

Gen2 Middleware Removal Planning

The Gen3 versions of the AP and DRP pipelines are already able to do at least as much as their Gen2 counterparts, and we currently have no reason to believe that they perform any worse at anything.

We also now have the green light to start removing the Gen2 versions of things, at least according to the release stability promised we've made in the past.

At present, however, some important test coverage still relies on Gen2 middleware and `CmdLineTasks` - this is a problem in its own right, because it means we're testing things under unrealistic, non-production conditions, and reliance on those tests make it harder to change our tasks in Gen3-only ways. This page attempts to capture all cases in which we use Gen2 as a test harness, and transform this information into a plan for removing different aspects of the Gen2 middleware.

1.1. Gen2 removal tickets

Tickets that involve removing Gen2 functionality should be given the label "gen2-removal" in Jira. That will make them show up here for easier tracking (until they're done).

"Blocks" and "Is blocked by" links should be used to capture dependencies between removal tickets. Linking removal tickets elsewhere in this page is also encouraged, especially if they are blockers for other removal tickets.

Key	Summary	T	Created	Updated	Due	Assignee	Reporter	P	Status	Resolution
No issues found										

2. Ongoing DRP Gen2-Gen3 parity testing


We've found some major bugs in the Gen3 pipelines only by checking for consistency with Gen2. Are we done with that effort? If not, what remains?

This is a blocker for removing any Gen2 task run in DRP, and a blocker for removing Gen2 obs package support for any instrument being used for these parity checks.

Everything else on this page assumes this blocker has been removed; it needs to be updated with more specific blockers in later sections if this work will have a long tail, instead of being a blocker for all Gen2 removal work.


2.1.1.1. Comments from Lauren

Summary: as of w_2022_03 (most significantly post-merge of tickets:



DM-30284 - Jira project doesn't exist or you don't have permission to view it.

&




DM-33195 - Jira project doesn't exist or you don't have permission to view it.

for the coaddition/multiband stages) I am ready to

declare "effective parity" between the Gen2 and Gen3 middlewares for "vanilla" processing though multiBand (by which I mean not considering external global calibrations, namely FGCM & jointcal).

Details (getting into the nitty gritty):


The above declaration is based on detailed comparisons of the outputs from the Gen2 & Gen3 processing runs. For the DC2 test-med-1 dataset, the regular processing runs have been used for the comparisons, where we only do the "vanilla" processing. For the HSC-RC2 dataset, both FGCM and jointcal are applied as global calibrations, so these runs can't be used to test the "vanilla" processing (I'll say more about this below). So, for HCS data, I made my comparisons on the ci_hsc_gen2 vs. ci_hsc_gen3 outputs (for which external calibrations do not get run separately, but rather the same pre-computed corrections get applied in both, so they cannot contribute to differences). As noted on

 DM-30284 - Jira project doesn't exist or you don't have permission to view it.


, with only a few minor exceptions (more details below),

"effective parity" was achieved between the two ci_hsc runs. For LSSTCam-imSim, the most recent w_2022_03 runs of the DC2 test-med-1 dataset are

the comparisons in play (

 DM-33258 - Jira project doesn't exist or you don't have permission to view it.

&

 DM-33223 - Jira project doesn't exist or you don't have permission to view it.

). I first had a look at the pipe_analysis plots for all

stages. Visit-level parity has long been established and maintained, but up until now, there were noticeable differences in the coadd-level plots. In this run, blinking the coaddAnalysis plots for the two runs reveals that they are all identical. These can be found at

Gen2: https://lsst.ncsa.illinois.edu/~lauren/DC2.2i_gen2/w_2022_03/plots/

Gen3: https://lsst.ncsa.illinois.edu/~lauren/DC2.2i_gen2/w_2022_03/vsGen3/plots/

As an example: if one blinks between (CModel fluxes):

https://lsst.ncsa.illinois.edu/~lauren/DC2.2i_gen2/w_2022_03/plots/r/tract-3829/plot-t3829-r-mag_modelfit_CModel_forced-psfMagHist.png

and

https://lsst.ncsa.illinois.edu/~lauren/DC2.2i_gen2/w_2022_03/vsGen3/plots/r/tract-3829/plot-t3829-r-mag_modelfit_CModel_forced-psfMagHist.png

or between (stellar locus):

https://lsst.ncsa.illinois.edu/~lauren/DC2.2i_gen2/w_2022_03/plots/color/tract-3829/plot-t3829-griPSF-wFit-fit.png

https://lsst.ncsa.illinois.edu/~lauren/DC2.2i_gen2/w_2022_03/vsGen3/plots/color/tract-3829/plot-t3829-griPSF-wFit-fit.png

the only difference is between the labels which indicate the repo locations (and the photoCal label which couldn't be easily hacked for the Gen3 repos in these scripts).

