blast2caom

Development Log

Version 1.0
2008-02-14 – current

Russell Redman
National Research Council of Canada
Herzberg Institute of Astrophysics
Canadian Astronomy Data Centre

Table of Contents

1 Development Log

The purpose of this document is to record the development of the blast2caom program, including all significant events, commands, comments, and decisions. major sections will be labeled by the date, with subsections for important topics dealt with on that date.

Items in the "BLAST Archive ICD" will be referenced with notation like (ICD 1.1) for section 1, subsection 1.

2 2008-02-14

2.1 Initial Creation

The program is to be maintained in SVN. It has been cloned from macho2caom.

```
dalmore:~...blast2caom> svn export ${CADCSVN}/macho2caom/trunk
dalmore:~...blast2caom> mv trunk init
```

Rename individual files to refer to BLAST rather than MACHO.

```
dalmore:~...blast2caom> cd init/src
dalmore:~...init/src> mv macho2caom.py blast2caom.py
dalmore:~...init/src> mv macho_split2mef.py blast_split2mef.py
```

Import into blast2caom/trunk (previously created by Sharon Goliath).

```
dalmore:~...init/src> cd ../..
dalmore:~/cadcsvn/blast2caom> svn checkout ${CADCSVN}/blast2caom/trunk
A    trunk/sql
A    trunk/scripts
A    trunk/src
A    trunk/buildstart
Checked out revision 1.
```

Copy in initial files:

```
dalmore:~/cadcsvn/blast2caom> cd trunk/scripts
dalmore:~...trunk/scripts> cp ../../init/scripts/* .
dalmore:~...trunk/scripts> ls
./  ../  jython*  .svn/
dalmore:~...trunk/scripts> svn add jython
A         jython
dalmore:~...trunk/scripts> cd ../src
dalmore:~...trunk/src> cp ../../init/src/* .
dalmore:~...trunk/src> ls
./  ../  blast2caom.py  blast_split2mef.py*  ingest.py*  master.py*  .svn/
dalmore:~...trunk/src> svn add *.py
A         blast2caom.py
A         blast_split2mef.py
A         ingest.py
A         master.py
dalmore:~...trunk/src> cd ../sql
dalmore:~...trunk/sql> cp ../../init/sql/* .
dalmore:~...trunk/sql> ls
```

```
./  ../  obsExport_artifact.tmpl  obsExport_obs.tmpl  .svn/
dalmore:~...trunk/sql> svn add *.tmpl
A         obsExport_artifact.tmpl
A         obsExport_obs.tmpl
dalmore:~...trunk/sql> cd ..
```

Commit to SVN:

```
dalmore:~...blast2caom/trunk> svn commit -m "initial commit of blast2caom cloned
from macho2caom"
Adding         scripts/jython
Adding         sql/obsExport_artifact.tmpl
Adding         sql/obsExport_obs.tmpl
Adding         src/blast2caom.py
Adding         src/blast_split2mef.py
Adding         src/ingest.py
Adding         src/master.py
Transmitting file data .......
Committed revision 2.
```

3 2008-02-28

3.1 Edit blast2caom.py – default values

Change all occurrences of MACO to BLAST.
Change all occurrences of macho to blast.

The database is called “blast”.

```
# Observation
obs_collection = 'BLAST'
obs_project = 'BLAST'
```

(ICD 2.4)

```
# Telescope
telescope_name = 'BLAST'
# BLAST is a baloon-borne telescope that moved a considerable distance
# during each observation
telescope_obsgeo_x = 0.0
telescope_obsgeo_y = 0.0
telescope_obsgeo_z = 0.0
```

(ICD 2.2)

```
# Instrument
instrument_name = 'BLAST'
```

(ICD 2.3)

```
# Plane
plane_project = 'BLAST'
```

(ICD 3.6)

```
# Observable - fill CUNIT from FITS BUNIT
observable_ctype = 'phot.flux.density.sb'
```

```
observable_cunit = 'TBD'
```

(ICD 3.1 – still need to code BUNIT)

```
# SpatialWCS
spatialAxis1 = 1
spatialAxis2 = 2
spatial_cunit1 = 'deg'
spatial_cunit2 = 'deg'
```

