Utilisation du stack LSST

- *Introduction: le stack LSST*
- *Mise en place de l'environnement*
- *Structure du répertoire de travail*
- *Fichiers en entrée*
- *demo.sh: analyse et exécution*
- *Fichiers en sortie*
- *Un peu de détails sur le code*

Cette présentation s'inspire notamment des investigations réalisées lors d'un TP sur le stack (Dominique, Résa, Christian, Eduardo – 15/11/2012 – http://lsst.in2p3.fr/wiki/index.php/TP_sur_le_stack)

Le stack LSST

- Le stack LSST: ensemble d'outils permettant de traiter les images issues du détecteur
- L'objectif est d'obtenir:
  - Des images corrigées et étalonnées
  - Des listes de sources (algorithmes)
  - Diverses observables liées aux sources (psf)
  - …
- Le stack peut traiter des images de LSST, SDSS, … et présente une structure suffisamment souple pour s'adapter à d'autres configurations.

Mise en place de l'environnement

- *On suppose que le stack a été installé dans un répertoire* **$LSST_HOME** *(par exemple au cc:* **/sps/lsst/Library/stack_Winter2013***)*
- *Installation de l'environnement pour la demo:*
- **curl -O https://dev.lsstcorp.org/cgit/contrib/demos/lsst_dm_stack_demo.git/snapshot/lsst_dm_stack_demo-master.tar.gz**
- **tar xzf lsst_dm_stack_demo-master.tar.gz**
- **cd lsst_dm_stack_demo-master -> répertoire de travail**
- **source $LSST_HOME/loadLSST.sh**
- **setup pipe_tasks** → Description de la caméra
- **setup obs_sdss**
- **Ces setups devront être refaits à chaque nouvelle session**

- *(procédure d'installation du stack: https://dev.lsstcorp.org/trac/wiki/Installing*

Cela fonctionne assez bien sur SL6 (virtuelle))

Structure du répertoire de travail

- **ls lsst_dm_stack_demo-master**
  - **README**
  - **astrometry_net_data/ -> fichiers (fits) pour points zéro photométriques**
  - **bin/ -> scripts d'exécution: demo.sh (bash) et export-results (python)**
  - **detected-sources.txt.expected -> liste des sources que l'on doit obtenir après exécution de la demo (pour vérification)**
  - **input/ -> répertoire de données (entrées)**
  - **output/ -> répertoire de données (entrées) (créé lors de l'exécution si inexistant)**

Données d'entrée

- **Les fichiers SDSS qui sont processés par le stack sont placés dans le répertoire input**
- **Ils sont repérés par quatre attributs:**
  - **run: Le numéro de run**
  - **rerun: Le numéro de reprocessing des données**
  - **camcol: resultat d'une colonne de CCD (1 <= camcol <= 6)**
  - **field: ensemble de pixels 2048x1489 -> pose pour cinq filtres (u,g,r,i,z) d'une même portion du ciel**
- **Pour chaque quadruplet(run,rerun,camcol, field) , les données sont placées dans les répertoires de input/run/rerun:**

Répertoire	Fichiers	Contenu des fichiers
astrom	asTrans-run.fit	table binaire (fits) avec transformations astrométriques pour chaque field pour un run donné
calibChuncks/camcol	tsField-run-camcol-rerun-field.fit	tables binaires (fits) contenant des informations pour chaque field (calibration et gain)
corr/camcol	fpc-run-fcamcol-field.fit.gz (f=u,g,r,i,z)	version non calibrée d'une image fits (2048x1489 pixels) corrigée (flat-field, bias, cosmic-ray, pixel-defect, sky subtracted, photometric calibration)
objcs/camcol	fpM-run-fcamcol-field.fit (f=u,g,r,i,z) psField-run-camcol-field.fit	image fits avec des bits masqués pour chaque pixel table binaire avec calibration photométrique préliminaire et fit final de la psf

demo.sh: analyse

- **Un script (demo.sh) permettant de lire les données SDSS est disponible dans bin:**

```
#!/bin/bash

eups list --setup obs_sdss >/dev/null 2>&1 || ( echo "obs_sdss does not appear to be setup, or eups is not configured co
rrectly."; exit 1; )
type processCcdSdss.py >/dev/null 2>&1 || { echo "Could not find processCcdSdss.py on your path. It is supposed to be in
 obs_sdss."; exit 1; }
test -d input || { echo "Could not find the 'input' directory. Run this script from the directory where the 'input' subd
irectory resides."; exit 1; }
test -d astrometry_net_data || { echo "Could not fine the 'astrometry_net_data' directory."; exit 1; }
test "$(type -t setup)" == "function" || { export SHELL=/bin/bash; source $LSST_HOME/eups/default/bin/setups.sh; } # Ens
ure 'setup' alias is defined (may happen if the user is not running bash)
set -e

# Tell the stack where to find astrometric reference catalogs
setup --nolocks -v -r ./astrometry_net_data astrometry_net_data

rm -rf output detected-sources.txt
processCcdSdss.py input --id run=4192 filter=u^g^r^i^z camcol=4 field=300 --id run=6377 filter=u^g^r^i^z camcol=4 field=
399 --output output
./bin/export-results output > detected-sources.txt

echo
echo "Processing completed successfully. The results are in detected-sources.txt."
```

demo.sh: analyse

- **Un script (demo.sh) permettant de lire les données SDSS est disponible dans bin:**

```
#!/bin/bash

eups list --setup obs_sdss >/dev/null 2>&1 || ( echo "obs_sdss does not appear to be setup, or eups is not configured co
rrectly."; exit 1; )
type processCcdSdss.py >/dev/null 2>&1 || { echo "Could not find processCcdSdss.py on your path. It is supposed to be in
 obs_sdss."; exit 1; }
test -d input || { echo "Could not find the 'input' directory. Run this script from the directory where the 'input' subd
irectory resides."; exit 1; }
test -d astrometry_net_data || { echo "Could not fine the 'astrometry_net_data' directory."; exit 1; }
test "$(type -t setup)" == "function" || { export SHELL=/bin/bash; source $LSST_HOME/eups/default/bin/setups.sh; } # Ens
ure 'setup' alias is defined (may happen if the user is not running bash)
set -e

# Tell the stack where to find astrometric reference catalogs
setup --nolocks -v -r ./astrometry_net_data astrometry_net_data

rm -rf output detected-sources.txt
processCcdSdss.py input --id run=4192 filter=u^g^r^i^z camcol=4 field=300 --id run=6377 filter=u^g^r^i^z camcol=4 field=
399 --output output
./bin/export-results output > detected-sources.txt

echo
echo "Processing completed successfully. The results are in detected-sources.txt."
```

- **Vérifications:**
  - **Setups**
  - **Fichiers astrometry-net**
- **Préparation pour l'exécution**

demo.sh: analyse

- **Un script (demo.sh) permettant de lire les données SDSS est disponible dans bin:**

```
#!/bin/bash

eups list --setup obs_sdss >/dev/null 2>&1 || ( echo "obs_sdss does not appear to be setup, or eups is not configured co
rrectly."; exit 1; )
type processCcdSdss.py >/dev/null 2>&1 || { echo "Could not find processCcdSdss.py on your path. It is supposed to be in
 obs_sdss."; exit 1; }
test -d input || { echo "Could not find the 'input' directory. Run this script from the directory where the 'input' subd
irectory resides."; exit 1; }
test -d astrometry_net_data || { echo "Could not fine the 'astrometry_net_data' directory."; exit 1; }
test "$(type -t setup)" == "function" || { export SHELL=/bin/bash; source $LSST_HOME/eups/default/bin/setups.sh; } # Ens
ure 'setup' alias is defined (may happen if the user is not running bash)
set -e

# Tell the stack where to find astrometric reference catalogs
setup --nolocks -v -r ./astrometry_net_data astrometry_net_data

rm -rf output detected-sources.txt
processCcdSdss.py input --id run=4192 filter=u^g^r^i^z camcol=4 field=300 --id run=6377 filter=u^g^r^i^z camcol=4 field=
399 --output output
./bin/export-results output > detected-sources.txt

echo
echo "Processing completed successfully. The results are in detected-sources.txt."
```

Processing des données

Mise en forme du résultat

demo.sh: exécution

. ./bin/demo.sh:

Setting up: astrometry_net_data Flavor: Linux64 Version: LOCAL:/home/pgris/Winter2013/lsst_dm_stack_demo-master/astrometry_net_data

: Loading config overrride file '/home/lsst/winter2013stack/Linux64/obs_sdss/6.1.0.2+1/config/processCcd.py'

: Config override file does not exist: '/home/lsst/winter2013stack/Linux64/obs_sdss/6.1.0.2+1/config/sdss/processCcd.py'

: input=/home/pgris/Winter2013/lsst_dm_stack_demo-master/input

: calib=None

: output=/home/pgris/Winter2013/lsst_dm_stack_demo-master/output

CameraMapper: Loading registry registry from /home/pgris/Winter2013/lsst_dm_stack_demo-master/output/_parent/registry.sqlite3

processCcd: Processing {'filter': 'u', 'field': 300, 'camcol': 4, 'run': 4192, 'rerun': 40}

processCcd: Loading WCS from asTrans

processCcd.calibrate: Initial PSF is from input exposure; ignoring config.initialPsf.

processCcd.calibrate.detection: Detected 20 positive sources to 5 sigma.

processCcd.calibrate.detection: Resubtracting the background after object detection

processCcd.calibrate.measurement: Measuring 20 sources

processCcd.calibrate.astrometry WARNING: No CCD associated with exposure; assuming null distortion

processCcd.calibrate.astrometry: Null distortion correction

processCcd.calibrate.astrometry: Using filter: 'u'

processCcd.calibrate.astrometry: forceKnownWcs is set: using the input exposure's WCS

processCcd.calibrate.astrometry: 14 astrometric matches

processCcd.calibrate.astrometry: Refitting WCS

processCcd.calibrate.astrometry WARNING: Not calculating a SIP solution; matches may be suspect

demo.sh: exécution

. ./bin/demo.sh:

Setting up: astrometry_net_data Flavor: Linux64 Version: LOCAL:/home/pgris/Winter2013/lsst_dm_stack_demo-master/astrometry_net_data

: Loading config overrride file '/home/lsst/winter2013stack/Linux64/obs_sdss/6.1.0.2+1/config/processCcd.py'

: Config override file does not exist: '/home/lsst/winter2013stack/Linux64/obs_sdss/6.1.0.2+1/config/sdss/processCcd.py'

: input=/home/pgris/Winter2013/lsst_dm_stack_demo-master/input

: calib=None

: output=/home/pgris/Winter2013/lsst_dm_stack_demo-master/output

CameraMapper: Loading registry registry from /home/pgris/Winter2013/lsst_dm_stack_demo-master/output/_parent/registry.sqlite3

processCcd: Processing {'filter': 'u', 'field': 300, 'camcol': 4, 'run': 4192, 'rerun': 40}

processCcd: Loading WCS from asTrans

processCcd.calibrate: Initial PSF is from input exposure; ignoring config.initialPsf.

processCcd.calibrate.detection: Detected 20 positive sources to 5 sigma.

processCcd.calibrate.detection: Resubtracting the background after object detection

processCcd.calibrate.measurement: Measuring 20 sources

processCcd.calibrate.astrometry WARNING: No CCD associated with exposure; assuming null distortion

processCcd.calibrate.astrometry: Null distortion correction

processCcd.calibrate.astrometry: Using filter: 'u'

processCcd.calibrate.astrometry: forceKnownWcs is set: using the input exposure's WCS

processCcd.calibrate.astrometry: 14 astrometric matches

processCcd.calibrate.astrometry: Refitting WCS

processCcd.calibrate.astrometry WARNING: Not calculating a SIP solution; matches may be suspect

demo.sh: exécution

. ./bin/demo.sh:

Setting up: astrometry_net_data Flavor: Linux64 Version: LOCAL:/home/pgris/Winter2013/lsst_dm_stack_demo-master/astrometry_net_data

: Loading config overrride file '/home/lsst/winter2013stack/Linux64/obs_sdss/6.1.0.2+1/config/processCcd.py'

: Config override file does not exist: '/home/lsst/winter2013stack/Linux64/obs_sdss/6.1.0.2+1/config/sdss/processCcd.py'

: input=/home/pgris/Winter2013/lsst_dm_stack_demo-master/input

: calib=None

: output=/home/pgris/Winter2013/lsst_dm_stack_demo-master/output

CameraMapper: Loading registry registry from /home/pgris/Winter2013/lsst_dm_stack_demo-master/output/_parent/registry.sqlite3

processCcd: Processing {'filter': 'u', 'field': 300, 'camcol': 4, 'run': 4192, 'rerun': 40}

processCcd: Loading WCS from asTrans

processCcd.calibrate: Initial PSF is from input exposure; ignoring config.initialPsf.

