Gen3 Butler Composites Design

This page has been completely rewritten from its original form, and the terminology has changed. In particular, because we've settled on an approach in which all datasets always have an entry in the Registry's Dataset table, we have repurposed "virtual" to mean something different.

Simplified Exposure as an Example

I'll use the Exposure StorageClass for most examples. For the purposes of this page, we'll assume it has the following simplified definition (pseudocode; not actual APIs):

```
Wcs = StorageClass(...)

Image = StorageClass(...)

MaskedImage = StorageClass(
    components={
        "image": Image,
        "variance": Image
    },
    ...
)

Exposure = StorageClass(
    components={
        "maskedImage": MaskedImage,
        "wcs": Wcs,
    },
    ...
)
```

Terminology

composite: a Dataset or DatasetType whose StorageClass defines a set of discrete named child datasets, called components

parent: synonym for composite

component: a Dataset or DatasetType that may be accessed as a child of a composite (in some cases may also be accessed in other ways)

child: synonym for component

virtual: a Dataset or DatasetType that is defined by its relationship to one or more other Datasets/DatasetTypes.

concrete: not virtual

immediate: all content is written in a single call to `Butler.put`

deferred: content is written via multiple calls to `Butler.put` (possibly in different SuperTasks) and associated later (possibly in yet another SuperTask) via a call to `Butler.link`.

This leads to four fundamental kinds of Dataset[Type]s:

- **concrete** (always immediate, may be a component or a composite or neither): this is a regular Dataset that is itself written by a single `Datastore.put` call and read by a single `Datastore.get` call. Writing a concrete composite also creates virtual components. Examples: a WCS written on its own, an Exposure written all at once into a single file[1], or a Wcs written to its own file when writing an Exposure to multiple files.
- **virtual component** (always immediate): a Dataset defined automatically when its parent Dataset is written. Example: the Wcs of Exposure written at once into a single file[1].
- **immediate virtual composite:** a Dataset written at once by recursively writing its components. Example: an Exposure written by writing all of its components into separate files (which could include writing an immediate virtual composite MaskedImage that in turn writes concrete Image and Variance).
- **deferred virtual composite:** a Dataset defined by associating pre-existing Datasets that are interpreted as its components. Example: an Exposure written by combining a MaskedImage from a different Exposure with a Wcs that was written on its own.

1. I'm saying "file" rather than "Dataset" here (and below) both to provide a clearer example and because I think concrete composite vs. immediate virtual composites is how we'll want to implement single-file Exposure vs. multi-file Exposure. This design doesn't actually guarantee that a Datastore will write a concrete composite as a single file (after all, it could even write some/all of it to a SQL database instead), but Datastores that actually do write files shouldn't *need* to be able to split up concrete composites into multiple files themselves. The design does guarantee that an immediate virtual composite will not be written as a single file.

Principles

- All Datasets have an entry in the Registry's Dataset table. This implies that a composite Dataset is *more* than the sum of its components: it also includes (Registry) information to associate them.
- Any provenance graph (both what's recorded after processing and the QuantumGraph produced by Preflight) *must* contain nodes for both composite and components, because:
    - a SuperTask may consume only some components of a composite, so all component nodes must be in the graph;
    - a deferred virtual composite must be created explicitly, so it cannot be considered implicit in the graph;
    - whether a particular composite DatasetType is defined as concrete, virtual, immediate, or deferred should not affect what procesing is done, so we cannot include just some composite nodes in the graph.
- All information needed to read a Dataset is saved at the level of the Dataset (in some combination of Registry or Datastore). No information necessary for reading a Dataset is stored at a Datastore-wide or Registry-wide level, and it should never be necessary to configure a Butler a certain way in order to read something.
- It must be possible to change whether a particular DatasetType is written as a concrete composite or immediate virtual composite by changing only the Butler/Datastore configuration provided when initializing a client. The same is true for configuring a Dataset to be a deferred virtual composite, though of course this also requires Butler users (e.g. SuperTasks) to use `link` as well as `put` to create it.
    - As a result, any composite StorageClass must be writeable as concrete (with virtual components), immediate virtual, or deferred virtual; the StorageClass itself shall not be specialized to one of these choices.
    - It should not be necessary to change persistent Registry or Datastore state (e.g. database tables or config files stored with the Datastore) in order to change how a composite DatasetType is written (it should be sufficient to change Butler/Datastore client information, though of course some configurations may be rejected by certain Datastores, and Datastores may provide persistent defaults for that configuration).