(Note that macho2caom codes the units as ‘degrees’, but the FITS standard, section 5.4, specifies the units as ‘deg’.)

```
# SpectralWCS
spectral_specsys = 'TOPOCENT'
spectral_ssysobs = 'TOPOCENT'
spectral_ctype = 'WAVE'
spectral_cunit = 'um'
spectral_crder = None
spectral_csyer = None
spectral_restfrq = None
spectral_restwav = None
spectral_velosys = None
spectral_zsource = None
spectral_ssyssrc = 'TOPOCENT'
spectral_velang = None
spectral_naxis = 1
spectral_crpix = 1
```

This is mostly a copy of the macho SpectralWCS defaults, except spectral_cunit is ‘um’ because the measured filter profiles use wavelengths in microns.
(ICD 4.1)

```
# TemporalWCS
temporal_ctype = 'MJD'
temporal_cunit = 'd'
temporal_crder = None
temporal_csyer = None
temporal_naxis = 1
temporal_crpix = 0.5
```

The TemporalWCS will be a single range in time during which the observations occurred. Setting crpix=0.5 means that crval is the starting time.
(ICD 4.3)

3.2 Edit blast2caom.py – createSimpleObservation

```
    """
    Create and return a Java CAOM SimpleObservation class.
    """
    observation = SimpleObservation()
    observation.collection = obs_collection
    observation.collectionID = getStringKeyword(header, 'OBSID')
    observation.project = obs_project
```

Note that BLAST will fill collectionID from the OBSID header (ICD 2.4). The other fields take their default values.

```
    observation.telescope = Telescope()
    observation.telescope.name = telescope_name
    observation.telescope.geoLocationX = telescope_obsgeo_x
    observation.telescope.geoLocationY = telescope_obsgeo_y
    observation.telescope.geoLocationZ = telescope_obsgeo_z
```

The Telescope fields all take default values.

```
    observation.instrument = Instrument()
    observation.instrument.name = instrument_name
```

The instrument fields take default values.

```
    observation.target = Target()
    observation.target.name = getStringKeyword(header, 'OBJECT')
    observation.target.classification = target_classification
    observation.target.redshift = None
```

The target.name will be read from the header OBJECT and the redshift left as NULL = None (ICD 2.1). I still need to recode the target_classification.

4 2008-02-29

4.1 Edit blast2caom.py – add command line option for classification

Modify usage

```
def usage():
    print 'usage: jython blast2caom.py --help --user --password --fileId --number -
-mapping --test'
    print ''
    print '    --help                       Print this procedure.'
    print '    --user                       Sybase username.'
    print '    --password                   Sybase password.'
    print '    --fileId                     Process only this fileId.'
    print '    --number                     Process number of files.'
    print '    --mapping                    Write FITS to CAOM mapping to std out.'
    print '    --test                       Process the FITS file,'
    print '    --tmpDir                     Temporary directory to write FITS
files,'
    print '                                 default is current directory.'
    print '                                 print the FITS to CAOM mapping,'
    print '                                 but do not presist to the database.'
    print '    --class                      Target classification'
```

In main program

```
# First get the command line arguments
try:
    opts, args = getopt.getopt(sys.argv[1:], '', \
                 ['help', 'mapping', 'test', 'user=', \
             'password=', 'fileId=', 'number=', 'tmpDir=', \
             'class='])
```

Check for a sensible value

```
if classification is None:
    classification = target_classification
```

Sincce createSimpleObservation is called by
    getObservationPlane, which is called by
    ingest, which is called by main,
we need to pass classification as an argument to these three subroutines.
In the block of defaults

```
# Target - fill this from a command-line argument
target_classification = 'field'
```

In main

```
# Process the FITS headers
        ingest(observationORM, HDUs, adURI, mapping, test, classification)
```

In ingest

```
def ingest(observationORM, HDUs, adURI, mapping, test, classification):
. . .
    # If we don't have an Observation, either get an existing Observation
    # or create a new one.
    if observation is None:
        observation, plane = getObservationPlane(observationORM, header, \
                                                  classification)
```

In getObservationPlane

```
def getObservationPlane(observationORM, header, classification):

    # Construct a Java CAOM SimpleObservation
    observation = createSimpleObservation(header, classification)
```

5 2008-03-04

5.1 Discuss structure of macho2caom with Jeff Burke

Since the structure of the blast2caom program is modeled after macho2caom, it is important to understand macho.

