EVLA Computing Memo #51
Preparing the VLA Legacy Archive Data for New Archive Access Tool Ingestion

Bryan Butler
May 7, 2021

1. Introduction

Legacy VLA data (pre-EVLA upgrade, or prior to January 12, 2010) was written in EXPORT format. This format is explained in great detail in Hunt & Sowinski (1996); hereafter denoted "H&S". It has been delivered to our user community via the Archive Access Tool (AAT) since ~2000. This online tool, developed and maintained mostly by John Benson, has been a fantastic resource for our users, and served NRAO well. The AAT also delivers GBT and VLBA data, but I will not discuss those data in this document. Despite the utility of the AAT, it has been recognized for many years that it needs an updated look-and-feel, updated functionality, and updated architecture, which fits more into the rest of the software in the observatory. The replacement for the AAT, which I'll call the New Archive Access Tool (NAAT, to distinguish from the existing AAT; the N can stand for "NRAO" or "New" as you prefer) has been in development for several years. It currently supports ALMA and post-upgrade VLA data, but needs to also support legacy (pre-upgrade) VLA data. This document describes the details of preparing the legacy VLA data for ingestion into the NAAT.

One might ask why the legacy VLA data has to be "prepared" at all? After all, we have been serving up this data from the AAT for almost twenty years now. The reasons are two-fold: 1 – the existing data that is in the AAT database has individual files that are corrupt, and also missing files; and 2 – the organization of the data is not ideal. For this second point, consider that the data was originally written on tapes, and the fundamental files in the archive are one-to-one matches with these tape files. From 1976Jul13-1987Dec31 these individual files are a maximum of ~35 MB of data (corresponding to how much data could fit on a 9-track tape at 1600 bpi density, which is what the original Modcomp computers wrote); from 1988Jan01-2003Sep06 they are a maximum of ~155 MB of data (corresponding to how much data could fit on a 9-track tape at 6250 bpi density, which is what the upgraded Modcomp computers wrote). From 2003Sep07-2010Jan11 a change was made in how data was recorded at the VLA – we moved from recording on 9-track tapes to recording on EXABYTE or DAT tapes, with one day's worth of data recorded on each tape, and eventually to recording directly to disk, again with one day's worth of data recorded in each file.

There is no organization of the raw EXPORT data files beyond these files (for any of the eras), so an observation can span many files. A much better organization is to have one file per observation. What I mean by an "observation" is one contiguous set of data records for a single subarray in the EXPORT files (i.e., the data coming from a single OBSERVE file). It is akin to a Scheduling Block (SB) in our modern parlance. I don't call it an SB because we never used that term during the legacy VLA days (well, until ALMA planning started happening around 2000). That organization is much cleaner in terms of access from the NAAT software.

This document describes how I converted the legacy VLA EXPORT data files from their format in the AAT (tape files and one-per-day files) to the format to be used in the NAAT (one-per-observation files). It also describes errors I found in the archive files and how I fixed them. Note that I'm not writing this for a general audience, but rather for NRAO employees, so I take some liberty in usage of jargon.

2. Retrieving the raw EXPORT files from the AAT database

The raw legacy VLA EXPORT files are stored on a lustre filesystem, with NGAS layered over it. Outside of using the AAT itself to pull them out (which would be extremely painful), there are two ways to get such a file out of NGAS: 1 – John Benson provided me with instructions on how to use the *ngamsCClient* program to do it; 2 – Stephan Witz showed me how to use *wget* to do it (*curl* could also be used; I chose *wget*). In the end, I used method (1) as it was more reliable. For either of these, however, you have to know the name of the file you want to retrieve. Fortunately, there is a naming convention for these files. From 1976Jul13-2003Sep06, the convention is: VLA_XH**YYMMM**_file**NN**.dat, where **YY** is the last two digits of the year, **MMM** is a three digit EXABYTE tape number (left-padded with zeroes), and **NN** is a two digit 9-track tape number (left-padded with zeroes). The EXABYTE tape number is an artifact from when the 9-track tapes were copied over to EXABYTE tapes in 19XX; multiple 9-track tapes were copied onto a single EXABYTE tape. From 2003Sep07-2010Jan11, the convention is: vla**YYYY-MM-DD**.dat, where **YYYY** is the year, **MM** is the month number (left-padded with zeroes), and **DD** is the day of the month (left-padded with zeroes).

I made a python script which loops over the years, and retrieves the files, using the above information. Note that I did all of the scripting described in this document in either python or perl.

For the 1976Jul13-2003Sep06 period, the difficulty is that you don't know in advance how many tape numbers there are, or how many files for a given tape, in a given year. To address that, I simply loop through tape numbers, and retrieve file numbers until there is a failure (at which point I increment the tape number). When there is a failure on file number 1 for a tape number, I know I'm done with that year.

