NRAO VLA Archive Survey Pipeline System Users Manual

Jared Crossley (jcrossle@nrao.edu)

Typeset August 6, 2008

Source file (in CVS repository): *filehost*:/users/jcrossle/cvsroot/Archive_Imaging_Doc/Arch_imag_doc.tex

Last revision: v1.5 04 Dec 2007.

Contents

This document describes the use of the NRAO VLA Archive Survey (NVAS) software pipeline, also known as the "VLA Pipeline System". The NVAS pipeline calibrates and images raw VLA data using the Astronomical Image Processing System (AIPS). The images and data output from the pipeline are published as part of the National Virtual Observatory and can be accessed online at http://www.aoc.nrao.edu/∼vlbacald. The pipeline system was written by Lorant Sjouwerman (lsjouwer@nrao.edu) and Jared Crossley.

The pipeline system consists of a series of Perl scripts, C-shell scripts, and AIPS run files (POPS[1] scripts) that

1. query the VLA raw data archive,
2. download VLA data files,
3. setup shell scripts to process the data files,
4. invoke the shell scripts to automatically calibrate the data and produce images,
5. move images, run files, log files, and calibrated data to the VLA archive, and
6. index and create web pages for browsing through images.

This guide will assume that the pipeline user is operating within a C-shell compatible Unix shell; this is accomplished, for example, by typing tcsh in a Unix terminal[2]. Pipeline programs *may* work from other shells, but this has not been tested. To run the pipeline scripts in the NRAO Array Operations Center in Socorro, a user must be a member of the groups vlbacald and aips. In this document, the term *shell script* refers to a C-shell script.

[1] POPS = People-Oriented Parsing System. POPS is the command line interpreter for AIPS. It was developed by Jerry Hudson at NRAO for the purpose of demonstrating that a language can be constructed in Fortran in normal order, instead of reverse polish. Thanks to Eric Greisen for this info.

[2] C-shell compatible shells include: C-shell and TC-shell. The shells sh, bash, and korn are not C-shell compatible.

1 Setting Up the Pipeline

The master copy of all pipeline program files is stored on the pipeline server, currently AOC computer `auto`. The pipeline is initially setup in a given directory on a client computer by executing the shell script `/home/auto/ArchSetup` from the directory in which the pipeline should be installed. In most cases this should be done by the pipeline administrator as user `vlbacald`.

```
> cd [installation directory]
> /home/auto/ArchSetup
```

This script copies the appropriate files and directories from `/home/auto` to the current directory and creates links in the current directory to other files on `/home/auto`. It also adds the program `Arch_Runs` as an hourly job in the crontab file on the client computer. To ensure the crontab file has been updated properly, the user that executed `ArchSetup` can list the crontab file with the command

```
> crontab -l
```

(For more about `cron` and `crontab`, see below in this section.)

The directory into which the pipeline is installed on each client computer is referred to here as the *pipeline working directory*. Unless otherwise specified, files named in this document are located in the pipeline working directory. On computer hamal this might be, for example, `/home/hamal2/vlbacald`.

If the pipeline is to be executed from a new computer, i.e. one that has not run the pipeline before, two steps must be taken. First, the AIPS manager on computer `auto` (Lorant Sjouwerman) must run AIPS script SYSETUP, and then add the appropriate entries to `auto` AIPS files DADEVS.LIST and NETSP. These steps will allow the new computer to execute the version of AIPS installed on `auto`[3].

The second step that must be taken before running the pipeline on a new computer is to edit the `~/.dadevs` file. Two lines must be added to this file to tell AIPS where to store its data. For example, computer zaurak stores pipeline AIPS data on the disk mounted to `/export/home/zaurak2`. The two lines that must be added to the `.dadevs` file are

```
- /export/home/zaurak2/AIPS/DA01
- /DATA/ZAURAK_2
```

Note that `.dadevs` must contain two spaces after the dash at the beginning of each line. If in doubt, ask the pipeline administrator (Lorant Sjouwerman) which lines should be added to `.dadevs`.

After running `ArchSetup`, the current directory will contain a file named `Arch_Defs`. This file defines environment variables used by other pipeline scripts and calls the AIPS setup file `/home/auto/aips/LOGIN.CSH`. It is wise for the user to check `Arch_Defs` to be sure the environment variables have been set correctly by `ArchSetup`. Two variables are of special importance. The variable `avla` holds the path to the pipeline working directory, where the pipeline is installed (and where `Arch_Defs` resides). This may, for instance, look like

```
setenv avla /home/hamal2/vlbacald
```

The variable `email` holds the E-mail address of the pipeline user. The pipeline uses the E-mail address to inform the user that certain tasks have been completed. The E-mail address may be set, for example, by

```
setenv email jcrossle@aoc.nrao.edu
```

It is recommended that the user sources `Arch_Defs` within their shell *rc*-file. If using the (T)C-shell, enter the following in the file `.(t)cshrc` in the user's home directory:

```
source /working_directory/Arch_Defs
```

Note that `Arch_Defs` is read at run time by some of the pipeline scripts, so it is not sufficient to only set environment variables; the file itself must be present and its contents up-to-date.

The pipeline uses the Unix program `cron` to initiate data processing. The cron daemon is controlled by each user's `crontab` file. A user can write their `crontab`

[3] The client computer's AIPS installation is not used because automatic AIPS updates ("midnight jobs") can introduce changes that cause the pipeline to fail; the version of AIPS on `auto` is updated only after testing to ensure compatibility with the pipeline.

file to the terminal using the command `crontab -l` (if `crontab` is empty, nothing will be printed). Each user's `crontab` file contains instructions that tell the cron daemon when to execute various programs. When the pipeline is properly installed by executing `ArchSetup`, the installing user's `crontab` is edited to tell `cron` to execute shell script `bin/Arch_Runs` every hour at 55 minutes past the hour. The cron table is automatically saved for future use.

2 Running the Pipeline

2.1 Preparation

Before running the pipeline, the user should verify that the `umask` is set to produce files with group read-write permissions. This is accomplished with the command

```
> umask 0002
```

The user must also source the file `Arch_Defs` in the pipeline working directory,

```
> tcsh
> cd [pipeline working directory]
> source Arch_Defs
```

A regular pipeline user should include the above `umask` and `source` commands in her/his shell rc-file: ∼`/.tcshrc` for TC-shell; ∼`/.cshrc` for C-shell.

