Memorandum

U.S. Department
of Transportation
**Volpe National
Transportation
Systems Center**

Subject: DELIVERABLE:
Volpe SuperFAR V6.0 Software and Support Documentation;
Letter Report V324-FB48B3-LR3

Date: 29SEP2017

From: Dave Read, IT Specialist, Acoustics, Volpe Center,
Environmental Measurement and Modeling Division

Reply to Attn. of: V324

To: Rebecca Cointin, Manager, Noise Division, FAA/AEE-100

This Letter Report serves to deliver the third external release version of the USDOT Volpe Center's SuperFAR Spectral Aircraft Noise Processing Software (Version 6.0). Earlier versions of the software were delivered to FAA in February 2015 and March 2016 via Volpe Letter Reports V324-FA5JB4-LR10 and V320-FA5JBH-LR7 respectively.

This version of the software represents substantial improvements in the underlying structure of the code, minor bug fixes, and minor improvements in functionality, including:

- Provision for generating verbose output to the Python shell for diagnostic purposes;
- Improved handling of input variables and filenames;
- Ability to generate and handle both single-spectrum and spectral time-history versions of cumulative test-day atmospheric absorption coefficients;
- Increased flexibility in determining the temperature used for calculation of test-day soundspeed, as well as the ability to select alternate methods for calculation of soundspeed itself;
- Improved reporting of test-day and reference bandsharing, as well as presentation of PNLTM with and without bandsharing applied;
- Replaced TC1k flag with variable TCLowBand to allow for Pseudotone elimination to start at user-specified band. (For situations where an applicant might select 800 Hz instead of 1kHz, for example.)

This version also benefits from having been exercised by Volpe as the primary resource for validation of applicants' noise certification software and methodologies since the previous version was released.

In addition to changes to the software itself, the User Guide has been updated to Version 2, and a new Developer's Manual has also been created. This Manual is provided in html form, and incorporates written text developed by Volpe, as well as specific documentation of individual modules, generated automatically from within the SuperFAR software. Volpe expects that this hybrid method of documentation will provide accurate, useful and up-to-date information and specifications for those interested in examining or developing the source code and software further. This document will automatically be updated with each new release of SuperFAR (including internal, incremental releases). The updated Users Guide covers the new modules and functionality added since the previous release.

Finally, a new example script and data set covering an updated preferred process is included to replace the examples in the previous releases. The data set has been further anonymized beyond what was previously provided, and the new script now exercises the meteorological data handling functions. It also implements the use of global variables within the script itself for passing data directly between modules without first writing to and reading from data files.

If you have any comments or questions, please do not hesitate to contact me.

–

Attachment:

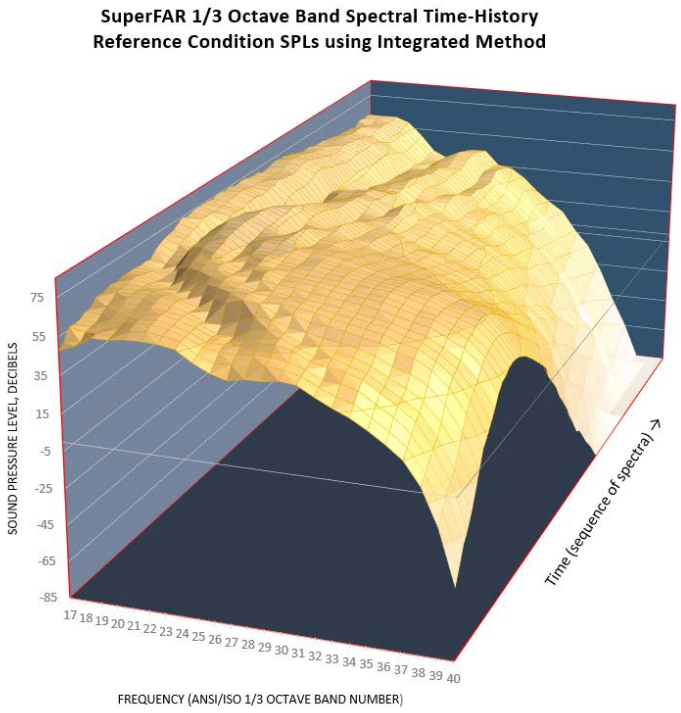
SuperFAR V6.0 distribution package;

cc:

M. Marsan, FAA, AEE-100
S. Liu, FAA, AEE-100
B. Conze, FAA (TADNCS), AIR-672
C. Cutler, Volpe, V324
C. Roof, Volpe, V324
C. Reherman, Volpe, V320
G. Fleming, Volpe, V320
E. O'Neil, Volpe Contractor – Safety Net Systems, V343

U.S. Department
of Transportation
**Volpe National
Transportation
Systems Center**

USDOT Volpe Center Acoustics
SuperFAR
(Spectral Data Processing Suite)
User Guide - V2



Dave Read & Chris Cutler, Environmental Measurement and Modeling Division, V-324

Eugene O'Neil (Contractor - Safety Management Systems)

SuperFAR V6.0 **29 September 2017**

TABLE OF CONTENTS

1. Introduction

SuperFAR is a collection of software modules – developed by the Environmental Measurement and Modeling Division of the US Department of Transportation's Volpe National Transportation Systems Center - that performs various elements of the EPNL calculation process used for aircraft noise certification in accordance with the specifications of 14CFR part 36 and ICAO Annex 16, Volume I. It is implemented in the Python programming language, and operation of the software is based on plain-text scripting. Input and output files are also plain-text, formatted as comma-separated values, and can be imported into any text-editor or spreadsheet software application. SuperFAR is currently only available as a Microsoft Windows installation package.

Starting with a time-history of 1/3 octave band sound pressure levels obtained from a real-time analyzer, in combination with aircraft position TSPI data and meteorological measurement data, the software allows for computation of reference-condition EPNL using a variety of methods. Users may select from various options for each module by assigning them in a script, and may change the sequence of operations by selecting the order in which various modules are called within a script. Example scripts are provided which include suggested input/output file-naming schemes, documentation of selectable inputs (including listings of options for enumerated inputs), and copious commenting to aid in understanding the available processes and methods within SuperFAR.

This User Guide provides information on installation and operation of SuperFAR, as well as information about editing and creating scripts, and preparing input data files. It also provides functional descriptions and guidance for each of the modules in the current version.

A Developer Manual is available which contains source code listings, as well as complete documentation of the various file-handling systems, data-object methods, qualities and attributes, and additional information not contained in the User Guide.

2. Installation and Uninstallation

The standard SuperFAR distribution zip file contains the latest SuperFAR library, with example scripts and data, as well as a copy of this User Guide.

SuperFAR requires Python 3.4.1 or later. A suitable Python installer for Windows (python-3.4.1.msi) is provided in the standard SuperFAR distribution. Execute this installer and make sure to select the option "add python.exe to path" in the customization settings during the install process. For all other installation options, the default settings should be sufficient.

Next install the SuperFAR library under Windows by executing the SuperFAR installation exe (superfar-4.*.win32.exe). Leaving all installation options at their default settings should be sufficient. Note that this installation exe will forcefully overwrite any version of SuperFAR that is already installed on the target machine, effectively acting as an upgrade or downgrade.

To uninstall SuperFAR from Windows, manually remove the directory “C:Python34\Lib\site-packages\superfar” and all of its contents.

Once Python 3.4 and the SuperFAR library are installed, copy the SuperFAR example scripts and data from the NEW Preferred Process subdirectory of the distribution archive to a working directory, and they should be ready to run.

The Source Code subdirectory contains the full source code of SuperFAR, which may be of interest to advanced developers.

3. Operation

Once SuperFAR and Python have been installed and set up per the instructions in Section 2 of this User Guide, a data folder needs to be selected or created. This folder must contain a Python script and any input data files required for operation of the SuperFAR modules called by the script. (See Sections 4 and 5 of this User Guide for general information about Scripts and Data Files, respectively, and see Appendix 1 on SuperFAR Modules and Appendix 2 on SuperFAR File Formats for details and specifications for particular file structures, formats and requirements.)

All python scripts that come with SuperFAR have a corresponding windows BAT file, with a matching name that differs only by having a ".bat" extension instead of a ".py" extension. Double clicking the BAT file in windows explorer will run the python script and save the output to a log file.

If the user wishes to write a new python script, either from scratch or as a modified version of an existing script, there is no need to edit a BAT file to execute it: merely copy and rename the included run_script.bat file to match the new script name (use the script name, but replace the .py extension with .bat), and the BAT file will dynamically deduce the name of the python script it should execute from its own name, as well as the name of the log file it saves the output of the script to.

Alternately, a SuperFAR script can be run from within a python IDE, such as "IDLE" which comes standard with Python for Windows, to test a script as it is being created, modified, or debugged. From the file directory of the data folder, right-click on the script you wish to debug and select "Edit with IDLE". The IDLE script-editing window opens. From the menu bar at the top of the window, click on "Run", and then select either "Check Module" (to check the syntax of the python script) or "Run Module" (to actually run the script). Either selection will open a Python Shell window. (When invoking a SuperFAR Python script from a batch file based on run_script.bat, the shell output is redirected to a .log text file that can be examined after the run has completed.) "Check Module" will open a shell with some version info and an interpreter prompt ">>>" with a blinking cursor if there are no Python issues with the script. "Run Module" will open a shell that contains any print statements within the script, plus diagnostic information about each module that is called, including echoing all of the input and output settings within the script. In either case, the shell will also provide error messages if any Python or file-related errors are encountered. An instance of a Python shell can be saved as a text file or sent to a printer for later reference. If there are no errors, and the shell is no longer needed, it can be closed, as can the script-editing window.

Output data files can be found in the data subfolder named within the script – typically "Results" – and can be viewed or edited with external tools (text editors, spreadsheet software, etc.).

See Appendix 3: Data analysis and visualization, for more information on evaluating input and output data files.

4. Scripts

SuperFAR processing and sequencing is controlled by scripting in Python. Python scripts are plain-text files that have a .py extension. They can be generated and edited in any Windows text editing software, such as Notepad, Notepad++, or WordPad, or within the provided Python GUI, IDLE. It is recommended that each SuperFAR script be located within a base directory folder that also contains the input data files for a particular project for one or more aircraft events. The folder should be named to reflect the process being used for that particular data set. (The example provided in the SuperFAR installation is named “NEW Preferred Process”.)

SuperFAR scripts can include comments and any Python commands or operators supported by Version 3. (See external Python references for more info.) Scripts that are intended to be Linux compatible should begin with this line:

```
#!/usr/bin/env python3
```

To access SuperFAR modules within Python, a script must include the following line prior to invoking any SuperFAR-specific functionality:

```
from superfar import *
```

To enable or disable “verbose” output for diagnostic purposes, the following line should be set appropriately. **True** will enable verbose output, **False** will disable (default setting) :

```
superfar.verbose = False
```

(Note that the verbose functionality can be switched on and off locally within a script)

The following line reports the current version of SuperFAR (essential for troubleshooting and diagnostics):

```
print (“SuperFAR Version ID: “ + SUPERFAR_VERSION)
```

The user can assign a descriptive “Project Name”, along with the “Event ID” prefix unique to each project by using the following lines:

```
superfar.ProjectName = 'SuperFAR CLTO1 Preferred Process Example Case'
EventID = "CLTO1"
```

It is also useful to set up paths and data subdirectories:

```
if not os.path.exists(“Results”):
        os.makedirs(“Results”)
```

```
if not os.path.exists("Temp"):
        os.makedirs("Temp")
```

The example script includes many commands and operators, including setting of input variables, assignment of filenames, commenting, interaction with the shell, and so on. For implementation of any specific functionality, please contact Volpe.

