1 Introduction

This document describes the principles and structure behind the xia2dpa data model. This is necessary because the problem is a complicated one. The basic principle here is to have a global "data repository" which is structured enough to have a hierarchy of data but simple enough that a value of something can be found easily.

This structure will need to have a few basic properties:

- Objects have two "parts" - an immutable identity and a set of defined properties which may be allowed to vary with time.
- Changes to objects should be recorded as updates, with earlier instances being kept. An example follows.
- Getting a mutable property for an immutable object will delegate the getting to an immutable child.

The example which follows is:

```
d = Dataset('12287_1_E1_001.img') -> Populate identity from the image
                                     headers for this data set. Initialise
                                     metadata about this dataset, and
                                     record the creation epoch.

d.index() -> Autoindex the data set. This task will be delegated to an
             autoindexer, which will take as an argument the dataset object.
             The results will be recorded in a list of autoindexing
             solutions and the latest instance in this list returned.
             This method will also check for assertions which may be
             relevant, for instance that the lattice is monoclinic.
             This assertion will also have an associated epoch. If ...

d.getCell() -> Is called and the assertion for the lattice is more recent
               than the source object that getCell gets the information from,
               the source object (in this case the indexing solution) needs
               to be reevaluated. This will mean that any further processing
               based on these results may need to be repeated.
```

The upshot of this is that if I autoindex the data set, process and find in cell refinement that the refinement breaks (or process in triclinic, and check the point group) I can assert that the lattice is something different. The next "get" method will then verify that it's information is up-to-date and if not will KNOW how to make it so.

This is going to get complicated, but is a fascinating way of working. It will mean that the knowledge on how to update objects will have to be delegated to the objects.

This comes back to the overarching idea that the main() routine in this could almost look like:

```
processing_results = Dataset('12287_1_E1_001.img').getProcessing_results()

... or almost ...

structure = Model(sequence = 1vpj.pir,
                  phases = Phase(Dataset({frames:['infl_001.img',
                                                  'lrem_lr_001.img',
                                                  'lrem_001.img'],
                                         id:[(0.9790, fp, fpp),
                                             (1.0002, fp, fpp)]}))
                  ).getStructure()
```

... taking this to the obvious conclusion, the objects would have ONLY get methods - everything else would be passed in through the constructor, and all actions would be implied by the get methods. Note that "private" methods would be needed in order to implement the result discovery delegation but this would be relatively doable.

Time to get a second brain fitted then. This is beginning to look a little like hard-core C++ programming.

This then means that the whole architecture is almost programmed in a functional manner[1]. Cool. It also means that the schema for the objects is of relatively little interest, though it would be handy to have some lightweight objects for handling this kind of information.

1.1 Hierarchy

This principle really works when ideas of hierarchy are included. For instance, the first pass at processing the first data set (by definition the reference) will result in one unit cell. This will then be set as the "globally correct" one until more information is available. When another set at this wavelength is processed, a weighted value for the unit cell may be derived from both sets. When combining all of the data, an overall unit cell may be computed from all of the available data sets.

This means, by implication, that each data processing "run" will have an associated local instance of a global data store. When it comes to merging or combining data sets, a further data store will be necessary. Does this mean that each stage is considered as it's own project, resulting in an idea that the global project data repository, referred to above, is some kind of weighted average of all of the local data repositories. Think of this like a tree-code.

2 Items of Interest

2.1 User Input

At the beginning, all that is known is what the user has passed in on the command line. From this a small number of simple things can be derived. In the example above, an example is the relationship between datasets and f values - e.g. wavelength for an image is read from the header. This is then used to associate different sweeps collected at the same wavelength and so on...

So - an object is needed to hold the information passed in by the user. Frameworks are then needed to communicate this information to the constructors as described above, perhaps as part of a dictionary which the constructor can interrogate as necessary.

[1] http://en.wikipedia.org/wiki/Functional_programming

The following items will be minimally necessary to make this work:

- Frames to use, defined by a frame from the set.
- ?Correct beam centre.
- ?Wavelength, f', f" values if available.

2.2 Learning Things

From the basic information derived from the command line, some extra information can be learned. For example:

- Lattice, unit cell.
- Spacegroup?
- Resolution.

At any stage any derived information is a hypothesis. There will be degrees of reliability associated with each of these, and when asked for the most recent (by definition most reliable) instance should be returned.