EVENT FILING SYSTEM (EFS): A BINARY SEISMIC DATA FORMAT FOR FAST I/O ON DEMAND ON MULTIPLE PLATFORMS

Wenyuan Fan[1*], Peter M. Shearer[1], Daniel T. Trugman[2], Robin S. Matoza[3], Janine Buehler[4], Nicholas Mancinelli[1], Wei Wang[5], S. Shawn Wei[6], Xiaowei Chen[7], Guoqing Lin[8], Tianze Liu[1], and Jianhua Gong[1]

[1]Scripps Institution of Oceanography, UC San Diego, La Jolla, CA, USA
[2]Jackson School of Geosciences, UT Austin, Austin, TX, USA
[3]Department of Earth Science and Earth Research Institute, UC Santa Barbara, Santa Barbara, CA, USA
[4]Swiss National Science Foundation, Berne, Switzerland
[5]Department of Earth Sciences, University of Southern California, Los Angeles, CA, USA
[6]Department of Earth and Environmental Sciences, Michigan State University, East Lansing, MI, USA
[7]School of Geosciences, the University of Oklahoma, Norman, OK, USA
[9]Rosenstiel School of Marine and Atmospheric Science, University of Miami, Miami, FL, USA

*Corresponding Author

Wenyuan Fan
Institute of Geophysics and Planetary Physics
Scripps Institution of Oceanography, UC San Diego
9500 Gilman Drive, MC 0225
La Jolla, CA 92093-0225 email: wenyuanfan@ucsd.edu

Fri 27th Aug, 2021, 14:22

ABSTRACT

We describe a seismic data format called EFS (Event Filing System), which groups seismograms for common events or times into a single binary file that includes the waveform records, event, and trace header information (metadata). The format is designed for fast input/output (I/O) operations with efficient access to individual traces. The data format is highly scalable and in principle can store traces for an arbitrarily large number of stations. For large waveform databases, use of EFS greatly reduces the number of individual files, which reduces indexing and backup issues for typical UNIX systems, and speeds data processing by minimizing the opening and closing of files. The binary format of EFS files allows seismic data to be read and written flexibly in most programming languages. The core EFS subroutines are written in C and Fortran 90. We also present Python and MATLAB codes to work with the EFS files, including subroutines to convert common seismic data formats into EFS files. The Python and MATLAB codes can be used independently without the Fortran subroutines. An example is provided as a case study.

INTRODUCTION

Seismology research has made major advances with rapidly growing seismic data volumes. Efficient storage and retrieval of digital seismograms is a key requirement for seismology research. Over the years, many different formats have been developed to serve the purpose. For example, miniSEED has become the standard for archiving and distributing data (Mariotti and Utheim, 2006). Additionally, there are a variety of standards for storing seismograms locally for research purposes (e.g., Nickerson et al., 1999; Beyreuther et al., 2010). Many of the most commonly used formats (e.g., SAC) require one file per seismogram (Goldstein and Snoke, 2005; Helffrich et al., 2013). While this provides flexibility in manipulating the traces, it is less feasible for large datasets. When the number of files becomes too large, simple tasks such as input/output (I/O) or traversing the entire file tree can end up consuming substantial computing resources. As an example, a dataset of 10,000 earthquakes recorded by a 333-station network of 3-component sensors would result in 10 million individual files, with each file consuming a small portion disk space. In this case, the overhead of indexing, copying, or backing up the data can be prohibitive for most local computer systems and can pose significant challenges for computing on remote systems.

For efficient I/O, a small number of large files is better than a large number of small files. For example, Hierarchical Data Format (HDF) is a type of data format designed for such purposes, storing and organizing large volumes of data in a single file (Koranne, 2011). The newest version of HDF5 includes two major types of object: multidimensional datasets and metadata. In seismology, HDF5 has been customized as Adaptable Seismic Data Format (ASDF), which can store a large number of waveforms with metadata and earthquake catalogs (Krischer et al., 2016). For large scale analysis, e.g., array processing, there are direct organizational benefits in grouping data from common events (or common times) and including metadata in the same file as the seismograms. The EFS (Event Filing System) format shares the same motivation to group waveforms and metadata in a single file, but is constructed in a simple sequential fashion without an explicit hierarchical structure.

