GΦL
Global Phasing Limited

MXLIMS
model and implementation

Global Phasing

MXCuBE/ISPyB Meeting, Hamburg, May 2025

GΦL
Global Phasing Limited

- MXLIMS goals
- Basic structure and implementation
- Model contents
- Topics

GΦL
Global Phasing Limited

MXLIMS goals

Scope, use cases, product, requirements

GΦL
Global Phasing Limited

Purpose

- Data model and scientific API for communication with LIMS and beamlines
- Intended and organised as a collaborative standard
- Structured for easy development and maintenance
- Initial scope: MX and related techniques
  - Extension to SSX contemplated
  - Can be extended as far as people want to take it.

GΦL
Global Phasing Limited

Use cases

- Communication in multi-synchrotron projects
  - Sample shipping, fetching raw and processed data, multi-site projects
- Application API
  - Diffraction plan → Strategy calculation → Acquisition queue → datasets and metadata → processing input/output
- Data storage
  - Data provenance tracking, integration with existing LIMS, possible implementation of data model in LIMS systems

GΦL
Global Phasing Limited

Requirements

- A single, clear specification for all of MXLIMS
  - Problem: inter-object links must be two-way and form a network; JSON only supports a *tree* of one-way links
- Flexible, maintainable, easy to modify, collaborative
  - Non-relational (Mongo-type) database model with few core tables; Full relational database (like ISPyB) is too inflexible.
  - Slots for site-specific and free data fields for fast extension
  - Model versioning
- Support for provenance tracking
  - Requires *network* of objects, not just a tree
    Experiment output → processing input
  - Unique objects IDs (UUID) for identification
  - Defined inter-object links
  - Class for dataset, experimental/processing job, sample, and sample holder

Global Phasing Limited

MXLIMS products

- Data model with specification files
  - *Decided to use JSON schemas to store specification*
- Message specification with validation against schemas
  - *JSON files as messages*
- API reflecting the data model to support data exchange
- In-memory implementation with type checking
  - *Custom implementation based on Pydantic*
- Support for (non-relational) database implementation