processCcd.calibrate.detection: Detected 20 positive sources to 5 sigma.

processCcd.calibrate.detection: Resubtracting the background after object detection

processCcd.calibrate.measurement: Measuring 20 sources

processCcd.calibrate.astrometry WARNING: No CCD associated with exposure; assuming null distortion

processCcd.calibrate.astrometry: Null distortion correction

processCcd.calibrate.astrometry: Using filter: 'u'

processCcd.calibrate.astrometry: forceKnownWcs is set: using the input exposure's WCS

processCcd.calibrate.astrometry: 14 astrometric matches

processCcd.calibrate.astrometry: Refitting WCS

processCcd.calibrate.astrometry WARNING: Not calculating a SIP solution; matches may be suspect

demo.sh: exécution

. ./bin/demo.sh:

Setting up: astrometry_net_data Flavor: Linux64 Version: LOCAL:/home/pgris/Winter2013/lsst_dm_stack_demo-master/astrometry_net_data

: Loading config overrride file '/home/lsst/winter2013stack/Linux64/obs_sdss/6.1.0.2+1/config/processCcd.py'

: Config override file does not exist: '/home/lsst/winter2013stack/Linux64/obs_sdss/6.1.0.2+1/config/sdss/processCcd.py'

: input=/home/pgris/Winter2013/lsst_dm_stack_demo-master/input

: calib=None

: output=/home/pgris/Winter2013/lsst_dm_stack_demo-master/output

CameraMapper: Loading registry registry from /home/pgris/Winter2013/lsst_dm_stack_demo-master/output/_parent/registry.sqlite3

processCcd: Processing {'filter': 'u', 'field': 300, 'camcol': 4, 'run': 4192, 'rerun': 40}

processCcd: Loading WCS from asTrans

processCcd.calibrate: Initial PSF is from input exposure; ignoring config.initialPsf.

processCcd.calibrate.detection: Detected 20 positive sources to 5 sigma.

processCcd.calibrate.detection: Resubtracting the background after object detection

processCcd.calibrate.measurement: Measuring 20 sources

processCcd.calibrate.astrometry WARNING: No CCD associated with exposure; assuming null distortion

processCcd.calibrate.astrometry: Null distortion correction

processCcd.calibrate.astrometry: Using filter: 'u'

processCcd.calibrate.astrometry: forceKnownWcs is set: using the input exposure's WCS

processCcd.calibrate.astrometry: 14 astrometric matches

processCcd.calibrate.astrometry: Refitting WCS

processCcd.calibrate.astrometry WARNING: Not calculating a SIP solution; matches may be suspect

demo.sh: exécution

. ./bin/demo.sh:

Setting up: astrometry_net_data Flavor: Linux64 Version: LOCAL:/home/pgris/Winter2013/lsst_dm_stack_demo-master/astrometry_net_data

: Loading config overrride file '/home/lsst/winter2013stack/Linux64/obs_sdss/6.1.0.2+1/config/processCcd.py'

: Config override file does not exist: '/home/lsst/winter2013stack/Linux64/obs_sdss/6.1.0.2+1/config/sdss/processCcd.py'

: input=/home/pgris/Winter2013/lsst_dm_stack_demo-master/input

: calib=None

: output=/home/pgris/Winter2013/lsst_dm_stack_demo-master/output

CameraMapper: Loading registry registry from /home/pgris/Winter2013/lsst_dm_stack_demo-master/output/_parent/registry.sqlite3

processCcd: Processing {'filter': 'u', 'field': 300, 'camcol': 4, 'run': 4192, 'rerun': 40}

processCcd: Loading WCS from asTrans

processCcd.calibrate: Initial PSF is from input exposure; ignoring config.initialPsf.

processCcd.calibrate.detection: Detected 20 positive sources to 5 sigma.

processCcd.calibrate.detection: Resubtracting the background after object detection

processCcd.calibrate.measurement: Measuring 20 sources

processCcd.calibrate.astrometry WARNING: No CCD associated with exposure; assuming null distortion

processCcd.calibrate.astrometry: Null distortion correction

processCcd.calibrate.astrometry: Using filter: 'u'

processCcd.calibrate.astrometry: forceKnownWcs is set: using the input exposure's WCS

processCcd.calibrate.astrometry: 14 astrometric matches

processCcd.calibrate.astrometry: Refitting WCS

processCcd.calibrate.astrometry WARNING: Not calculating a SIP solution; matches may be suspect

demo.sh: exécution

processCcd.calibrate: 'catalog' PSF star selector found 13 candidates
processCcd.calibrate: Not running PSF determination
processCcd.calibrate.ApertureCorrection: flux.psf to flux.sinc
processCcd.calibrate.ApertureCorrection: numGoodStars: 13
processCcd.calibrate.ApertureCorrection: numAvailStars: 13
processCcd.calibrate.ApertureCorrection: mean apCorr: 1.0063 +/- 0.0277
processCcd.calibrate: Central aperture correction using 13/13 stars: 0.996605 +/- 0.001685
processCcd.calibrate.measurement: Applying aperture correction to 20 sources
processCcd.calibrate.photocal: Magnitude zero point: 27.504209 +/- 0.011628 from 10 stars
processCcd.calibrate: Photometric zero-point: 27.504209
processCcd.detection: Detected 79 positive sources to 5 sigma.
processCcd.detection: Resubtracting the background after object detection
processCcd.measurement: Measuring 79 sources
processCcd.measurement: Applying aperture correction to 79 sources
processCcd.measurement: Classifying 79 sources
processCcd WARNING: Persisting background models as an image
processCcd: Matching icSource and Source catalogs to propagate flags.

demo.sh: exécution

processCcd.calibrate: 'catalog' PSF star selector found 13 candidates
processCcd.calibrate: Not running PSF determination
processCcd.calibrate.ApertureCorrection: flux.psf to flux.sinc
processCcd.calibrate.ApertureCorrection: numGoodStars: 13
processCcd.calibrate.ApertureCorrection: numAvailStars: 13
processCcd.calibrate.ApertureCorrection: mean apCorr: 1.0063 +/- 0.0277
processCcd.calibrate: Central aperture correction using 13/13 stars: 0.996605 +/- 0.001685
processCcd.calibrate.measurement: Applying aperture correction to 20 sources
processCcd.calibrate.photocal: Magnitude zero point: 27.504209 +/- 0.011628 from 10 stars
processCcd.calibrate: Photometric zero-point: 27.504209
processCcd.detection: Detected 79 positive sources to 5 sigma.
processCcd.detection: Resubtracting the background after object detection
processCcd.measurement: Measuring 79 sources
processCcd.measurement: Applying aperture correction to 79 sources
processCcd.measurement: Classifying 79 sources
processCcd WARNING: Persisting background models as an image
processCcd: Matching icSource and Source catalogs to propagate flags.

demo.sh: exécution

processCcd.calibrate: 'catalog' PSF star selector found 13 candidates
processCcd.calibrate: Not running PSF determination
processCcd.calibrate.ApertureCorrection: flux.psf to flux.sinc
processCcd.calibrate.ApertureCorrection: numGoodStars: 13
processCcd.calibrate.ApertureCorrection: numAvailStars: 13
processCcd.calibrate.ApertureCorrection: mean apCorr: 1.0063 +/- 0.0277
processCcd.calibrate: Central aperture correction using 13/13 stars: 0.996605 +/- 0.001685
processCcd.calibrate.measurement: Applying aperture correction to 20 sources
processCcd.calibrate.photocal: Magnitude zero point: 27.504209 +/- 0.011628 from 10 stars
processCcd.calibrate: Photometric zero-point: 27.504209
processCcd.detection: Detected 79 positive sources to 5 sigma.
processCcd.detection: Resubtracting the background after object detection
processCcd.measurement: Measuring 79 sources
processCcd.measurement: Applying aperture correction to 79 sources
processCcd.measurement: Classifying 79 sources
processCcd WARNING: Persisting background models as an image
processCcd: Matching icSource and Source catalogs to propagate flags.

demo.sh: exécution

processCcd.calibrate: 'catalog' PSF star selector found 13 candidates
processCcd.calibrate: Not running PSF determination
processCcd.calibrate.ApertureCorrection: flux.psf to flux.sinc
processCcd.calibrate.ApertureCorrection: numGoodStars: 13
processCcd.calibrate.ApertureCorrection: numAvailStars: 13
processCcd.calibrate.ApertureCorrection: mean apCorr: 1.0063 +/- 0.0277
processCcd.calibrate: Central aperture correction using 13/13 stars: 0.996605 +/- 0.001685
processCcd.calibrate.measurement: Applying aperture correction to 20 sources
processCcd.calibrate.photocal: Magnitude zero point: 27.504209 +/- 0.011628 from 10 stars
processCcd.calibrate: Photometric zero-point: 27.504209
processCcd.detection: Detected 79 positive sources to 5 sigma.
processCcd.detection: Resubtracting the background after object detection
processCcd.measurement: Measuring 79 sources
processCcd.measurement: Applying aperture correction to 79 sources
processCcd.measurement: Classifying 79 sources
processCcd WARNING: Persisting background models as an image
processCcd: Matching icSource and Source catalogs to propagate flags.

Données en sortie

- **Les résultats du processing sont placés dans le répertoire output/sci-results du répertoire de travail.**
- **Sous-répertoires: run/camcol/filtre (u,g,r,i,z)**

ls output/sci-results/4192/4/*
output/sci-results/4192/4/g:
apCorr calexp icMatch icSrc psf src

output/sci-results/4192/4/i:
apCorr calexp icMatch icSrc psf src

output/sci-results/4192/4/r:
apCorr calexp icMatch icSrc psf src

output/sci-results/4192/4/u:
apCorr calexp icMatch icSrc psf src

output/sci-results/4192/4/z:
apCorr calexp icMatch icSrc psf src

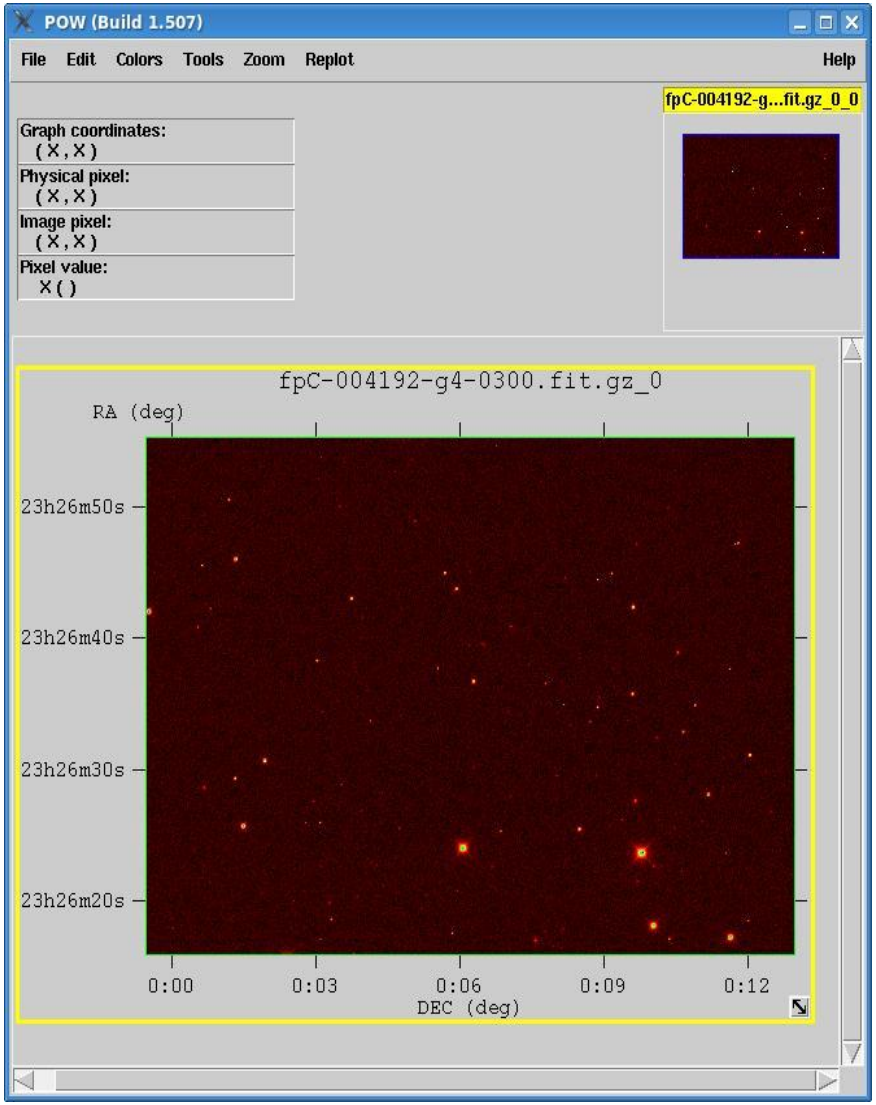
Données en sortie

- **Pour chaque triplet (run, camcol, filtre): 6 répertoires:**
  - **apCorr :**
    - **apCorr-run-filtrecamcol-field.pickle -> correction d'ouverture (format spécifique python)**
  - **calexp :**
    - **bkgd-calexp-run-filtrecamcol-field.fits -> images de fond ?**
    - **calexp-run-filtrecamcol-field.fits -> images calibrées**
  - **icMatch :**
    - **icMatch-run-filtrecamcol-field.fits -> semblent donner les numéros des étoiles associées (peut-être dans le fichier de référence astrometry_net_data)**
  - **icSrc:**
    - **icSrc-run-filtrecamcol-field.fits -> semblent contenir les informations (photométrie, etc...) des objets détectés sur les images**
  - **psf**
    - **psf-run-filtrecamcol-field.boost -> psf (format boost)**
  - **src**
    - **src-run-filtrecamcol-field.fits -> sources détectées**