When modifying an existing SuperFAR script, it is recommended to rename the file in a meaningful manner. Original versions of the example scripts can be restored from the installation zip file at any time.

When generating or modifying a SuperFAR script, there are several important elements to keep in mind:

- Header
- Commenting
- Calling
- Options
- Filespecs
- Shell

a. Header – this should include path info and invoke the SuperFAR code library, then identify the process being used.

b. Commenting – this is critical for documenting processes applied to a particular data set, and care should be taken to include sufficient and accurate commenting for future reference.

c. Calling – Each module is called by name, followed by options selections and I/O filespecs. Comments can be freely inserted within the call.

d. Options – within the calling structure for each module are settings for options within the module. Each of these should be carefully considered and selected, as they can have substantial effects on the output data values. There are several types of input options:

    1. Numerical values – typically real numbers that represent some decibel adjustment, physical quantity, or criterion. Usually found within a source data set. If uncertain, it is probably best to enter "0.0".

    2. Logical values – typically used to set flags for process options within a module, these should be entered as simple text: True or False.

3. Enumerated – these inputs must be selected from a group of pre-defined values. The example script provided with the SuperFAR installation includes comments that list all such optional values, wherever they are used. It is recommended to keep such comments in any modified scripts for easy access. Defined values for enumerations can also be found in the Appendix to this User Guide, within the description of the individual modules.

   An example of an enumerated input is the AveragingMethod setting in ReAvg: It can be set to any of the following:
   - 'CONTEXPO' – Continuous exponential averaging
   - '4S100' – four-sample, 100% coefficients
   - '4S95' – four-sample, 95% coefficients

4. Numerical values with selectable units – these are combination inputs that contain the units used as well as the value within those units. An example would be Temperature in the SSPDCalc module: It can be set as Celsius(24.869) which sets it to 24.869 degrees C.

e. Filespecs – Within SuperFAR scripts, these can be set as absolute or relative pathspecs concatenated with exact filenames. If no pathspec is given, then it is assumed that the current directory containing the script is the path. The example scripts include the setting of a local EventID variable that contains the base filename for all input and output files. This is typically a multi-character identifier unique to a particular aircraft noise event. Filespecs within SuperFAR scripts typically concatenate predetermined pathspecs and suffixes and extensions to these EventIDs. Example:
   - For a notional script located in C:/Test/Pref/ :
   - EventID = "A123"
   - InputData = EventID + "raw.STH.csv"
   - OutputData = "results/" + EventID + cooked.STH.csv"
   - This will access input file C:/Test/Pref/A123raw.STH.csv and generate output file C:/Test/Pref/results/A123cooked.STH.csv

   Note that when changing the sequence of operation within a script, filespecs for input and output files will likely need to be updated within the script to reflect the new process sequence. As an example, if slow time-averaging in ReAvg is performed in the original script after background noise adjustment via Badjer has been run, and the new modified script will perform slow time-averaging prior to performing background noise adjustment (ReAvg then Badjer), then the following example changes will need to be made to particular filespecs:

   Original script (Badjer then ReAvg):

(Badjer) input_STHData = EventID + "raw.STH.csv"
(Badjer) ADDCORR_STHData = "results/" + EventID + "BADJER.sth.csv"
(ReAvg) input_STHData = "results/" + EventID + "BADJER.sth.csv"
(ReAvg) output_STHData = "results/" + EventID + "SLOW.STH.csv"

New modified script (ReAvg then Badjer):

(ReAvg) input_STHData = EventID + "raw.STH.csv"
(ReAvg) output_STHData = "results/" + EventID + "SLOW.STH.csv"
(ReAvg) input_STHData = "results/" + EventID + "SLOW.STH.csv"
(Badjer) ADDCORR_STHData = "results/" + EventID +BADJER.sth.csv"

(Also note that the next module called, which originally used ["results/" + EventID + "SLOW.STH.csv"] as input, will now need to be edited to use ["results/" + EventID + "BADJER.sth.csv"] .

f. Shell – Interaction with the environment's shell can be performed from within scripts. (Refer to Python and IDLE references for more information.) The example script includes simple print statements to establish points of reference during a run where messages from the environment are being generated. For example, prior to calling each module, there is a simple print statement including the module's name, e.g.:

Print("Badjer")

The example scripts each also contain a print statement to indicate that the script has been completed:

Print("run completed")

Note that shell interaction is not available when running SuperFAR Python scripts based on the run_script.bat batch file. (Shell output from such batch runs is saved in .log files.)

5. Data Files

SuperFAR data files have been designed to be self-documenting to a large extent. The files are formatted as plain-text, with comma-separated values, so that they can be easily imported into spreadsheet software and text editors. The files are comprised of three main components: Header, Column Labels, and data values.

a. Header:

SuperFAR file headers contain machine-readable and human-readable information about the data file, including the name, creation date, user-provided comments, and important data values used in various processes, as well as information about the source data files and modules used to generate them. Each header element includes a label, designated by the last two characters being double asterisks: "**". (Common labels include "FileType**", "FileName**", FileDateTime**", ProjectName**", MicrophoneID**", etc.) Each of the header labels is followed by a comma, then by the value for that header element. Note that the sequence of Header fields is not important for machine-reading. Fields are identified and located by their labels. See the example SuperFAR file header fragment below:

FileType, Spectral Time-History**
FileName, CLTO1.sth.csv**
FileDateTime, 6/25/2009, 11:15 am**
ProjectName, anonymous Validation**
MicrophoneID, na**
AveragingMethod, LINEAR**
TimeStampType, MIDPT**
AdjustmentCode, No adjustments**
StartTime,12, 26, 29.5**
ReferenceTime, 12, 26, 29.5**
ReferenceTimeType, START**
GeneratedBy, manually from .csv file**
NumberOfGenerationFiles, 1**
GenFileName1, CLTO1-CSV.SPC.csv**
GenFileDateTime1, 6/18/2009, 10:22 am**
OtherRecords, 0**
NumberOfCommentLines, 1**
Anonymous Data,

b. Column Labels:

At the bottom of the file header section is a line of column labels for the data contained in the file. There is one label for each column of data values in the file, and each label is separated by a comma from the next. In some cases, there will be more than one line of Column Labels. When viewing data files in a text editor, users should be aware that column labels most likely will not

be aligned with the data values associated with that column. For ease of use, viewing data files from within a spreadsheet program like Excel is recommended.

Example Column Labels line excerpt:

Rec#, TODHH, TODMM, TODSS, RelTime, B17/50Hz, B18/63Hz, …

c. Data Values:

When the default application associated with .csv files is set as Excel in the operating system (refer to Microsoft Windows Help for the process for doing this), double-clicking on a SuperFAR data file will open it in a new tab in a new Excel worksheet. Data can be cut and pasted between Excel sheets / SuperFAR data files, and saved in .csv format to maintain compatibility with SuperFAR. This compatibility allows for hand-calculations or external processes on data between SuperFAR operations. In such cases, separate SuperFAR scripts should be created for the processing sequences prior to and subsequent to any spreadsheet processing.

Example Data line excerpt:

3, 07, 28, 43.542, 78.923, 81.1, 79.99, …

6. File preparation for initial measured data inputs

Prior to SuperFAR processing, raw data obtained from instrumentation in the field must be prepared as SuperFAR-formatted data files. This includes conversion of tabulated data values – such as 1/3 octave band SPLs, physical position distances from reference location, or temperature and relative humidity vs. height – to comma-separated-values format, as well as population of data file header information. Filenaming and data organization are also key elements for consideration, and will require editing of existing scripts or development of new scripts for proper processing.

Listed below are details on preparing the inputs required to run a SuperFAR process

a. Spectral Time-History data from analyzer
    1. Real-time 1/3 octave band analyzers typically output the individual sound pressure levels (SPLs) as a two-dimensional spectral time-history, with an individual 1/3 octave band frequency spectrum provided as a row of values, and multiple spectra obtained over time as a series of such rows. This is the format SuperFAR uses. Since the noise certification regulations specify processing of 1/3 octave band data from 50 Hz through 10 kHz (ANSI/ISO bands 17 through 40, inclusive), SuperFAR is preset to use these bands. (SuperFAR was designed to accommodate a wider bandwidth, but since there are no algorithms for some portions of the EPNL process outside this region, this flexibility has not yet been implemented.) SuperFAR will accept whatever resolution of SPLs is available from the analyzer – no rounding or truncation is necessary, although SPL outputs are typically provided to 0.01 dB in .SSR (Single spectrum record) or .STH (spectral Time-History) output data files. (SuperFAR provides additional resolution for some specific decibel outputs, such as bandsharing in EPNL.RPT files)
    2. A record number and “timestamp” precede the 1/3 octave band SPLs in each spectrum row in an .STH file, and are critical to synchronization between acoustic data and aircraft position data, and to a lesser extent, to meteorological data. Note that SuperFAR time-handling functions allow for I/O of time data as separate fields for hours, minutes, and seconds, or as total seconds since midnight, or other formats, but care should be exercised when including spreadsheet software such as Excel in the process, as it uses its own internal representation of time, which is not entirely compatible with SuperFAR. This incompatibility stems from the need for observation and inclusion of fractional seconds while still maintaining time-of-day within SuperFAR processes. Note that SuperFAR also includes the capability of working with relative time as an option to using time-of-day. There is also provision for specifying a reference time when relative time is utilized.
    3. Specific header information should be provided that is relevant to the analysis project.

b. Position Time-History data
    1. SuperFAR requires aircraft position information in local coordinates, where the centerline microphone is positioned at 0,0,0. A position time-history (PTH) dataset, containing X, Y, and Z coordinates of the aircraft vs. time, synchronized to the timebase for the acoustic measurements can be input directly, or after smoothing or other external reduction processes to the SuperFAR PTH file format: the file header should contain information about the project, the date of measurement and other details, followed by position records, each consisting of a single row of T,X,Y,Z measurement data for a moment in time. The time in each position record, Tp, should be the time at which each aircraft position measurement was made. (These position time histories typically come from DGPS instrumentation.)
    2. In some cases, direct position measurement histories are not available, such as when using photographic positioning techniques and still cameras. In such cases, a set of Single-Point Track descriptors are obtained externally to SuperFAR, and entered as inputs to the SPoinTrkIn module, which then creates a straight-line position time history file for use in determining aircraft noise geometry.
    3. Once obtained, the position time history data are used as inputs to the GeoCalc module, which in combination with acoustic spectral data timestamps, microphone position and meteorological information (to obtain soundspeed), generates aircraft noise geometry for the time of emission of each spectrum in the spectral time-history file. These geometry data elements are used for reconstruction of masked SPLs as well as for adjustment of test-day SPLs to reference conditions.

c. Meteorological Profiles
    1. SuperFAR requires temperature and relative humidity values measured at various height increments above the ground near the measurement site. Typically, an instrumented aircraft, weather balloon, or model aircraft performs meteorological flights in between series of aircraft noise test events. Each of these “met flights” results in a met profile that is represented by a single xxxx.MET.csv file. Measurement heights must include 10 meters (or 32.8 feet), as well as measured values for the upper and lower boundaries of atmospheric “layers” of 100 feet (or 30 meters) in depth. (These temperature and humidity values at layer boundaries can be determined externally by interpolation between actual met measurement heights prior to generation of the input met files.) Each met profile can be assigned a single time-of-day, or measurements at individual heights can each have their own measurement TODs associated with them.