2.2 ArchQuery

The pipeline is started with two user-executed shell scripts: `ArchQuery` and `ArchPreps`. The user must first query the VLA archive using `ArchQuery`. This program acts as a command line interface to the world-wide-web-based VLA data archive, accessed via the URL `http://archive.nrao.edu/`.

`ArchQuery` requires the start date be specified on the command line and allows two command line options, specifying the subarray number and project code.

`ArchQuery [-s#]` *YYYY mmm DD* `[`*Project code*`]`

The command line option `-s` must be given after '`ArchQuery`' and before the date; no space is allowed between the '`s`' and the subarray number, '`#`'; the subarray number should be a single digit between 1 and 5, inclusive. The date must be given in the specified format. For example, to query all VLA observations for December 5, 1995, type:

```
> ArchQuery 1995 dec 05
```

To limit this query to only observations with project code `AB628`, type:

```
> ArchQuery 1995 dec 05 AB628
```

If executed without the '`-s`' command line option, `ArchQuery` immediately produces a prompt requesting the user to specify the number of subarrays for which data should be requested of the VLA data archive.

```
** Select subarray (1 to 5):  [1]
```

The VLA can be divided into a maximum of five subarrays for any given observation, and data must be queried separately for each subarray configuration. The default query is subarray 1, because it is the most used subarray configuration. Subarrays 2 and higher usually contain VLBI observations and system tests.

The range of time queried by `ArchQuery` is set in file `Arch_Defs` by the variable `interval` in the form of days and hours, *dd*`:`*hh*. For example, a query interval of 2 days and 6 hours is specified in file `Arch_Defs` by the line

```
setenv interval 02:06
```

With `interval` set to this range, the command

```
ArchQuery 1995 dec 05
```

will return data files that have a start time between the beginning of December 05, 1995 and 06:00 hours on December 07, 1995.

Successful execution of `ArchQuery` generates a text file `$atmp/archfiles.dat`[4] that contains

1. a comment line stating the date and time range, subarray number, and optionally the specific VLA project number submitted in the VLA data archive query (the inputs to `ArchQuery`),

2. the command executed to query the VLA data archive, and

[4]The file `archfiles.dat` is used as input to `ArchPreps`, see § 2.3.

3. information on each file (or, equivalently, *data set*) returned by the VLA data archive in response to the query.

In addition to generating `archfiles.dat`, `ArchQuery` also echoes a filtered version of the VLA data archive response to the terminal (this output is created by executing `ArchPrep -`; see below). An example of a filtered response for December 5, 1995 is,

```
Selection of files:
-   AA169  D 95-Dec-06 07:43:49 95-Dec-06 12:42:20
-   AK403  B 95-Dec-05 03:20:39 95-Dec-05 08:46:49
-   AK410  C 95-Dec-05 23:52:50 95-Dec-06 07:43:10
-   AL364  D 95-Dec-07 04:11:10 95-Dec-07 08:37:10
-  APHOLO  F 95-Dec-05 01:21:56 95-Dec-05 03:18:03
-  MJCTST  D 95-Dec-05 22:27:20 95-Dec-05 23:44:19
```

The filtered response contains one data file in each row, and each row is divided into six columns. From left to right the columns are the project code, the segment, the start date, the start time, the end date, and the end time.

`ArchQuery` automatically filters out files from the archive response that contain project codes that should not be imaged by the pipeline. Such data includes VLA tests and large VLA surveys that have been or will be imaged separately (for example, NVSS, FIRST, and others). Project codes containing "OPS", "SYS", or "ZBL" should also be skipped; `ArchQuery` will automatically filter some of these project codes, but since some code names are assigned at test time and no exhaustive list of these project codes is available, `ArchQuery` will not filter them all.

2.3 ArchPreps

The script `ArchPreps` can be run manually following `ArchQuery`. `ArchPreps` downloads data files selected from `archfiles.dat`, loads the data in AIPS, checks that the data is appropriate for imaging in the pipeline, and, if appropriate, prepares the loaded data for imaging at a later time.

This data-loading program actually consists of two parts: a Perl library (*module*) called `Arch_Preps.pm` and a user interface to the library simply called `ArchPreps`. Under normal operation, the library should go unnoticed by the user. This document therefore refers to these two files by the name of the user interface, `ArchPreps`.

```
Up for download:  (select only one project name!)
-------------
- Select AA0169  D 95-Dec-06 07:43:49 95-Dec-06 12:42:20?  [Y]
n
-------------
- Select AB0628 AS 95-Dec-05 08:48:49 95-Dec-05 10:18:49?  [Y]
n
- Select AB0628 AS 95-Dec-05 10:18:54 95-Dec-05 14:10:34?  [Y]
n
- Select AB0628 AS 95-Dec-05 14:10:39 95-Dec-05 18:04:44?  [Y]
n
```

Figure 1: Example of `ArchPreps` output when run with no command line options.

2.3.1 Command Line Interface

`ArchPreps` allows for one command line option:

`ArchPreps [`*`project code`*`|-|+ `*`project code`*`|+]`

where *`project code`* is the project code to be loaded into the pipeline. These options have the following effects:

- Using no command line option causes `ArchPreps` to ask the user to make a selection from each file listed in `archfiles.dat`. Figure 1 shows an example of `ArchPreps` output when using no command line options.

- Using "*`project code`*" for the command line option causes `ArchPreps` to ask the user to make a selection only from the files with a matching project code. Project codes with additional pre- and post-pended characters will be listed for user selection.

  When run in one of the above two "selection" modes (using the *`project code`* option or no option) `ArchPreps` places an ASCII marker, such as

  ```
  -------------
  ```

  between files with different project codes and files that are separated in time by 6 hours or more. See example output in Figure 1.

- Using "`-`" for the command line option causes `ArchPreps` to echo a filtered version of the data set information contained in file `archfiles.dat`, identical to the filtered version echoed to the terminal by `ArchQuery`[5].

- Using "`+ project code`" for the command line option causes `ArchPreps` to load all matching files in `archfiles.dat` without the user confirming individual file selection. Since file selection is automated, project codes with additional pre- and post-pended characters are not matched.

- Using "`+`", with no project code, causes `ArchPreps` to automatically load each project code beginning on the query date. The *query date* is the date used in the most recent call to `ArchQuery`.