GΦL

Global Phasing Limited

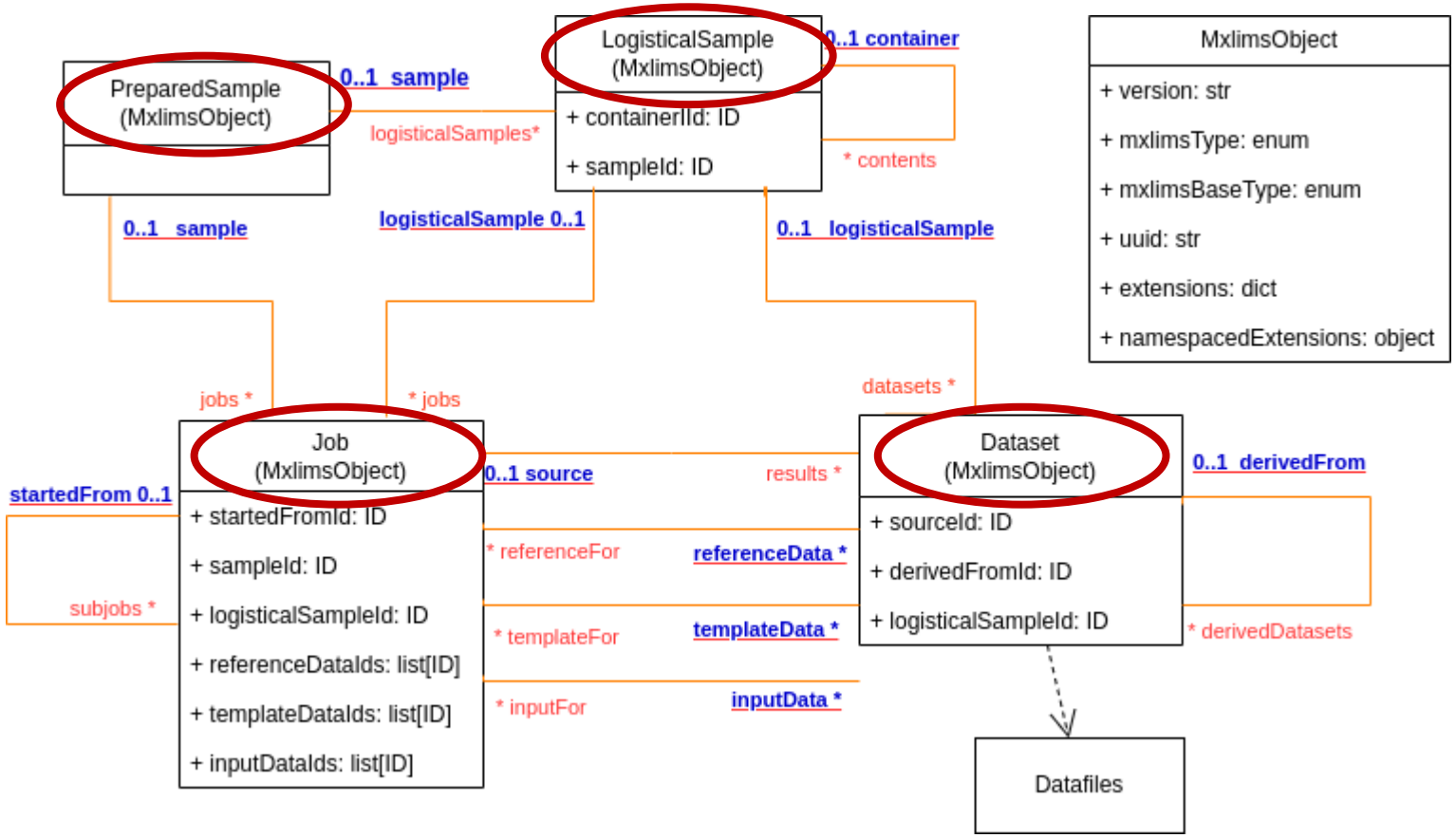
Model structure and implementation

The basic model, specification, code generation and implementation

Global Phasing Limited

Core model –classes

- PreparedSample: sample contents and characteristics
- LogisticalSample: sample objects and holders (Crystal, Pin, Puck, Plate, Well, …)
- Job: experiment or processing run that produces data
- Dataset: results, but also input or reference data, diffraction plan, …

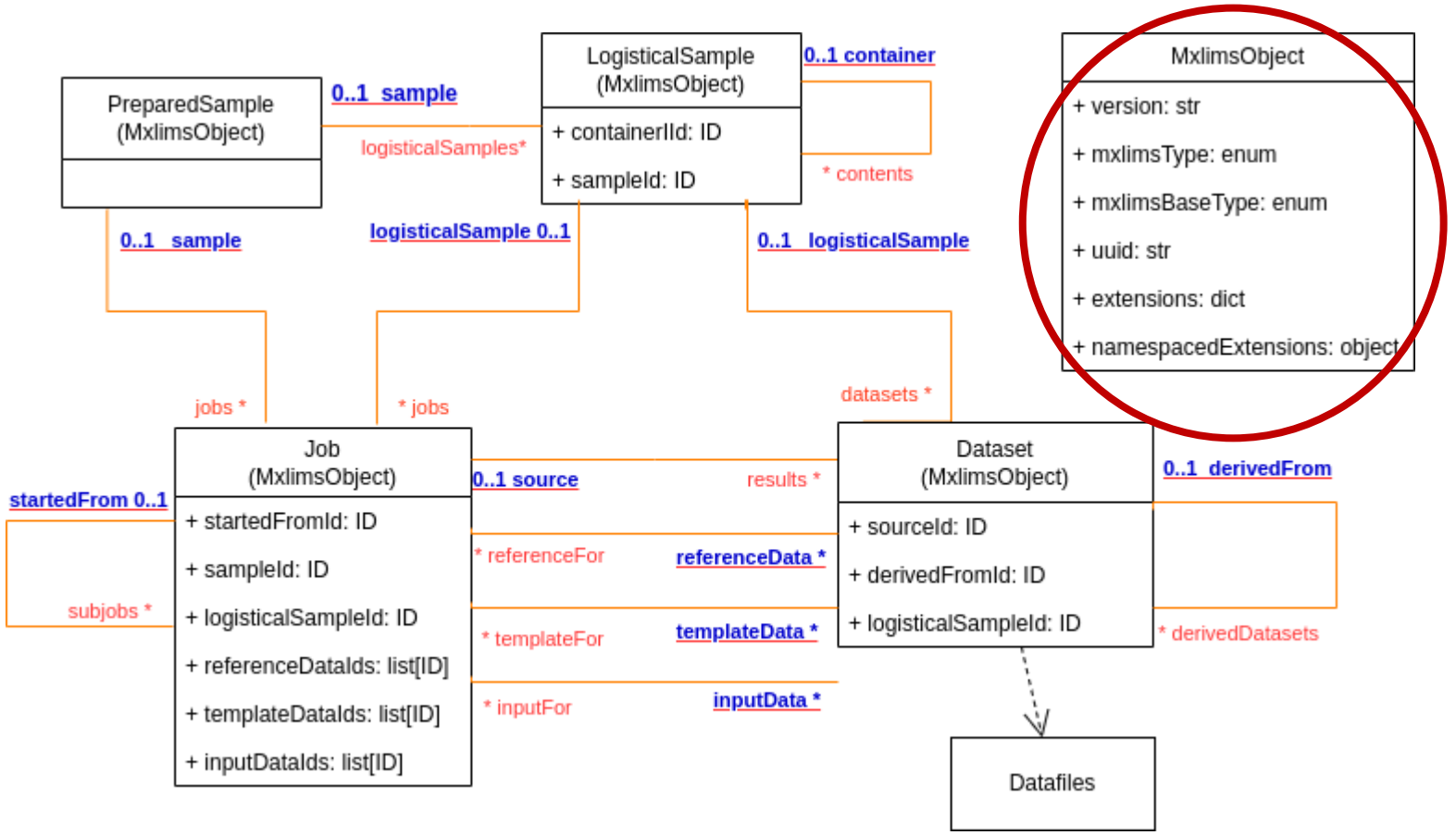


Global Phasing Limited

Core model – base type

All objects have a version string, UUID, mxlimsType (class name, e.g. ‘Puck’) and slots for free and schema-defined extensions.

‘mxlimsBaseType’ is for schema definition purposes, defines if object ia a Job, Dataset, PreparedSample or LogisticalSample.