2. For each aircraft noise event to be evaluated, determine an event time-of-day (ETOD), which can be any of the following:
    i. TOH – time when aircraft is overhead or abeam of the measurement microphone;
    ii. TCPA – time at which aircraft is at the geometrical closest-point-of-approach (minimum distance) to the measurement microphone;
    iii. TMAX – time at which the maximum PNLT level was measured (Note that this time cannot be obtained precisely prior to development of atmospheric absorption coefficients – alphas – which are based on measured temperature and humidity data, so approximation and / or recursion must be employed to obtain meaningful values of both TMAX and the cumulative test-day alphas.);
    iv. OTHER – any other TOD used to represent the aircraft event;
3. Identify the MET profile data closest in time **prior to** the aircraft noise event of interest. This data must be formatted and saved as a **xxxx.B4.MET.csv** file. Identify the MET profile closest in time **subsequent to** the aircraft noise event of interest. This data must be formatted and saved as a **xxxx.AFTR.MET.csv** file. Note that both MET files must use the same temperature units and distance units, and the heights included in the two profiles must match each other, and must represent measured values at the upper and lower boundaries of the atmospheric layers (10 meters and 100 foot or 30 meter increments from the ground). When times vary within a single MET profile, care should be taken to ensure that all of the TOD values in the xxxx.B4.MET.csv file occur prior to the aircraft event TOD (ETOD), and that all of the TODs in the xxxx.AFTR.MET.csv file occur after the ETOD.
4. The TDMet module will interpolate the temperature and relative humidity values in the xxxx.B4.MET.csv and xxxx.AFTR.MET.csv files to the aircraft ETOD. Then it will check if layering is required and will determine layered/averaged and cumulative atmospheric absorption coefficients (Alphas) for use in reconstruction of masked data and adjustment to reference conditions. These meteorological values will also be used for computation of soundspeed for use in noise geometry calculations dependent on sound propagation times. (Note that in some cases, cumulative alphas and soundspeed values may already be provided for a particular data set. In such cases, it is possible to skip preparation of .MET files.)

d. Microphone position data
    1. SuperFAR works with measured data for a single microphone at a time. The input data folder must include a xxxx.MIC.csv file which contains the microphone designation, the microphone site X, Y, and Z coordinates, and the microphone height above the local ground surface. Provision is made for including a microphone orientation angle, which could be used in determining

sound incidence angles for determination of non-uniform free-field corrections, but this functionality has not yet been implemented.

e. Background noise
    1. For each aircraft noise event, an average 1/3 octave band spectrum of pre-detection background noise levels is required. The average SPLs should be obtained by performing a 30-ssecond linear time average (or LEQ) of typical background noise at the microphone site that represents the background noise during the aircraft noise measurement. Pre-detection levels should be obtained at the same gain and sensitivity settings as the aircraft noise, so that additive system background noise is properly included. The average pre-detection SPLs should be provided in a file named “xxxx.PreD.SSRcsv”. (“SSR” stands for “Single Spectrum Record”.)
    2. Also required for each event is a 1/3 octave band spectrum of post-detection background noise levels. This data is required to identify the minimum level below which measured aircraft SPLs are considered to be non-valid. Pre-detection noise represents limitation in the recording and analysis system that are not additive in nature, such as amplitude windowing limits, or amplitude non-linearity exceedances. As such, post-detection levels can be shared among multiple aircraft noise datasets if appropriate. The post-detection noise levels should be provided in a file named “xxxx.postD.SSR.csv”.

f. Correction data
    1. Frequency-dependent corrections for test-day measurement system effects, including elements such as microphone pressure response, microphone free-field response, windscreen insertion effects, and system frequency-response testing, should be combined and provided in a single 1/3 octave band spectrum in a file named “xxxx.CORRS.SSR.csv”. These corrections should be determined in a way that allows them to be ADDED to the aircraft noise SPLs to properly account for frequency-dependent deviations in response.
    2. Broadband corrections – that is corrections that should be applied to all frequency bands equally – such as gain adjustments and sound calibrator corrections, as well as system sensitivity “drift” corrections (obtained by averaging or interpolating between sensitivity calibrations performed in the field before and after aircraft noise measurements) should be combined and applied as a script input to the BADJER module.

APPENDIX 1 - SuperFAR Module Reference

This appendix contains information and specifications on each of the implemented SuperFAR modules. Included in this reference are brief descriptions of the modules' functionality, information about processing options, suggested sequences of operation, and the names of all inputs and outputs, including filespecs (path and filename), variables, and control options.

Alphabetical listing of all currently-implemented SuperFAR modules:

a. ARP866A (Calculation of atmospheric absorption coefficients – alphas – per SAE ARP866A);
b. BADJER (Background noise Adjuster);
c. EPNLCalc (Effective Perceived Noise Level Calculations);
d. GeoCalc (test-day aircraft noise Geometry Calculations);
e. Integrated (Integrated procedure for adjustment to reference conditions);
f. Metrix (Metrics calculations from 1/3 octave band SPLs);
g. ReAvg (Re-averaging – simulation of slow timing to linear SPLs);
h. ReConstruct (Re-Construction of masked SPLs);
i. RefGeo (Reference condition aircraft noise Geometry);
j. SimpleStats (Basic Statistics for single-event levels for multiple events);
k. Simplified (Simplified procedure for adjustment to reference conditions);
l. SPoinTrkIn (Single-Point Tracking data Input – to obtain TXYZ history of aircraft position);
m. SPoinTrkOut (Single-Point Tracking data Output – to obtain single-point tracking descriptors from TXYZ time-history);
n. SSPDCalc (Calculation of test-day soundspeed, based on average temperature);
o. SSPDTemp (Calculation of average test-day temperature for soundspeed calculation);
p. TDMet (Test-Day Meteorological data – obtains cumulative Test-Day 1/3 octave band alphas);

a. **ARP866A** –

- Calculates atmospheric absorption coefficient - “alpha” - for a specified frequency based on input temperature and relative humidity per the algorithms presented in SAE Aerospace Recommended Practice ARP866A;

Inputs:

- Temperature (Fahrenheit, Celsius, Kelvin, [Rankine – not yet implemented]);
- Relative Humidity (%);
- Frequency (Hz) – note that if nominal center frequencies for ANSI bands 37 through 40 (5 kHz through 10 kHz) are included as inputs, the nominal lower band-edge for these bands will be substituted within the module, per ARP866A usage for aircraft noise certification (If absorption rates for the nominal center frequencies for these bands are desired, a work-around is to input a frequency value that is 1 Hz higher or lower than the desired frequency);
- Alpha Type: Select ‘dB1kft’ or ‘dB100m’;

Output:

- Alpha – a single atmospheric absorption coefficient value;

b. **BADJER** - Background noise Adjuster (combination of MaskMan and ValidAdj) –

- Identifies valid pre-detection background noise levels;
- Establishes masking criteria based on input window values (Defaults: Pre-detection + 3 dB; Post-detection + 1 dB);
- Identifies masked aircraft SPLs;
- Generates masking map for spectral time-history;
- Performs energy-subtraction of valid Pre-detection noise from valid aircraft SPLs;
- Applies overall gain and correction adjustments to all aircraft SPLs;
- Applies frequency-dependent system and microphone response corrections to valid aircraft SPLs.

Inputs:

- postdetectWindow – decibel value (default = 1.0 dB)
- predetectWindow – decibel value (default = 3.0 dB)
- postdetectSSR – filespec for 1/3 octave band spectrum of **post-detection** (non-additive) background noise SPLs
- predetectSSR – filespec for 1/3 octave band spectrum of average **pre-detection** (can contribute energy to measured noise levels – such as ambient noise at the

microphone site, or thermal electronic noise within the measurement system) background noise SPLs for a particular run

- STH – filespec for raw, 1/3 octave band aircraft noise spectral time-history
- correctionSSR – filespec for combined 1/3 octave band spectrum of system response corrections, including microphone pressure and free-field response, and windscreen insertion loss, along with system frequency response determined from pink noise or pure-tone testing
- correctionValue – decibel value for overall broadband correction, including any gain change, sound calibrator corrections, or system sensitivity "drift" correction.

Outputs:

- validpreSSR – filespec for 1/3 octave band spectrum (.SSR) of valid pre-detection SPLs
- maskcritSSR – filespec for 1/3 octave band spectrum (.SSR) of masking criteria SPLs
- badjerSTH – filespec for 1/3 octave band spectral time history (.STH) after all steps of BG noise adjustment have been performed
- badjerMap – filespec for intermediate masking map (.MAP) after all steps of BG noise adjustment have been performed
- ambisubSTH – filespec for intermediate 1/3 octave band spectral time history (.STH) after energy-subtraction has been performed
- ambisubMap – filespec for intermediate masking map (.MAP) after energy-subtraction has been performed
- gainadjSTH – filespec for intermediate 1/3 octave band spectral time history (.STH) after overall corrections have been applied

c. **EPNLCalc** – Calculates EPNL from xxxx.MTX.csv file generated by Metrix module:

- Identifies the record having the maximum value for each user-selected metric
- Identifies the first and last 10 dB-down points (the limits of the Noise Duration) for each user-selected metric
- Identifies any secondary peaks (PNLT values that are within 2 dB of the maximum PNLT value)
- Determines if a bandsharing condition is present and, if so, applies a bandsharing correction to the maximum PNLT value prior to energy integration
- Determines Effective Interval for each spectrum, based on spectrum timestamps
- Integrates the PNLT time-history over the noise duration (integrates each other user-selected metric over its own noise duration)

- Computes EPNL from PNLT using a reference duration of 10 seconds (Computes integrated values for other metrics using a reference duration of 1 second, except for PNL, where a 10 second reference duration is also used.)
- NOTE: Metrix module must have been run on spectral time-history prior to running EPNLCalc.

Inputs:

- InputMTX – the input metrics time-history file (.MTX)
- Comments – Provide a comment to appear in the output file header (no commas)

Outputs:

- epnlMTX (.MTX)
- EPNL (.RPT)

Note: This module is also called from Integrated, for use in determining Integrated Reference Condition EPNL.

d. **GeoCalc** - Geometry Calculation

Calculates test-day aircraft noise emission geometry relative to the microphone of interest.

Inputs:

- PTH – Position-Time History file for event of interest (.PTH)
- STH – Aircraft noise spectral time-history file, in the example script, the final output file from the Badjer module (.STH) – this data set is used to obtain measurement time for each spectrum
- MIC – Microphone coordinates and height (.MIC)
- SPO –Single-Point track information (.SPO)
- SoundSpeed – The speed of sound, as calculated by the SSPDCalc module
- NOTE: A position time-history of TXYZ and a spectral time-history, complete with Tm(k) measurement “timestamps” must be available prior to running this module.

Outputs:

- GTH – Geometry time-history file (.GTH)

e. **Integrated** – Adjusts test-day noise data to reference conditions using the integrated method defined in Annex 16 Appendix 2 and part 36 Appendix A.

NOTE: Full test-day processing, including a test-day EPNL and metric time-history, as well as test and reference day noise geometry must be available prior to running this module.

Inputs:

- tdSTH – the final test-day SPL spectral time-history data
- tdMAP – the map associated with the final test-day SPLs
- tdATH – the test-day atmospheric absorption coefficients (.ATH alpha time-history)
- tdALF – same as ATHData, but a single spectrum of alphas (.ALF)
- refALF – the reference condition atmospheric absorption coefficients (single spectrum) obtained using ARP866A algorithms and 25°C/70% RH pair
- tdEPNL – the test-day EPNL report
- refGTH – the test-day and reference condition noise geometry time-history
- Flags to select desired metrics:
    - AFlag – Logical value: when TRUE, includes ANSI A-weighted metric
    - CFlag– Logical value: when TRUE, includes ANSI C-weighted metric
    - DFlag– Logical value: when TRUE, includes ANSI D-weighted metric
    - OAFlag – Logical value: when TRUE, include unweighted metric
    - PFlag – Logical value: when TRUE, include PNL metric (required for EPNL)
    - TFlag – Logical value: when TRUE, include PNLTM metric (required for EPNL)
- Process-control flags:
    - TCB40Flag – Logical value: when TRUE, perform tone correction through band 40 (Otherwise stops tone-correction at LGB)
    - NoRoundFlag – Logical value: when TRUE, do not round SPLs to 0.1 dB before tone correction (ETM Guidance suggests rounding SPLs to 0.1 dB - within the tone-correction process only – in order to ameliorate artificial sensitivity to very small differences in 1/3 octave band SPLs)
    - HelicopterFlag – Logical value: when TRUE, start tone correction at 50 Hz instead of 80 Hz
    - TCLowBand – Optional lower limit for tone-correction when eliminating pseudotones: ANSI/ISO band number for lowest band to be included. (Replacement for TC1KFlag, which used to switch hard-wired B30 for elimination of p-tones).