  When the "`+`" option is not used, it is left to the user to decide which files with a given project code should be processed together. When in doubt, use the ASCII markers as a guide to project file grouping.

When using the "`+`" *auto-loading* option, all files with the same project code and beginning on the query date are grouped together. Files with the same project code beginning on subsequent days are joined to the group according to the following algorithm (files are inspected in chronological order):

1. **If** the gap between the previous file (starting on day 1 or later) and the next file (starting on day 2 or later) is < 30 minutes, **then** include the next file in the current group.

2. **Else, if** the gap is > 6 hours, **then** exclude the next file from the current group.

3. **Else, if** the gap is > 30 minutes and < 6 hours, **then**

   (a) **If** the difference between the start of the first file after the > 30 minute gap and the end of the $LAST$[6] file satisfying conditions (1) or (3) is < 6 hours, **then** process all files together;

   (b) **Else**, exclude the next and all subsequent files from the current group, so it can be processed with the subsequent day.

[5] Actually, the `ArchQuery` terminal output is produced by calling `ArchPreps` with option "`-`".

Table 1: File Grouping Tests

Day	Project Code	Condition Tested
1993 Mar 06	UH0002	3-b

Of the above three conditions, (1) is the most common, and (3) is the least common. It is therefore useful to keep a list of VLA archive data files that test the above conditions.

2.3.2 About Project Codes

`ArchPreps` will abort if two or more files with different project codes are requested by the user. This makes sense, because different projects should not be processed together.

A small number of project names may contain non-standard characters that cause `ArchPreps` to fail. The user should make a list of the project name and the archive date on which the name occurred. A future version of `ArchPreps` may be designed to deal with unusual project names, or the project name may require manual loading into AIPS.

If two archive files with the same project code on the same day in the same subarray are processed separately but on the same computer, the `ArchPreps` output files will overwrite each other. This occurs because the file naming scheme used by `ArchPreps` only accounts for the project code, date, and subarray number. If the two files are processed separately on different computers, when the pipeline output files are moved to the image archive, some of the output from the second file will overwrite some of the output from the first file. This is why files with the same project code, same start

[6] This is the *last* file to be added to the group; this condition is assumed true until the group is fully built, at which time the condition is assessed, and files are excised from the group as necessary.

date, and same subarray number should be processed together.

Remember that project codes containing "OPS", "SYS", "ZBL", or any other obvious system check, should be skipped. However, testing project codes such as "TEST" and "TST" should be loaded. Some of the unusable files will be excluded from `archfiles.dat` by `ArchQuery`. However, it is not possible to identify all unusable tests. The user should watch for files of the same project name that repeatedly fail to load in AIPS. By identifying such files, archive imaging will proceed at a faster rate.

When the files have been selected (if in selection mode), `ArchPreps` will either download the files from the VLA data archive to the `$avis` directory, or, if the files have already been downloaded, will ask the user to confirm use of the files already present in the download area.

2.3.3 Archive Data File Download

The latest version of `ArchPreps` automatically monitors archive file download. This is accomplished by monitoring the user's email file. The VLA Archive sends an email to the user when each file download is complete. This e-mail has a standardized format, and contains the full path name of the downloaded file; the file name itself contains the project code, date, and subarray number. `ArchPreps` searches email file `/users/`*username*`/mail/NVASmail` for this unique string, `ArchPreps` is able to verify download completion. Currently, this email file is hard-coded in `ArchPreps`, and it is the user is responsibility to filter archive email messages to this file; this can be accomplished with the Unix utility `procmail` (see §???).

NRAO policy is to "lock" all data files associated with project codes for which observations have been taken in the past year; a locked file can only be downloaded with a password given to the observer; *it cannot be downloaded by `ArchPreps`*. If `ArchPreps` attempts to download a locked file, the archive will send the user an E-mail containing a message similar to this,

```
Copy process canceled.  LOCKED archive file :
/home/archive_VLA/tapes/XH00035/file_6
```

If a data file is locked, it should be skipped and loaded into the pipeline at a later date.

`ArchPreps` creates run and log files during its execution. The run files are shell scripts that are executed either immediately by `ArchPreps` or later by `Arch_Runs`. The log files store diagnostic output during run file execution. The run and log file names differ only in their extension. Loading data into AIPS is accomplished by a run file with extension `fill.run`, and diagnostic messages are stored in a log file with extension `fill.log`. Data is moved to different AIPS numbers by a run file with extension `move.run`, and diagnostic messages are stored in a log file with extension `move.log`. For example, running `ArchPreps` on project AB998 on 2001 July 04 produces the following files:

```
AB998_2001JUL04_1.fill.run
AB998_2001JUL04_1.fill.log
AB998_2001JUL04_1.move.run
AB998_2001JUL04_1.move.log
```

When the user has received the download confirmation E-mail and pressed `ENTER` to confirm that the archive download is complete, `ArchPreps` will then load the downloaded file(s) into AIPS, split the data according to observing center frequency, and move each separate frequency to its own AIPS user number. C-shell scripts are then generated for each AIPS number in directory `run`; these scripts are referred to as *run files* and given a filename suffix `run`[7]. The run file names are made by concatenating the selected project name, the observing date, the subarray number, an extension, and a suffix. For projects with multiple observing frequencies (and therefore multiple assigned AIPS numbers) the run file extension contains a sequentially incremented number that distinguishes multiple run files. For example, run files for VLA project AD433, observed on March 1st, 2000, have file names,

```
AD433_2000MAR01_1.0.run
AD433_2000MAR01_1.1.run
AD433_2000MAR01_1.2.run
AD433_2000MAR01_1.3.run
```

`ArchPreps` returns to the command line after creating the run files.

[7] C-shell script *run files*, which execute Unix commands, should not be confused with *AIPS run files* which execute AIPS commands.

There are presently two conditions under which ArchPreps will skip some, but not necessarily all, data in a file:

1. If the data is spectral line[8], ArchPreps will skip it with explanation,

   'ZAP LINE :' '*AIPS file name*'

2. If the data is P-band (300 MHz), ArchPreps will skip it because it requires special atmospheric calibration. ArchPreps skips P-band data with explanation,

   'ZAP P-BAND :' '*AIPS file name*'

3. If the data is 4-band (74 MHz), ArchPreps will skip it because it requires special treatment of atmospheric calibration. ArchPreps skips 4-band with explanation,

   'ZAP 4-BAND :' '*AIPS file name*'

In both of these cases, if other acceptable data resides in the data file, the acceptable data will still be processed and imaged.

