from Configuration Control Board

Proposal for Metadata For Image-Like Objects

The main text of this proposal has not been changed significantly. Some clarifications have been added, particularly with regard to persistence of _extraMetadata items, as a result of the Middleware/Apps? Task Force meeting (see below).

The Present Design

We have the following image-like objects:

- Image
- Mask: an integer mask with named bit planes
- MaskedImage: an Image with associated variance and mask
- Exposure: a MaskedImage with associated WCS and possibly other metadata

In the present design Image and Mask both contain a _metaData field that is a DataProperty of name:value metadata pairs. If the Image or Mask is read from a FITS file then _metaData contains the associated FITS header records. MaskedImage adds no additional metadata. Exposure adds WCS (as a member variable), and the notes about Exposure in the EA model imply that additional metadata will be added as an Exposure is processed.

Issues With The Present Design

Duplication of Information

When an Exposure is read from FITS, three FITS files are read: one each for the image, variance and mask. The FITS header keywords are read into the associated _metaData for each of these objects. Some of this metadata is already duplicated. Then the WCS information in the image's _metaData (but not variance or mask) is interpreted and a WCS object is created. Thus we end up with at least two copies of the WCS information: one in the image's _metaData and one in the WCS object. If the variance and/or mask contain WCS information, that adds additional copies that do not necessarily agree with the first two.

In addition, when an Image is read from FITS then the image size and type are contained in _metaData even though they are redundant with the type of and size member variables of the associated Image and Mask objects.

Problems arise when the data needs to be altered, e.g. while making a subimage. The code may not alter both sets of metadata leaving an inconsistency. This is dangerous.

Other Issues

Accessing data in a DataProperty is clumsier and slower than accessing a member variable. When accessing data in a DataProperty, spelling errors become essentially undetectable. If the data item is stored or retrieved with the wrong name then the data appears to be missing.

On the other hand, it is probably going too far to expect all metadata to consist of member variables. Use cases for a DataProperty of name:value metadata include:

- Offers ready access to nonstandard FITS keywords, e.g. for images from other surveys
- People have argued for the ability to add arbitrary metadata "on the fly" for quickly solving problems without having to alter or subclass fundamental classes (though this can easily be abused).

Proposed Changes

Metadata In Exposure, Not Image

To avoid duplication, only Exposure will contain metadata. Image and Mask will contain no metadata beyond the fundamentals required by the class (e.g. the static mask plane dictionary for Mask).

Exposure will contain metadata that is appropriate to science and template images.

Image-like data that has no obvious associated variance and mask but requires metadata will be implemented as a new class. Two rather poor examples are Kernels and PSFs (poor because such objects are not always image-based but may also be function-based). Such classes may contain an Image or inherit from Image, as appropriate.

This is probably the most controversial part of the proposal. For more discussion see Issues With the Proposed Design:No Metadata for Images below.

Standard Metadata in Member Variables

To take advantage of C++ static type checking, standard LSST metadata will be in member variables (which may be simple data types or complex objects such as WCS).

Additional nonstandard metadata will be supported as name:value pairs in a DataProperty member variable named _extraMetadata (or some such name that discourages use).

This nonstandard metadata will be used for code that is experimental, in flux or intended for short-term use. It will also allow us to carry nonstandard metadata from non-LSST surveys. Such metadata will be persisted to FITS files (if possible) but probably not to the database (see "Should _extraMetadata Be Persisted To the Database?" below).

Duplication is forbidden: a given Exposure must never have two different metadata entries for the same datum (e.g. a member variable and _extraMetadata entry representing the same metadatum). We must also be careful to avoid partially overlapping metadata.

Validate Metadata While Setting It

Metadata sanity checking should be handled by the appropriate constructor or function that sets the data. Users of the data should not have to check the data themselves, other than making sure the data exists, if appropriate (for example an Exposure gains metadata as it makes its way through the pipeline).

FITS I/O Tweaks

It would be nice if Exposures could be persisted to single FITS files, e.g. using multi-extension FITS. This would keep the data together.

It would be nice if FITS input be done as a constructor for Exposure (and Image and Mask) rather than as a method on an existing object. This makes it a bit easier to read a FITS file and is arguably a natural fit to the idea of reading in a FITS file as an Exposure (or Image or Mask). However, K-T notes that this may not be practical.

