_comp_id=None*, *chemical_name=None*, *common_name=None*, *smiles=None*, *smiles_canonical=None*, *inchi=None*, *inchi_key=None*)

Description of a non-polymeric chemical component used in the experiment. For example, this might be a fluorescent probe or cross-linking agent. This class describes the chemical structure of the component, for example with a SMILES or INCHI descriptor, so that it is uniquely defined. A descriptor is typically assigned to a *ihm.restraint.CrossLinkRestraint*.

See *ihm.cross_linkers* for chemical descriptors of some commonly-used cross-linking agents.

Parameters

- **auth_name** (*str*) – Author-provided name
- **chem_comp_id** (*str*) – If this chemical is listed in the Chemical Component Dictionary, its three-letter identifier
- **chemical_name** (*str*) – The systematic (IUPAC) chemical name
- **common_name** (*str*) – Common name for the component
- **smiles** (*str*) – SMILES string
- **smiles_canonical** (*str*) – Canonical SMILES string
- **inchi** (*str*) – IUPAC INCHI descriptor
- **inchi_key** (*str*) – Hashed INCHI key

See also *System.orphan_chem_descriptors*.

1.7 The ihm.source Python module

Classes for describing the source of an entity.

class ihm.source.**Source**

Base class to describe the source of an *ihm.Entity*. See *Manipulated*, *Natural* and *Synthetic*.

class ihm.source.**Details**(*ncbi_taxonomy_id=None*, *scientific_name=None*, *common_name=None*, *strain=None*)

Identifying information for an entity source. See *Manipulated* or *Natural*.

Parameters

- **ncbi_taxonomy_id** – NCBI taxonomy identifier, e.g. “469008”
- **scientific_name** – Scientific name, e.g. “Escherichia coli”
- **common_name** – Common name
- **strain** – Strain, e.g. “BL21(DE3)PLYSS”

class ihm.source.**Manipulated**(*gene=None*, *host=None*)

An entity isolated from a genetically manipulated source. See Entity.

Parameters

- **gene** (*Details*) – Details about the gene source.
- **host** (*Details*) – Details about the host organism.

class ihm.source.**Natural**(*ncbi_taxonomy_id=None*, *scientific_name=None*, *common_name=None*, *strain=None*)

An entity isolated from a natural source. See Entity. See *Details* for a description of the parameters.

class ihm.source.**Synthetic**(*ncbi_taxonomy_id=None*, *scientific_name=None*, *common_name=None*, *strain=None*)
An entity obtained synthetically. See `Entity`. See *Details* for a description of the parameters.

1.8 The ihm.reference Python module

Classes for providing extra information about an *ihm.Entity*

class ihm.reference.**Reference**
Base class for extra information about an *ihm.Entity*.

This class is not used directly; instead, use a subclass such as *Sequence* or *UniProtSequence*. These objects are then typically passed to the *ihm.Entity* constructor.

class ihm.reference.**Sequence**(*db_name*, *db_code*, *accession*, *sequence*, *details=None*)
Point to the sequence of an *ihm.Entity* in a sequence database; convenience subclasses are provided for common sequence databases such as *UniProtSequence*.

These objects are typically passed to the *ihm.Entity* constructor.

See also *alignments* to describe the correspondence between the database and entity sequences.

Parameters

- **db_name** (*str*) – The name of the database.
- **db_code** (*str*) – The name of the sequence in the database.
- **accession** (*str*) – The database accession.
- **sequence** (*str*) – The complete sequence, as a string of one-letter codes.
- **details** (*str*) – Longer text describing the sequence.

alignments = None
All alignments between the reference and entity sequences, as *Alignment* objects. If none are provided, a simple 1:1 alignment is assumed.

class ihm.reference.**UniProtSequence**(*db_code*, *accession*, *sequence*, *details=None*)
Point to the sequence of an *ihm.Entity* in UniProt.

These objects are typically passed to the *ihm.Entity* constructor.

Parameters

- **db_code** (*str*) – The UniProt name (e.g. NUP84_YEAST)
- **accession** (*str*) – The UniProt accession (e.g. P52891)

See *Sequence* for a description of the remaining parameters.

classmethod from_accession(*accession*)
Create *UniProtSequence* from just an accession. This is done by querying the UniProt web API, so requires network access.

Parameters **accession** (*str*) – The UniProt accession (e.g. P52891)

class ihm.reference.**Alignment**(*db_begin=1*, *db_end=None*, *entity_begin=1*, *entity_end=None*, *seq_dif=[]*)
A sequence range that aligns between the database and the entity. This describes part of the sequence in the sequence database (*Sequence*) and in the *ihm.Entity*. The two ranges must be the same length and have the same primary sequence (any differences must be described with *SeqDif* objects).

Parameters

- **db_begin** (*int*) – The first residue in the database sequence that is used (defaults to the entire sequence).
- **db_end** (*int*) – The last residue in the database sequence that is used (or None, the default, to use the entire sequence).
- **entity_begin** (*int*) – The first residue in the *Entity* sequence that is taken from the reference (defaults to the entire entity sequence).
- **entity_end** (*int*) – The last residue in the *Entity* sequence that is taken from the reference (or None, the default, to use the entire sequence).
- **seq_dif** (Sequence of *SeqDif* objects.) – Single-point mutations made to the sequence.

class ihm.reference.**SeqDif**(*seq_id*, *db_monomer*, *monomer*, *details=None*)

Annotate a sequence difference between a reference and entity sequence. See *Alignment*.

Parameters

- **seq_id** (*int*) – The residue index in the entity sequence.
- **db_monomer** (*ihm.ChemComp*) – The monomer type (as a *ChemComp* object) in the reference sequence.
- **monomer** (*ihm.ChemComp*) – The monomer type (as a *ChemComp* object) in the entity sequence.
- **details** (*str*) – Descriptive text for the sequence difference.

1.9 The ihm.location Python module

Classes for tracking external data used by mmCIF models.

class ihm.location.**Location**(*details=None*)

Identifies the location where a resource can be found.

Do not use this class itself, but one of its subclasses. Typically the resource may be found in a file (either on the local disk or at a DOI) - for this use one of the subclasses of *FileLocation*. Alternatively the resource may be found in an experiment-specific database such as PDB or EMDB - for this use *DatabaseLocation* or one of its subclasses. A Location may be passed to

- a *Dataset* to point to where an experimental dataset may be found;
- an *Ensemble* to point to coordinates for an entire ensemble, for example as a DCD file;
- a LocalizationDensity to point to an external localization density, for example in MRC format;
- *ihm.System.locations* to point to other files relating to the modeling in general, such as a modeling control script (*WorkflowFileLocation*) or a command script for a visualization package such as ChimeraX (*VisualizationFileLocation*);
- a *ihm.protocol.Step* or *ihm.analysis.Step* to describe an individual modeling step;
- or a *StartingModel* to describe how a starting model was constructed.

Parameters **details** (*str*) – Additional details about the dataset, if known.

class ihm.location.**DatabaseLocation**(*db_name*, *db_code*, *version=None*, *details=None*)

A dataset stored in an official database (PDB, EMDB, PRIDE, etc.).

Parameters

- **db_name** (*str*) – The name of the database.
- **db_code** (*str*) – The accession code inside the database.
- **version** (*str*) – The version of the dataset in the database.
- **details** (*str*) – Additional details about the dataset, if known.

class ihm.location.**EMDBLocation**(*db_code*, *version=None*, *details=None*)
Something stored in the EMDB database. See *DatabaseLocation* for a description of the parameters and *Location* for discussion of the usage of these objects.

class ihm.location.**PDBLocation**(*db_code*, *version=None*, *details=None*)
Something stored in the PDB database. See *DatabaseLocation* for a description of the parameters and *Location* for discussion of the usage of these objects.

class ihm.location.**PDBDevLocation**(*db_code*, *version=None*, *details=None*)
Something stored in the PDB-Dev database. See *DatabaseLocation* for a description of the parameters and *Location* for discussion of the usage of these objects.

class ihm.location.**BMRBLocation**(*db_code*, *version=None*, *details=None*)
Something stored in the BMRB database. See *DatabaseLocation* for a description of the parameters and *Location* for discussion of the usage of these objects.

class ihm.location.**MassIVELocation**(*db_code*, *version=None*, *details=None*)
Something stored in the MassIVE database. See *DatabaseLocation* for a description of the parameters and *Location* for discussion of the usage of these objects.

class ihm.location.**EMPIARLocation**(*db_code*, *version=None*, *details=None*)
Something stored in the EMPIAR database. See *DatabaseLocation* for a description of the parameters and *Location* for discussion of the usage of these objects.

class ihm.location.**SASBDBLocation**(*db_code*, *version=None*, *details=None*)
Something stored in the SASBDB database. See *DatabaseLocation* for a description of the parameters and *Location* for discussion of the usage of these objects.

class ihm.location.**PRIDELocation**(*db_code*, *version=None*, *details=None*)
Something stored in the PRIDE database. See *DatabaseLocation* for a description of the parameters and *Location* for discussion of the usage of these objects.

class ihm.location.**BioGRIDLocation**(*db_code*, *version=None*, *details=None*)
Something stored in the BioGRID database. See *DatabaseLocation* for a description of the parameters and *Location* for discussion of the usage of these objects.

class ihm.location.**FileLocation**(*path*, *repo=None*, *details=None*)
Base class for an individual file or directory stored externally.

Parameters

- **path** (*str*) – the location of the file or directory (this can be *None* if *repo* is set, to refer to the entire repository)
- **repo** (*Repository*) – object that describes the repository containing the file, or *None* if it is stored on the local disk
- **details** (*str*) – optional description of the file

class ihm.location.**InputFileLocation**(*path*, *repo=None*, *details=None*)
An externally stored file used as input. See *FileLocation* for a description of the parameters and *Location* for discussion of the usage of these objects.

For example, any *Dataset* that isn't stored in a domain-specific database would use this class.

class ihm.location.**OutputFileLocation**(*path*, *repo=None*, *details=None*)
An externally stored file used for output. See *FileLocation* for a description of the parameters and *Location* for discussion of the usage of these objects.

For example, this can be used to point to an externally-stored *model ensemble* or a *localization density*.

class ihm.location.**WorkflowFileLocation**(*path*, *repo=None*, *details=None*)
An externally stored file that controls the workflow (e.g. a script). See *FileLocation* for a description of the parameters and *Location* for discussion of the usage of these objects.

Typically these objects are used to provide more information on how a *StartingModel* was generated, how an individual *ihm.protocol.Step* or *ihm.analysis.Step* was performed, or to describe the overall modeling (by addition to *ihm.System.locations*). This can be useful to capture fine details of the modeling that aren't covered by the mmCIF dictionary, and to allow models to be precisely reproduced.

class ihm.location.**VisualizationFileLocation**(*path*, *repo=None*, *details=None*)
An externally stored file that is used for visualization. See *FileLocation* for a description of the parameters and *Location* for discussion of the usage of these objects.

class ihm.location.**Repository**(*doi*, *root=None*, *url=None*, *top_directory=None*, *details=None*)
A repository containing modeling files, i.e. a collection of related files at a remote, public location. This can include code repositories such as GitHub, file archival services such as Zenodo, or any other service that provides a DOI, such as the supplementary information for a publication.

This can also be used if the script plus related files are part of a repository, which has been archived somewhere with a DOI. This will be used to construct permanent references to files used in this modeling, even if they haven't been uploaded to a database such as PDB or EMDB. See *ihm.System.update_locations_in_repositories()*.

See also *FileLocation*.

Parameters

- **doi** (*str*) – the Digital Object Identifier for the repository
- **root** (*str*) – the path on the local disk to the top-level directory of the repository, or *None* if files in this repository aren't checked out.
- **url** (*str*) – If given, a location that this repository can be downloaded from.
- **top_directory** (*str*) – If given, prefix all paths for files in this repository with this value. This is useful when the archived version of the repository is found in a subdirectory at the URL or DOI (for example, GitHub repositories archived at Zenodo get placed in a subdirectory named for the repository and git hash).
- **details** (*str*) – Additional text describing this repository

1.10 The ihm.dataset Python module

Classes for handling experimental datasets used by mmCIF models.

class ihm.dataset.**Dataset**(*location*, *details=None*)
A set of input data, for example, a crystal structure or EM map.

Parameters

- **location** (*ihm.location.Location*) – a pointer to where the dataset is stored. This is usually a subclass of *DatabaseLocation* if the dataset is deposited in a database such as PDB or EMDB, or *InputFileLocation* if the dataset is stored in an external file.

- **details** (*str*) – Text giving more information about the dataset.

add_primary(*dataset*)
Add another Dataset from which this one was ultimately derived.

parents = None
A list of *Dataset* and/or *TransformedDataset* objects from which this one was derived. For example, a 3D EM map may be derived from a set of 2D images.

class ihm.dataset.**TransformedDataset**(*dataset*, *transform*)
A *Dataset* that should be rotated or translated before using. This is typically used for derived datasets (see *Dataset.parents*) where the derived dataset lies in a different dataset from the parent (for example, it was moved to better align with the model's reference frame or other experimental data). The transformation that places the derived dataset on the parent is recorded here.

Parameters

- **dataset** (*Dataset*) – The (parent) dataset.
- **transform** (*ihm.geometry.Transformation*) – The rotation and translation that places a derived dataset on this dataset.

class ihm.dataset.**DatasetGroup**(*elements=()*, *name=None*, *application=None*, *details=None*)
A set of *Dataset* objects that are handled together. This is implemented as a simple list.

Parameters

- **elements** (*sequence*) – Initial set of datasets.
- **name** (*str*) – Short text name of this group.
- **application** (*str*) – Text that shows how this group is used.
- **details** (*str*) – Longer text that describes this group.