There are three conditions under which ArchPreps will abort, skipping *all* data in the file:

1. If there is no standard calibrator source in the data, the pipeline does not know how to set the flux scale. Thus, ArchPreps will abort with explanation,

   'ZAP - NO STD CAL SRC FOR :' '*AIPS file name*'

2. If the data was recorded as a single dish for the purpose of VLBI, an image cannot be made from the data. In this case, ArchPreps will abort with explanation,

   'ZAP S-DISH :' '*AIPS file name*'

3. If there are fewer than 8 antennas used in the observation, making a proper image is very difficult. Thus, ArchPreps will abort with explanation,

   'MINI ARRAY :' '*AIPS file name*'

If ArchPreps skips all data in a file or if the file is found to have no visibilities (no data), the AIPS data will be deleted, reserved AIPS numbers will be freed, and the following message will be issued along with an audible bell before quitting:

No files left to work on - see */working_directory*/run/*pcode_date_subarray*.fill.log

After splitting the data according to observing band and skipping data where appropriate, ArchPreps will write the AIPS number, project code, extension number, and log file path to bin/Arch_AIPS; one line is written for each AIPS number used. If all data is skipped, only one line is written and the five digit AIPS number is replaced with zeros. If only some of the data is skipped, the data that has not been skipped is entered in Arch_AIPS.

For each computer used to load data into the VLA pipeline, only one instance of ArchPreps should be allowed to load data into AIPS at any given time. If multiple instances of ArchPreps finish loading data at the same time, they will both try and write to Arch_AIPS, but no more than one program will succeed. When ArchPreps fails to write to Arch_AIPS, the AIPS numbers it used for loading data will be lost and may be reused by a subsequent instance of ArchPreps. In this case, the best course of action is to rerun ArchPreps on the same data so it will successfully store its AIPS numbers in Arch_AIPS. It is also advisable in such a situation for the user to check the Arch_AIPS file for potential errors caused by the conflict; for example, duplicate instances of the same AIPS number.

2.4 Auto-loading: ArchAuto

The highest level of data loading automation is attained through the Perl script ArchAuto. The script

[8] Future versions of the pipeline system will not skip spectral line observations.

takes a start and stop date as command line parameters and then calls `ArchQuery` and `ArchPreps` automatically for each date in the range, including the start and stop dates. A call to `ArchAuto` has the form

`ArchAuto` *`start_date stop_date`*

The `start_date` and `stop_date` are formated as

YYYY MMM DD

where YYYY is the numerical year, MMM is the first three letters of the month (upper, lower, or mixed case), and DD is the one or two digit day of the month. For example, to auto-load the entire month of February 1998, type

```
> ArchAuto 1998 feb 01 1998 feb 28
```

`ArchAuto` contains some internal error checking to verify appropriate start and stop dates. While `ArchAuto` runs, the standard output from `ArchQuery` and `ArchPreps` will be sent to the terminal. Audible beeps have been entered into `ArchPreps` to alter the user to circumstances of interest, such as when a project group spans multiple days. It is advisable to monitor `ArchAuto` periodically as it runs, to insure that it is working properly.

2.5 Run file execution: `Arch_Runs`

All new run files are executed automatically and sequentially by the program `Arch_Runs`. This does not require any user interaction; the `cron` daemon initiates `Arch_Runs` automatically every hour. By default, `Arch_Runs` checks the modification time of uninitiated run files and only executes them if they are more than 5 minutes old. This gives the user time to manually alter the script or data if desired. For example, in some cases the user may want to flag bad data manually before the pipeline begins imaging.

Alternatively, the user may choose to execute run files manually without delay by using the command line option "`-`":

```
Arch_Runs -
```

Execution with the "`-`" option causes `Arch_Runs` to request user verification that the data has been fully downloaded from the archive and loaded into AIPS. After responding to this prompt, the user may choose to background `Arch_Runs` using `control-z` and the linux command `bg`. This makes the terminal immediately available for additional commands while `Arch_Runs` continues to run in the background. Be aware that closing the terminal window while `Arch_Runs` is backgrounded may force `Arch_Runs` to exit early; the unix utility `nohup` may be used to avoid this. `Arch_Runs` is not dependent on the state of the user's terminal if it is initiated by the `cron` daemon.

When `Arch_Runs` is executed, all run files that meet the age requirement are renamed with extension `.auto`; this extension signifies that the run file has been initiated but not completed. When execution of a run file is complete, it is renamed with the original extension `.aips.run`. Likewise, the log files are given the name `.aips.log`. Note that `Arch_Runs` initiates immediately all run files meeting the age requirement, but executes the run files sequentially.

Since it is possible for `Arch_Runs` to run for longer than one hour, multiple instances of `Arch_Runs` may be running simultaneously. There are limitations to the number of instances of AIPS that one computer can run simultaneously. Furthermore, simultaneous execution slows every imaging process and all other processes running in the computer. To avoid this situation, `Arch_Runs` will not execute if two other instances of `Arch_Runs` are already in execution.

When all run files initiated by `Arch_Runs` have been completed, `Arch_Runs` deletes the AIPS data stored in each AIPS user number it has used and sends an E-mail to the user stating (1) the AIPS user number, (2) the run file name, and (3) the completion time for each file executed. The net start and end time of `Arch_Runs` is also given in the email. These time stamps can be used as an indicator of `Arch_Runs` success or failure. If a run file's completion time is within a few seconds of the previous run file's completion time, then something has probably gone wrong. In this case, check the log files in the `tmp` directory for possible causes of the error.

`Arch_Runs` stores its output in the pipeline working directory. When it has finished, the output image, visibility plots, u-v plots, log files, and raw data remain on

disk until a cron job moves them to the image archive server. Presently, this file transfer process occurs once each morning, on Tuesday through Saturday. The file transfer is accomplished by the program `Arch_Fill`. The data in AIPS is all removed automatically.