Image SDSS
2048x1489 pixels

Données en sortie

Image LSST (calexp)
2048x1361pixels



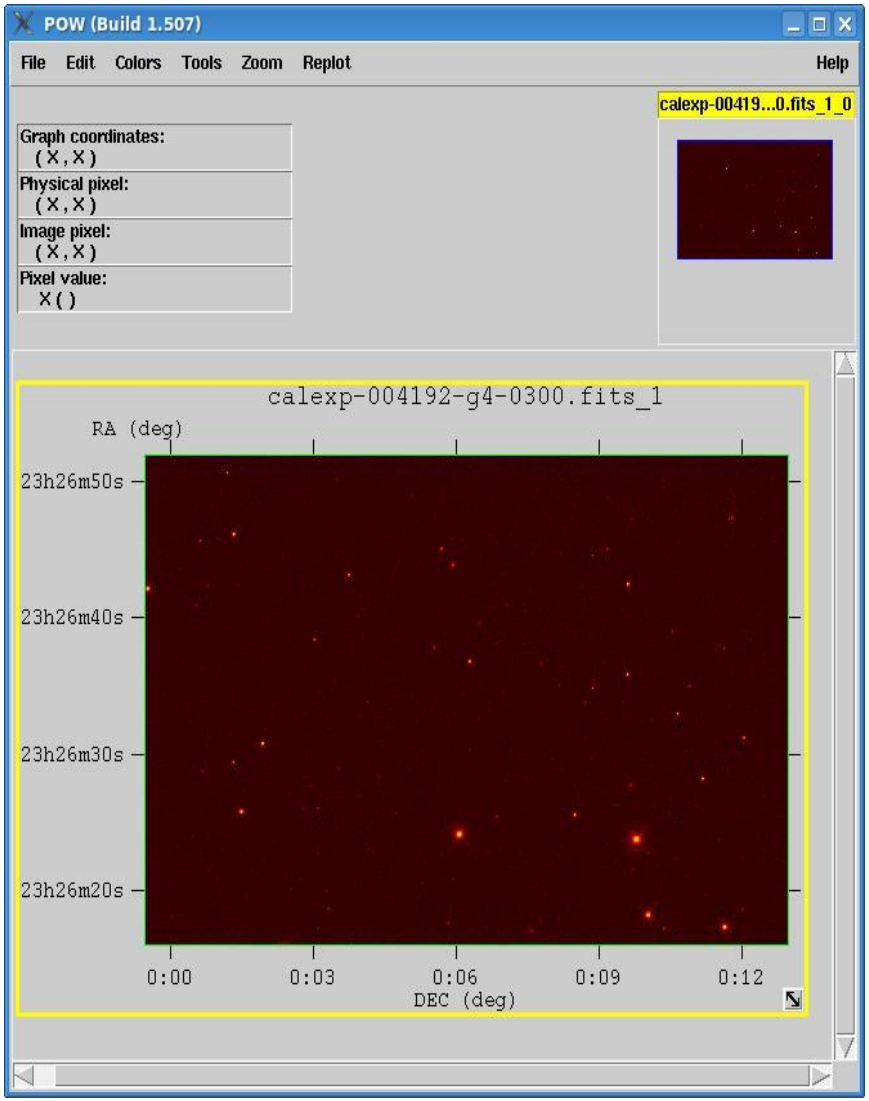


Image SDSS
2048x1489 pixels

Données en sortie

Image LSST (calexp)
2048x1361pixels

fv: Image of fpC-004192-g4-0300.fit.gz[0] in /home/pgris/Winter2013/lsst_dm_stack_demo-master/input/4192/40/corr/4/

File Edit Tools Help

Select All Invert

	1	2	3	4	5	6
1361	1.0970000000E+03	1.0880000000E+03	1.0920000000E+03	1.0890000000E+03	1.0920000000E+03	1.0940000000E+03
1360	1.0900000000E+03	1.1030000000E+03	1.0980000000E+03	1.0950000000E+03	1.1000000000E+03	1.0960000000E+03
1359	1.0950000000E+03	1.0950000000E+03	1.1010000000E+03	1.0930000000E+03	1.0940000000E+03	1.1040000000E+03
1358	1.0920000000E+03	1.0890000000E+03	1.1010000000E+03	1.0900000000E+03	1.0990000000E+03	1.0980000000E+03
1357	1.0930000000E+03	1.0990000000E+03	1.0960000000E+03	1.0860000000E+03	1.1010000000E+03	1.1090000000E+03
1356	1.0990000000E+03	1.1010000000E+03	1.1020000000E+03	1.0860000000E+03	1.0930000000E+03	1.0940000000E+03
1355	1.0870000000E+03	1.0930000000E+03	1.0950000000E+03	1.0950000000E+03	1.1000000000E+03	1.0940000000E+03
1354	1.0990000000E+03	1.0970000000E+03	1.1030000000E+03	1.0980000000E+03	1.0990000000E+03	1.1000000000E+03
1353	1.1010000000E+03	1.1010000000E+03	1.0960000000E+03	1.0970000000E+03	1.1040000000E+03	1.0980000000E+03
1352	1.1010000000E+03	1.0920000000E+03	1.0990000000E+03	1.0990000000E+03	1.1000000000E+03	1.1030000000E+03
1351	1.0950000000E+03	1.0900000000E+03	1.0990000000E+03	1.0990000000E+03	1.0970000000E+03	1.0950000000E+03
1350	1.1000000000E+03	1.1010000000E+03	1.0940000000E+03	1.1010000000E+03	1.0990000000E+03	1.0900000000E+03
1349	1.0940000000E+03	1.0960000000E+03	1.0940000000E+03	1.1040000000E+03	1.1030000000E+03	1.0990000000E+03
1348	1.0890000000E+03	1.0950000000E+03	1.0930000000E+03	1.1010000000E+03	1.1040000000E+03	1.0950000000E+03
1347	1.0960000000E+03	1.0950000000E+03	1.1020000000E+03	1.0950000000E+03	1.0910000000E+03	1.0980000000E+03
1346	1.1000000000E+03	1.1000000000E+03	1.0960000000E+03	1.1040000000E+03	1.0910000000E+03	1.0900000000E+03
1345	1.1010000000E+03	1.0980000000E+03	1.0870000000E+03	1.0900000000E+03	1.0960000000E+03	1.1010000000E+03
1344	1.0970000000E+03	1.0930000000E+03	1.0960000000E+03	1.0910000000E+03	1.1010000000E+03	1.1050000000E+03
1343	1.0910000000E+03	1.1070000000E+03	1.1020000000E+03	1.0990000000E+03	1.0960000000E+03	1.0900000000E+03
1342	1.0950000000E+03	1.0760000000E+03	1.0910000000E+03	1.1070000000E+03	1.0980000000E+03	1.0980000000E+03

Go to: Edit cell: Lock to Parent

fv: Image of calexp-004192-g4-0300.fits[1] in /home/pgris/Winter2013/lsst_dm_stack_demo-master/output/sci-resu

File Edit Tools Help

Select All Invert

	1	2	3	4	5	6
1361	0.80633	-8.19291	-4.19214	-7.19138	-4.19061	-2.18984
1360	-6.19254	6.80823	1.80899	-1.19024	3.81053	-0.18871
1359	-1.19140	-1.19064	4.81013	-3.18911	-2.18834	7.81243
1358	-4.19027	-7.18950	4.81126	-6.18797	2.81279	1.81356
1357	-3.18913	2.81163	-0.18760	-10.18684	4.81393	12.81470
1356	2.81201	4.81277	5.81353	-10.18570	-3.18493	-2.18417
1355	-9.18686	-3.18610	-1.18533	-1.18457	3.81620	-2.18304
1354	2.81428	0.81504	6.81580	1.81657	2.81733	3.81809
1353	4.81541	4.81618	-0.18306	0.81770	7.81846	1.81923
1352	4.81655	-4.18269	2.81807	2.81884	3.81960	6.82036
1351	-1.18232	-6.18155	2.81921	2.81997	0.82073	-1.17851
1350	3.81882	4.81958	-2.17966	4.82111	2.82187	-6.17737
1349	-2.18004	-0.17928	-2.17852	7.82224	6.82300	2.82376
1348	-7.17891	-1.17815	-3.17739	4.82337	7.82414	-1.17510
1347	-0.17777	-1.17701	5.82375	-1.17549	-5.17473	1.82603
1346	3.82336	3.82412	-0.17512	7.82564	-5.17360	-6.17284
1345	4.82450	1.82526	-9.17398	-6.17322	-0.17246	4.82830
1344	0.82563	-3.17361	-0.17285	-5.17209	4.82867	8.82943
1343	-5.17323	10.82753	5.82829	2.82905	-0.17019	-6.16943
1342	-1.17209	-20.17134	-5.17058	10.83018	1.83094	1.83170

Go to: Edit cell: Lock to Parent

Données en sortie

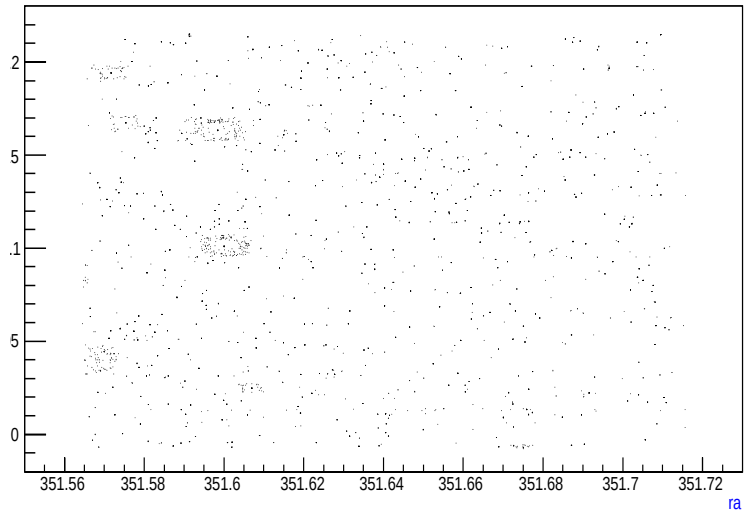
- **Fin de demo.sh:**
- **./bin/export-results output > detected-sources.txt**
- **Script python qui crée un fichier texte des sources détectées dont les caractéristiques sont choisies dans le script**
- **Un seul fichier est produit pour tous les filtres**

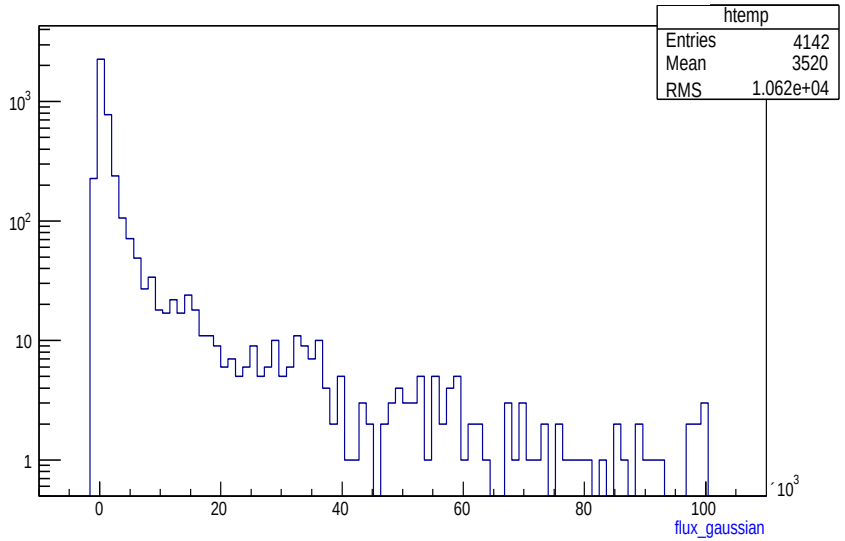
```
for filter in "ugriz":
    srcs = butler.get("src", run=4192, filter=filter, field=300, camcol=4)
    for s in srcs:
        print ' '.join([str(e(s)) for e in extractors])
```