Normally a group is passed to one or more *Protocol* or *Analysis* objects, although unused groups can still be included in the file if desired by adding them to *ihm.System.orphan_dataset_groups*.

class ihm.dataset.**CXMSDataset**(*location*, *details=None*)
Processed cross-links from a CX-MS experiment

class ihm.dataset.**MassSpecDataset**(*location*, *details=None*)
Raw mass spectrometry files such as peaklists

class ihm.dataset.**HDXDataset**(*location*, *details=None*)
Data from a hydrogen/deuterium exchange experiment

class ihm.dataset.**PDBDataset**(*location*, *details=None*)
An experimentally-determined 3D structure as a set of a coordinates, usually in a PDB file

class ihm.dataset.**ComparativeModelDataset**(*location*, *details=None*)
A 3D structure determined by comparative modeling

class ihm.dataset.**IntegrativeModelDataset**(*location*, *details=None*)
A 3D structure determined by integrative modeling

class ihm.dataset.**DeNovoModelDataset**(*location*, *details=None*)
A 3D structure determined by de novo modeling

class ihm.dataset.**NMRDataset**(*location*, *details=None*)
A nuclear magnetic resonance (NMR) dataset

class ihm.dataset.**MutagenesisDataset**(*location*, *details=None*)
Mutagenesis data

class ihm.dataset.**EMDensityDataset**(*location*, *details=None*)
A 3D electron microscopy dataset

class ihm.dataset.**EMMicrographsDataset**(*location*, *details=None*)
Raw 2D electron micrographs

class ihm.dataset.**EM2DClassDataset**(*location*, *details=None*)
2DEM class average

class ihm.dataset.**SASDataset**(*location*, *details=None*)
SAS data

class ihm.dataset.**FRETDataset**(*location*, *details=None*)
Data from a Förster resonance energy transfer (FRET) experiment

class ihm.dataset.**YeastTwoHybridDataset**(*location*, *details=None*)
Yeast two-hybrid data

class ihm.dataset.**GeneticInteractionsDataset**(*location*, *details=None*)
Quantitative measurements of genetic interactions

1.11 The ihm.metadata Python module

Classes to extract metadata from various input files.

Often input files contain metadata that would be useful to include in the mmCIF file, but the metadata is stored in a different way for each domain-specific file type. For example, MRC files used for electron microscopy maps may contain an EMDB identifier, which the mmCIF file can point to in preference to the local file.

This module provides classes for each file type to extract suitable metadata where available.

class ihm.metadata.**Parser**
Base class for all metadata parsers.

parse_file(*filename*)
Extract metadata from the given file.

Parameters **filename** (*str*) – the file to extract metadata from.

Returns a dict with extracted metadata (generally including a *Dataset*).

class ihm.metadata.**MRCParser**
Extract metadata from an EM density map (MRC file).

parse_file(*filename*)
Extract metadata. See *Parser.parse_file()* for details.

Returns a dict with key *dataset* pointing to the density map, as an EMDB entry if the file contains EMDB headers, otherwise to the file itself.

If the file turns out to be an EMDB entry, this will also query the EMDB web API (if available) to extract version information and details for the dataset.

class ihm.metadata.**PDBParser**
Extract metadata (e.g. PDB ID, comparative modeling templates) from a PDB file. This handles PDB headers added by the PDB database itself, comparative modeling packages such as MODELLER and Phyre2, and also some custom headers that can be used to indicate that a file has been locally modified in some way.

parse_file(*filename*)
Extract metadata. See *Parser.parse_file()* for details.

Parameters **filename** (*str*) – the file to extract metadata from.

Returns a dict with key *dataset* pointing to the PDB dataset; 'templates' pointing to a dict with keys the asym (chain) IDs in the PDB file and values the list of comparative model templates used to model that chain as *ihm.startmodel.Template* objects; 'entity_source' pointing to a dict with keys the asym IDs and values *ihm.source.Source* objects; 'software' pointing to a list of software used to generate the file (as *ihm.Software* objects); 'script' pointing to the script used to generate the file, if any (as *ihm.location.WorkflowFileLocation* objects); 'metadata' a list of PDB metadata records.

This parser looks at PDB headers. Standard PDB database headers are recognized, plus some added by common comparative modeling packages such as MODELLER and Phyre2, as well as some custom headers that can be used to denote that a PDB file is a locally-modified version of some other resource. Additional details will be extracted from other PDB headers if available, such as `TITLE` records.

If the first line of the file starts with `HEADER` then the file is assumed to live in the PDB database. For example, the following will be interpreted as PDB entry 2HBJ:

```
HEADER    HYDROLASE, GENE REGULATION              14-JUN-06   2HBJ
```

If the first line starts with `EXPDTA DERIVED FROM` then the file is assumed to derive from a given PDB ID or a comparative or integrative model available at a given DOI. `TITLE` records are expected to describe the nature of the transformation:

```
EXPDTA    DERIVED FROM PDB:1YKH
EXPDTA    DERIVED FROM COMPARATIVE MODEL, DOI:10.1093/nar/gkt704
EXPDTA    DERIVED FROM INTEGRATIVE MODEL, DOI:10.1016/j.str.2017.01.006
```

A first line starting with `REMARK 99 Chain ID :` is assumed to be a model generated by Phyre2. Template information can be added using Modeller-style headers, as below, if desired.

A first line starting with `EXPDTA THEORETICAL MODEL, MODELLER` is assumed to be a model generated by Modeller. Headers generated by modern versions of Modeller are parsed to extract information about the comparative modeling script, plus the templates used and their alignment. Templates named `1abcX` or `1abcX_N` are assumed to be structures deposited in PDB (in this case, chain X in structure 1ABC). A custom `TEMPLATE PATH` header can be used to point to templates that are not deposited in the PDB database. For example, the model below is assumed to be constructed using templates from PDB codes 3JRO and 3F3F, plus another template in `my_custom_pdb_file.pdb`, and the given alignment:

```
EXPDTA    THEORETICAL MODEL, MODELLER 9.18 2017/02/10 22:21:34
REMARK   6 ALIGNMENT: modeller_model.ali
REMARK   6 SCRIPT: model-default.py
REMARK   6 TEMPLATE PATH custom1 ../inputs/my_custom_pdb_file.pdb
REMARK   6 TEMPLATE: 3jroC 33:C - 424:C MODELS 33:A - 424:A AT 100.0%
REMARK   6 TEMPLATE: 3f3fG 482:G - 551:G MODELS 429:A - 488:A AT 10.0%
REMARK   6 TEMPLATE: custom1 9:A - 352:A MODELS 80:A - 414:A AT 32.0%
```

A first line starting with `TITLE SWISS-MODEL SERVER` is assumed to be a model generated by SWISS-MODEL, and information about the template(s) is extracted from `REMARK 3` records.

1.12 The `ihm.startmodel` Python module

Classes to handle starting models.

class ihm.startmodel.**SequenceIdentityDenominator**

The denominator used while calculating the sequence identity. One of these constants can be passed to *SequenceIdentity*.

MEAN_LENGTH = 4
Arithmetic mean sequence length

NUM_ALIGNED_WITHOUT_GAPS = 3
Number of aligned residue pairs (not including the gaps)

NUM_ALIGNED_WITH_GAPS = 2
Number of aligned positions (including gaps)

OTHER = 5
Another method not covered here

SHORTER_LENGTH = 1
Length of the shorter sequence

class ihm.startmodel.**SequenceIdentity**(*value*, *denominator=<SequenceIdentityDenominator.SHORTER_LENGTH: 1>*)
Describe the identity between template and target sequences. See *Template*.

Parameters

- **value** – Percentage sequence identity.
- **denominator** – Way in which sequence identity was calculated - see *SequenceIdentityDenominator*.

class ihm.startmodel.**Template**(*dataset*, *asym_id*, *seq_id_range*, *template_seq_id_range*, *sequence_identity*, *alignment_file=None*)
A PDB file used as a comparative modeling template for part of a starting model.

See *StartingModel*.

Parameters

- **dataset** (*Dataset*) – Pointer to where this template is stored.
- **asym_id** (*str*) – The asymmetric unit (chain) to use from the template dataset (not necessarily the same as the starting model's asym_id or the ID of the asym_unit in the final IHM model).
- **seq_id_range** (*tuple*) – The sequence range in the dataset that is modeled by this template. Note that this numbering may differ from the IHM numbering. See *offset* in *StartingModel*.
- **template_seq_id_range** (*tuple*) – The sequence range of the template that is used in comparative modeling.
- **sequence_identity** (*SequenceIdentity* or *float*) – Sequence identity between template and the target sequence.
- **alignment_file** (*Location*) – Reference to the external file containing the template-target alignment.

class ihm.startmodel.**StartingModel**(*asym_unit*, *dataset*, *asym_id*, *templates=None*, *offset=0*, *metadata=None*, *software=None*, *script_file=None*, *description=None*)
A starting guess for modeling of an asymmetric unit

See *ihm.representation.Segment* and *ihm.System.orphan_starting_models*.

Parameters

- **asym_unit** (*AsymUnit* or *AsymUnitRange*) – The asymmetric unit (or part of one) this starting model represents.

- **dataset** (*Dataset*) – Pointer to where this model is stored.
- **asym_id** (*str*) – The asymmetric unit (chain) to use from the starting model's dataset (not necessarily the same as the ID of the asym_unit in the final model).
- **templates** (*list*) – A list of *Template* objects, if this is a comparative model.
- **offset** (*int*) – Offset between the residue numbering in the dataset and the IHM model (the offset is added to the starting model numbering to give the IHM model numbering).
- **metadata** (*list*) – List of PDB metadata, such as HELIX records.
- **software** (*Software*) – The software used to generate the starting model.
- **script_file** (*Location*) – Reference to the external file containing the script used to generate the starting model (usually a *WorkflowFileLocation*).
- **description** (*str*) – Additional text describing the starting model.

add_atom(*atom*)
Add to the model's set of *Atom* objects.

See *get_atoms()* for more details.

add_seq_dif(*seq_dif*)
Add to the model's set of *SeqDif* objects.

See *get_atoms()* for more details.

get_atoms()
Yield *Atom* objects that represent this starting model. This allows the starting model coordinates to be embedded in the mmCIF file, which is useful if the starting model is not available elsewhere (or it has been modified).

The default implementation returns an internal list of atoms; it is usually necessary to subclass and override this method. See *ihm.model.Model.get_spheres()* for more details.

Note that the returned atoms should be those used in modeling, not those stored in the file. In particular, the numbering scheme should be that used in the IHM model (add *offset* to the dataset numbering). If any residues were changed (for example it is common to mutate MSE in the dataset to MET in the modeling) the final mutated name should be used (MET in this case) and *get_seq_dif()* overridden to note the change.

get_seq_dif()
Yield *SeqDif* objects for any sequence changes between the dataset and the starting model. See *get_atoms()*.

The default implementation returns an internal list of objects; it is usually necessary to subclass and override this method.

Note that this is always called *after* *get_atoms()*.

class ihm.startmodel.**PDBHelix**(*line*)
Represent a HELIX record from a PDB file.

class ihm.startmodel.**SeqDif**(*db_seq_id*, *seq_id*, *db_comp_id*, *details=None*)
Annotate a sequence difference between a dataset and starting model. See *StartingModel.get_seq_dif()* and *MSESeqDif*.

Parameters

- **db_seq_id** (*int*) – The residue index in the dataset.
- **seq_id** (*int*) – The residue index in the starting model. This should normally be *db_seq_id* + *offset*.

- **db_comp_id** (*str*) – The name of the residue in the dataset.
- **details** (*str*) – Descriptive text for the sequence difference.

class ihm.startmodel.**MSESeqDif**(*db_seq_id*, *seq_id*, *details='Conversion of modified residue MSE to MET'*)

Denote that a residue was mutated from MSE to MET. See *SeqDif* for a description of the parameters.

1.13 The ihm.representation Python module

Classes for handling representation of the system during modeling.

class ihm.representation.**Segment**

Base class for part of a *Representation*. See *AtomicSegment*, *ResidueSegment*, *MultiResidueSegment*, and *FeatureSegment*.

class ihm.representation.**AtomicSegment**(*asym_unit*, *rigid*, *starting_model=None*, *description=None*)

Part of the system modeled atomistically, stored in a *Representation*.

Parameters

- **asym_unit** (*AsymUnit* or *AsymUnitRange*) – The asymmetric unit (or part of one) that this segment represents.
- **rigid** (*bool*) – Whether internal coordinates of the segment were fixed during modeling.
- **starting_model** (*StartingModel*) – initial coordinates used for the segment (or None).
- **description** (*str*) – Additional text describing this segment.

class ihm.representation.**ResidueSegment**(*asym_unit*, *rigid*, *primitive*, *starting_model=None*, *description=None*)

Part of the system modeled as a set of residues, stored in a *Representation*.

Parameters

- **asym_unit** (*AsymUnit* or *AsymUnitRange*) – The asymmetric unit (or part of one) that this segment represents.
- **rigid** (*bool*) – Whether internal coordinates of the segment were fixed during modeling.
- **primitive** (*str*) – The type of object used to represent this segment (sphere/gaussian/other).
- **starting_model** (*StartingModel*) – initial coordinates used for the segment (or None).
- **description** (*str*) – Additional text describing this segment.

class ihm.representation.**MultiResidueSegment**(*asym_unit*, *rigid*, *primitive*, *starting_model=None*, *description=None*)

Part of the system modeled as a single object representing a range of residues, stored in a *Representation*.

