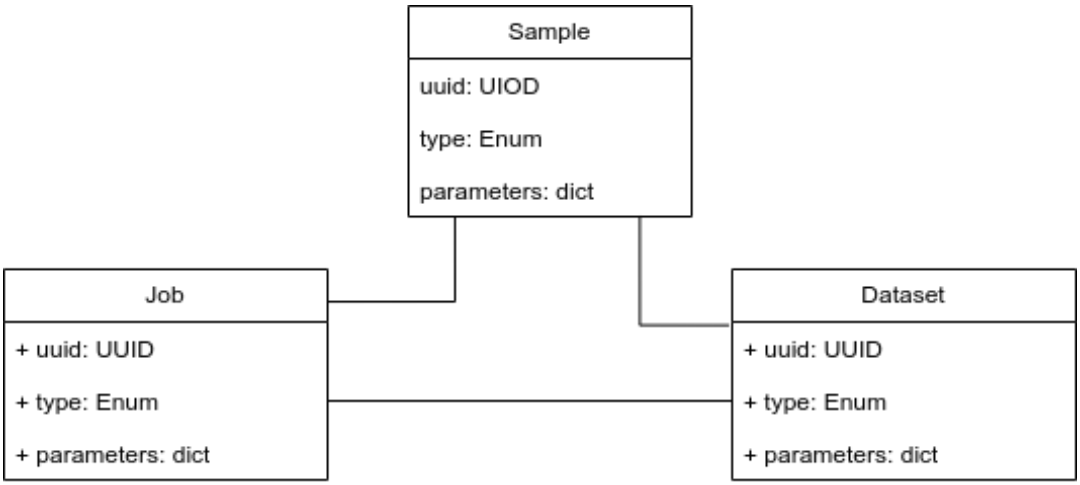
Basic model discussion – version 0.0.4

Introduction

This document shows the basic skeleton of the generic kind of LIMS model that the API/metadata are supposed to latch on to. This model is not the main purpose of the working group, and the implementation in actual LIMS systems may be quite different from this. But you cannot sensibly plan an API or a series of metadata without at least some general idea of the kind of slots available for your data to fit into. In practice there will be all kinds of different data structures modeled for specific kinds of data. But all of these can be seen as belonging to only three different kinds of 'thing':



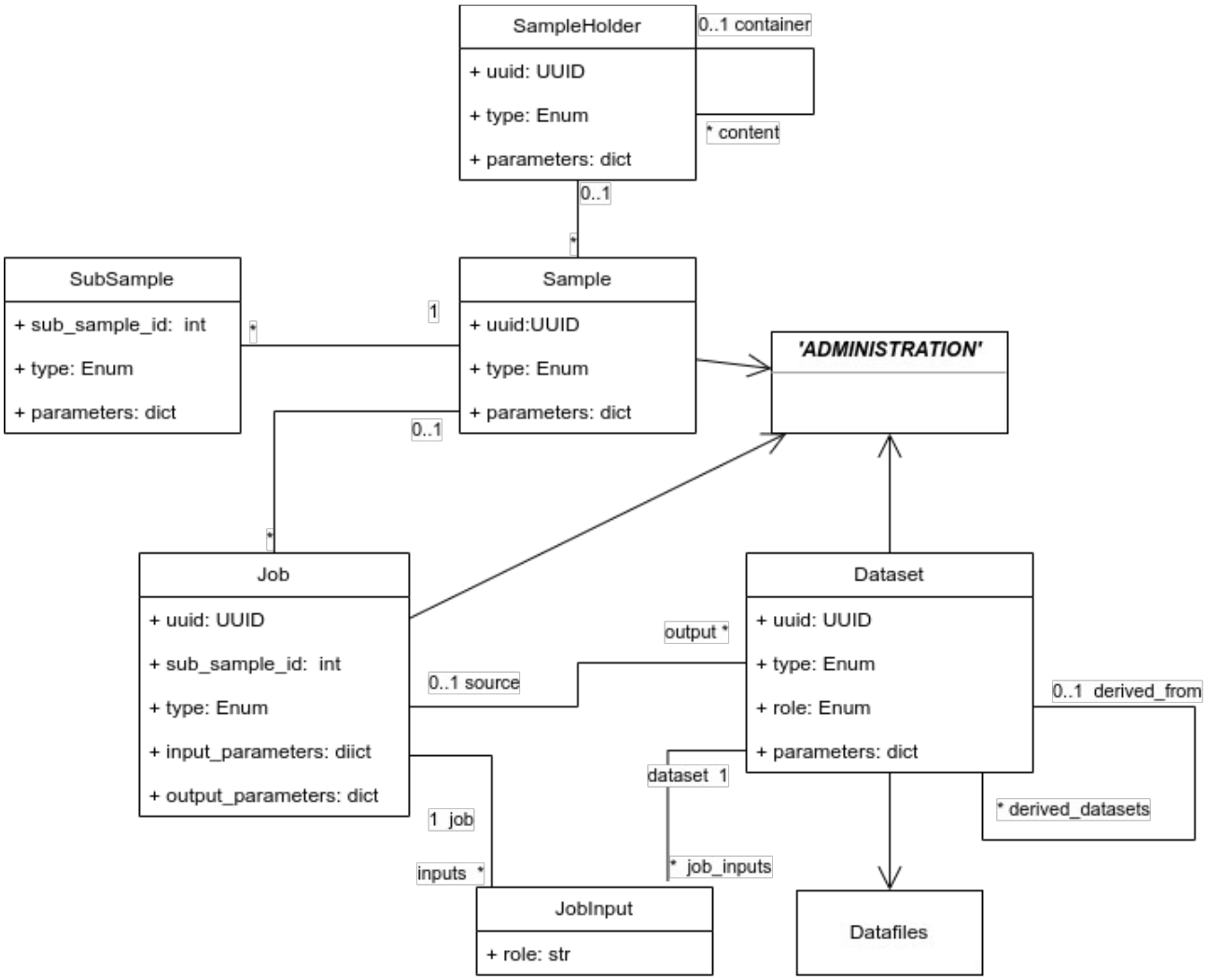
Overview diagram of the generic LIMS model