2.6 Image Validation

Data that has been processed by the pipeline is automatically exported to the image archive server. From this server the images, visibility plots, *u-v* plots, log files, run files, multisource FITS files, and single source FITS files can be accessed from the NVAS[9] web page:

`http://www.aoc.nrao.edu/~vlbacald/`

However, some percentage of the images produced by the pipeline are not suitable to be stored in the image archive; this is often a result of improper calibration or flagging. These images must be removed from the archive server by the user. A web-based "image validation" script is available for this purpose at this URL:

`http://www.aoc.nrao.edu/~vlbacald/val/`

Anyone can submit queries form the validation web page. However, the user must be granted permission by the pipeline administrator to remove images; a username and password is required to authorize image removal.

Images are queried by the image validation tool according to the date that the image was placed in the image archive. The script reports the total number of images found for the query, but will only display 200 images at one time. Thus, if there are more than 200 images returned by a query, the user must also specify the range of indices that should be displayed (for example: 1-200, 201-400, or 401-600).

Each query will return a table in which each row corresponds to one image. The table presently contains five columns. From left to right the table columns are:

1. the index number of the image in the query, a removal ("zap") check box, and the file name of the JPEG image[10];
2. a thumbnail of the VLA image;
3. a thumbnail plot of visibility amplitude versus baseline length;
4. a thumbnail plot of real versus imaginary parts of the image visibilities;
5. a thumbnail plot of *u-v* coverage.

Clicking on any thumbnail image will open a larger version of the image in a new browser window[11].

To remove images, check the "zap" box in the first column of each row containing a poor image. When all appropriate boxes have been checked, enter the user name and password and click the submission button.

Some example visibility and *u-v* plots are provided with comments on the NVAS README web page. These can be referenced for comparison when judging image quality. The root-mean-square flux, contained in the image file name in column one is also useful for assessing image quality.

When using the image validation script, be aware that the table of images returned by a query cannot be "refreshed" like a standard HTML file. The web page table is generated when the query is submitted, and to reload the page may require query resubmission. The open source web browser Firefox is able to reload a query page after submission. It does this using the html and image files in its cache. It will also reload checked boxes. Firefox can be obtained at `http://www.mozilla.com/en-US/firefox/`. Eventually, Firefox may delete old files from cache, after which time the web page cannot be reloaded without resubmitting the original query.

2.7 Maintaining Available Disk Space

Pipeline data processing can rapidly consume large amounts of hard disk space. Inadequate disk space can cause cron jobs to fail, sometimes running indefinitely in the background. It is recommended that the user check the available disk space daily.

[9] NVAS = NRAO VLA Archive Survey

[10] The image file name contains much information about the image. See the NVAS README page for instructions on how to interpret the file name: `http://www.aoc.nrao.edu/~vlbacald/read.shtml`.

[11] The size of the VLA image depends on the resolution and field of view available in the corresponding observation. In some cases, the original image will be smaller than the thumbnail.

One of the daily cron jobs maintained by the pipeline administrator sends email to the user giving a summary of the disk space available on each partition and each computer used for pipeline imaging. Another cron job setup by the pipeline administrator should remove older visibility files (via the script `clean-vis`, described below). However, if a very large amount of data is loaded on one computer in a 24 hour period, or on the weekend when cron jobs may not run, the available disk space may need to be checked and the old visibility files removed manually.

Available disk space can be checked with Unix command `df -h`. The amount of free memory available on the pipeline systems partition can be queried with the command

```
> df -h $avla
```

The environment variable `$avla` will only work after sourcing the file `Arch_Defs`, located in the pipeline working directory.

When free disk space is low, the user should run script `/home/auto/bin/clean-vis`. This script will delete all but the ten newest visibility data files from the `vis` directory; the script will remove files on *all* computers used for pipeline imaging.

3 Documenting Pipeline Work

3.1 The VLA Loaded Data Log

The VLA loaded data log (referred to as the VLA imaged data log in earlier versions) is a data table that contains information on every data file downloaded from the raw data archive and loaded into the pipeline. Presently, the loaded data log resides in a CVS repository on the AOC filehost:

`/users/jcrossle/cvsroot/VLA_log/VLA_log_2.txt`

The loaded data log is written as an ASCII file in comma-separated-variable format, and contains 8 columns:

1. the project code,
2. the observation date,
3. the name of the computer used to load and image the data,
4. the AIPS number,
5. the frequency (or extension) number that differentiates between different frequency observations within the data file,
6. the file status at the time `ArchPreps` finished ("Queued" means successfully loaded and awaiting imaging; "Skip" means the file is not appropriate for imaging and has been skipped)
7. a user comment,
8. and a comment date, which is typically the date the data was entered in the log file.

Data files that are found to be inappropriate for imaging are given the status "Skip" and AIPS number "0".

The VLA loaded data log is constructed from the pipeline data file `bin/Arch_AIPS` via the `bash` script

`/users/jcrossle/cvsroot/script/MakeLog`,

also located in a CVS repository. `MakeLog` uses `awk` to read `Arch_AIPS` on each computer used for pipeline processing. Any new data files found in `Arch_AIPS` and not in the VLA loaded data log are appended to the log file. Comment lines in `MakeLog` explain the details of how the log update is accomplished.

3.2 The Supplementary VLA Log

The supplementary VLA log contains all information important to the pipeline imaging process that is not contained in the VLA Loaded Data Log. The supplementary log is contained in a CVS repository,

`/users/jcrossle/cvsroot/VLA_log/VLA_log_2_sup.txt`.

The supplementary log includes, from top to bottom of the file,

1. a list of data files that failed to load along with any appropriate comments intended to aid the pipeline administrator in resolving the problem,
2. any useful comments for the user (for example, a list of project codes to skip, or dates that should not be processed for various reasons),

3. a list of pipeline improvements
4. a list of months in the archive for which data has been fully loaded,
5. a list of dates for which data has been fully loaded,
6. a list of dates that have been validated.

The supplementary log is maintained by the user during the data loading and validation process.

3.3 The Users Manual

The Pipeline Users Manual (this document) exists for the purpose of documenting, on the user's level, what the imaging pipeline does and how to operate it. The Users Manual is written and maintained with Latex. Latex can be obtained via links at the web page `http://www.latex-project.org/`. The present version of the Users Manual is formatted for compilation on the author's computer, in Macintosh program TeXShop. Some minor modifications may be necessary for compilation on other computers.