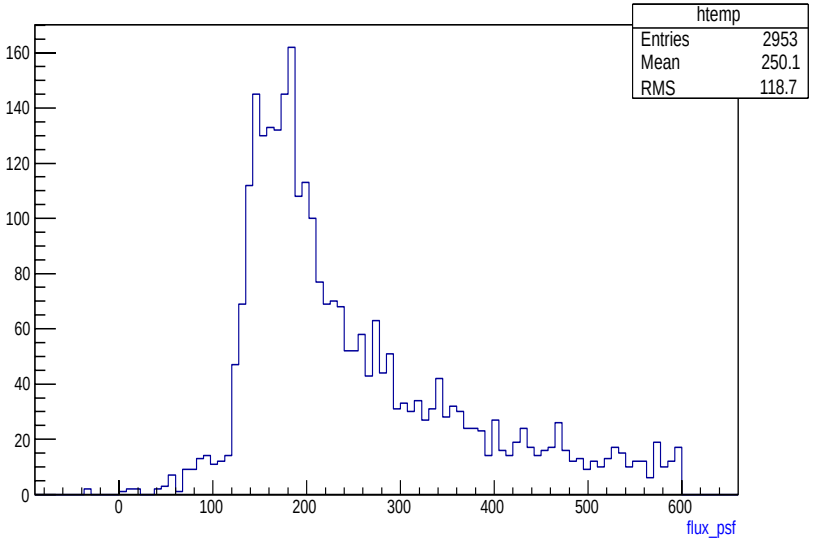
- **Attention: le triplet (run,field,camcol) est explicitement mentionné.**

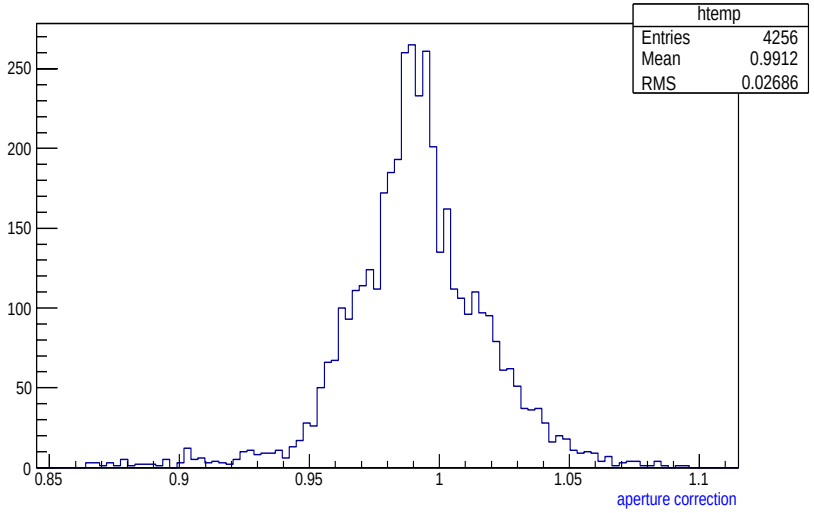
```
cols = ("id",
    "coord.ra",
    "coord.dec",
    "flags.negative",
    "flags.badcentroid",
    "flags.pixel.edge",
    "flags.pixel.interpolated.any",
    "flags.pixel.interpolated.center",
    "flags.pixel.saturated.any",
    "flags.pixel.saturated.center",
    "centroid.sdss.x",
    "centroid.sdss.y",
    "centroid.sdss.err.xx",
    "centroid.sdss.err.yy",
    "centroid.gaussian.x",
    "centroid.gaussian.y",
    "centroid.gaussian.err.xx",
    "centroid.gaussian.err.yy",
    "shape.sdss.ixx",
    "shape.sdss.ixy",
    "shape.sdss.iyy",
    "shape.sdss.err.ixx",
    "shape.sdss.err.ixy",
    "shape.sdss.err.iyy",
    "shape.sdss.flags",
    "flux.gaussian",
    "flux.gaussian.err",
    "flux.psf",
    "flux.psf.err",
    "flux.sinc",
    "flux.sinc.err",
    "multishapelet.exp.flux",
    "multishapelet.exp.flux.err",
    "multishapelet.dev.flux",
    "multishapelet.dev.flux.err",
    "multishapelet.combo.flux",
    "multishapelet.combo.flux.err",
    "classification.extendedness",
    "aperturecorrection",
    "aperturecorrection.err",
    )
```

Données en sortie









Un peu de détails…

- **type processCcdSdss.py:**

  processCcdSdss.py est /home/lsst/winter2013stack/Linux64/pipe_tasks/6.1.0.1+2/bin/processCcdSdss.py

----

from lsst.pipe.tasks.processCcdSdss import ProcessCcdSdssTask

ProcessCcdSdssTask.parseAndRun()

- **D'après la structure des sources, ProcessCcdSdssTask est présent dans:**

  $LSST_HOME/Linux64/pipe_tasks/6.1.0.1+2/python/lsst/pipe/tasks/processCcdSdss.py

Un peu de détails…

- **type processCcdSdss.py:**

  processCcdSdss.py est /home/lsst/winter2013stack/Linux64/pipe_tasks/6.1.0.1+2/bin/processCcdSdss.py

----

from lsst.pipe.tasks.processCcdSdss import ProcessCcdSdssTask

ProcessCcdSdssTask.parseAndRun()

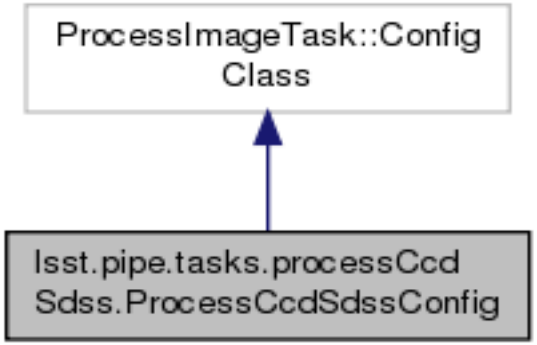
- **D'après la structure des sources, ProcessCcdSdssTask est présent dans:**

  $LSST_HOME/Linux64/pipe_tasks/6.1.0.1+2/python/lsst/pipe/tasks/processCcdSdss.py

Un peu de détails…

```
class ProcessCcdSdssConfig(ProcessImageTask.ConfigClass):
    """Config for ProcessCcdSdss"""
    removePedestal = pexConfig.Field(dtype=bool, default=True, doc="Remove SDSS pedestal from fpC file")
    pedestalVal = pexConfig.Field(dtype=int, default=1000, doc="Number of counts in the SDSS pedestal")

    removeOverlap =  pexConfig.Field(dtype=bool, default=True,
                                doc="Remove SDSS field overlap from fpC file")
    overlapSize = pexConfig.Field(dtype=int, default=128,
                                doc="Number of pixels to remove from top of the fpC file")
    loadSdssWcs = pexConfig.Field(
        dtype=bool, default=False,
        doc = ("Load WCS from asTrans; it can then be used as-is or updated by our own code, "
               "dependening on calibrate.astrometry parameters.")
    )
```

Un peu de détails…



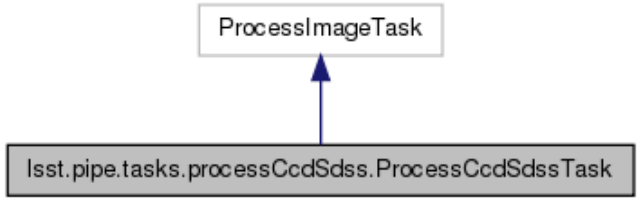

```
class ProcessCcdSdssConfig(ProcessImageTask.ConfigClass):
    """Config for ProcessCcdSdss"""
    removePedestal = pexConfig.Field(dtype=bool, default=True, doc="Remove SDSS pedestal from fpC file")
    pedestalVal = pexConfig.Field(dtype=int, default=1000, doc="Number of counts in the SDSS pedestal")

    removeOverlap =  pexConfig.Field(dtype=bool, default=True,
                                doc="Remove SDSS field overlap from fpC file")
    overlapSize = pexConfig.Field(dtype=int, default=128,
                            doc="Number of pixels to remove from top of the fpC file")
    loadSdssWcs = pexConfig.Field(
       dtype=bool, default=False,
       doc = ("Load WCS from asTrans; it can then be used as-is or updated by our own code, "
              "dependening on calibrate.astrometry parameters.")
    )
```

Paramètres de configuration

Un peu de détails…

```
class ProcessCcdSdssTask(ProcessImageTask):
    """Process a CCD for SDSS
    """
    ConfigClass = ProcessCcdSdssConfig
    _DefaultName = "processCcd"
    dataPrefix = ""

    def __init__(self, **kwargs):
        ProcessImageTask.__init__(self, **kwargs)

    @classmethod
    def _makeArgumentParser(cls):
        return pipeBase.ArgumentParser(name=cls._DefaultName, datasetType="fpC")

    def makeIdFactory(self, sensorRef):
        expBits = sensorRef.get("ccdExposureId_bits")
        expId = long(sensorRef.get("ccdExposureId"))
        return afwTable.IdFactory.makeSource(expId, 64 - expBits)

    @pipeBase.timeMethod
    def makeExp(self, sensorRef):
        image = sensorRef.get("fpC").convertF()
        if self.config.removePedestal:
            image -= self.config.pedestalVal
        mask  = sensorRef.get("fpM")
        wcs   = sensorRef.get("asTrans")
        calib, gain = sensorRef.get("tsField")
        var   = afwImage.ImageF(image, True)
        var  /= gain

        mi    = afwImage.MaskedImageF(image, mask, var)

        if self.config.removeOverlap:
            bbox   = mi.getBBox()
            begin  = bbox.getBegin()
            extent = bbox.getDimensions()
            extent -= afwGeom.Extent2I(0, self.config.overlapSize)
            tbbox  = afwGeom.BoxI(begin, extent)
            mi     = afwImage.MaskedImageF(mi, tbbox, True)

        exp   = afwImage.ExposureF(mi, wcs)
        exp.setCalib(calib)
        det = afwCameraGeom.Detector(
            afwCameraGeom.Id("%s%d" % (sensorRef.dataId["filter"],
sensorRef.dataId["camcol"]))
        )
        exp.setDetector(det)
        exp.setFilter(afwImage.Filter(sensorRef.dataId['filter']))

        # Install the SDSS PSF here; if we want to overwrite it later, we can.
        psf = sensorRef.get('psField')
        exp.setPsf(psf)

        return exp

    @pipeBase.timeMethod
    def run(self, sensorRef):
        """Process a CCD: including source detection, photometry and WCS determination

        @param sensorRef: sensor-level butler data reference to SDSS fpC file
        @return pipe_base Struct containing these fields:
        - exposure: calibrated exposure (calexp): as computed if config.doCalibrate,
            else as upersisted and updated if config.doDetection, else None
        - calib: object returned by calibration process if config.doCalibrate, else None
        - apCorr: aperture correction: as computed config.doCalibrate, else as unpersisted
            if config.doMeasure, else None
        - sources: detected source if config.doPhotometry, else None
        """
        self.log.info("Processing %s" % (sensorRef.dataId))

        if self.config.doCalibrate:
            exp = self.makeExp(sensorRef)
            if self.config.loadSdssWcs:
                self.log.info("Loading WCS from asTrans")
                wcs = sensorRef.get("asTrans")
                exp.setWcs(wcs)
        else:
            exp = None

        # delegate most of the work to ProcessImageTask
        result = self.process(sensorRef, exp)
        return result
```