I've gone through many such (random-ish) blinkings and have spotted no differences. However, and this is where the "effective" qualifier on parity comes in, when I looked at the much more sensitive compareCoadd plots (where I match the catalogs and do a direct difference of the measured quantities), the most minor of cracks were revealed. For example, in

https://lsst.ncsa.illinois.edu/~lauren/DC2.2i_gen2/w_2022_03/vsGen3/plots/r/tract-3829/compare-t3829-r-diff_modelfit_CModel-psfMagHist.png

you can see a small smattering of points that do not lie on the zero line. The number of outliers and their relative magnitudes are VERY small, so certainly of no scientific concern, but I still wanted to see where the difference kicks in. It turns out it goes back to at least the initial deepCoadd image (and possibly the directWarps, but I haven't run the script on all of those..the few I have come out identical, but there are a LOT of visits!), where, when I compare pixel by pixel, a given patch may have a small number of pixels that differ between the two. For example, for tract 3829, patch 5, r-band, my script tells me the following:

```
compareCoadd INFO: Comparing gen2 vs. gen3 deepCoadd image/mask/variance arrays and photoCalib's
compareCoadd WARN: ...patch 5: Pixel differences [-1.4901161e-08 -2.2204460e-16]
compareCoadd WARN: ...patch 5: Number of differing pixels 2 (out of 17220000)
```

i.e. there are two pixels with non-zero differences, and the difference is at the $\sim 10^{-8}$ level! This can then propagate into the deepCoadd_calexp image and into the catalogs. For the former, the differences are usually very similar (the number of differing pixels often, but not always, matches that of the deepCoadd). For the latter I see differences along the lines:

```

compareCoadd INFO: Comparing gen2 vs. gen3 deepCoadd_meas catalogs
compareCoadd WARN: ...patch 5: coord_ra Absolute diff (mas): mean: 1.0474261503092998e-09 min: 0.000 max 0.000
compareCoadd WARN: ...patch 5: coord_ra Number of differences: 2 of 34019 total
compareCoadd WARN: ...patch 5: coord_dec Absolute diff (mas): mean: 2.621257987984842e-09 min: 0.000 max 0.000
compareCoadd WARN: ...patch 5: coord_dec Number of differences: 2 of 34019 total
...
compareCoadd WARN: ...patch 5: base_SdssCentroid_x Absolute diff : mean: 4.169468700129304e-12 min: 0.000 max 0.000
compareCoadd WARN: ...patch 5: base_SdssCentroid_x Number of differences: 2 of 34019 total
compareCoadd WARN: ...patch 5: base_SdssCentroid_y Absolute diff : mean: 1.3141649038340044e-11 min: 0.000 max 0.000
compareCoadd WARN: ...patch 5: base_SdssCentroid_y Number of differences: 2 of 34019 total
...
compareCoadd WARN: ...patch 5: base_PsfFlux_instFlux Absolute diff : mean: 9.330902442036606e-11 min: 0.000 max 0.000
compareCoadd WARN: ...patch 5: base_PsfFlux_instFlux Number of differences: 16 of 34019 total
compareCoadd WARN: ...patch 5: base_PsfFlux_instFluxErr Absolute diff : mean: 2.8554662708866644e-14 min: 0.000 max 0.000
compareCoadd WARN: ...patch 5: base_PsfFlux_instFluxErr Number of differences: 2 of 34019 total
...
i.e. those two tiny image pixel differences led to tiny differences in the measurements for two of the catalog entries.

