Getting Started in AIPS++

K.R. Anantharamaiah, RRI
T.J. Cornwell, NRAO

Version 3.3/February 2004

Contents

1 Introduction

This document is intended as an introduction and guide for newcomers to the AIPS++ environment. You can find more detailed information and reference material at the AIPS++ Web site. This document should familiarize you with the basic philosophy, terminology, and structure of AIPS++ and to help you get started on useful processing of radio astronomical data, especially those from aperture synthesis arrays. This document is written in a semi-tutorial style and, for maximum benefit, should be used at a workstation as an introductory AIPS++ session.

2 Overview

General radio astronomical processing AIPS++ provides facilities for calibration, editing, image formation, image enhancement, and analysis of images and other astronomical data. A major focus is on reduction of data from both single-dish and aperture synthesis radio telescopes. Although the tools provided in AIPS++ are mainly designed for processing data from varieties of radio telescopes, the package is expected to also be useful for processing other types of astronomical data and images.

Tool-based approach AIPS++ has been designed to be tool-based and potentially programmable by the user. This means that terminology and the look and feel of AIPS++ is somewhat different from other astronomical processing packages. However, after a little bit of getting used to, you will find that most of the tools and applications that an astronomer is looking for (and hopefully many more) are available.

Access to all data In AIPS++, the user has direct access to all portions of a data set or an image. Therefore, all aspects of the data can be manipulated at will using either existing applications in AIPS++, or user developed applications, or simply using calculator-type commands from the terminal.

Graphical Interfaces AIPS++ is designed to take advantage of modern computer software and hardware. Mouse-driven Graphical User Interfaces (GUIs) are used extensively for handling various applications, computations, and image display.

Web-based documentation In order to access the extensive AIPS++ reference material and other documentation, the *Help* menu that appears on most GUIs can directly steer a Web browser to the appropriate section. For those of you who are more comfortable with a type-on-terminal approach, AIPS++ can be completely controlled from the Command Line window. A combination of both Command Line window and GUI windows is probably most appropriate.

Powerful interactive command language The command language used in AIPS++ is called *Glish*. This is a very capable interactive programming language which allows high level operations on vectors and arrays. All data can be accessed and manipulated from *Glish* and placed back into AIPS++. Tasks in AIPS++ are written in **C++** for efficiency and bound to *Glish*, therefore, all capabilities of interest to the user are controllable from the *Glish* Command Line window.

3 How to use the Documentation

The documentation for AIPS++ is layered and so it is useful for you to understand that layering so that you can use it effectively.

3.1 For the New User

3.1.1 Glish

- In the document you are reading now (Getting Started with AIPS++) there is a Glish Overview section that explains some of the basic syntax of *Glish*.
- *Getting Started with Glish* by Rick Fisher describes the basics of how to use Glish, the scripting language that we use to control AIPS++ `tools` from.

3.1.2 AIPS++

- The document you are reading now (Getting Started with AIPS++), gives you general context and information about the system. In reading this document, it is useful for you to know some simple *Glish* syntax because there are many example provided in *Glish*.
- Glossary - you might also find the Glossary a useful source of information.

- Dictionaries - we also provide some software translation dictionaries for users of other packages. If you are used to doing something in that package, this helps you find out how to do it in AIPS++.

  There are dictionaries for AIPS and Miriad.

3.2 For High-Level Information

3.2.1 Cookbook

The AIPS++ Cookbook is designed to provide information on how to do relatively high-level things that are likely to be relevant to an astronomer processing data. To use these Volumes effectively, you should have some familiarity with Glish, and you should understand the basic AIPS++ concepts regarding `tools` and how to use them.

The Cookbook comes in three volumes.

Volume 1 - Basic Tools gives examples of how to use some generic AIPS++ capability such as tables, plotting, image manipluation and data display.

Volume 2 - Generic Processing describes generic astronomical data processing.

Volume 3 - Telescope Specific Processing describes telescope-specific data processing.

The Cookbook volumes show you how to do *some* things. But for all the details of each `tool` that you might find in these examples, you will ultimately need to refer to the User Reference Manual.

3.2.2 Recipes

The Recipes Repository provides *Glish* scripts of limited scope which perform or demonstrate a useful operation.

3.2.3 News Letters

Another source of high-level information is in the series of News Letters (now terminated). These are aimed at describing some interesting piece of functionality for astronomers.

3.3 For Detailed Information

3.3.1 Glish

There is a comprehensive Glish Manual that you should use to gain a detailed understanding of *Glish*. It is a reference manual. Therefore if there is something you want to do in *Glish*, but don't know the syntax for, this manual is the place to look.

3.3.2 User Reference Manual

The ultimate repository of information about AIPS++ `tools` is the User Reference Manual. This manual does not generally offer you astronomical context (this is the job of the CookBook). It is truly a Reference Manual.

To use the URM effectively, you should be reasonably experienced with Glish and you *must* understand the basic AIPS++ concepts regarding `tools` and how to use them.

It is the URM that will enable you to write your own Glish scripts and create your own `tools` using other AIPS++ `tools`.

This manual is structured hierarchically by Package, Module and Tool. Each Package is targeted at a broad functional area such as Synthesis processing. Within each package there are a number of Modules, breaking the package structure into smaller components. Finally, each Module contains a number of `tools` and (global) functions.

Each Module has "Module" documentation to a varying degree. This documentation is expected to offer you information that is relevant to all of the `tools` in the module, and that which unifies those `tools`. There are usually some examples in the Module documentation.

Each `tool` is documented in the same structure. There will be some general discussion about what the `tool` is for and what its capabilties are and some examples of how to use it. Then each `tool` constructor and function is documented. These should describe what every argument to every constructor or function does. There should be some examples as well.

The quality of the User Reference Manual is uneven. There are some well-documented Modules and `tools` and some poorly documented ones.

3.4 For Further Detailed Technical Information

Other technical information can be found in the series of Notes. These Notes cover a wide range of interests from software techniques to algorithms. They are not aimed at the casual user.

4 Essential Concepts and Terminology

In any software package there are concepts and language that you must learn to be able to use the package effectively. This is especially true for AIPS++ because there are some *new* concepts compared to other packages that many of you might already be familiar with (e.g. AIPS, Miriad, GIPSY etc).

It is essential that you understand these concepts and terminology so that you can read the documentation and use AIPS++ effectively. In this section we will talk generally about these concepts. In the following section, we will show some examples of how to actually do things in AIPS++ with these concepts from the *Glish* command line.

4.1 What is a Tool ?

We use the word `tool` in AIPS++, and it has a meaning which is analogous to the "tools" we are all familiar with in our usual computing environment. A browser is a tool; you can start several instances of it, it has an interface (often GUI-based) and that gives you control over what the tool can do (e.g. format an HTML page, pull down a menu and select something). I.e. it is software 'program' which has some 'functions' to do things.

In AIPS++, a `tool` is something very similar; it is a collection of related functions (`tool` functions) that generally operate on some common data source. The range of things that can be done with each tool is always focussed. By analogy, a browser provides ways of formatting mark-up languages (e.g. HTML) but you don't use it to do image manipulation.

`tools` are "constructed" (or created) when you need one. In our browser analogy, your browser tool is constructed/created when you type in the program name at the Unix command line.

```
% mozilla
```

.

This creates (constructs) the browser process or tool (and you can create as many as you like) and gives you access to its functions through its GUI.

As well as `tools` and `tool` functions, in AIPS++ we also have a number of global functions (or just functions). These functions stand on their own and are not tied to a specific `tool` (see examples in followin section).

For those of you who are familiar with Objected Oriented programming, a `tool` is just an object and it has `tool` functions (methods) that you can execute. This Object Oriented interface is exposed to users at the command line.

Let us give a few examples of AIPS++ `tools`.

Image An Image `tool` is a `tool` which has functions which process image data. Now note an important thing. There is not just **one** Image `tool` which is used to process all the images you ever make in AIPS++; if you wish to access an AIPS++ image disk file, then you create (or "construct", as we generally say) an Image `tool` that is associated with that image file. Then all the functions of that Image `tool` operate on that data file (e.g. display it, compute statistics, regrid it etc). An Image `tool` cannot exist without a data source.

You will generally make quite a lot of Image `tools` in the course of processing images in AIPS++.

Imager An Imager `tool` is a `tool` which is used to make images from visibility data. Its data source is then a visibility disk file (a thing we call a Measurement Set) with which you construct an Imager `tool`. It has lots of functions which then process the visibilities from those data to make images. An Imager `tool` cannot exist without a data source. Like Image `tools`, you may construct more than one Imager `tool`, but generally you will only have a small number of them.

Quanta A Quanta `tool` has functions which manipulate values and units. For example, convert 5km/s to pc/h, or add 20km to 3cm and give me the answer in furlongs. A Quanta `tool` can exist without a data source. For this reason, the AIPS++ system provides one pre-made Quanta `tool` for you, and it's the only one you ever need to use. If you really want, you can make your own, but there is generally no point to doing so.

So some `tools` are associated with a data source, and some aren't. Finally, when you construct a `tool`, there may be more than one way to do it. For example, you can construct an Image tool in many ways. Here are three ways (of many).

- By converting a FITS file to an AIPS++ file
- By directly associating with an AIPS++ image disk file.
- By creating an empty AIPS++ image disk file from a specified shape and coordinate system.

4.2 Destroying Tools

When you construct a `tool`, it takes up some computer resources (e.g. memory). Because your `tool` may be associated with a data file, it may also have "locked" that data file so that other `tools` can't access it (e.g. if your `tool` is modifying it.

So it is very important that when you are finished with your `tool` that you destroy it. If you don't do this, its resources will not be freed up and you may leave files locked.

A thing you should **never** do is overwrite a `tool` without first destroying it. You will get in big trouble if you do this.

Every `tool` has a function called `done` which will destroy the tool, free up its resources and remove all locks. Note that this does not delete any underlying disk data files. There are examples in the following section.

4.3 Are Tools the same as Tasks ?

Astronomy packages such as Miriad and AIPS are not object oriented; they are process oriented. This means that there is a process, usually called a "Task" that has some particular functionality. For example, in Miriad there is a Task called "histo" which computes statistics from an image. The user supplies it with inputs such as the input image file name and executes (runs) the task. With AIPS++, the same functionality is achieved by constructing (creating) an Image `tool` from the image data file and then executing the `statistics` function.

4.4 Terminology Summary

In the above description there are just a few pieces of terminology that you will see over and over. Let us summarise them

- Tool - A software process which has a collection of related functions (usually) operating on a common data source.
- Constructor - a way in which a `tool` is constructed (created). Each `tool` type might have more than one constructor.
- Tool function - Allows you to execute specific functionality of a `tool`. Each `tool` will typically have many functions for you to execute.
- Global function - A function which is *not* tied to a specific tool

5 More Details about Tools and functions using the Command Line

In the previous section we talked in general terms about `tools` and `tool` functions. Now we show some examples using the *Glish* command line.

5.1 Loading the tool

Before you try to create a particular `tool`, you need to tell AIPS++ to load the commands that define the tool. For example, if you wish to create an Image tool, you need to first tell Glish about Image tools. Here are some examples of the most fundamental and useful tools in AIPS++.

```
% glish
- include 'table.g'                    # Access Table tools
- include 'quanta.g'                   # Access Quanta tools
- include 'measures.g'                 # Access Measures tools
- include 'image.g'                    # Access Image tools
- include 'pgplotter.g'                # Access PgPlotter tools
- include 'catalog.g'                  # Access Catalog tools
- include 'imager.g'                   # Access Imager tools
- include 'ms.g'                       # Access MS tools
- include 'toolmanager.g'              # Access the toolmanager tool
```

You might ask why don't we just load all of those tool scripts so you do don't have to bother about it ? We can and we do. If you type

```
% aips++
```

that's exactly what will happen. However, it all takes time to load these scripts into the computers memory. It is often more effective for you to just load the commands relevant to the `tool` you wish to work with. When writing scripts, this always how we use the system.