There is a known failure mode for this technique, however. Say file number N is missing from the archive (there are missing files, because when the 9-track tapes were copied to the EXABYTE tapes, some tapes were corrupt and could not be read at all). In that case, file numbers N+1, N+2, … will not be retrieved. Even worse, say file number 1 is missing – in that case the script will think that the entire year is done, and will not try to retrieve the files for higher tape numbers. There are some sanity checks to try to test for these conditions. For instance, in the period 1976-1987, up to 40 9-track tape files could be fit on an EXABYTE tape, so most tape numbers should have about that many files. In the period 1988-2003, that number was 17. Visual inspection of the retrieved set of files indicated some other problem areas (see below), which I fixed by hand. Finally, I got a list of all of the files in NGAS from this period from Daniel Lyons, and compared it to the list of retrieved files. That process found 7 missed files from the above process, which I retrieved by hand.

For the 2003Sep07-2010Jan11 period, things are much simpler, since you can just loop over years, months, and days, and retrieve the file for each day.

3. Problems with the retrieved files

There were a number of problems with the retrieved files, which I found in a variety of ways (much of it was just by hand inspection). I'll describe each of these, and how I dealt with them.

a. Missing data in April 1989

In April 1989, the following files were missing:

```
VLA_XH89016_file9.dat     89-Apr-12 15:13:10  to  89-Apr-13 10:47:40
...
VLA_XH89017_file1.dat  \
VLA_XH89017_file2.dat  | 89-Apr-15 22:54:49  to  89-Apr-18 10:02:59
VLA_XH89017_file3.dat  /
VLA_XH89017_file5.dat  \
VLA_XH89017_file6.dat   \
VLA_XH89017_file7.dat    \
VLA_XH89017_file8.dat     \
VLA_XH89017_file9.dat     | 89-Apr-18 13:34:00  to  89-Apr-26 06:42:35
VLA_XH89017_file10.dat    /
VLA_XH89017_file11.dat   /
VLA_XH89017_file12.dat  /
VLA_XH89018_file1.dat  /
VLA_XH89018_file6.dat  89-Apr-27 15:53:40  to  89-Apr-28 02:40:09
```

My original idea was to go to the tape vault to see if I could find these tapes. When I looked, however, I found those physical tapes missing! In one of the strangest

coincidences that has ever happened to me, that same day I went to talk to Juan Cordova about a different topic. Turned out these tapes were sitting on his desk. He said that when John Benson had moved offices, he found the tapes in a box, and brought them to Juan for some reason (Juan is the media specialist, but that is really for VLBA media)! The other 2 are: XH96055, XH99080, and it turned out that, sure enough, there were also files missing from those two tapes. So, I recovered the files from those tapes and put them in the right place. I put a description of how to do this in Appendix A, just so if for some reason anybody ever has to go back to the EXABYTE tapes, they have instructions on how to do it.

b. Other missing data requiring retrieval from the original tapes

In the course of all of my data file examination I found, much to my surprise, a number of files that were duplicates of other files in the existing archive. There were 38 such files, on 7 tapes. When trying to retrieve from tape, I found that some of these duplicate files did not exist on the original tapes; they were likely simply erroneously created as copies of other files during the transcription process from tape to disk. I did find 21 of these files, and retrieved them from the tapes:

VLA_XH01053_file7.dat
VLA_XH01053_file8.dat
VLA_XH01053_file9.dat
VLA_XH01053_file10.dat
VLA_XH01053_file11.dat
VLA_XH01056_file11.dat
VLA_XH01058_file3.dat
VLA_XH01058_file4.dat
VLA_XH01058_file5.dat
VLA_XH01058_file8.dat
VLA_XH01058_file9.dat
VLA_XH01058_file10.dat
VLA_XH01058_file11.dat
VLA_XH01060_file4.dat
VLA_XH01060_file11.dat
VLA_XH01105_file4.dat
VLA_XH01117_file2.dat
VLA_XH01117_file3.dat
VLA_XH01117_file11.dat
VLA_XH01117_file12.dat
VLA_XH03030_file4.dat

c. Missing data in 2003

My guess is that much of this was due to the change from writing data to 9-track tape to writing it to DAT tape, which formally happened starting on 2003Sep07 (so files were written one-per-day instead of one-per-tape, and the file naming convention described above happened).

Most of the period 2003Aug17-2003Sep06 was missing. In addition, files for 2003Sep09, 2003Sep23, and 2003Oct24 were missing. In looking in the AAT, it turned out that these files had a different naming convention, like: VLA_XH03104_file7.uvfits.dat. Stephan Witz investigated, and found that there were 292 files in 2003 that were named like this, but many of them have times that overlap with other files (notably, many are after 2003Sep07, when the change to the new file naming convention was made). 2003Aug17 starts in file VLA_XH03104_file1.dat, so I ignored files prior to that. I need to only go to 2003Sep07 00:00 for that missing chunk of data, which is in the middle of VLA_XH03116_file11.dat. So I retained all of the files up to that one, and made a special script to trim all of the data beyond 2003Sep07 00:00 from VLA_XH03116_file11.dat. That fixed up that chunk of missing data.