Eliminate MaskedImage

We should consider eliminating the MaskedImage class and using Exposure instead. These two classes are very similar, differing mainly in whether they contain exposure-related metadata. Furthermore, the main use case for MaskedImage is to contain exposure data, so the metadata is wanted.

The safest thing to do for now is probably modify Exposure to subclass MaskedImage rather than contain a MaskedImage. This gives users of Exposure more direct access to the image data and eliminates the need to deal with the MaskedImage class directly. We can then check later to see if there are any users of MaskedImage and if not, we can remove it without modifying any existing code.

Prototype Design

This is a prototype design showing the member variables for Exposure and its related classes. Note that I am not sure how to implement pure provenance (information we want to save, but that is not needed for data processing). It would be nice to keep them out of Exposure if we can reasonably do so (to avoid clutter). In any case, I list provenance items and borderline cases separately for that reason.

Image has:

- data: pointer to 0,0 pixel in array
- y stride

RawImage inherits from Image and adds a bit of metadata (this metadata could go into Exposure, but it complicates data processing so I'd rather add this class or transmit the information separately on the clipboard):

- data section (Region): section of image array that contains real data (binned pixels)
- bias section (Region): section of image array that contains overscan bias (binned pixels)

Mask has:

- data: pointer to 0,0 pixel in array
- y stride
- mask plane dictionary (a class variable, so common to all instances of Mask)

MaskedImage has:

- image (Image)
- variance (Image)

- mask (Mask)

Exposure inherits from (rather than its current design of containing) MaskedImage and adds lots of metadata:

- TAI MJD date of start of exposure (double). This may be provenance. Note that it will be slightly different for each amplifier because the shutter moves with finite speed.
- subframe origin (int[2]): index of (0,0) pixel of this Exposure within the full CCD (unbinned)
- bin factor (int[2])
- filter (int?)
- exposure time (double)
- amplifier info, an object that includes:
    - ♦ amplifier ID (int?); useful for crosstalk correction
    - ♦ size (unbinned pixels)
    - ♦ amplifier gain (double)
    - ♦ amplifier read noise (double)
    - ♦ amplifier bias (double)
    - ♦ position and orientation of amplifier in the focal plane
- exposure type code (int); e.g. science frame, bias frame, dark frame, dome flat, sky flat...
- photometric zeropoint (double)
- WCS (shared_ptr to WCS)
- ISR output (an object); this is any ISR output that will be wanted for later data processing (if there is any; much of it maybe provenance). This will be defined by Nicole.
- extraMetadata (a DataProperty); see metadata proposal

    the following are possible additional member variables, but I think they are too specialized and too closely allied with the sky and heavens:

- sky model (shared_ptr to Function or other object)
- PSF model (shared_ptr to PSF)

    borderline metadata/provenance cases:

- RA/Dec and orientation on sky of center of focal plane. I would exclude this from Exposure since WCS contains the same information in a more directly useful form.
- bad pixel mask ID (int?)
- CCD ID or info. This is likely to be a field of AmpInfo, in which case we have it in Exposure whether we want it or not. But I doubt data processing will have any use for it.
- ISR output that is provenance (an object); defined by Nicole.

TemplateExposure inherits from Exposure and adds:

- Component info (shared_ptr to object or combination of objects): which science images were used to create the template along with data such as weighting factors and PSFs. Some of this will be pure provenance. Some of it will be needed when using the template.

(This metadata could go into Exposure, but I'd rather not. There may be several kinds of Template, each a unique kind of component information.)

Notes:

- metadata that is not yet known or is irrelevant to the particular type of exposure will, of course, be indicated by nan or a null pointer.
- (int?) indicates some int-like data type or possibly the type of the relevant primary key field in the database.

Issues With the Proposed Design

No Metadata for Images

Plain Images will have no metadata (beyond the bare minimum required). This means that there is no object that maps directly to a single image FITS file. This has several implications:

- It may be useful to be able to write or read diagnostic FITS images of objects such as Kernels and PSFs with arbitrary bits of FITS header information, and if so, it is be more convenient and more natural to write such data as Images rather than Exposures.
- To read in data from other surveys one would have to read in an Exposure with a blank variance and mask. However, this is no great concern because the first operation in an LSST pipeline would be to compute that variance and mask.