In the examples below, we assume you have already loaded the appropriate scripts. All the ones you need are named in the examples above.

5.2 Types of constructors

In general `tools` are made using their corresponding `constructors`. Generally, a `constructor` goes by the descriptive name of the `tool`.

Making an imager, an ms, or an image `tool`

```
imgr:= imager('filename of an MS')
include 'ms.g'
img := image('filename of an image')
```

The variable on the left-hand side can have any name you choose.

The catalog and toolmanager `tools` can also construct a tool from a table name and an appropriate constructor (that is chosen for you).

Making an imager, an ms, or an image `tool` using the catalog or toolmanager

```
tm.tool('filename of an MS')
dc.tool('filename of an MS')
tm.tool('filename of an image')
dc.tool('filename of an image')
```

Some `constructors` create a `tool` after performing some operation such as format conversion. For example, the `constructor fitstoms` converts a FITS data file to an MeasurementSet, and creates an ms `tool`.

Converting a FITS file to a MeasurementSet

```
myms:=fitstoms('output MS file name', 'input FITS file name')
```

Other examples of these types of `constructors` are `imagefromfits` and `imagefromarray`.

Finally, Some `constructors` do not need an argument. Examples are

Constructors without arguments

```
cattool:=catalog()         # create a Catalog tool
myplotter:= pgplotter()    # create a Pgplotter tool
```

The list of `tools` that are constructed and active are displayed on the Tool Manager GUI in the Tools in Use view. Once a `tool` is created, its various `functions` which are meant for different operations can be used.

Constructing a Table `tool`

```
mytable:= table('3C273XC1.ms')
```

`table` is a general `tool` for operations on various types of AIPS++ Tables. This `tool` is not something that you may normally use in standard data processing but it can be very useful if you wish to do something out of the ordinary. The left-hand side of this expression, `mytable`, is the name of the Table `tool` you created. You can choose any name you like. The right-hand side is the `constructor` which takes a file name, in this case *3C273XC1.ms*, as an argument and 'constructs' a `tool`. If you click Refresh in the Tool Manger GUI, the name of the newly constructed `tool` now appears in the list of Tools in Use. This `tool` is now specifically meant for operations on the MeasurementSet *3C273XC1.ms*. You can create another Table `tool` (with a different name) for another MeasurementSet or some other AIPS++ Table.

5.3 Using `Tool Functions` from the Command Line window

The general syntax for using a `tool function` is

General syntax for tool functions

```
[output:=]tool.func(arg)
```

where `tool` is the name of the actual instance of the `tool` (*e.g.*, (`mytable`), `func` is the name of the `function` (*e.g.*, `getcol`), and `arg` are the arguments of the `function`. When no arguments are needed then the expression ends with (). The left-hand side is sometimes needed if the `function` produces an output which may have to go into a *Glish* variable.

Each `tool` may have several `functions`:

Some functions of a Table tool

```
summary         # get a summary of the contents of the Table
colnames        # return the names of columns of the Table
getcol          # get values of a specific column in to a glish variable
getcoldesc      # get the description of a column
getkeywords     # get all the keywords of the Table
ncols           # return number of columns
nrows           # return number of rows
name            # return the name of the Table on the dis
```

Execution any of these `functions` to `mytable` is done in the following way.

Executing tool functions

```
mytable.summary()   # watch the summary appear on the Logger window
mytable.colnames()  # column names appear on the Command Line window
a:= mytable.getcol('ANTENNA2') # column names now assigned to a
```

In the last example, the *Glish* array `a` now contains all the values of the column **ANTENNA2**. Note that the array size of `a` is equal to the number of rows, which was automatically set by the *Glish* statement above. You can check the shape of an array using the `::shape` attribute.

Check array size or shape

```
a::shape
b:=mytable.getcol('UVW')
b::shape
```

You have just come across one of the important features of *Glish*, namely automatic type assignment. The type of the variable (integer, floating, scalar, vector) is assigned automatically as it is created; therefore, you do not need to declare its type. There are many other nice features in *Glish* which you can learn by reading the document by Rick Fisher entitled *Getting Started with Glish.*

Destroy a Table tool

GUI Select `mytable` under the Tools in Use view column and click `Delete`

CLI
```
mytable.close(); # If mytable still open
tabledelete('3C273XC1.ms')
```

Note that in creating the `tool`, `mytable`, you did not make any copy of the MS (i.e., the original Table) *3C273XC1.ms*. By making a Table `tool` with the MS attached to it, you are able to gain access to various parts of the MS and use the `functions` of the Table `tool` to operate on them. This is the general way in which the various `tools` and `tool functions` are used in AIPS++. The above example is actually a special one, chosen for demonstrating access to the inside of an MS. One of the things you can do with an MS is create an Imager `tool`, which has various `functions` to make images and deconvolve them. You can also create a Calibrater `tool` with an MS that has `functions` for calibration of the UV data.

Below are some more examples of using `tool functions` from the Command Line window. The commands below were not designed to try out in this session. They are only examples.

More table toolfunctions

```
myms.summary()                # give the summary of myms
myms.range('TIME UVDIST')     # get the ranges of TIME and UVDIST
aa:=myms.range('TIME UVDIST'') # get the ranges into a record aa
```

The ranges are now available in the record `aa` as record members `aa.time`, `aa.uvdis`.

Setting imaging parameters

```
myimgr.setimage(cellx='5arcsec', celly='5arcsec', nx=256, ny=256,
    stokes = 'I')
```

Note that other arguments of `setimage` such as `doshift`, `phasecenter` take on default values. It is also possible to use outputs of other `functions` as arguments where applicable.

Various ways of setting the imager phasecenter

```
imgr.setimage(phasecenter=
              image('myother.image').coordmeasures().direction)
imgr.setimage(phasecenter=
              dm.direction('j2000', '05h30m','-30.2deg'))
```

5.4 More Examples of `tools` and `functions`

Before leaving the subject of `tools`, make one more `tool` which is for a general utility and use some of its `functions`.

Creating a pgplotter tool

GUI Select `display.plotter.pgplotter` and press Create. Then in the constructor view, leave the name as `myplotter` and just press Create to construct the `pgplotter` `tool`.

CLI Create a `pgplotter` `tool` by typing on the Command Line window:

```
include 'pgplotter.g'
myplotter:= pgplotter()
```

Note that this tool constructor can be used with no argument. It brings up a `pgplotter` GUI like the one you saw before while executing some functions earlier. The Tool Manager Tools in Use view now includes **myplotter**. The `pgplotter` GUI is shown in Figure 11. You can now use this plotter to make any plot you want using the various `functions` of the `pgplotter` `tool`.

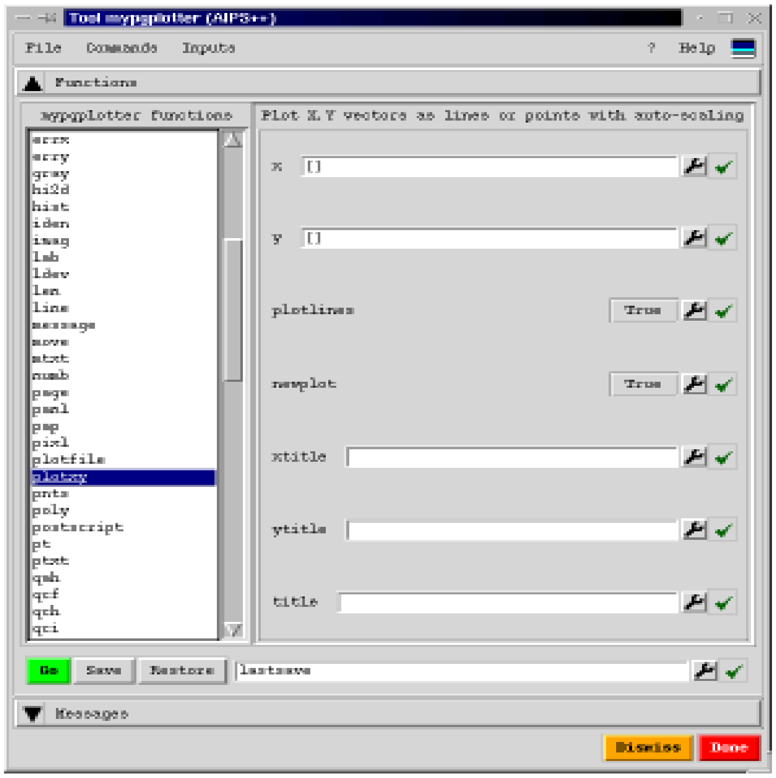

Figure 1: `pgplotter` GUI

Now let us use the GUI or the CLI to plot some points. First make two arrays `x` and `y`, with `x` having values from 1 to 10 and `y = x*x`, then plot `y` against `x`.

Plotting x vs y

GUI
- Select the `tool function plotxy`
- In `x`, type `1:10`
- In `y`, type `(1:10)*(1:10)`
- Press the Go button

CLI
```
x:=1:10
y:=x*x
print x, y
myplotter.plotxy(x,y)
```

`plotxy` is one of the `functions` of the `pgplotter` `tool`. For those of you who use PGPLOT a lot, you will be pleased to know that the `pgplotter` `tool` allows you to do all the things you can do with PGPLOT from the Command Line window, or through a *Glish* script of the various PGPLOT-type commands, or through the pgplotter GUI.

Label the axes and write some text on the plot

```
myplotter.lab('Xaxis label', 'Yaxis label', 'Top header')
myplotter.text(2,80,'Plot of X vs Y')
```

You can now dismiss the `pgplotter` GUI by clicking the Dismiss button.

Notice that the `pgplotter` GUI has options under the menu items *File, Tools and Edit*, These options allow you to save or print the plot, to add other items to the plot, to zoom. The `pgplotter` `tool` is useful for making your own plots or customizing plots that AIPS++ has made.

5.5 Destroying `tools`

We discussed in the previous chapter how it is very important to destroy tools when you are finished with them. This is because they take up some computer resources (e.g. memory) and may lock data files. Let's see some examples now of how to do this with the `done` function, which every `tool` should have.

When you destroy a `tool`, it does not delete any disk-based data file that it may be associated with.

Destroy a tool correctly

```
im1 := imagemaketestimage('zz')        # Create Image tool and write image into disk file 'zz'
im1.statistics()                       # Do statistics
im1.done()                             # Destroy tool
```

It is a very bad thing to overwrite a `tool`. You should never do this. It will leave resources un-freeable, leave data files locked and possibly cause AIPS++ to crash.

Overwrite a tool - BAD BAD BAD

```
im2 := image('zz')                       # Create Image tool from disk file
im2.statistics()                         # Do statistics
im2 := imagemaketestimage()              # Overwrite tool; BAD BAD BAD BAD
im2.done()                               # Try to destroy tool; will not work correctly
```

To deal with this situation correctly, we should have done

Destroy tool before reusing

```
im2 := image('zz')                       # Create Image tool from disk file
im2.statistics()                         # Do statistics
im2.done()                               # Destroy tool
im2 := imagemaketestimage()              # Construct a new tool (with the same name)
```

5.6 Global Functions

In all the examples discussed above, the `functions` which actually do the job are attached to a `tool`. To use the `function`, the `tool` to which it is attached need to be created first. As mentioned earlier, there are some useful `functions` which are not attached to any `tool`. These are known as `global functions` and they can be activated at any point - usually from the Command Line window. Although `global functions` may look different, they do belong to the AIPS++ hierarchy in the sequence packages.modules.`global functions`. Thus, the access to `global functions` through the Tool Manager GUI is by clicking the appropriate package and module. If the Run button below the column **global functions** in the Tool Manager GUI is clicked, then it will, in most case, open up a GUI from which the `global function` can be activated similar to the `tool functions` described earlier.