There are a couple of steps in the macho ingestion process that BLAST will not need. The “raw” files supplied by MACHO had to be restructured into MEFs before final ingestion. There are three scripts that manage this process:

- blast_split2mef.py – module containing code to restructure the files
- master.py – fetch files from AD, split into MEF using blast_split2mef.py, store back into AD
- ingest.py – fetch restructured file from AD and ingest metadata into CAOM

The scripts master.py and ingest.py record the progress of the restructuring and ingestion in the database table macho_mefs. The macho scripts start with the “raw” data files already present in AD, and do not cover the process that put them there.

BLAST files do not need to be restructured, so blast_split2mef.py and master.py are superfluous and will be dropped from the BLAST SVN repository. It may be possible to merge the blast2caom.py and ingest.py scripts, but for the sake of maintaining similar code in the different archive it might be better to leave the scripts as they are, customizing only the bits that are actually different. Especially, the ingestion state recorded in macho_mefs can be simplified considerably.

6 2008-03-11

6.1 Edit blast2caom.py - createPlane

The project is always BLAST, set in the defaults.

The release date can be set for each file using the RELEASE keyword with the format YYYY-MM-DD, and has he default value of the time of ingestion (i.e., make public immediately).

```
release_date_str = getStringKeyword(header, 'RELEASE')
    calendar = Calendar.getInstance()
    if release_date_str:
        calendar.clear()
        calendar.set(int(release_date_str[0:3]), \
                     int(release_date_str[5:6]), \
                     int(release_date_str[8:9]))
    plane.metaRelease = calendar.getTime()
    plane.dataRelease = calendar.getTime()
```

BLAST does not provide a keyword SCALE, so the block of code from MACHO that checks for this keyword has been deleted. Similarly, delete the check for GAIN.

Comment out code to define metrics, since the BLAST ICD does not define any metrics yet. (But do not delete it entirely because there may be metrics later.)

7 2008-03-12

7.1 Edit blast2caom.py – createPlane (continued)

Process, Input and Output are in the provenance project, which is installed. I will need to set my class path correctly to make this work, but for now I have imported the modules:

```
from ca.nrc.cadc.prov import *
from ca.nrc.cadc.orm.prov import *
from ca.nrc.cadc.orm.prov.hibernate import *
```

8 2008-03-16

8.1 Provenance Use Cases

The kinds of queries we want to resolve using the provenance tables include:

a) Find all Inputs required to reproduce the selected Output
b) Find all Outputs to which the selected Inputs have contributed
c) Find all Outputs that may have been affected by a particular Process
d) Make cutouts of images, weight maps, etc. that were produced by the same Process from the same Inputs.

In a) and b) we might want to place additional constraints for particular kinds of planes, and we may want to qualify the Inputs (Outputs) as proximal, ultimate or all. Thus a source catalog might have been derived from a deconvolved image that itself was derived from a basic reduced image:

REDUCED ⇒ DECONVOLVED ⇒ SOURCE CATALOG

In this example:

```
    proximalInputs("SOURCE CATALOG") returns (DECONVOLVED)
    ultimateInputs("SOURCE CATALOG") returns (REDUCED)
    inputs("SOURCE CATALOG") returns (REDUCED, DECONVOLVED)
```

(BLAST does not currently contain catalogs, but other, more complex archives like the JSA will have many examples of such Input-Output chains.)

8.2 Provenance Structure and Cutouts

Recording provenance dependency using indentation, the actual structure of a BLAST observation is:

```
    REDUCED
          DECONVOLVED
    NOISE
          DECONVOLVED NOISE
    HITS
```

The DECONVOLVED and "DECONVOLVED NOISE" products are not generated for all observations.

If the user requests a cutout from a REDUCED plane, the UI should be prepared to make corresponding cutouts from the NOISE and HITS planes.

QUESTION q1: Should the cutout propagate to Output planes like DECONVOLVED?
QUESTION q2: Should a cutout in DECONVOLVED propagate to Input planes like REDUCED?

8.3 Provenance Ingestion

The provenance library requires that all required observations and planes will already be present before the provenance can be constructed, and will throw an exception if an input or output plane cannot be located.

