Advanced ways to find and retrieve data in the LTA

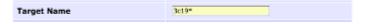
There are some useful ways to find and retrieve your data in the LTA that might not be immediately obvious. This page explains some of the more advanced options you have.

Queries

 You can use colons in numeric queries, to select ranges. This will for example give all observations and pipelines that have a SAS/Observation ID in the range from 432000 to 432190:



In textual entries, wildcards can be used.

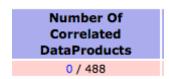


• You can put a list of SAS/Observation IDs in the query:

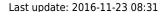


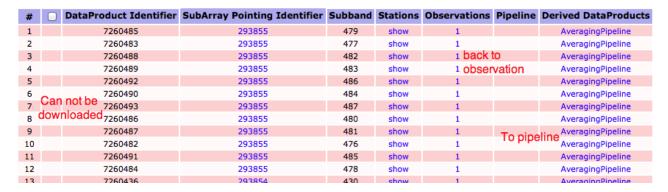
Viewing data

When you are looking at the results of a query you might see something like this:



This means that the observation is known in the LTA, it knows what data was produced, the produced data was not archived, but further processing happened on the raw data and the results of some of those pipelines were archived. If you click on the zero, you will see something like this:





This allows you to navigate from a pipeline back to the original observation, or from the observation to any pipelines that have run on the raw data.

Retrieving data

 You can retrieve data on the Observation and Pipeline level, you don't have to select all files individually.

		Observation	Observing	Antenna Set	Instrun
#		Id	Mode		Filte
1	⋖	146448	Interferometer	HBA Dual Inner	110-190
2		146447	Interferometer	HBA Dual Inner	110-190
3	⋖	146446	Interferometer	HBA Dual Inner	110-190
4		146445	Interferometer	HBA Dual Inner	110-190
5	⋖	146444	Interferometer	HBA Dual Inner	110-190
6	\checkmark	146443	Interferometer	HBA Dual Inner	110-190
7		146442	Interferometer	HBA Dual Inner	110-190
8	\checkmark	146441	Interferometer	HBA Dual Inner	110-190
9	⋖	146456	Interferometer	HBA Dual Inner	110-190
10	\checkmark	146455	Interferometer	HBA Dual Inner	110-190
11		146454	Interferometer	HBA Dual Inner	110-190
12		146453	Interferometer	HBA Dual Inner	110-190
13		146452	Interferometer	HBA Dual Inner	110-190
	_				

• If you have a query with more than 1000 results, you can open the multiple pages each in a separate tab/window.

Observation 1001 to 1100 (showing 100 of total 1156) -

With the small triangle next to a list, you can fold or unfold the list to get a better overview.

Folded entries

Observation 1 to 100 (showing 100 of total 1156)

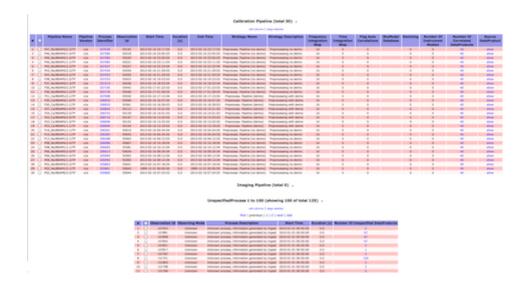
Averaging Pipeline 1 to 100 (showing 100 of total 4060)

Calibration Pipeline (total 30)

Imaging Pipeline (total 0)

UnspecifiedProcess 1 to 100 (showing 100 of total 125)

Unfolded entries



DBView

There is a server that gives the option to run your own queries on the database http://lofar-dbview.target.rug.nl/

A useful query might be this one, that gives you all files for a certain Obs Id (SAS VIC tree ID).

```
SELECT fo.URI, dp."dataProductType", dp."dataProductIdentifier",
    dp."processIdentifier"
FROM AWOPER."DataProduct+" dp,
        AWOPER.FileObject fo,
        AWOPER."Process+" pr
WHERE dp."processIdentifier" = pr."processIdentifier"
    AND pr."observationId" = '123456'
    AND fo.data_object = dp."object_id"
```

```
AND dp."isValid" > 0
```

In this '123456' should be replaced with the Obs Id of an Observation/Pipeline you're looking for. Pipelines also have an "observationId" == the SAS Id, even though that's a but confusing. To be able to run this query, you have to go to the link above, login as the right user, select the right project, and then put this query into the "Manual SQL".

Example You can also modify these queries. for example if you want to also know the MD5 checksum, you can run:

```
SELECT fo.URI, fo.hash_md5, dp."dataProductType",
dp."dataProductIdentifier",
dp."processIdentifier"
FROM AWOPER."DataProduct+" dp,
         AWOPER.FileObject fo,
         AWOPER."Process+" pr
WHERE dp."processIdentifier" = pr."processIdentifier"
   AND pr."observationId" = '123456'
AND fo.data_object = dp."object_id"
AND dp."isValid" > 0
```

AstroWise Python Interface

There is also a python interface to the LTA. With this, you can script some advanced queries. To have this working, you first need to install the LTA client in your machine. Be sure to have the latest version installed.