Examples of global functions

```
cat             # list the files and Tables in the Catalog
icat            # list images in the Catalog
tablecopy       # make a copy of a Table
```

```
tabledelete     # delete a Table
mean            # compute the mean of a set of number
moments         # compute the moments of a set of numbers
gaussian2d      # compute a 2-dimensional gaussia
```

Using `global functions` is very similar to using `tool functions`. The difference is that `global functions` do not require a `create` operation before they can be used. These `functions` are global to AIPS++ and they can be entered at any point so long as the relevant packages or modules are loaded using the `include` command. Some `global functions` have optional arguments and produce output records, plots, or GUIs. Currently, most `global functions` can be used from the Tool Manager window.

Using `global functions` from the CLI

```
include 'statistics.g'          # load the statistics functions
mean(1:100)                     # mean of numbers 1 to 100
x:=[1,3,7,8:15,19,23]           # vector assignment
mean(x)                         # mean of x
note('This is a note in the Logger file') # Note to the logger
help('moments')                 # help on moments
web()                           # Drive the Web Browser to
                                # the last requested help
moments(3,1:20)                 # compute up to 3rd moment of
                                  numbers 1 to 20
```

Use a gaussian1d function and plot it

```
include 'gaussian.g'
x:=1:100
y:=gaussian1d(x, height=5, center=40, fwhm=20)
include 'pgplotter.g'
mypgplotter:=pgplotter();
mypgplotter.gui()
mypgplotter.plotxy(x,y)
```

The plot, as shown in Figure 12, appears on your screen.

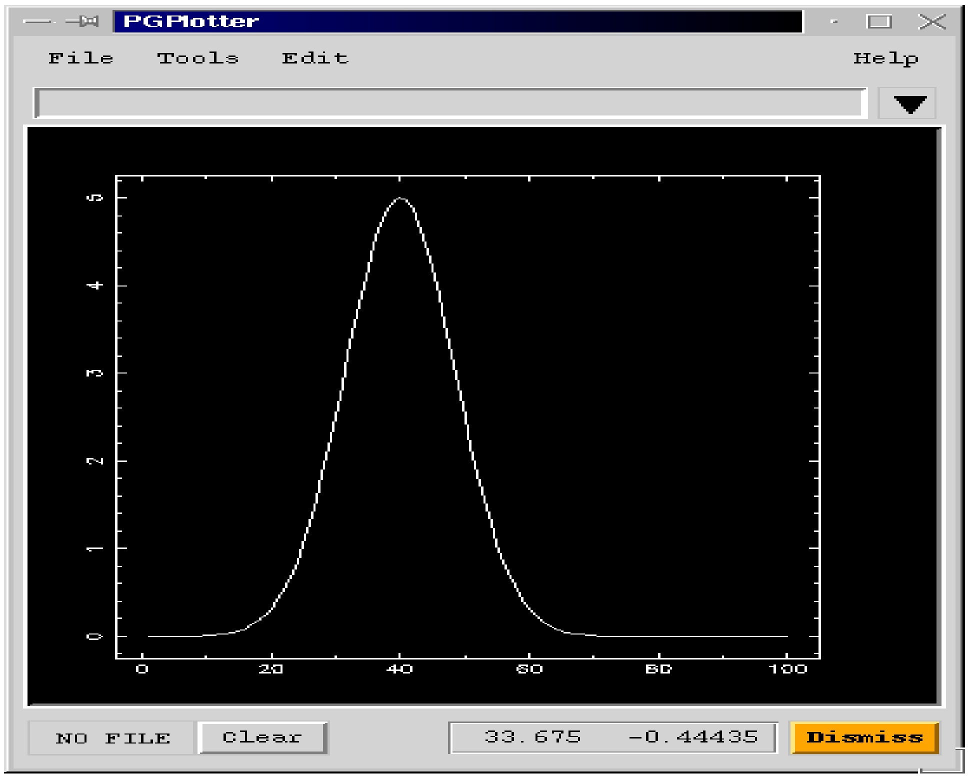

Figure 2: Gaussian 1D Plot

6 Data and Images in AIPS++

In AIPS++, all forms of data (*i.e.* visibility, images, spectra, history, header information, logs) are stored in Tables. Tables consist of information arranged in rows and columns. The columns have names (e.g. **UVW, TIME, ANTENNA1, SIGMA**) and the rows are numbered. Keywords may be attached to both the Table as a whole or to specific columns and are used to specify additional information. Examples of keywords are revision number of a Table, author of the Table, units in a column, coordinate information for a column and so on. Keywords may also be used to store other Tables.

6.1 Tables

A Table and its contents are stored on disk as a directory with the Table name (*e.g. 3C273XC1.ms*) and may contain a number of files. One technical aspect of Tables that you should be aware of is that some Tables reference other Tables. We try to avoid making copies of Tables so that if two Tables contain almost the same data except for one or two columns, then a reference Table is created that points back to the original Table for shared information. This mechanism is also used for sorts and selections. If you sort or select a Table to create another Table then the new Table points back to the original Table via a column of row numbers. The only reason you need to be aware of this is to avoid deleting the original Table.

Although there can be many different types of Tables in AIPS++, representing different types of data, there are two important types of Tables which you will commonly use: MeasurementSet (MS) (which represents data from a telescope, *e.g.* visibility data) and Image (which represents the distributions of intensity on the sky). An Image may have two or more dimensions, *e.g.* position, velocity, stokes.

6.2 MeasurementSet (MS)

A MeasurementSet(ms) is simply a Table obeying certain conventions that allow it to represent observations from many different types of radio telescopes. In addition to a number of data columns, it also contains sub-Tables stored as Table keywords (as discussed above). These keywords contain information on the antennas, feeds, fields, and spectral windows. An MS can be dynamic in that new columns can be written into it as data are processed. For example, the original measured UV data appears in the data column and subsequent processing can produce additional columns containing corrected data (for e.g. after a Selfcal) or Model data (a set of clean or other components)

6.3 Image

An Image is also a Table. The keywords in this Table contain information on units and coordinates. The pixels are stored as a single cell in a column. Other information about the image (e.g. its co-ordinates, origin, history, characteristics) are stored under separate keywords.

7 Running AIPS++

In this section, we describe how to set up your environment to run AIPS++. This is quite straightforward and only requires a few steps.

7.1 Initializing your environment to run AIPS++

Because AIPS++ is run from a Command Line window interpreter/control hub, before starting AIPS++, you are going to need to load some environment variables. This is the most site specific part of using AIPS++. You need to know where your local installations are kept, and what the update policy is. We keep four copies at NRAO-SOC and NRAO-CV, they are `/aips++/release`, `/aips++/stable`, `/aips++/weekly`, and `/aips++/daily`. The first (`release`) is the production version that corresponds to the last formal release, and the second (`stable`) is a regularly updated snapshot of the current development system. If possible, you should use release or stable. The latter two installations (`weekly` and (`daily`) are developmental copies and should not be used unless you need to use something specific that is available only there. Of these two, `weekly` is more stable, but `daily` gets bug fixes and new features more quickly. `Daily` is recompiled often, so you may see strange results if the executable you need is either not available or is deleted prior to a re-compilation.

Thus, if the installation you need is kept in *e.g.*, `/aips++/release`

Initializing your shell to run AIPS++

```
. /aips++/release/aipsinit.sh       # Bourne/Korn/Bash
source /aips++/release/aipsinit.csh # csh and similar shells
```

7.2 Creating working space

We recommend that you create a separate directory on your disk with plenty of space to work in AIPS++. Unlike AIPS, AIPS++ does not have its own catalog and instead relies upon the file system of your computer to manage the files. Although it is possible to specify full directory paths for file names that need to be read or written by AIPS++, it is most convenient to work in the directory containing the files on which you wish to work. Thus you should create a separate directory to run AIPS++ and move to that directory.

Create a separate directory to run AIPS++

```
mkdir new_directory_name
cd new_directory_name
```

7.3 Setting X-windows to run AIPS++

AIPS++ can be run from a simple character-based terminal, but you will get better results running it with a windowing interface.

Setting your DISPLAY environment

```
export DISPLAY; DISPLAY=:0.0       # Bourne/Korn/Bash shell:
setenv DISPLAY :0.0                # csh shell
```

7.4 Running AIPS++

To run AIPS++:

Starting AIPS++

```
aips++
```

This does three things: Starts *Glish*, opens a Logger window where messages are displayed, and opens the Tool Manager GUI. The very first time that you start AIPS++, it may take some time to start up, as some initializing files are being created. Subsequent start ups will be much faster.

Before proceeding any further, there are three things you absolutely must know:

Essential commands

```
exit    # Exit from Glish
bug()   # Report a bug
ask()   # Ask the local contact or AIPS++ Project Staff
help()  # Get help
```

Control-c and Control-z will abort and suspend *Glish*, respectively.

7.5 Getting Help

In AIPS++, help is available at various stages, both from the *Glish* Command Line window and from the GUIs for various applications. Reference material for most things are usually presented through your Web browser.

At the *Glish* Command Line window, help can be obtained by typing:

Get some help on a module, `tool`, or `function`

```
help('general')       # modules in the general package
help('*name*')        # wild cards are allowed
help('catalog')       # tools and functions of the catalog
help('catalog.show')  # function show of the Catalog
help('*cat*')         # all occurrences of 'cat' in help
```

You can also get more detailed information on a specific subject by using your browser.

Help via your Web browser

```
web()           # Drives to last requested help section
help('Refman:') # Drives to the Refman
help('Glish:')  # Drives to the Glish User Manual
```

When using GUIs, some help is available for each of the buttons. If you position your mouse pointer over a given button for about a second, a brief help pops up. The GUIs also have Help or Web buttons which steer your Web browser to the appropriate section or provide some additional help. The Help button is usually located at the top right corner of the GUIs.

7.6 Fine-tuning up your environment

AIPS++ uses a file called `.aipsrc` to control how it works. To create a version of this file, or to edit existing values, use the tool `setupwizard`.

Setting up the AIPS++ environment

```
include 'setupwizard.g'
setupwizard()
```

This will start a gui-based wizard for setting values in the `.aipsrc` file. If you don't have a GUI available for the wizard, then you can create the `.aipsrc` file by hand. See Appendix B for more details.

7.7 Brief overview of AIPS++ windows and GUIs

After starting AIPS++, the window from which you executed the `aips++` command is now running the *Glish* Command Line window. *Glish* is alive and well when a '-' (or sometimes a '+') prompt appears on the screen. You use the *Glish* Command Line window to type commands. At this point rearrange the three windows (terminal, Logger GUI, and Tool Manager GUI) any way that suits your convenience.

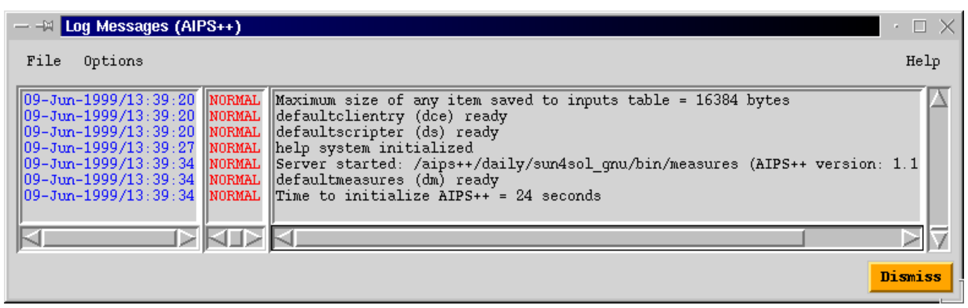

Figure 3: The Logger window

The Logger window shown in Figure 1, is something you may watch as action takes place in response to your commands. Most actions are reported to the logger, with only a few exceptions, such as (*e.g.*, *Help*) that are reported to the *Glish* CLI window. The Logger window has three frames. From left to right, they are the **time**, the **priority**, and the **message**. In addition, a fourth frame may be activated which displays the module from which the message arose. These frames are toggled on and off from the *Options* menu in the window. The *File* menu allows you to:

query Select a subset of the messages using a query. You will be prompted to fill in the details of the query.

purge clear the log messages,

print make a hardcopy of the log messages (you are prompted for the number of messages to print),

script write commands in the log to a script

Notice that some messages have already been logged in the Logger window after you started AIPS++. These messages tell you that several default `tools` have already been loaded (*e.g.*, `viewer tool 'dv'`, `catalog tool 'dc'`).