Permitted Combinations

1. A virtual component must be a part of exactly one concrete composite. For example, if Exposure A and Exposure B are each written as a single file, then A.wcs cannot also be a component of B.
2. A virtual component may be a part of one or more deferred virtual composites. For example, if Exposure A and Exposure B are each written as a single file, then an Exposure C may be defined such that C.wcs = A.wcs and C.maskedImage = B.maskedImage.
3. A virtual component may be a part of at most one immediate virtual composite, but only indirectly: it must be a component of a concrete composite that is in turn a component of the immediate virtual composite. For example, if Exposure D is a virtual immediate composite, and its maskedImage is concrete, writing D writes D.maskedImage to a single file (probably[1]), creating D.maskedImage.image, a virtual component.
4. A concrete composite always contains virtual components. For example, writing an Exposure A as a single file always implies that a A.wcs is a valid dataset (though it may be permitted to be None/null).
5. A concrete composite may not contain concrete components.
6. A concrete composite may not contain virtual composites.
7. An immediate virtual composite may contain one or more concrete components. For example, if Exposure D is an immediate virtual composite, its maskedImage and wcs components will be written (when D is put) as separate files.
8. An immediate virtual composite may contain one or more virtual components, but only indirectly (this is a restatement of (3)).
9. An immediate virtual composite may contain other immediate virtual composites. For example, if Exposure E is an immediate virtual composite, its maskedImage component may also be an immediate virtual composite, which means that the E.maskedImage.image and E.maskedImage.variance will each be written as a distinct concrete datasets (i.e. separate files) when E is put.
10. An immediate virtual composite may not contain deferred virtual composites. (Datasets are put or linked, but never both).
11. A deferred virtual composite may contain one or more concrete components. For example, we could write a stand-alone Wcs F, then later define an Exposure G such that G.wcs is F.
12. A deferred virtual composite may contain one or more virtual components. Those virtual components must still have concrete composite parents, but those concrete composite parents need not be children of the deferred virtual composite. This is a restatement of (2).
13. A deferred virtual composite may contain one or more immediate virtual composites. For example, we could write a MaskedImage H as an intermediate virtual composite, resulting in F.image and F.variance being written as separate files. We could then define an Exposure J such that J.maskedImage is H.
14. A deferred virtual composite may contain one or more other deferred virtual composites. For example, we could write two Images K and L, then define a MaskedImage M such that M.image=K and M.variance=L, and then define an Exposure N such that N.maskedImage=M (and N.image=K and N.variance=L).

Configuration

Registry

When a DatasetType with a composite StorageClass is declared to a Registry, DatasetTypes for each of the named components are also declared, with names constructed as "{parent-dataset-type-name}.{component-name}", the same DataUnits types as the parent, and StorageClasses that are the same as what would be used to write the component as a standalone Dataset.

For example (pseudocode):

```
# Given the StorageClasses defined above, this line:
registry.registerDatasetType(CalExp=DatasetType(StorageClass=Exposure, DataUnits=("Visit", "Sensor")))
# implies:
# registry.registerDatasetType(CalExp.wcs=DatasetType(StorageClass=Wcs, DataUnits=("Visit", "Sensor")))
# registry.registerDatasetType(CalExp.maskedImage=DatasetType(StorageClass=MaskedImage, DataUnits=("Visit",
"Sensor")))
```

As we'll see below, these component DatasetTypes will be used by virtual components of concrete composites and concrete components of immediate virtual composites, but will not be used for components of deferred virtual composites (because those will have already been written and added to the Registry using some other DatasetType).

Datastore/Butler

To configure a composite DatasetType X to be written (put) as a concrete composite with virtual components:

- Provide a write formatter for X itself.
- Provide a read formatter for X itself.
- Provide a read formatter for each X.<component> DatasetType that reads just that component from a file written using the write formatter for X.
- Write formatters for X.<component> must not be provided.

To configure a composite DatasetType X to be written (put) as an immediate virtual composite:

- Provide an assembler for X.
- Provide a disassembler for X.
- Provide a read and write formatter for each X.<component> DatasetType that read/write that component from/to its own file.

To configure a composite DatasetType X to be written (linked) as a deferred virtual composite:

- Provide an assembler for X.

