# NRAO Development Study Proposal: ADMIT Preliminary Design

Peter Teuben, Marc Pound, Doug Friedel, Lee Mundy, Leslie Looney

September 20, 2013

## 1 Overview

The ALMA Data Mining Toolkit (ADMIT) is a value-added software package which integrates with the ALMA archive and CASA to provide scientists with quick access to traditional science data products such as moment maps, and with new innovative tools for exploring data cubes. The goals of the package are to (1) make the scientific value of ALMA data more immediate to all users, (2) create an analysis infrastructure that allows users to build new tools, (3) provide new types of tools for mining the science in ALMA data, and (4) increase the scientific value of the rich data archive that ALMA is creating.

For each ALMA science project a set of science quality image cubes exist. ADMIT runs a series of "ADMIT tasks," which are essentially beefed up CASA tools/tasks, and produces a set of Basic Data Products (BDP). ADMIT provides a wrapper around these tasks for use within the ALMA pipeline and to have a persistent state for re-execution later on by the end user in ADMIT's own pipeline. ADMIT products are contained in a compressed archive file *admit.zip*, in parallel with the existing Alma Data Products (ADP)[1].

Once users have downloaded the ADMIT files, they can preview the work the ALMA pipeline has created, without the immediate need to download the much larger ADP. They will also be able to re-run selected portions of the ADMIT pipeline from either the (casapy) commandline, or a (CASA) GUI, and compare and improve upon the pipeline-produced results. For this some of the ADP's may be needed.

ADMIT introduces the concept of a "Virtual Project," which is a framework for a collection of similar ALMA projects. ADMIT is used to manage tasks run across all ALMA projects in the Virtual Project, after which selected results can be data-mined and correlated.

Below, we first outline use cases driving the ADMIT design and the resulting required functionality and products. Then we lay out the core XML components of ADMIT, describe the structure of Basic Data Products and tools, and conceptualize the software layers and interfaces. Finally we outline the requirements of

[1]Examples of ADP in a project are the Raw Visibility Data (in ASDM format) and the Science Data Cubes (in FITS format) for each source and each band

the ADMIT GUI, design and development of which is staged after the ADMIT XML and Python are in place. In the Appendices, we give greater detail on proposed XML structures.

# 2 Use Cases

## 2.1 Archive

This case covers a principal goal for ADMIT: produce a small file in the ALMA Archive that users can quickly download to inspect their data without needing to download the large full data cubes. ADMIT should produce in the ALMA pipeline a single, compressed file, `admit.zip` for a given project. A project contains one or more sources, and for each source the ALMA Archive will create four image data cubes, in FITS format. From these cubes, the ADMIT pipeline, running on the ALMA Archive, will produce Basic Data Products summarizing the observations and a descriptive `admit.xml` file and add this as `admit.zip` to the archive. The BDPs created for data inspection will be

1. A summary of the observations (source information, band information)
2. A list of detected spectral lines in each band (name, rest frequency, $V_{LSR}$, detection probability)
3. A small datacube centered in velocity around each detected spectral line.
4. Zeroth, first, and second moment maps for each detected spectral line
5. For each detected spectral line, a spatially-averaged spectrum through the peak in the zeroth moment map.
6. For each plane in each band cube created by the ALMA archive, a simple table of statistics: minimum, maximum, mean, median.
7. An overlap integral map that shows a combined view of zeroth moment maps of each spectral line.
8. Where applicable, a continuum map made from emission-free channels.

## 2.2 First Look

The user that has gone to his/her project on the archive, and downloaded the `admit.zip` file. The ADMIT environment in casapy will then either print to screen, or use a GUI, outlining all BDPs present, and allow looping over and visualizing them via Python methods. There is no essential computing needed in this step.

### 2.3 Rerun (Sections of) Pipeline

After a first look, the user may wish to modify the BDPs, for instance, by adding spectral line to the line catalogue (LineList), recreating moment maps based on a lower clip level or by a different method of computing moments. The input methods and parameters to compute a BDP can be changed, through by or, in later ADMIT versions, through a GUI.