The other three dates were a bit trickier. 2003Sep09 is MJD 52891. MJD Start and stop times for the appropriate files:

VLA_XH03118_file1.dat 52890.702911 52891.283891
VLA_XH03118_file2.dat 52891.284007 52891.747356
VLA_XH03118_file3.dat 52891.747357 52892.054300

I needed to strip out the 2003Sep08 stuff from VLA_XH03118_file1.dat, then APPEND VLA_XH03118_file2.dat to that, then strip out the 2003Sep10 stuff from VLA_XH03118_file3.dat and append it. Did that with the scripted mentioned above, and put the result in vla2003-09-09.dat. Now 2003Sep23, which is MJD 52905:

VLA_XH03123_file11.dat 52904.893498 52905.186495
VLA_XH03124_file1.dat 52905.186611 52905.238232
VLA_XH03124_file2.dat 52905.238347 52905.289968
VLA_XH03124_file3.dat 52905.290083 52905.341357
VLA_XH03124_file4.dat 52905.341472 52905.393556
VLA_XH03124_file5.dat 52905.393671 52905.505882
VLA_XH03124_file6.dat 52905.505940 52905.604493
VLA_XH03124_file7.dat 52905.955188 52906.697549

Similar to how I did 2003Sep09, I needed to strip data before 2003Sep23 out of VLA_XH03123_file11.dat, strip data after 2003Sep23 out of VLA_XH03124_file7.dat, then concatenate them all into vla2003-09-23.dat. Finally, 2003Oct24, which is MJD 52936:

VLA_XH03141_file2.dat 52935.862093 52936.060205
VLA_XH03141_file3.dat 52936.060321 52936.645622

VLA_XH03141_file4.dat 52936.645737 52937.081485

I did the same thing here, to produce vla2003-10-24.dat.

d. Other missing files in the 1976Jul13-2003Sep06 era

These were found as a result of the investigation described above, where I compared the list that Daniel Lyons pulled out of the AAT database to what I had retrieved via script. In 1996, there were two files missing from my repository, VLA_XH96055_file9.dat and VLA_XH96055_file10.dat. Those were retrieved and added. But missing in the AAT database are files 2 through 8. The missing date/time range is: 1996-08-07 15:17 through 1996-08-10 17:44; I don't know what projects that covers. It is unclear why these files are missing, but they may have been bad 9-track tapes that couldn't be read. In 1999, there were five files missing from my repository: VLA_XH99080_file8.dat, VLA_XH99080_file9.dat, VLA_XH99080_file10.dat, VLA_XH99080_file11.dat, and VLA_XH99080_file12.dat. Those were retrieved and added. But missing in the AAT is file 7. Interestingly, there is no gap in time between files 6 and 8; no idea what happened there.

e. Other missing files in the 2003Sep07-2010Jan11 era

Other missing data in the 2003Sep07-2010Jan11 era are:

vla2005-02-15.dat (period 2005-02-14 14:32 to 2005-02-16 19:38)

vla2006-01-01.dat (entire day)

vla2006-06-22.dat (entire day)

For the 2005Feb15 data, it appears that the array was down and not taking data at all. From the operator's log webpage:

Date	Time	Code	File
2005-02-17	01:27	MAINT	pdf
2005-02-14	12:32	AR545	pdf
2005-02-14	02:04	AP452	pdf
2005-02-14	00:04	AC789	pdf
2005-02-13	23:05	AB1156	pdf
2005-02-13	17:05	BD103	pdf
2005-02-13	14:06	AH869	pdf
2005-02-13	05:08	AD506	pdf
2005-02-13	04:08	AR545	pdf

So it looks like there is simply nothing there. The 2006Jan01 data looks the same. There is a gap in logs, so it's likely that no observing was happening.

The 2006Jun22 data is different. Stephan Witz found the following files in NGAS: vla2006-06-22.AA304.dat, vla2006-06-22.AG715.dat, vla2006-06-22.AM868.dat, vla2006-06-22.AR604.dat. They have the following timeranges:

vla2006-06-22.AA304.dat - 2006-06-22 04:10:17.156 to 2006-06-22 09:38:51.323

vla2006-06-22.AG715.dat - 2006-06-23 04:36:17.156 to 2006-06-23 09:35:07.156

vla2006-06-22.AM868.dat - 2006-06-22 23:39:28.927 to 2006-06-23 04:35:58.927
vla2006-06-22.AR604.dat - 2006-06-22 09:41:37.208 to 2006-06-23 12:12:01.844

vla2006-06-21 goes right to the end of the previous day. After that the sequence in time is:

1: vla2006-06-22.AA304.dat
2: vla2006-06-22.AR604.dat
3: vla2006-06-22.AM868.dat

And vla2006-06-22.AG715.dat isn't even on 2006Jun22. vla2006-06-23.dat starts right at the beginning of that day. So periods of missing data are 00:00 to 04:10 and 12:12 to 23:39 (between AR604 and AM868). There is really nothing to be done about that. Fortunately, no science observing was occurring then; the operator logs from that day show that in the first missing period, START was running (startup after a maintenance day), and in the second missing period, SOFTW was running (software work). To make the final file for that day, take vla2006-06-22.AA304.dat + vla2006-06-22.AR604.dat + the portion of vla2006-06-22.AM868.dat on June 22, using the script noted above, and put them into vla2006-06-22.dat.

f. Zero size files

There were two files that had zero file size: vla2009-12-25.dat and vla2010-01-12.dat. The first is during the Christmas shutdown, and the operator logs show no observing that day. The second was after we had transitioned to automatic ingestion of data into NGAS (post-EVLA upgrade). I deleted both of these files; I'm not sure why they got created in the first place.

g. Files that have corrupt data in them

There are many files with corrupt data in them. Attempts to parse them without care fail, because following the various pointers into different areas, and reading data in the expected amounts, will cause you to end up in random places in the file, and then you're stuck. Per-year, I found the following numbers of such files: 1976: 1; 1981: 1; 1982: 4; 1983: 16; 1984: 39; 1985: 23; 1986: 35; 1987: 11; 1988: 29; 1989: 9; 1990: 5; 1992: 4; 1993: 1; 1994: 5; 1995: 3; 1996: 8; 1997: 35; 1998: 22; 1999: 6; 2000: 11; 2001: 5; 2002: 4; 2003: 9; 2004: 11; 2005: 18; 2006: 10; 2007: 9; 2008: 9; 2009: 1. This is a total of 344 files, which is not bad out of 30775 total files (about 1%). Even more nefariously, there are two files in that group that have pointers which point backwards in the file, but land on a record boundary! I made a script to repair these files. The script checks a number of variables in the RCA area of the EXPORT file, and if any of them are outside their expected value, the read pointer in the file is advanced one byte at a time until a valid RCA record is found. This was run on all of the corrupt files to fix them.

h. Duplicated data at year boundaries

In the process of stitching the years together (see discussion below), I found many instances of duplicated data. I don't see a similar situation with the Thanksgiving or Christmas shutdowns, so I don't think it's directly related to the New Year's Day shutdown, but is rather some combination of that plus the year boundary plus what the operators were doing with tapes at those times. There were four instances of files that had data that was entirely duplicated in another file (either the preceding or succeeding file), and four other instances of files that were partially duplicated. I made modifications to those latter four, to avoid overlaps in time in the data.

i. Other problems found with the data in the files

There are a number of small problems with actual data in the files that I found as I read through them, which I had to put checks and fixes for in the final conversion script. I list these here.

i. Corrupt data in SDA string values

The "Observing Program ID" (as called in H&S) in the SDA is an ASCII string with six characters. Normally there is no leading whitespace. However, there is at least one example where there is a leading space in this field. Furthermore, there exist Observing Program IDs that contain a "/" in them – this is a problem in dealing with such a file because of disk directory structure and the file naming convention described below. In those cases, I have replaced the "/" with a "_". Finally, in some records, there are non-printable characters as part of this string. The ones I found when I discovered this problem were all 0x00, or all bits = 0, but there may be different values. Additional variables in the SDA that are of interest for the NAAT metadata (see below) that have the same problem (non-printable ASCII values in string variables) are: "Source Name" (sixteen character ASCII string); "Correlator Mode" (four character ASCII string); "Observing Mode" (two character ASCII string), and "Array Processor Options" (four character ASCII string, though only the first three characters are used). These corrupt string values are almost certainly a casualty of the copying of the 9-track tapes to EXABYTE tapes in the 1990's.

ii. Bad data in source coordinate

The "Source RA, at standard epoch (Radians)" and "Source Dec at standard epoch (Radians)" (as called in H&S) in the SDA are double precision values. In some records these are non-sensical. For example, a right ascension that is greater than 2π or less than 0, or a declination that is less than $-\pi$ or greater than π.

j. Program code conventions and oddities

Program codes went through various conventions throughout the years, and there was little enforcement of what went into the OBSERVE files and hence ended up in the EXPORT files, even in later years. In early days (prior to 1982) program codes

were simply abbreviations of last names, or other convenient labels. In later years, the convention of using the first letter 'A', then the first letter of the last name of the PI, then a proposal number, was used, for VLA projects. However, even this was not strictly enforced, so you could have one project, say AB123, that might have a program code in the EXPORT file of 'AB123' in one observation, 'AB0123' in another, and 'AB 123' in yet a third (note the space). I decided to try to regularize all of these by stripping out spaces, and stripping out leading (padded) 0's in program codes, so there should be no program codes like 'AB0123' in the new archive – only 'AB123'. This may lead to some irregularities, but I believe it is a better situation than the present. Valid science codes, after those early years, were of the form: 'Z[A-Z][0-9]+' (using regexp notation) where Z was one of: 'A' (VLA – 1982-2010); 'B' (VLBA – 1990-2010); 'G' (Global VLBI – 1990-2009); 'S' (joint spacecraft – 2003-2009); 'U' (other VLBI – 1990-1993); 'V' (VLBI network – 1982-1998); 'W' (Halca – 1999-2002).