The Tool Manager GUI (Figure 2) is central to AIPS++. It is used to control AIPS++ as well as to obtain help on various `tools` and functions in AIPS++. It is not desirable to resize this (or other AIPS++) windows using the window manager on your workstation. If you do so, you may get strange results in subsequent operations.

8 Use of the Tool Manager

The Tool Manager is key to the interactive GUI-based use of AIPS++. To use AIPS++, it helps to have mastered the use of the Tool Manager. For this reason, we suggest that you work through this section especially carefully, performing the suggested examples as you go along.

8.1 Views in the Tool Manager

Views of various aspects of AIPS++ can selected using the tab buttons just underneath the top menu. The three principal views are:

Manager For traversing the hierachy of `tools`, and for creating new instances of `tools`. This view (and the associated tab) is always available.

Tools in Use For interacting with existing `tools`. This view (and the associated tab) is always available.

Tools, Constructors, or Global Functions Each tool, constructor, or global function has a view dedicated to it. These are added and removed from the set of tabs as appropriate.

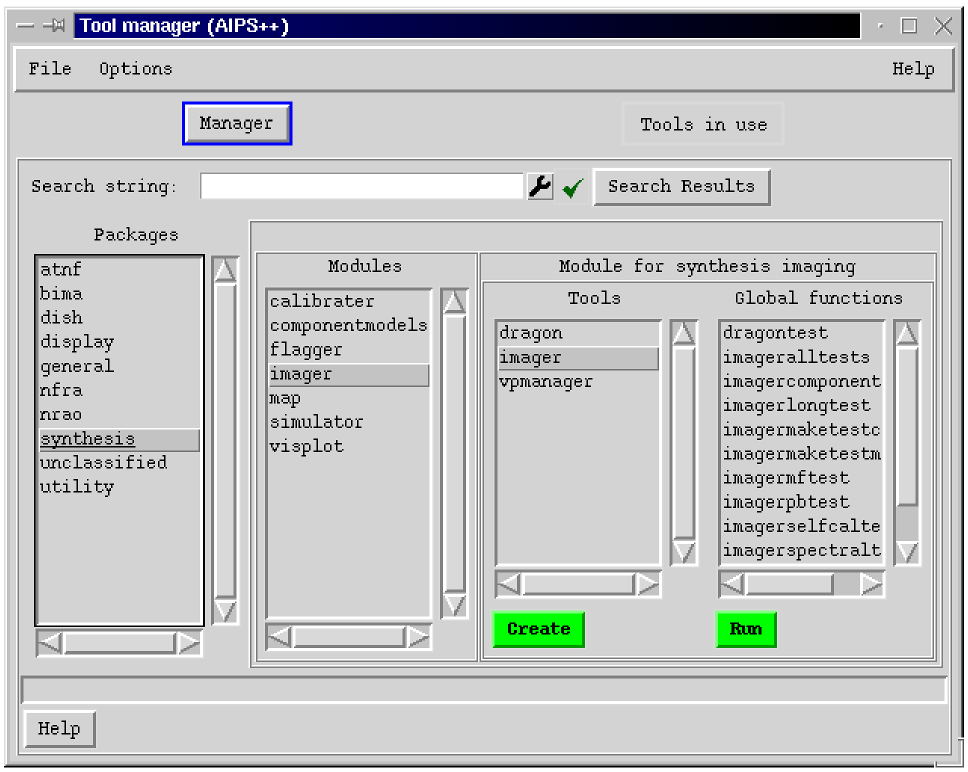

Figure 4: The Tool Manager GUI as it appears when AIPS++ is started. This is the Manager view.

As described above, the various `tools` and `functions` in AIPS++ are arranged in a modular, hierarchical fashion. The overall structure of AIPS++ from a user point of view can be seen using this view of the Tool Manager GUI.

Under the Tools in Use view, if you press the button right button to "Show all tools", you will see a list of `tools` with names like **dc** (default

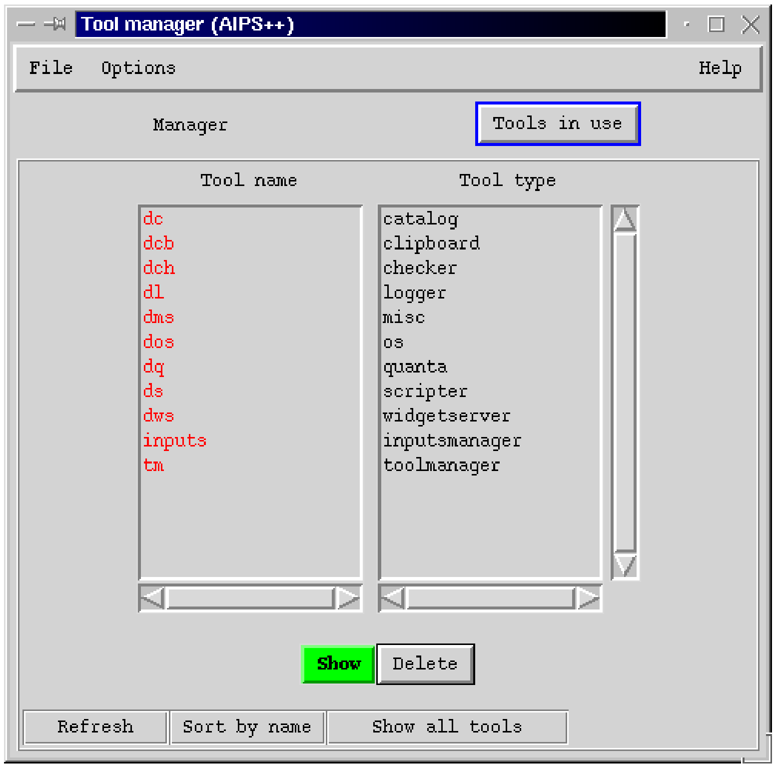

Figure 5: The Tool Manager GUI with the Tools in Use view displayed

catalog), **dv** (default viewer), **dl** (default logger), **dm** (default measures). `dm` is an innovative `tool` which performs varieties of operations related to co-ordinates, time, quantities, and conversions. Any of these `tools` can be activated by clicking the required `tool`name and then clicking the Show button in the same window. To see how this works, notice the tool `dl` (Logger) which was already activated by default. Click the Dismiss button on the Logger window. The Logger window is removed. Now select `dl` from list of Tools in Use and click Show. The Logger window reappears. This is the standard way in which the GUIs (Graphic User Interfaces) of `tools` that are not in use can be removed (it helps to reduce the clutter on your screen) and brought back when needed. Note that at the bottom of this view there is a line of buttons. These control which tools are shown, how they are listed, and how often the list is updated.

Now switch back to the Manager view. The top layer of AIPS++, packages, is listed in the left most column. These are known as *packages*. Currently there are nine packages in AIPS++ (`general, utility, display, synthesis, dish, BIMA, NRAO, NFRA, ATNF`). In addition, you will see an entry `unclassified` when a classification is not possible.

Each package contains one or more *modules*. The list of modules in each package can be seen by clicking on the name of the package in column 1 of the Tool Manager GUI. The list of modules appears in Column 2.

For instance, click on the package `synthesis`. The following list of modules appears in the second column:

`calibrater` : module for calibrating UV data

`componentmodels` : module for making source components for use in deconvolution or self-calibration

`imager` : module for most imaging needs

`simulator` : module for simulating UV and other data

`msplot` : module for plotting Visibility data in various ways

Each module can contain *tools* and *global functions*:

`tools` are collections of related functions that operate on some common data source. To use the `functions` of a `tool`, the `tool` needs to be connected to the data source. This connection is usually called "creation" or "construction", and is performed using special functions called `constructors`. Each `tool` may have one or more types of `constructors`. Once a `tool` has been constructed, the user can use functions of the `tool` to perform the various operations needed for data processing.

`Global functions` perform single straightforward operations that cannot be naturally grouped. Invocation of the global function is very straightforward, involving just the arguments of the function.

As an example, the Mathematics module in the `utility` package contains many `tools` (including, fftserver, matrix, polyfitter) and `global functions` (including, gauss1dfitter, mean, moments).

`Global functions, tool constructors`, and `tool functions`perform the actual operations associated with data processing. These `functions` and `constructors` have various input arguments (with sensible defaults) and optional outputs. These will be shown once you press the Create or Run buttons.

Finally, the Tool Manager GUI contains a Search entry on the top. This entry can be used to search for the occurrence of any string of characters in the entire AIPS++ system. There is one entry **Search String** and a button to the right to display the results of the search. The search option is very useful for locating the appropriate modules or `functions` for a given operation for which you may only know a keyword.

- For instance, type `advise` in the **Search String** window and press return. The results can be seen by pressing the **Search Results** button.
- You get the answer `synthesis.imager.imager.advise`
- This implies, that `advise` is a `function` of the `tool` imager which is contained in the module `imager` of the package *synthesis*.
- The general syntax for representing an "entity" in the AIPS++ hierarchy is thus
  - `package.module.tool` or
  - `package.module.tool.function` or
  - `package.module.tool.constructor` or
  - `package.module.global function`
- As a further experiment, type `imager` for the search string and see the various occurrence of this string displayed in the window. Notice that the same name (*e.g.* imager) is used for a module, a `tool` and a `constructor`. There is no clash in this usage.

8.2 An example of the use of the Tool Manager

The concepts introduced above should become more understandable once actually do something, so let's make an `imager tool`. `Tools` can be constructed from the Tool Manager GUI by selecting the `tool` name and then clicking the Create button. This operation generates a view of the `tool` constructor with various arguments.

Making and using an `imager tool`

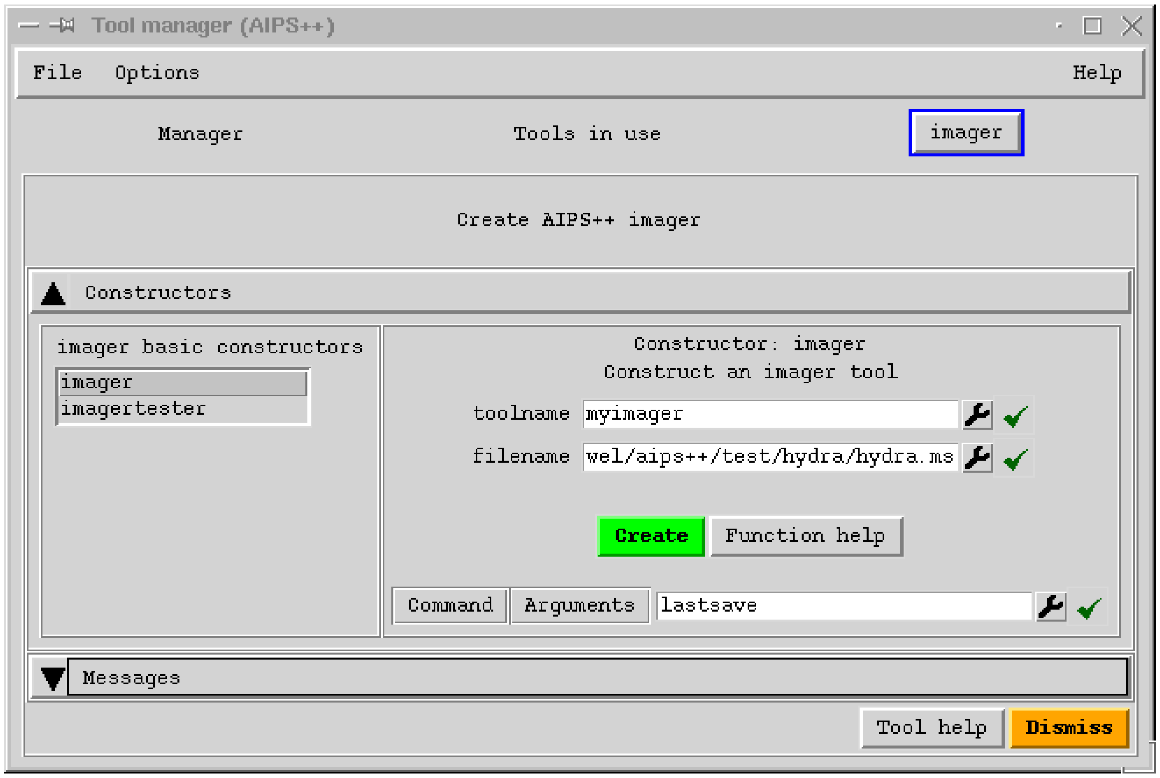

Figure 6: Tool Manager view of the imager constructor

- Click the Manager tab on the Tool Manager GUI. Then Select the `imager tool` by sequentially clicking **`synthesis`** under *package*, `imager` under *module* and finally `imager` under *Tools*.
- Click Create. A Constructor GUI appears as shown in Figure 4. On the left hand size this GUI shows that there are two imager `constructors`, called imager and imagertester.
- Click `imagertester` under the list of *imager constructors*. The GUI displays the inputs to this `constructor`. This constructor creates a standard MeasurementSet and connects the imager tool to it. All the default arguments will work just fine for this demonstration. (Note that the other constructor takes the name of a MeasurementSet and so if you have one, you could use that constructor instead. Both constructors lead to an `imager` tool that is used in exactly the same way.)