```

I will note that in almost all cases, one thing that differs between the catalogs for the two middlewares is any column measuring a "time" value (so, e.g.

```

compareCoadd INFO: Comparing gen2 vs. gen3 deepCoadd_meas catalogs
compareCoadd WARN: ...patch 0: deblend_runtime Absolute diff : mean: 250.34848022460938 min: 0.000 max 98489.484
compareCoadd WARN: ...patch 0: deblend_runtime Number of differences: 11010 of 28673 total
compareCoadd WARN: ...patch 0: modelfit_CModel_initial_time Absolute diff : mean: 0.0004544018065776166 min: 0.000 max 0.125
compareCoadd WARN: ...patch 0: modelfit_CModel_initial_time Number of differences: 28549 of 28673 total
compareCoadd WARN: ...patch 0: modelfit_CModel_exp_time Absolute diff : mean: 0.00037350294702333204 min: 0.000 max 0.024
compareCoadd WARN: ...patch 0: modelfit_CModel_exp_time Number of differences: 28488 of 28673 total
compareCoadd WARN: ...patch 0: modelfit_CModel_dev_time Absolute diff : mean: 0.0006463836361734035 min: 0.000 max 0.162
compareCoadd WARN: ...patch 0: modelfit_CModel_dev_time Number of differences: 28488 of 28673 total
compareCoadd WARN: ...patch 1: deblend_runtime Absolute diff : mean: 277.5063781738281 min: 0.000 max 189626.109
compareCoadd WARN: ...patch 1: deblend_runtime Number of differences: 11021 of 27570 total

```

but I suspect those are unimportant (could be architecture-dependent?)

And that's it! As for the FGCM/jointcal comparisons, a lot of effort has been put into making these as close as they can get between the two middlewares, so I am expecting very close, but not exact parity (e.g. there is at least one input in FGCM that we simply can't match exactly). This will get born out in the current RC2 runs with w_2022_04 at which point we can make a final(?) call on effective parity for all the processing steps considered here.

Oh, one final note: I have only done some spot-checking comparing the rolled up and DPDD-ified parquet tables in detail. The per-detector/per-patch non DPDD versions are already effectively tested since those are used in the plotting scripts. So far, the only differences I've found are a few different column entries;

objectTables: the Gen3 has one column entry that the Gen2 does not: {'skymap'}

sourceTables: Gen3 has {'band', 'instrument', 'physical_filter', 'visit_system'}

Gen2 has {'detectorName', 'expId', 'raftName', 'run'}

UPDATE (Feb 21, 2022): as of



DM-29820 - Jira project doesn't exist or you don't have permission to view it.

, both Eli Rykoff and I are

ready to pull the plug on Gen2 support for FGCM.

I (Lauren) have also signed off on jointcal, pending review of



DM-33760 - Jira project doesn't exist or you don't have permission to view it.

UPDATE (Feb 25, 2022): As of the review of the above ticket, John Parejko and I agree that we are ready to pull the plug on Gen2 support for jointcal.

This concludes my Gen2/Gen3 science parity testing for all stages from single frame through multiBand included in the LSSTCam-imSim/DC2 and HSC-RC2 regular processing runs. As such, Gen2 versions of such processing are no longer necessary and Gen2 support for all relevant tasks can be removed from this perspective (i.e. the rollout for Gen2 removal detailed below is no longer blocked by science parity testing as far as I, Lauren MacArthur, am concerned).

3. Known Gen2 integration tests and multi-task unit tests

3.1. ci_hsc_gen2

This:

1. Runs the full Gen2 DRP pipeline on HSC data.
2. Tests that the expected output datasets are actually produced and meet very minimal expectations.
3. Runs and tests the Gen2 Gen3 repository conversion tools.
4. Tests the pipe.base.ShimButler transition convenience code.

(1) is already done by `ci_hsc_gen3`.

(2) is not, and this is probably the single biggest blocker for removing `CmdLineTask` inheritance from most hybrid `PipelineTask/CmdLineTasks`; we cannot remove `ci_hsc_gen2` until `ci_hsc_gen3` includes those checks (



DM-28806 - Jira project doesn't exist or you don't have permission to view

it.

(3) and (4) are worth keeping around as long as the rest of `ci_hsc_gen2` has to stay around, since they provide coverage for tools that may still be useful while we are converting Gen2 test code to Gen3.

3.2. pipe.tasks.mocks and nopytest_test_coadds.py

This test utility subpackage and the (only) test that uses it provide a lot of early coverage for our coaddition and coadd-processing tasks. They use very simple (i.e. Gaussian stars, little or no noise) simulations, in order to check that the bookkeeping for `CoaddPsf` is exactly right on a small amount of data, rather than statistically right on a lot of noisy data. There's a lot of infrastructure here just to mimic a Gen2 `CameraMapper`, but it's also tied up with the code that does the simulations.