Once you have installed the client, set up your user name and password. These are the same as for MoM. Remember that this is just a different interface to the LTA catalogue: you will need the same credentials as for the web interface.

After installing the LTA client, the file .awe/Environment.cfg will appear in your home directory. Add the following lines to the file

```
database_user : <your username>
database_password : <your password>
```

then create the environment variable AWETARGET and set it to awlofar, and AWEVERSION to AWBASE. In a bash shell, you can do so by adding the following to your .profile file:

```
export AWETARGET=awlofar export AWEVERSION=AWBASE
```

Finally, your hostname may cause an error, if it does not contain a full domain. In this case, you will need to edit your /etc/hosts file (you will need root/superuser privileges to do so). In that file, you should find a line that looks like this

```
127.0.0.1 localhost
```

Change that line into

```
127.0.0.1 localhost <your_host_name>
```

If you do not know your hostname, just type hostname in a shell and you will get it as an output.

Now, you can use the following script as a test. It may still give you some warnings, but if it prints out a list of pointings, then you are ready to go. You may need to kill the script, because it will print out all the observations in a certain patch of the sky archived in the LTA.

```
# python code
from pprint import pprint
from awlofar.main.aweimports import Observation, Pointing, SubArrayPointing
from common.database.Context import context
result = {}
for project in sorted(context.get_projects()) :
    print "Project %(project)s" % vars()
    ok = context.set project(project)
    # do your query
    obs ids = set()
    query = (Pointing.rightAscension > 95)
            (Pointing.rightAscension < 105) & \
            (Pointing.declination
                                   > 20) & \
            (Pointing.declination
                                     < 30)
    print "Total Pointings %d" % len(query)
    for pointing in query :
        print "Pointing found RA %f DEC %f" % (pointing.rightAscension,
pointing.declination)
        query_subarr = SubArrayPointing.pointing == pointing
        for subarr in query subarr:
            query_obs = Observation.subArrayPointings.contains(subarr)
            for obs in query obs :
                obs ids.add(obs.observationId)
    result[project] = sorted(list(obs ids))
    print result[project]
pprint(result)
```

If you get errors and do not manage to view the list of pointings, there may be the need to open some port on the firewall at your institution. Specifically, port 1521 should be open. In case of trouble, get in contact with Science Support.

Examples

Once you have tested that your connection to the catalogue is working, you are ready to browse the archive and stage the data you need. Here we will list a few examples of python scripts that can be used to access the LTA. All of them will need to import some modules:

```
from datetime import datetime
```

```
from awlofar.database.Context import context
from awlofar.main.aweimports import CorrelatedDataProduct, \
   FileObject, \
   Observation
from awlofar.toolbox.LtaStager import LtaStager, LtaStagerError
```

The lines above must be added to each of the scripts below for these to work.

This simple script will allow you to find all data within a single project, for example LC2_035. Please change the project name to the code of a project of yours. If you also want to stage the data you found, just set the do_stage variable to True. Be careful with how many files you stage and what size they have: the same limits as for the web interface apply here.

```
# Should the found files be staged ?
do stage = False
# The project to guery, LC2 035 has public data
project = 'LC2 035'
# The class of data to query
cls = CorrelatedDataProduct
# Query for private data of the project, you must be member of the project
private data = False
# To see private data of this project, you must be member of this project
if private data :
    context.set project(project)
    if project != context.get_current_project().name:
        raise Exception("You are not member of project %s" % project)
query observations = Observation.select all().project only(project)
uris = set() # All URIS to stage
for observation in query observations :
    print("Querying ObservationID %s" % observation.observationId)
    # Instead of querying on the Observations of the DataProduct, all
DataProducts could have been queried
    dataproduct query = cls.observations.contains(observation)
    # isValid = 1 means there should be an associated URI
    dataproduct query &= cls.isValid == 1
    for dataproduct in dataproduct query :
        # This DataProduct should have an associated URL
        fileobject = ((FileObject.data object == dataproduct) &
(FileObject.isValid > 0)).max('creation date')
        if fileobject :
            print("URI found %s" % fileobject.URI)
            uris.add(fileobject.URI)
        else :
            print("No URI found for %s with dataProductIdentifier %d" %
(dataproduct.__class__.__name__, dataproduct.dataProductIdentifier))
print("Total URI's found %d" % len(uris))
if do_stage :
```

```
stager = LtaStager()
stager.stage_uris(uris)
```

The following script will find subbands 301 and 302 for all targets within two different projects.

Pay attention to the difference between the keys subband and stationSubband; the former is a sequential number assigned to each subband in an observation, while the latter is linked to the frequency at which the observation was performed. Example: an observation was set up covering the range 30-77.3 MHz with two simultaneous beams using 244 subbands each. In this case, subband will range from 0 to 487, while stationSubband from 153 to 396. The stationSubband information is stored in the observation, but not in the pipeline products (which instead contain the frequency). If you want to search on stationSubband, you must perform your search on observations first, then fetch the pipelines linked to those observations. If you use frequency, you can search directly on pipelines.

As a general advise, before performing a search, you need to **understand thoroughly the meaning of the keywords that you are using and where their values are stored**, otherwise you may not find the data you are looking for.

```
do stage = False
project1 = 'LC2 016'
project2 = 'LC2 012'
subband1 = 301
subband2 = 302
cls = CorrelatedDataProduct
# Query for private data of the project, you must be member of the project
private data = False
# All URIS to stage
uris = {
   project1: set(),
   project2: set(),
}
for project in (project1, project2) :
   print("Using project %s" % project)
    if private_data :
        context.set project(project)
        if project != context.get current project().name:
            raise Exception("You are not member of project %s" % project)
    query observations = Observation.select all().project only(project)
    for observation in query observations :
        print("Querying ObservationID %s" % observation.observationId)
        dataproduct query = cls.observations.contains(observation)
        # isValid = 1 means there should be an associated URI
        dataproduct query &= cls.isValid == 1
        dataproduct query &= ((cls.subband == subband1) | (cls.subband ==
subband2))
        # Or for stationSubband do :
        #dataproduct guery &= ((cls.stationSubband == subband1) |
(cls.stationSubband == subband2))
        for dataproduct in dataproduct_query :
```

```
# This DataProduct should have an associated URL
            fileobject = ((FileObject.data object == dataproduct) &
(FileObject.isValid > 0)).max('creation_date')
            if fileobject :
                print("URI found %s" % fileobject.URI)
                uris[project].add(fileobject.URI)
            else :
                print("No URI found for %s with dataProductIdentifier %d" %
(dataproduct.__class__.__name__, dataproduct.dataProductIdentifier))
for project in (project1, project2) :
    print("Total URI's found for project %s: %d" % (project,
len(uris[project])))
stager = LtaStager()
if do stage :
   for project in (project1, project2) :
        stager.stage uris(uris[project])
```

Here, we find data between freq1 and freq2 taken within one project between day1 and day2

```
do stage = False
project = 'LC2_033'
freg1 = 172.0
freq2 = 178.0
day1 = datetime(2014,8,26) # this could include time; ie hours, minutes,
secondes
day2 = datetime(2014, 8, 29) # idem
# DataProduct class to query; CorrelatedDataProduct, SkyImageDataProduct,
etc ...
cls = CorrelatedDataProduct
# Query for private data of the project, you must be member of the project
private data = False
# To see private data of this project, you must be member of this project
if private data :
    context.set project(project)
    if project != context.get_current_project().name:
        raise Exception("You are not member of project %s" % project)
query observations = (
    (Observation.startTime >= day1) &
    (Observation.endTime < day2) ).project only(project)</pre>
uris = set()
for observation in query observations :
    print("Querying ObservationID %s" % observation.observationId)
    dataproduct guery = cls.observations.contains(observation)
   # isValid = 1 means there should be an associated URI
   dataproduct query &= cls.isValid == 1
    dataproduct query &= cls.minimumFrequency >= freq1
```

```
dataproduct_query &= cls.maximumFrequency < freq2
for dataproduct in dataproduct_query :
    # This DataProduct should have an associated URL
    fileobject = ((FileObject.data_object == dataproduct) &

(FileObject.isValid > 0)).max('creation_date')
    if fileobject :
        print("URI found %s" % fileobject.URI)
        uris.add(fileobject.URI)
    else :
        print("No URI found for %s with dataProductIdentifier %d" %

(dataproduct.__class__.__name__, dataproduct.dataProductIdentifier))

print("Total URI's found %d" % len(uris))

if do_stage :
    stager = LtaStager()
    stager.stage_uris(uris)
```

Python Module for Staging

The python interaction with the LTA catalog can be complemented with the use of specific module developed to allow users more control over their staging requests. The module is made available * here and its functions are self-explanatory.

For a description of what the user can do, we list here the functions that are available.

stage(surls) It takes in a list of surls, queue a staging request for those urls, and outputs the ID of the request.

get status(stageid) It tells the user if a request is queued, in progress or finished (success).

abort(stageid) It allows users to end a staging request.

get_surls_online(stageid) It gives a list of the surls that have been staged for the relative request. The list is updated whenever a new surl comes on line.

get_srm_token(stageid) The srm token is useful to interact directly with the SRM site through GRID/SRM tools.

reschedule(stageid) If a request failed, it can be rescheduled.

get progress() No input needed. It returns the statuses of all the requests owned by the user.

From:

https://www.astron.nl/lofarwiki/ - LOFAR Wiki

Permanent link:

https://www.astron.nl/lofarwiki/doku.php?id=public:lta_tricks&rev=1479889907

Last update: 2016-11-23 08:31