Un peu de détails…




```
class ProcessCcdSdssTask(ProcessImageTask):
    """Process a CCD for SDSS
    """
    ConfigClass = ProcessCcdSdssConfig
    _DefaultName = "processCcd
    dataPrefix = ""

    def __init__(self, **kwargs):
        ProcessImageTask.__init__(self, **kwargs)

    @classmethod
    def _makeArgumentParser(cls):
        return pipeBase.ArgumentParser(name=cls._DefaultName, datasetType="fpC")

    def makeIdFactory(self, sensorRef):
        expBits = sensorRef.get("ccdExposureId_bits")
        expId = long(sensorRef.get("ccdExposureId"))
        return afwTable.IdFactory.makeSource(expId, 64 - expBits)

    @pipeBase.timeMethod
    def makeExp(self, sensorRef):
        image = sensorRef.get("fpC").convertF()
        if self.config.removePedestal:
            image -= self.config.pedestalVal
        mask  = sensorRef.get("fpM")
        wcs   = sensorRef.get("asTrans")
        calib, gain = sensorRef.get("tsField")
        var   = afwImage.ImageF(image, True)
        var  /= gain

        mi    = afwImage.MaskedImageF(image, mask, var)

        if self.config.removeOverlap:
            bbox   = mi.getBBox()
            begin  = bbox.getBegin()
            extent = bbox.getDimensions()
            extent -= afwGeom.Extent2I(0, self.config.overlapSize)
            tbbox  = afwGeom.BoxI(begin, extent)
            mi     = afwImage.MaskedImageF(mi, tbbox, True)

        exp   = afwImage.ExposureF(mi, wcs)
        exp.setCalib(calib)
                                    m.Detector(
                                    d("%s%d" % (sensorRef.dataId["filter"],
                                    l"]))


                                    ge.Filter(sensorRef.dataId['filter']))

                                    here; if we want to overwrite it later, we can.
        psf = sensorRef.get('psField')
        exp.setPsf(psf)

        return exp

    @pipeBase.timeMethod
    def run(self, sensorRef):
        """Process a CCD: including source detection, photometry and WCS determination

        @param sensorRef: sensor-level butler data reference to SDSS fpC file
        @return pipe_base Struct containing these fields:
        - exposure: calibrated exposure (calexp): as computed if config.doCalibrate,
            else as upersisted and updated if config.doDetection, else None
        - calib: object returned by calibration process if config.doCalibrate, else None
        - apCorr: aperture correction: as computed config.doCalibrate, else as unpersisted
            if config.doMeasure, else None
        - sources: detected source if config.doPhotometry, else None
        """
        self.log.info("Processing %s" % (sensorRef.dataId))

        if self.config.doCalibrate:
            exp = self.makeExp(sensorRef)
            if self.config.loadSdssWcs:
                self.log.info("Loading WCS from asTrans")
                wcs = sensorRef.get("asTrans")
                exp.setWcs(wcs)
        else:
            exp = None

        # delegate most of the work to ProcessImageTask
        result = self.process(sensorRef, exp)
        return result
```

Un peu de détails…

```
class ProcessCcdSdssTask(ProcessImageTask):
    """Process a CCD for SDSS
    """
    ConfigClass = ProcessCcdSdssConfig
    _DefaultName = "processCcd"
    dataPrefix = ""

    def __init__(self, **kwargs):
        ProcessImageTask.__init__(self, **kwargs)

    @classmethod
    def _makeArgumentParser(cls):
        return pipeBase.ArgumentParser(name=cls._DefaultName, datasetType="fpC")

    def makeIdFactory(self, sensorRef):
        expBits = sensorRef.get("ccdExposureId_bits")
        expId = long(sensorRef.get("ccdExposureId"))
        return afwTable.IdFactory.makeSource(expId, 64 - expBits)

    @pipeBase.timeMethod
    def makeExp(self, sensorRef):
        image = sensorRef.get("fpC").convertF()
        if self.config.removePedestal:
            image -= self.config.pedestalVal
        mask  = sensorRef.get("fpM")
        wcs   = sensorRef.get("asTrans")
        calib, gain = sensorRef.get("tsField")
        var   = afwImage.ImageF(image, True)
        var  /= gain

        mi    = afwImage.MaskedImageF(image, mask, var)

        if self.config.removeOverlap:
            bbox   = mi.getBBox()
            begin  = bbox.getBegin()
            extent = bbox.getDimensions()
            extent -= afwGeom.Extent2I(0, self.config.overlapSize)
            tbbox  = afwGeom.BoxI(begin, extent)
            mi     = afwImage.MaskedImageF(mi, tbbox, True)

        exp   = afwImage.ExposureF(mi, wcs)
        exp.setCalib(calib)
        det = afwCameraGeom.Detector(
            afwCameraGeom.Id("%s%d" % (sensorRef.dataId["filter"],
sensorRef.dataId["camcol"]))
        )
        exp.setDetector(det)
        exp.setFilter(afwImage.Filter(sensorRef.dataId['filter']))

        # Install the SDSS PSF here; if we want to overwrite it later, we can.
        psf = sensorRef.get('psField')
        exp.setPsf(psf)

        return exp

    @pipeBase.timeMethod
    def run(self, sensorRef):
        """Process a CCD: including source detection, photometry and WCS determination

        @param sensorRef: sensor-level butler data reference to SDSS fpC file
        @return pipe_base Struct containing these fields:
        - exposure: calibrated exposure (calexp): as computed if config.doCalibrate,
            else as upersisted and updated if config.doDetection, else None
        - calib: object returned by calibration process if config.doCalibrate, else None
        - apCorr: aperture correction: as computed config.doCalibrate, else as unpersisted
            if config.doMeasure, else None
        - sources: detected source if config.doPhotometry, else None
        """
        self.log.info("Processing %s" % (sensorRef.dataId))

        if self.config.doCalibrate:
            exp = self.makeExp(sensorRef)
            if self.config.loadSdssWcs:
                self.log.info("Loading WCS from asTrans")
                wcs = sensorRef.get("asTrans")
                exp.setWcs(wcs)
        else:
            exp = None

        # delegate most of the work to ProcessImageTask
        result = self.process(sensorRef, exp)
        return result
```

Un peu de détails…

```
class ProcessCcdSdssTask(ProcessImageTask):
    """Process a CCD for SDSS
    """
    ConfigClass = ProcessCcdSdssConfig
    _DefaultName = "processCcd"
    dataPrefix = ""

    def __init__(self, **kwargs):
        ProcessImageTask.__init__(self, **kwargs)

    @classmethod
    def _makeArgumentParser(cls):
        return pipeBase.ArgumentParser(name=cls._DefaultName, datasetType="fpC")

    def makeIdFactory(self, sensorRef):
        expBits = sensorRef.get("ccdExposureId_bits")
        expId = long(sensorRef.get("ccdExposureId"))
        return afwTable.IdFactory.makeSource(expId, 64 - expBits)

    @pipeBase.timeMethod
    def makeExp(self, sensorRef):
        image = sensorRef.get("fpC").convertF()
        if self.config.removePedestal:
            image -= self.config.pedestalVal
        mask  = sensorRef.get("fpM")
        wcs   = sensorRef.get("asTrans")
        calib, gain = sensorRef.get("tsField")
        var   = afwImage.ImageF(image, True)
        var  /= gain

        mi    = afwImage.MaskedImageF(image, mask, var)

        if self.config.removeOverlap:
            bbox   = mi.getBBox()
            begin  = bbox.getBegin()
            extent = bbox.getDimensions()
            extent -= afwGeom.Extent2I(0, self.config.overlapSize)
            tbbox  = afwGeom.BoxI(begin, extent)
            mi     = afwImage.MaskedImageF(mi, tbbox, True)
        exp   = afwImage.ExposureF(mi, wcs)
        exp.setCalib(calib)
        det = afwCameraGeom.Detector(
            afwCameraGeom.Id("%s%d" % (sensorRef.dataId["filter"],
sensorRef.dataId["camcol"]))
        )
        exp.setDetector(det)
        exp.setFilter(afwImage.Filter(sensorRef.dataId['filter']))

        # Install the SDSS PSF here; if we want to overwrite it later, we can.
        psf = sensorRef.get('psField')
        exp.setPsf(psf)

        return exp

    @pipeBase.timeMethod
    def run(self, sensorRef):
        """Process a CCD: including source detection, photometry and WCS determination

        @param sensorRef: sensor-level butler data reference to SDSS fpC file
        @return pipe_base Struct containing these fields:
        - exposure: calibrated exposure (calexp): as computed if config.doCalibrate,
            else as upersisted and updated if config.doDetection, else None
        - calib: object returned by calibration process if config.doCalibrate, else None
        - apCorr: aperture correction: as computed config.doCalibrate, else as unpersisted
            if config.doMeasure, else None
        - sources: detected source if config.doPhotometry, else None
        """
        self.log.info("Processing %s" % (sensorRef.dataId))

        if self.config.doCalibrate:
            exp = self.makeExp(sensorRef)
            if self.config.loadSdssWcs:
                self.log.info("Loading WCS from asTrans")
                wcs = sensorRef.get("asTrans")
                exp.setWcs(wcs)
        else:
            exp = None

        # delegate most of the work to ProcessImageTask
        result = self.process(sensorRef, exp)
        return result
```

Un peu de détails…

```
class ProcessCcdSdssTask(ProcessImageTask):
    """Process a CCD for SDSS
    """
    ConfigClass = ProcessCcdSdssConfig
    _DefaultName = "processCcd"
    dataPrefix = ""

    def __init__(self, **kwargs):
        ProcessImageTask.__init__(self, **kwargs)

    @classmethod
    def _makeArgumentParser(cls):
        return pipeBase.ArgumentParser(name=cls._DefaultName, datasetType="fpC")

    def makeIdFactory(self, sensorRef):
        expBits = sensorRef.get("ccdExposureId_bits")
        expId = long(sensorRef.get("ccdExposureId"))
        return afwTable.IdFactory.makeSource(expId, 64 - expBits)

    @pipeBase.timeMethod
    def makeExp(self, sensorRef):
        image = sensorRef.get("fpC").convertF()
        if self.config.removePedestal:
            image -= self.config.pedestalVal
        mask  = sensorRef.get("fpM")
        wcs   = sensorRef.get("asTrans")
        calib, gain = sensorRef.get("tsField")
        var   = afwImage.ImageF(image, True)
        var  /= gain

        mi    = afwImage.MaskedImageF(image, mask, var)

        if self.config.removeOverlap:
            bbox   = mi.getBBox()
            begin  = bbox.getBegin()
            extent = bbox.getDimensions()
            extent -= afwGeom.Extent2I(0, self.config.overlapSize)
            tbbox  = afwGeom.BoxI(begin, extent)
            mi     = afwImage.MaskedImageF(mi, tbbox, True)

        exp   = afwImage.ExposureF(mi, wcs)
        exp.setCalib(calib)
        det = afwCameraGeom.Detector(
            afwCameraGeom.Id("%s%d" % (sensorRef.dataId["filter"],
sensorRef.dataId["camcol"]))
        )
        exp.setDetector(det)
        exp.setFilter(afwImage.Filter(sensorRef.dataId['filter']))

        # Install the SDSS PSF here; if we want to overwrite it later, we can.
        psf = sensorRef.get('psField')
        exp.setPsf(psf)

        return exp

    @pipeBase.timeMethod
    def run(self, sensorRef):
        """Process a CCD: including source detection, photometry and WCS determination

        @param sensorRef: sensor-level butler data reference to SDSS fpC file
        @return pipe_base Struct containing these fields:
        - exposure: calibrated exposure (calexp): as computed if config.doCalibrate,
            else as upersisted and updated if config.doDetection, else None
        - calib: object returned by calibration process if config.doCalibrate, else None
        - apCorr: aperture correction: as computed config.doCalibrate, else as unpersisted
            if config.doMeasure, else None
        - sources: detected sources if config.doPhotometry, else None
        """
        self.log.info("Processing %s" % (sensorRef.dataId))

        if self.config.doCalibrate:
            exp = self.makeExp(sensorRef)
            if self.config.loadSdssWcs:
                self.log.info("Loading WCS from asTrans")
                wcs = sensorRef.get("asTrans")
                exp.setWcs(wcs)
        else:
            exp = None

        # delegate most of the work to ProcessImageTask
        result = self.process(sensorRef, exp)
        return result
```

Soustraction pieds

Chargement calibration et gain (fichiers tsField)