When ingesting a batch of related files, the order of processing cannot easily be constrained to ensure that the provenance dependencies are satisfied before each file is ingested. It is NOT a good idea to construct the provenance at the same time a plane that uses the provenance is constructed. A simple solution that is probably appropriate for BLAST would be to postpone the construction of the provenance until all of the available files have been ingested into CAOM. This might not be practical for more complex archives.

QUESTION q3: Since make and ant are good at tracking dependencies, and it would be easy to construct a make/build file automatically, would it be useful to employ make or ant to ensure that the products are ingested in the correct order?

The simple model of data processing that the Provenance captures is that a single invocation of a Process takes a set of Inputs and produces a set of Outputs. In real data processing systems, each invocation of the reduction system make take numerous sets of Inputs and produce the corresponding sets of Outputs. In such cases, we can almost always group the Inputs and Outputs into distinct sets that belong together. For BLAST each of these sets will correspond to a single Observation. Note that in the Provenance model the lists of Inputs and Outputs are aggregated into the Process, so each distinct set of Inputs and Outputs defines a unique instance of the corresponding Process, even if all the arguments to the Process constructor itself are the same.

In more complex systems there may be a chain of Input->Outputs that represent the stages of the reduction, all managed by the same data reduction process. The hypothetical chain

REDUCED ⇒ DECONVOLVED ⇒ SOURCE CATALOG

would be an example. Again, the Process constructor would have the same arguments for the stage REDUCED ⇒ DECONVOLVED as for the stage DECONVOLVED ⇒ SOURCE CATALOG and would create separate rows in the process table, but the different instances would be distinguished by their Input and Output lists.

It is not necessary for a Process to have any Inputs. This is an important principle for BLAST because we do not archive the raw data, nor would anyone outside the BLAST team be able to re-reduce it. The Output planes REDUCED, NOISE and HITS are all produced by the SANEPIC process, but have no Inputs.

The Output planes DECONVOLVED and "DECONVOLVED NOISE" are also produced by SANEPIC, but have the Inputs REDUCED and NOISE, respectively.

Each BLAST observation would therefore result in three rows in the process table, all for the SANEPIC process, but one with no Inputs and three Outputs (REDUCED, NOISE and HITS), one with the Input REDUCED and the Output DECONVOLVED, and one with the Input NOISE and the Output "DECONVOLVED NOISE".

QUESTION q4: Would it be useful for the UI if EVERY plane had an associated Output that labeled the role that the data fills in the data reduction system (e.g. "RAW" for raw data), independent of the physical description provided by CAOM?

Note that this question does not ask whether the role should be in CAOM – it should not because this is a data reduction concept that may be handled very differently for different archives and instruments. The question is whether a particular archive/instrument could simplify its UI if the ingestion software assigned an Output to each Plane.

9 2008-03-19

Ed clarifies that in the Artifact Energy we can use the SpectralFunctions and bandpassNames defined in the ICD for all of the data, not just the Swedish data from which they were derived.

9.1 Provenance again

The CONVOLVED and “CONVOLVED NOISE” depend upon both REDUCED and NOISE, so in these files we should find PRVCNT=2 and keywords PRV1 and PRV2.

This version of the blast2caom.py script will not attempt to create the provenance structures, which require planeIDs for the Input and Output objects that are not available until the script has run to completion. It is unclear at the moment whether it would be better to control the order in which files are ingested (using make or ant) or simply to write a separate script to build the provenance after the files have all been ingested.

In any case, I have commented out the bits of code that tried to create Output and Process objects and will work on getting this script to ingest metadata only into the CAOM tables. Without addressing the provenance structures, the createPlane routine should be complete as it is.

9.2 createArtifact

Pat has pointed out an error in the original macho code. The values of positionAxis1 and positionAxis2 should always (for MACHO and BLAST) be 1 and 2, respectively. The original MACHO code set these values from NAXIS1 and NAXIS2. This has been corrected in the BLAST code.

With this change, createArtifact should be complete.

9.3 createSpatialWCS

The coordsys and equinox code was designed in MACHO to handle only equatorial coordinate systems. I have amended it to accept galactic coordinates as well. I have also made the code tolerant of both RADECSYS and RADESYS to specify the coordinate system (ICD 4.1). The RADESYS and EQUINOX keywords are redundant for galactic coordinates, and do not have to be present according to the FITS standard.

```
    ctype1 = getStringKeyword(header, 'CTYPE' + axis1)
    ctype2 = getStringKeyword(header, 'CTYPE' + axis2)
    if ctype1[0:1] == 'RA':
        if header.containsKey('RADECSYS'):
            coordsys = getStringKeyword(header, 'RADECSYS')
        elif header.containsKey('RADESYS'):
            coordsys = getStringKeyword(header, 'RADESYS')
        else:
            raise RuntimeError, 'Neither RADECSYS nor RADESYS keyword'+\
                          ' found in FITS header for equatorial coordsys.'

        if not header.containsKey('EQUINOX'):
            raise RuntimeError, 'EQUINOX keyword not found in FITS header.'
        equinox = header.getDoubleValue('EQUINOX')
    elif ctype[0:1] == 'GL':
        coordsys = None
        equinox  = None
    else:
        raise RuntimeError, 'Coordsys is neither equatorial nor galactic.'
```

(ICD 4.1)

CAOM requires that the FITS headers use the CD matrix format for the special WCS, so it is reasonable to demand that the matrix elements CD1_1 and CD2_2 be present. However, if the coordinate axes are aligned with the array axes, some FITS writers will refuse to write out the matrix elements CD1_2 and CD2_1, which have the default value 0.0 according to the WCS standard (WCS1, section 2.1.2).

```
    if header.containsKey('CD1_2'):
        cd12 = header.getDoubleValue('CD1_2')
    else:
        cd12 = 0.0

    if not header.containsKey('CD2_1'):
        cd21 = header.getDoubleValue('CD2_1')
    else:
        cd21 = 0.0
```

10 2008-03-20

10.1 createSpectralWCS

The MACHO variant of this class is very macho-specific. Unfortunately, the BLAST variant will also be very

BLAST-specific, using the filter name in the FITS header FILTER to select the correct SpectralWCS. The table translating the bandpassName to the SpectralFunction arguments in ICD 4.2 (and ICD 4.2.1) has been hardcoded, with an exception raised if the value of FILTER is anything but 250, 350 or 500:

```
    if not header.containsKey('FILTER'):
        raise RuntimeError, 'FILTER keyword not found in FITS header.'
    bandpass_name = header.getStringValue('FILTER')

    if bandpass_name == "250":
        crval = 210.0
        cdelt = 92.0
    elif bandpass_name = "350":
        crval = 295.0
        cdelt = 130.0
    elif bandpass_name = "500":
        crval = 417.0
        cdelt = 207.0
    else:
        raise RuntimeError, 'FILTER value must be one of 250, 350 or 500.'
```

The remaining arguments have defaults assigned at the top of the program (ICD 4.2.2).

10.2 createEnergyInterval

The code in createEnergyInterval is almost identical to that in createSpectralWCS, except that we need to set the interval ends instead of the cdelt:

```
    if not header.containsKey('FILTER'):
        raise RuntimeError, 'FILTER keyword not found in FITS header.'
    bandpassName = header.getStringValue('FILTER')

    if bandpassName == "250":
        crval1 = 210.0
        crval2 = 302.0
    elif bandpassName = "350":
        crval1 = 295.0
        crval2 = 425.0
    elif bandpassName = "500":
        crval1 = 417.0
        crval2 = 624.0
    else:
        raise RuntimeError, 'FILTER value must be one of 250, 350 or 500.'

    return EnergyInterval(cval1, cval2, crval2-crval1, bandpassName, None)
```

11 2008-03-23

11.1 createTimeInterval

The observed time interval is recorded in the FITS headers DATE-OBS and DATE-END. (ICD 4.3: The former is an official FITS standard, the latter is specific to the JCMT and BLAST.)

```
    if not header.containsKey('DAT-OBS'):
        raise RuntimeError, 'DATE-OBS keyword not found in FITS header.'
    dateobs = header.getStringValue('DATE-OBS')
    if not header.containsKey('DATE-END'):
        raise RuntimeError, 'DATE-END keyword not found in FITS header.'
    dateend = header.getStringValue('DATE-END')

    cval1 = MJD(int(dateobs[0:3]), \   # YYYY
                int(dateobs[5:6]), \   # MM
                int(dateobs[8:9]), \   # DD
                0, 0, 0)

    cval2 = MJD(int(dateend[0:3]), \   # YYYY
                int(dateend[5:6]), \   # MM
                int(dateend[8:9]), \   # DD
                0, 0, 0)
```