Parameters

- **asym_unit** (*AsymUnit* or *AsymUnitRange*) – The asymmetric unit (or part of one) that this segment represents.
- **rigid** (*bool*) – Whether internal coordinates of the segment were fixed during modeling.
- **primitive** (*str*) – The type of object used to represent this segment (sphere/gaussian/other).

- **starting_model** (*StartingModel*) – initial coordinates used for the segment (or None).
- **description** (*str*) – Additional text describing this segment.

class ihm.representation.**FeatureSegment**(*asym_unit*, *rigid*, *primitive*, *count*, *starting_model=None*, *description=None*)

Part of the system modeled as a number of geometric features, stored in a *Representation*.

Parameters

- **asym_unit** (*AsymUnit* or *AsymUnitRange*) – The asymmetric unit (or part of one) that this segment represents.
- **rigid** (*bool*) – Whether internal coordinates of the segment were fixed during modeling.
- **primitive** (*str*) – The type of object used to represent this segment (sphere/gaussian/other).
- **count** (*int*) – The number of objects used to represent this segment.
- **starting_model** (*StartingModel*) – initial coordinates used for the segment (or None).
- **description** (*str*) – Additional text describing this segment.

class ihm.representation.**Representation**(*elements=()*, *name=None*, *details=None*)

Part of the system modeled as a set of geometric objects, such as spheres or atoms. This is implemented as a simple list of *Segment* objects.

Parameters

- **elements** (*sequence*) – Initial set of segments.
- **name** (*str*) – A short descriptive name.
- **details** (*str*) – A longer description of the representation.

Typically a Representation is assigned to a *Model*. See also *ihm.System.orphan_representations*.

Multiple representations of the same system are possible (multi-scale).

1.14 The ihm.geometry Python module

Classes for handling geometry.

Geometric objects (see *GeometricObject*) are usually used in *GeometricRestraint* objects.

class ihm.geometry.**Center**(*x*, *y*, *z*)

Define the center of a geometric object in Cartesian space.

Parameters

- **x** (*float*) – x coordinate
- **y** (*float*) – y coordinate
- **z** (*float*) – z coordinate

class ihm.geometry.**Transformation**(*rot_matrix*, *tr_vector*)

Rotation and translation applied to an object.

Transformation objects are typically used in subclasses of *GeometricObject*, or by *ihm.dataset.TransformedDataset*.

Parameters

- **rot_matrix** – Rotation matrix (as a 3x3 array of floats) that places the object in its final position.
- **tr_vector** – Translation vector (as a 3-element float list) that places the object in its final position.

class ihm.geometry.**GeometricObject**(*name=None*, *description=None*)

A generic geometric object. See also *Sphere*, *Torus*, *Axis*, *Plane*.

Geometric objects are typically assigned to one or more *GeometricRestraint* objects.

Parameters

- **name** (*str*) – A short user-provided name.
- **description** (*str*) – A brief description of the object.

class ihm.geometry.**Sphere**(*center*, *radius*, *transformation=None*, *name=None*, *description=None*)

A sphere in Cartesian space.

Parameters

- **center** (*Center*) – Coordinates of the center of the sphere.
- **radius** – Radius of the sphere.
- **transformation** (*Transformation*) – Rotation and translation that moves the sphere from the original center to its final location, if any.
- **name** (*str*) – A short user-provided name.
- **description** (*str*) – A brief description of the object.

class ihm.geometry.**Torus**(*center*, *major_radius*, *minor_radius*, *transformation=None*, *name=None*, *description=None*)

A torus in Cartesian space.

Parameters

- **center** (*Center*) – Coordinates of the center of the torus.
- **major_radius** – The major radius - the distance from the center of the tube to the center of the torus.
- **minor_radius** – The minor radius - the radius of the tube.
- **transformation** (*Transformation*) – Rotation and translation that moves the torus (which by default lies in the xy plane) from the original center to its final location, if any.
- **name** (*str*) – A short user-provided name.
- **description** (*str*) – A brief description of the object.

class ihm.geometry.**HalfTorus**(*center*, *major_radius*, *minor_radius*, *thickness*, *transformation=None*, *inner=None*, *name=None*, *description=None*)

A section of a *Torus*. This is defined as a surface over part of the torus with a given thickness, and is often used to represent a membrane.

Parameters

- **thickness** – The thickness of the surface.
- **inner** – True if the surface is the 'inner' half of the torus (i.e. closer to the center), False for the outer surface, or None for some other section (described in *description*).

See *Torus* for a description of the other parameters.

class ihm.geometry.**Axis**(*transformation=None*, *name=None*, *description=None*)
One of the three Cartesian axes - see *XAxis*, *YAxis*, *ZAxis*.

Parameters

- **transformation** (*Transformation*) – Rotation and translation that moves the axis from the original Cartesian axis to its final location, if any.
- **name** (*str*) – A short user-provided name.
- **description** (*str*) – A brief description of the object.

class ihm.geometry.**XAxis**(*transformation=None*, *name=None*, *description=None*)
The x Cartesian axis.

See *GeometricObject* for a description of the parameters.

class ihm.geometry.**YAxis**(*transformation=None*, *name=None*, *description=None*)
The y Cartesian axis.

See *GeometricObject* for a description of the parameters.

class ihm.geometry.**ZAxis**(*transformation=None*, *name=None*, *description=None*)
The z Cartesian axis.

See *GeometricObject* for a description of the parameters.

class ihm.geometry.**Plane**(*transformation=None*, *name=None*, *description=None*)
A plane in Cartesian space - see *XYPlane*, *YZPlane*, *XZPlane*.

Parameters

- **transformation** (*Transformation*) – Rotation and translation that moves the plane from the original position to its final location, if any.
- **name** (*str*) – A short user-provided name.
- **description** (*str*) – A brief description of the object.

class ihm.geometry.**XYPlane**(*transformation=None*, *name=None*, *description=None*)
The xy plane in Cartesian space.

See *GeometricObject* for a description of the parameters.

class ihm.geometry.**YZPlane**(*transformation=None*, *name=None*, *description=None*)
The yz plane in Cartesian space.

See *GeometricObject* for a description of the parameters.

class ihm.geometry.**XZPlane**(*transformation=None*, *name=None*, *description=None*)
The xz plane in Cartesian space.

See *GeometricObject* for a description of the parameters.

1.15 The ihm.restraint Python module

Classes for handling restraints on the system.

class ihm.restraint.**PseudoSite**(*x*, *y*, *z*, *radius=None*, *description=None*)
Selection of a pseudo position in the system. Pseudo positions are typically used to reference a point or sphere that is not explcitly represented, in a *PseudoSiteFeature* or *CrossLinkPseudoSite*.

Parameters

- **x** (*float*) – Cartesian X coordinate of this site.
- **y** (*float*) – Cartesian Y coordinate of this site.
- **z** (*float*) – Cartesian Z coordinate of this site.
- **radius** (*float*) – Radius of the site, if applicable.
- **description** (*str*) – Additional text describing this feature.

class ihm.restraint.**Restraint**
Base class for all restraints. See *ihm.System.restraints*.

class ihm.restraint.**RestraintGroup**
A set of related *Restraint* objects. This is implemented as a simple list.

Note that due to limitations of the underlying dictionary, only certain combinations of restraints can be placed in groups. In particular, all objects in a group must be of the same type, and only certain types (currently only *DerivedDistanceRestraint* and *PredictedContactRestraint*) can be grouped.

Empty groups can be created, but will be ignored on output as the dictionary does not support them.

Restraint groups should be stored in the system by adding them to *ihm.System.restraint_groups*.

class ihm.restraint.**EM3DRestraint**(*dataset*, *assembly*, *segment=None*, *fitting_method=None*, *fitting_method_citation=None*, *number_of_gaussians=None*, *details=None*)
Restrain part of the system to match an electron microscopy density map.

Parameters

- **dataset** (*Dataset*) – Reference to the density map data (usually an *EMDensityDataset*).
- **assembly** (*Assembly*) – The part of the system that is fit into the map.
- **segment** (*bool*) – True iff the map has been segmented.
- **fitting_method** (*str*) – The method used to fit the model into the map.
- **fitting_method_citation** (*Citation*) – The publication describing the fitting method.
- **number_of_gaussians** (*int*) – Number of Gaussians used to represent the map as a Gaussian Mixture Model (GMM), if applicable.
- **details** (*str*) – Additional details regarding the fitting.

fits = None
Information about the fit of each model to this restraint's data. This is a Python dict where keys are *Model* objects and values are *EM3DRestraintFit* objects.

class ihm.restraint.**EM3DRestraintFit**(*cross_correlation_coefficient=None*)
Information on the fit of a model to an *EM3DRestraint*. See EM3DRestaint.fits.

Parameters **cross_correlation_coefficient** (*float*) – The fit between the model and the map.

class ihm.restraint.**SASRestraint**(*dataset*, *assembly*, *segment=None*, *fitting_method=None*, *fitting_atom_type=None*, *multi_state=None*, *radius_of_gyration=None*, *details=None*)
Restrain part of the system to match small angle scattering (SAS) data.

Parameters

- **dataset** (*Dataset*) – Reference to the SAS data (usually an *SASDataset*).

- **assembly** (*Assembly*) – The part of the system that is fit against SAS data.
- **segment** (*bool*) – True iff the SAS profile has been segmented.
- **fitting_method** (*str*) – The method used to fit the model against the SAS data (e.g. FoXS, DAMMIF).
- **fitting_atom_type** (*str*) – The set of atoms fit against the data (e.g. "Heavy atoms", "All atoms").
- **multi_state** (*bool*) – Whether multiple state fitting was done.
- **radius_of_gyration** (*float*) – Radius of gyration obtained from the SAS profile, if used as part of the restraint.
- **details** (*str*) – Additional details regarding the fitting.

fits = None
Information about the fit of each model to this restraint's data. This is a Python dict where keys are *Model* objects and values are *SASRestraintFit* objects.

class ihm.restraint.**SASRestraintFit**(*chi_value=None*)
Information on the fit of a model to a *SASRestraint*. See SASRestaint.fits.

Parameters **chi_value** (*float*) – The fit between the model and the SAS data.

class ihm.restraint.**DistanceRestraint**
Abstract base class for all distance restraints. These are typically used in a *DerivedDistanceRestraint*.

Do not use this class directly but instead use a derived class such as *HarmonicDistanceRestraint*, *UpperBoundDistanceRestraint*, *LowerBoundDistanceRestraint*, or *LowerUpperBoundDistanceRestraint*.

class ihm.restraint.**HarmonicDistanceRestraint**(*distance*)
Harmonically restrain two objects to be close to a given distance apart. These objects are typically used in a *DerivedDistanceRestraint*.

Parameters **distance** (*float*) – Equilibrium distance

class ihm.restraint.**UpperBoundDistanceRestraint**(*distance*)
Harmonically restrain two objects to be below a given distance apart. These objects are typically used in a *DerivedDistanceRestraint*.

Parameters **distance** (*float*) – Distance threshold

class ihm.restraint.**LowerBoundDistanceRestraint**(*distance*)
Harmonically restrain two objects to be above a given distance apart. These objects are typically used in a *DerivedDistanceRestraint*.

Parameters **distance** (*float*) – Distance threshold

class ihm.restraint.**LowerUpperBoundDistanceRestraint**(*distance_lower_limit*, *distance_upper_limit*)
Harmonically restrain two objects to be above a given distance and below another distance apart. These objects are typically used in a *DerivedDistanceRestraint*.

Parameters

- **distance_lower_limit** (*float*) – Lower bound on the distance.
- **distance_upper_limit** (*float*) – Upper bound on the distance.

class ihm.restraint.**CrossLinkRestraint**(*dataset*, *linker*)
Restrain part of the system to match a set of cross-links.

Parameters

- **dataset** (*Dataset*) – Reference to the cross-link data (usually a *CXMSDataset*).
- **linker** (*ihm.ChemDescriptor*) – The type of chemical linker used.

cross_links = None
All cross-links used in the modeling, as a list of *CrossLink* objects.

experimental_cross_links = None
All cross-links identified in the experiment, as a simple list of lists of *ExperimentalCrossLink* objects. All cross-links in the same sublist are treated as experimentally ambiguous. For example, xl2 and xl3 here are considered ambiguous:

```
restraint.experimental_cross_links.append([xl1])
restraint.experimental_cross_links.append([xl2, xl3])
```

class ihm.restraint.**ExperimentalCrossLink**(*residue1*, *residue2*, *details=None*)
A cross-link identified in the experiment.

Parameters

- **residue1** (*ihm.Residue*) – The first residue linked by the cross-link.
- **residue2** (*ihm.Residue*) – The second residue linked by the cross-link.
- **details** (*str*) – Additional text describing the cross-link.

class ihm.restraint.**CrossLinkPseudoSite**(*site*, *model=None*)
Pseudo site corresponding to one end of a cross-link.

These objects are used when the end of a cross-link is not represented in the model but its position is known (e.g. it may have been approximated given the position of nearby residues). They are passed as the `pseudo1` or `pseudo2` arguments to *CrossLink* subclasses.

Parameters

- **site** (*PseudoSite*) – The pseudo site coordinates
- **model** (*ihm.model.Model*) – The model in whose coordinate system the pseudo site is active (if not specified, the coordinates are assumed to be valid for all models using this cross-link).

class ihm.restraint.**CrossLink**
Base class for all cross-links used in the modeling. Do not use this class directly, but instead use a subclass: *ResidueCrossLink*, *AtomCrossLink*, or *FeatureCrossLink*.

class ihm.restraint.**ResidueCrossLink**(*experimental_cross_link*, *asym1*, *asym2*, *distance*, *psi=None*, *sigma1=None*, *sigma2=None*, *restrain_all=None*, *pseudo1=None*, *pseudo2=None*)
A cross-link used in the modeling, applied to residue alpha carbon atoms.