### 2.4 Morphological Analysis

The user wish to make use of advanced data analysis techniques to discover structure in a source or compare structure across maps of different spectral lines. This could be done by providing an interface to invoke standard algorithms such as *clumpfind* or dendrograms. A new algorithm we propose to develop, taken from computer science visualization techniques, is using *description vectors* to determine image saliency. This provides an application-independent, purely mathematical measurement of information. For astronomical images, description vectors can be chosen from any number of properties of the emission. Since these algorithms are resource intensive, and not all users may want them, the associated BDPs would not be computed in the ALMA archive pipeline.

### 2.5 Virtual Projects

In this more advanced use case, the user selects a number of sources from one or more projects in a virtual ADMIT container. Any of the basic ADMIT procedures can now be re-run and/or reviewed, but looped over the selected sources. Users can write their own procedures, and store persistent data back into `admit.zip`. ADMIT has interfaces using Python to extract numbers and Python arrays out of `admit.zip`, and thereby allow for flexible data mining, linked data plotting, etc.

## 3 Basic Data Products

The pipeline results and outputs of ADMIT tasks are described as Basic Data Products. A BDP may contain computed or harvested information as well as point to an external resource, such as an image file. We expect for all BDPs there will be a one to one correspondence with an ADMIT task. All BDP instances will have the following structure:

```
BDP
  name
  data element 1
  data element 2
  ...
  data element N
  task
```

`dependencies`

where *name* is the descriptive name ("summary," "linelist"), *data elements* are harvested information, computed values, tables, or pointers to images (with a True/False value if that image is exported to `admit.zip` as well), *task* is the ADMIT/CASA task used to create this BDP with all all task parameter values specified, and *dependencies* are other BDPs upon which this one depends. If a dependency BDP for this BDP changes, then this BDP should be recomputed. For instance, the overlap integral depends on moment maps, so if the moment maps are recomputed the overlap integral should be as well.

# 4 XML Structure and Types

The XML structure should be complete enough to capture proposed use cases, but not so restrictive that it precludes future use cases. To that end, a limited hierarchy is desirable, with basic "objects" that can be contained within one another but are not *defined* within one another. However, we do need to follow what we understand to be the basic hierarchical structure that the archive produces Project→ Source → Band Cube → Line Cube. We argue it is not necessary to expose the Project container to the user, e.g. in the case of a data mining operation that spans multiple projects. The science is in the sources not the project. To that end, the Project container is extremely thin: it contains only the identificiation code, everything else is inside source containers.

The following list describes the basic XML types from which `admit.xml` and the Basic Data Products may be composed. Note all tables will be stored in VOTable format. Details of the XML structure of each are given in Appendix B.

- **Project** – This is a thin structure giving the name of the project and containing one or more Sources.
- **Source** – This container describes the source parameters (name, coordinates, etc) and contains all BDPs for this source.
- **Image** – An image reference, typically FITS or PNG. The image data file itself is normally not contained inside the XML, but external on disk.
- **Spectrum** – A one-dimensional spectrum through a data cube at a given location.
- **Statistics** – A summary of statistics computed for an image or spectrum
- **Task** – A full description of the task used to create a particular BDP. This includes task name and all parameters with which the task was invoked.
- **Dependencies** – A list of BDPs upon which a particular BDP depends.
- **Date** – The date and time when a particular BDP was computed.