Outputs:

- epnlMTX – the metrics time-history for the reference condition data set
- refSTH – the reference condition SPL spectral time-history
- EPNL – the reference condition EPNL obtained using the integrated method

f. **Metrix** – Computes metric values for each record in a spectral time-history (.STH)
   i. Applies selected frequency-weighting to each one-third octave spectrum
   ii. Calculates broadband level for each selected weighting
   iii. If selected, calculates Perceived Noise Level (PNL), Tone-Correction (TC), and PNLT, saving them to a metrics time-history file (.MTX)
   iv. Compares Masking and adjustment codes from the .MAP files for individual bands in each spectrum to criteria provided in the Background Noise Adjustment Procedure, which Volpe developed for AC36-4C and the ETM.
   v. Generates NVR (Non Valid Record) code for each spectrum, and saves them in the .MTX file.

- NOTE: A spectral time-history, complete with Tm(k) measurement "timestamps" must be available prior to running this module. In some processes, this module may be run more than once, for example prior to and subsequent to simulating slow time-averaging.

  Inputs:
  - STH – the SPL spectral time-history to obtain metrics for
  - Map – the associated map data
  - Flags to select desired metrics:
    - AFlag – Logical value: when TRUE, includes ANSI A-weighted metric
    - CFlag– Logical value: when TRUE, includes ANSI C-weighted metric
    - DFlag– Logical value: when TRUE, includes ANSI D-weighted metric
    - OAFlag – Logical value: when TRUE, include unweighted metric
    - PFlag – Logical value: when TRUE, include PNL metric (required for EPNL)
    - TFlag – Logical value: when TRUE, include PNLTM metric (required for EPNL)
  - Process-control flags:
    - TCB40Flag – Logical value: when TRUE, perform tone correction through band 40
    - NoRoundFlag – Logical value: when TRUE, do not round SPLs to 0.1 dB before tone correction
    - HelicopterFlag – Logical value: when TRUE, start tone correction at 50 Hz instead of 80 Hz
    - TCLowBand – Optional lower limit for tone-correction when eliminating pseudotones: ANSI/ISO band number for lowest band to be included. (Replacement for TC1KFlag, which used to switch hard-wired B30 for elimination of p-tones).
  - Outputs:
    - MTX – the metrics time-history for the input data set

g. **ReAvg** – simulation of slow time-averaging –

- Applies simulated slow time-response weighting to data obtained using linear averaging in the analyzer using one of two methods: continuous or multi-sample running average
- NOTE: This module can be run at various points in the process. If it is performed after determination of test-day noise geometry, it may be necessary to rerun GeoCalc with the newly-obtained slow spectral time-history in order for reconstruction of masked SPLs and/or adjustment of test-day SPLs to reference conditions.

Inputs:

- inputSTH – filespec for linear 1/3 octave band spectral time-history (.STH)
- input_MapData – filespec for masking and adjustment map (.MAP) for linear data
- AveragingMethod – enumerated input:
  - 'CONTEXPO' – continuous exponential function
  - '4S100' – four-sample running average, with coefficents that sum to 100%
  - '4S95' – [DEPRECATED] four-sample running average, with coefficients that sum to 95% (Was developed for a particular hardware analyzer from the 1980's)
- TimeStamp – enumerated input:
  - 'ICAOSLO' – current Annex and part 36 specification: time for slow average is assigned to the instant 0.75 seconds prior to the output of the most recent linear sample within the average
  - 'OLDSLO' – obsolete part 36 specification for midpoint of the notional two-second averaging period
  - 'START' – start of the linear sample
  - 'MIDPT' – [DEPRECATED] mid-point of the averaging period (Still applies for linear data; would have been 1.0 seconds prior to the output of the most recent sample within the average for slow data)
  - 'END' – output time of the linear sample

Outputs:

- outputSTH – filespec for slow 1/3 octave band spectral time-history (.STH)
- outputMap – filespec for slow masking and adjustment map (.MAP)

h. Reconstruct – Performs Reconstruction of masked SPLs

- Performs averaging of adjacent SPLs for a single masked LF SPL
- Performs frequency-extrapolation from highest-frequency valid SPL for HF SPLs (up to 7 highest-frequency bands masked)
- Performs time-extrapolation from nearest-in-time valid SPL in a particular HF band (7-12 highest-frequency bands masked)
- NOTE: Test-day Noise geometry and atmospheric absorption coefficients (alphas) must be available prior to running this module.

Inputs:

- TXRec [optional] – default: 1 – recnum for spectrum to be used as the center point for time-extrapolation
-
- TXCType [optional] – method of identifying center for timex (‘Slice’ , or ‘Spectrum’)
- VCAF03Flag – default: TRUE - Logical value: When TRUE (default), uses Volpe 2003 method (same as ICAO ETM method); uses LGB concepts and assumes flat aircraft spectrum at a distance of 60 meters from the source under reference atmospheric conditions
- AC364BFlag – default: FALSE - Logical value: when TRUE, uses the frequency extrapolation method provided in AC36-4B, spectrum flat at zero distance from the source.
- FixedSlopeFlag – default: FALSE - Logical value: when TRUE, applies a fixed slope in dB per one-third octave band
- Use Masked Flag – Logical value: when TRUE, uses the masked value when a masked aircraft SPL is less than the reconstructed value.
- BirdBugFlag – Logical value: when TRUE, allows use to select a particular band to reconstruct in a special manner when many records have contamination in a single band due to buzzing or chirps. (Not yet implemented).
- TXOutwardFlag [optional] – default: TRUE – Logical value: when TRUE, searches inward toward the Maximum SPL in a band for a valid SPL.
- AvAdjFlag – Logical value: when TRUE, allows the reconstruction of single, masked, low-frequency band SPLs by averaging the SPL values of the adjacent bands.
- FreqXFlag – Logical value: when TRUE, allows reconstruction of masked high-frequency bands by frequency-extrapolation.
- TimeXFlag – Logical value: when TRUE, allows reconstruction of masked high-frequency bands by time-extrapolation.
- ffSR – distance value: when using the ETM frequency extrapolation method, provide the “free-field” distance (defaults to 60m).
- inputSTH – input test-day SPL time-history file (.STH) – identification of masked SPLs must be available (BADJER must have already been run), and test-day noise geometry and cumulative alphas must be available
- inputMap – the associated masking and adjustment map file (.MAP)
- inputGTH – test-day aircraft noise geometry time-history file (.GTH)
- tdATH – test-day atmospheric absorption coefficients (“alphas”) as an alpha time-history (.ATH) with a separate spectrum of alphas for each acoustic data record in the .STH file.
- tdALF – test-day atmospheric absorption as a single spectrum (.ALF)

- ffALF – reference condition alphas to be used for propagation effects over the free-field distance of 60m (.ALF)

Outputs:

- OutputSTH – final output combining valid SPLs and SPLs reconstructed using all methods
- OutputMap – the map for the final Reconstruct SPL output
- AvgAdjSTH – diagnostic output: includes valid SPLs and SPLs that have been reconstructed using the average adjacent method
- AvgAdjMap – the associated map for valid SPLs and SPLs reconstructed using the average adjacent method
- FreqSTH – diagnostic output: includes valid SPLs and frequency-extrapolated SPLs
- FreqMap – the associated map for valid SPLs and SPLs reconstructed using the frequency-extrapolation method
- TimeSTH - diagnostic output: includes valid SPLs and time-extrapolated SPLs
- TimeMap – the associated map for valid SPLs and SPLs reconstructed using the time-extrapolation method

i. **RefGeo** – Calculates reference condition noise geometry relative to the reference microphone based on test-day acoustic emission angle single-point reference track information:
    - Calculates the reference condition emission coordinates, sound propagation distances, reception times, and effective intervals
    - See ICAO ETM Vol. I, Chapter 4, Section 4.3.1.2
    - Assumes that test-day noise geometry has already been obtained by running GeoCalc

    Inputs:

    - SPO (.SPO) [optional] – single-point track descriptors for test-day noise geometry (if not available in SPO file, then can be provided as individual inputs in script)
    - tdSTH (.STH) – test-day SPL time-history, used to obtain test-day spectrum measurement times
    - tdGTH (.GTH) – test-day noise geometry
    - TOHR – time string input: the reference time at overhead as “hh:mm:ss.sss”
    - ZOHR – distance input (feet, meters): the reference condition aircraft height above the ground at overhead

- YMICR – distance input (feet, meters): the lateral reference microphone horizontal offset from the reference ground track (typically 150m or 450m – 0 for centerline)
- VGR – speed input: the reference condition groundspeed ('KTS', "MPH', "KMH', 'FPS', 'M/S')
- RGAMMA – degrees input: the reference condition climb/descent angle
- TOH – test day time at overhead [only required if no .SPO file provided]
- ZOH – test day height at TOH [only required if no .SPO file provided]
- VG – test-day groundspeed [only required if no .SPO file provided]
- GAMMA – test-day climb/descent angle [only required if no .SPO file provided]
- CPA – test-day closest-point-of-approach [only required if no .SPO file provided]

Outputs:

- refGTH (.GTH) –test-day and reference condition noise geometry time-history

j. **SimpleStats** – performs simple statistical functions on noise levels from a group of events, including average level, standard deviation and 90% confidence interval per the guidance in ICAO's ETM, Vol. I.

NOTE: This module is typically run after EPNL (or other final noise metric) has been computed for a series of events.

