

High-level analysis of changes indicated by the recent workshop at Fermilab.

Iteration one. – Dlp,

Article I. Summary:

This note is a high-level analysis of the changes to the “wrappers” recommended by the recent two day workshop held at Fermilab. Changes are needed to accommodate the new file tree, unique file naming, naming files to improve operability, and to give the system an controllable architecture which is needed to enable data management, and to make the system modular to cope with future changes.

We identify changes and find there is a way to incorporate the changes to the system in a way that allows gradual change.

Article II. Review of salient aspects of framework:

We can think of the processing system as having three components,

- 1) A central, unified data management and job management framework.
- 2) Some number of science codes, not all of which community code, that are not customized to fit into the data management system
- 3) “wrappers” – code which adapt science (and possibly other) codes to the framework.

This note focuses on a specification on the data interfaces between the framework and the wrappers.

DESDM framework is based on the exchange of data to and from wrappers via files and database interactions

DESDM provides a batch computing environment that is managed by the framework and runs science codes via their wrappers. The framework makes input files available to the codes. By convention

- "data and calibrations" are provided in a "file management system"
- Executable codes are provided by a "code management system".
- Configuration files are managed as code
- DB-resident data may be presented and manipulated by science codes.

In addition to supporting the science codes, the DESDM needs to support efficient operations over the lifetime of the survey. This means that the interface between wrappers and the framework must support the underlying evolution of DESDM.

Article III. Wrappers

A wrapper is a set of codes whose role includes adapting science code so that it interacts properly with the framework. A wrapper needs to tell us about the things it knows (but the framework does not know).

In particular,

- Wrappers shall emit information on the subset of data offered to them by the framework was actually used by the computation.
- Wrappers ensure that new files are declared to the system according to the rules of the framework.
- Wrappers emit detailed information about the transformations used on the data. e.g. what was the exact call to the wrapped science software?
- Wrapper must inform us about database tables read or updated.

There are other duties, which include: emitting Quality controls, and telling the framework to "halt". These are not discussed here.

DESDM requires data, calibration, configuration, and code provenance. While it is good to insert provenance information in file headers, DESDM needs the data to also be database resident. This is as much for data management operations, as it is for the scientific consumer of the data. Example use cases in operations are:

- Identify all data affected by a calibration.
- Identify all data affected by a version of a science code.
- Identify all data affected by a particular exposure.
- Identify all z-band data affected by a configuration file.
- Identify all data affected by a CCD over a certain date range.

- Identify all data processed with a command line switch.
- Identify which files have been modified in a processing step.

Section 1. Files:

For operations, DESDM locates files by recording a partial path name and basename for a file. Files retain this name when some (or all) of the file tree is present under a “rooted area” on a file system. However, the last portion of the file name is allowed to vary. This is to accommodate lossy and lossless compression, the idea being that such files contain equivalent information. The current framework supports fpack and gzip compression. Moreover, the pathnames are directly related to abstract properties of the files: “red” encodes “reduced” fileclass, etc. This linking fixes the file system to a structure, and makes it impossible to change as operational problems arise: e.g limiting the number of files in a directory, or merely navigating the system.

When a science code runs, the framework arranges for files to be provided to the wrapper. A “.des” file contains template commands to call a wrapper. The framework fills out the template at run-time. Part of the template language specifies queries that will identify the files required for this run of the wrapper. The framework runs these queries, and arranges for the files to be staged to a DES file tree that is available to the science code. The framework builds lists containing the file names, (and any other attributes that can be obtained from the database). These lists are passed to the wrapper.

As the wrapper and science codes run, new files are deposited into the DES file tree.

In the “as-is” system files in the tree may be “edited” typically by overwriting a file with a new file of an equivalent name.

In the as-is system, the new files are discovered after all computations in a “block” have finished. The framework runs codes which discover new files in a run, then applies > 4000 lines of business logic to find provenance relationships, extract and record meta-data and record these files in a replica catalog (the location table). This operation is also a considerable performance bottleneck. This bottleneck is not intrinsic to the problem, it is intrinsic to the constraints of the as-is system.

Section 2. Databases:

Basic information, such as “what database tables are read or updated by which programs” is lacking, and obtainable only by code reading or reading logs.

Section 3. Programs

The current system can record the invocation of the wrapper, but cannot record the invocation of the (community) science code. It's this invocation that is of interest to the DES community.