Parameters

- **experimental_cross_link** (*ExperimentalCrossLink*) – The corresponding cross-link identified by experiment. Multiple cross-links can map to a single experimental identification.
- **asym1** (*ihm.AsymUnit*) – The asymmetric unit containing the first linked residue.
- **asym2** (*ihm.AsymUnit*) – The asymmetric unit containing the second linked residue.
- **distance** (*DistanceRestraint*) – Restraint on the distance.
- **psi** (*float*) – Initial uncertainty in the experimental data.
- **sigma1** (*float*) – Initial uncertainty in the position of the first residue.

- **sigma2** (*float*) – Initial uncertainty in the position of the second residue.
- **restrain_all** (*bool*) – If True, all cross-links are restrained.
- **pseudo1** (List of *CrossLinkPseudoSite*) – List of pseudo sites representing the position of the first residue (if applicable).
- **pseudo2** (List of *CrossLinkPseudoSite*) – List of pseudo sites representing the position of the second residue (if applicable).

fits = None
Information about the fit of each model to this cross-link This is a Python dict where keys are *Model* objects and values are *CrossLinkFit* objects.

class ihm.restraint.**FeatureCrossLink**(*experimental_cross_link*, *asym1*, *asym2*, *distance*, *psi=None*, *sigma1=None*, *sigma2=None*, *restrain_all=None*, *pseudo1=None*, *pseudo2=None*)
A cross-link used in the modeling, applied to the closest primitive object with the highest resolution.

Parameters

- **experimental_cross_link** (*ExperimentalCrossLink*) – The corresponding cross-link identified by experiment. Multiple cross-links can map to a single experimental identification.
- **asym1** (*ihm.AsymUnit*) – The asymmetric unit containing the first linked residue.
- **asym2** (*ihm.AsymUnit*) – The asymmetric unit containing the second linked residue.
- **distance** (*DistanceRestraint*) – Restraint on the distance.
- **psi** (*float*) – Initial uncertainty in the experimental data.
- **sigma1** (*float*) – Initial uncertainty in the position of the first residue.
- **sigma2** (*float*) – Initial uncertainty in the position of the second residue.
- **restrain_all** (*bool*) – If True, all cross-links are restrained.
- **pseudo1** (List of *CrossLinkPseudoSite*) – List of pseudo sites representing the position of the first residue (if applicable).
- **pseudo2** (List of *CrossLinkPseudoSite*) – List of pseudo sites representing the position of the second residue (if applicable).

fits = None
Information about the fit of each model to this cross-link This is a Python dict where keys are *Model* objects and values are *CrossLinkFit* objects.

class ihm.restraint.**AtomCrossLink**(*experimental_cross_link*, *asym1*, *asym2*, *atom1*, *atom2*, *distance*, *psi=None*, *sigma1=None*, *sigma2=None*, *restrain_all=None*, *pseudo1=None*, *pseudo2=None*)
A cross-link used in the modeling, applied to the specified atoms.

Parameters

- **experimental_cross_link** (*ExperimentalCrossLink*) – The corresponding cross-link identified by experiment. Multiple cross-links can map to a single experimental identification.
- **asym1** (*ihm.AsymUnit*) – The asymmetric unit containing the first linked residue.
- **asym2** (*ihm.AsymUnit*) – The asymmetric unit containing the second linked residue.
- **atom1** (*str*) – The name of the first linked atom.

- **atom2** (*str*) – The name of the second linked atom.
- **distance** (*DistanceRestraint*) – Restraint on the distance.
- **psi** (*float*) – Initial uncertainty in the experimental data.
- **sigma1** (*float*) – Initial uncertainty in the position of the first residue.
- **sigma2** (*float*) – Initial uncertainty in the position of the second residue.
- **restrain_all** (*bool*) – If True, all cross-links are restrained.
- **pseudo1** (List of *CrossLinkPseudoSite*) – List of pseudo sites representing the position of the first residue (if applicable).
- **pseudo2** (List of *CrossLinkPseudoSite*) – List of pseudo sites representing the position of the second residue (if applicable).

fits = None
Information about the fit of each model to this cross-link This is a Python dict where keys are *Model* objects and values are *CrossLinkFit* objects.

class ihm.restraint.**CrossLinkFit**(*psi=None*, *sigma1=None*, *sigma2=None*)
Information on the fit of a model to a *CrossLink*. See *ResidueCrossLink.fits*, *AtomCrossLink.fits*, or *FeatureCrossLink.fits*.

Parameters

- **psi** (*float*) – Uncertainty in the experimental data.
- **sigma1** (*float*) – Uncertainty in the position of the first residue.
- **sigma2** (*float*) – Uncertainty in the position of the second residue.

class ihm.restraint.**Feature**
Base class for selecting parts of the system that a restraint acts on. See *ResidueFeature*, *AtomFeature*, *NonPolyFeature*, and *PseudoSiteFeature*.

Features are typically assigned to one or more *GeometricRestraint* or *DerivedDistanceRestraint* objects.

class ihm.restraint.**ResidueFeature**(*ranges*, *details=None*)
Selection of one or more residues from the system.

Residues can be selected from both `AsymUnit` and `Entity` (the latter implies that it selects residues in all instances of that entity). Individual residues can also be selected by passing `Residue` objects.

Parameters

- **ranges** (*sequence*) – A list of `AsymUnitRange`, `AsymUnit`, `EntityRange`, `Residue`, and/or `Entity` objects.
- **details** (*str*) – Additional text describing this feature.

class ihm.restraint.**AtomFeature**(*atoms*, *details=None*)
Selection of one or more atoms from the system. Atoms can be selected from polymers or non-polymers (but not both). Atoms can also be selected from both `AsymUnit` and `Entity` (the latter implies that it selects atoms in all instances of that entity). For selecting an entire polymer or residue(s), see *ResidueFeature*. For selecting an entire non-polymer, see *NonPolyFeature*.

Parameters

- **atoms** (*sequence*) – A list of *ihm.Atom* objects.
- **details** (*str*) – Additional text describing this feature.

class ihm.restraint.**NonPolyFeature**(*objs*, *details=None*)

Selection of one or more non-polymers from the system. To select individual atoms from a non-polymer, see *AtomFeature*.

Features can include both `AsymUnit` and `Entity` (the latter implies that it selects non-polymers in all instances of that entity).

Parameters

- **objs** (*sequence*) – A list of `AsymUnit` and/or `Entity` objects.
- **details** (*str*) – Additional text describing this feature.

class ihm.restraint.**PseudoSiteFeature**(*site*)

Selection of a pseudo position in the system.

Parameters **site** (*PseudoSite*) – The pseudo site to use for the feature.

class ihm.restraint.**GeometricRestraint**(*dataset*, *geometric_object*, *feature*, *distance*, *harmonic_force_constant=None*, *restrain_all=None*, *pseudo1=None*, *pseudo2=None*)

A restraint between part of the system and some part of a geometric object. See *CenterGeometricRestraint*, *InnerSurfaceGeometricRestraint*, *OuterSurfaceGeometricRestraint*.

Parameters

- **dataset** (*Dataset*) – Reference to the data from which the restraint is derived.
- **geometric_object** (*ihm.geometry.GeometricObject*) – The geometric object to restrain against.
- **feature** (*Feature*) – The part of the system to restrain.
- **distance** (*DistanceRestraint*) – Restraint on the distance.
- **harmonic_force_constant** (*float*) – Force constant, if applicable.
- **restrain_all** (*bool*) – If True, all distances are restrained.

class ihm.restraint.**CenterGeometricRestraint**(*dataset*, *geometric_object*, *feature*, *distance*, *harmonic_force_constant=None*, *restrain_all=None*, *pseudo1=None*, *pseudo2=None*)

A restraint between part of the system and the center of a geometric object. See *GeometricRestraint* for a description of the parameters.

class ihm.restraint.**InnerSurfaceGeometricRestraint**(*dataset*, *geometric_object*, *feature*, *distance*, *harmonic_force_constant=None*, *restrain_all=None*, *pseudo1=None*, *pseudo2=None*)

A restraint between part of the system and the inner surface of a geometric object. See *GeometricRestraint* for a description of the parameters.

class ihm.restraint.**OuterSurfaceGeometricRestraint**(*dataset*, *geometric_object*, *feature*, *distance*, *harmonic_force_constant=None*, *restrain_all=None*, *pseudo1=None*, *pseudo2=None*)

A restraint between part of the system and the outer surface of a geometric object. See *GeometricRestraint* for a description of the parameters.

class ihm.restraint.**DerivedDistanceRestraint**(*dataset*, *feature1*, *feature2*, *distance*, *probability=None*, *restrain_all=None*, *mic_value=None*)

A restraint between two parts of the system, derived from experimental data.

Parameters

- **dataset** (*Dataset*) – Reference to the data from which the restraint is derived.
- **feature1** (*Feature*) – The first part of the system to restrain.
- **feature2** (*Feature*) – The second part of the system to restrain.
- **distance** (*DistanceRestraint*) – Restraint on the distance.
- **probability** (*float*) – Likelihood that restraint is correct (0. - 1.)
- **restrain_all** (*bool*) – If True, all distances are restrained.
- **mic_value** (*float*) – Value of the Maximal Information Coefficient (MIC) for this interaction, if applicable.

class ihm.restraint.**PredictedContactRestraint**(*dataset*, *resatom1*, *resatom2*, *distance*, *by_residue*, *probability=None*, *software=None*)

A predicted contact between two parts of the system, derived from various computational tools.

Parameters

- **dataset** (*Dataset*) – Reference to the data from which the restraint is derived.
- **resatom1** (*ihm.Residue* or *ihm.Atom*) – The first residue or atom to restrain.
- **resatom2** (*ihm.Residue* or *ihm.Atom*) – The second residue or atom to restrain.
- **distance** (*DistanceRestraint*) – Restraint on the distance.
- **by_residue** (*bool*) – If True, the restraint is applied to specific residues; otherwise, it is applied to the closest primitive object with the highest resolution.
- **probability** (*float*) – Likelihood that restraint is correct (0. - 1.)
- **software** (*Software*) – The software used to generate the contact.

1.16 The ihm.cross_linkers Python module

Chemical descriptors of commonly-used cross-linkers.

Each of these is an instance of the *ihm.ChemDescriptor* class, and so can be used anywhere these objects are required, generally for *ihm.restraint.CrossLinkRestraint*.

ihm.cross_linkers.**dss**
DSS cross-linker that links a primary amine with another primary amine (non-water-soluble).

ihm.cross_linkers.**dsg**
DSG cross-linker that links a primary amine with another primary amine (non-water-soluble).

ihm.cross_linkers.**bs3**
BS3 cross-linker that links a primary amine with another primary amine (water-soluble).

ihm.cross_linkers.**dsso**
DSSO cross-linker that links a primary amine with another primary amine (non-water-soluble). It is similar to DSS but can be cleaved in the gas phase using collision-induced dissociation.

ihm.cross_linkers.**edc**
EDC cross-linker that links a carboxyl group with a primary amine.

ihm.cross_linkers.**dhso**
DHSO (dihydrazide sulfoxide) MS-cleavable cross-linker that links carboxyl groups, described in Gutierrez et al, 2016.

ihm.cross_linkers.**bmso**
BMSO (bismaleimide sulfoxide) MS-cleavable cross-linker that links cysteines, described in Gutierrez et al, 2018.

ihm.cross_linkers.**sda**
SDA (NHS-Diazirine) (succinimidyl 4,4-azipentanoate) cross-linker that links primary amines with nearly any other functional group via long-wave UV-light activation.

1.17 The `ihm.protocol` Python module

Classes for handling modeling protocols.

class ihm.protocol.**Step**(*assembly*, *dataset_group*, *method*, *num_models_begin=None*, *num_models_end=None*, *software=None*, *script_file=None*, *multi_scale=False*, *multi_state=False*, *ordered=False*, *name=None*, *description=None*)
A single step in a *Protocol*.

Parameters

- **assembly** (*Assembly*) – The part of the system modeled in this step
- **dataset_group** (*DatasetGroup*) – The collection of datasets used in this modeling
- **method** (*str*) – Description of the method used (e.g. “Monte Carlo”)
- **name** (*str*) – A descriptive name for the step
- **num_models_begin** (*int*) – The number of models at the beginning of the step
- **num_models_end** (*int*) – The number of models at the end of the step
- **software** (*Software*) – The software used in this step
- **script_file** (*Location*) – Reference to the external file containing the script used in this step (usually a *WorkflowFileLocation*).
- **multi_scale** (*bool*) – Indicates if the modeling is multi-scale
- **multi_state** (*bool*) – Indicates if the modeling is multi-state
- **ordered** (*bool*) – Indicates if the modeling is ordered
- **description** (*str*) – Additional text describing the step

class ihm.protocol.**Protocol**(*name=None*)
A modeling protocol. Each protocol consists of a number of protocol steps (e.g. sampling, refinement) followed by a number of analyses.

Normally a protocol is passed to one or more *Model* objects, although unused protocols can still be included in the file if desired by adding them to *orphan_protocols*.

analyses = None
All analyses (*Analysis* objects)

steps = None
All modeling steps (*Step* objects)

1.18 The ihm.analysis Python module

Classes for handling the analysis of a modeling run.

class ihm.analysis.**Step**(*feature*, *num_models_begin*, *num_models_end*, *assembly=None*, *dataset_group=None*, *software=None*, *script_file=None*, *details=None*)
A single step in an *Analysis*.

Normally one of the base classes is used; see *FilterStep*, *ClusterStep*, *RescoreStep*, *ValidationStep*, OtherStep, and *EmptyStep*.