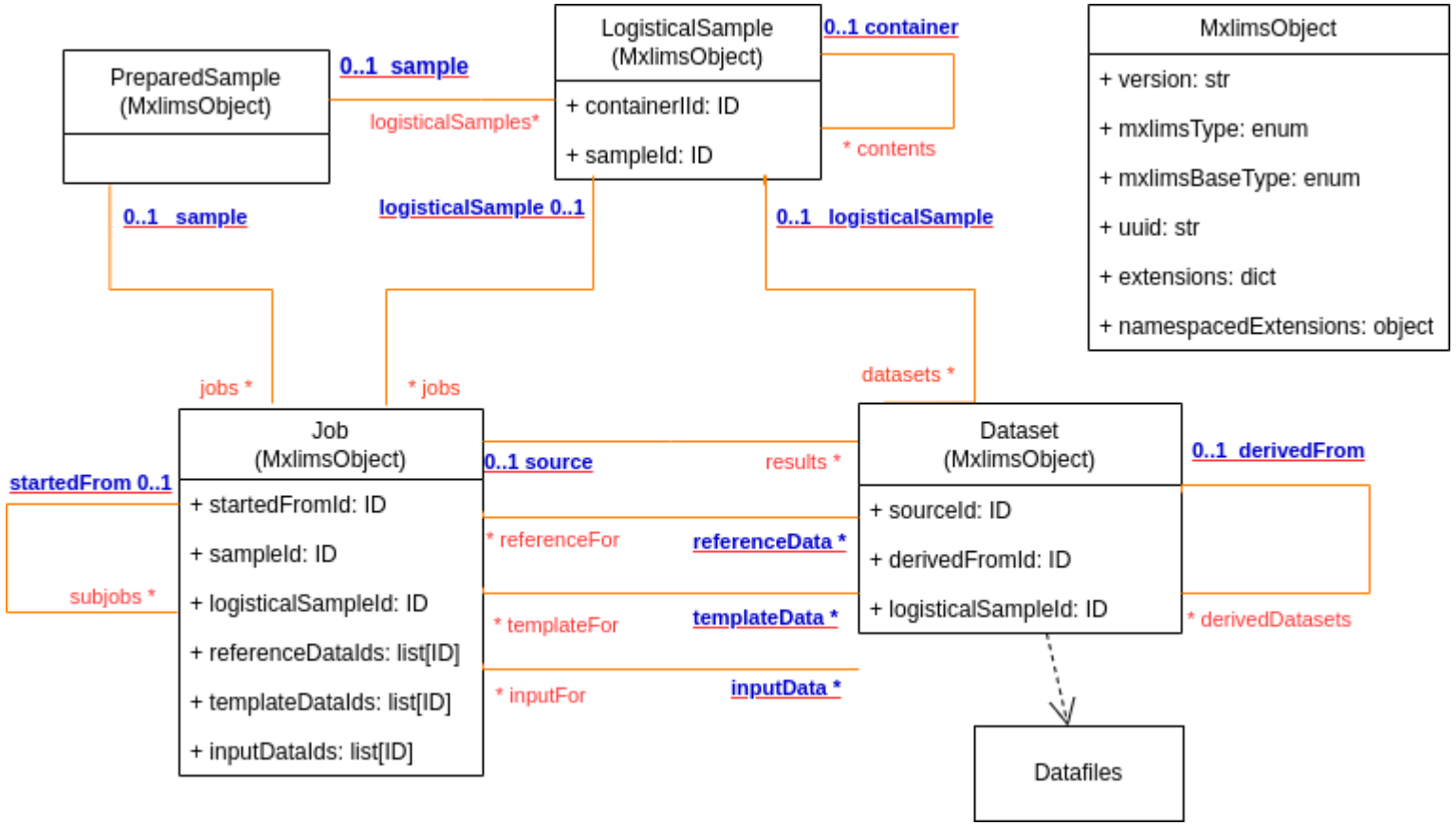


Global Phasing Limited

Core model - links

- JSON schemas have defined one-way typed links (in blue)
- In-memory objects and databases handle links using untyped foreign-key ID attributes (in black)
- The API supports two-way typed links (red and orange)

