CASA2ARPSPPI: A Walkthrough

The program casa2arpsppi is designed to take tier 2a radar observation files from the CASA network and process them for output onto a specified ARPS grid in the horizontal and radar elevation levels in the vertical. Data from all scans within a given assimilation window are used via interpolation to construct a set of observations valid at the assimilation time that are as complete as possible. The processed data output by casa2arpsppi is suitable for use in the ARPS EnKF system.

Programs Used:

- *scantier2a*   (for producing a list of available radar data files)
- *casa2arpsppi* (for performing the radar processing)
- *ppiplt* (for plotting the output of casa2arpsppi)

Data and Input Files Needed:

- Tier 2a radar files from the CASA radars in NetCDF format
- A copy of arps.input edited for your specific model grid and domain
- A copy of ppiplt.input for each radar you want to plot data for
- A copy of casa2arps.input to edit

Creating the radar listfile:

The first thing you will need to do is create a listfile that contains information about the various NetCDF files you want to process. This file is used by *casa2arpsppi* to locate data files for processing.

1) Sort your NetCDF files into directories named according to CASA radar site

This is necessary so that *scantier2a* properly locates the files. As of spring 2008, the possible choices for directory names are KCYR, KLWE, KRSP, and KSAO, corresponding to Cyril, Lawton, Rush Springs, and Chickasha respectively.

2) Edit the *scantier2a* script

scantier2a is a perl script created by Keith Brewster that reads through your radar data directories and creates the listfile that casa2arpsppi needs. There are four variables you will need to edit: set “tmpdir” to a directory of your choosing—it will be used for temporary storage during file decompression, if necessary. Set “listdir” to the directory you want the listfile to be written to. Set “netraddir” to the full path name (starting from /home…, /scratch…, etc.) of the directory containing your four radar data directories. Set the list in “netrad_radars” to contain the four letter identifiers of each radar you have data for. It will simplify your life later on if, instead of listing all radars, you list just one radar and then repeat this step and the following step for each radar—that way you’ll have a separate listfile for each radar containing only the data for that radar.

3) Run *scantier2a* by executing the following:

```
./scantier2a
```

This command will run *scantier2a* and create your listfile. Note the name of the listfile—you will need it in future steps. To create a separate listfile for each radar, you’ll need to repeat steps 2 and 3 for each one—it may seem like a hassle now, but it *will* simplify things later on.

Preparing and running *casa2arpsppi*:

casa2arpsppi is an ARPS processing module. Its behavior is controlled by the file ‘casa2arpsppi.f90’, located (by default) in the directory src/arps/88d2arps within your ARPS directory.

4) Edit ‘casa2arps.input’

There are four variables you can set in ‘casa2arps.input’ in the top namelist block – you will need to select the correct listfile (the one you just created using *scantier2a*), the correct assimilation time (use the format hhmmss; if you wanted 02:30 UTC, you would need to put 023000, for 18 UTC you would put 180000), and the correct minimum and maximum range of the radar, expressed in meters.

Below are the standard contents of the ‘arps.input’ file – set the grid dimensions and other variables accordingly.

5) Compile *casa2arpsppi* by executing the following within your ARPS directory:

```
./makearps casa2arpsppi
```

This command will compile the *casa2arpsppi* executable for you.

Note: Your copy of the ‘makearps’ script must be properly configured to recognize and compile casa2arpsppi—if it is not, you will need to obtain a copy that is before you can continue.

6) Run *casa2arpsppi* by executing the following within your ARPS directory:

```
bin/casa2arpsppi < casa2arps.input
```

If your input file is named something different than ‘arps.input’ or is located in a different directory, put the full pathname and filename of your input file in place of ‘arps.input’ in the above command. If you have not yet edited ‘arps.input’ so that it is tailored to your chosen model domain, grid spacing, and terrain settings, you will need to do this before you can run *casa2arpsppi*. Also, if you have chosen to use terrain data and have not created the terrain data files, you will need to do that before you can run *casa2arpsppi*.

When *casa2arpsppi* finishes, you will find three output files: a ‘.tilts’ file suitable for use in 3DVAR or ADAS, a ‘.gridtilt’ file (which is of little use to you), and a file with a name of the format “KXXX.YYMMDD.HHMMSS”, where KXXX is the radar identifier, YY is the year, MM the month, DD the day, HH the hour, MM the minutes, and SS the seconds that the data is valid. This last file is the one you can use for assimilation by ARPS EnKF or plotting using *ppiplt*.

7) Repeat steps (4) through (6) for each radar at each time you want data.

If you’re feeling particularly ambitious, you could create a c-shell, perl, or python script that invokes sed or something similar to edit ‘casa2arpsppi.f90’, compile, and run it for each radar and assimilation time you want data. However, unless you have a *large* number of assimilation cycles (or you are a prodigy when it comes to writing scripts) you probably won’t save much time by doing this.

Plotting the data:

It’s important to plot the output from *casa2arpsppi* to make sure that you are satisfied with it, and to make sure that no bugs or errors occurred. You’ll be using the *ppiplt* program to do this.

8) Prepare a copy of ‘ppiplt.input’.

This is the input file for the *ppiplt* executable. You’ll need to edit it so that it has the correct location for your radar data file (“KXXX.YYMMDD.HHMMSS”).

9) Compile *ppiplt* by executing the following:

```
pgf90 -o ppiplt ppiplt.f90
```

If you are not in the same directory as ppiplt.f90, you’ll need to include the relative or full path name instead of just ‘ppiplt.f90’ when compiling.

10) Run *ppiplt* by executing the following:

ppiplt < ppiplt.input

Here, 'ppiplt.input' is the copy that you edited in step 8. *ppiplt* will output a file 'gmeta' that contains the plots. You can open and inspect this file using the program *idt*, or by converting it to postscript with the *meta2gif* script.

11) View the plotted data with *idt* or convert it to a postscript file to view by executing one of the following:
idt gmeta (for viewing the 'gmeta' file)
-or-
meta2gif gmeta (to convert the file to postscript – you can then view it with the postscript viewer of your choice).

If the plotted file looks like you expect it to, then congratulations—you're ready to use the data in your ARPS forecast!