Before describing the format in detail, some history may be of interest. During the 1980s, Prof. Guy Masters at Scripps Institution of Oceanography, UC San Diego developed a format called GFS

("Guy's Filing System") in which a GFS file was actually a directory containing individual binary files (one file per trace) and an additional file containing indexing and event information. Guy provided a set of Fortran subroutines to open GFS directories and read or write data and/or header information from individual files. The actual I/O was done using C subroutines, which provided greater flexibility than Fortran at the time and avoided the extra bytes that Fortran includes at the beginning and end of binary read/write calls. Fast I/O was achieved by reading entire arrays with single calls. For header information, this requires grouping mixed-format variables (strings, integers, reals) into a single binary array, which can be done conveniently in C using structures. Fortran at the time did not include structures, but Guy was able to mimic them with the crafty use of equivalence statements to align Fortran arrays with common blocks. GFS was used by many of Guy's students and colleagues and facilitated considerable and impactful research in global seismology (e.g., Houser et al., 2008; Laske et al., 2013; Ma et al., 2014), but never achieved common acceptance.

The EFS format was designed to preserve many of the advantages of GFS while reducing the number of files required to store large data sets. Low-level I/O operations continue to use C to read and write binary blocks, but the key Fortran subroutines now use Fortran 90 user-defined types to replace the GFS equivalence statements to create data structures for header information. Another improvement is that byte-swapping is performed automatically in cases where the computer reading the file uses a different byte order than the computer writing the file. The header area is also larger than in GFS in order to accommodate 8-character station names and other fields, including space set aside for future expansion or user customization. Although significant effort was made to include the most commonly required parameters for seismic data analysis, the format can be easily modified by the user. The EFS data format has been widely used in studying a variety of problems, including both earthquakes and seismic structures (Lin et al., 2007, 2010; Matoza et al., 2013; Chen and Shearer, 2013; Matoza et al., 2014; Trugman and Shearer, 2017a; Trugman et al., 2017; Wang and Shearer, 2017; Trugman and Shearer, 2018; Shearer and Buehler, 2019; Wei et al., 2020; Tian et al., 2020; Matoza et al., 2021; Liu and Shearer, 2021).

The core Fortran and C subroutines and some example programs are available at: Github (https://github.com/wenyfan/EFS.git). There is a *readme* file that provides a detailed binary description of the format, instructions for compiling the code, and a list of the subroutines and their functionalities. Further, we provide sets of codes in Python and MATLAB to facilitate seismic data processing across multiple programming languages and operation platforms.

DATA FORMAT DESCRIPTION

The EFS file format contains multiple records for an event, e.g., earthquake or explosion, and the associated metadata of both the records and the event in a single file. It is a binary format and stores all the information in a sequential order (Figure 1). Each file consists of a file header (*fhead*), an event header (*ehead*), followed by a byte position array (*bytepos*) that gives the byte locations of the time series, a time series header (*tshead*), a time series (*ts*), another pair of time series header and time series, and more pairs of time series header and time series until the end of the file. In principle, the EFS file can contain as many records as desired and the records can have arbitrary lengths that can be different from record to record.

The file header of an EFS file defines the *ehead* and *tshead* attributes, including their types and the number of bytes of these metadata in the EFS file. The event header lists the event location, type, time, magnitudes (e.g., mb, Ms, and Mw), moment, and focal mechanism information. All of these fields are optional, as the relevant event metadata can differ markedly between datasets. The *bytepos* array stores the starting byte positions of the time series headers. Each *tshead* documents the length of the time series, station information (location, sensor type, gain, units, etc.), low and high frequency limits of any band-pass filter, time of the starting point of the time series, source-station information (distance, azimuth, time difference between the event origin time and the time series starting time), and up to four phase picks. The time series (*ts*) are strings of 4-byte binary numbers with the lengths defined in the time series headers. The time series header can be read without reading its accompanying time series, which can save time in cases where only data from particular stations, channels, source-receiver distances, etc., are to be analyzed.

Although the event header was designed to store earthquake information, the EFS format can also be used simply to group and store traces sharing the same time period, as, for example, might be used in ambient noise cross-correlation analysis or detection analysis through continuous records

(Fan et al., 2019, 2020). In this case, the event information can be left out of the header, although for convenience we generally set the event time to the common targeted start time of all the individual time series. Just as in earthquake studies, EFS provides advantages in analysis of continuous data by greatly reducing the total number of files and providing rapid binary read/write operations.