Inputs:

- StatMethod – enumerated selection:
  - 'clustered' – for measured data
  - 'regressed' – for data obtained analytically for similar conditions
- K – when method is "regressed", enumerated selection for type of regression:
  - '1' – linear
  - '2' – quadratic
  - '3' - cubic
- data – filespec for input of user-compiled table of event IDs and associated noise levels

Outputs:

- report – filespec for output – values for average level, standard deviation, 90% confidence interval, and deltas and squared deltas

k. **Simplified** – Adjusts test-day data to reference conditions using the simplified method

- Adjusts 1/3 octave band SPLs in the test-day Maximum PNLT spectrum (and any secondary peaks within 2 dB of PNLTM) to reference conditions accounting for differences in test & reference sound propagation distances and test & reference atmospheric absorption. (Actually adjusts every spectrum)
- Computes the following simplified adjustment terms:
    1. DEL1 – the difference in dBPNLT between PNLTMR and the test day PNLTM (includes test-day delta for bandsharing in both values)
    2. DELPEAK – an adjustment to DEL1 if any of the secondary peaks within 2 dB of PNLTM adjusted to a higher level than the PNLTM spectrum
    3. DEL2 – A duration adjustment for EPNL accounting for the differences in test & reference speeds and test & reference sound propagation distances
    4. DEL3 – a user-input source noise EPNL adjustment
- Computes the simplified reference condition EPNL by adding each of the simplified adjustment terms to the test-day EPNL
- NOTE: Requires that test-day process has been completed, as well as RefGeo, prior to execution of Simplified

Inputs:

- tdSTH - Test-day spectral time-history (.STH)
- tdMAP – map file associated with the test-day STH (.MAP)
- tdALF –single spectrum of test-day alphas (.ALF)
- tdATH – alpha time-history of test-day alphas (.ATH)
- refALF – spectrum of reference condition alphas (.ALF)
- tdEPNL – information about test-day EPNL (.EPNL.RPT)
- refGTH – Noise geometry time-history for test-day and reference conditions (REF.GTH)
- tdMTX – Test-day EPNL metrics time-history (EPNL.MTX)
- VG – average test-day groundspeed
- VGR – reference condition groundspeed
- DEL3 – Source noise adjustment (determined externally)
- Flags to select desired metrics:
    - AFlag – Logical value: when TRUE, includes ANSI A-weighted metric
    - CFlag– Logical value: when TRUE, includes ANSI C-weighted metric
    - DFlag– Logical value: when TRUE, includes ANSI D-weighted metric
    - OAFlag – Logical value: when TRUE, include unweighted metric
    - PFlag – Logical value: when TRUE, include PNL metric (required for EPNL)
    - TFlag – Logical value: when TRUE, include PNLTM metric (required for EPNL)

- Process-control flags:
    - TCB40Flag – Logical value: when TRUE, perform tone correction through band 40
    - NoRoundFlag – Logical value: when TRUE, do not round SPLs to 0.1 dB before tone correction
    - HelicopterTCFlag – Logical value: when TRUE, start tone correction at 50 Hz instead of 80 Hz
    - TCLowBand – Optional lower limit for tone-correction when eliminating pseudotones: ANSI/ISO band number for lowest band to be included. (Replacement for TC1KFlag, which used to switch hard-wired B30 for elimination of p-tones).

Outputs:

- refSTH – reference condition spectral time history (REF.STH)
- outputMTX – reference condition metrics time history (REF.MTX)
- refEPNL – simplified EPNL report (SIMP.REF.RPT)

l. **SPoinTrkIn** –

- Calculates a straight line vector and generates a Position Time History (PTH) file from Single-Point Track Descriptor inputs.
- NOTE: if only single-point tracking data are available for a particular event, SPoinTrkIn must be run prior to the calculation of test-day noise geometry using GeoCalc.

Inputs:

- TOH – Time at OverHead – the moment in time at which the aircraft is either directly overhead or abeam of the microphone. The position along the X axis at this moment, XOH, is assumed to be zero:
    - Note that TOH can be entered as separate hours, minutes, and seconds, or as total seconds elapsed since midnight;

- Yoff – The lateral offset from the reference flight track at TOH;

- AltOH – The aircraft height above the ground at TOH;

- GAMMA – The average climb/descent angle of the aircraft flight path (Note that per recent updates to Annex 16 and the ETM, the name for this angle is now “GAMMA”. Previously, Volpe used “PHI”, as well as “CD”, and some documentation may still reflect this):
    - Note that GAMMA can be entered as an angle (when input_PHI_Flag is TRUE), or as a gradient slope percentage (when input_CDSlope_Flag is

TRUE), or as individual “Rise” and “Run” values (When input_CDRiseRun_Flag is TRUE);

- CHI – The average horizontal cross-track angle of the aircraft flight path (Note that per recent updates to Annex 16 and the ETM, the name for this angle is now “CHI”. Previously, Volpe used “CYang”, as well as “GAMMA” – [awkward] – and some documentation may still reflect this):
    - Note that CHI can be entered as an angle (when input_CYAng_Flag is TRUE), or as a gradient slope percentage (when input_CYSlope_Flag is TRUE), or as individual X and Y values (When input_CYYYX_Flag is TRUE);

- GSPD – The average groundspeed of the aircraft along the flight path:
    - Note that Groundspeed can be entered directly when input_GSPD_Flag is TRUE, or speed along the flight path (“VSPD”) can be entered if input_GSPD_Flag is FALSE;

- StartTOD – The time-of-Day for the first TXYZ record in the output PTH:
    - Note that StartTOD can be entered as separate hours, minutes, and seconds, or as total seconds elapsed since midnight;

- EndTOD – The time-of-Day for the last TXYZ record in the output PTH:
    - Note that EndTOD can be entered as separate hours, minutes, and seconds, or as total seconds elapsed since midnight;

- TInt – the time interval between TXYZ records in the output PTH – defaults to 0.5 seconds
- input_PHI_Flag – [default: TRUE] when TRUE, enter GAMMA as an angle in degrees
- input_CDSlope_Flag – [default: FALSE] when TRUE, enter GAMMA as a vertical gradient slope percentage
- input_CDRiseRun_Flag – [default: FALSE] when TRUE, enter GAMMA as separate “rise” (vertical Z-axis) and “run” (horizontal X-axis) values
- input_CYAng_Flag – [default: TRUE] when TRUE, enter CHI as angle in degrees
- input_CYSlope_Flag – [default: FALSE] when TRUE, enter CHI as horizontal gradient slope percentage
- input_CYYYX_Flag – [default: FALSE] when TRUE, enter CHI as separate “rise” (horizontal Y-axis) and “run” (horizontal X-axis) values
- input_GSPD_Flag – [default: TRUE] when TRUE, input groundspeed; when FALSE, enter Vector speed

Outputs:

- PTHData – Position Time History file (Xxxx.PTH.csv)

m. **SPoinTrkOut** – Single Point Track Output

- Generates average single-point track geometry values from an input Position Time-History or Geometry-Time History. Run after SPoinTrkIn in order to generate an .SPO file that contains the Single-Point Track information in a form that subsequent programs can read.
- NOTE: Should be run on test-day noise geometry time history prior to calculation of reference condition noise geometry using RefGeo.

Inputs:

- PTH – [preferred] filespec for Position-Time History (.PTH)
- GTH – [optional] filespec for Geometry-Time History (.GTH)
- MTH – [optional when a GTH file is input] metrics time-history (.MTX)
- MIC – filespec for microphone coordinates and height (.MIC)
- AvStart – [optional input when using .GTH and .MTX inputs] the start record for the averaging process
- AvEnd – [optional input when using .GTH and .MTX inputs] the end record for the averaging process

Outputs:

- SPO – filespec that contains Single-Point Tracking information (.SPO)

n. SSPDCalc – Sound Speed Calculator

- Computes the speed of sound in FPS from input temperature using user selection of one of five available methods.
- This module was previously integrated in the code for GeoCalc, but has now been isolated as a separate, independent module.

Inputs:

- Temperature – Average test-day temperature to be used in calculating test-day soundspeed for sound propagation (degrees Celsius or Fahrenheit).
- SSPDCalcType – Soundspeed calculation method selector
  - ICAO_FIXED (method provided in current ETM, based on Celsius and Kelvin):

$$1135.5 * \sqrt{[(Temp\ ^{\circ}\mathrm{C}) + 273.15]/298.15}\ \ FPS$$

  - RICKLEY (Older validations may have used this. Based on Fahrenheit and Rankine):

$$49.025 * \sqrt{(Temp\ °\text{F}) + 459.67}\ FPS$$

- ICAO_TM (erroneous equation provided in older version of ETM):

$$1125.9 * \sqrt{[(Temp\ °\text{C}) + 273.15]/293.15}\ FPS$$

- SUPR_EZ (Dave Read's very simple approximation, based on Beranek's EZ version; pretty good answers over range of temperature used for aircraft noise certification):

$$[1050.9 + 1.092 * (Temp\ °\text{F})]\ FPS$$

- BARANEK_EZ (Leo Beranek's simple approximation):

$$[(1053.5 + 1.067 * (Temp\ °\text{F})]\ FPS$$

Outputs:

- SoundSpeed [output as global variable] – Speed of sound based on selected calculation method. This variable is used as the SoundSpeed input to the GeoCalc module.

o. SSPDTemp – Calculates average Test-Day temperature for use in sound speed calculation

- Computes an average test-day temperature in one of seven methods, from either ETOD Met data or pre Met and post Met data sets.

Inputs:

- mode – method for determining average test-day temperature:
  - 1 – Preferred (Requires ETOD Met): Interpolate ETOD temperature over height to height of aircraft (ZOH or ZPNLTM), then averages ETOD temps at aircraft height and 10m;
  - 2 – ETOD Approximation (Requires ETOD Met): Use ETOD temperature at closest Met height to aircraft height; average it with ETOD 10m temperature;
  - 3 – Preferred Alternate (Requires pre & post Met files): Interpolates pre & post Met temperatures over height to height of aircraft (ZOH or ZPNLTM). Then interpolates aircraft height temperatures and 10m temperatures over time to ETOD; finally, averages ETOD Temperature at aircraft height and ETOD 10m temperature;
  - 4 – Average Approximation (Requires pre and post Met files): Averages pre & post Met temperatures at Met height closest to aircraft height, and averages pre and post Met 10m temperatures; Finally, averages average temperature at met height closest to aircraft height with average 10m temperature;

  - 5 – ETOD 10m (Requires ETOD Met file): Use ETOD 10m temp directly;
  - 6 – ETOD 10m Alternate (Requires pre & post Met files): Interpolate 10m temperatures over time to ETOD;
  - 7 – Average 10m (Requires pre & post Met files): Averages pre and post 10m temperatures;
- Height – aircraft height (either ZOH or ZPNLTM);
- EtodMET – [optional – alternate is to input pre & post Met files] filespec for single input Met (Temperature and RH% vs. height) file;
- preMET – [optional if not using etodMET file] filespec for input Met file measured prior to aircraft noise event;
- postMET – [optional if not using etodMET file] filespec for input Met file measured subsequent to aircraft noise event;

Output:

- SoundSpeedTemp (output to shell only, must be captured as global variable within script):
  This line should follow the call to SSPDTemp, within the calling script:
  - SoundSpeedTemp = SSPDTempresult.SSPDAvgTemp

p. **TDMet** –

- Computes cumulative sound propagation-path atmospheric absorption coefficients ("alphas") from measured test-day weather profiles (temperature & relative humidity vs. height and time).
- Must be run prior to reconstruction of masked SPLs using Reconstruct, or adjustment to reference conditions using either Simplified or Integrated.