The Users Manual uses the latex package glossary. This package can be obtained along with installation instructions at this URL: `http://tug.ctan.org/tex-archive/macros/latex/contrib/glossary/` The glossary installation package contains a perl script, `makeglos.pl`, that must be run each time glossary entries are updated.

3.4 Going Deeper...

To learn more about the pipeline system, the user can contact the pipeline administrator or delve into the pipeline system source code. A few remarks may be useful for accomplishing the latter.

A Short List of Pipeline User Rules

1. Before loading data, check to be sure there is sufficient disk space. It is difficult to gauge what "sufficient" is, but a rule of thumb might be 2 GB free for each archive day loaded. Avoid computers with less than 5 GB free disk space.
2. Do not simultaneously run multiple instances of `ArchPreps` and/or `ArchAuto` on one computer. To be more specific, do not allow multiple instances of `ArchPreps` to simultaneously load data into AIPS on any one computer. This can sometimes cause AIPS user number conflicts that may crash `ArchPreps`.
3. Keep a record of dates that have been loaded and dates that have been validated. The pipeline does not do this automatically.

B Why use `cron`?

I was recently asked why the pipeline system uses `cron` to run the data processing and data archive stages. Pondering this has led me to an interesting realization.

Lorant Sjouwerman designed the pipeline system to use `cron` to start `Arch_Runs` and, less frequently, to run the archiving and filesystem monitor and cleanup scripts. In earlier days (late 2006), the pipeline data loading stage was quite interactive. There was no `ArchAuto` and no autoload command line option in `ArchPreps`. The data processing however, was fully automated. `cron` thus provided a "pipeline interface" between more interactive and less interactive stages. The user interactively loaded data into the pipeline, and then could go away while `cron` processed all the data. Without an automated program like `cron` to start the data processing stage, the user would need to load data and wait for it to finish imaging before loading more data.

`cron` also allows the pipeline operation to switch from one user to another. For example, user jcrossle operates the data loading, and user vlbacald operates the data processing and archiving. This can be advan-

tageous, if the two users do not want to accidentally interfere with each other's stages of the pipeline.

C Using CVS to Store Pipeline User Records

A pipeline user must maintain the VLA Loaded Data Log, the Supplementary VLA Log, and the Pipeline Users Manual. A version control system can simplify maintenance of these records[12]. Presently the open source program Concurrent Versions System, or CVS, is used. CVS can be obtained at `http://www.nongnu.org/cvs/`. Documentation on CVS is available at `http://ximbiot.com/cvs/manual/`. Users unfamiliar with CVS are referred to this documentation. Some of the more important commands are briefly explained below.

The pipeline CVS repository is currently located on AOC computers in `/users/jcrossle/cvsroot`. A user wishing to checkout pipeline records from the repository can do so by entering the command

```
cvs -d hamal.aoc.nrao.edu:/users/jcrossle/cvsroot
checkout Archive_Imaging_Doc aips_run VLA_log
```

This command will create directories `Archive_Imaging_Doc`, `aips_run`, and `VLA_log` in the users current working directory. The files contained in these directories can then be read, written, and executed by the user as usual.

Changes made within one directory can be saved to the repository with the command

```
cvs commit
```

executed from within the modified directory.

The users working directories can be made concurrent with any changes made to files in the repository with the command

```
cvs -d hamal.aoc.nrao.edu:/users/jcrossle/cvsroot
update
```

executed from the top-level working directory, that is, the directory containing `Archive_Imaging_Doc`, `aips_run`, and `VLA_log`.

[12]Note that version control is not strictly necessary for maintaining pipeline user records; it provides no great advantage for a *local* user; however, version control is quite advantageous for a remote users since it offers an easy way of backing-up the user's files. Since version control is currently being used, some details are given here.

D Using AIPS Remotely

In the present version of the pipeline, the user needs only access AIPS directly for troubleshooting purposes. Even this might be avoided if the pipeline administrator is available to address problems that arise. However, such is not always the case, and on occasion the user may need to access AIPS. These notes provide instructions for using AIPS remotely.

If the user wishes to run the pipeline on a remote computer, AIPS will need to be run remotely but using the local TV[13]. To accomplish this AIPS must be installed on the local computer. An AIPS installation and installation instructions can be obtained at `http://www.aoc.nrao.edu/aips/`.

Start the local copy of AIPS so that the TV, message server, and Tek server windows open. To accomplish this, any normal AIPS user number can be used, for example `101`. Then exit AIPS with the `EXIT` command.

Login to the remote machine using `ssh`. If AIPS has not been run remotely using this pair of client and host computers, it is necessary to be sure that the remote host can "see" the client. If the client has `ssh`-ed directly to the remote host, the environment variable `$REMOTEHOST` may contain the client IP address; otherwise, the command `who` may give the client IP address or host name. Use `ping` or `ssh` with the given IP address or host name to insure that the remote host can "see" the client. If the client computer has a firewall or is behind a firewall, access to TCP and UDP ports 5000-5017 from outside sources will have to be allowed; see the AIPS Managers FAQ for details, `http://www.aoc.nrao.edu/aips/aipsmgr/`.

Start AIPS on the remote computer using the following command:

```
aips tv=client tvok
```

Here, *client* is the IP address or host name of the client computer. A simple test of the functionality of

[13]X-forwarding may be able to display a remote TV on a local machine, but this will introduce additional lag.)

the AIPS TV can be done by typing TVINIT at the AIPS prompt; if no error messages are returned, then congratulations, the TV is working.

E Instructions and Procedures from Older Versions of the Pipeline

The following sections detail procedures that were used in previous versions of the VLA pipeline, but which are currently not necessary. These notes may be useful for investigating errors or interpreting old log files.

E.1 Image Validation and Archiving in AIPS

In the present version of the imaging pipeline, images are automatically placed in the public image archive after they are created. The user may then remove low quality images by following the process described in § 2.6. In older versions of the pipeline, image quality was verified in AIPS and images were exported to the image archive manually. The following discussion about validating and exporting images from AIPS is preserved here for posterity.

When Arch_Runs successfully finishes, images have been created for each unique source and center frequency pair in the data set. The user must check for image validity by starting AIPS with the appropriate user number (contained in the Arch_Runs results E-mail), and load each image.

When all images in a data set (under a single AIPS user number) have been validated by the user, the AIPS command NVASARCH should be manually executed. NVASARCH creates FITS[14] files for each image and places them in directory img/*source_name*/, where *source_name* is the source coordinate name. NVASARCH also stores the log files in subdirectory dat/*project_code*/, where *project_code* is the VLA project code.