FILES AND PROGRAM DESCRIPTION

The EFS format is designed for efficient storage and fast retrieval of seismograms. The Fortran 90 package performs the basic I/O operations in C by calling routines from disk.c, which must be compiled separately. The Python and MATLAB routines read and write EFS files following the same data structure as defined in Fortran 90, but do not need to be complied.

3.1 Fortran Package

This includes the C I/O routines, disk.c, and the C byteswapping routines, swapsubs.c. These may be compiled using (assuming gcc compiler):

```
gcc -c disk.c -o disk.o
gcc -c swapsubs.c -o swapsubs.o
```

The main Fortran subroutine library is efs_subs.f90 and includes the subroutines described below. To compile the efs_subs.f90 source code, enter (assuming gfortran compiler):

```
gfortran -c efs_subs.f90 -o efs_subs.o
```

An example Makefile for the program testefs.f90 is as follows:

```
OBJS2 = disk.o \
        swapsubs.o \
        efs_subs.o
testefs: Makefile testefs.f90 $(OBJS2)
       gfortran $(OBJS2) testefs.f90 -o testefs      #use tab before gfortran!
```

The header formats take advantage of the ’type’ construct in Fortran 90, which provides something similar to structures in C. These structures are defined in a module at the beginning of the efs_subs.f90 source code. Thus the first line of any program that uses these routines should be:

```
use EFS_MODULE
```

The EFS Fortran subroutines are as follows:

```
EFS_OPEN_NEW creates new EFS file
EFS_OPEN_OLD opens existing EFS file
EFS_CLOSE closes EFS file
EFS_INIT_EHEAD sets event header to default values
EFS_WRITE_EHEAD writes event header
EFS_UPDATE_EHEAD updates event header
EFS READ_EHEAD reads event header
EFS LIST_EHEAD lists contents of event header
EFS WRITE_TSHEAD writes time series header
EFS_READ_TSHEAD reads time series header
EFS_INIT_TSHEAD sets time series header to default values
EFS_WRITE_TS write times series to EFS file
EFS_READ_HEADTS reads time series header AND time series
EFS_READ_TS reads time series
```

Argument lists for these subroutines are explained in an efs.readme.txt file. Some simple example programs are also included:

```
listefs.f90 -- lists the contents of an EFS file
copyefs.f90 -- reads an EFS file and copies the contents to a new EFS file
sac2efs.f90 -- reads from a list of SAC files and write the contents to a single EFS file
```

3.1.1 Speed comparison

We performed a test using a Mac laptop with solid state storage using a directory containing 721 SAC files for a southern California earthquake, comparing the time to open, read, and close all the SAC files versus opening, reading, and closing the corresponding single EFS file.

```
      Run times (s)
SAC     0.32
EFS     0.03
```

Thus, reading all of the data for this earthquake can be performed about 10 times faster using a single EFS file compared to using individual SAC files.

3.2 Python Package

The Python programming language is widely used by the seismology community due to its public, open source framework and the availability of seismology-tailored packages such as ObsPy (Beyreuther et al., 2010). Currently, ObsPy supports a wide range of seismological data formats, including MiniSEED, SAC, SEGY, and WAV, though does not at present provide direct integration with EFS. Here we provide Python software tools to process and analyze EFS files as well as convert EFS data into Stream objects for direct manipulation using ObsPy. Specifically, we construct a Python module (*EFSpy_module.py*) to contain definitions and functions to work with EFS files. The Python package has been tested using version 3.8.

By calling *EFS*, an EFS file can be read into Python as a structure array. The structure contains dictionaries to store information from *fhead*, *ehead*, *tshead*, and ts. The time series (ts) and its time series header are combined together as one list *waveforms*. Here *EFS* can read EFS files that have values in *bytepos* saved as 32-bit signed integers and values in ts saved as single-precision floating-point numbers. Users can customize the data (*ts*) and *bytepos* precisions, although the data precisions in the Fortran package are set as single-precision floating-point numbers *ts* and 32-bit signed integers for *bytepos*. For example, *EFS(filename,ts_type,bp_type)* can read EFS files with *bp_type* and *ts_type* specifying the precisions for the *bytepos* and *ts* arrays.

```python
import obspy
import numpy as np
from EFSpy_module import *
efsname = 'EFS_Example.efs';
```

```python
# Read EFS into a Python structure with the time series written in float32 and the bytepos written in 32-bit integer
efs_data = EFS(filename,np.float32,np.int32)
# Write EFS to file converting the time series format to 32-bit integer and and the bytepos remain as 32-bit integer
filename_export = 'EFS_Example_export.efs'
export_efs('./', filename_export, efs_data,np.int32,'i')
# Convert EFS to obspy
st = efs_data.to_obspy()
```

Listing 1: Python example of processing EFS files

3.3 MATLAB Package

MATLAB is a programming language and numeric computing environment, specializing in matrix manipulations and digital signal processing (DSP). We provide MATLAB codes to read and write EFS files. The EFS data will be converted into a nested structure array in MATLAB, and standard DSP packages in MATLAB can be directly applied to the seismic data.