The test script has also attracted a number of miscellaneous test methods that simply want to check something about the outputs of the tasks the test runs.

Porting all of this coverage to Gen3 would be a lot of work, and I'm skeptical that it's all useful:

- The zero-noise, simple-PSF simulation framework was intended to be useful for more than `CoaddPsf`, but no other tests made use of that (that I know of, at least).
- While that test was helpful in getting `CoaddPsf` up and running initially, `CoaddPsf` *still* shipped with a critical bug in weights that the test was too simple to catch, and a much bigger problem with `CoaddPsf` involving sigma-clipping plagued us for a while, too. My takeaway from all of this is that for something like `CoaddPsf`, large-scale integration tests on realistic data (~RC2 or test-med-1, not just `ci_hsc` or `ci_imsim`) are necessary; if we have those, they are also sufficient, and a smaller test like this is only useful as a way to catch *some* problems early, especially when there is high churn.
- `CoaddPsf` isn't experiencing high churn at all, and in fact we hope to drop it in favor of cell-based coadds someday instead of develop it further.
- The coverage from running the other tasks here is again mostly just coverage that they run at all, and hence it duplicates coverage we get from `ci_hsc_gen3` and `ci_imsim`. Nobody should be making changes to these tasks without running those `ci_` packages before merge, and is probably developing against real data, not this test, so I don't think discovering failures "earlier" from this test is a strong argument.

So, my tentative recommendation here is to port the test methods that perform checks on the processing outputs to `ci_hsc_gen3` and/or `ci_imsim`, and scrap the rest.

3.3. pipe_tasks' test_processCcd.py

This runs `ProcessCcdTask` on data from `obs_test`. It seems to mostly check that the tasks run at all, but also has some purely empirical regression tests ("we got this value once; check that the new one isn't worse").

`obs_test` is a pure-Gen2 thing, and the `pipelines_check` package runs the exact same tasks already. Let's move the checks to tests in that package, and drop this test entirely.

3.4. obs_decam's nopytest_test_processCcd.py

This runs `processCcd` and checks that some output files are `gen2` butler gettable and with some very trivial checks on metadata. This is one of the longest runtime portions of building `obs_decam`. I suspect it should be replaced with some more direct tests of DECAM ISR, and otherwise removed. The most important component, I think, is `testWcPostIsr`.

3.5. obs_test

`obs_test` is used by one test in `meas_algorithms` (`test_htmlIndex.py`) as well as `pipe_tasks` mentioned above. `pipe_tasks` also uses it in `test_fakeProcessing.py`, `test_makeDiscreteSkyMap.py`, and `test_transform.py`.

4. Guidelines for Gen2 content removal

4.1. Guidance on using pytest coverage

Our [pytest coverage summary information](#) can help us check whether the removal of tests has caused significant chunks of code to be not executed by tests.

If you are planning to remove a test, run [pytest+coverage on just that test file](#), remove the test, and then re-run to see what lines of code are no longer executed. Note that other tests in a package may execute a given line, so a line may not show up as "uncovered" until you've removed multiple tests. Try to understand what the test was doing and whether any lines that are "uncovered" were an important part of that test. If so, we may need to add new `gen3` tests of that functionality.

4.2. Removing CmdLineTask inheritance from hybrid tasks

Blocked if the task is still run by any Gen2 integration or multi-task unit tests.

What to change:

- Remove `CmdLineTask` inheritance
- Remove `RunnerClass` class attribute if it exists.
- Remove any custom `TaskRunner` subclass used by only this task.
- Remove class methods, if they exist: `applyOverrides`, `parseAndRun`, `_makeArgumentParser`
- Remove regular methods, if they exist: `runDataRef`, `writeConfig`, `writeSchemas`, `writeMetadata`, `writePackageVersions`, `_getConfigName`, `_getMetadataName`
- Modify `__init__` signature to match that of `PipelineTask`. Most frequently this will involve making `config` a keyword-only argument and removing any `butler` or `schema` arguments that exist (if changes are needed at all).
- Replace any use of `lsst.pipe.base.ShimButler` in the `run` method with native Gen3 butler calls.
- Search for the string "DataRef" in docs, comments, and type annotations. Replace it with one of the following:
 - `lsst.daf.butler.DeferredDatasetHandle` to refer to what the `run` method is given for a connection with `deferLoad=True` (this is the Gen3 object that behaves most like a Gen2 `DataRef`);
 - `lsst.daf.butler.DatasetRef` to refer to the combination of a data ID and a dataset type.
- Deprecate (but do not immediately remove) Gen2-only configuration options.
- Remove the associated script in `bin.src`.