One possible variant design that addresses this concern is as follows:

> Put the _extraMetadata member variable (see above) in Image instead of Exposure. As above, standard metadata is placed in member variables in Exposure (except those bits required to create the two Images and Mask in the Exposure).
>
> Just as above, when reading an image, variance and mask FITS file into an Exposure only the FITS header information from the image component is used. Thus variance._extraMetadata will thus start out empty, and one hopes it will be left that way.

One gains a bit of flexibility for reading and writing single images. The cost is that metadata for Exposures is divided between Exposure member variables (for the standard data) and the image component's _extraMetadata (for the rest). Also the Exposure gains an undesirable extra _extraMetadata DataProperty in the variance Image.

A variant is to have a subclass of Image that contain metadata and are used to represent FITS and FITS-like files. Such an object could be used to save an "image with metadata" and for reading in FITS files, but would not be normally used within the pipeline. We may want something like this to represent FITS MEF files in any case.

Object-Relational Mappers

In commenting on an earlier version of this proposal, Tim Axelrod wrote:

> In looking toward a viable design, it's fair to note that when image metadata is stored, it naturally seems to end up with a table (keyword - value) representation. This is true not only for FITS, but also for AVM and XMP, both of which we will probably have to deal with in our interface to the public.
>
> To me, this sounds very close to the problem that object-relational mappers (ORM) are

> intended to solve. So my suggestion is that we look at the capabilities of the ones that are out there and at a minimum use them for design ideas. What we are trying to do is pretty simple compared to what some of the ORM solutions are designed for. And particularly given that we are already using ORM for persistence (even though we decided to ditch the one we were using), it might not be too much of a stretch to use that technology here as well.
>
> The proposal does not use an ORM. Presumably this makes it more complicated to persist these classes. How much more complicated? Enough so to justify using an ORM? My own fear is that using an ORM will make the metadata itself too hard to use.

RO talked to K-T about this; here is RO's summary of K-T's response:

- It is important to be able to have metadata stored as member variables. Thus persistence needs to support that (as well as DataProperty name:value metadata).
- If it turns out that we should use an ORM for persistence we can apply the necessary changes to the classes without changing the API.
- Thus we can safely agree on a metadata handling proposal without deciding on whether to use an ORM for persistence.

Should _extraMetadata Be Persisted To the Database?

K-T noted that in the present design, only standard known entries from Image._metaData are persisted to the database; other fields are ignored. (FITS files are different; all entries are persisted to FITS files, if possible). In the new design, to enable round-tripping of FITS files from non-LSST sources, we will persist all _extraMetadata values to a separate table, likely as strings, and retrieve them when the object is retrieved.

Extracted from notes of 26-Aug-2008 Middleware/Applications? Task Force telecon

Discussion:

- Proposal: Instance variables for most metadata. Persistence Formatter maps file/database form to/from instance variables. DataProperty for new items or items that are not processed by code.
- TA agrees that Formatter is the equivalent of the adapter he was seeking.
- Can't get rid of the DataProperty; similarly, can't get rid of the instance variables. Do need to enforce consistency between the two unless there is no duplication.
- Non-LSST metadata needs to be preserved.
- May need to maintain readability of prior releases' data using later releases' software.
- Boundary between instance variables and DataProperty may vary with time.
- Analysts will want to annotate data arbitrarily at any time; camera may add metadata arbitrarily.
- Would like to have a single interface for accessing metadata, both from code and from SQL. Looks like it is impossible to do from SQL with an RDBMS. Could do from code accessing metadata in objects (with some difficulty) or from code accessing the database (relatively easily).
- FITS keywords need not be read into a DataProperty first, although that may be the simplest implementation, since many will end up there.

Decisions:

- Have instance variables and a separate DataProperty as in the proposal. These are mutually exclusive; no metadata is in both.
- Formatter converts to/from FITS keywords and database columns.
- Round-trip from input FITS to output FITS must be guaranteed not to lose any information (may gain some), even if object is persisted to a database in between.
- We will investigate whether we should seek to define a "namespace" for LSST-specific keywords in FITS files.