It is HARD to support all three in a single specification



GΦL
Global Phasing Limited

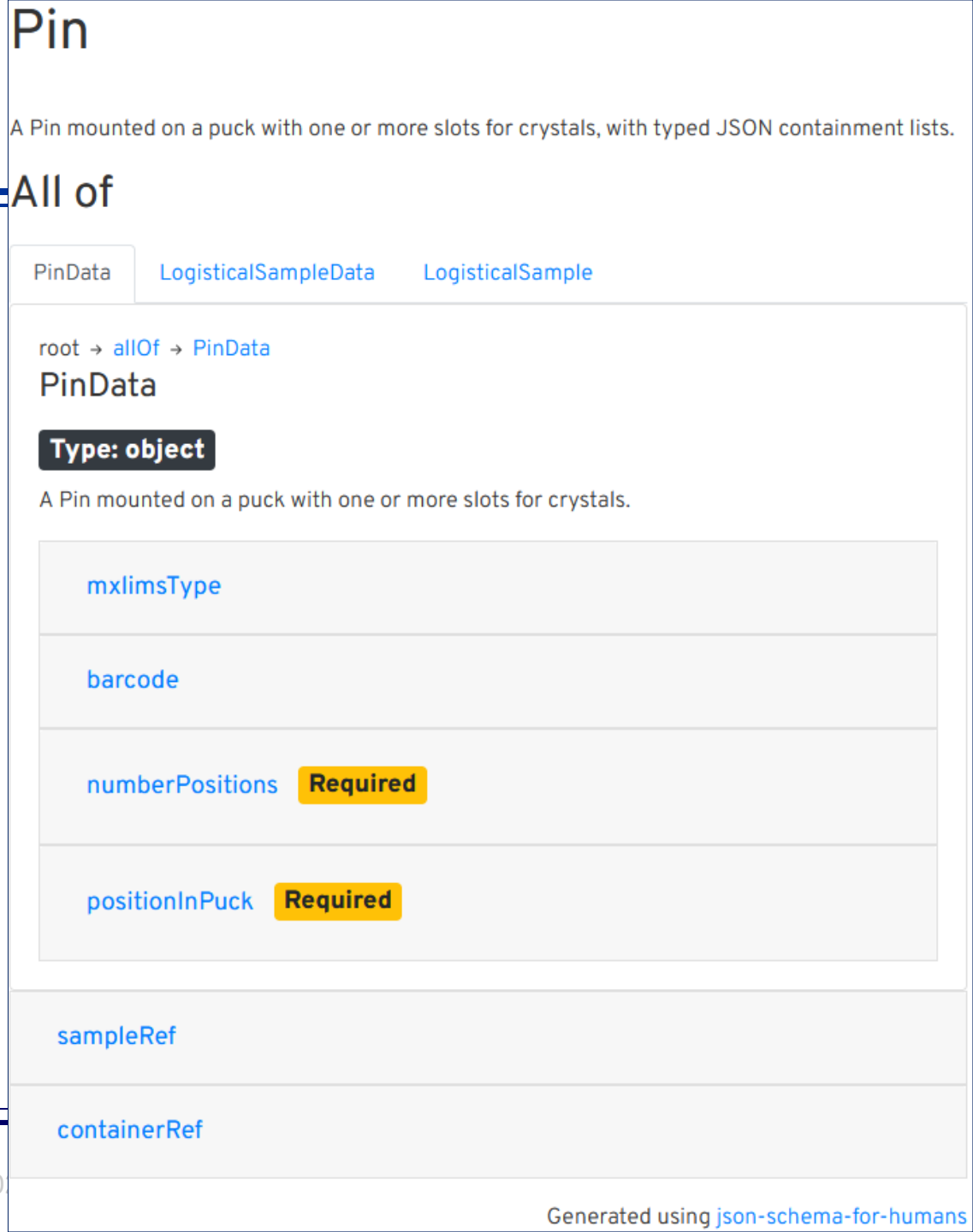
Specification – schemas

core/	*Core classes (with foreign keys). In-memory* only
data/	*(Meta)data fields for each class*
datatypes/	*Independent data types (Unitcell, Scan, enums, …)*
messages/	*JSON file schemas to specify/validate messages*
objects/	*Links (with types) for each class*
references/	*Reference-to-class objects. Autogenerated*

Global Phasing Limited

Pin JSON schema

- JSON schema for Pin class; Documentation autogenerated from schema
- The schema is combined from the schemas for PinData, LogisticalSampleData, and Logistical Sample.
- Inter-object links (sampleRef, containerRef) are set separately, with type information.



Pin

A Pin mounted on a puck with one or more slots for crystals, with typed JSON containment lists.

All of

PinData | LogisticalSampleData | LogisticalSample

root → allOf → PinData

PinData

Type: object

A Pin mounted on a puck with one or more slots for crystals.

- mxlimsType
- barcode
- numberPositions **Required**
- positionInPuck **Required**
- sampleRef
- containerRef

Generated using json-schema-for-humans

GΦL
Global Phasing Limited

Object specification - Pin

```
{    "$schema": "https://json-schema.org/draft-07/schema",
   "description": "A Pin mounted on a puck with one or more slots for crystals",
   "title": "Pin",
   "type": "object",
   "allOf": [       {"$ref": "../data/PinData.json"},
                    {"$ref“: ../data/LogisticalSampleData.json},
                    {"$ref": "../objects/LogisticalSample.json"}    ],
   "properties": {
      "sampleRef": {"allOf": [
                              {"reverseLinkName": "logisticalSamples",
                              {"$ref": "../references/CrystallographicSampleRef.json"}  ]},
      "containerRef": {"allOf":[
                               {"reverseLinkName": "contents"},
                               {$ref": "../references/PuckRef.json"}]}
}}
```

GΦL
Global Phasing Limited

Implementation - Pydantic

- Core, data, datatypes, references generated by JSON-Pydantic conversion
- Objects, messages generated by bespoke scripts
- Implementation class contains all persistence and link handling
  - Currently just four UUID:object dictionaries
  - Could be replaced simply by proper database access

GΦL
Global Phasing Limited

Pydantic class (API) - Pin

```
class Pin(PinData, LogisticalSampleData, LogisticalSample, MxlimsImplementation):
    def __init__(self, **data: Any) -> None:
        super().__init__(**data)
        MxlimsImplementation.__init__(self)

    def container(self) -> Optional[Puck]:
    @container.setter   def container(self, value: Optional[Puck]):

    def contents(self) -> list[PinPosition]:
    @contents.setter    def contents(self, values: list[PinPosition]):

    def datasets(self) -> list[Union[CollectionSweep, ReflectionSet]]:
    @datasets.setter    def datasets(self, values:
                                     list[Union[CollectionSweep, ReflectionSet]]):

    def jobs(self) -> list[Union[MxExperiment, MxProcessing]]:
    @jobs.setter    def jobs(self, values: list[Union[MxExperiment, MxProcessing]]):


    def sample(self) -> Optional[CrystallographicSample]:
    @sample.setter    def sample(self, value: Optional[CrystallographicSample]):
```

GΦL
Global Phasing Limited

Messages

- All objects are kept at the top level of the message
- Inter-object links are handled using JSON $ref
  e.g.

```
"sourceRef": {
   "mxlimsType": "MxExperiment",
   "$ref":
"/MxExperiment/86bf8e30-6c5b-4851-859c-7366ab575f49"
            }
```

- Links to objects not in the message (when permitted) are handled using object stubs. This allows JSON validation fo all links and their type.