Section 4. Summary of concerns

- The system ties abstract qualities to concrete path components. This makes it impossible to address certain critical problems in the file tree.
- The Framework-resident code must figure out the parent-child relationships, extract meta-data, etc. This results in a large module with tragic coupling and cohesion characteristics. Maintenance of this code is at risk, and changes are slow enough that files are buried in unnatural places, (Likely obfuscating their abstract characteristics).
- File ingest is done when a block is done, and in particular is not done as a job finishes, so ingest information is not available for end-of-job processing by the framework.
- We need to understand in some routine way, what codes access what tables in the database.
- We need to capture the invocations of the community codes, not just the invocations of the wrappers.

Article IV. What provenance/metadata specification would make this better?

Disregarding whether the project could afford to implement this, what are the principles that should be pursued when investigating an alternative?

Section 1. Principles

Principle

The wrapper knows best – the

Addresses concern

Overly coupled, non-coherent

***wrapper knows the science, and actually knows what data was selected from the data made available to it and know what configuration files were uses. (Recall that the configuration files are managed as code.)
The wrapper knows which codes it invoked, and how they were invoked.***

It's settled computer science that Provenance for files can be transmitted as a set of 3-tuples, generically of the form (subject, relationship, object). This idea should be incorporated into the interface

Wrappers need to record provenance information in a way that's accessible to the framework. Recording must happen as the wrapper causes files to be moved or created or edited or replaced into the file management system file tree. Meta-data elements are recorded at the same time as provenance.

Meta-data elements are treated separately from path names.

Wrappers are presented with

software results in not keeping up with change requests, and de-facto mis-application of the existing system.

Address scaling and lack of performance of current file ingest.

A general interface allows for the system to be completed, and for a regular underlying implementation.

Currently, multiple inputs are handled only as special cases; not all cases are covered.

Allows that capture of configuration and code provenance can be symmetric to all other provenance collection.

The framework cannot understand what files in the tree are from which unit of computation. (e.g job) in real time, as unit of computation (or scale)

The framework needs information about files.

The current directory is inflexible, as path names are tied to certain, specific meta-data. We are unable to address present and future difficulties in the file tree. Compression is a function of

the uncompressed form of the data. If science-sensitive parameters are needed to lossy-compress the data, the wrapper transmits that information to the framework as meta-data. (needs thought?)

the framework, not science codes.

Other metadata is understood by the wrappers

File ingest needs to open most files, extract meta data from each file, and map the meta-data to the columns one of the following tables: IMAGE, SOURCE, COADD WL, SN, and CATALOG. Coupling and cohesion disease is rampant. The wrapper is in the best position to make this mapping

Section 2. Provenance Specifications

The basic notion for provenance is inspired by RDF -- provenance is declared to the system in (subject, relationship, object) triples. Subjects and objects are names of things in or used by DESDM. Relationships form a finite vocabulary of terms that relate these things together.

In full blown RDF these terms are qualified with a URL-like syntax, this is to prevent name collisions when people develop things independently, We do not need this for DESDM. (but can generate this if required, and in particular can dump our provenance to already-developed full blown provenance engine where we can get answers to questions like the ones in the use cases, above.

The framework is in a position to collect further provenance, which might include useful information about the detailed configuration of the software sandbox, and the computing platform used,

While further research is warranted, We are unaware of any standards for astronomy production that we can conform to (further research is warranted). Our strawman for the as-is system is as follows.

Section 3. Subjects and Objects.

Its possible to develop a rich and increasingly fine grained granularity of specification. Remember that this method of specification is flexible, and that we should be guided by the fact that we currently have broken model and that a coarse grained system would gain us a lot. We also need to be conscious of the intellectual burden placed on the (distributed) set of people who maintain wrappers. We have to present the simplest thing that is not too simple, and shield wrapper writers from any work that is best done in the framework.

Here is a strawman of the things a wrapper is in the best position to acquire data about, and also satisfy our immediate requirements:

Thing	<i>How named in API (Proposed)</i>	<i>Comment</i>
<i>File in file management system</i>	<i>Canonical name</i>	<i>What we consider to be "data"; covers files not well treated in the current system, like flats bias calibration frames Includes things like input and output lists passed to science programs.</i>
<i>Config file from source tree</i>	<i>Base name of the file in distribution.</i>	<i>Allows us to find data affected by a specific configuration file.</i>
<i>Program in source tree</i>	<i>Base name of file in distribution.</i>	<i>Name the program that converted inputs to provides the outputs.</i>
<i>Commandline</i>	<i>The text of the full commandline used to invoke a science code.</i>	<i>Name the command line used to invoke the science code.</i>
<i>Database table</i>	<i>Name of table.</i>	<i>Allows u to know what code touched what table, something that is not under control</i>

today

Relationships:

These are the relationship that a wrapper needs to concern itself about.