- type in the `tool` name you want, by placing the cursor on that window. (or just leave it at its default `myimager`)
- Press the **Create** button. The imager constructor then starts to run. Afte 10-15 seconds, the view of the constructor will disappear and it will be replaced by a view of the tool functions (figure 5). The tabs at the tool will be updated.

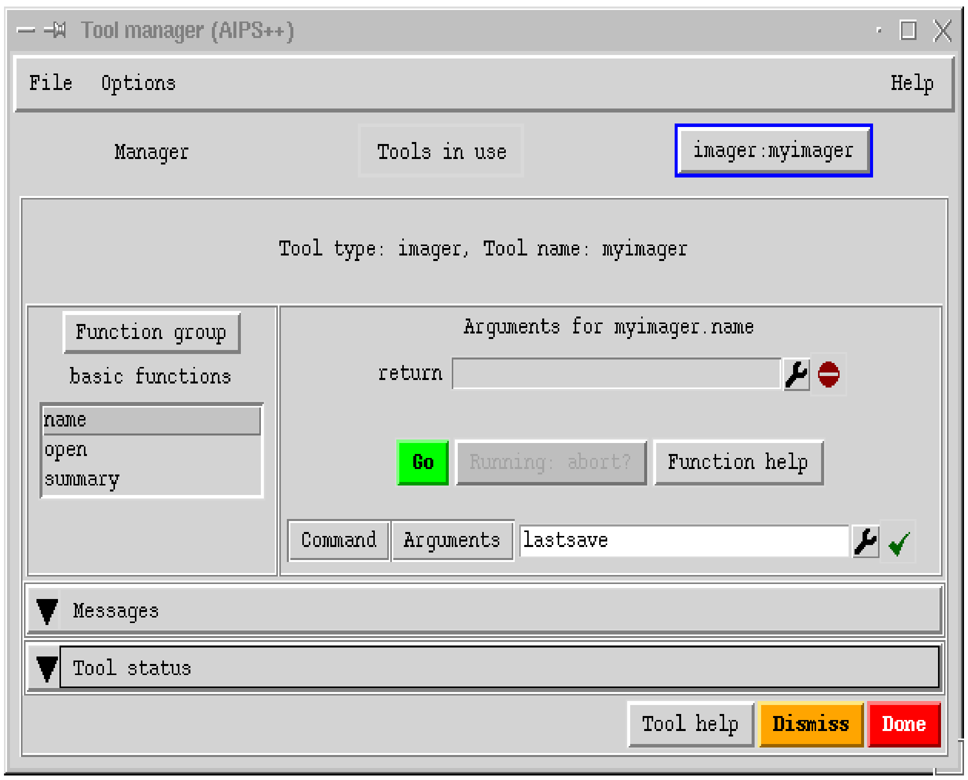

Figure 7: Tool manager view of an imager tool

On the left of the view, you will see a scrollable list of the functions of the tool. Selecting one of these functions causes the corresponding tool function arguments to be shown on the right hand side. Input arguments have a white background, output arguments have a grey background and cannot be edited.

For some complicated tools, such as imager, the functions may be grouped. The particular group may be selected using the button labelled 'Function groups' at the top of the list of functions. If you do not find this helpful, you may access all functions at once using the group 'all'.

To use a specific `function` of the imager `tool`, do:

Using a `tool function`

---

1. If present, choose the required group of functions by clicking on the 'Function group' button.
2. Choose the required `tool function` by clicking it. This operation gives a one line description of the `tool function` and a list of various input arguments. At any point, you may press the **web** button to get detailed information on that function.
3. Enter the various input arguments. Some `tool functions` may have no arguments. Short popup help on each argument is available by placing the mouse pointer on the argument name. More detailed help can send to your browser using the **web** button.
4. Click Go

Try this now for the `advise` `tool function`found in the helpers function group. This `tool function`calculates maximum cell sizes for the selected data in the associated MeasurementSet. You can also set a field of view to get a recommended number of pixels.

Another good `tool function` to try is `plotvis` to plot the visibility data.

At this point you may switch views in a number of ways:

Switching Tool Manager views

- Switch to the Manager view using the tabs. Note that a new tab `imager:myimager` now appears. Switch back to the imager tool using this option.

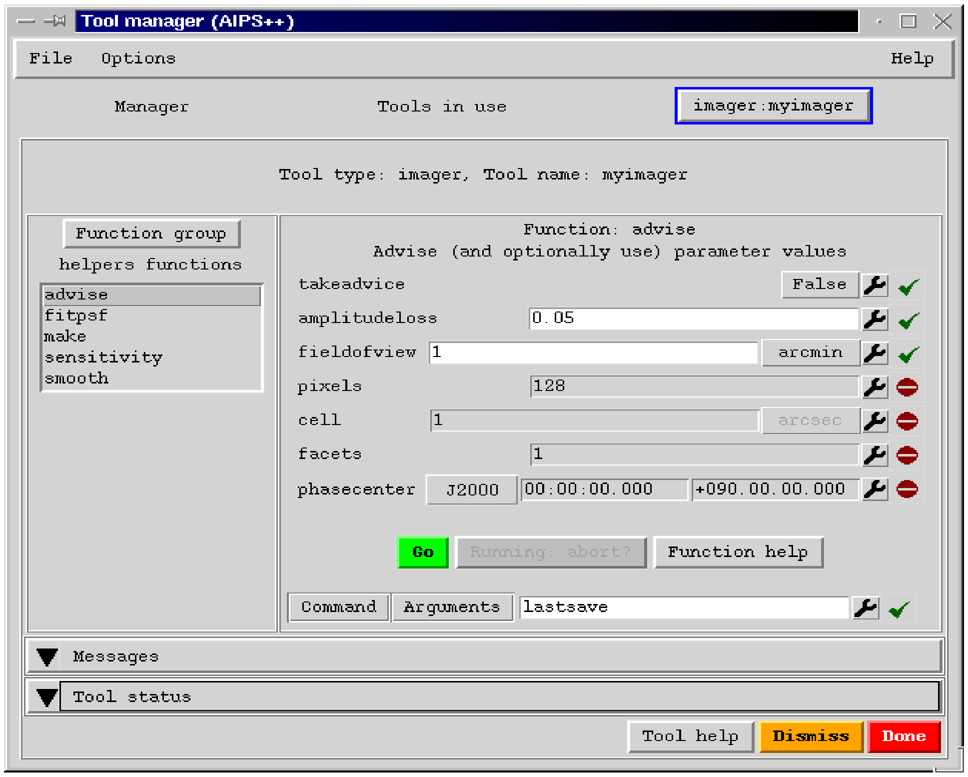

Figure 8: Tool manager view of an imager tool showing the advise function

- Dismiss this view of `imager:myimager` using the the Dismiss button. The tool still exists but this view has been removed from the Tool Manager. The view can be recreated by selecting the `tool` name `myimager` in the Tools in Use view and clicking Show.
- End the use of this tool (and the view) by pressing the Done button. This will destroy the `tool` altogether so that it can no longer be used.
- Switch to the Tools in Use view using the tabs.

8.3 Saving and restoring function arguments

The Save button at the bottom of the GUI, when pressed, saves the current inputs of the currently selected `tool function` to a default *Glish* record named *lastsave*. Similarly Restore fills in the arguments of that `function` from previously saved values in *lastsave*. Also when the Go button is pressed the values used for the `tool function` are automatically saved into *lastsave*. The *Glish* record *lastsave* is written into a file called *aips++.inputs.table* in your AIPS++ directory. This AIPS++ Table can, in principle, contain many *Glish* records. The inputs to a `function` can be saved into a different record by changing the name from *lastsave* to some other name, say, *mysave1*. In this case *mysave1* is written as another *Glish* record in the same table *aips++.inputs.table* (This Table can be browsed from the File Catalog using the View button). Same rules apply for Restore.

Note that Save and Go buttons do not save the entire environment of AIPS++. Each action only saves the inputs of only that `tool function` which is executed. Thus the saved records in *aips++.inputs.table* can be of different length.

8.3.1 The Spanner or Wrench

For each of the parameters of a function, the Spanner (or Wrench) button offers the various options by which parameters can be assigned or used.

- **Original**: Returns the original value the argument had when you opened the `function`. Useful if you typed something wrong and wanted to go back to the original value.
- **Default** and **Clear** are self explanatory.
- **Copy**: Copies the variable (whatever it is - a number, a string, an array) to the Clipboard (see below). Useful if the `function` has an output which you may want to use somewhere. Or if you want to transfer an argument from one `function` to another.

  Paste does the reverse of copy.
- **Unset**: Sometimes not assigning anything to a variable is sensible.
- **Set**: The reverse of unset: sets the value back to a default value.
- **View**: This option comes up depending on the context. If the argument is an image name (*e.g.* an output image in `CLEAN`), then **view** brings up the Viewer tool. Or if it is a Table, it will bring up the Table Browser.
- **Tool**: This option comes up depending on the context. If the argument is a table name then **Tool** constructs an appropriate tool (*e.g.* `ms`

for a MeasurementSet, `image` for an Image, *etc.*, and then uses the toolmanager to display the GUI.

- **Manage**: This option comes up depending on the context. For a region, the regionmanager is started. The regionmanager allows one to specify an area in an image (*e.g.* blc, trc) or a polygon.
- **From catalog** Opens up the File Catalog GUI from which a file name can be picked.
- **From ms** Opens up a context-sensitive graphical tool to select items such as antennas or baselines.
- **From image** Opens up the viewer to select either a point, region, or statistics.
- **Save** Saves a record to the recordmanager. The recordmanager allows you to keep copies of useful records from session to session, and independently of the inputs system. Use `drcm.gui()` to see the GUI for the recordmanager.
- **Restore** Restores a record from the recordmanager.

8.4 The `clipboard`

The `clipboard` is just a clipboard. Any *Glish* variable (A filename, an array, a string of characters, a number, a set of variables) can be copied to the `clipboard` and pasted back to the same or some other place. For instance under the button Inputs, near the top of a tool view, the option **Copy to Clipboard** will copy all the inputs of the function to the Clipboard. They can be pasted back using **Paste** option under the same button. Under the Commands button, which is next to it, the option **copy to clipboard** will copy an equivalent *Glish* command string to the Clipboard. This string, when typed on the Command Line window will execute the function just like pressing the Go button on the GUI. This string can, for example, be copied on to the Command Line window using

---

Pasting from the clipboard

```
x := dcb.paste()     # Set x to the value in the clipboard
eval(dcb.paste())    # Evaulate the contents of the clipboard
```

where `eval` is a very useful *Glish* function that evaluates the supplied string and returns the result.