Using these types, the `admit.xml` and BDP XML definitions take shape:

```
<project name="c1234_abcde_hijkl">
<source name="abcde">
<BDPName type="bdp">
   <element1>
   <element2>
   ...
   <elementN>
   <task>
   <dependencies>
</BDPName>
...
<BDPName type="bdp">
   <element1>
   <element2>
   ...
   <elementN>
   <task>
   <dependencies>
</BDPName>
</source>
</project>
```

As a detailed BDP example, a first moment map BDP might look like:

```
<moment type="bdp">
   <integral>1</integral>
   <description>velocity centroid</description>
   <method>clip</method>
   <image>...</image>  % e.g. the FITS image
   <image>...</image>  % e.g. the PNG image
   <task type="function" name="moment" category="imagefu">
        <param type="float" name="clip">
           <value>0.1</value>
        </param>
        <param type="float" name="vmin">
           <value> -123.0</value>
        </param>
        <param type="float" name="vmax">
           <value> 234.0</value>
        </param>
   </task>
   <dependencies>linecube</dependencies>
   <date>YYYY-MM-DDTHH:MM:SS</date>
</moment>
```

## 4.1 Virtual Project

A virtual project is a user or ADMIT defined structure that can span multiple projects, sources, and bands. The members of the virtual project can be grouped into one or more subgroups that can be processed in similar manners or to a similar depth.

The virtual projects defined by the user will have their own XML structure, not saved in the `admit.zip` file(s), but in a separate XML file. This file will

have links to the members of the virtual project. The virtual project will contain nodes for dividing up the individual parts so that they can be grouped by the user or ADMIT, for further analysis. A sample XML structure follows:

```
<project type="virtual">
    <group name="name1">
        <common item: cutoff, molecule, etc>
        <member>
            <file name="file name"/>
            <id name=""/>
        </member>
        ...
    </group>
    ...
</project>
```

Each source/band should have a unique ID (e.g. project-source-band) so that the virtual projects can link to all associated data. The link would include both the absolute file name and path and the ID for each member of the virtual project. Each time the virtual project is opened these links should be followed and verified (possibly loading the data also). If a file is not where it is expected the user will be prompted to locate the file again. The links would be like the Unix hard link, i.e. it will point to the original data/images unless the analysis is redone, at which point there will be a local version (in the virtual project) of the products, leaving the originals intact for comparison. There will also be an option for the user to package up the associated parts into its own `admit.zip` file so that it is portable. The virtual projects will retain the same project-source-band structure as the typical project so that processing in ADMIT and other programs is straight forward.

### 4.1.1 Virtual Project Example

There are two types of `admit.zip` files: the ones belonging to a genuine ALMA archive project, e.g. `c0123_admit.zip` and the ones belonging to a virtual project, e.g. `test1_admit.zip`. Virtual Projects inherit their projects by virtue of a hard link. Let's say `c0001`, `c0012`, and `c0123` are three projects that are in a virtual project, called `test1`, the directory `test1` will contain hard links the directories `c0001`, `c0012`, and `c0123`, and the sub-directories needed by virtual of their selection. This way, if in the virtual project one of the BDPs has changed, they overwrite their original version. [2]

Assume the user has downloaded a few ADMIT files, for projects `p1` and `p2` and `p1` has two sources, `s1` and `s2`.

```
% ls
  p1_admit.zip
  p2_admit.zip
% admit --extract p*zip
```

[2] Linux command: `cp -al`, Mac command: `pax -rwl`. Or `rsync`

```
p1/admit.xml              (via admit)
   p1.s1.b1.fits          (only present from archive if downloaded)
   p1.s1.b2.fits          (unique names containing P, S and B)
   p1.s1.b3.fits
   p1.s1.b4.fits
   p1.s2.b1.fits
   p1.s2.b2.fits
   p1.s2.b3.fits
   p1.s2.b4.fits
   s1/l1/l1_cube.im       (a recomputed casa cube)
         mom0.fits        (fits and/or jpg, via admit)
         mom1.fits
         mom2.fits
      l2/l2_cube.im
      ...
      overlap1.jpg        (overlap intergral from all Lines)
      ...
   s2/
   ...
p2/admit.xml
   ...
...
```

# 5 Interfaces

Since CASA interfaces are all in Python, most – if not all – of ADMIT will be in Python. Some adaptations to CASA routines will likely be needed, and this may invoke some changes to the CASA C++ core code. A simple example we encountered is a robust median, which is not in CASA core but is quite useful in computing statistics. We envision the ADMIT software structure to be three layers: the base layer encapsulates all XML operations, the middle layer is the ADMIT pipeline infrastructure and task interface, and the top layer is the user interface.