Application gain

Application mask

Un peu de détails…

```
class ProcessCcdSdssTask(ProcessImageTask):
    """Process a CCD for SDSS
    """
    ConfigClass = ProcessCcdSdssConfig
    _DefaultName = "processCcd"
    dataPrefix = ""

    def __init__(self, **kwargs):
        ProcessImageTask.__init__(self, **kwargs)

    @classmethod
    def _makeArgumentParser(cls):
        return pipeBase.ArgumentParser(name=cls._DefaultName, datasetType="fpC")

    def makeIdFactory(self, sensorRef):
        expBits = sensorRef.get("ccdExposureId_bits")
        expId = long(sensorRef.get("ccdExposureId"))
        return afwTable.IdFactory.makeSource(expId, 64 - expBits)

    @pipeBase.timeMethod
    def makeExp(self, sensorRef):
        image = sensorRef.get("fpC").convertF()
        if self.config.removePedestal:
            image -= self.config.pedestalVal
        mask  = sensorRef.get("fpM")
        wcs   = sensorRef.get("asTrans")
        calib, gain = sensorRef.get("tsField")
        var   = afwImage.ImageF(image, True)
        var  /= gain

        mi    = afwImage.MaskedImageF(image, mask, var)

        if self.config.removeOverlap:
            bbox   = mi.getBBox()
            begin  = bbox.getBegin()
            extent = bbox.getDimensions()
            extent -= afwGeom.Extent2I(0, self.config.overlapSize)
            tbbox  = afwGeom.BoxI(begin, extent)
            mi     = afwImage.MaskedImageF(mi, tbbox, True)
        exp   = afwImage.ExposureF(mi, wcs)
        exp.setCalib(calib)
        det = afwCameraGeom.Detector(
            afwCameraGeom.Id("%s%d" % (sensorRef.dataId["filter"],
sensorRef.dataId["camcol"]))
        )
        exp.setDetector(det)
        exp.setFilter(afwImage.Filter(sensorRef.dataId['filter']))

        # Install the SDSS PSF here; if we want to overwrite it later, we can.
        psf = sensorRef.get('psField')
        exp.setPsf(psf)

        return exp

    @pipeBase.timeMethod
    def run(self, sensorRef):
        """Process a CCD: including source detection, photometry and WCS determination

        @param sensorRef: sensor-level butler data reference to SDSS fpC file
        @return pipe_base Struct containing these fields:
        - exposure: calibrated exposure (calexp): as computed if config.doCalibrate,
            else as upersisted and updated if config.doDetection, else None
        - calib: object returned by calibration process if config.doCalibrate, else None
        - apCorr: aperture correction: as computed config.doCalibrate, else as unpersisted
            if config.doMeasure, else None
        - sources: detected source if config.doPhotometry, else None
        """
        self.log.info("Processing %s" % (sensorRef.dataId))

        if self.config.doCalibrate:
            exp = self.makeExp(sensorRef)
            if self.config.loadSdssWcs:
                self.log.info("Loading WCS from asTrans")
                wcs = sensorRef.get("asTrans")
                exp.setWcs(wcs)
        else:
            exp = None

        # delegate most of the work to ProcessImageTask
        result = self.process(sensorRef, exp)
        return result
```

Application calib

PSF

WCS

Un peu de détails…

```
class ProcessCcdSdssTask(ProcessImageTask):
    """Process a CCD for SDSS
    """
    ConfigClass = ProcessCcdSdssConfig
    _DefaultName = "processCcd"
    dataPrefix = ""

    def __init__(self, **kwargs):
        ProcessImageTask.__init__(self, **kwargs)

    @classmethod
    def _makeArgumentParser(cls):
        return pipeBase.ArgumentParser(name=cls._DefaultName, datasetType="fpC")

    def makeIdFactory(self, sensorRef):
        expBits = sensorRef.get("ccdExposureId_bits")
        expId = long(sensorRef.get("ccdExposureId"))
        return afwTable.IdFactory.makeSource(expId, 64 - expBits)

    @pipeBase.timeMethod
    def makeExp(self, sensorRef):
        image = sensorRef.get("fpC").convertF()
        if self.config.removePedestal:
            image -= self.config.pedestalVal
        mask  = sensorRef.get("fpM")
        wcs   = sensorRef.get("asTrans")
        calib, gain = sensorRef.get("tsField")
        var   = afwImage.ImageF(image, True)
        var  /= gain

        mi    = afwImage.MaskedImageF(image, mask, var)

        if self.config.removeOverlap:
            bbox   = mi.getBBox()
            begin  = bbox.getBegin()
            extent = bbox.getDimensions()
            extent -= afwGeom.Extent2I(0, self.config.overlapSize)
            tbbox  = afwGeom.BoxI(begin, extent)
            mi     = afwImage.MaskedImageF(mi, tbbox, True)
        exp   = afwImage.ExposureF(mi, wcs)
        exp.setCalib(calib)
        det = afwCameraGeom.Detector(
            afwCameraGeom.Id("%s%d" % (sensorRef.dataId["filter"],
sensorRef.dataId["camcol"]))
        )
        exp.setDetector(det)
        exp.setFilter(afwImage.Filter(sensorRef.dataId['filter']))

        # Install the SDSS PSF here; if we want to overwrite it later, we can.
        psf = sensorRef.get('psField')
        exp.setPsf(psf)

        return exp

    @pipeBase.timeMethod
    def run(self, sensorRef):
        """Process a CCD: including source detection, photometry and WCS determination

        @param sensorRef: sensor-level butler data reference to SDSS fpC file
        @return pipe_base Struct containing these fields:
        - exposure: calibrated exposure (calexp): as computed if config.doCalibrate,
            else as upersisted and updated if config.doDetection, else None
        - calib: object returned by calibration process if config.doCalibrate, else None
        - apCorr: aperture correction: as computed config.doCalibrate, else as unpersisted
            if config.doMeasure, else None
        - sources: detected source if config.doPhotometry, else None
        """
        self.log.info("Processing %s" % (sensorRef.dataId))

        if self.config.doCalibrate:
            exp = self.makeExp(sensorRef)
            if self.config.loadSdssWcs:
                self.log.info("Loading WCS from asTrans")
                wcs = sensorRef.get("asTrans")
                exp.setWcs(wcs)
        else:
            exp = None

        # delegate most of the work to ProcessImageTask
        result = self.process(sensorRef, exp)
        return result
```

Un peu de détails…

$LSST_HOME/pipe_tasks/6.1.0.1+2/python/lsst/pipe/tasks/processImage.py

class ProcessImageTask(pipeBase.CmdLineTask):

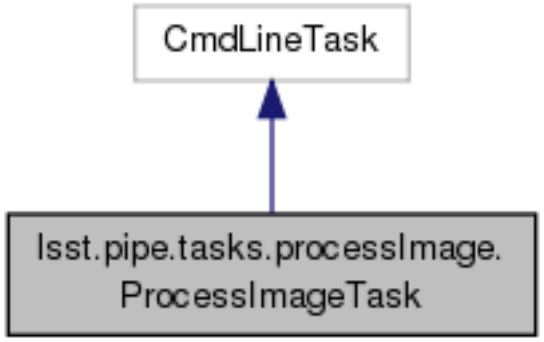
```
 def __init__(self, **kwargs):
      pipeBase.CmdLineTask.__init__(self, **kwargs)
      self.makeSubtask("calibrate")
      self.schema = kwargs.pop("schema", None)
      if self.schema is None:
         self.schema = afwTable.SourceTable.makeMinimalSchema()
      # add fields needed to identify stars used in the calibration step
      self.makeSubtask("calibrate")

      # Setup our schema by starting with fields we want to propagate
from icSrc.
      calibSchema = self.calibrate.schema
      self.schemaMapper = afwTable.SchemaMapper(calibSchema)

self.schemaMapper.addMinimalSchema(afwTable.SourceTable.makeMi
nimalSchema(), False)
      self.calibSourceKey = self.schemaMapper.addOutputField(
         afwTable.Field["Flag"]("calib.detected", "Source was detected as
an icSrc")
         )
      for key in self.calibrate.getCalibKeys():
         self.schemaMapper.addMapping(key)
      self.schema = self.schemaMapper.getOutputSchema()
      self.algMetadata = dafBase.PropertyList()
      if self.config.doDetection:
         self.makeSubtask("detection", schema=self.schema)
      if self.config.doDeblend:
         self.makeSubtask("deblend", schema=self.schema)
      if self.config.doMeasurement:
         self.makeSubtask("measurement", schema=self.schema,
algMetadata=self.algMetadata)
```

Un peu de détails…

$LSST_HOME/pipe_tasks/6.1.0.1+2/python/lsst/pipe/tasks/processImage.py

class ProcessImageTask(pipeBase.CmdLineTask):

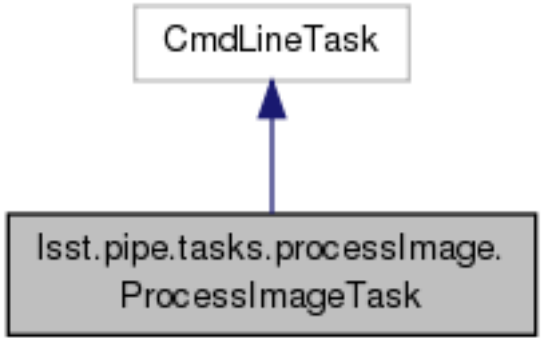
```
 def __init__(self, **kwargs):
      pipeBase.CmdLineTask.__init__(self, **kwargs)
      self.makeSubtask("calibrate")
      self.schema = kwargs.pop("schema", None)
      if self.schema is None:
         self.schema = afwTable.SourceTable.makeMinimalSchema()
      # add fields needed to identify stars used in the calibration step
      self.makeSubtask("calibrate")

      # Setup our schema by starting with fields we want to propagate
from icSrc.
      calibSchema = self.calibrate.schema
      self.schemaMapper = afwTable.SchemaMapper(calibSchema)

self.schemaMapper.addMinimalSchema(afwTable.SourceTable.makeMi
nimalSchema(), False)
      self.calibSourceKey = self.schemaMapper.addOutputField(
         afwTable.Field["Flag"]("calib.detected", "Source was detected as
an icSrc")
         )
      for key in self.calibrate.getCalibKeys():
         self.schemaMapper.addMapping(key)
      self.schema = self.schemaMapper.getOutputSchema()
      self.algMetadata = dafBase.PropertyList()
      if self.config.doDetection:
         self.makeSubtask("detection", schema=self.schema)
      if self.config.doDeblend:
         self.makeSubtask("deblend", schema=self.schema)
      if self.config.doMeasurement:
         self.makeSubtask("measurement", schema=self.schema,
algMetadata=self.algMetadata)
```



Un peu de détails…

$LSST_HOME/pipe_tasks/6.1.0.1+2/python/lsst/pipe/tasks/processImage.py

```
class ProcessImageTask(pipeBase.CmdLineTask):


 def __init__(self, **kwargs):
      pipeBase.CmdLineTask.__init__(self, **kwargs)
      self.makeSubtask("calibrate")
      self.schema = kwargs.pop("schema", None)
      if self.schema is None:
         self.schema = afwTable.SourceTable.makeMinimalSchema()
      # add fields needed to identify stars used in the calibration step
      self.makeSubtask("calibrate")

      # Setup our schema by starting with fields we want to propagate
from icSrc.
      calibSchema = self.calibrate.schema
      self.schemaMapper = afwTable.SchemaMapper(calibSchema)

self.schemaMapper.addMinimalSchema(afwTable.SourceTable.makeMi
nimalSchema(), False)
      self.calibSourceKey = self.schemaMapper.addOutputField(
         afwTable.Field["Flag"]("calib.detected", "Source was detected as
an icSrc")
         )
      for key in self.calibrate.getCalibKeys():
         self.schemaMapper.addMapping(key)
      self.schema = self.schemaMapper.getOutputSchema()
      self.algMetadata = dafBase.PropertyList()
      if self.config.doDetection:
         self.makeSubtask("detection", schema=self.schema)
      if self.config.doDeblend:
         self.makeSubtask("deblend", schema=self.schema)
      if self.config.doMeasurement:
         self.makeSubtask("measurement", schema=self.schema,
algMetadata=self.algMetadata)
```



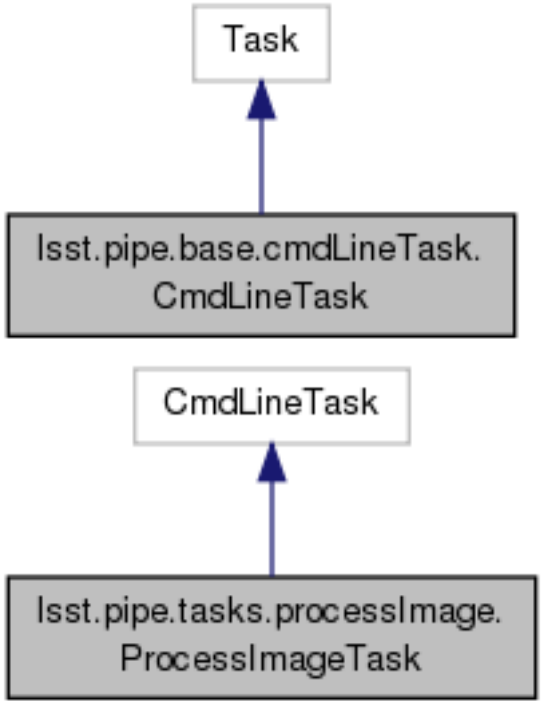
Création de subtasks
Enregistrement en tant que champ de l'objet parent