`dcb` is in fact a `tool` (*defaultclipboard*) which has two functions **copy** and **paste**. It belongs to the module `tasking` in the package `Utility`. In principle, it is available under the Tools in Use view, but you will hardly ever need to use it from there.

Similarly variables can copied to the `clipboard`: copied into a *Glish* variable by using:

Copying to the clipboard

```
dcb.copy(x)                     # Copy the glish variable x
dcb.copy(dm.source('3C273'))  # Copy the position of 3C273
```

8.5 The arguments status button

At the right end of each entry is an indicator of the status of the value. The states are:

Green tick The value is valid. Execution can start.

Red cross The value is invalid. Execution cannot start until this value is changed.

No entry sign No entry is allowed (*i.e.* output only field.

8.6 The `scripter`

Each function that is executed by the Tool Manager has an equivalent *Glish* command string which could be used from the *Glish* prompt or in a *Glish* script to achieve the same result. A `tool scripter` is available to record all equivalent *Glish*commands for a Tool Manager interactive session. It can be enabled using the option **Write to Scripter** under the Commands button in the Tool Manager. This will start recording all subsequent commands to the `scripter` (actually the `defaultscripter`: see the Tools in Use view of the Tool Manager).

`scripter` can save the accumulated *Glish* commands to a file, in addition to other capabilities. This file can be used as a *Glish* script, and can be edited and executed. This is a good starting point for generating customized *Glish* scripts. A given script can be executed from *Glish* using the command `include 'myscript.g'`. A utility is available to edit the script before execution as `dc.edit('myscript.g')`.

9 The File Catalog GUI

Besides the Logger and the Tool Manager, the other important GUI is the File Catalog GUI. This displays the contents of the directory in which you have started AIPS++. Assuming that you started AIPS++ in a new directory that you created for this purpose, the only content of this directory is the sub-directory `./cache` which was also created by you on startup. There are no other files in this directory right now.

The File Catalog GUI offers many options and functions, including:

- listing files and directories in the catalog
- selectively deleting files or directories
- making a copy of a file or rename a file
- giving a summary of the header information of images and data files
- creating or editing an ASCII file
- executing a *Glish* script
- viewing or displaying ASCII files, images, and data sets
- selecting and transmitting names of files to other processes
- constructing a tool from a table using an appropriate constructor

---

Showing the Catalog GUI

GUI Activate the File Catalog GUI from the Tools in Use view of the Tool Manager GUI. Choose `dc` and Click Show

CLI
```
include 'catalog.g'
dc.gui()
```

The GUI for the File Catalog is activated and it is as shown in Figure 6, Of course, the file names you get may be different from the ones shown in Fig 6.

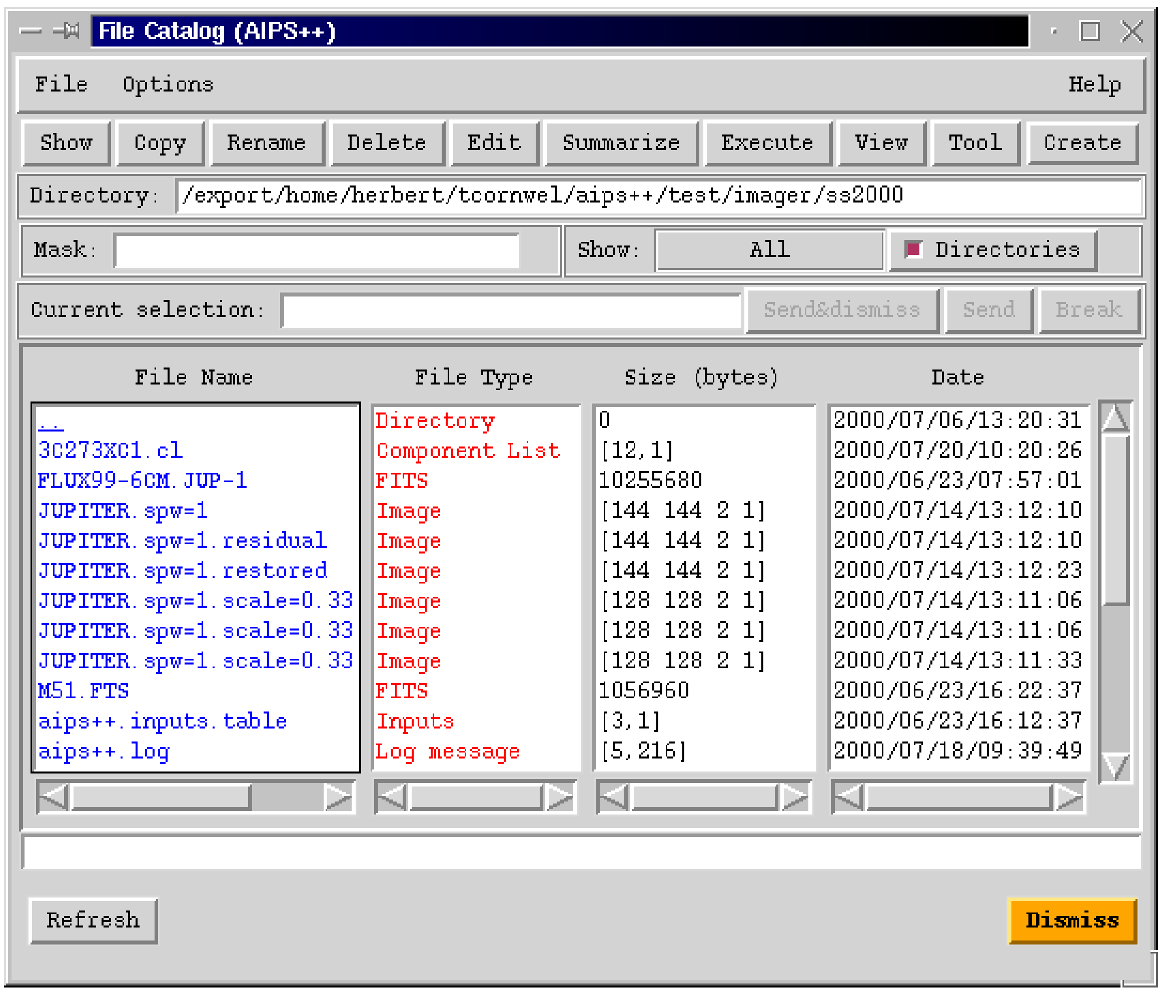

Figure 9: The File Catalog GUI

Let us now take a look at some uses of the File Catalog GUI. Various buttons on the GUI (i.e., Show and Copy) are self explanatory. Placing your mouse pointer over any of these buttons for a few seconds gives you a brief summary of help for that button.

Note that the View button on the File Catalog GUI is intelligent, in that it brings up a Table Browser GUI for an MS, the Viewer for an Image, and uses the `less` command in UNIX for an ASCII file.

Double clicking on a file name is equivalent to selecting the file and clicking the View button.

10 A Practical Demonstration: Making an Image from a visibility dataset

To demonstrate the use of the Tool Manager, we will now make an image. For completeness, we also show how to accomplish the same goal using the *Glish* command line.

10.1 Loading package and module definitions

When you start AIPS++, only package *general*, which is for general processing, is automatically loaded and readily available for use. In order to access other packages or modules it is necessary to explicitly load them.

Loading definitions

GUI Any operation from the Tool Manager (*e.g.* creating a new tool or running a `global function`) normally loads the required modules automatically.

CLI The `include` command can be used to load either a whole package or individual modules:

```
include 'synthesis.g'  # load the synthesis package
include 'table.g'      # load the table tool
include 'ms.g'         # load the module ms
include 'msplot.g'     # load the ms plotting function
include 'utility.g'    # load the utilities module
```

If you write your own script (*e.g.*, `myscript.g`), then to execute the script:

Executing a script

GUI In the File Catalog GUI, double click on `myscript.g`

CLI Use the glish 'include' command:

```
include 'myscript.g'   # execute myscript.g
```

10.2 Create a UV Data Set & Make an Image

For our example, we will need to create a UV data set in the AIPS++ format from the data which has been supplied with the AIPS++ distribution for test purposes. It is a continuum data of the radio source 3C273, taken using the VLA C-array at a frequency near 8.5 GHz. To create this data set in your directory in standard AIPS++ format, do the following:

Creating a test MeasurementSet

GUI Run the global function `synthesis.imager.imagermaketestms`

CLI
```
include 'imager.g'              # Load imager definitions
imagermaketestms('3C273XC1.ms') # Make the MeasurementSet
```

The global function `imagermaketestms` creates the MeasurementSet with the name *'3C273XC1.ms'* from s FITS data file supplied with the AIPS++ distribution.

If the File Catalog is running, click the Refresh or Show button on the File Catalog GUI and notice that a new file name *'3C273XC1.ms'* has been added to the directory. Select this file by clicking it and then click Summarize to get a brief summary of the MeasurementSet on the Logger window.

To make images from this UV data set:

Making images from a MeasurementSet

GUI

- Create an imager using `synthesis.imager.imager` with the filename 3C273XC1.ms.

- In the resulting view for `imager:myimager`, use `setimage` to set the cellsizes to 0.7 arcsec, and the image sizes to 300 pixels. Don't forget to press Go.
- Use `clean` to clean the image. All the defaults are fine.

CLI
```
myimager:=imager('3C273XC1.ms')
myimager.setimage(cellx='0.7arcsec',celly='0.7arcsec',
                  nx=300,ny=300)
myimager.clean(image='3C273XC1.restored', model='3C273XC1.clean',
               residual='3C273XC1.residual')
```

When these operations are finished, click the Show button on the File Catalog GUI to see the new entries.

10.3 View an Image using the Viewer

Images are best viewed using the Viewer. The Viewer can be invoked in a number of different ways.

Viewing an image

GUI

- **Using the catalog:**

  `In the File Catalog, double click on the Image name`

- **Using the tool manager:** Make sure you create a viewer tool (`display.viewer.viewer`) **AND** an image tool (`general.images.image`). If you have created the image tool after the viewer tool Update the viewer data manager and select the Display type in the Load Data panel. The image will appear in the Datasets box. When the image is selected it will be made available on the Display Panel. Select the image in the DisplayData menu and it will be shown.

CLI

- **Using the catalog:**

  ```
  include 'catalog.g'
  dc.view('3C273XC1.restored')
  ```

- **Using the image tool:**

```
include 'image.g'
myimage:=image('3C273XC1.restored')
myimage.view();
```

This pops up a window for viewing the image. Be patient when doing this for the first time in a session. Loading the Viewer currently takes about 10-15 seconds. Subsequent views will be very quick. More than one image can be viewed at the same time. Double clicking on a second image brings up another display panel.