The routine MJD is non-standard and should be replaced with a library routine:

```
################################################################################
#
# Calculate Modified Julian Date from Gregorian YYYY, MM, DD, hh, mm, ss.
#
################################################################################
def MJD(YYYY, MM, DD, hh, mm, ss):
    """
    Return MJD.  need to test this code, or use library routines
    """

    d = Calendar.getInstance(Timezone.getTimeZone('GMT+0'))
    d.clear()
    d.set( YYYY, MM, DD, hh, mm, ss)

    mjd0 = Calendar.getInstance(Timezone.getTimeZone('GMT+0'))
    mjd0.clear()
    mjd0.set( 1858, 11, 16, 12, 0, 0)

    return ((d.getTimeInMillis() - mjd0.getTimeInMillis())/86400000.0)
```

11.2 getEnergy

The getEnergy uses the previously updated routine createSpectralWCS and is correct as it is.

12 2008-03-31

12.1 Criteria for the re-use or creation of CAOM objects

Use an existing Observation if
        collectionID matches OBSID
otherwise create a new Observation.

Use an existing Plane and Output (note the 1-1 relationship) if
    Plane->Observation.collectionID matches OBSID
    Plane->Output.name matches PRODUCT

```
    Plane->Output.version matches VERSION
otherwise create new Plane and Output objects.
```

```
Use an existing Process if
        Process.name matches RECIPE
        if PRODUCT in ("REDUCED", "NOISE", "HITS")
                foreach Output o in Process.OutputList
                        o->Plane->Observation.collectionID matches OBSID
                        o.name in ("REDUCED", "NOISE", "HITS")
                        o.version matches VERSION
        else if PRODUCT in ("DECONVOLVED", "DECONVOLVED NOISE")
                foreach Output o in Process.OutputList
                        o->Plane->Observation.collectionID matches OBSID
                        o.name in ("DECONVOLVED", "DECONVOLVED NOISE")
                        o.version matches VERSION
otherwise create a new Process.
Add the current Output to its OutputList.
```

(Is OutputList a Python Set to prevent duplicate entries? Matching Process.name to RECIPE is not strictly necessary because the BLAST team have agreed to change VERSION if they use a different process to reduce the data, but this criterion must be true and should generalize properly to other archives.)

```
Use an existing MetaReadAccess if
        getDate() < RELEASE
        MetaReadAccess.planeID matches Plane.planeID
otherwise create a new MetaReadAccess.
```

(Not needed for initial release.)

```
Use an existing DataReadAccess if
        getDate() < RELEASE
        DataReadAccess.planeID matches Plane.planeID
otherwise create a new DataReadAccess.
```

(Not needed for initial release.)

```
Use an existing Artifact if
        Artifact->Plane->Observation.collectionID matches OBSID
        Artifact->Plane->Output.name matches PRODUCT
        Artifact->Plane->Output.version matches VERSION
        Artifact.data matches AdFile("JCMT","FILEID")
otherwise create a new Artifact.
```

13 2008-04-01

13.1 make file for ingestion

For BLAST the dependency rules to ensure that all inputs are ingested before all outputs can be quite simple, because we have a strong assurance that inputs and outputs will be generated and ingested in complete batches. If we ingest all REDUCED and NOISE products before all DECONVOLVED and "DECONVOLVED NOISE" products, then we can be sure that the input dependencies will be satisfied when the provenance objects are created. This can be handled with a very simple make file.

The directory tree in use at the CADC has no place for a random make file, but an easy workaround to to put a perl script in the scripts directory that writes a suitable make file into the current directory. I have created such a script called makeBlastIngest.pl:

```
#!/usr/bin/env perl
if ( -e blastIngest.make ) { unlink "blastIngest.make";}

open MAKEFILE,">blastIngest.make";
print MAKEFILE "
REDUCED    := \$(patsubst %.fits,%.fits.done,\$(wildcard *reduced*.fits))
NOISE      := \$(patsubst %.fits,%.fits.done,\$(wildcard *noise*.fits))
HITS       := \$(patsubst %.fits,%.fits.done,\$(wildcard *hits*.fits))
DECON      := \$(patsubst %.fits,%.fits.done,\$(wildcard *decon*.fits))
DECONNOISE := \$(patsubst %.fits,%.fits.done,\$(wildcard *deconnoise*.fits))

.PHONY: all
all:    \$(REDUCED) \$(NOISE) \$(HITS) \$(DECON) \$(DECONNOISE)

.PHONY: clean
clean:
\t/bin/rm *.done

%.fits.done: %.fits
\t\@echo doing \$<
\ttouch \$\@
";
close MAKEFILE;
exit;
```