Un peu de détails…

$LSST_HOME/pipe_tasks/6.1.0.1+2/python/lsst/pipe/tasks/processImage.py

class ProcessImageTask(pipeBase.CmdLineTask):

```
def __init__(self, **kwargs):
    pipeBase.CmdLineTask.__init__(self, **kwargs)
    self.makeSubtask("calibrate")
    self.schema = kwargs.pop("schema", None)
    if self.schema is None:
        self.schema = afwTable.SourceTable.makeMinimalSchema()
    # add fields needed to identify stars used in the calibration step
    self.makeSubtask("calibrate")

    # Setup our schema by starting with fields we want to propagate from icSrc.
    calibSchema = self.calibrate.schema
    self.schemaMapper = afwTable.SchemaMapper(calibSchema)

self.schemaMapper.addMinimalSchema(afwTable.SourceTable.makeMinimalSchema(), False)
    self.calibSourceKey = self.schemaMapper.addOutputField(
        afwTable.Field["Flag"]("calib.detected", "Source was detected as an icSrc")
        )
    for key in self.calibrate.getCalibKeys():
        self.schemaMapper.addMapping(key)
    self.schema = self.schemaMapper.getOutputSchema()
    self.algMetadata = dafBase.PropertyList()
    if self.config.doDetection:
        self.makeSubtask("detection", schema=self.schema)
    if self.config.doDeblend:
        self.makeSubtask("deblend", schema=self.schema)
    if self.config.doMeasurement:
        self.makeSubtask("measurement", schema=self.schema, algMetadata=self.algMetadata)
```



Création de subtasks
Enregistrement en tant que champ de l'objet parent

Un peu de détails…

$LSST_HOME/pipe_tasks/6.1.0.1+2/python/lsst/pipe/tasks/processImage.py

class ProcessImageTask(pipeBase.CmdLineTask):

```
def process(self, dataRef, inputExposure):
    """Process an Image

    @param dataRef: data reference that corresponds to the input image
    @param inputExposure:  exposure to process

    @return pipe_base Struct containing these fields:
    - postIsrExposure: exposure after ISR performed if calib.doIsr or config.doCalibrate, else None
    - exposure: calibrated exposure (calexp): as computed if config.doCalibrate,
      else as upersisted and updated if config.doDetection, else None
    - calib: object returned by calibration process if config.doCalibrate, else None
    - apCorr: aperture correction: as computed config.doCalibrate, else as unpersisted
      if config.doMeasure, else None
    - sources: detected source if config.doPhotometry, else None
    """
    idFactory = self.makeIdFactory(dataRef)

    # initialize outputs
    calExposure = None
    calib = None
    apCorr = None
    sources = None
    psf = None
    backgrounds = []
    if self.config.doCalibrate:
        calib = self.calibrate.run(inputExposure, idFactory=idFactory)
        psf = calib.psf
        calExposure = calib.exposure
        apCorr = calib.apCorr
```

```
    if self.config.doWriteCalibrate:
            dataRef.put(calib.sources, self.dataPrefix + "icSrc")
            if calib.psf is not None:
                dataRef.put(calib.psf, self.dataPrefix + "psf")
            if calib.apCorr is not None:
                dataRef.put(calib.apCorr, self.dataPrefix + "apCorr")
            if calib.matches is not None and self.config.doWriteCalibrateMatches:
                normalizedMatches = afwTable.packMatches(calib.matches)
                normalizedMatches.table.setMetadata(calib.matchMeta)
                dataRef.put(normalizedMatches, self.dataPrefix + "icMatch")
        try:
            for bg in calib.backgrounds:
                backgrounds.append(bg)
        except TypeError:
            backgrounds.append(calib.backgrounds)
        except AttributeError:
            self.log.warn("The calibration task did not return any backgrounds.  Any background subtracted in the calibration process cannot be persisted.")
    else:
        calib = None

    if self.config.doDetection:
        if calExposure is None:
            if not dataRef.datasetExists(self.dataPrefix + "calexp"):
                raise pipeBase.TaskError("doCalibrate false, doDetection true and calexp does not exist")
            calExposure = dataRef.get(self.dataPrefix + "calexp")
        if calib is None or calib.psf is None:
            psf = dataRef.get(self.dataPrefix + "psf")
            calExposure.setPsf(psf)
        table = afwTable.SourceTable.make(self.schema, idFactory)
        table.setMetadata(self.algMetadata)
        detections = self.detection.makeSourceCatalog(table, calExposure)
        sources = detections.sources
        fpSets = detections.fpSets
        if fpSets.background:
            backgrounds.append(fpSets.background)

    if self.config.doDeblend:
        if calExposure is None:
            calExposure = dataRef.get(self.dataPrefix + 'calexp')
        if psf is None:
            psf = dataRef.get(self.dataPrefix + 'psf')

        self.deblend.run(calExposure, sources, psf)

    if self.config.doMeasurement:
        if apCorr is None:
            apCorr = dataRef.get(self.dataPrefix + "apCorr")
        self.measurement.run(calExposure, sources, apCorr)
```

Un peu de détails…

$LSST_HOME/pipe_tasks/6.1.0.1+2/python/lsst/pipe/tasks/processImage.py

class ProcessImageTask(pipeBase.CmdLineTask):

```
def process(self, dataRef, inputExposure):
    """Process an Image

    @param dataRef: data reference that corresponds to the input image
    @param inputExposure:  exposure to process

    @return pipe_base Struct containing these fields:
    - postIsrExposure: exposure after ISR performed if calib.doIsr or config.doCalibrate,
else None
    - exposure: calibrated exposure (calexp): as computed if config.doCalibrate,
      else as upersisted and updated if config.doDetection, else None
    - calib: object returned by calibration process if config.doCalibrate, else None
    - apCorr: aperture correction: as computed config.doCalibrate, else as unpersisted
      if config.doMeasure, else None
    - sources: detected source if config.doPhotometry, else None
    """
    idFactory = self.makeIdFactory(dataRef)

    # initialize outputs
    calExposure = None
    calib = None
    apCorr = None
    sources = None
    psf = None
    backgrounds = []
    if self.config.doCalibrate:
        calib = self.calibrate.run(inputExposure, idFactory=idFactory)
        psf = calib.psf
        calExposure = calib.exposure
        apCorr = calib.apCorr
```

```
if self.config.doWriteCalibrate:
            dataRef.put(calib.sources, self.dataPrefix + "icSrc")
            if calib.psf is not None:
                dataRef.put(calib.psf, self.dataPrefix + "psf")
            if calib.apCorr is not None:
                dataRef.put(calib.apCorr, self.dataPrefix + "apCorr")
            if calib.matches is not None and self.config.doWriteCalibrateMatches:
                normalizedMatches = afwTable.packMatches(calib.matches)
                normalizedMatches.table.setMetadata(calib.matchMeta)
                dataRef.put(normalizedMatches, self.dataPrefix + "icMatch")
        try:
            for bg in calib.backgrounds:
                backgrounds.append(bg)
        except TypeError:
            backgrounds.append(calib.backgrounds)
        except AttributeError:
            self.log.warn("The calibration task did not return any backgrounds.  Any background
subtracted in the calibration process cannot be persisted.")
    else:
        calib = None

    if self.config.doDetection:
        if calExposure is None:
            if not dataRef.datasetExists(self.dataPrefix + "calexp"):
                raise pipeBase.TaskError("doCalibrate false, doDetection true and calexp does not exist")
            calExposure = dataRef.get(self.dataPrefix + "calexp")
        if calib is None or calib.psf is None:
            psf = dataRef.get(self.dataPrefix + "psf")
            calExposure.setPsf(psf)
        table = afwTable.SourceTable.make(self.schema, idFactory)
        table.setMetadata(self.algMetadata)
        detections = self.detection.makeSourceCatalog(table, calExposure)
        sources = detections.sources
        fpSets = detections.fpSets
        if fpSets.background:
            backgrounds.append(fpSets.background)

    if self.config.doDeblend:
        if calExposure is None:
            calExposure = dataRef.get(self.dataPrefix + 'calexp')
        if psf is None:
            psf = dataRef.get(self.dataPrefix + 'psf')

        self.deblend.run(calExposure, sources, psf)

    if self.config.doMeasurement:
        if apCorr is None:
            apCorr = dataRef.get(self.dataPrefix + "apCorr")
        self.measurement.run(calExposure, sources, apCorr)
```

Un peu de détails…

Préparation pour sauvegarde

$LSST_HOME/pipe_tasks/6.1.0.1+2/python/lsst/pipe/tasks/processImage.py

class ProcessImageTask(pipeBase.CmdLineTask):

```
def process(self, dataRef, inputExposure):
    """Process an Image

    @param dataRef: data reference that corresponds to the input image
    @param inputExposure:  exposure to process

    @return pipe_base Struct containing these fields:
    - postIsrExposure: exposure after ISR performed if calib.doIsr or config.doCalibrate, else None
    - exposure: calibrated exposure (calexp): as computed if config.doCalibrate,
      else as upersisted and updated if config.doDetection, else None
    - calib: object returned by calibration process if config.doCalibrate, else None
    - apCorr: aperture correction: as computed config.doCalibrate, else as unpersisted
      if config.doMeasure, else None
    - sources: detected source if config.doPhotometry, else None
    """
    idFactory = self.makeIdFactory(dataRef)

    # initialize outputs
    calExposure = None
    calib = None
    apCorr = None
    sources = None
    psf = None
    backgrounds = []
    if self.config.doCalibrate:
        calib = self.calibrate.run(inputExposure, idFactory=idFactory)
        psf = calib.psf
        calExposure = calib.exposure
        apCorr = calib.apCorr
```

```
if self.config.doWriteCalibrate:
            dataRef.put(calib.sources, self.dataPrefix + "icSrc")
            if calib.psf is not None:
                dataRef.put(calib.psf, self.dataPrefix + "psf")
            if calib.apCorr is not None:
                dataRef.put(calib.apCorr, self.dataPrefix + "apCorr")
            if calib.matches is not None and self.config.doWriteCalibrateMatches:
                normalizedMatches = afwTable.packMatches(calib.matches)
                normalizedMatches.table.setMetadata(calib.matchMeta)
                dataRef.put(normalizedMatches, self.dataPrefix + "icMatch")
        try:
            for bg in calib.backgrounds:
                backgrounds.append(bg)
        except TypeError:
            backgrounds.append(calib.backgrounds)
        except AttributeError:
            self.log.warn("The calibration task did not return any backgrounds.  Any background subtracted in the calibration process cannot be persisted.")
    else:
        calib = None

    if self.config.doDetection:
        if calExposure is None:
            if not dataRef.datasetExists(self.dataPrefix + "calexp"):
                raise pipeBase.TaskError("doCalibrate false, doDetection true and calexp does not exist")
            calExposure = dataRef.get(self.dataPrefix + "calexp")
        if calib is None or calib.psf is None:
            psf = dataRef.get(self.dataPrefix + "psf")
            calExposure.setPsf(psf)
        table = afwTable.SourceTable.make(self.schema, idFactory)
        table.setMetadata(self.algMetadata)
        detections = self.detection.makeSourceCatalog(table, calExposure)
        sources = detections.sources
        fpSets = detections.fpSets
        if fpSets.background:
            backgrounds.append(fpSets.background)

    if self.config.doDeblend:
        if calExposure is None:
            calExposure = dataRef.get(self.dataPrefix + 'calexp')
        if psf is None:
            psf = dataRef.get(self.dataPrefix + 'psf')

        self.deblend.run(calExposure, sources, psf)

    if self.config.doMeasurement:
        if apCorr is None:
            apCorr = dataRef.get(self.dataPrefix + "apCorr")
        self.measurement.run(calExposure, sources, apCorr)
```