Writing Datasets

Pseudocode, no error-handling/transactions:

```
def Butler.put(self, obj, datasetType, dataId, producer=None):
    """Write a dataset.

    May not be a virtual component or a deferred virtual composite.
    """
    datasetType = self.registry.getDatasetType(datasetType)  # argument may have just been a string; now it's
an object
    ref = self.registry.addDataset(datasetType, dataId, run=self.run, producer=producer)
    disassembler = self.config.getDisassembler(datasetType)
    if disassembler is not None:  # this is an immediate virtual composite
        childObjs = disassembler(obj)
        for childName, childDatasetType in datasetType.components.items():
            childRef = self.put(childObj, childDatasetType, dataId, producer=producer)
            self.registry.attachComponent(parent=ref, child=childRef)
        self.registry.setAssembler(ref, self.config.getAssembler(datasetType))  # Could also consider putting
this in Datastore
    else:   # this is concrete (and maybe a composite)
        for childName, childDatasetType in datasetType.components.items():  # if not a composite, loop body is
never executed
            childRef = self.registry.addDataset(childDatasetType, dataId, run=self.run, producer=producer)
                    self.registry.attachComponent(parent=ref, child=childRef)
        self.datastore.put(obj, ref)  # also associates readers with each component
    return ref

def Butler.link(self, datasetType, childRefs, dataId, producer=None):
    """Create a deferred virtual composite dataset by associating existing datasets.

    There are two link overloads; this one is more powerful but less convenient in the common case.
    """
    ref = self.registry.addDataset(datasetType, dataId, run=self.run, producer=producer)
    for childRef in childRefs:
        self.registry.attachComponent(ref, childRef)
    self.registry.setAssembler(ref, self.config.getAssembler(datasetType))

def Butler.link(self, datasetType, childDatasetTypes, dataId, producer=None):
    """Create a deferred virtual composite dataset by associating existing datasets.

    There are two link overloads; this one is less powerful but more convenient in the common case.
    """
    # Look up the DatasetRefs using the given DataID and then call the other overload.
        self.link(datasetType,
              [self.registry.find(childDatasetType, dataId) for childDatasetType in childDatasetTypes],
              dataId)


def Registry.addDataset(self, datasetType, dataId, run, producer=None):
    dataset_id, registry_id = self.execute("INSERT INTO Dataset ...")
    return DatasetRef(datasetType, dataId, dataset_id, registry_id, ...)

def Registry.attachComponent(self, parent, child):
    self.execute("INSERT INTO DatasetComposition (parent_dataset_id, parent_registry_id, component_name,
child_dataset_id, child_registry_id) ...")

def Registry.setAssembler(self, ref, assembler):
    self.execute("UPDATE Dataset SET assembler=? WHERE dataset_id=? AND registry_id=?", assembler.name, ref.
dataset_id, ref.registry_id)


def Datastore.put(self, obj, ref):
    # ... actually write a file or something ...
    self.registry.execute("INSERT INTO Storage (dataset_id, registry_id, datastore_name, md4, size) ...")
    self.addReader(ref, self.config.getReadFormatter(ref.datasetType))
        for child in children:
        self.addReader(...)

def Datastore.addReader(self, ref, formatter):
    # ... record somewhere that we should use the given formatter when asked to read back ref ...
```

Reading Datasets

Again, just pseudocode, no error handling:

```
def Butler.get(self, datasetType, dataId):
    ref = self.registry.find(datasetType, dataId)
    if ref.assembler is not None:   # virtual composite
        return ref.assembler({childName: self.datastore.get(childRef.dataset_id, childRef.registry_id)
                              for childName, childRef in ref.components.items()})
    else:  # virtual component or concrete
        return self.datastore.get(ref.dataset_id, ref.registry_id)


def Registry.find(self, datasetType, dataId):
    ref = DatasetRef(self.execute("SELECT * FROM Dataset WHERE dataset_type_name=? AND ...", datasetType.name,
dataId))
    if ref is None:
        # If this is a component of a deferred virtual composite, the Dataset table will have its original
        # dataset_type_name and data ID, and the above query will fail.  Instead we look for that composite
        # and then find the component by name.
        parentDatasetTypeName, componentName = datasetType.split()   # split the part before the first "." from
the part after it
        datasetEntry = self.execute("""
            SELECT Child.* FROM Dataset AS Child INNER JOIN DatasetComposition ON (...) INNER JOIN Dataset AS
Parent ON (...)
            WHERE DatasetComposition.component_name=? AND Parent.dataset_type_name=? AND Parent.
{data_unit_fields}=...
            """,
            componentName, parentDatasetTypeName, dataId
        )
    if datasetType.components:
        ref.components = {
            result["name"]: Dataset(result)
            for result in self.execute("""
                SELECT Child.* FROM Dataset AS Child INNER JOIN DatasetComposition ON (...)
                WHERE DatasetComposition.parent_dataset_id=? AND DatasetComposition.parent_registry_id=?
                """,
                ref.dataset_id, ref.registry_id
            )
        }
    return ref
```