Parameters

- **feature** (*str*) – feature energy/score;RMSD;dRMSD;other
- **num_models_begin** (*int*) – The number of models at the beginning of the step
- **num_models_end** (*int*) – The number of models at the end of the step
- **assembly** (*Assembly*) – The part of the system analyzed in this step
- **dataset_group** (*DatasetGroup*) – The collection of datasets used in this analysis, if applicable
- **software** (*Software*) – The software used in this step
- **script_file** (*Location*) – Reference to the external file containing the script used in this step (usually a *WorkflowFileLocation*).
- **details** (*str*) – Additional text describing this step

class ihm.analysis.**FilterStep**(*feature*, *num_models_begin*, *num_models_end*, *assembly=None*, *dataset_group=None*, *software=None*, *script_file=None*, *details=None*)
A single filtering step in an *Analysis*. See *Step* for a description of the parameters.

class ihm.analysis.**ClusterStep**(*feature*, *num_models_begin*, *num_models_end*, *assembly=None*, *dataset_group=None*, *software=None*, *script_file=None*, *details=None*)
A single clustering step in an *Analysis*. See *Step* for a description of the parameters.

class ihm.analysis.**RescoreStep**(*feature*, *num_models_begin*, *num_models_end*, *assembly=None*, *dataset_group=None*, *software=None*, *script_file=None*, *details=None*)
A single rescoring step in an *Analysis*. See *Step* for a description of the parameters.

class ihm.analysis.**ValidationStep**(*feature*, *num_models_begin*, *num_models_end*, *assembly=None*, *dataset_group=None*, *software=None*, *script_file=None*, *details=None*)
A single validation step in an *Analysis*. See *Step* for a description of the parameters.

class ihm.analysis.**EmptyStep**
A 'do nothing' step in an *Analysis*. This can be used if modeling outputs were used directly without any kind of analysis.

class ihm.analysis.**Analysis**
Analysis of a modeling run. Each analysis consists of a number of steps (e.g. filtering, rescoring, clustering). A modeling run may be followed by any number of separate analyses.

See *ihm.protocol.Protocol.analyses*.

steps = None
All analysis steps (*Step* objects)

1.19 The ihm.model Python module

Classes for handling models (sets of coordinates) as well as groups of models.

class ihm.model.**Sphere**(*asym_unit*, *seq_id_range*, *x*, *y*, *z*, *radius*, *rmsf=None*)
Coordinates of part of the model represented by a sphere.

Parameters

- **asym_unit** (*ihm.AsymUnit*) – The asymmetric unit that this sphere represents
- **seq_id_range** (*tuple*) – The range of residues represented by this sphere (as a two-element tuple)
- **x** (*float*) – x coordinate of the center of the sphere
- **y** (*float*) – y coordinate of the center of the sphere
- **z** (*float*) – z coordinate of the center of the sphere
- **radius** (*float*) – radius of the sphere
- **rmsf** (*float*) – root-mean-square fluctuation of the coordinates

class ihm.model.**Atom**(*asym_unit*, *seq_id*, *atom_id*, *type_symbol*, *x*, *y*, *z*, *het=False*, *biso=None*, *occupancy=None*)
Coordinates of part of the model represented by an atom.

Parameters

- **asym_unit** (*ihm.AsymUnit*) – The asymmetric unit that this sphere represents
- **seq_id** (*int*) – The residue index represented by this atom (can be None for HETATM sites)
- **atom_id** (*str*) – The name of the atom in the residue
- **type_symbol** (*str*) – Element name
- **x** (*float*) – x coordinate of the atom
- **y** (*float*) – y coordinate of the atom
- **z** (*float*) – z coordinate of the atom
- **het** (*bool*) – True for HETATM sites, False (default) for ATOM
- **biso** (*float*) – Temperature factor or equivalent (if applicable)
- **occupancy** (*float*) – Fraction of the atom type present (if applicable)

class ihm.model.**Model**(*assembly*, *protocol*, *representation*, *name=None*)
A single set of coordinates (conformation).

Models are added to the system by placing them inside *ModelGroup* objects, which in turn are placed inside *State* objects, which are grouped in *StateGroup* objects, which are finally added to the system via *ihm.System.state_groups*.

Parameters

- **assembly** (*Assembly*) – The parts of the system that were modeled.
- **protocol** (*Protocol*) – Description of how the modeling was done.
- **representation** (*Representation*) – Level of detail at which the system was represented.
- **name** (*str*) – Descriptive name for this model.

add_atom(*atom*)
Add to the model's set of *Atom* objects.

See *get_spheres()* for more details.

add_sphere(*sphere*)
Add to the model's set of *Sphere* objects.

See *get_spheres()* for more details.

get_atoms()
Yield *Atom* objects that represent this model.

See *get_spheres()* for more details.

get_spheres()
Yield *Sphere* objects that represent this model.

The default implementation simply iterates over an internal list of spheres, but this is not very memory-efficient, particularly if the spheres are already stored somewhere else, e.g. in the software's own data structures. It is recommended to subclass and provide a more efficient implementation. For example, the modeling of Nup133 uses a custom subclass to pass BioPython objects through to python-ihm.

Note that the set of spheres should match the model's *Representation*. This is not currently enforced.

class ihm.model.**ModelGroup**(*elements=()*, *name=None*)
A set of related models. See *Model*. It is implemented as a simple list of the models.

These objects are typically stored in a *State*, *Ensemble*, or *OrderedProcess*.

Parameters

- **elements** – Initial set of models in the group.
- **name** (*str*) – Descriptive name for the group.

class ihm.model.**State**(*elements=()*, *type=None*, *name=None*, *details=None*, *experiment_type=None*, *population_fraction=None*)
A set of model groups that constitute a single state of the system. It is implemented as a simple list of the model groups. See *StateGroup*.

Parameters **elements** – The initial set of *ModelGroup* objects in this state.

class ihm.model.**StateGroup**(*elements=()*)
A set of related states. See *State* and *ihm.System.state_groups*. It is implemented as a simple list of the states.

Parameters **elements** – Initial set of states in the group.

class ihm.model.**Ensemble**(*model_group*, *num_models*, *post_process=None*, *clustering_method=None*, *clustering_feature=None*, *name=None*, *precision=None*, *file=None*, *details=None*)
Details about a model cluster or ensemble. See *ihm.System.ensembles*.

Parameters

- **model_group** (*ModelGroup*) – The set of models in this ensemble.

- **num_models** (*int*) – The total number of models in this ensemble. This may be more than the number of models in *model_group*, for example if only representative or top-scoring models are deposited.
- **post_process** (*ihm.analysis.Step*) – The final analysis step that generated this ensemble.
- **clustering_method** (*str*) – The method used to obtain the ensemble, if applicable.
- **clustering_feature** (*str*) – The feature used for clustering the models, if applicable.
- **name** (*str*) – A descriptive name for this ensemble.
- **precision** (*float*) – The precision of the entire ensemble.
- **file** (*ihm.location.OutputFileLocation*) – A reference to an external file containing coordinates for the entire ensemble, for example as a DCD file (see *DCDWriter*). See also *subsamples*.
- **details** (*str*) – Additional text describing this ensemble

densities = None
All localization densities for this ensemble, as *LocalizationDensity* objects

num_models_deposited
Number of models in this ensemble that are in the mmCIF file

subsamples = None
All subsamples that make up this ensemble (if applicable), as *Subsample* objects

class ihm.model.**OrderedProcess**(*ordered_by*, *description=None*)
Details about a process that orders two or more model groups.

A process is represented as a directed graph, where the nodes are *ModelGroup* objects and the edges represent transitions.

These objects are generally added to *ihm.System.ordered_processes*.

Parameters

- **ordered_by** (*str*) – Text that explains how the ordering is done, such as "time steps".
- **description** (*str*) – Text that describes this process.

steps = None
All steps in this process, as a simple list of *ProcessStep* objects

class ihm.model.**ProcessStep**(*elements=()*, *description=None*)
A single step in an *OrderedProcess*.

This is implemented as a simple list of *ProcessEdge* objects, each of which orders two *ModelGroup* objects. (To order more than two groups, for example to represent a branched reaction step that generates two products, simply add multiple edges to the step.)

Parameters

- **elements** (*sequence*) – Initial set of *ProcessEdge* objects.
- **description** (*str*) – Text that describes this step.

class ihm.model.**ProcessEdge**(*group_begin*, *group_end*, *description=None*)
A single directed edge in the graph for a *OrderedProcess*, representing the transition from one *ModelGroup* to another. These objects are added to *ProcessStep* objects.

Parameters

- **group_begin** (*ModelGroup*) – The set of models at the origin of the edge.
- **group_end** (*ModelGroup*) – The set of models at the end of the edge.
- **description** (*str*) – Text that describes this edge.

class ihm.model.**LocalizationDensity**(*file*, *asym_unit*)
Localization density of part of the system, over all models in an ensemble.

See *Ensemble.densities*.

Parameters

- **file** (*ihm.location.OutputFileLocation*) – A reference to an external file containing the density, for example as an MRC file.
- **asym_unit** (*AsymUnit* or *AsymUnitRange*) – The asymmetric unit (or part of one) that this density represents.

class ihm.model.**DCDWriter**(*fh*)
Utility class to write model coordinates to a binary DCD file.

See *Ensemble* and *Model*. Since mmCIF is a text-based format, it is not efficient to store entire ensembles in this format. Instead, representative models should be deposited as mmCIF and the *Ensemble* then linked to an external file containing only model coordinates. One such format is CHARMM/NAMD's DCD, which is written out by this class. The DCD files simply contain the xyz coordinates of all *Atom* and *Sphere* objects in each *Model*. (Note that no other data is stored, such as sphere radii or restraint parameters.)

Parameters **fh** (*file*) – The filelike object to write the coordinates to. This should be open in binary mode and should be a seekable object.

add_model(*model*)
Add the coordinates for the given *Model* to the file as a new frame. All models in the file should have the same number of atoms and/or spheres, in the same order.

Parameters **model** (*Model*) – Model with coordinates to write to the file.

class ihm.model.**Subsample**(*name*, *num_models*, *model_group=None*, *file=None*)
Base class for a subsample within an ensemble.

In some cases the models that make up an *Ensemble* may be partitioned into subsamples, for example to determine if the sampling was exhaustive (see Viswanath et al. 2017). This base class can be used to describe the set of models in the subsample, for example by pointing to an externally-deposited set of conformations.

Usually a derived class (*RandomSubsample* or *IndependentSubsample*) is used instead of this class. Instances are stored in *Ensemble.subsamples*. All of the subsamples in a given ensemble must be of the same type.

Parameters

- **name** (*str*) – A descriptive name for this sample
- **num_models** (*int*) – The total number of models in this sample
- **model_group** (*ModelGroup*) – The set of models in this sample, if applicable.
- **file** (*ihm.location.OutputFileLocation*) – A reference to an external file containing coordinates for the entire sample, for example as a DCD file (see *DCDWriter*).

num_models_deposited
Number of models in this subsample that are in the mmCIF file

class ihm.model.**RandomSubsample**(*name*, *num_models*, *model_group=None*, *file=None*)
A subsample generated by picking a random subset of the models that make up the entire ensemble. See *Subsample*.

class ihm.model.**IndependentSubsample**(*name*, *num_models*, *model_group=None*, *file=None*)
A subsample generated in the same fashion as other subsamples but by an independent simulation. See *Subsample*.

1.20 The ihm.format Python module

Utility classes to handle CIF format.

This module provides classes to read in and write out mmCIF files. It is only concerned with handling syntactically correct CIF - it does not know the set of tables or the mapping to ihm objects. For that, see *ihm.dumper* for writing and *ihm.reader* for reading.

class ihm.format.**CifWriter**(*fh*)
Write information to a CIF file. The constructor takes a single argument - a Python filelike object to write to - and provides methods to write Python objects to that file. Most simple Python types are supported (string, float, bool, int). The Python bool type is mapped to CIF strings 'NO' and 'YES'. Floats are always represented with 3 decimal places; if a different amount of precision is desired, convert the float to a string first.

category(*category*)
Return a context manager to write a CIF category. A CIF category is a simple list of key:value pairs.

Parameters **category** (*str*) – the name of the category (e.g. "_struct_conf_type").

Returns an object with a single method *write* which takes keyword arguments.

For example:

```
with writer.category("_struct_conf_type") as l:
    l.write(id='HELX_P', criteria=writer.unknown)
```

loop(*category*, *keys*)
Return a context manager to write a CIF loop.

Parameters

- **category** (*str*) – the name of the category (e.g. "_struct_conf")
- **keys** (*list*) – the field keys in that category

Returns an object with a single method *write* which takes keyword arguments; this can be called any number of times to add entries to the loop. Any field keys in *keys* that are not provided as arguments to *write*, or values that are the Python value *None*, will get the CIF omitted value ('.'), while arguments to *write* that are not present in *keys* will be ignored.

For example:

```
with writer.loop("_struct_conf", ["id", "conf_type_id"]) as l:
    for i in range(5):
        l.write(id='HELX_P1%d' % i, conf_type_id='HELX_P')
```

start_block(*name*)
Start a new data block in the file with the given name.

write_comment(*comment*)
Write a simple comment to the CIF file. The comment will be wrapped if necessary for readability.

class ihm.format.**CifReader**(*fh*, *category_handler*, *unknown_category_handler=None*, *unknown_keyword_handler=None*)
Class to read an mmCIF file and extract some or all of its data.

Use *read_file()* to actually read the file.

Parameters

- **fh** (*file*) – Open handle to the mmCIF file
- **category_handler** (*dict*) – A dict to handle data extracted from the file. Keys are category names (e.g. "_entry") and values are objects that have a *__call__* method. The names of the arguments to this *__call__* method are mmCIF keywords that are extracted from the file (for the keywords tr_vector[N] and rot_matrix[N][M] simply omit the [and] characters, since these are not valid for Python identifiers). The object will be called with the data from the file as a set of strings, or None for any keyword that is not present in the file or is the mmCIF omitted value (.). (mmCIF keywords are case insensitive, so this class always treats them as lowercase regardless of the file contents.)
- **unknown_category_handler** – A callable (or *None*) that is called for each category in the file that isn't handled; it is given two arguments: the name of the category, and the line in the file at which the category was encountered (if known, otherwise None).
- **unknown_keyword_handler** – A callable (or *None*) that is called for each keyword in the file that isn't handled (within a category that is handled); it is given three arguments: the names of the category and keyword, and the line in the file at which the keyword was encountered (if known, otherwise None).

read_file()

Read the file and extract data. Category handlers will be called as data becomes available - for `loop_` constructs, this will be once for each row in the loop; for categories (e.g. `_entry.id model`), this will be once at the very end of the file.

If the C-accelerated _format module is available, then it is used instead of the (much slower) Python tokenizer.

CifParserError will be raised if the file cannot be parsed.

Returns True iff more data blocks are available to be read.

exception ihm.format.**CifParserError**

Exception raised for invalid format mmCIF files

1.21 The ihm.format_bcif Python module

Utility classes to handle BinaryCIF format.

See https://github.com/dsehnal/BinaryCIF for a description of the BinaryCIF file format.

This module provides classes to read in and write out BinaryCIF files. It is only concerned with handling syntactically correct BinaryCIF - it does not know the set of tables or the mapping to ihm objects. For that, see *ihm.reader*.

class ihm.format_bcif.**BinaryCifWriter**(*fh*)

Write information to a BinaryCIF file. See *ihm.format.CifWriter* for more information. The constructor takes a single argument - a Python filelike object, open for writing in binary mode.

category(*category*)

See *ihm.format.CifWriter.category()*.

loop(*category*, *keys*)

See *ihm.format.CifWriter.loop()*.

start_block(*name*)

See *ihm.format.CifWriter.start_block()*.

write_comment(*comment*)
See *ihm.format.CifWriter.write_comment()*. @note BinaryCIF does not support comments, so this is a noop

class ihm.format_bcif.**BinaryCifReader**(*fh*, *category_handler*, *unknown_category_handler=None*, *unknown_keyword_handler=None*)
Class to read a BinaryCIF file and extract some or all of its data.

Use *read_file()* to actually read the file. See *ihm.format.CifReader* for a description of the parameters.

read_file()
Read the file and extract data. :return: True iff more data blocks are available to be read.

1.22 The ihm.dumper Python module

Utility classes to dump out information in mmCIF or BinaryCIF format

class ihm.dumper.**Dumper**
Base class for helpers to dump output to mmCIF or BinaryCIF. See *write()*.

dump(*system*, *writer*)
Use *writer* to write information about *system* to mmCIF or BinaryCIF.

Parameters

- **system** (*ihm.System*) – The *ihm.System* object containing all information about the system.
- **writer** (*ihm.format.CifWriter* or *ihm.format_bcif.BinaryCifWriter*.) – Utility class to write data to the output file.

finalize(*system*)
Called for all dumpers prior to *dump()*. This can be used to assign numeric IDs to objects, check for sanity, etc.

ihm.dumper.**write**(*fh*, *systems*, *format='mmCIF'*, *dumpers=[]*)
Write out all *systems* to the file handle *fh*. Files can be written in either the text-based mmCIF format or the BinaryCIF format. The BinaryCIF writer needs the msgpack Python module to function.

Parameters

- **fh** (*file*) – The file handle to write to.
- **systems** (*list*) – The list of *ihm.System* objects to write.
- **format** (*str*) – The format of the file. This can be 'mmCIF' (the default) for the (text-based) mmCIF format or 'BCIF' for BinaryCIF.
- **dumpers** (*list*) – A list of *Dumper* classes (not objects). These can be used to add extra categories to the file.

1.23 The ihm.reader Python module

Utility classes to read in information in mmCIF or BinaryCIF format

ihm.reader.**read**(*fh*, *model_class=<class 'ihm.model.Model'>*, *format='mmCIF'*, *handlers=[]*, *warn_unknown_category=False*, *warn_unknown_keyword=False*, *read_starting_model_coord=True*, *starting_model_class=<class 'ihm.startmodel.StartingModel'>*, *reject_old_file=False*)

Read data from the file handle *fh*.

Note that the reader currently expects to see a file compliant with the PDBx and/or IHM dictionaries. It is not particularly tolerant of noncompliant or incomplete files, and will probably throw an exception rather than warning about and trying to handle such files. Please open an issue if you encounter such a problem.

Files can be read in either the text-based mmCIF format or the BinaryCIF format. The mmCIF reader works by breaking the file into tokens, and using this stream of tokens to populate Python data structures. Two tokenizers are available: a pure Python implementation and a C-accelerated version. The C-accelerated version is much faster and so is used if built. The BinaryCIF reader needs the msgpack Python module to function.

Parameters

- **fh** (*file*) – The file handle to read from. (For BinaryCIF files, the file should be opened in binary mode. For mmCIF files, files opened in binary mode with Python 3 will be treated as if they are Latin-1-encoded.)
- **model_class** – The class to use to store model information (such as coordinates). For use with other software, it is recommended to subclass *ihm.model.Model* and override *add_sphere()* and/or *add_atom()*, and provide that subclass here. See *ihm.model.Model.get_spheres()* for more information.
- **format** (*str*) – The format of the file. This can be 'mmCIF' (the default) for the (text-based) mmCIF format or 'BCIF' for BinaryCIF.
- **handlers** (*list*) – A list of *Handler* classes (not objects). These can be used to read extra categories from the file.
- **warn_unknown_category** (*bool*) – if set, emit an *UnknownCategoryWarning* for each unknown category encountered in the file.
- **warn_unknown_keyword** (*bool*) – if set, emit an *UnknownKeywordWarning* for each unknown keyword (within an otherwise-handled category) encountered in the file.
- **read_starting_model_coord** (*bool*) – if set, read coordinates for starting models, if provided in the file.
- **starting_model_class** – The class to use to store starting model information. If *read_starting_model_coord* is also set, it is recommended to subclass *ihm.startmodel.StartingModel* and override *add_atom()* and/or *add_seq_dif()*.
- **reject_old_file** (*bool*) – If True, raise an *ihm.reader.OldFileError* if the file conforms to an older version of the dictionary than this library supports (by default the library will read what it can from the file).

Returns A list of *ihm.System* objects.

exception ihm.reader.**UnknownCategoryWarning**

Warning for unknown categories encountered in the file by *read()*

exception ihm.reader.**UnknownKeywordWarning**

Warning for unknown keywords encountered in the file by *read()*

exception ihm.reader.**OldFileError**

Exception raised if a file conforms to too old a version of the IHM extension dictionary. See *read()*.

class ihm.reader.**Handler**(*sysr*)

Base class for all handlers of mmCIF data. Each class handles a single category in the mmCIF or BinaryCIF

file. To add a new handler (for example to handle a custom category) make a subclass and set the class attribute *category* to the mmCIF category name (e.g. *_struct*). Provide a *__call__* method. This will be called for each category (multiple times for loop constructs) with the parameters to *__call__* filled in with the same-named mmCIF keywords. For example the class:

```
class CustomHandler(Handler):
    category = "_custom"
    def __call__(self, key1, key2):
        pass
```

will be called with arguments *"x", "y"* when given the mmCIF input:

```
_custom.key1 x
_custom.key2 y
```

Note that the arguments will always be strings when reading an mmCIF file. To convert to integer, floating point, or boolean, use the utility methods *get_int()*, *get_float()* or *get_bool()* respectively.

copy_if_present(*obj*, *data*, *keys=[]*, *mapkeys={}*)
Set obj.x from data['x'] for each x in keys if present in data. The dict mapkeys is handled similarly except that its keys are looked up in data and the corresponding value used to set obj.

end_save_frame()
Called at the end of each save frame.

finalize()
Called at the end of each data block.

get_bool(*val*)
Convert val to bool and return, or leave as is if None or ihm.unknown

get_float(*val*)
Return float(val) or leave as is if None or ihm.unknown

get_int(*val*)
Return int(val) or leave as is if None or ihm.unknown

get_int_or_string(*val*)
Return val as an int or str as appropriate, or leave as is if None or ihm.unknown

get_lower(*val*)
Return lowercase string val or leave as is if None or ihm.unknown

ignored_keywords = []
Keywords which are explicitly ignored (read() will not warn about their presence in the file). These are usually things like ordinal fields which we don't use.

not_in_file = None
Value passed to *__call__* for keywords not in the file

omitted = None
Value passed to *__call__* for data marked as omitted ('.') in the file

sysr = None
Utility class to map IDs to Python objects.

system
The *ihm.System* object to read into

unknown = ?
Value passed to *__call__* for data marked as unknown ('?') in the file

class ihm.reader.**SystemReader**(*model_class*, *starting_model_class*)

Utility class to track global information for a *ihm.System* being read from a file, such as the mapping from IDs to objects (as *IDMapper* objects). This can be used by *Handler* subclasses.

alignments = None
Mapping from ID to *ihm.reference.Alignment* objects

analyses = None
Mapping from ID to *ihm.analysis.Analysis* objects

analysis_steps = None
Mapping from ID to *ihm.analysis.Step* objects

assemblies = None
Mapping from ID to *ihm.Assembly* objects

asym_units = None
Mapping from ID to *ihm.AsymUnit* objects

centers = None
Mapping from ID to *ihm.geometry.Center* objects

chem_comps = None
Mapping from ID to *ihm.ChemComp* objects

chem_descriptors = None
Mapping from ID to *ihm.ChemDescriptor* objects

citations = None
Mapping from ID to *ihm.Citation* objects

cross_link_pseudo_sites = None
Mapping from ID to *ihm.restraint.CrossLinkPseudoSite*

cross_links = None
Mapping from ID to *ihm.restraint.CrossLink*

data_transformations = None
Mapping from ID to *ihm.geometry.Transformation* objects used by *ihm.dataset.TransformedDataset* objects (this is distinct from *transformations* since they are stored in separate tables, with different IDs, in the mmCIF file).

dataset_groups = None
Mapping from ID to *ihm.dataset.DatasetGroup* objects

datasets = None
Mapping from ID to *ihm.dataset.Dataset* objects

db_locations = None
Mapping from ID to *ihm.location.DatabaseLocation* objects

densities = None
Mapping from ID to *ihm.model.LocalizationDensity* objects

dist_restraint_groups = None
Mapping from ID to *ihm.restraint.RestraintGroup* of *ihm.restraint.DerivedDistanceRestraint* objects

dist_restraints = None
Mapping from ID to *ihm.restraint.DerivedDistanceRestraint* objects

em2d_restraints = None
Mapping from ID to ihm.restraint.EM2DRestraint objects

em3d_restraints = None
Mapping from ID to *ihm.restraint.EM3DRestraint* objects

ensembles = None
Mapping from ID to *ihm.model.Ensemble* objects

entities = None
Mapping from ID to *ihm.Entity* objects

experimental_xl_groups = None
Mapping from ID to groups of *ihm.restraint.ExperimentalCrossLink* objects

experimental_xls = None
Mapping from ID to *ihm.restraint.ExperimentalCrossLink* objects

external_files = None
Mapping from ID to *ihm.location.FileLocation* objects

features = None
Mapping from ID to *ihm.restraint.Feature* objects

flr_data = None
Mapping from ID to *ihm.flr.FLRData* objects

flr_entity_assemblies = None
Mapping from ID to *ihm.flr.EntityAssembly* objects

flr_exp_conditions = None
Mapping from ID to *ihm.flr.ExpCondition* objects

flr_experiments = None
Mapping from ID to *ihm.flr.Experiment* objects

flr_fps_av_modeling = None
Mapping from ID to *ihm.flr.FPSAVModeling* objects

flr_fps_av_parameters = None
Mapping from ID to *ihm.flr.FPSAVParameter* objects

flr_fps_global_parameters = None
Mapping from ID to *ihm.flr.FPSGlobalParameters* objects

flr_fps_mean_probe_positions = None
Mapping from ID to *ihm.flr.FPSMeanProbePosition* objects

flr_fps_modeling = None
Mapping from ID to *ihm.flr.FPSModeling* objects

flr_fps_mpp_atom_position_groups = None
Mapping from ID to *ihm.flr.FPSMPPAtomPositionGroup* objects

flr_fps_mpp_atom_positions = None
Mapping from ID to *ihm.flr.FPSMPPAtomPosition* objects

flr_fps_mpp_modeling = None
Mapping from ID to *ihm.flr.FPSMPPModeling* objects

flr_fret_analyses = None
Mapping from ID to *ihm.flr.FRETAnalysis* objects

flr_fret_calibration_parameters = None
Mapping from ID to *ihm.flr.FRETCalibrationParameters* objects

flr_fret_distance_restraint_groups = None
Mapping from ID to *ihm.flr.FRETDistanceRestraintGroup* objects

flr_fret_distance_restraints = None
Mapping from ID to *ihm.flr.FRETDistanceRestraint* objects

flr_fret_forster_radius = None
Mapping from ID to *ihm.flr.FRETForsterRadius* objects

flr_fret_model_distances = None
Mapping from ID to *ihm.flr.FRETModelDistance* objects

flr_fret_model_qualities = None
Mapping from ID to *ihm.flr.FRETModelQuality* objects