## 5.1 XML I/O and Manipulation Layer

This layer is to standardize the interaction with the XML for all higher level routines and to enable and simplity the XML stucture that we decide upon in the XML definitions. A choice between SAX and DOM parsing of XML will have to be made. SAX appears to be the favorite, based on much smaller memory footprint and more flexible parsing. The end user will not have access to this layer.

## 5.2 Pipeline Infrastructure and Tasks

The basic workflow of ADMIT is to open a project, compute a set of tasks, in a serial fashion, and close the project. This workflow is is the same regardless if ADMIT is run in the ALMA archive or by an end-user. Especially in the latter case, decisions are made based on interactive use with the pipeline. In

particular it will be important to be able to compare Basic Data Products of different invocations of an ADMIT task, for example, comparing the velocity field of a line using two different methods of computing the moments.

The following is the required pipeline functionality.

1. Opening a (single) project that is not ADMIT-enabled yet. There is no `admit.zip` present, so everything needs to be initialized: find the sources, the number of bands per source, and the variables describing each source/band (e.g. RA,DEC,vlsr,freq ranges etc.). The ADMIT XML needs to be created. No other tasks will be run in the pipeline, although this summary could be seen as the default initialization task.

2. (Re)opening from an existing XML. All structures will need to be intialized. Status and Dependency list of the various data products need to be set. Missing ADP and BDP are to be identified.

3. Opening from an existing ZIP. This is very similar to the previous item, but must also create a directory structure hierarchy and populate basic data products.

4. (Re)compute a Basic Data Product. The dependencies need to be reviewed (multiple are possible), as well as allow to bypass recomputing these if not desired. The method needs to be selected, parameters for this method set, as well as the style of execution When all is set, the task can be executed, and the BDP is created. All of this is wrapped in the `task` XML tag, compatible with the CASA `task` XML tag for inclusion into CASA.

5. Save a project: either the ADMIT Python object is serialized into XML, or in addition all associated allowed basic data products are wrapped in a single `admit.zip` file. Not all data products are normally wrapped in `admit.zip`. For example spectral line cubes are not, given their size. Moment maps have both PNG and FITS versions, both could be exported.

Another part of the "middleware" are the base class definitions of an ADMIT task and a Basic Data Product. We expect the ADMIT task base class will follow the CASA task structure closely if not identically. Using this class, all ADMIT tasks can be constructed and serialized to XML. The ADMIT task base class will be flexible enough to allow user-defined tasks as a future enhancement. Similarly, the Basic Data Product base class is the foundation of all BDPs and is serializable to XML.

## 5.3 User Interface

This is the layer that enables the user interaction with CASA and the pipeline. The initial user interface will be purely through Python/CASA commands. In addition to Python interfaces for all ADMIT tasks, We will provide shortcut commands for common operations, e.g., displaying a summary page.

A graphical user interface is anticipated to guide users through using ADMIT in an intuitive way. As a project is opened, a tabbed browser is display with for each source presented in a separate tab. This will for instance give an overview of cube statistics, a PV diagram, moment maps, lines detected. Any of the BDP can be recomputed with a pop-up parameter editor. Selections on projects/sources/lines can be made and grouped into a virtual project for re-processing these data in a common way.

# 6 Specific ADMIT Tasks

Here we describe a number of core ADMIT tasks in more detail.

## 6.1 CubeStats

The primary idea of CubeStats is to provide initial guidance for line identification. For each channel it tabulates the min, max, RMS, mean and median. A Robust Median is preferred, but is computationally expensive. (A 256x256x2048 cube requires 5 minutes as opposed to 5 seconds for a standard median).