Comment by TA: I agree with all the points above, and offer the following scenario to make explicit some of the concerns surrounding the shifting boundary between instance variables and DataProperty:

- The Camera team decides after the survey has been running for a while that there is a significant parameter they want to add to the image metadata, which they call "TEMP32". They add it, and as a result, there is a new "TEMP32" item in the DataProperty that goes into the database as the metadata associated with every image. This addition does not require any software change for DM.
- Data quality analysts create various plots and metrics that utilize "TEMP32", which they access via an SQL query involving the image metadata (no, I don't know exactly how to do that either...) (KTL: Something like "`SELECT CAST(value AS DECIMAL(10,6)) FROM ImageMetadata WHERE key = 'TEMP32' AND imageId = ...`". We could also provide a view that makes known entries in the ImageMetadata table look like columns in the view, but this could be expensive. Finally, we could provide a `getMetadata` stored procedure that would hide the table location.)
- Six months later, Nicole decides that she can use "TEMP32" to compensate for a CTE effect that has been a problem for the ISR, so she promotes "TEMP32" to be an instance variable of the Image class. She tries it out, and decides to include this in a new release of the Nightly pipeline software. (KTL: This would also require a new release of the `ImageFormatter`, or `ExposureFormatter`.)

Now we have a couple of problems:

- Since "TEMP32" has been promoted to an instance variable, the DB schema has to include a new column for the Image table (really one of the Exposure tables, if we are sticking to the schema as it is). Somehow we have to accommodate this schema change "on the fly", by which I mean that it happens between data releases. (KTL: This would involve yet another release of the `ExposureFormatter`, a new column in the database, and a query to transform all of the entries in `ImageMetadata` into entries in the new column.)
- Jacek and KT wave their magic wands and change the schema, but now the DQ analysts, and all those external scientists (who have also gotten addicted to using "TEMP32") find all their software is broken, since "TEMP32" is no longer in the Image metadata. (KTL: Unless they were using the view or stored procedure with SQL or a similar `getMetadata()` method on Exposure.)

Extracted from notes of July 27, 08 Applications phonecon

Discussion of the Proposal for Metadata For Image-Like Objects

(please feel free to post corrections and additions; it was hard to keep up with the discussion and write notes at the same time)

Russell wants this proposal discussed and agreed to as soon as possible so we can design the ISR with it in mind. To that end we spent the rest of the meeting discussing it.

- Does it make sense for LSST standard metadata to be represented as metadata-specific member variables, rather than as name:value pairs in a general DataProperty. (Note: name:value DataProperty metadata will still be supported for experimental code, test code, etc.)

Andy Becker worries about needing to modify the class every time we add a bit of metadata.

On the whole people seemed to agree that this was acceptable as long as the updates occurred occasionally and at specific times (that we could run for awhile using new metadata in name:value pairs).

- Should new metadata should be persisted in the database and if so, how?

The obvious way to handle new metadata is to add it to a Character Large Object (CLOB) or something similar. This gets the data *into* the database, but it is not easily queried.

Jacek proposed that if we initially use a CLOB for new metadata that we actually want to persist (much of it can be reconstructed and so may not need to be persisted). Then during the next data release we turn that metadata into new database fields (which *can* easily be queried).

The consensus on persisting seemed to be that Jacek's proposal was the best solution, offering a good balance of power, ease of use and ease of implementation.

However, K-T brought up the issue of backwards compatibility issues cause by the database changing:

- queries on newer databases may not work on older databases.
- There are compatibility issues for persistence, and thus running different versions of the LSST software on different versions of the data release database. Everyone agreed this is a serious issue that needs further thought. But it does seem to be largely independent of the image metadata proposal.

- Should the Image and Mask classes have metadata (beyond the bare minimum required for the classes)?

Advantages to not having metadata (all metadata in Exposure only):

- There is only one obvious place to look for the metadata
- If one gets an Image out of an Exposure there is no issue about whether to copy the metadata from the Exposure to the Image

Disadvantages:

- There is no afw object that is a really direct representation of a simple FITS file. Thus it is a bit harder to read and write simple FITS images (e.g. have to carry around the variance and mask of the Exposure in afw).

In the end it was agreed that if the applications developers were comfortable giving up the metadata in Image and Mask then it was fine. Nicole, Andy and Russell all feel it is OK, but Andy would like to give Tim Axelrod a chance to weigh in.

- Should we keep MaskedImage or combine it with Exposure into one class? Also, do we want more than one kind of Exposure?