The Sample slot covers crystals, pucks, plates, pins and grids, etc.. The Job slot covers both actual experiments and calculations, which both produce Datasets. And Datasets can be any heterogeneous collection of data files that are considered, and addressed as a unit. Each slot has a `parameters` field for structured metadata, and a`type` slot that distinguishes between e.g. plates and grids, MX measurements, XRF spectra, and calculations, rotation data and mesh scans, and that allows you to validate the data you attach..

These are 'things' that need to be identified and tracked (hence the UUID), For e.g. a Unit Cell, you do not need to know whether two records point to the 'same' unit cell – you just compare the values. But for a Dataset you need to be able to connect it back to specific samples, experiments and calculations. These three 'things', with the links between them, allow you to track the provenance of all your data through the calculations that produced it, the experiments that measured it, and to the samples that it was derived from. Another way of looking at this is to think of a MongoDB storage. Most of the information is stored as values in JSON data structures, but there are a few objects that have to appear in actual database tables, with links and foreign keys, to connect the data together.

Details

If you are mainly interested in the API/metadata it is probably enough to read the introduction and skip the rest of this document. But here is the detailed working out of the model, as justified by the use cases in the next section



Detail diagram of generic LIMS mopdel

This diagram shows the underlying LIMS model, with the attributes and links necessary to connect the data – and that must one way or the other be dealt with in the LIMS. The detailed modeling will be handled in the API/metadata, which is dealt with in another document. The ‘type’ field determines which metadata can be attached to each object. The ‘parameters’ fields hold structured metadata, which will include a field for site- or program-specific data. Each Job has a one-to-many link to output Datasets, and a many-to-many link (transmitted via the JobInput class) to different kinds of input Datasets. The Sample, SubSample and SampleHolder are discussed in more detail below; the Sample Holder is only necessary for sample tracking. The ADMINISTRATION’ box is a placeholder for things like sessions, projects, users, and access permissions, which are not part of

the model shown here, but which will require some minimum interaction in the API in order to define access permission on newly created objects.

It will be noted that the entire model requires only a few database tables, with dictionaries for metadata, and that there is no requirement (so far) for supporting object IDs / crosslinks within the metadata. In a point raised by Dectris, it would be possible if desired to add an extra table for tags, that would allow grouping Datasets (through shared tags) and connecting metadata to the groups.

The metadata structure would use the same data schemas as the API. The structured data that could be attached to an object would be determined by its ‘type’ field, but the schemas could be used in a number of different contexts. For instance, a UnitCell data structure could be used in whatever context required unit cell data. To simplify the modeling we would use the same schema to specify input parameters and to store with the actual result, so that a diffraction plan or processing plan would use the same schema as the experiment or processing job. In a more complex example, the schema specifying a diffraction Sweep could serve both in the diffraction plan, in passing information to the acquisition queue, and in describing the resulte Datase.

Job class

Depending on the type, the Job could specify either an actual experiment, or a processing job or calculation. Since a Job (or experiment) can have multiple input and output data sets, the Job, rather than the Dataset, is the natural unit for the user to look at in overviews. The Job can have links to multiple input Datasets, with the type of the link specified by the JobInput.role, allowing you to distinguish e.g. between calculation input and reference data.

Dataset class

A Dataset could be produced either by an experiment or a processing job (or an ‘import job’, if desired) and could be of any type, as specified by the `type` field. The `role` serves to distinguish between different outputs of the same type from as single source, e.g. between characterisation and acquisition sweep data from the same experiment. The Dataset points to the files with the actual data, in a manner which would depend on the type and using information stored as part of the metadata. The same data could be stored in a number of complex ways, e.g. as individual images, as a single multi-image files, or as part of a larger HDF5 data structure, and the file specification will have to support any of these, including specifying subsets of larger data structures.

The (optional) ‘source’ link leads from the Dataset to the Job that created it, for tracking purposes. The ‘derived_from’ link provides an alternative provenance. It may be that the input to a calculation uses only part of a Dataset, e.g. after filtering out unwanted images. It may also be that you need to attach parameters to calculation input that belong neither to the Dataset as such, not to the calculation, but to the combination of the two, e.g. a weight parameter. In either case you can create a duplicate Dataset record with the data contents and parameters you need, and use the derived_from link for provenance tracking. In general one would not duplicate the actual data files, but only the Dataset record pointing to them.

JobInput class

The JobInput class is necessary to support a many-to-many link between Jobs and Datasets. The `role` attribute describes only the role of the dataset within the Job, e.g. ‘reference’ would mean that this particular dataset was used as reference data within the job. The `role` type has been left as `str` rather than `Enum` for now, in case it prove difficult to model all the roles that might be needed by various processing programs. If we want to make the Job.inputs links ordered (i.e. lists instead of sets) we could add an `ordinal` attribute to JobInput.

Sample handling

The area of Samples requires some explanation. The SampleHolder is something that (recursively) contains Samples, e.g. a Dewar, puck or plate. These serve only for sample tracking. A SubSample, on the other hand, is the diffracting matter that you are using to acquire a single Dataset – in most cases we could also have called it ‘Crystal’. A Sample, finally, is a sample holder that is associated with a given preparation and sample composition, and that directly contains crystals (SubSamples). A Sample could be a drop, a loop on a pin, or a grid or other SSX sample holder. The model needs to deal with both Samples and SubSamples because Samples (e.g. for SSX) can contain a great number of SubSamples, possibly in random locations, and because the SubSamples may only be identified during the experiment, e.g. by grid scan. An experiment is performed on only one Sample, but may apply to either a single crystal or multiple crystals within it. To deal with these situation we are putting the SampleData (macromolecule, ligands, unit cells, ...) inside the Sample and linking the Sample to the Job. SubSamples are identified by the Sample they are part of and a sub_sample_id. Jobs have an optional sub_sample_id for the (common) case where an experiment only applies to a single crystal. Experimental Datasets can be connected to SubSamples via the Job-Sample link and a sub_sample_id in the Dataset parameters. Processing Jobs and processed Datasets should not have direct link to a Sample object, since they quire frequently do not arise from a single Sample. To provide sample information to processing jobs, e.g. protein, ligands, unit cells, we would instead put SampleData as part of the Job and Dataset metadata.

Use case examples

Examples of model use for the use cases in the discussion (see https://github.com/rhfogh/mxlims_data_model/discussions/8). Note that the `type` and `role` values have been chosen to be illustrative, and are not intended as the final values to use. The number of `Characterisation` and `Centring` Datasets will vary between actual experiments, so this is just an example.

Use case 1 – Single-sweep MX acquisition

Simple single-sweep data acquisition. ‘Images’ is the role name to use in the JobInput class.

ID	type	role	sourceID	inputIDs
1	MXexperiment			

2	MeshScan	Centring	1	
3	LineScan	Centring	1	
4	LineScan	Centring	1	
5	Sweep	Characterisation	1	
5	Sweep	Acquisition	1	
6	MXprocessing	-	-	Images:(5,)
7	MXreflections	ResultAniso	6	

Use case 2 – Multi-sweep or multi-crystal MX acquisition

For a single-crystal, multi-sweep experiment you would need a sub_sample_id at the Job level, but not for the individual Sweeps. In the multi-crystal case the crystals would have to be from the same Sample; with sub_sample_id on the individual sweeps. Note that this case would appear in viewers as a single experiment record, with a single processing result.