## 6.2 LineList

A LineList is a simple table, tabulating which lines are present in the cube, and a likelihood the line is present. The likelihood is computed as *Signal/(Signal+Noise)*, thus is always between between 0 (unlikely) and 1 (likely). Based on more advanced concepts (e.g. PVCorr), a revised line list could be derived, resulting in a better line list source detections algorithms get more advanced in the pipeline.

## 6.3 PVCorr

In this task a position-velocity diagram is used more accurately determine the frequencies of detected lines. To create a good PV diagram, a position angle is needed, which can be obtained by summing up all emission above some value (e.g., a few times the RMS noise) and using a moment of inertia in the resulting summed emission map. When the axis ratio of this emission is round enough, an XYVcorr might give better S/N. The strongest line defines an area that is then cross-correlated in the V (frequency) direction and this defines a signal and noise within which line identification can take place.

## 6.4 XYVcorr

This is the 3D extension of the 2D PVcorr method. In theory should be more sensitive/accurate, if the emission is really not along some major axis. This has not been well tested.

### 6.5 Moment

Computing the moments of a line cube (really: deriving the total emission, the mean velocity and velocity dispersion) can be done with a number of methods, ranging from simple moments along the velocity axis, perhaps aided by masks generated from convolved data, to fitting model spectra. All of these are encapsulated.

### 6.6 FeatureList

A method has to be selected that assigns features (blobs, clumps, dendograms) to a line cube. For non-overlapping features, a simple table will suffice, for embedded features (e.g. dendogram), a hierarchical table will be needed. This latter model is not one of the basic data types (tables, image/cubes) that we support, so a new table type might be needed here.

### 6.7 DescriptionVector

A "DV" can be operated in in a line cube (or even a band cube) by assigning such a vector belonging to the features in the cube. But it can also be spanning multiple lines, in each line cube computing a vector that is now spanning lines, instead of features. There can be a Description Vector per Line Cube, but also per source, or even per (virtual) project. For example, a Description Vector in a Line Cube could be Feature based (e.g., the moments of inertia) or across different Lines, much like an Overlap Integral (cf. PCA), where they represent a multi-dimensional "color" of a feature.

## 7 Summary and Future Work

**: WE NEED SOME KIND OF WRAP-UP SUMMARY HERE**

## A ADMIT Tree Overview

Without the full, cumbersome to read XML syntax, here is an overview of the tree that `admit.zip` will contain. The `[N]` notation means this item will occur `N` times at this level, for example for ALMA data you would see `NB=4`.

```
Project(name)[NP]
   Summary
      <atask name=at_summary>
   Source(name)[NS]
      Summary
         ra,dec,vlsr,...
         <atask name=at_summary>
      Band(number)[NB]
         URI:im
         Summary
```

```
      FreqMin,FreqMax,FreqStep
   CubeStats
       VoTable:tab
       <atask name=at_cubestats>
       <dep>
          URI:im
   PosVelSlice
       URI:im
       file:jpg
       <atask name=at_pv>
       <dep>
          URI:im
   LineList
       VoTable:tab
       <atask name=at_band2line>
       <dep>
          CubeStats
LineList
    votable:tab
    <atask name=at_linemerge>
    <dep>
       band[NB].LineList
Continuum(name)
    URI:im
    file.jpg
    <atask name=at_continuum>
    <dep>
       Band(this)
Line(name)[NL]
    LineCube
       URI:im
       <atask name=at_reframe>
       <dep>
          LineList
    RMS (since cubestats can differ per channel)
    Mom0
       URI:im
       file:jpg
       <atask name=at_moment>
       <dep>
          LineCube
    Mom1
    Mom2
    PeakSpectrum
       VoTable:tab
       <summary>
          Peak, RMS, V0, FWHM, SdV
       <atask name=at_spectrum>
       <dep>
          LineCube
    IntegratedSpectrum
       VoTable:tab
       <summary>
          Peak, RMS, V0, FWHM, SdV
       <atask name=at_spectrum>
       <dep>
          LineCube
```

```
        FeatureList
           VoTable:tab
           <atask name=at_feature>
           <dep>
              LineCube
        DescriptionVector
           VoTable:tab
    DescriptionVector
        VoTable:tab
DescriptionVector
    VoTable:tab
```

# B Basic XML Types

.