Currently, AVLAPIPE is not configured to create mosaic images. Data that is intended to be made into a mosaic can be identified using AIPS task LISTR with adverb OPTYPE'SCAN'[15]: below the "Scan summary listing", LISTR will print the "Source summary"; if many sources listed here have very similar names and coordinates that are very close together (that is, the coordinates differ by less than a few degrees) the data is intended to be made into a mosaic. This data should not be exported under the current version of the pipeline.

[14]FITS = Flexible Image Transport System

E.2 Efficient Image Validation: SETPIX

In the present version of the pipeline direct access to AIPS is not required for image validation. These notes document an AIPS run file that was used to display images in earlier versions of the pipeline.

An AIPS run file has been written to ease the task of verifying pipeline images. This file is named SETPIX.001 and is stored in the CVS repository (see Glossary); SETPIX.001 must be copied to the bin directory of the pipeline working directory to be accessible, along with the other pipeline scripts, from within AIPS (see discussion of distrib.bash, below). This run file contains two AIPS procedures: LISTRSC and SETPIX. LISTRSC runs AIPS task LISTR on catalog file 1 with adverb OPTYPE set to 'SCAN'. SETPIX is an automated procedure that displays a set of images within a range specified by input parameters; SETPIX automatically adjusts the pixel gray scale range to allow for easier image validation. The command

```
SETPIX(10,15)
```

will display catalog images 10 through 15. SETPIX does this by first running AIPS verb IMSTAT to determine the minimum and maximum flux of the image. The range of flux to be displayed on the TV is then specified by setting AIPS adverb PIXRANGE(1) and PIXRANGE(2), which set the lower and upper bounds on the displayed flux, respectively. PIXRANGE(1) is set to the minimum image flux. PIXRANGE(2) is set to the larger of

$$\frac{\text{(flux maximum)}}{100} \quad \text{and} \quad \text{(flux minimum)} \times 5,$$

[15]The procedure LISTRSC can setup and run LISTR automatically. See § E.2.

unless this larger value is greater than the flux maximum, in which case `PIXRANGE(2)` is set to the flux maximum. After setting the displayed flux range, `SETPIX` runs AIPS verb `QHEADER` followed by `TVALL`.

A simple bash shell script has been written to copy the latest version of `SETPIX.001` to all pipeline working directories. This file is named `distrib.bash` and is stored in the CVS repository (see Glossary).

E.3 VLA Imaged Data Log: `VLA_LOG1.ods`

In the present version of the VLA pipeline the Loaded Data Log is maintained via a shell script that reads the `Arch_AIPS` *file on each computer and rewrites the data in a convenient format. In earlier versions of the pipeline, the log (then called the Imaged Data Log) was manually updated each time a project was loaded in the archive. This log still exists and contains information about projects that were successfully and unsuccessfully loaded and imaged in the pipeline at that time. The comments below are a guide to interpreting this log file.*

The VLA imaged data log is a spreadsheet file that exists for the purpose of maintaining a record of data files that have been imaged and exported, data files that are in the process of being imaged, data files that have been skipped, or data files that have failed to be imaged. The imaged data log also contains information on image quality, pipeline bugs, and pipeline improvements.

The data log is presently a spreadsheet maintained with OpenOffice.org Calc. OpenOffice.org is an open source office suite distributed by SUN Microsystems. Installation packages for a variety of operating systems can be obtained at `http://www.openoffice.org/`. Calc is the spreadsheet program within the OpenOffice.org suite.

The data log consists of 6 worksheets: 'Process', 'Success', 'Skip', 'Fail', 'Dates', and 'Fixes'. The first four worksheets contain the same column formats and hold information about VLA data files; each of these worksheets contains one row entry for each observing frequency in each VLA project processed with the pipeline. The 11 columns used in the first four worksheets are described in Table 2.

Table 2: VLA Imaged Data Log Column Headings

Column Heading	Description
Project Code	VLA project code
Date	Date of observation
Comp	Name of computer on which data was processe
AIPS #	AIPS user number used to process this data
File #	Run file extension number (see § 2.3)
Freq (Hz)	Observing center frequency
Status	Short imaging status description
Description	Long imaging status and image quality descrip
Comment Date	Date of last modification of entries in this sprea
Pipeline Issues	Comments to developers about pipeline bugs (evidenced in this data
Comments	Any additional comments about this data.

Worksheet 'Process' holds entries for data that are currently being imaged or have been imaged by the pipeline but have not yet been exported. Worksheet 'Success' holds entries for data that have successfully been imaged and exported. Worksheet 'Skip' holds entries for raw data that have been intentionally skipped, most often because the pipeline is not yet capable of imaging this type of data. Worksheet 'Fail' holds entries for data that the pipeline has unexpectedly failed to image; entries in 'Fail' are often only temporary since failures are usually followed by fixes or recognition that this type of data should presently be skipped.

Worksheet 'Dates' holds dates for which all observed data have been imaged and exported or skipped. This worksheet serves as a convenient log for using ArchQuery, which requires the user to specify a date and date range. To this end, the worksheet contains 5 columns for the 5 different subarray numbers that can be queried using ArchQuery. The user can thus keep a record of which dates have been processed for each subarray number.

Worksheet 'Fixes' holds a brief description of pipeline improvements (or fixes) along with the date of the improvement.

E.4 Efficient use of the Imaged Data Log

In addition to maintaining a log of imaged and skipped data files, imaged data characteristics, and pipeline improvements, the Imaged Data Log can be used to efficiently load data into the pipeline and validate the resulting image. For this purpose, each data file processed by `ArchPreps` can be recorded in the Imaged Data Log, in the "Process" worksheet. Each reserved AIPS number should be written to a separate row. Data files that `ArchPreps` fails to load should be cut from the "Process" worksheet and pasted into the "Skip" or "Fail" worksheet, as appropriate. What remains in the "Process" worksheet are the successfully loaded data files. This worksheet can be arranged or sorted at the users convenience to aid in image validation. When all images in a data file have been validated, the data file can be cut from "Process" and pasted into worksheet "Success".