Global Phasing Limited

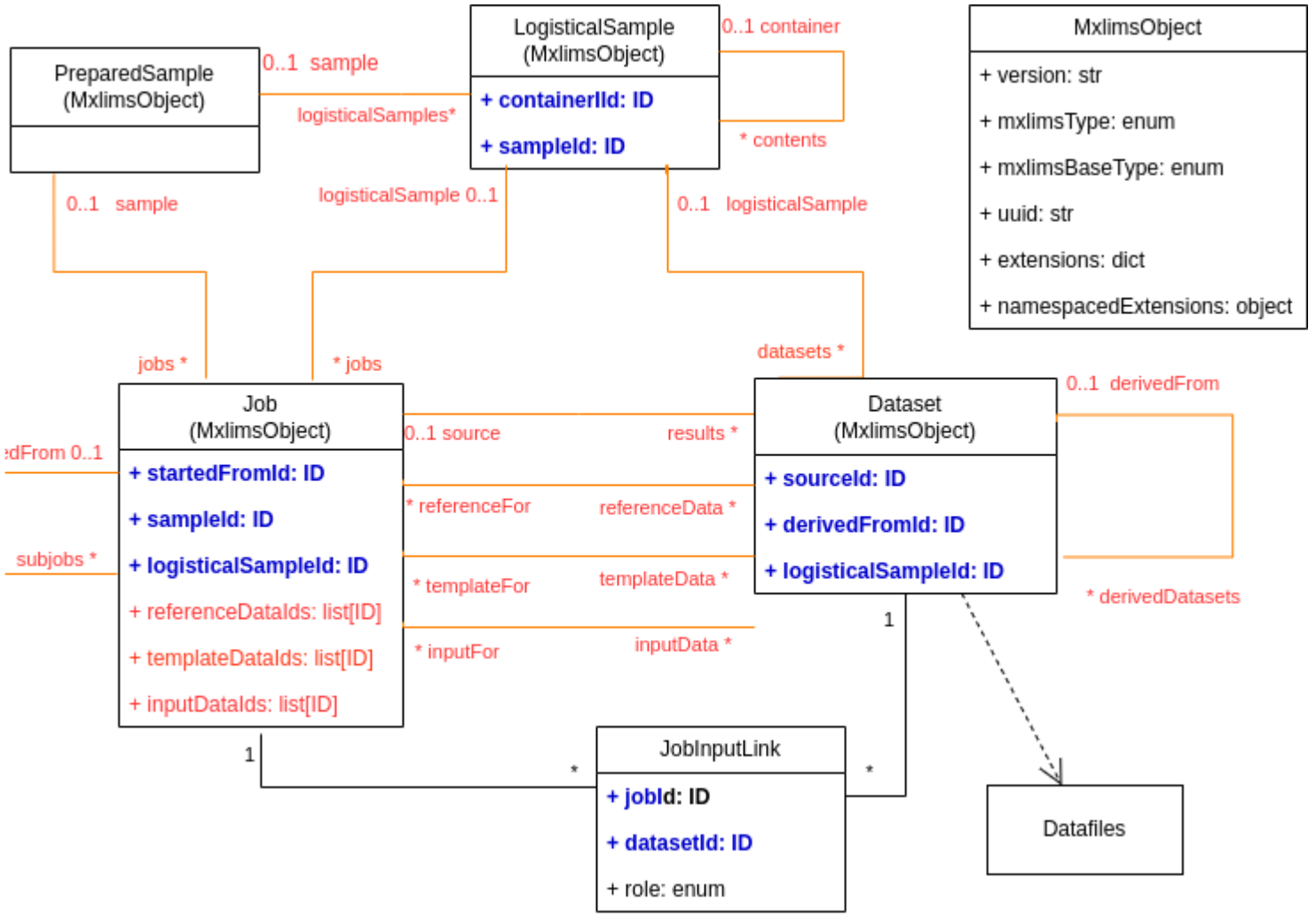
Message organisation

```
{

    “CollectionSweep":
    { uuid : CollectionSweep }
    ”Crystal“:
    { uuid : Crystal }
    ”CrystallographicSample“:                                                { uuid : CrystallographicSample }
    “Dewar”:
    { uuid : Dewar }
    ”DropRegion“:
    { uuid : DropRegion }
    ”MxExperiment“:
    { uuid : MxExperiment }
    ”MxProcessing“:
    { uuid : MxProcessing }
    ”Pin“:
    { uuid : Pin }
    ”PinPosition“:
    { uuid : PinPosition }
    ”Plate“:
    { uuid : Plate }
    ”PlateWell“:
    { uuid : PlateWell }
    ”Puck“:
    { uuid : Puck }
    ”ReflectionSet“:
    { uuid : ReflectionSet }
    ”Shipment“:
    { uuid : Shipment }
    ”WellDrop“:
    { uuid : WellDrop }


  “Dataset“:
    { uuid : DatasetStub }
    “Job“:
    { uuid : JobStub }
    “LogisticalSample“:
    { uuid : LogisticalSampleStub }
    “PreparedSample”:
    { uuid : PreparedSampleStub}


}
```

Global Phasing Limited

Database implementation

- Many-to-many links are handled by a link table
- Foreign keys (in blue) are stored in the database; links and e.g. `Job.referenceDataIds` are supported in the API.