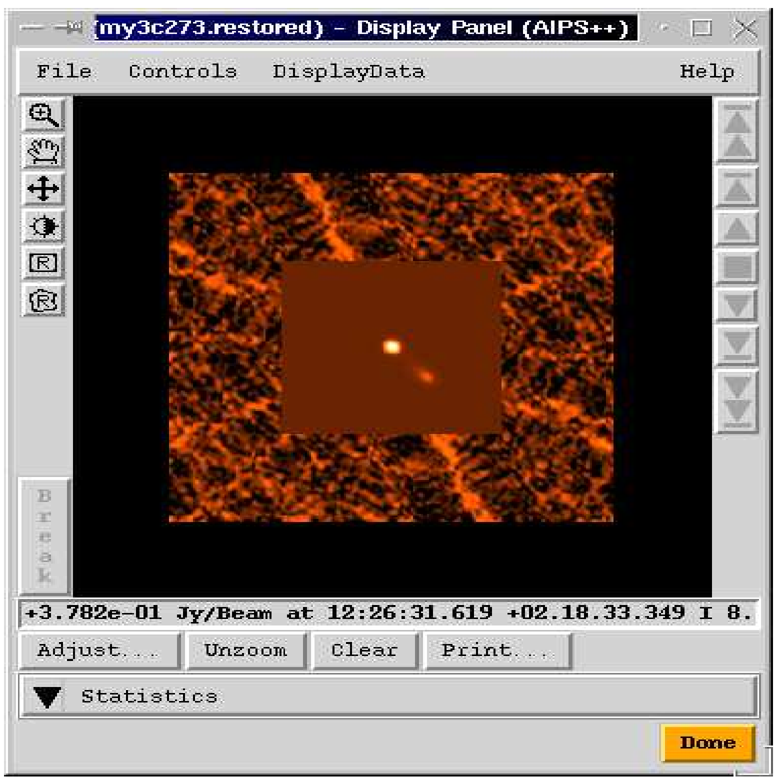

Figure 10: Viewer

Brightness/colormap fiddling.. The color map can be changed in a number of ways. The colormap may be fiddled directly by down-clicking and moving the middle button of the mouse. The data maximum and minimum, scaling power, and type of color map can be set in the Adjust panel.

Adjust Use the Adjust... to obtain a control for the display. This allows a large degree of control over the displayed image and axes.

Zoom/Unzoom To zoom, down-click the left mouse button and drag out the region you want to zoom. Then double-click inside the selected area. This will zoom. To unzoom, press the Unzoom button. To cancel selection of a region, use the Escape key.

Clear Pressing the Clear button clears the display

Print The Print button allows printing of the image, as displayed, to either Postscript or to the X11 pixmap format.

Animation To step plane by plane, or continuously through the other axes of the image, you can use the tape-deck buttons on the right hand side of the Viewer. To change the third axis displayed, use the Adjust control.

Selection of region to display This may be done interactively using the zoom capability described above, or using the **Image Region** specification in the Adjust panel *e.g.* to display the inner quarter of the image, enter `drm.quarter()` in the **Image Region** entry.

The Viewer has many other capabilities. For example, to overlay a raster display with a contour display (not necessarily of the same image), follow these steps:

Displaying a contour on top of a raster

---

1. On the *File* menu, select **Data Manager**
2. On the Data Manager window, Press the Load data panel.
3. In the Display Type window, select **Contour**
4. In the Dataset window, select *image:3C273XC1.restored* (you could select a completely different image here)
5. Back on the Viewer, go to *DisplayData*, and select the second image.

6. To adjust the contour levels, use the Adjust button on the Viewer or the Adjust Data window on the Data Manager.

10.4 Browse an MS using the Table Browser

The MeasurementSet *3C273XC1.ms* and the images *3C273XC1.restored* are AIPS++ Tables. As mentioned earlier, a Table in AIPS++ is actually stored as a directory with the same name as the Tbale. For instance, in the File Catalog GUI click on the file name *3C273XC1.ms* and then click Show. The File Catalog GUI now displays the list of files in the directory named `3C273XC1.ms`. AIPS++knows to recognize this whole collection of files and subdirectories as a single Table. Similarly, *3C273XC1.restored* is a Table representing the restored image and contains many files under the directory with its name. Restore the File Catalog GUI to the display the contents of your main AIPS++ directory by selecting [..] in the File Name column and click Show.

Although you may not do this during regular processing of the data from a telescope, it is useful at this stage to have a look at the insides of an MS and how it is orgnized in AIPS++. To do this you will use the Table Browser GUI..

The Table Browser GUI has many options, for example, you can pull out all of the entries of a given column into a *Glish* array, manipulate it and put it back. You can also plot one column, or a mathematical combination of several columns against other columns or their combinations. Here you will use it just to browse the Table.

Browsing a MeasurementSet

GUI In the File Catalog, double click on the MeasurementSet name

CLI
```
t:=table('3C273XC1.ms') # to make a table tool.
t.browse()              # to invoke the browse function.
t.close()               # when finished, close the tool
```

The name of your Table may not be exactly as shown in Figure 9. For convenience in viewing, use the mouse and stretch the Table Browser GUI to occupy the entire width of your screen. Wait about 10 seconds while

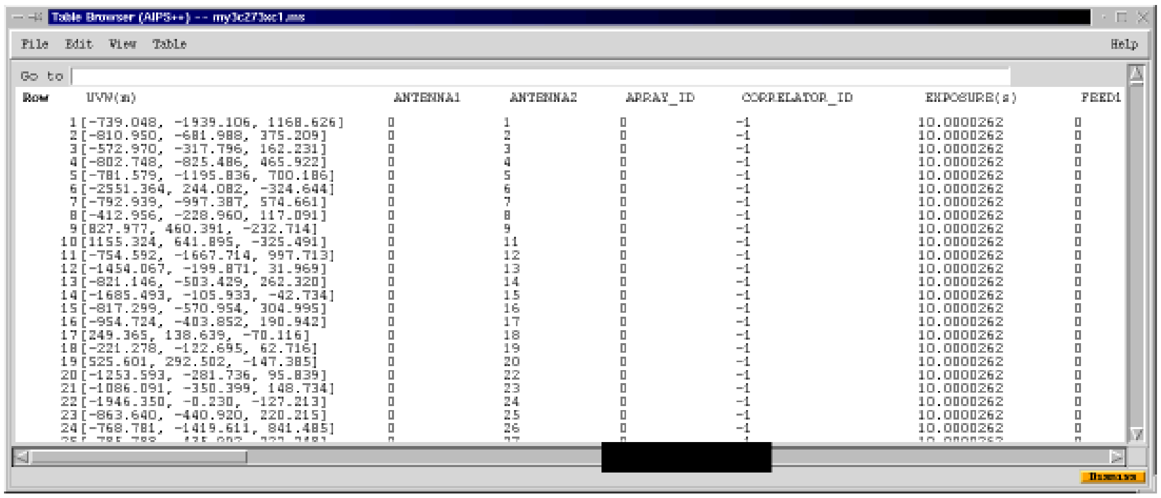

Figure 11: View of MeasurementSet with a Table Browser GUI

several columns of data appear in the window. The row number is given in the first column. It stretches all the way to the number of visibilities (7669 in this case). However, at a given time the Table Browser GUI reads only about 1000 rows. Now use the horizontal scroll bar at the bottom of the Table Browser GUI to look at the various columns (**UVW**, **ANT1**, **ANT2**). This is basically to give you a feel for the MS and also to show that the insides of any AIPS++ Table can be examined or modified. On the Table Browser GUI, use the *View* menu and choose **Table Summary**. A new Summary GUI appears on your screen as shown in Figure 10.

The top part of this GUI gives a brief summary of the MS and the bottom part contains a list of keywords for the MS. Scroll down the list of keywords using the scroll bar on the right side of the GUI. Each one of these keywords represent a sub-Table containing information indicated by the keyword. Click the keyword ANTENNA, and notice that another Table Browser GUI comes up giving details of this sub-Table. Browse the various columns of this Table using the horizontal scroll bar.

After you examine this Table, be sure to use the Dismiss button on this GUI to close the window. Otherwise you will end up with a lot of GUIs on your screen.

Having browsed through some of the sub-Tables of the MS, you can now dismiss the Summary GUI by clicking Dismiss, and remove the Table

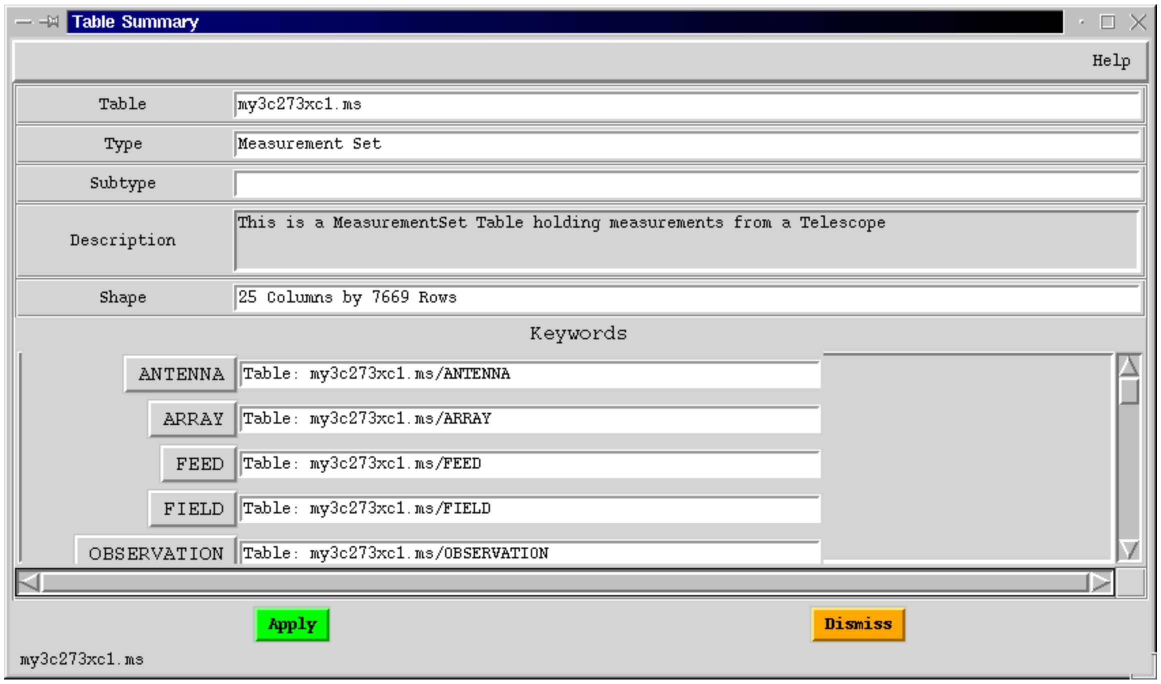

Figure 12: Table Summary GUI

Browser GUI by choosing **Exit** under its *File* menu.

Another example of an Image is the Table *3C273XC1.restored*. Images are best viewed using the Viewer, which is invoked by selecting the file name on the File Catalog GUI and clicking the View button. However, although it is not very instructive, you can also use the Table Browser GUI to get a feel for the Image Table.

11 A Guide to a Typical AIPS++ Data Reduction Session

We describe here what `tools` and `tool functions` could be used in a typical AIPS++ data reduction session. We recommend a GUI based approach. Although a command line approach is possible for using the same `tools` and `tool functions`, it will require a lot of typing and ability to remember the various arguments for the required `tool function`. In GUI based approach, you only need to know how to create the required AIPS++ `tool` and how to use the `tool functions`.

11.1 Data Loading

At this stage it is recommended that you import externally calibrated (and edited) data into AIPS++. Write a FITS disk file of the calibrated UV data into the same directory in which you intend to run AIPS++. To keep things organized, you may choose to create a FITS sub-directory into which any FITS files can be written.

To convert the FITS UV data file to an AIPS++ MeasurementSet (MS), use the `constructor fitstoms` of the `tool ms` which is in the module `ms` of the package *general*. We represent this `constructor` and the hierarchy it belongs in the format `package.module.tool.constructor` as:

`general.ms.ms.fitstoms`

Similarly, if you want to convert a FITS image to an AIPS++ image use

`general.images.image.imagefromfits`

If you want to directly 'fill' VLA data from the archive, then you could use the `global functions`:

`nrao.vla.vlafillerfromtape`

`nrao.vla.vlafillerfromdisk`

11.2 Data Examination

A summary of the data can be obtained using the File Catalog GUI which, if it is not already active, can be invoked from the Tool Manger (Click on **dc** in the Tools in Use view and click Show). To obtain the summary select the MS file name by clicking on it and then click on **summary**. The summary appears in the Logger window.

You can also get the summary using the `tool function`

`general.ms.ms.summary`

Visibility data can be plotted in various ways using the `tool`

`synthesis.msplot.msplot`

Various columns and sub-Tables of the MeasurementSet can be examined using the Table Browser which can be invoked from the File Catalog GUI by selecting the MS name and then clicking on view.