Glossary of Pipeline Files

In the following file names, *`pcode`* is an abbreviation for *project code*, which, in the actual file name, will be replaced by the VLA project code of the corresponding data set; *`date`* will be replaced in the actual file name by the date of the observation; "`#`" will be be replaced by a numerical value that serves to differentiate this file from files that would otherwise have the same name.

`aips`

Link to AIPS directory on computer `auto`, `/home/auto/aips/`.
`client``:/`*`working_directory`*`/aips`

`Arch_AIPS`

ASCII file containing a list of each AIPS user number reserved by the pipeline in addition to the associated project code, unique run file number, and pipeline log file.
`client``:/`*`working_directory`*`/bin/Arch_AIPS`

`Arch_Cprt`

A copyright message placed in publicly available log files.
`client``:/`*`working_directory`*`/Arch_Cprt`

`Arch_Defs`

A C-shell script that defines environment variables used by the pipeline and calls AIPS setup file `LOGIN.CSH`. This file is also read by `ArchQuery` at run time.
`client``:/`*`working_directory`*`/Arch_Defs`

`Arch_Docs`

Short text file documentation of the pipeline.
`client``:/`*`working_directory`*`/Arch_Docs`

`Arch_Runs`

A C-shell script that looks for uninitiated pipeline run files in directory `run/`. All files found are marked as initiated by being given the extension `.auto`, or, when run file execution is complete, the extension `.aips`.
`client``:/`*`working_directory`*`/bin/Arch_Runs`

`ArchPreps`

Perl script that loads files listed in the most recent execution of `ArchQuery` into the VLA imaging pipeline. The user interactively confirms the files to be loaded at runtime. `ArchPreps` allows the user to specify the project code of interest as the command line option. When executed with the command line option "`-`", `ArchPreps` echoes the most recent `ArchQuery` results.
`client``:/`*`working_directory`*`/bin/ArchQuery`

`ArchQuery`

Perl script that queries the VLA archive for data within a given time range, the start of which is specified by a command line option. The subarray number to be queried is specified interactively by the user at run time. The range of time queried is controlled by the variable `interval` in file `Arch_Defs`, which has the form *`days`*`:`*`hours`* (2:12 = time range of 2 days and 12 hours); the observing bands queried are also specified in file `Arch_Defs` by variable `obsbands`.
`client``:/`*`working_directory`*`/bin/ArchQuery`

ArchSetup

A C-shell setup script that copies pipeline files and directories to the client computer, from which the pipeline will be run; links are also created on the client computer to files that are maintained on the pipeline server.
auto:/home/auto/ArchSetup

AVLAPIPE.001

AIPS run file containing AVLAPIPE and other associated procedures.
client:/*working_directory*/bin/AVLAPIPE.001

AVLAPIPE.HLP

AIPS help file for AIPS procedure AVLAPIPE.
client:/*working_directory*/bin/AVLAPIPE.HLP

AVLAPOST.001

AIPS run file containing AVLAPOST and associated procedures.
client:/*working_directory*/bin/AVLAPOST.001

AVLAPREP.001

AIPS run file containing AVLAPREP and associated procedures.
client:/*working_directory*/bin/AVLAPREP.001

bin/

Directory that holds shell scripts, perl scripts, AIPS run files, AIPS executables, and other executables used by the pipeline.
client:/*working_directory*/bin/

clean-vis

C-shell script that recovers disk space on each computer used for pipeline imaging. Disk space is recovered by removing all but the ten newest files from directory vis.
client:/home/auto/bin/clean-vis

dat/

Directory that contains *pcode_date*_#.data.log files; these files receive messages from shell and perl scripts stored in bin/.
client:/*working_directory*/dat/

distrib.bash

Bash shell script that copies file SETPIX.001 from its CVS working directory to the bin subdirectory of all pipeline working directories. The list of pipeline working directories in this file should be maintained separately by each user.
(CVS Repository)
filehost:/users/jcrossle/cvsroot/aips_run/distrib.ba

img/

Directory that contains FITS images.
client:/*working_directory*/img/

LOGIN.CSH

C-shell script that defines where the AIPS shell scripts are and sets the environment variable AIPS_ROOT so that AIPS is run from computer auto rather than from the local host.
client:/*working_directory*/aips/LOGIN.CSH

NVASDEFS.001

AIPS run file that initializes the inputs to AVLAPIPE. Execution requires "RUN NVASDEFS" within AIPS. Unlike other AIPS run files, for example AVLAPIPE.001, NVASDEFS.001 does not contain AIPS procedures and thus does not need to be executed like a procedure.
client:/*working_directory*/bin/NVASDEFS.001

pcode_date_#_#

Raw visibility data file downloaded from the VLA archive. In the actual file name, "#" is replaced by numbers that distinguish the file from those that would otherwise have the same name, for example, when there are multiple files with the same project code and same date.
client:/*working_directory*/vis/*pcode_date_#_#*

*pcode_date*_#.fill.log

Log file for C-shell script *pcode_date*_#.fill.run.
client:/*working_directory*/run/*pcode_date*_#.fill.log

pcode_date_#.fill.run

C-shell script created and executed by ArchPreps that loads downloaded data into AIPS and verifies that the data is acceptable for processing. Writes to log file *pcode_date_#*.fill.log
client:/*working_directory*/run/*pcode_date_#*.fill.run

run/

Directory that contains pipeline run files, and run file logs.
client:/*working_directory*/run/

SETPIX.001

AIPS run file containing procedures SETPIX and LISTRSC that aid in efficient validation of pipeline images. LISTRSC runs AIPS task LISTR on catalogue file 1 with OPTYPE set to 'SCAN'. SETPIX uses a simple algorithm to display a specified range of images with the optimal range of displayed flux; see § E.2 for details.
(CVS Repository)
filehost:/users/jcrossle/cvsroot/aips_run/SETPIX.001

tmp/

Directory that holds shell scripts generated and executed hourly by cron and log files of the cron scripts. Also contains the ArchQuery log file archfiles.dat that holds the full text of the archive query and archive response.
client:/*working_directory*/tmp/

uvf

Obsolete directory.
client:/*working_directory*/uvf/

vis/

Directory that holds raw (visibility) data downloaded from the VLA data archive.
client:/*working_directory*/vis/

web

Link to directory containing pipeline web page documentation files; this web page can be accessed via URL http://www.aoc.nrao.edu/∼vlbacald/src.shtml.
client:/*working_directory*/web