We provide three MATLAB functions to work with EFS files, including *load_efs.m*, *write_structure2efs.m*, and *write_iris2efs.m*. Function *load_efs.m* can read EFS files into MATLAB structures, and function *write_structure2efs.m* can write MATLAB structures into binary EFS files. Additionally, we also provide function *write_iris2efs.m* to combine with using *irisFetch* (Trabant et al., 2012; Hutko et al., 2017) to download data from IRIS DMC in MATLAB and then convert the data into EFS files. The MATLAB package has been tested using version 2020a.

Similar to a Python structure, a structure array in MATLAB groups data in data containers, which are called fields. For a EFS structure, there are four fields, including the file name, *fhead*, *ehead*, and *waveforms*. The *fhead* and *ehead* fields contain the same information as those in the EFS file. The *waveforms* field contains both the station information (*tshead*) and the time series (ts). These data, e.g. a station name of the ith station, can be accessed using dot notation as *efsStructure.waveforms(i).stname*.

```matlab
% EFS file names
efsname = 'EFS_Example.efs';
```

```
efsname_export = 'EFS_Example_export.efs';
% itype specifies the bytepos value format and the ts value format, e.g., 32-bit signed integers (int32) or single-precision floating-point values (single)
itype = 1;
% Read in the example EFS file
efsStructure=load_efs(efsname,itype);
% Write the MATLAB structure into an EFS file
iosuc =write_structure2efs('./',efsname_export,...
                              efsStructure,itype);
```

Listing 2: MATLAB example of processing EFS files

EXAMPLE: 2019 MW 7.1 RIDGECREST EARTHQUAKE

Here we take processing seismic data of the 2019 Mw 7.1 Ridgecrest earthquake in Python as an example to showcase using EFS files. The data are from HHZ component records of broadband seismic stations in southern California, CI network. We first obtain the event, station, and waveform data in ObsPy format and then create an EFS file from this information, and demonstrate file export and conversion.

```
# Import statements
import numpy as np
import struct
import obspy
from obspy import UTCDateTime, geodetics
from EFSpy_module import *
from obspy.clients.fdsn import Client
import os

# Specify file paths
EFSPATH = "../EX_DATA/"
miniSEED_PATH = '../EX_DATA/CI/'

# Download event information for Ridgecrest using ObsPy
print("\nDownloading Ridgecrest event information...")
```

```python
origin_time = obspy.UTCDateTime(2019, 7, 6, 3, 19, 53)
clntnm = 'IRIS'
client = Client(clntnm)
cat = client.get_events(starttime = origin_time - 20, endtime = origin_time + 20,
    minmagnitude = 7)
print("Done\n")

# Read waveforms into obspy Stream and Inventory
iPATH = miniSEED_PATH + 'waveforms/'
iPATH_inv = miniSEED_PATH + 'stations/'
try:
    st1 = obspy.read(iPATH + '*')
    inv1 = obspy.read_inventory(iPATH_inv + '*')
    print("Reading data from:", iPATH)
    print(st1)
    print("First trace:")
    print(st1[0].data)
except:
    print('Oops, no data')

# Initialize an event header for the EFS file
ehead = cat2ehead(st1,cat)
npts_max = 100000 # set max points per trace in file
ntr_max = 1000 # set max number of traces in file

# Create EFS file from Obspy
print("\nCreating EFS object.")
efs_data = EFS.from_obspy(st1, ehead, inv1)
efs_data.ehead['bytepos'] = 248 + ntr_max * 4 + 1 + np.arange(0, ntr_max) * npts_max * 4
print("Done.")

# Write EFS to file
print("\nTesting EFS export.")
filename = 'EFS_Example.efs'
# save arrays as default f32 precision, bytepos as i32
```

```python
export_efs(EFSPATH, filename, efs_data,np.float32,"i")
filename2 = 'EFS_Example2.efs'
# save arrays as i32 precision, bytepos as i32
export_efs(EFSPATH, filename2, efs_data,np.int32,"i")
print("Done.")

# Read EFS with customized precision as they were stored
efs_data_2 = EFS(EFSPATH + filename,np.float32,np.int32)
efs_data_3 = EFS(EFSPATH + filename2,np.int32,np.int32)
# Convert EFS to obspy
print("\nConverting EFS object back to a stream.")
st2 = efs_data_2.to_obspy()
st3 = efs_data_3.to_obspy()
```

