DC3 Pex Harness Tutorial

updated: Feb 11, 2009
This is a quick and dirty tutorial for apps developers. It does not explain how things work, just how to use them. If you would like to learn about how things actually work, take a jump over to DC3PexHarnessDesign

Writing a Pipeline

1. Write your stages. Encapsulate reusable algorithms into separate stages. You do this by creating your own derived class of an lsst.pex.harness.Stage. (You may also inherit from the other predefined Stage classes if you wish tighter control over how they do what they do). The usual place your Stage classes is in /python/lsst/*your*/*package*/*path*/pipeline. (For example python/lsst/ip/diffim/pipeline)
2. Write policy files to configure your Stages.
3. Design the behavior and interaction of your stages. Decide whether a stage should block for events and if so which ones. Your design is reflected in the pipeline policy file you write. **Important:** make sure that event names, and clipboard key names match throughout your pipeline.
4. Note that application stages should never do I/O of any kind. They should interact solely through the Clipboard and be configured by Policy. Any I/O, whether to files, the database, or events, should be handled by middleware Stages.

Running a Pipeline

1. Create a pipeline directory. If you are unsure what the pipeline should look like in production, or are providing examples of how to use Stages, you might choose to place your pipeline in a <package-path>/examples/<pipeline name>. where <package-path> would be replaced by the path of the package within which you are developing, and <pipeline name> is the what you would like to name the pipeline. For example:

```
DMS/ip/diffim/examples/testPipeline
```

2. Place your pipeline policy file in the pipeline directory. For example:

```
DMS/ip/diffim/examples/testPipeline/pipeline_policy.paf
```

3. Create a policy directory within your pipeline directory, and place all your stage policies in that folder. For example:

```
DMS/ip/diffim/examples/testPipeline/policy
```

4. Place all your stage policy files within your policy directory:

```
DMS/ip/diffim/examples/testPipeline/policy/subtractionStagePolicy.paf
DMS/ip/diffim/examples/testPipeline/policy/detectionStagePolicy.paf
```

5. Make sure that in your pipeline policy file, the path given for each stage's policy (which should be a relative path, not absolute) matches the directory structure you have created:

```
...
appStage: {
  stageName: "lsst.ip.diffim.pipeline.SubtractionStage"
  ...
  stagePolicy: "policy/subtractionStagePolicy.paf"
}
...
```

6. Create (or copy from one of the pex/harness/examples) a nodelist.scr file. This file contains a description of which NCSA cluster nodes you want to run your pipeline on followed by the number of process to run on that node
    - A single-node nodelist.scr file might look like this:

```
lsst8.ncsa.uiuc.edu:2
```

    - A multi-node nodelist.scr file might look like this:

```
lsst8.ncsa.uiuc.edu:2
lsst6.ncsa.uiuc.edu:2
lsst9.ncsa.uiuc.edu:6
```

7. Before you can test or run your pipeline you will need to setup your environment.
    - In your package's root directory run:

```
%setup -r .
```

    - build your package:

```
%scons
```

    - make sure that the pex_harness and ctrl_events are also setup. They should be part of your package's dependencies already (in the ups/<packageName>.table file), but we can check using:

```
%eups list pex_harness
%eups list ctrl_events
```

8. If your stages block for events, you will want to create a python script to generate each of those events. The script should be run in its own terminal once the pipeline is started. Here is an example event generation script that fires an event named "triggerDiffimEvent":

```
import threading
import lsst.daf.base as dafBase
from lsst.daf.base import *

import lsst.ctrl.events as events
import time

if __name__ == "__main__":
    eventBrokerHost = "lsst8.ncsa.uiuc.edu"
    externalEventTransmitter = events.EventTransmitter(eventBrokerHost, "triggerDiffimEvent")

    root = PropertySet()

    root.setInt("visitId", 1)
    externalEventTransmitter.publish(root)
```

9. To run the actual pipeline, you will need to use mpd. In case you are interested, here is a link to the official ?mpd documentation. But that's probably more than you want. Fortunately the good guys over at middleware have made our lives easy yet again by providing a script to run it. So two options:
    - Using the script:
        - ◊ You can get it here: ?run.sh
        - ◊ Copy the script to your pipeline directory, and run it with the following command, replacing <pipeline-policy> with the your pipeline policy file, and <run-id> with a string identified for this run of your pipeline.

```
%run.sh <pipeline-policy> <run-id>
```

        - ◊ If your pipeline blocks for events, run your event generator script from a separate shell to supply those events.

- Doing it by hand.
  - Start mpd on every node you are working on. with the following command, replacing <nodes> by the number of nodes you are running on.

```
mpdboot --totalnum=<nodes> --file=nodelist.scr --verbose
```

  - You can ensure that mpd started correctly on all nodes with the command (wait a few seconds after the previous command as it might take a few seconds to start mpd on all nodes)

```
mpdtrace
```

  - Now start the pipeline. In the pipeline directory you created above, run the following command, where <usize> is your universe size (which is the number of slices + 1), <policy-file> is your pipeline policy file, and run-Id is the name you give this run

```
mpiexec --usize <usize> -machinefile nodelist.scr -np 1 runPipeline.py <policy-file> <ru
```

  - After your pipeline has executed (or to terminate it early) run the command:

```
mpdallexit
```