Inputs (Include measured temperature and relative humidity vs. height, at times before and after the aircraft noise event measurement):

- preMET – T&H vs. height, prior to the event (.MET);
- postMET – T&H vs. height, subsequent to the event (.MET);
- ETOD – The aircraft noise Event Time-Of-Day, a time chosen to represent the event;
- ETODType – An enumerated input. Choices include:
  - 'TOH' – Time at OverHead – the moment when the aircraft is directly over or abeam of the microphone, X=0.0;
  - 'TCPA' – Time at Closest Point of Approach – the moment when the aircraft is at either the notional or actual shortest straight-line distance from the microphone – When TXYZ data are available, TCPA can be determined by calculating the

slant distance between the aircraft and the microphone for each measured position, and the time associated with the shortest of these can be chosen. When a straight-line approximation of the flight path is used, CPA occurs when the acoustic emission angle is 90 degrees, or when the slant distance between the aircraft and microphone is normal to the flight path vector;
    - 'TMAX' – Time when the maximum PNLT spectrum is measured (In some cases, the emission time for this spectrum is used – the certification requirements and guidance are not clear: they only reference "time of PNLTM". Differences should be extremely minor – in fact, Volpe considers use of TOH or TCPA to be equivalent to TMAX.);
    - 'OTHER' – any other time chosen to represent the aircraft noise event time;

- ACHeight – the aircraft height at the "…time of PNLTM…" (- or at TOH or TCPA - Volpe considers all of these to be equivalent.);

- DistanceUnits [optional – defaults to units used in input files] – select the distance units to be used in output files (enumerated):
    - 'Feet';
    - 'Meters';
    - 'Miles' (should not be used for this module);
    - "NM' (Nautical miles – should not be used for this module);

- TemperatureUnits[optional] – select the temperature units to be used in output files (enumerated):
    - 'Fahrenheit';
    - 'Celsius';
    - 'Kelvin';
    - 'Rankine' [not implemented yet];

- AlfaType [optional] – select the alpha type to be used in output files (enumerated):
    - 'dB100M' – decibels per 100 meters;
    - 'dB1KFT' – decibels per 1000 feet;
    - 'dBM' – decibels per meter;
    - 'dBFT' – decibels per foot [not implemented yet];

- LayrCrit [optional, defaults to '0.5' dB per 100M if DistanceUnits="Meters", or '1.6' dB per 1000 ft if DistanceUnits = 'Feet'.] – allows user to input an alternate criterion for triggering atmospheric layering;

- ForceLay – Logical value; when TRUE, perform layering regardless of LayrCrit

- ForceFull – Logical value; when TRUE, perform full apportioning over sound propagation path; when FALSE, use simple average of layer alphas;

- SPFlag – Logical value; when TRUE, use single-point (typically 10-meter) alphas (this is standard for helicopters);

Outputs:

- ETODMet – Measured T & RH vs. height, interpolated over time to ETOD (the aircraft noise Event Time-Of-Day);

- ETODAlf – 1/3 octave band atmospheric absorption coefficients (“alphas”) for the T&RH pairs vs. height in ETOD.MET;

- LayrALF – 1/3 octave band alphas at the center of each atmospheric layer – used in computation of cumulative sound propagation path alphas which are used for reconstruction of masked data and for adjustment of measured test-day SPLs to reference conditions;

- AvgAlpha – The final cumulative sound propagation path alphas;

- FullAvgAlpha – Fully-apportioned cumulative alphas;

- SimpleAvgAlpha – “simple” cumulative alphas from layers;

- TwoPointAvgAlpha – 2-point cumulative alphas;

- OnePointAvgAlpha – single-point (10-meter) alphas;

APPENDIX 2 - SuperFAR Data File Specifications

For SuperFAR code to execute properly, input data files must be provided in plaintext, comma-separated-values format, with the header and data structures as specified in the following pages. The following file types are described:

1. Xxxx.ALF.csv – Atmospheric absorption coefficient (Alphas) single spectrum – can have multiple single-spectrum entries representing different heights
2. Xxxx.ATH.csv – Alpha Time-History – contains a single spectrum of alphas for each spectral record in the STH file
3. Xxxx.EPNL.MTX.csv – special EPNL Metrics Time-History data
4. Xxxx.EPNL.RPT.csv – EPNL report data
5. Xxxx.GTH.csv – Noise Geometry Time-History data – contains one geometry record for each spectral record in the STH file
6. Xxxx.Integ.REF.EPNL.RPT.csv – Integrated Method Reference EPNL Report data
7. Xxxx.MAP.csv – masking / adjustment map data (incl. masking/adjustment codes) – includes one map row for each spectral record in the STH file
8. Xxxx.MET.csv – Meteorological Profile (Temp & RH vs Height)
9. Xxxx.MIC.csv – Microphone coordinate data
10. Xxxx.MTX.csv – Metrics Time-History data – contains one row of broadband metrics for each spectral record in the STH file
11. Xxxx.PTH.csv – Position Time-History data – records are independent of spectral records in STH file
12. Xxxx.REF.GTH.csv – Reference condition noise geometry time history – contains one geometry record for each spectral record in the STH file
13. Xxxx.Simp.REF.EPNL.RPT.csv – Simplified Method Reference EPNL Report data
14. Xxxx.SPO.csv – Single-Point Tracking data report
15. Xxxx.SSR.csv – Single spectrum record of 1/OB SPLs
16. Xxxx.STATS.RPT.csv – Statistical data results report
17. Xxxx.STH.csv – 1/3 octave band spectral time-history

Note: the order of file Header fields is not important to SuperFAR. All modules can identify required header info from the Header Labels themselves. Also note that in the following file references, some header labels are presented in light gray text. These labels are either optional or in some cases, deprecated. Further, additional Header Fields beyond those presented in this reference may be found in some output files.

1) **Xxxx.ALF.csv** – Atmospheric Absorption coefficients (Alphas) data

Description:
A single spectrum of atmospheric absorption coefficients (“alphas”) in units of dB per 1000 feet (“dB1KFT”), dB per 100 meters (“dB100M”), dB per foot (“dBFT”), or dB per meter (“dBM”).

Header:

FileType**
FileName**
FileDateTime**
ProjectName**
MicrophoneID**
LAYERFLAG**
AlphaType**
GeneratedBy**
NumberOfGenerationFiles**
OtherRecords**
NumberOfCommentLines**
AlphaUnits**
DistanceUnits**
ETODType**

Column Labels:

TODHH TODMM TODSS Alpha17:50Hz Alpha18:63Hz Alpha19:80Hz Alpha20:100Hz Alpha21:125Hz Alpha22:160Hz

… [continues through Alpha40: 10kHz]

Data:

12 26 30.75 0.029528 0.036089 0.045932 0.055774 0.072178 0.091864

2) Xxxx.ATH.csv – Alpha time-history

Description:
A spectral time-history of atmospheric absorption coefficients (“alphas”) in units of dB per 1000 feet (“dB1KFT”), dB per 100 meters (“dB100M”), dB per foot (“dBFT”), or dB per meter (“dBM”). There should be one spectral record row of alphas for each spectral record in the acoustic test-day spectral time history data set.

Header:

FileType**
FileName**
FileDateTime**
ProjectName**
MicrophoneID**
LAYERFLAG**
AlphaType**
GeneratedBy**
NumberOfGenerationFiles**
OtherRecords**
NumberOfCommentLines**
All records same as in ALF file

Column Labels:

TODHH TODMM TODSS Alpha17:50Hz Alpha18:63Hz Alpha19:80Hz Alpha20:100Hz Alpha21:125Hz Alpha22:160Hz

… [continues through Alpha40: 10kHz]

Data:

1	12	26	29.75	0.029528	0.036089	0.045932	0.055774	0.072178	0.091864
2	12	26	30.25	0.029528	0.036089	0.045932	0.055774	0.072178	0.091864
3	12	26	30.75	0.029528	0.036089	0.045932	0.055774	0.072178	0.091864
4	12	26	31.25	0.029528	0.036089	0.045932	0.055774	0.072178	0.091864
5	12	26	31.75	0.029528	0.036089	0.045932	0.055774	0.072178	0.091864
6	12	26	32.25	0.029528	0.036089	0.045932	0.055774	0.072178	0.091864
7	12	26	32.75	0.029528	0.036089	0.045932	0.055774	0.072178	0.091864
8	12	26	33.25	0.029528	0.036089	0.045932	0.055774	0.072178	0.091864
9	12	26	33.75	0.029528	0.036089	0.045932	0.055774	0.072178	0.091864
10	12	26	34.25	0.029528	0.036089	0.045932	0.055774	0.072178	0.091864
11	12	26	34.75	0.029528	0.036089	0.045932	0.055774	0.072178	0.091864

3) **Xxxx.EPNL.MTX.csv** – special EPNL Metrics Time-History data

Description:
A time-history of broadband noise levels output by the EPNLCalc module. Each row contains information for a single spectral record in the acoustic test-day spectral time history data set. Metrics to be included are selected using flags in the script. (The user-selected metrics must also be available in the .MTX input file). This file is similar to the xxxx.MTX.csv file, but contains additional columns for identification of maximum levels and ten dB-down points. These columns are labelled “AINT”, “DINT”, “OINT”, “PINT”, and “TINT” for Awt, Dwt, Overall (unweighted), PNL, and PNLT respectively. The “INT” part stands for “Integration labels”. “NVR” stands for “Non-Valid Record” and if a code exists in this column it indicates that the record is not suitable for inclusion in aircraft noise certification EPNL calculation.

Header:

FileType**
FileName**
FileDateTime**
ProjectName**
MicrophoneID**
AveragingMethod**
TimeStampType**
AdjustmentCode**
StartTime**
ReferenceTime**
ReferenceTimeType**
TCLowBand**
NoRndFlag**
TCB40Flag**
HeliTCFlag**
PNLTM + Delta Bandshare**
Delta Bandshare**
PNLTM Recnum**

Bands**
GeneratedBy**
NumberOfGenerationFiles**
GenFileName1**
GenFileDateTime1**
OtherRecords**
NumberOfCommentLines**
Test-Day EPNLCalc

Column Labels:

Rec#	TODhh	TODmm	TODss	RelTime	EFFINT	PNL		PINT	PNLT		TINT	TONECOR	TONEBND	LGB	NVR

Data:

1	12	26	29.25	-0.25	0.5	96.14385		96.14385			0	30	37	AVG	
2	12	26	29.75	0.25	0.5	98.77273		98.77273			0	30	37	AVG	
3	12	26	30.25	0.75	0.5	101.4529		101.564			0.111111	37	38	AVG	
4	12	26	30.75	1.25	0.5	103.8464		103.8464			0	30	38	AVG	
5	12	26	31.25	1.75	0.5	104.4049		104.4938			0.088889	37	38	AVG	
6	12	26	31.75	2.25	0.5	105.6265		105.6265			0	30	39	AVG	
7	12	26	32.25	2.75	0.5	108.248	F10	108.248	F10		0	30	39		
8	12	26	32.75	3.25	0.5	112.2018		112.2018			0	30	40		
9	12	26	33.25	3.75	0.5	113.9053		113.9053			0	30	40		
10	12	26	33.75	4.25	0.5	114.6499		114.6499			0	30	40		
11	12	26	34.25	4.75	0.5	115.059		115.059			0	30	40		
12	12	26	34.75	5.25	0.5	116.6315		116.6315			0	30	40		
13	12	26	35.25	5.75	0.5	118.187	2ND	118.187	2ND		0	30	40		
14	12	26	35.75	6.25	0.5	118.4238	2ND	118.4238	2ND		0	30	40		
15	12	26	36.25	6.75	0.5	119.3976	2ND	119.3976	2ND		0	30	40		
16	12	26	36.75	7.25	0.5	119.6509	2ND	119.7842	2ND		0.133333	38	40		
17	12	26	37.25	7.75	0.5	119.7306	MAX	119.9195	MAX		0.188889	38	40		

18	12	26	37.75	8.25	0.5	119.4625	2ND	119.6291	2ND	0.166667	38	40
19	12	26	38.25	8.75	0.5	118.9601	2ND	119.0601	2ND	0.1	38	40
20	12	26	38.75	9.25	0.5	119.09	2ND	119.1122	2ND	0.022222	38	40
21	12	26	39.25	9.75	0.5	118.4034	2ND	118.4145	2ND	0.011111	38	40
22	12	26	39.75	10.25	0.5	117.9098	2ND	117.9098		0	30	40
23	12	26	40.25	10.75	0.5	117.523		117.523		0	30	39
24	12	26	40.75	11.25	0.5	116.7359		116.7359		0	30	39
25	12	26	41.25	11.75	0.5	116.0746		116.0746		0	30	39
26	12	26	41.75	12.25	0.5	115.7101		115.7101		0	30	39
27	12	26	42.25	12.75	0.5	115.183		115.183		0	30	38
28	12	26	42.75	13.25	0.5	114.3039		114.3039		0	30	38
29	12	26	43.25	13.75	0.5	113.4176		113.4176		0	30	38
30	12	26	43.75	14.25	0.5	112.8192		112.8192		0	30	37
31	12	26	44.25	14.75	0.5	112.6094		112.6094		0	30	38
32	12	26	44.75	15.25	0.5	111.5336		111.5336		0	30	37
33	12	26	45.25	15.75	0.5	110.5836		110.5836		0	30	37
34	12	26	45.75	16.25	0.5	109.5513	L10	109.5513	L10	0	30	37
35	12	26	46.25	16.75	0.5	108.6907		108.6907		0	30	37
36	12	26	46.75	17.25	0.5	107.57		107.57		0	30	36
37	12	26	47.25	17.75	0.5	106.3527		106.3527		0	30	36
38	12	26	47.75	18.25	0.5	104.8743		104.8743		0	30	36
39	12	26	48.25	18.75	0.5	103.8074		103.8074		0	30	36
40	12	26	48.75	19.25	0.5	103.3158		103.3158		0	30	36

4) **Xxxx.EPNL.RPT.csv** – EPNL report data

Description:

Output data related to selected time-integrated metrics, including (if selected) LAE (A-weighted SEL), LDE (D-weighted SEL), unweighted SEL, Special EPNL without tone-correction, and EPNL (based on PNLT). Includes max level, identification of max time and record number, identification of first and last 10 dB-down point times and record numbers, and a code to indicate the quality of the time-integration. Header includes Delta for Bandsharing, as well as individual tone-correction values for the five records centered on the max record. Each row in the file contains data based on a specific metric.