Listing 3: Example of the 2019 Mw 7.1 Ridgecrest earthquake

DISCUSSIONS AND SUMMARY

The EFS file format offers great flexibility for processing seismic records. For example, depending on the data volume, the *bytepos* and the *ts* arrays can be saved in different data types with the desired precision, e.g., 4-byte integer or 8-byte integer. Another advantage is that EFS files can be processed in multiple programming languages across different operating systems. Our Fortran subroutines can handle large volumes of datasets efficiently, while the Python and MATLAB scripts can bridge the binary data format with ObsPy and other tools seamlessly. Additionally, it is easy to modify the codes to customize the EFS data formats to accommodate individual research on different topics. For instance, we built one of the largest databases of SS precursors using a data format that derived from EFS (Wei et al., 2020; Tian et al., 2020). Further, EFS files can be integrated as a part of multiple open-source research software, including the GrowClust earthquake relocation algorithm (Trugman and Shearer, 2017b; Lin, 2018). In the future, we plan to continue developing and publishing algorithms, including waveform cross-correlation codes, to build an ecosystem to efficiently work with the EFS data format for seismological research.

In summary, we propose a new seismic data format, the EFS file format, which groups multiple seismic records for an event in a single file. The file contains metadata of both the event and the records, and we leave additional space for future purposes. We also publish a suite of codes in Fortran, Python, and MATLAB to work with EFS files on multiple platforms. The EFS files are binary and they are constructed in a simple sequential fashion. The EFS system is efficient for frequent I/O operations, which is ideal for managing large volume seismic datasets.

ACKNOWLEDGEMENTS

S.S.W. wants to thank the Cecil H. and Ida M. Green Foundation. R.S.M. acknowledges NSF grant EAR-1446543.

REFERENCES

Beyreuther, M, **Barsch, R**, **Krischer, L**, **Megies, T**, **Behr, Y**, **Wassermann, J**. ObsPy: A Python toolbox for seismology. *Seismological Research Letters 81*: 530–533, 2010.

Chen, X, **Shearer, PM**. California foreshock sequences suggest aseismic triggering process. *Geophysical Research Letters 40*: 2602–2607, 2013.

Fan, W, **McGuire, JJ**, **de Groot-Hedlin, CD**, **Hedlin, MA**, **Coats, S**, **Fiedler, JW**. Stormquakes. *Geophysical Research Letters 46*: 12909–12918, 2019.

Fan, W, **McGuire, JJ**, **Shearer, PM**. Abundant spontaneous and dynamically triggered submarine landslides in the Gulf of Mexico. *Geophysical Research Letters 47*: e2020GL087213, 2020.

Goldstein, P, **Snoke, A**. SAC availability for the IRIS community. *Incorporated Research Institutions for Seismology Newsletter 7*, 2005.

Helffrich, G, **Wookey, J**, **Bastow, I**. *The seismic analysis code: A primer and user's guide*. Cambridge University Press, 2013.

Houser, C, **Masters, G**, **Shearer, P**, **Laske, G**. Shear and compressional velocity models of the mantle from cluster analysis of long-period waveforms. *Geophysical Journal International 174*: 195–212, 2008.

Hutko, AR, **Bahavar, M**, **Trabant, C**, **Weekly, RT**, **Fossen, MV**, **Ahern, T**. Data products at the IRIS-DMC: Growth and usage. *Seismological Research Letters 88*: 892–903, 2017.

Koranne, S. Hierarchical data format 5: HDF5. In: *Handbook of Open Source Tools*, Springer, 191–200, 2011.

Krischer, L, **Smith, J**, **Lei, W**, **Lefebvre, M**, **Ruan, Y**, **de Andrade, ES**, **Podhorszki, N**, **Bozdağ, E**, **Tromp, J**. An adaptable seismic data format. *Geophysical Supplements to the Monthly Notices of the Royal Astronomical Society 207*: 1003–1011, 2016.