There were 37 program codes that were simply blank in the entire observation. For these, I manually inspected the data and tried to match it to a science program, if appropriate (16 were of this sort), or a related test program (I could usually tell because the sources and bands would be the same in surrounding data, the program code just somehow got blanked).

There are a number of program codes that start with a number. Particularly in the era from 1992-1997 there are a number with a prepended '0', so, for instance, instead of 'AD303' the program code is '0AD303'. There was a reason for doing this at the time, but nobody can remember what it was. There are other instances where the letters and numbers of the program code were transposed, so, for instance, the program code is '79AM' instead of 'AM79'. There are others that I'm not sure how the ended up like they are. For all of these I either fixed them, or manually inspected (as for the blank program codes described above) to get a reasonable program code.

There are program codes that follow the naming convention above (or close to), but are clearly not related to a project with that code. An example is AQ1159 – the highest proposal number for AQ was AQ13! Other examples are things like 'AH0000', 'AX3C28', 'AX', etc. I could not think of a rational way to catch all of them, so left them as-is.

There is one program code of '<666>' which I left as that (Kevin Marvel, I'm looking at you!).

4. Characteristics of the resulting dataset

The final legacy VLA archive input dataset comprises 30,775 files with total volume ~5.37 TB. Table 1 shows the breakdown by year.

Table 1. Number of files and data volume by year for the legacy VLA archive input dataset.

Year	Number of files	Data volume (MB)
1976	17	577
1977	92	3263
1978	160	5986
1979	235	9006
1980	428	15121
1981	821	27027
1982	952	30864
1983	1333	41111
1984	1460	43304
1985	1472	45421
1986	1632	51918
1987	1503	47225
1988	588	63940
1989	646	88015
1990	637	89680
1991	814	114664
1992	924	129859
1993	910	137006
1994	1048	153081
1995	1095	169837
1996	1071	161925
1997	1161	172266
1998	1420	218457
1999	1266	192299
2000	1890	295624
2001	1720	267110
2002	1851	290618
2003	1434	324487
2004	365	276864
2005	363	271071
2006	363	243774
2007	364	445773
2008	365	521640
2009	364	391745
2010	11	23786
Total	30775	5365343

5. Converting the files

Here I describe the various elements of the python script that I wrote to convert the input dataset to the output dataset (the output dataset being per-observation).

a. Metadata

As the data files are processed, certain metadata needs to be stored in the program, per subarray. This is to enable the check for a change in observation (by checking Observing Program ID), the check for a scan change (see definition below), and other checks. In addition, since I'm crawling all of the data, I may as well keep track of metadata that might be of interest to the NAAT, and write it to a file that can be used during the NGAS ingest process to populate the metadata database that the NAAT will use to enable searches. This could be done on-the-fly, but would significantly lengthen that ingest process. In fact, some of the metadata could be retrieved on-the-fly by the NAAT as searches are requested, but, again, it would slow down the search process, which would annoy users. As a result, the metadata I keep track of and write is a combination of various values displayed and allowed to be searched on in both the AAT and the NAAT (for current VLA or ALMA data). I made the decision to write the observation metadata as an XML file, because I'm familiar with XML, and it's well-supported in almost every programming language, so should be easy to parse. I have an XML schema (an .xsd file) that can be used to validate the XML if desired. The final list of metadata is:

- Observing Program ID (a)
- Observing "Segment" (b)
- Configuration (a)
- Observer Name (b)
- Subarray ID (a)
- Observation Start Time (MJD UTC)
- Observation End Time (MJD UTC)
- Input File Names (c)
- Associated Raw (EXPORT) Data File Name (b)
- Associated Raw (EXPORT) Data File Size (bytes)
- Associated Subarray Metadata File Names (b)
- Observing Bands (b)
- Scans
    - Scan Number
    - Source Name (a)
    - Start Time (MJD UTC)
    - End Time (MJD UTC)
    - Integration (Dump) Time (LST seconds)
    - Observing Mode (a)
    - Correlator Mode (a)
    - Number of Antennas (a)
    - Source RA at Standard Epoch (radians) (a)
    - Source Dec at Standard Epoch (radians) (a)
    - Source RA J2000 (radians) (b)
    - Source Dec J2000 (radians) (b)
    - Epoch year (a)
    - Array Processor Options (a)
    - Spectral Windows
        - Center Frequency (GHz) (b)

Bandwidth (MHz) (b)
Number of complex frequency channels per baseline for each CDA (a)
Polarizations (b)

(a) Taken directly from the input data, as defined in H&S.
(b) See relevant section below on derivation of this quantity.
(c) This is the list of files that the observation came from in the legacy VLA input dataset.

