Data Formats: Using self-describing data formats

Curt Tilmes
NASA

Version 1.0
Review Date

Overview

- Self-describing data formats have become a well accepted way of archiving and disseminating scientific data.

Background

- Before self-describing data formats became widely used, each project often invented their own data formats, often raw binary or even ASCII.
- These approaches had a number of problems:
  - Machine dependent byte ordering or floating point organizations
  - Required a ‘key’ to be able to open the file and read the right data.
  - A new custom reader is needed for each different data organization. Working in a new language could be very difficult since you have to redevelop the reader anew.

Self-describing data formats

- Information describing the data contents of the file are embedded within the data file itself:
  - Names for various fields
  - Data types – Standardized, portable, machine independent
  - Pointers to various fields, making it efficient to extract the particular fields you want without reading the entire file
  - Attributes and flags related to the primary fields with extra information such as units, fill values, etc.
- Include a standard API and portable data access libraries in a variety of languages
- There are tools that can open and work with arbitrary files, using the embedded descriptions to interpret the data.

Some example formats

- HDF – Hierarchical Data Format
  - HDF4 and HDF5 versions are in use today
  - A NASA variant called HDF-EOS is used within the Earth Observing System program.
- NetCDF – Network Common Data Form
  - Widely used by agencies including NASA and NOAA
  - Climate and forecast (CF) metadata conventions help standardize some things into NetCDF in a common manner.

Best practices

- Choosing a self-describing format is a good first step, but it isn't a panacea. You still have to decide how to encode your data into the format.
- Think carefully about the how you use the format:
  - Layout of data within the file
  - Unambiguous names for fields; Use standard names if possible
  - Units
  - Fill values
- Keep the users/readers of your files in mind.
- Some formats support seamless internal compression that can help with file sizes.

Case Study: Format abuse

- A project had to distribute NORAD Two-Line Element (TLE) Sets

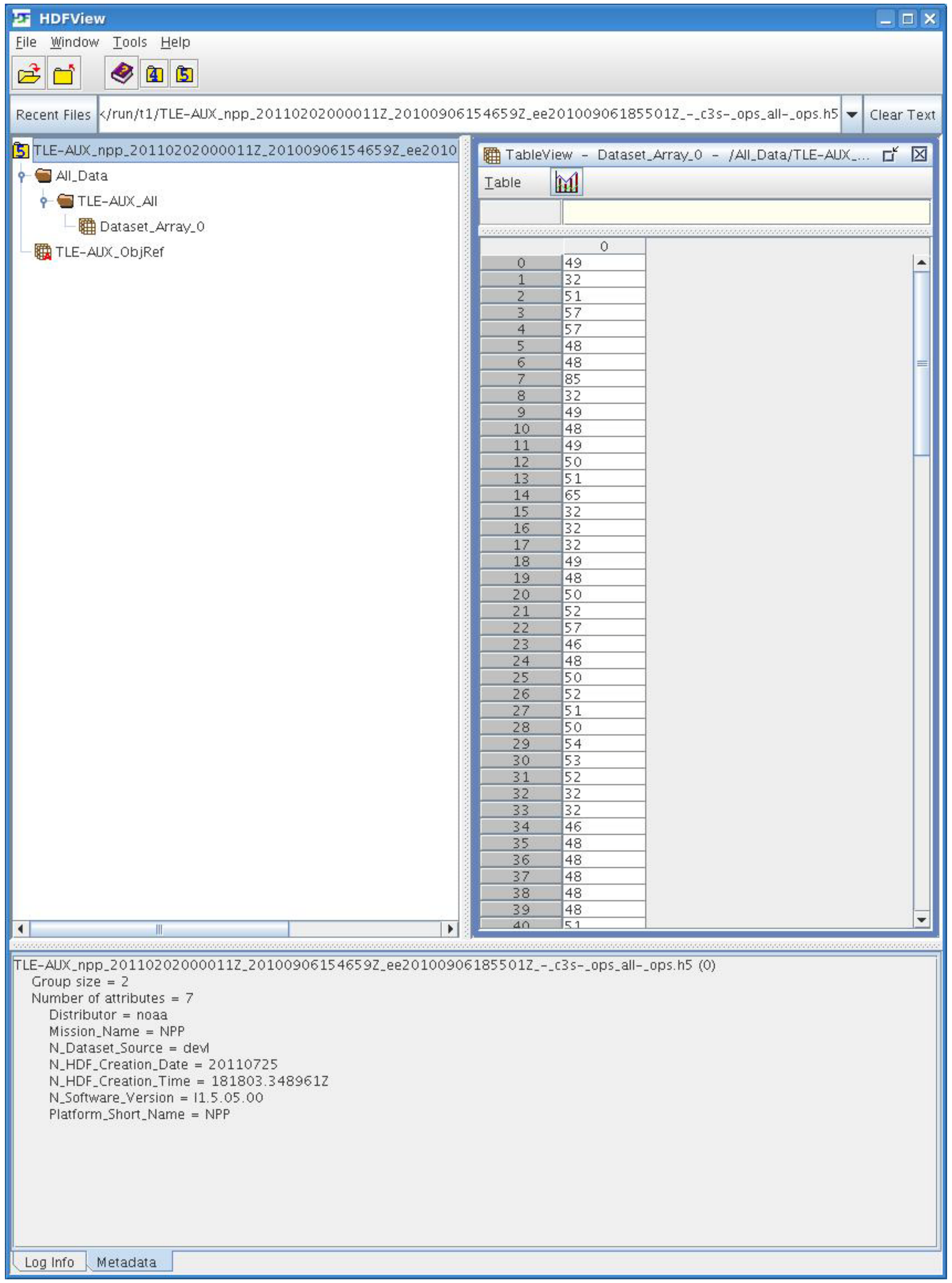
```
1 39900U 10123A 10249.02432654 .00000388 00001-0 14877-3 0 3039
2 39900 098.6793 188.3954 0009896 294.6098 065.4121 14.19557889216547
```

- This is a small amount of data, in a well defined format within ASCII, widely used and common.
  - ASCII isn't the best format, but for a small amount of data like this, especially in a widely used and understood format, it would have been fine.
  - People understand the TLE format and have standard ways to parse it.
  - Nevertheless, it isn't self-describing, and people unfamiliar with TLE wouldn't have a clue what those numbers mean.
- They chose to encode into HDF

Case Study: Format abuse (cont)

- A straightforward encoding would be to parse the fields, create fields with the right types (floating point) and name them according to their actual content from the TLE spec.
- They chose instead to maintain the ASCII text, encoding the individual characters of the file in their raw numerical form as an array of bytes.
- To read this data from the HDF file, you first have to extract the ASCII bytes, then parse yourself according to the TLE spec.
- Rather than attaching metadata to the data fields, they created a separate empty dataset just to hold the metadata.
- This is just bizarre. Don’t do it like that.

Case Study: Format abuse (cont)

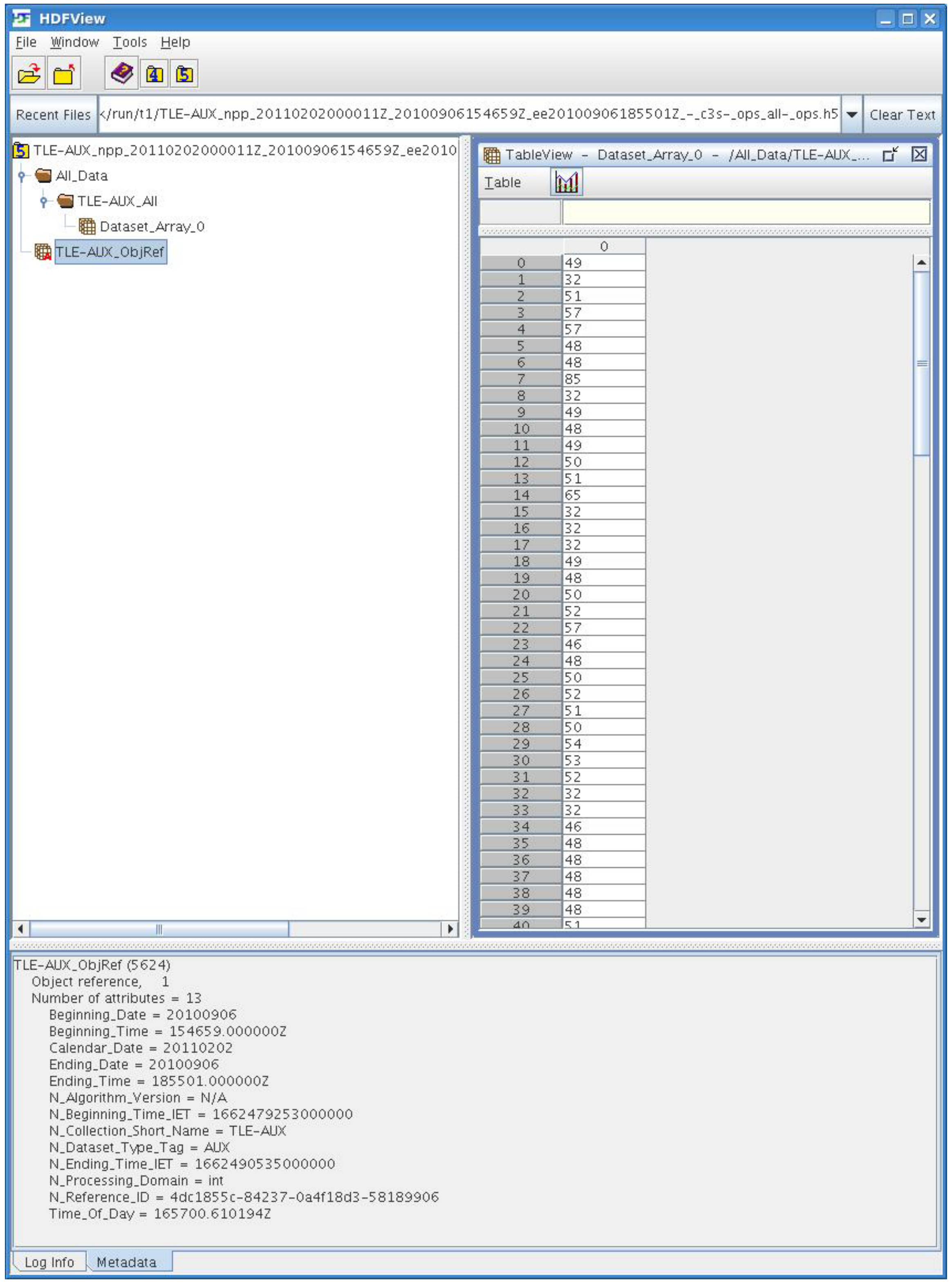


```
</run/t1/TLE-AUX_npp_20110202000011Z_20100906154659Z_ee20100906185501Z_-_c3s-_ops_all-_ops.h5

TLE-AUX_npp_20110202000011Z_20100906154659Z_ee2010
  All_Data
    TLE-AUX_All
      Dataset_Array_0
  TLE-AUX_ObjRef
```