Laske, G, **Masters, G**, **Ma, Z**, **Pasyanos, M**. Update on CRUST1. 0—A 1-degree global model of Earth's crust. In: *Geophys. res. abstr*, 2013, volume 15, 2658.

Lin, G. The source-specific station term and waveform cross-correlation earthquake location package and its applications to California and New Zealand. *Seismological Research Letters 89*: 1877–1885, 2018.

Lin, G, **Shearer, PM**, **Hauksson, E**. Applying a three-dimensional velocity model, waveform cross correlation, and cluster analysis to locate southern California seismicity from 1981 to 2005. *Journal of Geophysical Research: Solid Earth 112*, 2007.

Lin, G, **Thurber, CH**, **Zhang, H**, **Hauksson, E**, **Shearer, PM**, **Waldhauser, F**, **Brocher, TM**, **Hardebeck, J**. A California statewide three-dimensional seismic velocity model from both absolute and differential times. *Bulletin of the Seismological Society of America 100*: 225–240, 2010.

Liu, T, **Shearer, PM**. Complicated lithospheric structure beneath the contiguous US revealed by teleseismic S-reflections. *Journal of Geophysical Research: Solid Earth 126*: e2020JB021624, 2021.

Ma, Z, **Masters, G**, **Laske, G**, **Pasyanos, M**. A comprehensive dispersion model of surface wave phase and group velocity for the globe. *Geophysical Journal International 199*: 113–135, 2014.

Mariotti, M, **Utheim, T**. A white paper about MiniSEED for LISS and data compression using Steim1 and Steim2. *Dept of Earth Science, University of Bergen* , 2006.

Matoza, RS, **Okubo, PG**, **Shearer, PM**. Comprehensive high-precision relocation of seismicity on the Island of Hawai ‘i 1986–2018. *Earth and Space Science 8*, 2021.

Matoza, RS, **Shearer, PM**, **Lin, G**, **Wolfe, CJ**, **Okubo, PG**. Systematic relocation of seismicity on Hawaii Island from 1992 to 2009 using waveform cross correlation and cluster analysis. *Journal of Geophysical Research: Solid Earth 118*: 2275–2288, 2013.

Matoza, RS, **Shearer, PM**, **Okubo, PG**. High-precision relocation of long-period events beneath the summit region of Kilauea Hawai'i, from 1986 to 2009. *Geophysical Research Letters 41*: 3413–3421, 2014.

Nickerson, BG, **Judd, PA**, **Mayer, LA**. Data structures for fast searching of SEG-Y seismic data. *Computers & Geosciences 25*: 179–190, 1999.

Shearer, PM, **Buehler, J**. Imaging upper-mantle structure under USArray using long-period reflection seismology. *Journal of Geophysical Research: Solid Earth 124*: 9638–9652, 2019.

Tian, D, **Lv, M**, **Wei, SS**, **Dorfman, SM**, **Shearer, PM**. Global variations of earth's 520- and 560-km discontinuities. *Earth and Planetary Science Letters 552*: 116600, 2020.

Trabant, C, **Hutko, AR**, **Bahavar, M**, **Karstens, R**, **Ahern, T**, **Aster, R**. Data products at the IRIS DMC: Stepping stones for research and other applications. *Seismological Research Letters 83*: 846–854, 2012.

Trugman, DT, **Dougherty, SL**, **Cochran, ES**, **Shearer, PM**. Source spectral properties of small to moderate earthquakes in southern Kansas. *Journal of Geophysical Research: Solid Earth 122*: 8021–8034, 2017.

Trugman, DT, **Shearer, PM**. Application of an improved spectral decomposition method to examine earthquake source scaling in Southern California. *Journal of Geophysical Research: Solid Earth 122*: 2890–2910, 2017a.

Trugman, DT, **Shearer, PM**. Growclust: A hierarchical clustering algorithm for relative earthquake relocation, with application to the Spanish Springs and Sheldon, Nevada, earthquake sequences. *Seismological Research Letters 88*: 379–391, 2017b.

Trugman, DT, **Shearer, PM**. Strong correlation between stress drop and peak ground acceleration for recent M 1–4 earthquakes in the San Francisco Bay area. *Bulletin of the Seismological Society of America 108*: 929–945, 2018.