4.3. Removing Gen2 support from obs packages

Blocked if the `obs` package is still used in a Gen2 integration test. For DRP, this just means `obs_subaru`. **Need input from AP on what they still need for Gen2.**

What to change:

- Remove the `CameraMapper` subclass.
- Remove YAML mapper policy files.
- Remove Gen2 ingestion tasks (inherits things in from `lsst.pipe.tasks.ingest`; Gen3 ingest framework is in `lsst.obs.base`)
- Inspect all scripts in `bin/bin.src` and remove any that are Gen2 only.
 - Follow [this procedure](#) if Gen3 replacement is unclear.
 - Look for Python module code that exists only to implement these scripts.

4.4. Removing Gen2 support from obs_base

Blocked by the existence of Gen2 support in any concrete `obs` package.

(Not going to work out what to remove in detail here; that's just part of doing the work.)


4.5. Removing Gen2 support from pipe_base

Blocked by the existence of concrete `CmdLineTasks` in downstream packages.

(Not going to work out what to remove in detail here; that's just part of doing the work.)

4.6. Removing pipe_drivers and ctrl_pool

Blocked by

 [DM-29835](#) - Jira project doesn't exist or you don't have permission to view it.

: one task needs to be moved from pipe_dr


ivers to pipe_task.

I expect the removal of `ctrl_pool` to allow us to remove some dependencies from our conda third-party package list (anything MPI related).

4.7. Removing daf_persistence

Blocked by the existence of any `lsst.daf.persistence` import downstream.

•

 [DM-33470](#) - Jira project doesn't exist or you don't have permission to view it.

4.8. Removing other Gen2-only code

Blocked if the code is still run by any Gen2 integration or multi-task unit tests (e.g. if the `Gen2 CmdLineTask` and `Gen3 PipelineTask` for a single conceptual step that calls the same underlying subtasks are distinct).

If the task/script's functionality does not have a clear Gen3 counterpart:

- First file a ticket with label "gen2-functionality-removal" for removing it, but don't start work on it immediately.
- Some person or group of people (TBD) will look at those tickets, and may create a new ticket to replace the functionality.
- We will remove the label when we've decided what to do, and may make a replace-functionality ticket a blocker on the removal ticket.
- If there is no blocker for the ticket and the "gen2-functionality-removal" label has been removed, the removal ticket may proceed.

Key	Summary	T	Created	Updated	Due	Assignee	Reporter	P	Status	Resolution
-----	---------	---	---------	---------	-----	----------	----------	---	--------	------------


No issues found

5. Post-removal cleanups

5.1. Moving configuration from obs_ packages to _pipe packages

When a hybrid task has had its `CmdLineTask` side removed, we're no longer blocked from moving its configuration content to pipeline files in `_pipe`

packages, as per

 [RFC-775](#) - Jira project doesn't exist or you don't have permission to view it.

But I'm not in a hurry to do this unless it solves some other problem, and we should be careful not to move configuration content that does actually belong in the `obs_` package. Most ISR configuration should stay where it is, as well as anything that corresponds to the set of *available* filters for the instrument (not the set of filters being run together for any particular processing run, which *is* pipeline material).

5.2. Fixing up reference catalog loaders

There is a lot of duplication between the Gen2 and Gen3 code paths that we'll hopefully be able to remove. But this code is super fragile and poorly tested in at least some edge cases (padding, padding, padding, in so many places!). I suspect that changing `__init__` signatures and deprecating Gen2-only configs will push us to try to remove/fix this more aggressively than we should. When there is any doubt about preserving the Gen3 behavior, we should be willing to leave unused bits of Gen2 refcat code in place and flag them for removal in a later, more careful ticket.

5.3. Renaming things that say "gen3"

Let's wait until the Gen2 things are fully gone, and then think about whether the disruption is worth it in each case. I don't currently see a reason to prioritize this.