Nicole and Andy could not think of any use case for a bare MaskedImage, but neither they nor Russell have carefully looked through the EA model to be sure.

K-T then asked if we should have different kinds of Exposures for different kinds of images, e.g. bias vs science image. This would be sensible if the metadata is very different (especially if the metadata is in instance variables). Andy felt that the different kinds of images weren't different enough to justify this.

Ray pointed out that we can avoid the need for different kinds of Exposures if we use a DataProperty for most metadata (as we do now). In other words, using dynamically typed metadata.

However, Andy was pretty sure we only need one kind of Exposure in any case.

The consensus was that we probably only need one kind of Exposure and that we can probably do without MaskedImage. But given the uncertainties it makes sense to change Exposure to inherit from MaskedImage (rather than contain a MaskedImage, as now) and prefer Exposure to maskedImage and see how that works out. If it works well then we can get rid of Exposure with minimal pain later.

- What about obscure metadata that is only needed by one or two tiny bits of code?

K-T brought up the issue that there is a lot of metadata associated with an image and some of it may only be needed by one bit of code, perhaps only sometimes. He pointed out that there was a proposal for lazy evaluation of such metadata and he felt this would still be a good idea. He proposed that we add a new metadata member variable "lazyMetadata" for such metadata (since it would probably want to be some new kind of object).

We wrapped up by summarizing the conclusions:

- Get rid of metadata in Image provided Tim agrees.
- Avoid duplicate metadata. There should only be one obvious place to look for an item of metadata and it should never be duplicated in a given object.
- Combine MaskedImage and Exposure only if MaskedImage truly not needed and there is only one kind of Exposure. (Change Exposure to inherit from MaskedImage otherwise).
- It is OK to put standard LSST metadata in member variables. But we probably need to add a new lazyMetadata member variable.

Comments by Tim Axelrod on phonecon discussion

re: Does it make sense for LSST standard metadata to be represented as metadata-specific member variables?

I think the separation of metadata into class members and a list of DataProperties? needlessly complicates a situation which is intrinsically complicated anyway. It offers yet another opportunity for a given metadata item to be represented in two locations with conflicting values. Changing the class definition every time we decide we want another metadata item usefully available will lead to a maintenance nightmare, as others have suggested.

It still seems to me that we need a uniform way of making all image metadata available to a class, and a list of DataProperties? is not a bad way to do it. I recognize the convenience of having some frequently used metadata values available as class members. Perhaps this could be done in a way that avoids the maintenance issues by constructing some sort of cache of these values from the DataProperty list when a class member is

constructed. The cache entries would point to the relevant list members, thus avoiding the need to search the list whenever you need something. I haven't thought this through, and it has some obvious problems - but perhaps it has a useful germ of an idea.

- Robert Lupton comments on above

```
We need to think about what we want (input) metadata for.  I'd much
rather have type-safe descriptions of well-defined concepts than
just push them all into some dictionary.

An example is WCS.  It happens to be commonly persisted as fits
keywords, but it really isn't dictionary-style metadata at all;
it's some class that describes the geometry.  So we unpersist (?)
it from the header, DELETE those keys, and create a WCS.

I'd argue that bad-pixel information from the camera team should
be treated the same way --- they should be providing data on the
bad columns in a defined way, not writing metadata to headers.

So there will be some dictionary-style free form metadata (which
needs to be captured to the DB), but I'd home very little.
```

re: Should new metadata should be persisted in the database and if so, how?

I emphatically agree that the metadata needs to be persisted in the database. Nothing useful to add on implementation, I'm afraid.

re: Should the Image and Mask classes have metadata (beyond the bare minimum required for the classes)

This seems hard to work with. Consider this scenario: The camera team creates a Mask that has planes for bad pixels, charge traps, etc. That Mask naturally has metadata, perhaps lots of it, including parameters that were used to run the bad pixel finding algorithms, etc. I now want to create a MaskedImage from that Mask and a raw camera image, and I reasonably expect to have all the relevant metadata accessible. How do I do that in the proposed scheme?

re: What about obscure metadata that is only needed by one or two tiny bits of code?

I haven't yet seen the proposal for lazy evaluation, but I feel pretty strongly that we want a uniform way of handling metadata. What is relevant for my bit of code is irrelevant for yours and vice versa. We do not want to have endless discussions over what metadata goes where.

Add comment

Your email or username: