First Astro-Wise Course

INAF-OAC/VSTceN Napoli, 23-26 June 2008

GOAL: the aim of this course is to give an overview of the Astro-Wise (AW) system by showing an example of reduction of WFI images, from the raw frames to the final coadded mosaic, photometrically and astrometrically calibrated (WFI=Wide Field Imager at the 2.2m ESO telescope). The data will be reduced during the course. Each participant will have the possibility to reduce the same data set and might decide to use different algorithms and/or parameters. At the end of the course, the results may be compared.

1. Documentation: the Astro-WISE Information System

All the documentation that will be used in this course is available at the AW web site:
http://www.astro-wise.org/ → *AWE Information System.*

The core of the AWE Information System is given by the so-called "HOW-TOs" and the Manual.

The "HOW-TOs" are a powerful tool that gives the possibility to new users to start using the system, even without any specific knowledge of python or SQL. They are available at:
http://www.astro-wise.org/ → *AWE Information System* → *Howtos & Manual*
(or directly *http://www.astro-wise.org/portal/aw_howtos.shtml*). The example of data reduction in the following sections illustrates only a few general cases. Looking in the "HOW-TOs" you can find many more examples.

The User and Development Manual, at *http://www.astro-wise.org/docs/Manual.pdf* , contains all the HOW-TOs, a general overview of the system and of the web services (not completely updated), an overwiev of the pipeline and of the various reduction steps and finally a few sections dedicated to more advanced users and developers.

2. The data set

The data set that will be used in this course consists of 6 R-band scientific images (6 ditherings) on the field 3C273 plus the calibration frames: 10 BIAS, 3 dome FFs, 3 twilight FFs and 9 photometric standard fields. The field 3C273 has been selected because it is being used for ongoing tests and optimization of the pipeline.

All the raw files are available in the contaext "ALL" of the data base (DB), which is accessible to all participants. All the output files produced by each participant will be saved in his own "MYDB" context (called e.g. "AWRSILVOTTI"), which is password protected and visible only to that particular user. At the end of the tests each participant can move (or "migrate") the final results (basically the final coadded mosaic and the weight map) to the "ALL" context so that each user will be able to see them. It will be thus possible to compare the results.

Note that what is actually moved from one context to another is the metadata in the DB. The real data (fits file) remain in their current location in the data storage (DS). In other words the different contexts are defined only in the DB.

Note also that the results of any reduction step can be saved or not in the DB (and in the DS for the fits files) depending whether the parameter *commit* is set to 1 or 0 respectively (see next sections).

It is possible to reduce single CCDs or the whole CCD mosaic in parallel. In both cases statistical information, quality flags and other information is given in a log file visible on the screen and that is saved in the main directory (when requested).

With *task.execute()*, single CCDs are reduced indipendently. Raw files are automatically retrieved from the DS and kept on your local disk in beo2 (master). The results are always saved in the local disk, nomatter whether *commit* is set to 0 or 1. This way to reduce data is used mainly for tests.

With the so-called *dpu runner* (distributed processing), all CCDs are reduced in parallel using all the nodes and the CPUs of the cluster. In this case the raw files are copied in the different beo2 nodes and then removed automatically. The results are also automatically removed from the various beo2 nodes once the process is finished. This means that, if *commit* is set to 0, the results will not be saved at all. The only reason to run the *dpu runner* with *commit*=0 is to check whether everything is going fine, without error or warning messages, before committing. The *dpu runner* is the method normally used to reduce mosaic data.

3. The awe prompt: few python examples

To enter the Astro-Wise environment (AWe), just go in the proper directory of beo2 and type "awe":

```
> ssh -X beo2
> mkdir 3C273
> cd 3C273
> awe
```

you will receive a few lines of information ending with e.g.:

```
Current profile:
- username : AWRSILVOTTI
- database : odin
- project  : AWRSILVOTTI

> awe
```

The awe prompt keeps track of the previous commands: the *.awehistory* file is updated and saved in the main directory at the end of each awe session. Linux commands are accepted in this way:

```
awe> os.system('ls')
```

An help is available, e.g.:

```
awe> help(RegridTask)
```

In the **awe** interactive shell you can operate with numbers:

```
awe> (2-2*3+4**2)/2
6
awe> (2-2*3+4**2)/2.
6.0
```

or with strings:

```
awe> str="Hello World!"
awe> str[0]
'H'
awe> str[1]
'e'
awe> str[-1]
'!'
awe> str[-2]
'd'
awe> print str[0]
H
awe> print str
Hello World!
```

Or you can create a plot:

```
awe> x = range(10)
awe> y = [3*z**2 + 10 for z in x]
awe> pylab.plot(x,y)
awe> pylab.scatter(x,y)
```

Note that indentation is mandatory in python:

```
awe>  x = range(10)
  File "<stdin>", line 1
    x = range(10)
    ^
IndentationError: unexpected indent
```

The following command close your **awe** session:

```
awe> exit()
or CTRL-D
```

4. Rules of the game and hardware/network limitations

Due to our limited hardware resources, each user will have on beo2 a quota of about 5 GB. As it is easy to produce a lot of intermediate products when doing tests, you will be forced to remove periodically old files from your diskspace. Moreover, if you want to display an image, it is recommended, when possible, to launch ds9 or skycat from your laptop, without risking to saturate the RAM of the beo2 master. The total network band width from this room to beo2 is about 240 Mb/s, divided by the number of users (but each user has not more than ∼80 Mb/s available at maximum). To retrieve an 8k×8k image from beo2 to your laptop requires about 25s × # of users on that particular outlet. In the data storage there are no quotas and the total space available at the moment is around 1.5 TB. How to "clean" your "MYDB" context (and therefore remove the corresponding files in the DS), after having published the good files in the "ALL" context, is explained in section 9.

5. DB queries

There are two possibilities to see what is in the DB.

a) Use normal SQL-type queries.

Here is an example:

```
awe> q = (RawBiasFrame.instrument.name == 'WFI') & (RawBiasFrame.chip.name == 'ccd50')
awe> len(q)
10
awe> for b in list(q): print b.filename,b.imstat.median,b.imstat.stdev
...
WFI.2003-03-03T19:23:24.843_1.fits 183.0 28.1000311576
WFI.2003-03-03T18:02:46.015_1.fits 181.0 28.2747493223
WFI.2003-03-03T19:24:24.163_1.fits 183.0 28.1375143292
WFI.2003-03-03T19:20:11.815_1.fits 182.0 27.9773514139
WFI.2003-03-03T19:18:48.680_1.fits 181.0 27.9431557611
WFI.2003-03-03T19:25:31.810_1.fits 183.0 28.139588185
WFI.2003-03-03T18:59:58.474_1.fits 180.0 28.3047197324
WFI.2003-03-03T19:22:22.821_1.fits 183.0 28.0414561131
WFI.2003-03-03T19:21:15.723_1.fits 182.0 28.0809736488
WFI.2003-03-03T19:01:05.890_1.fits 180.0 28.1317630175
```

And a more sofisticated example, where we want to sort the files in increasing order of the seeing.

```
[select all the regridded frames]

awe> q = RegriddedFrame.filename != ''
awe> len(q)
awe> q.order_by('psf_radius')
awe> for f in q: print f.DATE_OBS,f.psf_radius
...
2003-03-04 05:07:47 0.931678438187
2003-03-04 05:07:47 0.931708669662
2003-03-04 05:07:47 0.934878349304
.......
2003-03-04 00:15:04 1.70493936539
```

b) Use the web based DBviewer

To enter the DB viewer, print "beo2" on your browser. Then select "AstroWise" and "DBView". Or simply go to:

```
http://beo2/?c=AstroWise&w=DBView
```

After that, you will have to select the context ("SET context"). There are two possibilities: "ALL" and "MYDB" (your own personal context e.g. "AWRSILVOTTI").

The DB viewer allows to see all the content of the DB, according with the user's privileges. For example the KIDS context will be accessible only to the KIDS members. As the AW DB is federated, all the KIDS members at the different AW nodes in Napoli, Groningen or Munich will be able to see and retrieve raw and reduced files through DBviewer. From each image, catalogue or even single entry in a catalogue it is possible to go back in its history. From a single source in a catalogue it is possible to build on the flight all the stamps of all the images that have been used to obtain that catalogue, back to the FF and BIAS raw frames.

6. Reducing single CCD data

Reducing single CCDs with *task.execute()* is a good starting point to gather experience with the system. Here is an example of reduction up to the regridding of the astrometrically and photometrically calibrated reduced frames.

```
# check which raw Bias are in the DB

awe> q = (RawBiasFrame.instrument.name == 'WFI') and (RawBiasFrame.chip.name == 'ccd50')
awe> len(q)
10
awe> for b in list(q): print b.filename,b.imstat.median,b.imstat.stdev
...
WFI.2003-03-03T19:23:24.843_1.fits 183.0 28.1000311576
WFI.2003-03-03T18:02:46.015_1.fits 181.0 28.2747493223
WFI.2003-03-03T19:24:24.163_1.fits 183.0 28.1375143292
WFI.2003-03-03T19:20:11.815_1.fits 182.0 27.9773514139
WFI.2003-03-03T19:18:48.680_1.fits 181.0 27.9431557611
WFI.2003-03-03T19:25:31.810_1.fits 183.0 28.139588185
WFI.2003-03-03T18:59:58.474_1.fits 180.0 28.3047197324
WFI.2003-03-03T19:22:22.821_1.fits 183.0 28.0414561131
WFI.2003-03-03T19:21:15.723_1.fits 182.0 28.0809736488
WFI.2003-03-03T19:01:05.890_1.fits 180.0 28.1317630175

# create a master BIAS, without saving it in the DB (but it is automatically
# saved on local disk)

awe> task=BiasTask(instrument='WFI', chip='ccd50', overscan=6, commit=0)
awe> task.execute()

# create a master Bias and save it in the DB (and DS)

awe> task=BiasTask(instrument='WFI', chip='ccd50', overscan=6, commit=1)
awe> task.execute()

# create and save a hot pixels mask from the master Bias

awe> task=HotPixelsTask(instrument='WFI', chip='ccd50', date='2003-03-03', commit=1)
awe> task.execute()

# check which raw Dome flats are in the DB. Note that the 'select' method
# automatically ignores invalid data with nonzero quality_flags

awe> q = RawDomeFlatFrame.select(instrument='WFI', chip='ccd50', filter='#844')
awe> len(q)
3
awe> for d in list(q): print d.filename,d.imstat.median,d.imstat.stdev
...
WFI.2003-03-03T19:46:58.069_1.fits 17945.0 1784.94231813
WFI.2003-03-03T19:48:27.232_1.fits 18192.0 1809.19862947
WFI.2003-03-03T19:45:17.277_1.fits 17217.0 1712.8807216

# create a master Dome Flat and save it in the DB

awe> task=DomeFlatTask(instrument='WFI', chip='ccd50', date='2003-03-03', \
```

```
     filter='#844', commit=1)
awe> task.execute()

# create and save a cold pixels mask from the master Dome Flat

task=ColdPixelsTask(instrument='WFI', chip='ccd50', date='2003-03-03',filter='#844',\
     commit=1)
awe> task.execute()

# DB query

awe> q = RawTwilightFlatFrame.select(instrument='WFI', chip='ccd50', filter='#844')
awe> len(q)
3
awe> for t in list(q): print t.filename,t.imstat.median,t.imstat.stdev
WFI.2003-03-03T23:48:12.051_1.fits 17919.0 1716.94695861
WFI.2003-03-03T23:50:42.573_1.fits 19911.0 2031.54084755
WFI.2003-03-03T23:46:15.975_1.fits 17270.0 1604.92565176

# create a master Twilight Flat and save it in the DB

awe> task=TwilightFlatTask(instrument='WFI', chip='ccd50', date='2003-03-03',\
     filter='#844', commit=1)
awe> task.execute()

# create a Master Flat (combination of Dome and Twilight) and save it in the DB

awe> task=MasterFlatTask(instrument='WFI', chip='ccd50', date='2003-03-03',\
     filter='#844', commit=1)
awe> task.execute()

# reduce a single science frame (standard field)

awe> task = ReduceTask(raw_filenames=['WFI.2003-03-04T04:42:29.496_1.fits'], commit=1)
awe> task.execute()

# calculate the astrometric solution and inspect the result
# a figure showing the quality of the astrometric solution will be produced

awe> task = AstrometricParametersTask(red_filenames=['Sci-RSILVOTTI-WFI-------#844\
...  -ccd50-Red---Sci-54637.9045189-f077a92d897a896ad15758c0b586205d5a1d20f4.fits'],\
...  inspect=1, commit=0)
awe> task.execute()

# if quality is ok, commit. Note that the file can be selected using either
# the reduced_filenames (as above) or the raw_filenames (below)

awe> task = AstrometricParametersTask(raw_filenames=['WFI.2003-03-04T04:42:29.496_1.fits'],\
...  inspect=1, commit=1)
awe> task.execute()

# normally, at this point, an Illumination Correction Map (ICM) is created
# from the reduced standard field

# if an ICM is already existing in any accessible context (in our case it is in
```

```
# "ALL"), the ICM is automatically used when "ReduceTask" is launched.
# It is therefore important to check whether an ICM is already present !

# To look at the existing ICM:

awe> red  = (ReducedScienceFrame.filename == 'Sci-RSILVOTTI-WFI-------#844-ccd50-Red---\
Sci-54629.9887099-60c5de313b00e2dcce6638ca6f27f5a33bf30b46.fits' )[0]

awe> red.illumination.illuminationcorrection.inspect()

# In our case the reduced standard field produced is already corrected for IC
# and therefore it is ready to calculate the photometric solution

# photometric solution: create a catalog of standard stars

awe> task=PhotcatTask(instrument='WFI', raw_filenames=['WFI.2003-03-04T04:42:29.496_1.fits'],\
...  inspect=1)
awe> task.execute()
awe> task=PhotcatTask(instrument='WFI', raw_filenames=['WFI.2003-03-04T04:42:29.496_1.fits'],\
...  commit=1)
awe> task.execute()

# photometric solution: calculate zero point

awe> task=PhotomTask(instrument='WFI',raw_filenames=['WFI.2003-03-04T04:42:29.496_1.fits'],\
...  inspect=1, commit=0)
awe> task.execute()
awe> task=PhotomTask(instrument='WFI',raw_filenames=['WFI.2003-03-04T04:42:29.496_1.fits'],\
...  commit=1)
awe> task.execute()

# reduce the 6 scientific frames

awe> task = ReduceTask(instrument='WFI',chip='ccd50',object='3C*',commit=1)
awe> task.execute()

# calculate the astrometric solution and inspect the results

awe> task = AstrometricParametersTask(instrument='WFI',chip='ccd50',object='3C*',\
...  inspect=1,commit=0)
awe> task.execute()
awe> task = AstrometricParametersTask(instrument='WFI',chip='ccd50',object='3C*',commit=1)
awe> task.execute()

# global astrometry: create a list of sources

awe> help(GAstromSourceListTask)
awe> task = GAstromSourceListTask(instrument='WFI',chip='ccd50',object='3C*',commit=1)
awe> task.execute()

# global astrometry: calculate improved solution

awe> help(GAstromTask)
awe> task = GAstromTask(instrument='WFI',chip='ccd50',object='3C*',inspect=1,commit=0)
awe> task.execute()
```

```
awe> task = GAstromTask(instrument='WFI',chip='ccd50',object='3C*',inspect=1,commit=1)
awe> task.execute()

# regridding after having specified that we want to keep the original WFI pixel scale.
# If not specified, the pixel scale is set to 0.2 by default !!
# BACKGROUND_SUBTRACTION_TYPE = 1 means that the sky is subtracted
# (default is 0, i.e. non-subtracted)
# The use of "pars" is described in section 8.

awe> pars=Pars(RegridTask)
awe> pars.RegriddedFrame.process_params.BACKGROUND_SUBTRACTION_TYPE = 1
awe> pars.RegriddedFrame.swarpconf.PIXEL_SCALE=0.238

awe> task = RegridTask(instrument='WFI',chip='ccd50',object='3C*',inspect=1,commit=0)
awe> task.execute()
```

7. Reducing mosaics in parallel (DPU runner)

When using the DPU runner for distributed processing, it is important to verify the status of the jobs. This can be seen through a web interface, the Astro-Wise RemoteProcessingUnit at

```
http://beo2/?c=AstroWise&w=DPU
```

Here is an example of reduction in parallel mode.

```
# in order to see the various tasks:
awe> dpu.run('aaaaaa',instrument='WFI',date='2003-03-03')

# create a master Bias, without saving it in the DB and produce a log file

awe> dpu.run('Bias',instrument='WFI',date='2003-03-03', commit=0); dpu.get_logs()

# create a master Bias and save it in the DB.
# The "dpu.wait_for_jobs()" command means that the next job will have to wait
# until this job is completely finished. It should always be used when the next
# job uses the results of the current job.

awe> dpu.run('Bias',instrument='WFI',date='2003-03-03', commit=1); dpu.wait_for_jobs()

# create a Hot Pixels Map

awe> dpu.run('HotPixels',instrument='WFI',date='2003-03-03', commit=0)

# create a Dome Flat Field

awe> dpu.run('DomeFlat',instrument='WFI',date='2003-03-03',filter='#844',commit=0);\
...  dpu.wait_for_jobs()

# create a Cold Pixels Map

awe> dpu.run('ColdPixels',instrument='WFI',date='2003-03-03',filter='#844',commit=0)

# create a Twilight Flat Field
```

```
awe> dpu.run('TwilightFlat',instrument='WFI',date='2003-03-03',filter='#844',commit=0);\
...  dpu.wait_for_jobs()

# create a Master Flat Field (combination of Dome and Twilight)

awe> dpu.run('MasterFlat',instrument='WFI',date='2003-03-03',filter='#844', \
...  commit=1); dpu.wait_for_jobs(); dpu.get_logs()

# reduce one raw science frame (standard field)

awe> dpu.run('Reduce',i='WFI',raw_filenames=['WFI.2003-03-04T04:42:29.496_1.fits',\
... 'WFI.2003-03-04T04:42:29.496_2.fits','WFI.2003-03-04T04:42:29.496_3.fits',\
... 'WFI.2003-03-04T04:42:29.496_4.fits','WFI.2003-03-04T04:42:29.496_5.fits',\
... 'WFI.2003-03-04T04:42:29.496_6.fits','WFI.2003-03-04T04:42:29.496_7.fits',\
... 'WFI.2003-03-04T04:42:29.496_8.fits'], commit=0)

# or (note that "for i in range(1,9)" means i=1,2,3,4,5,6,7,8 !!)

awe> dpu.run('Reduce',i='WFI',raw_filenames=\
...  ['WFI.2003-03-04T04:42:29.496_'+str(i)+'.fits' for i in range(1,9)], commit=0)

# calculate astrometric solution (each CCD independently)

awe> dpu.run('Astrometry', ,i='WFI',raw_filenames=\
...  ['WFI.2003-03-04T04:42:29.496_'+str(i)+'.fits' for i in range(1,9)] \
...  commit=1)
```

Photometric solution: even though there is the possibility to use a "dpu.run('Photom')" recipe (see HOW-TO Image Pipeline → Overview), we suggest to use "task=PhotcatTask" and "task=PhotomTask" to calculate the independent zero points in each CCD.

```
# reduce all the science frames (scientific fields)

awe> dpu.run('Reduce', instrument='WFI', filter='#844',object='3C273', \
...  pars=pars.get(), commit=1)

# calculate astrometric solution (each CCD independently)

awe> dpu.run('Astrometry', instrument='WFI', object='3C273', commit=1)

awe> dpu.get_logs()

# global astrometry

awe> dpu.run('GAstromSL',instrument='WFI', object='3C273', commit=0);\
...  dpu.wait_for_jobs()
awe> dpu.run('GAstrom',instrument='WFI', filter='#844', object='3C273',\
...  commit=0); dpu.wait_for_jobs()

# apply regridding after having changed parameters (see next section)

awe> pars=Pars(RegridTask)
awe> pars.RegriddedFrame.process_params.BACKGROUND_SUBTRACTION_TYPE = 1
awe> pars.RegriddedFrame.swarpconf.PIXEL_SCALE=0.238
```

```
awe> dpu.run('Regrid', instrument='WFI', filter='#844',object='3C273', \
...  pars=pars.get(), commit=1)

awe> dpu.wait_for_jobs()

# coaddition: creation of the coadded mosaic + weight map

awe> dpu.run('Coadd', instrument='WFI', filter='#844',object='3C273',commit=1)
```

There is also the possibility to run the pipeline in a more automatic way, using pre-defined sequences of tasks (see HOW-TO Image Pipeline → Overview). This aspect might be discussed during the course.

8. Changing parameters: the "pars" class

It is possible to change the value of several parameters without changing the python (or C/C++) code using the "pars" command.

```
# to see the parameters of "BiasTask"

awe> pars=Pars(BiasTask)
awe> pars.show()
BiasFrame
 |
 +--process_params
 |   |
 |   +--MAXIMUM_ABS_MEAN: 10.0
 |   +--MAXIMUM_STDEV: 10.0
 |   +--MAXIMUM_STDEV_DIFFERENCE: 10.0
 |   +--MAXIMUM_SUBWIN_FLATNESS: 100000.0
 |   +--MAXIMUM_SUBWIN_STDEV: 100000.0
 |   +--OVERSCAN_CORRECTION: 6
 |   +--SIGMA_CLIP: 3.0
```

Here is an example of changing parameters for the satellite tracks removal:

```
awe> pars=Pars(SatelliteMap)
awe> pars.show()
SatelliteMap
 |
 +--process_params
 |   |
 |   +--DETECTION_THRESHOLD: 5.0
 |   +--HOUGH_THRESHOLD: 1000.0

awe> pars.SatelliteMap.process_params.DETECTION_THRESHOLD=3.0
awe> pars.SatelliteMap.process_params.HOUGH_THRESHOLD = 900.0
awe> pars.show()
SatelliteMap
 |
 +--process_params
 |   |
```

```
 |   +--DETECTION_THRESHOLD: 3.0
 |   +--HOUGH_THRESHOLD: 900.0

awe> pars.get()    ????????????????
```

Then the reduction can be done again with the new parameters:

```
awe> dpu.run('Reduce', instrument='WFI', filter='#844',object='3C273',pars=pars.get(),\
...  commit=1)
```

9. Publishing the good results and cleaning MYDB

When good results are obtained, these can be published in the context "ALL", (they "migrate" from "MYDB" to "ALL") where all the other users can see them:

```
awe> c = (CoaddedRegriddedFrame.OBJECT == '3C273').max('creation_date')
awe> c.filename
'Sci-RSILVOTTI-WFI-------#844---Coadd---Sci-54635.5621107-ab94774c1fe979e98e861dda35ee25
e33c791c83.fits'

awe> context.migrate(c, 'ALL')

# check that the file is now in "ALL"

awe> context.set_project('ALL')
awe> context.get_current_project().name
'ALL'

awe> c = (CoaddedRegriddedFrame.OBJECT == '3C273')
awe> for a in list(c): print a.filename
...
Sci-RSILVOTTI-WFI-------#844---Coadd---Sci-54636.6664862-e5d69680d28fa5a9a8639194f3963
ccb054fafe1.fits

# return to "MYDB"

awe> context.set_project('AWRSILVOTTI')
awe> context.get_current_project().name
'AWRSILVOTTI'
```

Note that also all the objects on which this file depends do migrate automatically, in order to keep all the dependencies.

It is also possible to "clean" MYDB, removing single objects or even the whole data (including associated objects on data server). This aspect might be discussed during the course.

10. Extracting and "associating" catalogs (source lists)

```
# select the latest version of the final coadded mosaic

awe> c = (CoaddedRegriddedFrame.OBJECT == '3C273' ).max('creation_date')
```

```
awe> c.filename
'Sci-RSILVOTTI-WFI-------#844---Coadd---Sci-54636.6664862-e5d69680d28fa5a9a8639194f3963
ccb054fafe1.fits'

# create a source list (SL). The following two commands are equivalent:

awe> task = SourceListTask( filenames=[c.filename], commit=1)
awe> task = SourceListTask(filenames=['Sci-RSILVOTTI-WFI-------#844---Coadd---\
Sci-54636.6664862-e5d69680d28fa5a9a8639194f3963ccb054fafe1.fits'], commit=1)

awe> task.execute()

# Another example: create a set of SLs from the 6 Regridded Frames of ccd55

awe> q = (RegriddedFrame.OBJECT == '3C273' )  &  (RegriddedFrame.chip.name == 'ccd55')
awe> filenames_list = [reg.filename for reg in list(q)]
awe> task = SourceListTask( filenames= filenames_list , commit=1)
awe> task.execute()

# "associate" (i.e. find the overlapping sources of) 2 SLs extracted before
# ("SLID" is the Source List ID that identifies each particular source list)

awe> q = (SourceList.OBJECT == '3C273') & ( SourceList.chip.name == 'ccd55')
awe> sl_list = list(q)
awe> print sl_list[0].SLID, sl_list[1].SLID
1530 1790
awe> task = AssociateListTask(ids=[( sl_list[0].SLID  , 's'), ( sl_list[1].SLID, 's')],\
...  commit=1 )
awe> task.execute()
```

In order to look at the source lists just created and "associated", you have to select the class "source lists" in the DB viewer.