This make file records the ingestion of each fits file by touching an empty %.fits.done file. Obviously, this is just a test version of the real makefile, which would run blast2caom.py instead of echoing a message to sysout.

It would be slick, but not necessary for BLAST, to have a script that read the file headers and made real file-by-file dependencies in the make file by reading the FILEID and PRVn headers.

This addresses the problem raised in sections 8.3, 9.1, and 12.1 that provenance objects are needed to create CAOM planes, but require that all Input planes be present in the database before the related Output planes can be created. With this issue settled, blast2caom.py will also create the provenance objects as required.

13.2 SQL definitions for blast_entry and blast_entry_history

(Actually done on March 19, but not recorded.)
These tables are modeled after the CAOM 1.0 equivalents for the JCMT archive, except they include an additional column "status" to record the ingestion status of each file. The status field will have the values

```
-- status will be one of:
-- NULL - new, not in AD
-- 'A' - put into AD successfully
-- 'I' - ingesting into CAOM
-- 'C' - successfully ingested into CAOM
-- 'P' - creating provenance
-- 'Y' - success
-- status will NOT be updated if an ingest into CAOM or provenance creation fails
```

The blast_entry_history.tbl definition was updated to include status today.

Following the model of the CAOM tables, the tbl files first try to drop an existing file before creating the new table.

The commands to create these tables in DEVSYBASE is

```
dalmore:~...trunk/sql> sqsh -S DEVSYBASE -U redmanr -D blast -i blast_entry_history.tbl
Password:
Dropping existing blast_entry_history.
Creating blast_entry_history table
dalmore:~...trunk/sql> sqsh -S DEVSYBASE -U redmanr -D blast -i blast_entry.tbl
Password:
Dropping existing blast_entry.
Creating blast_entry table
```

Note that blast_entry_history must be defined first because blast_entry.tbl tries to set triggers to copy rows from blast_entry to blast_entry_history.

13.3 python blastEntry class

The blast_entry table will be managed in blast2caom.py using a new class blastEntry.

14 2008-04-02

14.1 python blastEntry class (cont.)

Since blast_entry is defined in DEVSYBASE rather than SYBASE during early development, it is necessary to direct all db queries to ulkatcho port 4200 instead of ucluelet port 4100. The default remains SYBASE:

```
servername = 'ucluelet'
server = 'SYBASE'
port = '4100'
```

To use DEVSYBASE, I have added a new switch --devsybase that change the configuration:

```
# First get the command line arguments
try:
    opts, args = getopt.getopt(sys.argv[1:], '', \
         ['help', 'mapping', 'test', 'user=', \
              'password=', 'fileId=', 'devsybase'])
except getopt.GetoptError:
    usage()
    sys.exit(1)
```

Note that this change also removed the MACHO-specific switches --tmpDir and --class.

```
    # use DEVSYBASE instead of SYBASE
    if o == 'devsybase':
        print 'server = DEVSYBASE'
        server = 'DEVSYBASE'
        servername = 'ulkatcho'
        port = '4200'
```

There was a corresponding change in the usage() routine to print out the switch definitions.

It seems that the version of jython in use here is currently 2.1, which is based on Python 2.1. This is before "new-style" classes were introduced into Python, so the syntaxt of the class statement does not allow derivation from object. A request has been made to update jython to the current version (2.2.1).

The interface for class BlastEntry is

```
class BlastEntry:
    def __init__( self, con, fileid, partnum):
    def privateUpdate( self, value):
    def startAD( self):
    def endAD( self):
    def startCAOM( self):
    def endCAOM( self, obsID, planeID, artifactID):
    def startPRV( self):
    def endPRV( self):
```

The constructor __init__ takes a database connection, a fileid and part number (aka extension; always 0 for BLAST). The constructor checks first to see if a row already exists in blast_entry for this entry. If so, it nulls the obsID, planeID and artifactID columns, sets id_type to the default 'FITS', and sets the status to isNew ('N').