Note: The column marked "TILE" represents the Time-Integrated Level (i.e., LAE for A-weighting, EPNL for PNLT). "TileDur" is the integrated noise duration in seconds. "F10" indicates the first 10 dB-down point, and "L10" the last 10 dB-down point. "10DownCode" identifies whether both 10 dB-down points are valid or whether neither, first-only, or last-only are available. "2ndPeaks" identifies the number of secondary peaks (records with levels within 2 dB of max) exist.

Header:

FileType**
FileName**
FileDateTime**
ProjectName**
GeneratedBy**
NumberOfGenerationFiles**
GenFileName1**
GenFileDateTime1**
OtherRecords**
NumberOfCommentLines**
TCLowBand**
PNLTM**
DeltaBndShr**
ToneCorr(max-2)**
ToneCorr(max-1)**

ToneCorr(max)**
ToneCorr(max+1)**
ToneCorr(max+2)**
AverageTC**

Column Labels:

Metric	Max	MaxRec	MaxTimehh	MaxTimemm	MaxTimess	TILE	TILEDur	F10db	F10Rec	F10Timehh	F10Timemm	F10Timess	L10db	L10Rec	L10Timehh	L10Timemm	L10Timess	10DownCode	2ndPeaks

Data:

PNLT	119.9195277	17	12	26	37.25	118.1003	14	108.248	7	12	26	32.25	109.5513	34	12	26	45.75	BOTH	8
PNL	119.7306388	17	12	26	37.25	118.0566	14	108.248	7	12	26	32.25	109.5513	34	12	26	45.75	BOTH	9

5) **Xxxx.GTH.csv** – Noise Geometry Time-History data

Description:
A time-history of test-day noise geometry, including sound propagation distance, sound emission coordinates, and sound emission angles for acoustic record in the spectral time history.

Note: the values for incidence angle (IncidAng) are set to “66” to indicate that this functionality has not yet been implemented.

Header:

FileType**
FileName**
FileDateTime**
ProjectName**
MicrophoneID**
Microphone(x y z h)**
Microphone Horizontal Angle**
Microphone Vertical Angle**
SoundSpeed (ft/sec)**
PromptTemperature**
AngleUnits**
AlphaUnits**
DistanceUnits**
TemperatureUnits**
GeneratedBy**
NumberOfGenerationFiles**
GenFileName1**
GenFileDateTime1**
GenFileName2**

GenFileDateTime2**
GenFileName3**
GenFileDateTime3**
GenFileName4**
GenFileDateTime4**
OtherRecords**
NumberOfCommentLines**

Column Labels:

Rec# TmTODhh TmTODmm TmTODss TeTODhh TeTODmm TeTODss Tprope Xe Ye Ze SRe THETAe BETAe IncidAng

Data:

1	12	26	29.25	12	26	26.41	2.842757	-3207.6	132.274	333.2784	3227.174	19.7487	5.856255	66
2	12	26	29.75	12	26	27.09	2.661408	-2994.71	118.3722	385.8598	3021.28	21.15684	7.261037	66
3	12	26	30.25	12	26	27.77	2.482638	-2782.63	104.523	438.2421	2818.244	22.76325	8.863587	66
4	12	26	30.75	12	26	28.44	2.306697	-2571.42	90.73156	490.4061	2618.596	24.60904	10.7049	66
5	12	26	31.25	12	26	29.12	2.134732	-2361.46	77.02121	542.2633	2423.257	26.74319	12.83381	66
6	12	26	31.75	12	26	29.78	1.967418	-2152.96	63.40575	593.7615	2233.174	29.22856	15.31296	66
7	12	26	32.25	12	26	30.44	1.805517	-1946.14	49.90074	644.8421	2049.546	32.14361	18.22052	66
8	12	26	32.75	12	26	31.1	1.651322	-1741.73	36.55296	695.3279	1874.276	35.57699	21.64475	66
9	12	26	33.25	12	26	31.74	1.506444	-1540.24	23.39526	745.0947	1709.416	39.63587	25.69223	66
10	12	26	33.75	12	26	32.38	1.372545	-1342.17	10.46159	794.0143	1557.452	44.43887	30.48068	66
11	12	26	34.25	12	26	33	1.252191	-1148.34	-2.19573	841.8885	1421.528	50.09372	36.11643	66

6) **Xxxx.Integ.REF.EPNL.RPT.csv** – Integrated Method Reference EPNL Report data

Description:

Output from the Integrated module, and similar to the xxxx.EPNL.RPT.csv file generated by EPNLCalc, this one contains information about Integrated reference condition EPNL.

Note: The column marked “TILE” represents the Time-Integrated Level (i.e., LAE for A-weighting, EPNL for PNLT). “TileDur” is the integrated noise duration in seconds. “F10” indicates the first 10 dB-down point, and “L10” the last 10 dB-down point. “10DownCode” identifies whether both 10 dB-down points are valid or whether neither, first-only, or last-only are available. “2ndPeaks” identifies the number of secondary peaks (records with levels within 2 dB of max) exist.

Header:

FileType**
FileName**
FileDateTime**
ProjectName**
GeneratedBy**
NumberOfGenerationFiles**
GenFileName1**
GenFileDateTime1**
GenFileName2**
GenFileDateTime2**
GenFileName3**
GenFileDateTime3**
GenFileName4**
GenFileDateTime4**
GenFileName5**
GenFileDateTime5**

GenFileName6**
GenFileDateTime6**
OtherRecords**
NumberOfCommentLines**
TCLowBand**
PNLTM**
DeltaBndShr**
ToneCorr(max-2)**
ToneCorr(max-1)**
ToneCorr(max)**
ToneCorr(max+1)**
ToneCorr(max+2)**
AverageTC**

Column Labels:

Metric	Max	MaxRec	MaxTimehh	MaxTimemm	MaxTimess	TILE	TILEDur	F10db	F10Rec	F10Timehh	F10Timemm	F10Timess	L10db	L10Rec	L10Timehh	L10Timemm	L10Timess	10DownCode	2ndPeaks			

Data:

PNLT	113.1923	20	12	26	41.68	114.1206	23.59	105.085	8	12	26	31.11	103.4029	34	12	26	53.82	BOTH	10			
PNL	112.9545	20	12	26	41.68	113.9366	23.59	104.9517	8	12	26	31.11	103.4029	34	12	26	53.82	BOTH	10			

7) **Xxxx.MAP.csv** – masking / adjustment map

Description:

Conceived as a “layer” under the spectral time-history data, the map has a 2-character code corresponding to each 1/3 octave band SPL in the associated .STH file. The first character in each code represents the masking condition, while the second character indicates the adjustments that have been performed.

Masking codes:
(Appear as first character in two-character map code)

"_"	Not masked
"M"	Masked by Masking Criterion
"A"	Masked by Pre-Detection Noise ("Ambi")
"F"	Masked by Post-Detection Noise ("Floor")
"X"	Masked (usu appears after averaging but used anytime source of masking is unknown)

Adjust Codes:
(Appear as 2nd character in two-character map code)

"_"	None
"C"	Frequency-dependent corrections applied
"N"	Energy-subtraction performed
"A"	Average of adjacent SPLs
"F"	Frequency-extrapolated value
"T"	Time-extrapolated value
"S"	Fixed slope or shaping
"B"	Special Bird/Bug reconstruction
"R"	Reconstructed (other method or unknown)

Examples:

"_N"	Not masked; Energy-subtraction applied
"_C"	Not masked; frequency-dependent corrections applied
"MF"	Masked by masking criterion; reconstructed using frequency-extrapolation
"AA"	Masked by pre-detection; Reconstructed by averaging adjacent SPLs
"FT"	Masked by post-detection; Reconstructed using time-extrapolation
"__"	Not masked; Not adjusted

Header:

FileType**
FileName**
FileDateTime**
ProjectName**
GeneratedBy**
NumberOfGenerationFiles**
GenFileName1**
GenFileDateTime1**
GenFileName2**
GenFileDateTime2**
GenFileName3**
GenFileDateTime3**
GenFileName4**
GenFileDateTime4**
GenFileName5**

GenFileDateTime5**
OtherRecords**
NumberOfCommentLines**

Column Labels:

Rec#	B17/50Hz	B18/63Hz	B19/80Hz	B20/100Hz	B21/125Hz	B22/160Hz

… [continues to B40: 10kHz, then LGB and NVR]

Data:

1	_C	_C	_C	_C	_C	_C
2	_C	_C	_C	_C	_C	_C
3	_C	_C	_C	_C	_C	_C
4	_C	_C	_C	_C	_C	_C
5	_C	_C	_C	_C	_C	_C
6	_C	_C	_C	_C	_C	_C
7	_C	_C	_C	_C	_C	_C
8	_C	_C	_C	_C	_C	_C
9	_C	_C	_C	_C	_C	_C
10	_C	_C	_C	_C	_C	_C
11	_C	_C	_C	_C	_C	_C

8) **Xxxx.MET.csv** – Meteorological data profile

Description:
Temperature and Relative Humidity vs. height for a single meteorological measurement (i.e., "met flight'). Individual times can be associated with each height entry. Temperature units can be selected as Celsius, Fahrenheit, Kelvin, or Rankine. Relative humidity values must be provided as %. Heights can be provided in feet or meters. Heights should have already been normalized to either 100 feet or 30 meters, depending on distance units used. Absorption coefficient units and distance units should match. (i.e., "feet' and "dBFT' or 'dB1KFT') a separate MET file should be provided for prior to and subsequent to each series of aircraft noise events to be processed.

Header:

FileType**
FileName**
FileDateTime**
ProjectName**
DistanceUnits**
TemperatureUnits**

Column Labels:

height	todhh	todmm	todss	temp	rh

Data:

32.8	12	21	44.444	67.71	72.777
100	12	21	57.2	70.7	65.445
200	12	22	5.313	72.222	60.001
300	12	22	8.99	73.333	58.585
400	12	22	13.13	70.7	58.1

9) **Xxxx.MIC.csv** – Microphone coordinate data

Description:

An input file containing information about the microphone and site. Note that the microphone angles are currently place-holders for incidence angle calculations that have not yet been implemented.

Header:

FileType**
FileName**
FileDateTime**
ProjectName**
MicrophoneID**
Units**
GeneratedBy**
NumberOfGenerationFiles**
OtherRecords**
NumberOfCommentLines**

Column Labels:

X Y Z Height MVertAng MHorzAng

Data:

0 0 0 4 0 0

10) **Xxxx.MTX.csv** – Metrics Time-History data – contains one row of broadband metrics for each spectral record in the STH file

Description:

A time-history of broadband noise levels output by the Metrix module. Each row contains information for a single spectral record in the acoustic test-day spectral time history data set. Metrics to be included are selected using flags in the script. This file is similar to the xxxx.EPNL.MTX.csv file generated by EPNLCalc, but does not contain columns for identification of maximum levels and ten dB-down points. The column labelled “LGB” contains the ANSI/ISO band number for the Last Good Band used in the background noise adjustment methodology.