All XML tags will be lower case; no mixed case tags allowed. In general, we follow the principal that data go in elements and metadata about elements go in attributes. (However, CASA tasks follow the convention that the *name* is an attribute so we will also.) For instance, if an element has a fixed type, or should not be changed by the user, we would use an attribute:

```
<myelement type="float"> 123.4 ></myelement>
<myotherelement type="string" readonly="true">Can't touch this</myotherelement>
```

## B.1 Project

```
<project name="(the ALMA specified project name)"
</project>
```

## B.2 Source

```
<source>
    <name></name>
    <coordinate>
        <!-- map this from/to the official FITS WCS convention -->
        <crvalN>
        <ctypeN>
    </coordinate>
    <equinox>
    <type> [galactic, extragalactic, etc]
</source>
```

## B.3 Image

```
<image>
   <URI>
   <type> [data or thumbnail]
   <exported> [ true if contained in admit.zip]
   <description> [moment zero, moment one, spectral cutout, full cube.]
   <naxisN>
   <crpixN>
   <crvalN>
   <ctypeN>
   <statistics></statistics>
</image>
```

## B.4 Spectrum

```
<spectrum>
   <!--
     Watch out for gridding effects:  If frequency has uniform gridding,
     velocity will not unless a regrid in that axis was done.
     Need proper WCS for this.
   -->
```

```
   <start freq>
   <start vel>
   <dvel>
   <nchan>   <!-- perhaps VOtable encapsulates number of channels already -->
   <statistics> </statistics>
   <votable> <!-- The spectrum actual data -->
</spectrum>
```

## B.5 Statistics

```
   <statistics>
<!-- One can have multiple areas over which statistics are measured.
     Does casa multiple boxes print two stats or one?
 -->
        <region>
            <number>
            <!-- CASA Region Text Format, including channel or velocity range.
                 See
                 http://casaguides.nrao.edu/index.php?title=CASA_Region_Format
              -->
            <crtf type="string" >
            <mean>
            <max>
            <rms>
            <clip>
        </region>
   </statistics>
```

## B.6 Task

We expect the ADMIT task will defined exactly as a CASA task, for which the XML representation is already defined as follows:

```
<task type="function" name="foobar" category="fumbar">
    <shortdescription>
    <description>
    <input>
    <param>
        <description>
        <any>
        <value>
    <returns>
    <example>
</task>
```

The only difference is that `casapy` keeps a local `task.last`, and a read-only version in `$CASA/share/xml/task.xml`, in ADMIT the current values are stored in the current `atask` of `admit.xml`.

## B.7 Dependencies

This is a simple comma-separated list of BDP names. Note these are *not* chained dependencies. If BDP1 depends on BDP2 and BDP2 depends on BDP3, then BDP1 lists only BDP2 in its dependency list.

```
<dependencies>Name1,Name2,...,NameN</dependencies>
```

## B.8 Date

This is the date and time following the FITS specification:

```
<date>YYYY-MMM-DDTHH:MM:SS.SSS</date>
```

# C BDP XML Definitions

Here we detail XML definitions of specific Basic Data Products. Those marked "TBD" are still undergoing design.