The "private" method privateUpdate handles the simple case of updating only the status and is not intended to be called directly. The methods startAD, endAD, startCAOM, startPRV and endPRV simply call privateUpdate with the appropriate status value.

This class has been developed in the file blastEntry.py, which is currently in SVN. It includes a simple main program to test the operation of the class. This is not a Python module, however, and my intension is to copy the class definition back into blast2caom.py, which will function as a self-contained program.

15 2008-04-03

15.1 blast2coam main program

The main program will ingest a single file from the disk into AD and the database. A make file will be used to ensure that the files are ingested in an appropriate order. As discussed above, the creation of observations, planes and artifacts in CAOM also requires the creation of appropriate Provenance structures as well. We must take an approach different from that used in MACHO to ingest files into the archive.

To start, the command line switches must be different. The --number switch will not be needed because one file will be ingested at a time. The --tmpDir switch that defines a temporary directory for processing will also not be needed. As mentioned above, the --class has been added to allow the specification of an arbitrary target classification. In place of a --fileId switch that would be used to fetch files from AD, a --file switch allows the specification of a file name. The --user, --password, and --file switches are now mandatory.

The loop that iterated over files in AD has been eliminated. The try block that encases ingestion is retained, but reading the FITS file must now be the first event, followed immediately by the extraction of the header values needed to control the creation/re-use of the CAOM and Provenance objects.

All occurrences of fileID (formerly set by a command line switch) have been replaced by fileid (read from the FITS header FILEID.

16 2008-04-04

16.1 Flag for accessControl

The createPlane subroutine checks whether the RELEASE date is in the future and sets a flag needAccessControl to signal that records should be created if not already present in the meta_read_access and data_read_access tables. These records require the planeID, which will not be present at this stage of ingestion. The flag is passed up to getObservationPlane, which in turn passes it up to ingest.

The logic is not correct here, because the code does not check whether the release date is in the future when an existing file is replaced. Also, I am not implementing code to create the access control records at this time.

16.2 processORM

Talking with Pat Dowler about how to make the connection between the observation object returned by observationORM(collection, collectionID) and the list of Process objects that have Outputs connected to the Planes in the observation, it seems we will want a generic routine that fetches a list of Process objects, with an interface like:

```
processList = processORM.getByOutputPlaneID( planeIDList[] )
```

We can identify a unique observation from the OBSID header in the FITS file. To find the correct output plane, it will be necessary to gather the list of planeID's in this object, find the list of processes with this call, and then apply archive-specific logic to determine whether an existing process produced this file and whether it is necessary to add another Output to the process.

It would also be useful to do a similar search by Input planeIDs:

```
processList = processORM.getByInputPlaneID( planeIDList[] )
```

Pat has added this to the development schedule for the provenance project, but warns that the code will not be available for another month or so.

Pat warns (and my own experience with Python confirms) that it is NOT safe to interact with the database outside of the hibernate system when dealing with data types like obsID, planeID, etc. Sybase does not have a long integer type, and treats these items as byte arrays. There are serious troubles with byte-order when fetching and storing these values that make hand-crafted code very brittle and ultimately impossible to maintain. I will probably have to rewrite the BlastEntry class (or the blast_entry table) to take account of this fact. The ORMs have methods getID() that safely get ID values, and these methods should be used for all such database interactions.

17 References

ICD – Redman, R. O., 2008, "BLAST Archive ICD"

WCS1 – Greisen, E. W., and Calabretta, M. R., 2002, A&A, 395, 1061, "Representations of world coordinates in FITS"

18 TO-DO List

- Recode the target_classification.
- Set the MetaReadAccess and DataReadAccess for proprietary observations. These are defined in the accessControl project.
- The obsExport views in trunk/sql belong in the obsExport project (in svn). I think for the migration it will be beneficial to have them all together and jointly installable. Feel free to add them there and/or start from macho (since it is based on CAOM 1.0).
- Check with Pat for the correct settings for coordsys and equinox when using galactic coordinates ((None, None) or (ICRS, 2000.0)).