For the detailed definitions and allowed values for most of the above quantities, please see H&S.

b. Converting Modcomp float & double formats

The Modcomp computers (both the original and upgraded) wrote floats and doubles in a non-IEEE format. This is described in detail in H&S, Appendix B. How I convert these to IEEE is shown in the python code in Appendix B.

c. Converting Modcomp "nibbles"

A nibble is 4 binary digits (bits), or half of an 8-bit byte. In the EXPORT files there are two quantities of interest that are defined this way: "Number of complex frequency channels per baseline for each CDA," and "Bandwidth Codes for IFs A-D." The former is explicitly part of the metadata listed above; the latter is used in the calculation of Bandwidth, for revisions 20-23 (see description below). The four values (for IFs A-D) are packed into two bytes. I used the standard method of masking off bits and converting to integers to extract these values.

d. Observer Name

In many observations, especially early on, the "Observer's AIPS Number" was filled in in the OBSERVE file, and is available in the SDA. The observer's name can then be pulled from the USERNO.LIS file in the standard AIPS distribution. For many observations, however, this value is 0, meaning it wasn't provided in the OBSERVE file. In those cases, I use a list provided by Barry Clark that has every legacy VLA observing project's PI name. Some observations (notably tests and operations observations) will not be covered by this, so I define the Observer Name as "Unknown."

e. Observing Bands

Observing bands (4, L, C, X, Ku, K, Q, for legacy VLA) are not listed explicitly in the SDA. I determine the observing bands for an observation by cycling through all of the spectral windows, and checking the center frequency. I use the band edges defined in FILLM in AIPS, but expanded somewhat (because I found them too restrictive).

f. VLBA segments

VLBA observations, both Y27 and Y1, often had an observing "segment" associated with them, appended to the end of the Observing Program ID, which

incremented from “a” to “b”, etc. These have to be trapped, and the segment extracted and removed from the Observing Program ID. I use a regex to trap this: "^(B[A-Z][0-9]+)".

g. Subarrays

When the array was being used for a single observation, but broken into subarrays, one EXPORT file and associated metadata file will be written for each subarray. These individual subarray files are associated, however, and can be identified by inspecting the program ID and the start and stop times of those individual subarray files. I have made a script to do just that, and put the results into the Associated Subarray Metadata File Names element of the XML file. So, if there are 3 subarrays, then the metadata file for subarray 1 will have the names of the metadata files for subarrays 2 and 3, the metadata file for subarray 2 will have the names of the metadata files for subarrays 1 and 3, and the metadata file for subarray 3 will have the names of the metadata files for subarrays 1 and 2.

h. Coordinate conversions

There are two coordinates stored in the SDA: coordinates of “epoch,” and coordinates of “date.” The epoch is stored in the SDA in the Epoch year quantity, though that quantity was only added in revision 10. There are four possible values: 0 (this is only true for revisions 1-9); -1 (indicates an “of date” coordinate, which was used for observations of moving objects); 1950 (B1950.0); and 2000 (J2000.0). For the 0 values, there are really only two possibilities: B1950.0 or “of date.” Which it is can be determined by whether the coordinates of epoch and coordinates of date are different or the same. For consistency with post-upgrade VLA data, it seems advisable to calculate J2000.0 positions for all scans, and store them in the metadata, although I’m retaining the epoch coordinates in the metadata so we could decide to stick with those if desired. I do this conversion using the Astropy SkyCoord class. For conversions I use the SkyCoord frame ‘icrs’ for J2000.0. For B1950.0 I use ‘fk4’ and for “of date” I use ‘cirs’ (see Figure 2 in the SOFA reference for the definition of this frame).

i. Time conversions

Times in the SDA are all IAT, as this was the fundamental timeframe in the Modcomp online system. For consistency with post-upgrade VLA data, it seems advisable to convert these times to UTC, which is the fundamental timeframe in the new online system, and what is stored in the SDMs and BDFs. I do these conversions with the Astropy Time class. I add the “Date (MJAD)” value from the RCA, to get full MJD UTC date/time.

j. Channel width problem

As noted in H&S, revisions 20-23 in the archive have various issues with recorded channel widths (the “Channel separation codes for IFs A-D (the k in 50MHz/2**k)” field). For these revisions, the channel width is re-calculated from the bandwidth code (“Bandwidth codes for IFs A-D”), with additional information about whether Hanning smoothing was used or not (from the Array Processor