## C.1 Band

```
<band type="bdp">
  <number>
  <image>

  <summary type="bdp>
  <linelist type="bdp">
  <cubestats type="bdp">
  <posvelslice type="bdp">

  <!-- An "image" dependency means this BDP depends on
       its enclosed image URI or, if it has none, that of its parent.
   -->
  <dependencies> image </dependencies>

  <date>YYYY-MM-DD HH:MM:SS</date>
</band>
```

## C.2 Line List

```
<linelist type="bdp">
   <votable>
   <!-- columns: name, restfreq, peakchannel, probability, where -->
   <!--     name  : fully qualified name. e.g. 12C17O(1-0)       -->
   <!--  restfreq : rest frequency in GHz                        -->
   <!--  peakchannel : channel of peak probability               -->
   <!--  probability : number between 0 and 1 indicating probability
                       that peak channel is this line.
                       0 = 0% probability, 1 = 100% probability.
     -->
   </votable>
   <task type="function" name="linelist" category="admit"> </task>
   <dependencies>cubestats</dependencies>
</linelist>
```

## C.3 Cube Statistics

```
<cubestats type="bdp">
<!-- There will be one of <statistics> for each plane of the cube.
     The default region will be all of xy space, with the region
     keyword defining the boundaries and channel.
-->
  <statistics>  <!-- first channel -->
  ...
  <statistics>  <!-- last channel -->
```

```
  <!-- Depends on its parent's image -->
  <dependencies>image</dependencies>

  <task type="function" name="cubestats" category="admit"> </task>

</cubestats>
```

## C.4 Moment

```
<moment type="bdp">
   <!-- moment value: 0,1,2 -->
   <integral>

   <description>
   <method>

   <!-- The FITS or CASA image -->
   <image>
       <type>data</type>
       <exported>false</exported>
       ...
   </image>

   <!-- The PNG image -->
   <image>
       <type>thumbnail</type>
       <exported>true</exported>
       ...
   </image>
   <task type="function" name="moment" category="admit">
        <param type="float" name="clip">
           <value></value>
        </param>
        <param type="float" name="vmin">
           <value></value>
        </param>
        <param type="float" name="vmax">
           <value></value>
        </param>
   </task>
   <dependencies>linecube</dependencies>
   <date>YYYY-MM-DD HH:MM:SS</date>
</moment>
```

## C.5 Position Velocity Slice

```
<pvslice type="bdp">
<param type="float" name="pa"> Position Angle </param>
<image></image> <!-- the position velocity diagram -->
<image></image> <!-- Reference to the line cube from which the PV diagram was computed -->
<dependencies>linecube</dependencies>
<date>YYYY-MM-DD HH:MM:SS</date>
</pvslice>
```

## C.6 Continuum Map

```
<continuum type="bdp">
<param type="int" name="nchan"> number of channels used</param>
<param type="float" name="bw"> total bandwidth used</param>
<image></image> <!-- the continuum image -->
</continuum>
```

## C.7 Summary

TBD

## C.8 Line Cube

TBD

## C.9 Overlap Integral

TBD

## C.10 Feature List

TBD – Depends on morphological analysis algorithms.

## C.11 Description Vector

TBD – Depends on analysis algorithm.

# D VOTable

All ADMIT tables will be VOTables. THe VOTable XML format is as follows:

```
<!DOCTYPE VOTABLE SYSTEM "http://us-vo.org/xml/VOTable.dtd">
<votable>
   <description>...</description>
   <resource  type="results">
      <description>...</description>
      <info name="QUERY_STATUS"  value="OK"/>
      <info name="distinct_dataset"  value="70"/>
      <table id="t1">
         <description>...</description>
         <field id="Ra"  unit="deg" datatype="char" name="ra_targ"  ucd="POS_EQ_RA_MAIN">
            <description>...</description>
         </field>
         <field id="Dec" unit="deg" datatype="char" name="dec_targ" ucd="POS_EQ_DEC_MAIN">
            <description>...</description>
         </field>
         <data>
             <tabledata>
             <tr>
                <td> 161.657 </td>
                <td>  17.152 </td>
             </tr>
             </tabledata>
         </data>
      </table>
   </resource>
</votable>
```