Un peu de détails…

$LSST_HOME/pipe_tasks/6.1.0.1+2/python/lsst/pipe/tasks/processImage.py

class ProcessImageTask(pipeBase.CmdLineTask):

```
def process(self, dataRef, inputExposure):
    """Process an Image

    @param dataRef: data reference that corresponds to the input image
    @param inputExposure:  exposure to process

    @return pipe_base Struct containing these fields:
    - postIsrExposure: exposure after ISR performed if calib.doIsr or config.doCalibrate, else None
    - exposure: calibrated exposure (calexp): as computed if config.doCalibrate,
      else as upersisted and updated if config.doDetection, else None
    - calib: object returned by calibration process if config.doCalibrate, else None
    - apCorr: aperture correction: as computed config.doCalibrate, else as unpersisted
      if config.doMeasure, else None
    - sources: detected source if config.doPhotometry, else None
    """
    idFactory = self.makeIdFactory(dataRef)

    # initialize outputs
    calExposure = None
    calib = None
    apCorr = None
    sources = None
    psf = None
    backgrounds = []
    if self.config.doCalibrate:
        calib = self.calibrate.run(inputExposure, idFactory=idFactory)
        psf = calib.psf
        calExposure = calib.exposure
        apCorr = calib.apCorr
```

```
    if self.config.doWriteCalibrate:
            dataRef.put(calib.sources, self.dataPrefix + "icSrc")
            if calib.psf is not None:
                dataRef.put(calib.psf, self.dataPrefix + "psf")
            if calib.apCorr is not None:
                dataRef.put(calib.apCorr, self.dataPrefix + "apCorr")
            if calib.matches is not None and self.config.doWriteCalibrateMatches:
                normalizedMatches = afwTable.packMatches(calib.matches)
                normalizedMatches.table.setMetadata(calib.matchMeta)
                dataRef.put(normalizedMatches, self.dataPrefix + "icMatch")
        try:
            for bg in calib.backgrounds:
                backgrounds.append(bg)
        except TypeError:
            backgrounds.append(calib.backgrounds)
        except AttributeError:
            self.log.warn("The calibration task did not return any backgrounds.  Any background subtracted in the calibration process cannot be persisted.")
    else:
        calib = None

    if self.config.doDetection:
        if calExposure is None:
            if not dataRef.datasetExists(self.dataPrefix + "calexp"):
                raise pipeBase.TaskError("doCalibrate false, doDetection true and calexp does not exist")
            calExposure = dataRef.get(self.dataPrefix + "calexp")
        if calib is None or calib.psf is None:
            psf = dataRef.get(self.dataPrefix + "psf")
            calExposure.setPsf(psf)
        table = afwTable.SourceTable.make(self.schema, idFactory)
        table.setMetadata(self.algMetadata)
        detections = self.detection.makeSourceCatalog(table, calExposure)
        sources = detections.sources
        fpSets = detections.fpSets
        if fpSets.background:
            backgrounds.append(fpSets.background)

    if self.config.doDeblend:
        if calExposure is None:
            calExposure = dataRef.get(self.dataPrefix + 'calexp')
        if psf is None:
            psf = dataRef.get(self.dataPrefix + 'psf')

        self.deblend.run(calExposure, sources, psf)

    if self.config.doMeasurement:
        if apCorr is None:
            apCorr = dataRef.get(self.dataPrefix + "apCorr")
        self.measurement.run(calExposure, sources, apCorr)
```

Image calibrée (calexposure)

Un peu de détails…

$LSST_HOME/pipe_tasks/6.1.0.1+2/python/lsst/pipe/tasks/processImage.py

class ProcessImageTask(pipeBase.CmdLineTask):

```
def process(self, dataRef, inputExposure):
    """Process an Image

    @param dataRef: data reference that corresponds to the input image
    @param inputExposure:  exposure to process

    @return pipe_base Struct containing these fields:
    - postIsrExposure: exposure after ISR performed if calib.doIsr or config.doCalibrate, else None
    - exposure: calibrated exposure (calexp): as computed if config.doCalibrate,
      else as upersisted and updated if config.doDetection, else None
    - calib: object returned by calibration process if config.doCalibrate, else None
    - apCorr: aperture correction: as computed config.doCalibrate, else as unpersisted
      if config.doMeasure, else None
    - sources: detected source if config.doPhotometry, else None
    """
    idFactory = self.makeIdFactory(dataRef)

    # initialize outputs
    calExposure = None
    calib = None
    apCorr = None
    sources = None
    psf = None
    backgrounds = []
    if self.config.doCalibrate:
        calib = self.calibrate.run(inputExposure, idFactory=idFactory)
        psf = calib.psf
        calExposure = calib.exposure
        apCorr = calib.apCorr
```

Catalogue de sources

```
    if self.config.doWriteCalibrate:
            dataRef.put(calib.sources, self.dataPrefix + "icSrc")
            if calib.psf is not None:
                dataRef.put(calib.psf, self.dataPrefix + "psf")
            if calib.apCorr is not None:
                dataRef.put(calib.apCorr, self.dataPrefix + "apCorr")
            if calib.matches is not None and self.config.doWriteCalibrateMatches:
                normalizedMatches = afwTable.packMatches(calib.matches)
                normalizedMatches.table.setMetadata(calib.matchMeta)
                dataRef.put(normalizedMatches, self.dataPrefix + "icMatch")
        try:
            for bg in calib.backgrounds:
                backgrounds.append(bg)
        except TypeError:
            backgrounds.append(calib.backgrounds)
        except AttributeError:
            self.log.warn("The calibration task did not return any backgrounds.  Any background subtracted in the calibration process cannot be persisted.")
    else:
        calib = None

    if self.config.doDetection:
        if calExposure is None:
            if not dataRef.datasetExists(self.dataPrefix + "calexp"):
                raise pipeBase.TaskError("doCalibrate false, doDetection true and calexp does not exist")
            calExposure = dataRef.get(self.dataPrefix + "calexp")
        if calib is None or calib.psf is None:
            psf = dataRef.get(self.dataPrefix + "psf")
            calExposure.setPsf(psf)
        table = afwTable.SourceTable.make(self.schema, idFactory)
        table.setMetadata(self.algMetadata)
        detections = self.detection.makeSourceCatalog(table, calExposure)
        sources = detections.sources
        fpSets = detections.fpSets
        if fpSets.background:
            backgrounds.append(fpSets.background)

    if self.config.doDeblend:
        if calExposure is None:
            calExposure = dataRef.get(self.dataPrefix + 'calexp')
        if psf is None:
            psf = dataRef.get(self.dataPrefix + 'psf')

        self.deblend.run(calExposure, sources, psf)

    if self.config.doMeasurement:
        if apCorr is None:
            apCorr = dataRef.get(self.dataPrefix + "apCorr")
        self.measurement.run(calExposure, sources, apCorr)
```

Application psf

Un peu de détails…

$LSST_HOME/pipe_tasks/6.1.0.1+2/python/lsst/pipe/tasks/processImage.py

class ProcessImageTask(pipeBase.CmdLineTask):

```
def process(self, dataRef, inputExposure):
    """Process an Image

    @param dataRef: data reference that corresponds to the input image
    @param inputExposure:  exposure to process

    @return pipe_base Struct containing these fields:
    - postIsrExposure: exposure after ISR performed if calib.doIsr or config.doCalibrate,
else None
    - exposure: calibrated exposure (calexp): as computed if config.doCalibrate,
      else as upersisted and updated if config.doDetection, else None
    - calib: object returned by calibration process if config.doCalibrate, else None
    - apCorr: aperture correction: as computed config.doCalibrate, else as unpersisted
      if config.doMeasure, else None
    - sources: detected source if config.doPhotometry, else None
    """
    idFactory = self.makeIdFactory(dataRef)

    # initialize outputs
    calExposure = None
    calib = None
    apCorr = None
    sources = None
    psf = None
    backgrounds = []
    if self.config.doCalibrate:
        calib = self.calibrate.run(inputExposure, idFactory=idFactory)
        psf = calib.psf
        calExposure = calib.exposure
        apCorr = calib.apCorr
```

```
if self.config.doWriteCalibrate:
            dataRef.put(calib.sources, self.dataPrefix + "icSrc")
            if calib.psf is not None:
                dataRef.put(calib.psf, self.dataPrefix + "psf")
            if calib.apCorr is not None:
                dataRef.put(calib.apCorr, self.dataPrefix + "apCorr")
            if calib.matches is not None and self.config.doWriteCalibrateMatches:
                normalizedMatches = afwTable.packMatches(calib.matches)
                normalizedMatches.table.setMetadata(calib.matchMeta)
                dataRef.put(normalizedMatches, self.dataPrefix + "icMatch")
        try:
            for bg in calib.backgrounds:
                backgrounds.append(bg)
        except TypeError:
            backgrounds.append(calib.backgrounds)
        except AttributeError:
            self.log.warn("The calibration task did not return any backgrounds.  Any background
subtracted in the calibration process cannot be persisted.")
    else:
        calib = None

    if self.config.doDetection:
        if calExposure is None:
            if not dataRef.datasetExists(self.dataPrefix + "calexp"):
                raise pipeBase.TaskError("doCalibrate false, doDetection true and calexp does not exist")
            calExposure = dataRef.get(self.dataPrefix + "calexp")
        if calib is None or calib.psf is None:
            psf = dataRef.get(self.dataPrefix + "psf")
            calExposure.setPsf(psf)
        table = afwTable.SourceTable.make(self.schema, idFactory)
        table.setMetadata(self.algMetadata)
        detections = self.detection.makeSourceCatalog(table, calExposure)
        sources = detections.sources
        fpSets = detections.fpSets
        if fpSets.background:
            backgrounds.append(fpSets.background)

    if self.config.doDeblend:
        if calExposure is None:
            calExposure = dataRef.get(self.dataPrefix + 'calexp')
        if psf is None:
            psf = dataRef.get(self.dataPrefix + 'psf')

        self.deblend.run(calExposure, sources, psf)

    if self.config.doMeasurement:
        if apCorr is None:
            apCorr = dataRef.get(self.dataPrefix + "apCorr")
        self.measurement.run(calExposure, sources, apCorr)
```

Corrections d'ouverture

Un peu de détails…

```
        if self.config.doWriteCalibrate:
    # wait until after detection and measurement, since detection sets detected mask bits and both require
    # a background subtracted exposure;
    # note that this overwrites an existing calexp if doCalibrate false

    if calExposure is None:
      self.log.warn("calibrated exposure is None; cannot save it")
    else:
      if self.config.persistBackgroundModel:
        self.log.warn("Persisting background models as an image")
        bg = backgrounds[0].getImageF()
        for b in backgrounds[1:]:
          bg += b.getImageF()
        dataRef.put(bg, self.dataPrefix+"calexpBackground")
        del bg
      else:
        mi = calExposure.getMaskedImage()
        for bg in backgrounds:
          mi += bg.getImageF()
```

mask + image de fond ?

```
        mi += bg.getImageF()
        dataRef.put(calExposure, self.dataPrefix + "calexp")

    if calib is not None:
      self.propagateCalibFlags(calib.sources, sources)

    if sources is not None and self.config.doWriteSources:
      if self.config.doWriteHeavyFootprintsInSources:
        sources.setWriteHeavyFootprints(True)
      dataRef.put(sources, self.dataPrefix + 'src')

    if self.config.doWriteSourceMatches:
      self.log.info("Matching src to reference catalogue" % (dataRef.dataId))
      srcMatches, srcMatchMeta = self.matchSources(calExposure, sources)

      normalizedSrcMatches = afwTable.packMatches(srcMatches)
      normalizedSrcMatches.table.setMetadata(srcMatchMeta)
      dataRef.put(normalizedSrcMatches, self.dataPrefix + "srcMatch")
    else:
      srcMatches = None; srcMatchMeta = None

    return pipeBase.Struct(
      inputExposure = inputExposure,
      exposure = calExposure,
      calib = calib,
      apCorr = apCorr,
      sources = sources,
      matches = srcMatches,
      matchMeta = srcMatchMeta,
      backgrounds = backgrounds,
    )
```

Un peu de détails…

```
        if self.config.doWriteCalibrate:
    # wait until after detection and measurement, since detection sets detected mask bits and both require
    # a background subtracted exposure;
    # note that this overwrites an existing calexp if doCalibrate false

    if calExposure is None:
        self.log.warn("calibrated exposure is None; cannot save it")
    else:
        if self.config.persistBackgroundModel:
            self.log.warn("Persisting background models as an image")
            bg = backgrounds[0].getImageF()
            for b in backgrounds[1:]:
                bg += b.getImageF()
            dataRef.put(bg, self.dataPrefix+"calexpBackground")
            del bg
        else:
            mi = calExposure.getMaskedImage()
            for bg in backgrounds:
                mi += bg.getImageF()
```

Fichiers résultats

```
            mi += bg.getImageF()
        dataRef.put(calExposure, self.dataPrefix + "calexp")

    if calib is not None:
        self.propagateCalibFlags(calib.sources, sources)

    if sources is not None and self.config.doWriteSources:
        if self.config.doWriteHeavyFootprintsInSources:
            sources.setWriteHeavyFootprints(True)
        dataRef.put(sources, self.dataPrefix + 'src')

    if self.config.doWriteSourceMatches:
        self.log.info("Matching src to reference catalogue" % (dataRef.dataId))
        srcMatches, srcMatchMeta = self.matchSources(calExposure, sources)

        normalizedSrcMatches = afwTable.packMatches(srcMatches)
        normalizedSrcMatches.table.setMetadata(srcMatchMeta)
        dataRef.put(normalizedSrcMatches, self.dataPrefix + "srcMatch")
    else:
        srcMatches = None; srcMatchMeta = None

    return pipeBase.Struct(
        inputExposure = inputExposure,
        exposure = calExposure,
        calib = calib,
        apCorr = apCorr,
        sources = sources,
        matches = srcMatches,
        matchMeta = srcMatchMeta,
        backgrounds = backgrounds,
    )
```

Conclusion

- *Encore beaucoup de choses à comprendre:*
  - *traitement effectif des images*
  - *application des diverses corrections et calibrations*
  - *mesures effectuées (formes, psf, ...)*
  - *définitions des sources*
  - *structure du code (python+C++, ...) -> présentation de Dominique*

- *demo.sh permet d'aboutir à un catalogue de sources -> quid des objets ?*