Options field) and the number of IFs (which can be deduced from the correlator mode). Given a bandwidth code value b_i for each bandwidth code i, and a Hanning value, h, which is 0 if Hanning smoothing was not used, and 1 if it was, and a number of IFs, n_{IF} the channel width cw can be re-calculated:

$$cw = \frac{6.25\ n_{IF}\ 2^h}{2^{b_i}}$$

For bandwidth codes 5 and less, $b_i = 2\,i$. For bandwidth code 6, $b_i = 11$, for bandwidth code 8, $b_i = 12$, and for bandwidth code 9, $b_i = 13$. The number of IFs is 4 if the Correlator Mode starts with "4" or "P", 2 if the Correlator Mode starts with "2", and 1 if the Correlator Mode starts with "1".

k. Channel offset problem and center frequencies

With the legacy VLA, it was possible to select a channel offset for recorded data (in the Offset card in the OBSERVE file). When this was selected, the center frequencies recorded in the "Sky Frequency at Band Center (channel 0) for IFs A-D (GHz)" field of the SDA are not correct, and they must be re-calculated. This is done using information in the "Signed sum of LOs for IFs A-D (GHz)" field, the "Channel offsets for IFs A-D" field, the "Number of complex frequency channels per baseline for each CDA" field, and the "Channel separation codes for IFs A-D (the k in 50MHz/2**k)" field (or the re-calculated channel width described above. Given that information, the center frequency for each IF is re-calculated via:

$$f_{sky} = f_{SSLO} + \frac{n_{offset} + {}^{n_{chan}}/_{2}}{cw}$$

Note that for revision 20 (dates from 88Jan02 to 89Jan18) channel offsets are identically 0 in the EXPORT files. The only way to try to remedy this situation would be to recover the original OBSERVE files (see discussion below), but it doesn't seem worth the effort.

l. Polarizations

The polarizations present in a spectral window can be determined from the Correlator Mode. The legacy VLA could observe either in continuum or spectral line mode. I have chosen to define continuum observations as having two spectral windows, with polarizations RR, LL, RL, and LR. For spectral line observations, the mapping between IFs and polarizations is that IFs A and B were RR, IFs C and D were LL. For spectral line observations with a Correlator Mode that starts with "P," all four polarizations are present: RR, LL, RL, and LR. The second character of these modes defines only which IFs were used (AC or BD), and the distinction is unimportant as far as defining the spectral window is concerned. For spectral line observations with a Correlator Mode that starts with "1," the second character of the Correlator Mode defines the polarization from the above IF-to-polarization mapping, e.g., for Correlator Mode "1A" the polarization is RR. For spectral line observations with a Correlator Mode that starts with "2," the two polarizations are defined by the second and third characters of the Correlator Mode, e.g., for Correlator Mode "2AC" the polarizations are RR and LL, and for Correlator Mode "2AB" the only polarization present is RR.

m. Defining scan boundaries

Because of the way the observing system worked for the legacy VLA, where "scans" that were defined in the OBSERVE file (the Source cards) didn't actually correspond to what happened on the array, and hence went into the EXPORT file. This is because the operator or staff running the observation could go back and forth in the OBSERVE file, so interpreting when there is a new "scan" can be tricky. I define a new scan as satisfying three conditions:

1. The "Start Time (LST radians)" field in the SDA changed;
2. This is not a subscan. For scans that contained submodes (effectively subscans in the new parlance), which includes raster modes, pointing modes, etc., the value in item 1 above changes, but it's not really a new "scan;"
3. Important information changed. I define "important" information as anything contained in the above-defined metadata.

n. Output file naming convention

The per-observation EXPORT and metadata files are named with the convention:

ObservingProgramID_subarray_MJDstart_MJDend.exp

where the "exp" is replaced with "xml" for the metadata files. MJD_start and MJD_end have fractions, with five digits after the decimal (to be accurate to seconds). An example is: AG566_1_57934.77672_57934.94258.exp.

o. Script structure and dealing with the whole dataset

The script is designed to deal with all EXPORT files in the current directory. It first makes a list of all of them, then determines the start time of each of them, then treats them in time order. This is critical, of course, because observations cross file boundaries. As it processes each record, it fills the metadata information in for each subarray, and writes the record to an output file. When the Observing Program ID changes for a subarray, the output file is closed, and renamed (I have to wait for the last record to know what the final time is), and the metadata XML file is written. When the last record of the last file is read in, all the subarrays with active data are dealt with (file closed, renamed, and metadata XML file written).

I kept all of the data in directories by year and processed them one at a time, then stitched the boundaries between the years together by hand. In principle, I could have done it all in a single directory, but that directory would have had more than 100,000 files in the end, and filesystem operations become cumbersome in that situation.

There is the main script that does the bulk conversion (*convert_export_files.py*), the script that does the subarray association finding and metadata editing (*fix_subarrays.pl*), the scripts mentioned above for retrieving the EXPORT files from NGAS, the script for repairing EXPORT files (*repair_one_export_file.py* – which can also be used to only copy certain record numbers), and any number of utility python and perl scripts to parse and print EXPORT files, check various things, driver files for bulk repair, etc. All of these are available upon request.