Currently there are no `tools` to directly print or type selected portions of the UV data (like LISTR in **AIPS**). However, if you are adventurous, you can use the tool `utilities.table.table` and its various `tool functions` such as `colnames, getcol` to extract various colums of the MS into *Glish* arrays (see next section) and use the *Glish* command `print` to examine the column values. You can even modify these values and put them back in to the MS using the Table `tool function` `putcol`, which can of course be dangerous.

`synthesis.msplot` is useful for viewing various types of plots of the visibility data, but currently, it has only rudimentary tools for interactive editing of visibility data.

`synthesis.flagger` contains some simple functions for non-interactive editing.

11.3 Calibration

The `tool` to use for calibration is:

`synthesis.calibrater.calibrater`

11.4 Imaging

This is a well developed section of AIPS++. `tools` are available for a variety of imaging requirements such as 3D, wide-field, multi-field, multi-frequency synthesis and mosaicing. The `tools` also allow self-calibration to be incorporated into the loop.

Starting from a calibrated MeasurementSet, the main `tools` for imaging are

synthesis.imager.imager
synthesis.imager.dragon
synthesis.imager.imagerwizard

In the `tool imager`, the `tool functions` to use are:

`setimage`	# To set the image size, cellsize, *etc.*
`setdata`	# To select the require data from the MS
`uvrange`	# Set the UV range
`setimage`	# set the image characterics, size, cell
`weight`	# setting Weighting of UV (robust etc)
`setbeam`	# Specify the convolving beam
`mask`	# Setup a mask to specify "cleaning" areas
`clean`	# Make the image and deconvolve

Some of the steps above are optional, in the sense that sensible defaults will be used. Note that executing each of these `tool functions` will specify the parameters for the `imager tool` for that session. In other words you are specifying a *state* for the Imager `tool`. Repeated execution of `clean`, for example, will use this state. This is a general feature of all `tools` in AIPS++.

11.5 Self Calibration

A self-calibration loop can be inserted in both `imager` and `dragon` tools. In `imager` the `tool function` is `selfcal` which takes as input the name of a `calibrater tool` which should be setup appropriately before hand.

In `dragon`, the self-cal loop is built into the `tool function image`. The relevant parameters to set are *amplitudelevel* and *timescale*.

11.6 Image Analysis

Analysis of images is provided through several `tools`:

`image`, `imagefitter`, `imageprofilefitter`, `imagepol`, `regionmanager`, and `coordsys`.

The basic Image-related `tool` is the `image tool`, in which functions are available for various types of analysis and manipulation. Some of the `tool` `constructors` are:

`image`	# Construct Image tool from an AIPS++ image file
`imagefromfits`	# Construct Image tool and convert from a FITS file
`imagemaketestimage`	# Construct Image tool from a built-in test image

Some of the `tool functions` are

`view`	# View image with Viewer
`summary`	# Generate a summary of the image
`statistics`	# Compute and plot various statistics of the image
`sepconvolve`	# Independent convolution off different axes
`moments`	# Compute various moments of images and cubes
`regrid`	# Regrid image
`subimage`	# Make a subimage
`tofits`	# write FITS file of the image
`calc`	# Mathematical combinations of images

`Imagefitter` is a comprehensive `tool` which can perform component fits to selected regions of the image aided by an integrated image display with zoom, pan, region drawing and other options. Similarly, `imageprofilefitter` is used to fit 1-D models to profiles interactively extracted from an image.

The `imagepol tool` provides specialized polarimetric image anlaysis.

The `regionmanager tool` manages image regions of interest.

The `coordsys tool` manages Coordinate Systems including coordinate conversions.

11.7 Image Display and Contour plots

The main `tool` for displaying, contour and grey scale plotting of images is `viewer`

The Viewer is started from File Catalog GUI by choosing the file name and clicking on View. Multiple images can be viewed simultaneously in different display panels. Clicking on View after selecting a file name starts up a new display panel.

Another way of starting the Viewer is to use the `tool function` `image.view`.

Grey scale range and contour levels can be controlled using the Adjust button on the display panel.

Once the desired contour or grey scale plot is drawn on the Viewer, the Print button on the display panel can be used to send the plot to a printer.

11.8 Data Archiving

The suggested method of data archiving is to use the UNIX tar facility. While MeasurementSets and Images can be tarred directly (*i.e.* in their AIPS++ table format) if they are intended to be read back into AIPS++ directories, we recommend that these be converted to FITS files (using mstofits and image.tofits) and then archived using tar. Use of the FITS format is currently preferred as a long term archive format since the AIPS++ formats may change.

12 *Glish* Overview

Glish is a programming language which allows you to perform mathematical computations, both at the Command Line level and also by writing *Glish* programs or scripts. *Glish* handles the following types of variables: `boolean, byte, short, integer, float, double, complex, double complex, character strings`, and `records`. Another feature of *Glish* is that it allows Automatic Variable Typing in that variables are automatically assigned a type when they are created. No declaration statements are necessary. While this is convenient, it provides no warning if you inadvertently reassign a variable. A whole array can be wiped out by assigning it to an integer. Variables can be protected by using the keyword constant.

Protect a variable

```
const Pi:=3.14159
```

Arithmetic operators are:

```
+ - * / % ^
```

where % is the modulus operator and ^ is the exponentiation. Values of variables and constants in an arithmetic expression are all converted to the type of highest precision in the expression before performing the mathematical operations.

Combine arithmetic and assignment operators

`y:=1:10`	#a 1-d array containing values 1 to 10
`y +:= 2`	# add 2 to every element of the array
`y ^:= 2`	# square every element of the array

Array operations are one of the *Glish*'s strong points for computing efficiency and code economy. For instance

```
x:=1:10; y:= 11:20   # creates two arrays x and y of length 10
z:= x*y              # performs z(i)=x(i)*y(i);i=1 to 10
```

Indexing arrays can be done in various ways.

For contiguous range

```
y[3:7]
```

Random indexing is possible with

`index := [3,5,6,8]`	# an index array
`y[index]`	# get y(3), y(5), y(6) and y(8)

Another special type of variable in *Glish* is the record. A record can contain heterogeneous data structures. A record goes by a single variable name and can be passed on as a single unit to a `global function`. Some `functions` return a record. You can create a record all at once or add to it piece by piece.

- **Create a record**

```
header:= [name='3C273', ra=12.12345, dec=2.3318]
```

This example makes a record by name header with three members **name** (variable type: string), **ra** (variable type: float) and **dec** (float).

Add other members to the record

```
header.flux:=27.6
header.type:='quasar'
header.frequency:=5.0
```

Accessing record members by name or by number

```
header.dec
header['dec']
header[3]
```

Glish variables can be assigned an "attribute"

***Glish* variable attributes**

```
- header.flux::units := 'Jy'
- header.frequency::unit :=  'GHz'
- velocity:=20
- velocity::definition:= 'heliocentric'
- velocity::units:= 'km/s'
- print velocity
20
- print velocity::
[units=km/s, definition=heliocentric]
```

Like record members, attributes can be accessed by name or by number:

Accessing record members by name or by number

```
velocity::[1]
velocity::units
velocity::['units']
```

The fields of a record many be retrieved using the `field_names` function:

Retrieval of field names

```
field_names(header)
```

Some `tool functions` and `global functions` in AIPS++ return a record with many members as their output.

image.coordmeasures

```
myimg:=image('my3C273XC1.restored')   # Make an image tool
posinfo:=myimg.coordmeasures([128,128]) # get info about center pixel
print posinfo
field_names(posinfo)
```

We can now use some of the features of *Glish* in combination with one of the salient features of AIPS++: namely that all the data, either from an MS or an Image can be accessed from *Glish*, manipulated and placed back, if needed. Many of the `tools`, such as Table, MS, and Image, have `functions` which can extract the data from specified portions and create *Glish*variables or arrays with those data. Thus, it is possible to write *Glish* scripts for special user specific applications or do simple computations at the Command Line window itself. Furthermore, since `tools` for printing and plotting are available at the Command Line level, they can also be included in such applications.

Manipulating MeasurementSet data in *Glish*

```
tablecopy('3C273XC1.ms', 'tempcopy.ms') # keep original
mytab:=table('tempcopy.ms', readonly=F) # open with write permission
uvw:=mytab.getcol('UVW')     # get the column named UVW
uvw*:=2.0                    # double the whole Glish array
mytab.putcol('UVW', uvw)     # put the Glish array back into the colum
mytab.flush()                # flush the Table
```

13 The Measures System in AIPS++

AIPS++ has modules called `Measures` and `quanta` (in the package *general*), to handle various astronomical measures and quantities. A *quantity* is value with a Unit (*e.g.* 5km/s, 7arcsec) and *measures* is a quantity with a reference frame (*e.g.* J2000, UTC). Many tools in AIPS++ use the `tools` of the `Measures` and `quanta` modules in their computations. These modules can also be used directly for various astronomical computations and conversions. A module called `measures_data` contains (and updated periodically) a variety of astronomical data (*e.g.* solar system parameters, time related parameters, some source co-ordinates) which are utilized by the various `tools` in the modules `Measures` and `quanta`. All the computations can be handled using GUIs as well as *Glish* commands.

For an introduction to the Measures system, see *The Measures system* by Wim Brouw and *The Measures system: II* by Wim Brouw in April and June 98 AIPS++News Letters.

14 Frequently Asked Questions

To answer Frequently Asked Questions, there is a *FAQ* at http://aips2.nrao.edu/docs/faq/faq.html. If you have a question that is not answered here then use the `ask()` function to ask us.

Appendix A: Structure of AIPS++

To simplify use of AIPS++, the available functionality is split into a hierachy of packages and modules. One can choose to use just those packages and modules which are of direct interest. Each module holds a number of `tools`and global functions.

14.0.1 packages

The top layer of the AIPS++ software system contains packages. Each package contains specific modules. Some examples of packages are:

`synthesis` : for processing aperture synthesis data with modules for calibrating, editing, visibility plotting, imaging, and deconvolving.

`general` : contains modules of general utility such as units and measures, coordinate conversions, image analysis, and manipulations.

`utility` : for printing, catalog handling, and mathematical operations.

`display` : for plotting, and displaying.

`dish` : for processing single-dish data.

`atnf` : contains modules relevant for data from the Australia Telescope.

`nrao` : contains modules relevant for data from NRAO telescopes, currently the GBT and the VLA.

14.0.2 modules

Each package contains a specific set of modules. modules are a group of closely related applications. For example, the modules of package *Synthesis* are:

- `calibrater`: calibrating UV data

- `componentmodels`: making source component models
- `imager`: imaging and deconvolution, wide-field imaging, mosaicing, *etc.*.
- `simulator`: simulating UV and other data
- `msplot`: plotting visibility data in various ways

Appendix B: Creating your own `.aipsrc` file

As described above, the `.aipsrc` file is used to control the detailed behavior of AIPS++. This may be set up using the setupwizard or you may create one by hand:

- **Create a file in your root area named**:

  `.aipsrc`

An example of an *.aipsrc* file is given below, you can find more information about *.aipsrc* in the `aipsrcdata` section of the User Reference Manual.

```
# Work in the current directory
#
user.aipsdir:            .
#
# Put scratch files here: can be a list of directories
#
user.directories.work:   /export/home/xherb4
#
# Put the aips++ cache files in one place: shouldn't be
# over an NFS mount
#
user.cache:             $HOME/aips++/cache
#
# Set these so that bug(), ask(), etc. know who I am
#
userinfo.name: Tim Cornwell
userinfo.email: tcornwel@nrao.edu
userinfo.org:   National Radio Astronomy Observatory
#
# Switch on the display of memory used by each server
```

```
#
user.display.memory:    True
#
# Uncomment this to send log messages to the deep blue yonder
#
#logger.file:           none
#
# Initial height of the logger window
#
logger.height:          20
#
# Disable progress meters?
#
progress.show:          F
#
#system.resources.memory: 128
```