	0
0	49
1	32
2	51
3	57
4	57
5	48
6	48
7	85
8	32
9	49
10	48
11	49
12	50
13	51
14	65
15	32
16	32
17	32
18	49
19	48
20	50
21	52
22	57
23	46
24	48
25	50
26	52
27	51
28	50
29	54
30	53
31	52
32	32
33	32
34	46
35	48
36	48
37	48
38	48
39	48
40	51

```
TLE-AUX_npp_20110202000011Z_20100906154659Z_ee20100906185501Z_-_c3s-_ops_all-_ops.h5 (0)
   Group size = 2
   Number of attributes = 7
      Distributor = noaa
      Mission_Name = NPP
      N_Dataset_Source = devl
      N_HDF_Creation_Date = 20110725
      N_HDF_Creation_Time = 181803.348961Z
      N_Software_Version = I1.5.05.00
      Platform_Short_Name = NPP
```



```
</run/t1/TLE-AUX_npp_20110202000011Z_20100906154659Z_ee20100906185501Z_-_c3s-_ops_all-_ops.h5

TLE-AUX_npp_20110202000011Z_20100906154659Z_ee2010
  All_Data
    TLE-AUX_All
      Dataset_Array_0
  TLE-AUX_ObjRef
```

	0
0	49
1	32
2	51
3	57
4	57
5	48
6	48
7	85
8	32
9	49
10	48
11	49
12	50
13	51
14	65
15	32
16	32
17	32
18	49
19	48
20	50
21	52
22	57
23	46
24	48
25	50
26	52
27	51
28	50
29	54
30	53
31	52
32	32
33	32
34	46
35	48
36	48
37	48
38	48
39	48
40	51

```
TLE-AUX_ObjRef (5624)
   Object reference,    1
   Number of attributes = 13
      Beginning_Date = 20100906
      Beginning_Time = 154659.000000Z
      Calendar_Date = 20110202
      Ending_Date = 20100906
      Ending_Time = 185501.000000Z
      N_Algorithm_Version = N/A
      N_Beginning_Time_IET = 1662479253000000
      N_Collection_Short_Name = TLE-AUX
      N_Dataset_Type_Tag = AUX
      N_Ending_Time_IET = 1662490535000000
      N_Processing_Domain = int
      N_Reference_ID = 4dc1855c-84237-0a4f18d3-58189906
      Time_Of_Day = 165700.610194Z
```

References and Resources

- HDF: http://www.hdfgroup.org
- HDF-EOS: http://hdfeos.org
- NetCDF: http://www.unidata.ucar.edu/software/netcdf
- CF: http://cf-pcmdi.llnl.gov/

Other Relevant Modules

- Avoiding proprietary formats
- Choosing and adopting community accepted standards
- Building understandable spreadsheets