6. Associated files

Each "observation" has a number of associated files, which we could ingest into the archive and then serve up with the NAAT. The main three associated files are: 1 – the OBSERVE file which drove the observation; 2 – the operator log of the observation; 3 – the proposal for the overall project. Not all observations have all of these; for instance, test observations do not have an associated proposal. Furthermore, we don't have access to all of this data for all observations. We have all of the science proposals digitized now (that was done after we sent them to CV), but it is the full proposal, and only the cover sheets are public, so those cover sheets would have to be stripped out. We have operator logs going back to 2003, but prior to that we only have the paper copies. For OBSERVE files, Ken Sowinski made a backup on disk of all OBSERVE files from 1988-2010, but there are many missing, and there would need to be some code written to link those OBSERVE files with the observation files.

This effort is beyond the scope of what I have done to date with the VLA legacy archive EXPORT files, but I feel like we should not lose sight of it, and in the future it may be worth pursuing getting these associated files into the archive, and able to be served up from the NAAT.

Acknowledgements

I couldn't have written this memo or done this work without the expert input of Ken Sowinski. Both from his memo 188 (with Gareth Hunt), and from me pestering him with innumerable questions when I ran across things I didn't understand in what I saw in the data. His depth of understanding on the topic is unmatched. I also could not have done the work without the ability to retrieve the raw EXPORT files from NGAS, provided by John Benson and Stephan Witz. And, lastly, I couldn't have retrieved the files I needed from the ExaByte tapes without Schlake's help.

References

Hunt, G. C., and K. P. Sowinski, VLA Archive Data Format, VLA Computer Memo 188, 1996.

IAU SOFA Board, SOFA Tools for Earth Attitude, 2019.

Appendix A – Extracting files from EXABYTE tapes

The VLA EXPORT EXABYTE tapes are organized like:

1. 80-byte ASCII volume header file
2. First data file
3. Second data file
4. etc.

Each data file is comprised of a number of 26624-byte blocks. If you set the blocksize on the tape device to variable, and then retrieve the files using *dd*, with an input block size that is larger than 26624, you don't have to specify that value precisely. To read the files off of one of the tapes:

1. Get CIS (Schlake) to put one of the old EXABYTE readers on a public machine.
2. Load the tape into the drive.
3. Set the block size to variable:
   phecda% mt -f /dev/nst0 setblk 0
4. Get the header:
   phecda% dd if=/dev/nst0 of=header.txt ibs=27k
5. Read the EXPORT files:
   phecda% dd if=/dev/nst0 of=file1.dat ibs=27k
   phecda% dd if=/dev/nst0 of=file2.dat ibs=27k
   phecda% dd if=/dev/nst0 of=file3.dat ibs=27k
   ...

Eventually you'll get a message that 0 bytes were read:

```
phecda% dd if=/dev/nst0 of=file15.dat ibs=27k
0+0 records in
0+0 records out
0 bytes (0 B) copied, 0.00808578 s, 0.0 kB/s
```

which means you're at EOT.

If you encounter an error on tape (e.g., file 1 of XH89017), like:

```
phecda% dd if=/dev/nst0 of=file1.dat ibs=27k
dd: reading `/dev/nst0': Input/output error
0+2374 records in
113952+0 records out
58343424 bytes (58 MB) copied, 257.874 s, 226 kB/s
```

then you have to rewind and skip over that file and start from there:

```
phecda% mt -f /dev/nst0 rewind
phecda% mt -f /dev/nst0 fsf 1
...
```

Appendix B – Converting Modcomp floats and doubles to IEEE

```
def modcomp_to_IEEE_float (modcomp_value):
#
# Convert from Modcomp float to IEEE.
#

    u4 = struct.unpack('>I', struct.pack('>f',modcomp_value))[0]
    sign = 0x80000000 & u4
    if sign != 0:
        sign = 1
        u4 = ~(u4 - 1);
    exponent = ((u4 & 0x7fc00000) >> 22)
    mantissa = u4 & 0x003fffff
    IEEE_value = (-1)**sign * mantissa/(2.0**22) * 2.0**(exponent-256)
    return IEEE_value


def modcomp_to_IEEE_double (modcomp_value):
#
# Convert from Modcomp double to IEEE.
#

    u8 = struct.unpack('>Q', struct.pack('>d',modcomp_value))[0]
    sign = 0x8000000000000000 & u8
    if sign != 0:
        sign = 1
        u8 = ~(u8 - 1);
    exponent = ((u8 & 0x7fc0000000000000) >> 54)
    mantissa = u8 & 0x003fffffffffffff
    IEEE_value = (-1)**sign * mantissa/(2.0**54) * 2.0**(exponent-256)
    return IEEE_value
```