GΦL
Global Phasing Limited

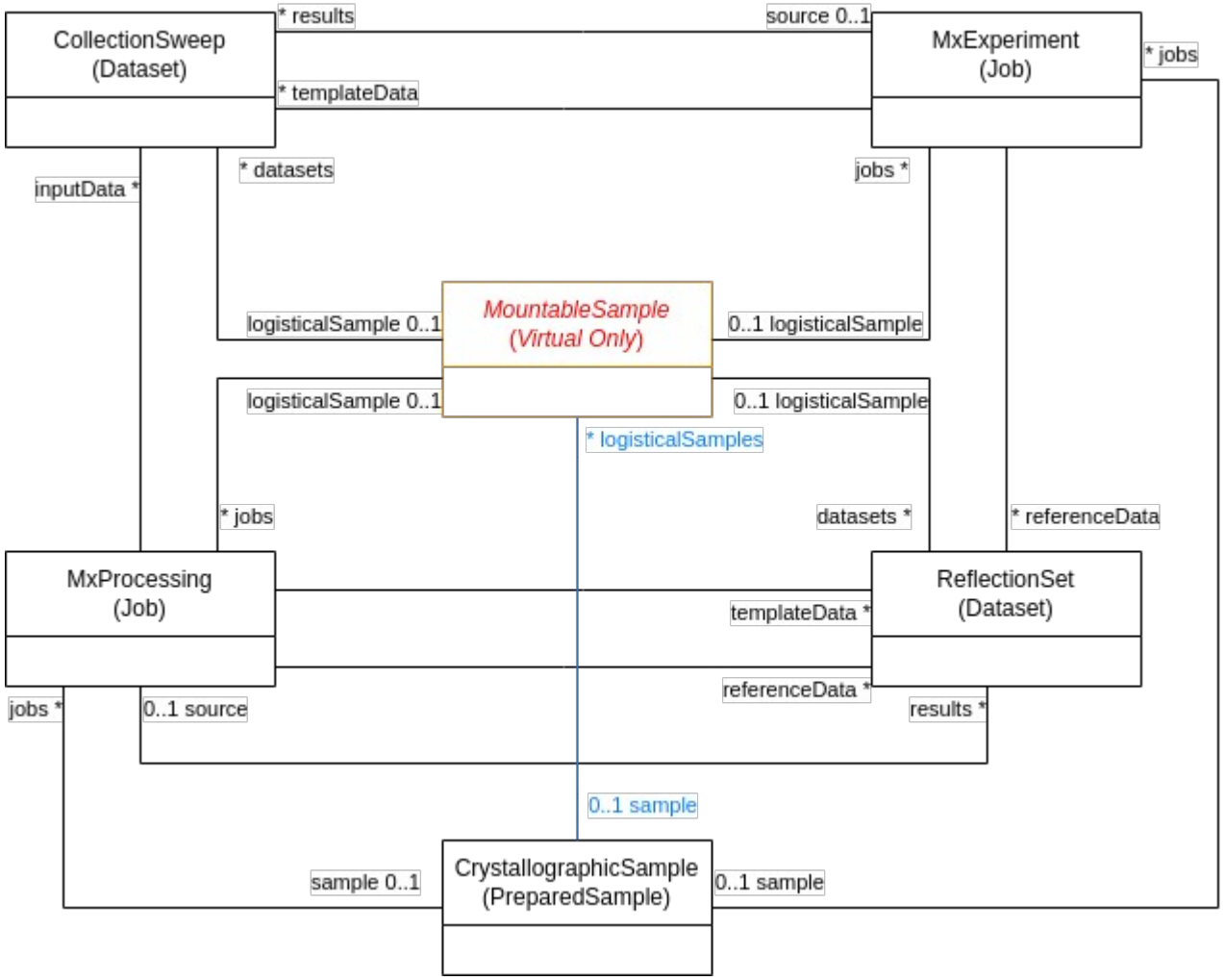
Model Contents

A quick overview of the classes in the (current) model

Global Phasing Limited

Model classes 1

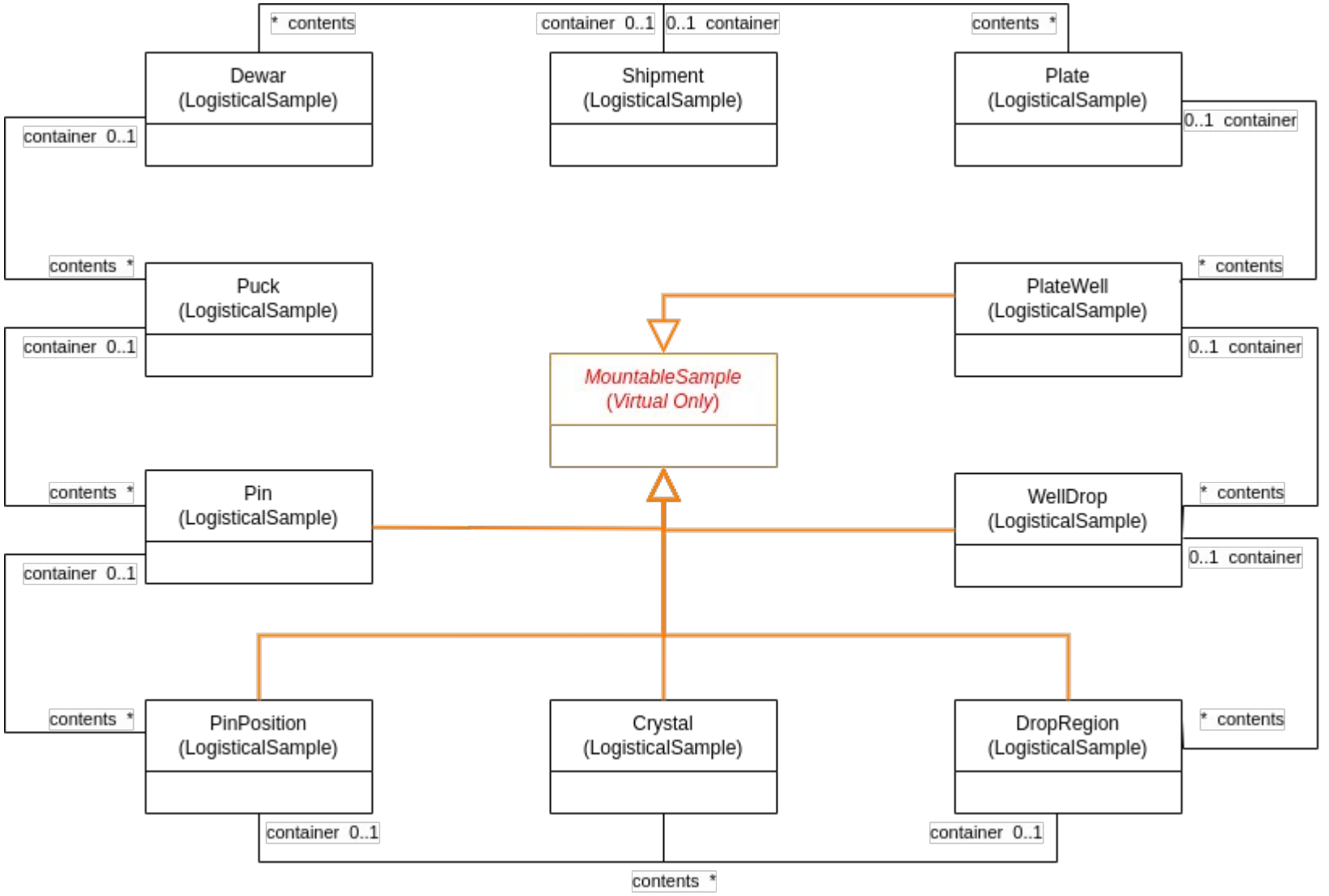
- Jobs,Datasets,and PreparedSamples, and their links.
- Jobs and Datasets have LogisticalSamples (e.g. crystals) but only Jobs have PreparedSamples
- Note that the type of results is different for MxExperiments and MxProcessing, and that only the latter has InputData.



Global Phasing Limited

Model classes 2

- LogisticalSamples and shipping, with container hierarchy
- Only some types can be mounted as the subject of an experiment, and so can be linked to Jobs.
- Crystals may be discovered only during the experiment (SSX, LCP, …)



GΦL
Global Phasing Limited

Illustrative model details

CrystallographicSample (MxlimsObject)
+ name: string
+ macromolecule: SampleComponent
+ components: list[SampleComponent]
+ spaceGroupName: string
+ unitCell: UnitCell
+ identifiers: Dict[str, str]

MxExperiment (MxlimsObject)
+ experimentStrategy: string
+ expectedResolution: float
+ doseBudget: float
+ measuredFlux: float