flr_inst_settings = None
Mapping from ID to *ihm.flr.InstSetting* objects

flr_instruments = None
Mapping from ID to *ihm.flr.Instrument* objects

flr_lifetime_fit_models = None
Mapping from ID to *ihm.flr.LifetimeFitModel* objects

flr_peak_assignments = None
Mapping from ID to *ihm.flr.PeakAssignment* objects

flr_poly_probe_conjugates = None
Mapping from ID to *ihm.flr.PolyProbeConjugate* objects

flr_poly_probe_positions = None
Mapping from ID to *ihm.flr.PolyProbePosition* objects

flr_probes = None
Mapping from ID to *ihm.flr.Probe* objects

flr_ref_measurement_groups = None
Mapping from ID to *ihm.flr.RefMeasurementGroup* objects

flr_ref_measurement_lifetimes = None
Mapping from ID to *ihm.flr.RefMeasurementLifetime* objects

flr_ref_measurements = None
Mapping from ID to *ihm.flr.RefMeasurement* objects

flr_sample_conditions = None
Mapping from ID to *ihm.flr.SampleCondition* objects

flr_sample_probe_details = None
Mapping from ID to *ihm.flr.SampleProbeDetails* objects

flr_samples = None
Mapping from ID to *ihm.flr.Sample* objects

geom_restraints = None
Mapping from ID to *ihm.restraint.GeometricRestraint* objects

geometries = None
Mapping from ID to *ihm.geometry.GeometricObject* objects

model_groups = None
Mapping from ID to *ihm.model.ModelGroup* objects

models = None
Mapping from ID to *ihm.model.Model* objects

ordered_procs = None
Mapping from ID to *ihm.model.OrderedProcess* objects

ordered_steps = None
Mapping from ID to *ihm.model.ProcessStep* objects

pred_cont_restraint_groups = None
Mapping from ID to *ihm.restraint.RestraintGroup* of *ihm.restraint.PredictedContactRestraint* objects

pred_cont_restraints = None
Mapping from ID to *ihm.restraint.PredictedContactRestraint* objects

protocols = None
Mapping from ID to *ihm.protocol.Protocol* objects

pseudo_sites = None
Mapping from ID to *ihm.restraint.PseudoSite* objects

ranges = None
Mapping from ID to *ihm.AsymUnitRange* or `EntityRange` objects

references = None
Mapping from ID to *ihm.reference.Reference* objects

repos = None
Mapping from ID to *ihm.location.Repository* objects

representations = None
Mapping from ID to *ihm.representation.Representation* objects

sas_restraints = None
Mapping from ID to *ihm.restraint.SASRestraint* objects

software = None
Mapping from ID to *ihm.Software* objects

src_gens = None
Mapping from ID to *ihm.source.Manipulated* objects

src_nats = None
Mapping from ID to *ihm.source.Natural* objects

src_syns = None
Mapping from ID to *ihm.source.Synthetic* objects

starting_models = None
Mapping from ID to *ihm.startmodel.StartingModel* objects

state_groups = None
Mapping from ID to *ihm.model.StateGroup* objects

states = None
Mapping from ID to *ihm.model.State* objects

system = None
The *ihm.System* object being read in

transformations = None
Mapping from ID to *ihm.geometry.Transformation* objects

xl_restraints = None
Mapping from ID to *ihm.restraint.CrossLinkRestraint* objects

class ihm.reader.**IDMapper**(*system_list*, *cls*, **cls_args*, ***cls_keys*)
Utility class to handle mapping from mmCIF IDs to Python objects.

Parameters

- **system_list** (*list*) – The list in *ihm.System* that keeps track of these objects.
- **cls** (*class*) – The base class for the Python objects.

get_all()
Yield all objects seen so far (unordered)

get_by_id(*objid*, *newcls=None*)
Get the object with given ID, creating it if it doesn't already exist. If *newcls* is specified, the object will be an instance of that class (this is commonly used when different subclasses are employed depending on a type specified in the mmCIF file, such as the various subclasses of *ihm.dataset.Dataset*).

get_by_id_or_none(*objid*, *newcls=None*)
Get the object with given ID, creating it if it doesn't already exist. If ID is None or ihm.unknown, return None instead.

class ihm.reader.**RangeIDMapper**
Utility class to handle mapping from mmCIF IDs to *ihm.AsymUnitRange* or EntityRange objects.

get(*asym_or_entity*, *range_id*)
Get a range from an ID.

Parameters

- **asym_or_entity** – An *ihm.Entity* or *ihm.AsymUnit* object representing the part of the system to which the range will be applied.
- **range_id** (*str*) – mmCIF ID

Returns A range as a *ihm.Entity*, *ihm.AsymUnit*, *ihm.EntityRange* or *ihm.AsymUnitRange* object.

set(*range_id*, *seq_id_begin*, *seq_id_end*)
Add a range.

Parameters

- **range_id** (*str*) – mmCIF ID
- **seq_id_begin** (*int*) – Index of the start of the range
- **seq_id_end** (*int*) – Index of the end of the range

1.24 The ihm.dictionary Python module

Classes to read in and represent an mmCIF extension dictionary

class ihm.dictionary.**Dictionary**
Representation of an mmCIF dictionary. See *read()* to create a Dictionary from a file.

Multiple Dictionaries can be added together to yield a Dictionary that includes all the data in the original Dictionaries.

See the validator example for an example of using this class.

categories = None
Mapping from name to *Category* objects

linked_items = None
Links between items; keys are children, values are parents e.g. `linked_items['_ihm_starting_model_details.asym_id'] = '_struct_asym.id'`

validate(*fh*, *format='mmCIF'*)
Validate the given file against this dictionary.

Parameters

- **fh** (*file*) – The file handle to read from.
- **format** (*str*) – The format of the file. This can be 'mmCIF' (the default) for the (text-based) mmCIF format or 'BCIF' for BinaryCIF.

Raises *ValidatorError* if the file fails to validate.

Note: Only basic validation is performed. In particular, extra categories or keywords that are not present in the dictionary are ignored rather than treated as errors.

---

class ihm.dictionary.**Category**
Representation of a single category in a *Dictionary*.

description = None
Human-readable text

keywords = None
Mapping from name to *Keyword* objects

mandatory = None
True iff this category is required in a compliant mmCIF file

name = None
Category name

class ihm.dictionary.**ItemType**(*name*, *primitive_code*, *construct*)
Represent the type of a data item. This keeps the set of valid strings for values of a given *Keyword*. For example, integer values can only contain the digits 0-9 with an optional +/- prefix.

case_sensitive
True iff this type is case sensitive

class ihm.dictionary.**Keyword**
Representation of a single keyword in a *Category*.

enumeration = None
Set of acceptable values, or None

item_type = None
ItemType for this keyword, or None

mandatory = None
True iff this keyword is required in a compliant mmCIF file

name = None
Keyword name

ihm.dictionary.**read**(*fh*)
Read dictionary data from the mmCIF file handle *fh*.

Returns The dictionary data.

Return type *Dictionary*

exception ihm.dictionary.**ValidatorError**
Exception raised if a file fails to validate. See *Dictionary.validate()*.

1.25 The ihm.flr Python module

Classes to handle fluorescence data. The classes roughly correspond to categories in the FLR dictionary. See the top level *FLRData* class for more information.

class ihm.flr.**Probe**(*probe_list_entry=None*, *probe_descriptor=None*)
Defines a fluorescent probe.

This class is not in the FLR dictionary, but it collects all the information connected by the probe_ids.

Parameters

- **probe_list_entry** (*ProbeList*) – A probe list object.
- **probe_descriptor** (*ProbeDescriptor*) – A probe descriptor.

class ihm.flr.**ProbeDescriptor**(*reactive_probe_chem_descriptor*, *chromophore_chem_descriptor*, *chromophore_center_atom=None*)
Collects the chemical descriptors for a fluorescent probe.

This includes the chemical descriptor of the reactive probe and the chromophore.

Parameters

- **reactive_probe_chem_descriptor** (*ihm.ChemDescriptor*) – The chemical descriptor for the reactive probe.
- **chromophore_chem_descriptor** (*ihm.ChemDescriptor*) – The chemical descriptor of the chromophore.
- **chromophore_center_atom** – The atom describing the center of the chromophore.

class ihm.flr.**ProbeList**(*chromophore_name*, *reactive_probe_flag=False*, *reactive_probe_name=None*, *probe_origin=None*, *probe_link_type=None*)
Store the chromophore name, whether there is a reactive probe available, the origin of the probe and the type of linkage of the probe.

Parameters

- **chromophore_name** (*str*) – The name of the chromophore.
- **reactive_probe_flag** (*bool*) – Flag to indicate whether a reactive probe is given.
- **reactive_probe_name** (*str*) – The name of the reactive probe.
- **probe_origin** (*str*) – The origin of the probe (intrinsic or extrinsic).
- **probe_link_type** (*str*) – The type of linkage for the probe (covalent or ligand).

class ihm.flr.**SampleProbeDetails**(*sample*, *probe*, *fluorophore_type='unspecified'*, *poly_probe_position=None*, *description=None*)
Connects a probe to a sample.

Parameters

- **sample** (*Sample*) – The sample.
- **probe** (*Probe*) – A probe that is attached to the sample.

- **fluorophore_type** (*str*) – The type of the fluorophore (donor, acceptor, or unspecified).
- **poly_probe_position** (*PolyProbePosition*) – The position on the polymer where the dye is attached to.
- **description** (*str*) – A description of the sample-probe-connection.

class ihm.flr.**PolyProbeConjugate**(*sample_probe*, *chem_descriptor*, *ambiguous_stoichiometry=False*, *probe_stoichiometry=None*)

Describes the conjugate of polymer residue and probe (including possible linker)

Parameters

- **sample_probe** (*SampleProbeDetails*) – The *SampleProbeDetails* object to which the conjugate is related.
- **chem_descriptor** (*ihm.ChemDescriptor*) – The chemical descriptor of the conjugate of polymer residue and probe.
- **ambiguous_stoichiometry** (*bool*) – Flag whether the labeling is ambiguous.
- **probe_stoichiometry** (*float*) – The stoichiometry of the ambiguous labeling.

class ihm.flr.**PolyProbePosition**(*resatom*, *mutation_flag=False*, *modification_flag=False*, *auth_name=None*, *mutated_chem_comp_id=None*, *modified_chem_descriptor=None*)

Describes a position on the polymer used for attaching the probe.

This class combines Poly_probe_position, Poly_probe_position_modified, and Poly_probe_position_mutated from the FLR dictionary.

Parameters

- **resatom** (*ihm.Residue* or *ihm.Atom*) – The residue or atom that the probe is attached to.
- **mutation_flag** (*bool*) – Flag whether the residue was mutated (e.g. a Cys mutation).
- **modification_flag** (*bool*) – Flag whether the residue was modified (e.g. replacement of a residue with a labeled residue in case of nucleic acids).
- **auth_name** (*str*) – An author-given name for the position.
- **mutated_chem_comp_id** – The chemical component ID of the mutated residue.
- **modified_chem_descriptor** (*ihm.ChemDescriptor*) – The chemical descriptor of the modified residue.

class ihm.flr.**Sample**(*entity_assembly*, *num_of_probes*, *condition*, *description=None*, *details=None*, *solvent_phase=None*)

Sample corresponds to a measurement.

Parameters

- **entity_assembly** (*EntityAssembly*) – The assembly of the entities that was measured.
- **num_of_probes** (*int*) – The number of probes in the sample.
- **condition** (*SampleCondition*) – The sample conditions for the Sample.
- **description** (*str*) – A description of the sample.
- **details** (*str*) – Details about the sample.
- **solvent_phase** – The solvent phase of the sample (liquid, vitrified, or other).

class ihm.flr.**EntityAssembly**(*entity=None*, *num_copies=0*)
The assembly of the entities that are in the system.

Parameters

- **entity** (Entity) – The entity to add.
- **num_copies** – The number of copies for the entity in the assembly.

class ihm.flr.**SampleCondition**(*details=None*)
Description of the sample conditions.

Currently this is only text, but will be extended in the future.

Parameters **details** (*str*) – Description of the sample conditions.

class ihm.flr.**Experiment**(*instrument=None*, *inst_setting=None*, *exp_condition=None*, *sample=None*, *details=None*)
The Experiment collects combinations of instrument, experimental settings and sample.

Parameters

- **instrument** (*Instrument*) – The instrument.
- **inst_setting** (*InstSetting*) – The instrument setting.
- **exp_condition** (*ExpCondition*) – The experimental conditions.
- **sample** (*Sample*) – The sample.
- **details** – Details on the experiment.

add_entry(*instrument*, *inst_setting*, *exp_condition*, *sample*, *details=None*)
Entries to the experiment object can also be added one by one.

contains(*instrument*, *inst_setting*, *exp_condition*, *sample*)
Checks whether a combination of *Instrument*, *InstSetting*, *ExpCondition*, *Sample* is already included in the experiment object.

get_entry_by_index(*index*)
Returns the combination of *Instrument*, *InstSetting*, *ExpCondition*, *Sample*, and details for a given index.

class ihm.flr.**Instrument**(*details=None*)
Description of the Instrument used for the measurements.

Currently this is only text, but will be extended in the future.

Parameters **details** – Description of the instrument used for the measurements.

class ihm.flr.**InstSetting**(*details=None*)
Description of the instrument settings.

Currently this is only text, but will be extended in the future.

Parameters **details** (*str*) – Description of the instrument settings used for the measurement (e.g. laser power or size of observation volume in case of confocal measurements).

class ihm.flr.**ExpCondition**(*details=None*)
Description of the experimental conditions.