Relationship	Valid Subjects	Valid Objects	Comment
Read	Program	File in the file management system	Program read file in the file management system
Produced	Program	File in file management System	Program made a file with a new canonical name.
Edited	Program	File in file management system	Program altered or replaced a existing file in the file management sistem.
Was-Configured	Program	Configuration file in the source kit	File was used as configuration for program.
Was-Invoked_as	Program	Commandline	Commandline used to invoke program
Inserted_into	Program	Database table	Program added one or more rows to named table
Updated_table	Program	Database table	Program edited one or more rows to table

Section 4. Other Meta-Data

For each FITS file, the current system extracts and maps data from the header into a generic table that most closely matches the FITS file. This mapping is best performed by the wrapper, which knows the type of the file, and knows how to transform the data to the form required by the

table. This meta-data is held in the IMAGE, COADD, CATALOG, WL, SN, and EXPOSURE tables.

Other meta-data is currently hard-wired into the file paths. If we wish to decouple paths from this data, the meta-data must be supplied by external API as well. If we have a gradual change in the framework, some of this meta-data is needed to form path-names in the way the current system works, and so the framework can marshal data computed in the new framework and provide it to “old style” wrappers -- so that change can be incremental, and old wrappers can work until they are re-written.

This meta-data that is currently related to the “path names’ or “file names” is in the LOCATION table. The relevant sub-set is

column_name	Comments
FILECLASS	Describes DES fileclasses
FILETYPE	Describes DESDM filetypes
RUN	DES pipeline run identifier. Of the form: CCYYMMDDHHMMSS_NITE
NITE	Observing nite data were taken
BAND	Filter used in exposure: g,r,i,z,Y
TILENAME	DES Tilename
EXPOSURENAME	Filename of exposure image current image was derived from. The exposure is the MEF fits file generated at the telescope.
CCD,NUMBER	CCD number of current image or catalog
PROJECT	Project file was taken for: DES, BCS, SCS, etc...

Article V. What about other issues?

Section 1. Looping in wrappers

There are a number of concerns:

- Log files and other forensic data are more complex, since they are large and record information about many program invocations. Trolling these files is arduous, and having multiple invocations makes the work harder. So forensics is harder than need be.
- It that it is unclear what happens when a program fails on some processing attempts, but not others.

These factors indicate that there is a trade off of computational efficiency, compared to operability that needs to be further studied. This proposal could be adapted to w looping or loop-less system.

Section 2. Reaction to partial errors, and data quality:

The existing framework is used at various grain-sizes. Core data products use many blocks and many modules. Super Nova processing uses one block and one module and one wrapper invocation. E.g. the framework invokes a independently constructed workflow system designed by the super nova team.

So the interesting case is “what role does the framework have to offer when modules and blocks are chained together?”

In the as-is system, essential meta-data is placed in one of a small number of tables, eg. source, catalog, image, etc. These tables are available to the queries which are executed by the framework to marshall files for a block. It is worth noting that the WL group noted that some data may be suitable for many survey purposes, but not suitable for WL. So there are real issues, we may have co-add runs for WL, and other runs for the general collaboration, where the salient difference is the suitability of the input images.

Exposing the writing of this meta-data to the wrappers allows for quality data to be interested into these tables, and used by orchestration queries. However fitting arbitrary quality data into a static table seems problematic, and needs to be investigated.

Adding a (<file>, has_quality, <quality-literal>) tuple to the provenance system,

Allowing wrappers to provide this data and supporting queries against this information might provide a better hook.

Article VI. How can we move to a new system, given it is so late?

The strategy that comes to mind is incremental change. What are the hooks for incremental change? They seem to be:

- Directories above, and including file type are currently controlled by the framework, and do not affect current wrappers.
- If we proceed in an upstream to downstream way, as long as we collect the meta-data in the location table, we can make files appear with the pathnames used by old wrappers.
- We can neuter out specific kinds of files in file-ingest, as they are entered in the new framework.

This suggests that migration is feasible if we proceed in an upstream to downstream manner, where we modify wrappers, orchestration and migrate to the new ingest framework.