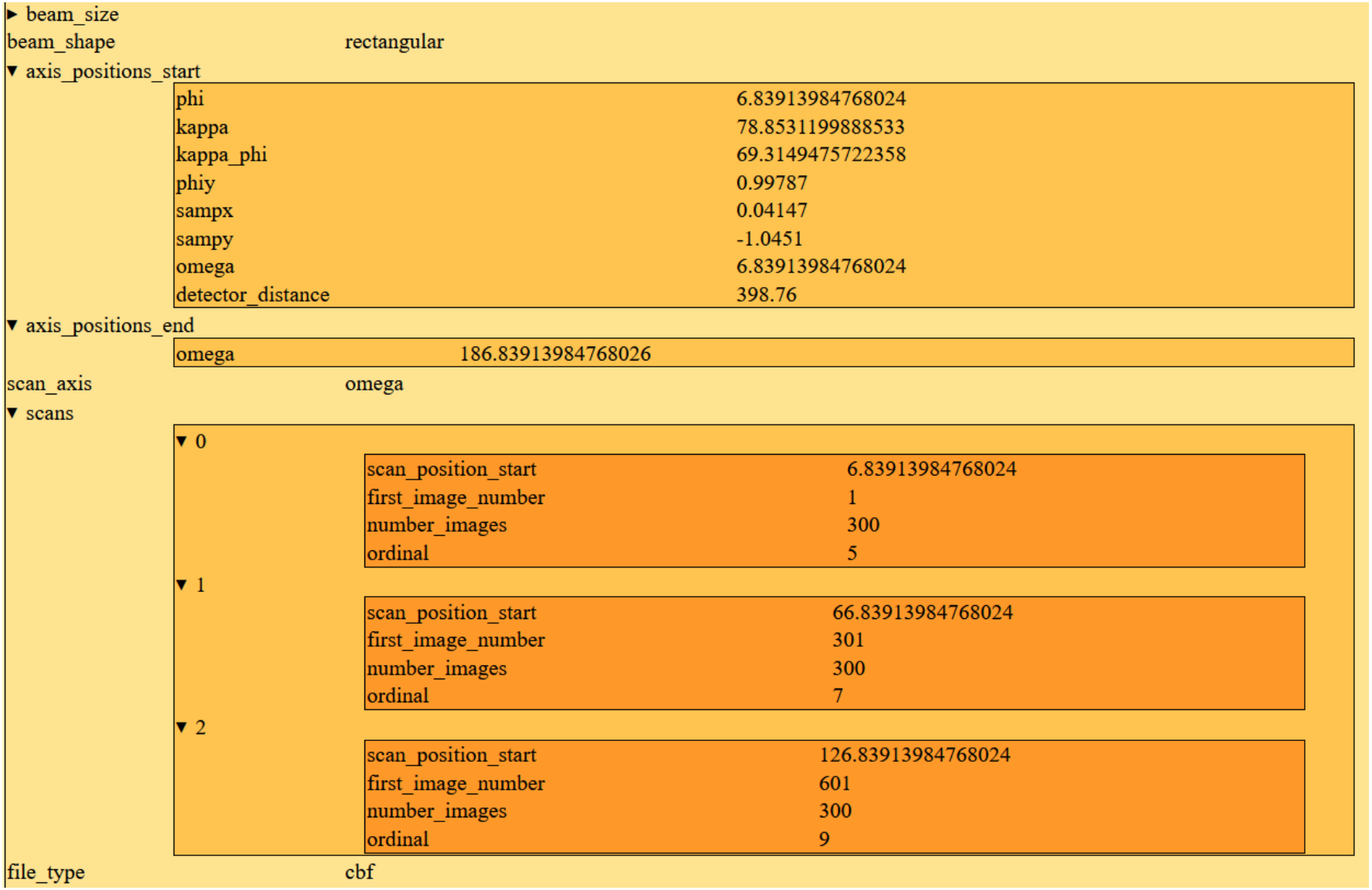
CollectionSweep (MxlimsObject)
+ role: string
+ imageWidth: float
+ exposureTime: float
+ energy: float
+ numberImages: int
+ numberTriggers: int
+ numberLines: int
+ meshRange: [float, float]
+ filenameTemplate: string
+ path: string

ReflectionSet (MxlimsObject)
+ role: string
+ unitCell: UnitCell
+ spaceGroupName: string
+ operationalResolution: float
+ diffractionLimitsEstimated: [float, float, float]
+ wavelengths: List[float]
+ numberReflections: int
+ reflectionStatisticsOverall: ReflectionStatistics
+ reflectionStatisticsShells: List[ReflectionStatistics]
+ binningMode: string
+ filename: string
+ path: string

Global Phasing Limited

Interleaving: sweeps and scans

- A CollectionSweep is a range of images with start and end positions.
- Sweeps are divided into scans, that may be acquired interleaved (as here) or out of order.
- Axis_position_end can have more motors (e.g. for helical scans or grid scans).



Global Phasing Limited

Topics

Questions we might discuss

GΦL
Global Phasing Limited

In-memory implementation and LIMS

- The prototype implementation has no access control, keeps objects alive as long as the program runs, and does not store them.
- What would be acceptable as a minimal implementation for use in e.g. MXCuBE?
- How could MXLIMS be integrated with (existing) LIMS systems?

GΦL
Global Phasing Limited

Experiment control input

- The model envisages using MxExperiments and template Datasets to pass in diffraction plan information. Worth a look?
- What modeling is missing before we can use MXLIMS to control acquisition queue input?
  - X-ray centring is currently under discussion in the MXCuBE Automation WG
  - The model allows jobs to be nested as subjobs. Can/should we organise e.g. centring, characterization and acquisition as separate subjobs under a master job, similar to DataCollectionGroups?

GΦL
Global Phasing Limited

Combining data and processing

- How can orientation_ids, merge_ids etc. already in use be mapped to the MXLIMS model, and what new attributes are needed??
- How to use MXLIMS for specifying processing parameters, and for reporting and comparing processing results?
- How to represent multiple processing programs, single- v. multi-sweep processing, and in general LIMS system result displays?
- How to deal with combining sweeps from different samples and crystals, e.g. for SSX

GΦL
Global Phasing Limited

Organisation

- In the short term MXLIMS should become a coalition of the willing (much like MXCuBE developers, maybe). That would then require some practical agreements on version control, how to approve changes, maybe a shared-ownership Github repository.
- In the long term a collaborative standard cannot be owned and controlled by a single participant (no, not even Global Phasing ;-)). What kind of set-up could we come to?

GΦL
Global Phasing Limited

Acknowledgements

- The main sources of inspiration for the model so far have been mmCIF, the IceBear data model, ICAT, the MXCuBE queue model with workflows, autoPROC multisweep data processing, working group discussions on use cases for SSX and complex samples.

- Particular thanks to Ed Daniel for all his input in the modeling

- Thanks to the many, many people who have participated in the MXLIMS working group, the MXCuBE automation WG, or contributed information, discussions and feedback.