- Currently this is only text, but will be extended in the future.*

Parameters **details** (*str*) – Description of the experimental conditions (e.g. the temperature at which the experiment was carried out).

class ihm.flr.**FRETAnalysis**(*experiment*, *sample_probe_1*, *sample_probe_2*, *forster_radius*, *type*, *calibration_parameters=None*, *lifetime_fit_model=None*, *ref_measurement_group=None*, *method_name=None*, *details=None*, *chi_square_reduced=None*, *donor_only_fraction=None*, *dataset=None*, *external_file=None*, *software=None*)

An analysis of FRET data that was performed.

Parameters

- **experiment** (*Experiment*) – The Experiment object for this FRET analysis.
- **sample_probe_1** (*SampleProbeDetails*) – The combination of sample and probe for the first probe.
- **sample_probe_2** (*SampleProbeDetails*) – The combination of sample and probe for the second probe.
- **forster_radius** (*FRETForsterRadius*.) – The Förster radius object for this FRET analysis.
- **type** (*str*) – The type of the FRET analysis (intensity-based or lifetime-based).
- **calibration_parameters** (*FRETCalibrationParameters*) – The calibration parameters used for this analysis (only in case of intensity-based analyses).
- **lifetime_fit_model** (*LifetimeFitModel*) – The fit model used in case of lifetime-based analyses.
- **ref_measurement_group** (LifetimeRefMeasurementGroup) – The group of reference measurements in case of lifetime-based analyses.
- **method_name** (*str*) – The method used for the analysis.
- **chi_square_reduced** (*float*) – The chi-square reduced as a quality measure for the fit.
- **donor_only_fraction** (*float*) – The donor-only fraction.
- **dataset** (*ihm.dataset.Dataset*) – The dataset used.
- **external_file** – The external file that contains (results of) the analysis.
- **software** (*ihm.Software*) – The software used for the analysis.

class ihm.flr.**LifetimeFitModel**(*name*, *description*, *external_file=None*, *citation=None*)

A lifetime-fit model used for lifetime-based analysis.

Parameters

- **name** (*str*) – The name of the fit model.
- **description** (*str*) – A description of the fit model.
- **external_file** – An external file that contains additional information on the fit model.
- **citation** (*ihm.Citation*) – A citation for the fit model.

class ihm.flr.**RefMeasurementGroup**(*details=None*)

A Group containing reference measurements for lifetime-based analysis.

Parameters **details** (*str*) – Details on the Group of reference measurements.

add_ref_measurement(*ref_measurement*)

Add a lifetime reference measurement to a ref_measurement_group.

class ihm.flr.**RefMeasurement**(*ref_sample_probe*, *details=None*, *list_of_lifetimes=None*)

A reference measurement for lifetime-based analysis.

Parameters

- **ref_sample_probe** (*SampleProbeDetails*) – The combination of sample and probe used for the reference measurement.
- **details** (*str*) – Details on the measurement.
- **list_of_lifetimes** (List of *RefMeasurementLifetime*) – A list of the results from the reference measurement.

add_lifetime(*lifetime*)
Add a lifetime to the list_of_lifetimes.

class ihm.flr.**RefMeasurementLifetime**(*species_fraction*, *lifetime*, *species_name=None*)
Lifetime for a species in a reference measurement. :param float species_fraction: The species-fraction for the respective lifetime. :param float lifetime: The lifetime (in ns). :param str species_name: A name for the species.

class ihm.flr.**FRETDistanceRestraintGroup**
A collection of FRET distance restraints that are used together.

add_distance_restraint(*distance_restraint*)
Add a distance restraint to a distance_restraint_group

class ihm.flr.**FRETDistanceRestraint**(*sample_probe_1*, *sample_probe_2*, *analysis*, *distance*, *distance_error_plus=0.0*, *distance_error_minus=0.0*, *distance_type=None*, *state=None*, *population_fraction=0.0*, *peak_assignment=None*)
A distance restraint from FRET.

Parameters

- **sample_probe_1** (*SampleProbeDetails*) – The combination of sample and probe for the first probe.
- **sample_probe_2** (*SampleProbeDetails*) – The combination of sample and probe for the second probe.
- **analysis** (*FRETAnalysis*) – The FRET analysis from which the distance restraint originated.
- **distance** (*float*) – The distance of the restraint.
- **distance_error_plus** (*float*) – The (absolute, e.g. in Angstrom) error in the upper direction, such that `upper boundary = distance + distance_error_plus`.
- **distance_error_minus** (*float*) – The (absolute, e.g. in Angstrom) error in the lower direction, such that `lower boundary = distance + distance_error_minus`.
- **distance_type** (*str*) – The type of distance (<R_DA>, <R_DA>_E, or R_mp).
- **state** (*ihm.model.State*) – The state the distance restraints is connected to. Important for multi-state models.
- **population_fraction** (*float*) – The population fraction of the state in case of multi-state models.
- **peak_assignment** (*PeakAssignment*) – The method how a peak was assigned.

class ihm.flr.**FRETForsterRadius**(*donor_probe*, *acceptor_probe*, *forster_radius*, *reduced_forster_radius=None*)
The FRET Förster radius between two probes.

Parameters

- **donor_probe** (*Probe*) – The donor probe.
- **acceptor_probe** (*Probe*) – The acceptor probe.
- **forster_radius** (*float*) – The Förster radius between the two probes.
- **reduced_forster_radius** (*float*) – The reduced Förster radius between the two probes.

class ihm.flr.**FRETCalibrationParameters**(*phi_acceptor=None*, *alpha=None*, *alpha_sd=None*, *gg_gr_ratio=None*, *beta=None*, *gamma=None*, *delta=None*, *a_b=None*)

The calibration parameter from the FRET measurements. For the definitions of the parameters see Hellenkamp et al. Nat. Methods 2018.

Parameters

- **phi_acceptor** (*float*) – The quantum yield of the acceptor.
- **alpha** (*float*) – The alpha parameter.
- **alpha_sd** (*float*) – The standard deviation of the alpha parameter.
- **gg_gr_ratio** (*float*) – The ratio of the green and red detection efficiencies.
- **beta** (*float*) – The beta parameter.
- **gamma** (*float*) – The gamma parameter.
- **delta** (*float*) – The delta parameter.
- **a_b** (*float*) – The fraction of bright molecules.

class ihm.flr.**PeakAssignment**(*method_name*, *details=None*)

The method of peak assignment in case of multiple peaks, e.g. by population.

Parameters

- **method_name** (*str*) – The method used for peak assignment.
- **details** (*str*) – The details of the peak assignment procedure.

class ihm.flr.**FRETModelQuality**(*model*, *chi_square_reduced*, *dataset_group*, *method*, *details=None*)

The quality measure for a Model based on FRET data.

Parameters

- **model** (*ihm.model.Model*) – The model being described.
- **chi_square_reduced** – The quality of the model in terms of chi_square_reduced based on the Distance restraints used for the modeling.
- **dataset_group** (*ihm.dataset.DatasetGroup*) – The group of datasets that was used for the quality estimation.
- **method** – The method used for judging the model quality.
- **details** (*str*) – Details on the model quality.

class ihm.flr.**FRETModelDistance**(*restraint*, *model*, *distance*, *distance_deviation=None*)

The distance in a model for a certain distance restraint.

Parameters

- **restraint** (*FRETDistanceRestraint*) – The Distance restraint.
- **model** (*ihm.model.Model*) – The model the distance applies to.

- **distance** – The distance obtained for the distance restraint in the current model.
- **distance_deviation** – The deviation of the distance in the model compared to the value of the distance restraint.

class ihm.flr.**FPSModeling**(*protocol*, *restraint_group*, *global_parameter*, *probe_modeling_method*, *details=None*)

Collect the modeling parameters for different steps of FPS, e.g. Docking, Refinement, or Error estimation.

Parameters

- **protocol** (*ihm.protocol.Protocol*) – The modeling protocol to which the FPS modeling step belongs.
- **restraint_group** (*FRETDistanceRestraintGroup*) – The restraint group used for the modeling.
- **global_parameter** (*FPSGlobalParameters*) – The global FPS parameters used.
- **probe_modeling_method** (*str*) – either "AV" or "MPP".
- **details** (*str*) – Details on the FPS modeling.

class ihm.flr.**FPSGlobalParameters**(*forster_radius*, *conversion_function_polynom_order*, *repetition*, *av_grid_rel*, *av_min_grid_a*, *av_allowed_sphere*, *av_search_nodes*, *av_e_samples_k*, *sim_viscosity_adjustment*, *sim_dt_adjustment*, *sim_max_iter_k*, *sim_max_force*, *sim_clash_tolerance_a*, *sim_reciprocal_kt*, *sim_clash_potential*, *convergence_e*, *convergence_k*, *convergence_f*, *convergence_t*, *optimized_distances='All'*)

The global parameters in the FPS program.

For a description of the parameters, see also the FPS manual.

Parameters

- **forster_radius** (*float*) – The Förster radius used in the FPS program.
- **conversion_function_polynom_order** (*int*) – Order of the polynom for the conversion function between Rmp and <RDA>E.
- **repetition** (*int*) – The number of repetitions.
- **av_grid_rel** (*float*) – The AV grid spacing relative to the smallest dye or linker dimension.
- **av_min_grid_a** (*float*) – The minimal AV grid spacing in Angstrom.
- **av_allowed_sphere** (*float*) – The allowed sphere radius.
- **av_search_nodes** (*int*) – Number of neighboring positions to be scanned for clashes.
- **av_e_samples_k** (*float*) – The number of samples for calculation of E (in thousand).
- **sim_viscosity_adjustment** (*float*) – Daming rate during docking and refinement.
- **sim_dt_adjustment** (*float*) – Time step during simulation.
- **sim_max_iter_k** (*float*) – Maximal number of iterations (in thousand).
- **sim_max_force** (*float*) – Maximal force.
- **sim_clash_tolerance_a** (*float*) – Clash tolerance in Angstrom.
- **sim_reciprocal_kt** (*float*) – reciprocal kT.

- **sim_clash_potential** (*str*) – The clash potential.
- **convergence_e** (*float*) – Convergence criterion E.
- **convergence_k** (*float*) – Convergence criterion K.
- **convergence_f** (*float*) – Convergence criterion F.
- **convergence_t** (*float*) – Convergence criterion T.
- **optimized_distances** (*str*) – Which distances are optimized?

class ihm.flr.**FPSAVModeling**(*fps_modeling*, *sample_probe*, *parameter*)

FPS modeling using AV. This object connects the FPS_modeling step, the sample_probe and the respective AV parameters.

Parameters

- **fps_modeling** (*FPSModeling*) – The FPS modeling ID.
- **sample_probe** (*SampleProbeDetails*) – The Sample probe ID.
- **parameter** (*FPSAVParameter*) – The FPS AV parameters used.

class ihm.flr.**FPSAVParameter**(*num_linker_atoms*, *linker_length*, *linker_width*, *probe_radius_1*, *probe_radius_2=None*, *probe_radius_3=None*)

The AV parameters used for the modeling using FPS.

Parameters

- **num_linker_atoms** (*int*) – The number of atoms in the linker.
- **linker_length** (*float*) – The length of the linker in Angstrom.
- **linker_width** (*float*) – The width of the linker in Angstrom.
- **probe_radius_1** (*float*) – The first radius of the probe.
- **probe_radius_2** (*float*) – If AV3 is used, the second radius of the probe.
- **probe_radius_3** (*float*) – If AV3 is used, the third radius of the probe.

class ihm.flr.**FPSMPPModeling**(*fps_modeling*, *mpp*, *mpp_atom_position_group*)

Maps the FPSModeling object to a mean probe position and connects it to the reference coordinate system.

Parameters

- **fps_modeling** (*FPSModeling*) – The FPS modeling object.
- **mpp** (*FPSMeanProbePosition*) – The ID of the mean probe position.
- **mpp_atom_position_group** (*FPSMPPAtomPositionGroup*) –

class ihm.flr.**FPSMeanProbePosition**(*sample_probe*, *x*, *y*, *z*)

The mean probe position of an AV, which can be used instead of an AV.

It is usually not recommended to use this. Use AVs instead. The coordinates are with respect to a reference coordinate system defined by *FPSMPPAtomPositionGroup*.

Parameters

- **sample_probe** (*SampleProbeDetails*) – The Sample probe.
- **x** (*float*) – The x-coordinate of the mean probe position.
- **y** (*float*) – The y-coordinate of the mean probe position.
- **z** (*float*) – The z-coordinate of the mean probe position.

class ihm.flr.**FPSMPPAtomPositionGroup**
A group of atom positions used to define the coordinate system of a mean probe position. *Not part of the FLR dictionary.*

class ihm.flr.**FPSMPPAtomPosition**(*atom*, *x*, *y*, *z*)
An atom used to describe the coordinate system for a mean probe position

Parameters

- **atom** (*ihm.Atom*) – The atom being described.
- **x** (*float*) – The x-coordinate of the atom.
- **y** (*float*) – The y-coordinate of the atom.
- **z** (*float*) – The z-coordinate of the atom.

class ihm.flr.**FLRData**
A collection of the fluorescence data to be added to the system.

Instances of this class are generally added to *flr_data*.

distance_restraint_groups = None
All groups of FRET distance restraints. See *FRETDistanceRestraintGroup*.

fps_modeling = None
All modeling objects. See *FPSAVModeling* and *FPSMPPModeling*.

fret_model_distances = None
All distances in models for distance restraints. See *FRETModelDistance*.

fret_model_qualities = None
All quality measures for models based on FRET data. See *FRETModelQuality*.

poly_probe_conjugates = None
All conjugates of polymer residue and probe. See *PolyProbeConjugate*.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

i

Index

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

X

Y

Z