Wang, W, **Shearer, PM**. Using direct and coda wave envelopes to resolve the scattering and intrinsic attenuation structure of southern california. *Journal of Geophysical Research: Solid Earth 122*: 7236–7251, 2017.

Wei, SS, **Shearer, PM**, **Lithgow-Bertelloni, C**, **Stixrude, L**, **Tian, D**. Oceanic plateau of the hawaiian mantle plume head subducted to the uppermost lower mantle. *Science 370*: 983–987, 2020.

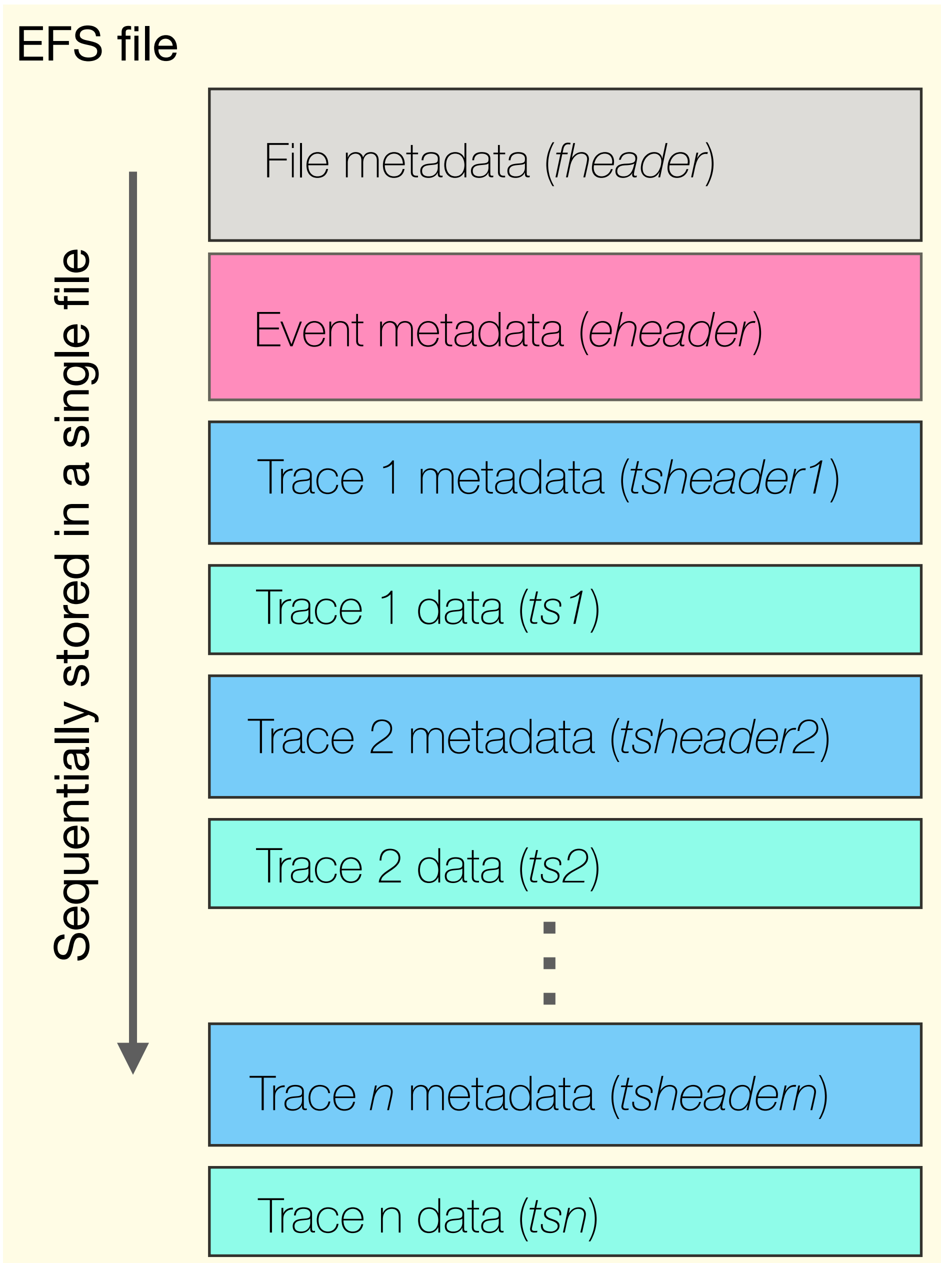

Figure 1: EFS data structure.