ID	type	role	sourceID	inputIDs	crystalID
1	MXexperiment				
2	MeshScan	Centring	1		
3	LineScan	Centring	1		
4	LineScan	Centring	1		
5	Sweep	Characterisation	1		
6	MeshScan	Centring	1		
7	LineScan	Centring	1		
8	LineScan	Centring	1		
9	Sweep	Acquisition	1		
10	MeshScan	Centring	1		
11	LineScan	Centring	1		
12	LineScan	Centring	1		
13	Sweep	Acquisition	1		
14	MXprocessing	-	-	Images:(9,13)	
15	MXreflections	ResultAniso	14		

Use case 3 – interleaved multi-wavelength acquisition

The experiments have separate Datasets and separate reflection files for the two wavelengths. As shown, the interleaved acquisition has been organised with one Dataset for each wavelength. This would be the recommended approach; the metadata would then hold the information on how the images had been grouped into scans, and in which order the scans had been acquired. Alternatively the results could be organised with a Dataset for each wedge. The same situation would apply to inverse-beam acquisition.

Class	ID	type	role	sourceID	inputIDs	Comment
Job	1	MXexperiment				
Dataset	2	MeshScan	Centring	1		
Dataset	3	LineScan	Centring	1		
Dataset	4	LineScan	Centring	1		
Dataset	5	Sweep	Characterisation	1		
Dataset	6	Sweep	Acquisition	1		Wavelength A
Dataset	7	Sweep	Acquisition	1		Wavelength B
Job	8	MXprocessing	-	-	Images:(6, 7)	
Dataset	9	MXreflections	ResultAniso	8		Wavelength A
Dataset	10	MXreflections	ResultAniso	8		Wavelength B

Use case 4 – SSX-type experiments

We are not supporting SSX from the beginning, but this example shows an important use case, of how to handle multi-experiment and multi-sample processing, and how to select subsets of data. Datasets 5 and 6, that are used for processing, contain the complete set of metadata and pointers to the actual data files. The derived_from links would not be needed to access the necessary metadata, but serve for provenance tracking only. In general the actual data files would not be copied; the new Datasets would point to the same files, but might specify only a subset of the original data, e.g. selecting a subset of grid positions. This mechanism would also allow the addition of metadata (possibly program-specific) that could not be put into the original Dataset, since they would apply to a specific processing only.

Class	ID	type	role	sourceID	inputIDs	sampleID	derivedFrom
Job	1	SSXexperiment				Sample A	
Dataset	2	MultiCrystalImage	Acquisition	1			
Job	3	SSXexperiment				Sample B	
Dataset	4	MultiCrystalImage	Acquisition	3			
Dataset	5	MultiCrystalImage	Acquisition				2
Dataset	6	MultiCrystalImage	Acquisition				4
Job	7	SSXprocessing	-	-	Images:(5, 6)		
Dataset	8	MXreflections	ResultAniso	7			

Use case 5 – complex processing

This example shows the features available for more complex processing, groped togetehr in a single example. Processing job 200 works on data from experiment 203 and 207, but uses a derived Dataset for one of the sweeps, possibly to exclude some image ranges, or to add some program-specific metadata parameter. In addition to the images, the processing job takes a reference MTZ file as input, and produces more than one output Dataset. Since the processing uses data from more than one experiment it is not associated with a single Sample. Sample data (macromolecule, ligands, expected unit cell, etc.) will instad be put into the input_parameters of Job 209, as a SampleData record.

Class	ID	type	role	sourceID	inputIDs	derivedFrom
Dataset	82	MXreflections	ResultAniso	81		
Job	203	MXexperiment				
Dataset	204	Sweep	Acquisition	203		
Dataset	205	Sweep	Acquisition	203		
Dataset	206	Sweep	Acquisition			205
Job	207	MXExperiment				
Dataset	208	Sweep	Acqusition	207		
Job	209	MXprocessing	-	-	Images:(204,206, 208) Reference:(82,)	
Dataset	210	MXreflections	ResultAniso	207		
Dataset	211	MXreflections	ResultIso	207		

	job_ID	dataset_ID	role
JobInput	209	82	Reference
JobInput	209	204	Images
JobInput	209	206	Images
JobInput	209	208	Images