Header:

FileType**
FileName**
FileDateTime**
ProjectName**
MicrophoneID**
AveragingMethod**
TimeStampType**
AdjustmentCode**
StartTime**
ReferenceTime**
ReferenceTimeType**
TCLowBand**
NoRndFlag**
TCB40Flag**
HeliTCFlag**
Bands**
GeneratedBy**
NumberOfGenerationFiles**

GenFileName1**
GenFileDateTime1**
GenFileName2**
GenFileDateTime2**
OtherRecords**
NumberOfCommentLines**

Column Labels:

Rec# TODhh TODmm TODss RelTime PNL PNLT TONECOR TONEBND LGB NVR

Data:

1	12	26	29.25	-0.25	96.14385	96.14385	0	30	37	AVG
2	12	26	29.75	0.25	98.77273	98.77273	0	30	37	AVG
3	12	26	30.25	0.75	101.4529	101.564	0.111111	37	38	AVG
4	12	26	30.75	1.25	103.8464	103.8464	0	30	38	AVG
5	12	26	31.25	1.75	104.4049	104.4938	0.088889	37	38	AVG
6	12	26	31.75	2.25	105.6265	105.6265	0	30	39	AVG
7	12	26	32.25	2.75	108.248	108.248	0	30	39	
8	12	26	32.75	3.25	112.2018	112.2018	0	30	40	
9	12	26	33.25	3.75	113.9053	113.9053	0	30	40	
10	12	26	33.75	4.25	114.6499	114.6499	0	30	40	
11	12	26	34.25	4.75	115.059	115.059	0	30	40	
12	12	26	34.75	5.25	116.6315	116.6315	0	30	40	
13	12	26	35.25	5.75	118.187	118.187	0	30	40	
14	12	26	35.75	6.25	118.4238	118.4238	0	30	40	

11) **Xxxx.PTH.csv** – Position Time-History data

Description:

This file contains a time-history of measured aircraft position in the local coordinate TXYZ system. There is an assumption that the centerline microphone is located at the origin (0,0,0,), but the equations for noise geometry calculations should be able to accommodate other situations. The time of each position measurement, Tp, should be provided in the common timebase used for acoustical measurements and meteorological measurements. The data sets for this file can be obtained directly form TSPI system outputs or can be generated for an average straight-line flight path by inputting single-point tracking data elements (As might be obtained from photo-positioning or other methods) to the SPoinTrkIn module.

Note that the PTH records and their timestamps are independent of spectral records in the spectral time-history.

Header:

FileType**
FileName**
FileDateTime**
ProjectName**
Units**
GeneratedBy**
NumberOfGenerationFiles**
OtherRecords**
NumberOfCommentLines**

Column Labels:

TODHH	TODMM	TODSS	X	Y	Z

Data:

12	26	20	-5209.59	263.0034	-161.184
12	26	20.5	-5053.36	252.8017	-122.598
12	26	21	-4897.13	242.6	-84.0118
12	26	21.5	-4740.91	232.3983	-45.4256

12	26	22	-4584.68	222.1966	-6.83938
12	26	22.5	-4428.45	211.995	31.74685
12	26	23	-4272.22	201.7933	70.33308

12) **Xxxx.REF.GTH.csv** – Reference Condition Noise Geometry Time-History data

Description:
A time-history of test-day and reference condition noise geometry, including test and reference sound propagation distances, sound emission coordinates, and sound emission angles for each acoustic record in the spectral time history. Tm is the acoustic measurement time (or timestamp), while Te is the calculated time of emission of the sound that was measured at Tm. ("Retarded" time, which accounts for the time it takes for sound propagation between the aircraft and the microphone). Xe, Ye, Ze are the emission coordinates of the aircraft at Te. SR is the Slant Range, or straight line sound propagation distance between the aircraft and the microphone. THETA is the three-dimensional sound emission angle (angle between the aircraft direction of flight and the straight line sound propagation path between the aircraft and the microphone). BETA is the two-dimensional sound elevation angle (vertical angle above the microphone height to the aircraft height at Te.) Tr is the reference measurement time, based on the reference condition flight path and the reference condition soundspeed. EFFINT is the effective interval in seconds for a particular record, to be used in the calculation of Integrated Reference EPNL.

Header:

FileType**
FileName**
FileDateTime**
ProjectName**
DistanceUnits**
AngleUnits**
SpeedUnits**
Test TOH (Time at OverHead)**
Reference TOH (Time at OverHead)**
Test ZOH (Height above ground at OverHead)**
Ref. ZOH (Ref. Height above ground at OverHead)**
Average Test Groundspeed**

Reference Groundspeed**
Average test climb/descent angle**
Ref. climb/descent angle**
Test soundspeed**
Ref. soundspeed**
Test microphone X Y Z coordinates**
Test microphone height above local ground**
Ref. microphone Y coordinate**
Ref. microphone height above local ground**
Test flight path CPA (Closest Point of Approach)**
Ref. flight path CPAR (Closest Point of Approach)**
Ref flight path CPAOHR (Closest Point of Approach for centerline location)**
GeneratedBy**
NumberOfGenerationFiles**

Column Labels:

Rec# Tmhh Tmmm Tmss Tehh Temm Tess Xe Ye Ze SR THETA TeRhh TeRmm TeRss XeR YeR ZeR SRR TpropR TRhh TRmm TRss EFFINT

Data:

1 12 26 29.25 12 26 26.41 -3207.604931 132.2740366 333.2783791 3227.173697 19.74869775 12 26 20.17 -5013.773554 0 1804 5327.093509 4.691413 12 26 24.86 0.8943855
2 12 26 29.75 12 26 27.09 -2994.712829 118.3722198 385.8597552 3021.280156 21.1568388 12 26 21.36 -4651.066003 0 1804 4987.225176 4.392101 12 26 25.76 0.8943855
3 12 26 30.25 12 26 27.77 -2782.626527 104.5230216 438.24211 2818.244429 22.76324766 12 26 22.55 -4289.731305 0 1804 4652.074233 4.096944 12 26 26.65 0.8941864
4 12 26 30.75 12 26 28.44 -2571.424378 90.73155859 490.406091 2618.596271 24.60903847 12 26 23.74 -3929.902954 0 1804 4322.515151 3.806711 12 26 27.54 0.8938184
5 12 26 31.25 12 26 29.12 -2361.464428 77.02121089 542.2632663 2423.25687 26.74318933 12 26 24.91 -3572.190957 0 1804 4000.068529 3.522742 12 26 28.44 0.8932134

13) **Xxxx.Simp.REF.EPNL.RPT.csv** – Simplified Method Reference EPNL Report data

Description:

Output by the Simplified module, this file contains information about the simplified reference condition EPNL, including calculated values for all Deltas, as well as the reference-condition 1/3 octave band spectrum for PNLTM, adjusted from test-day. Note that Del Bandshare is not reported separately in this file, as it is included in DEL1. (Del Bandshare can be obtained from the associated xxxx.TD.EPNL.RPT.csv or xxxx.TD.EPNL.MTX.csv file.)

Header:

FileType**
FileName**
FileDateTime**
ProjectName**
EPNLRSimp**
PNLTMRSimp**
DEL1**
DELPeak**
DEL2**
DEL2D**
DEL2S**
DEL3**
Max2Peak**
Max2PeakK**
GeneratedBy**
NumberOfGenerationFiles**
GenFileName1**
GenFileDateTime1**
GenFileName2**
GenFileDateTime2**
GenFileName3**

GenFileDateTime3**
GenFileName4**
GenFileDateTime4**
GenFileName5**
GenFileDateTime5**
GenFileName6**
GenFileDateTime6**
GenFileName7**
GenFileDateTime7**
OtherRecords**
NumberOfCommentLines**
Test-Day Delta Bandshare is included in the Simplified PNLMTR and EPNLR

Column Labels:

B17/50Hz B18/63Hz B19/80Hz B20/100Hz B21/125Hz B22/160Hz

…[continues to B40/10kHz]

Data:

83.88764 81.91851 80.01769 91.69731 96.88302 97.96787

14) **Xxxx.SPO.csv** – Single-Point Tracking data report

Description:

This file contains single-point tracking data elements as might be obtained from aircraft position data obtained in the field using photo-positioning techniques or other methods. The information can also be derived from either a position time history (.PTH file), or a noise geometry time-history (.GTH file) using the SPoinTrkOut module. The information required has been specified in the ICAO ETM, and is considered sufficient to define an average, straight-line flight path approximation, suitable for processing aircraft noise data for certification purposes. All information is contained in the header of this file only. There is no "body" of data.

Header:

FileType**
FileName**
FileDateTime**
ProjectName**
GeneratedBy**
NumberOfGenerationFiles**
GenFileName1**
GenFileDateTime1**
GenFileName2**
GenFileDateTime2**
OtherRecords**
NumberOfCommentLines**
TOH (hour minute seconds)
XOH (feet)
YOH (feet)
ZOH (feet)
Mic X (feet)
Mic Y (feet)
Mic Z (feet)
Mic Height (feet)

Single-Point CPA (feet)
Time-History CPA (feet)
AvStartTime
AvEndTIme
Average Climb/Descent Angle (degrees)
Average Lateral Cross-Angle (degrees)
Average Ground Speed (feet/second)
Average Vector Speed (feet/second)

Column Labels:

n/a

Data:

n/a

15) **Xxxx.SSR.csv** – Single spectrum record of 1/OB SPLs

Description:

A single spectrum of 1/3 octave band values. These may represent corrections to be applied to SPLs in dB, actual average SPLs for background noise data sets, or other cases where a single spectrum is needed. The filename and comments should provide sufficient info to identify the data.

Note: .ALF files have basically the same format, but represent atmospheric absorption coefficients in dB/distance, and may have additional columns at the start of the data row.

Header:

FileType**
FileName**
FileDateTime**
ProjectName**
GeneratedBy**
NumberOfGenerationFiles**
OtherRecords**
NumberOfCommentLines**

Column Labels:

B17/50Hz B18/63Hz B19/80Hz B20/100Hz B21/125Hz B22/160Hz

…[continues to B40/10kHz]

Data:

0.18 0.03 0.08 0.08 0.12 0.08

16) **Xxxx.STAT.csv** – Statistical data results report

Description:

Output report from the SimpleStats module. Provides values for Average level, Standard Deviation and 90% Confidence Interval for a set of data, and identifies whether the data set was clustered or regressed and if regressed, the order of regression. Also reports Degrees of Freedom (DOF). Data rows (one for each event) include noise level, delta and delta-squared.

Header:

FileType**
FileName**
FileDateTime**
ProjectName**
GeneratedBy**
NumberOfGenerationFiles**
GenFileName1**
GenFileDateTime1**
ScriptFile**
ScriptLine**
OtherRecords**
NumberOfCommentLines**
example input data file for Simplestats module
Average**
StdDev**
90% Confidence Interval**
Degrees of Freedom**
Student's T**
Sum of Deltas Squared**

Data Set Type**

Column Labels:

ID	Value	Delta	DS
Data:			
1	92.4	-1.18333	1.400278
2	91.1	-2.48333	6.166944
3	93.3	-0.28333	0.080278
4	95.2	1.616667	2.613611
5	94.4	0.816667	0.666944
6	95.1	1.516667	2.300278

17) **Xxxx.STH.csv** – 1/3 octave band spectral time-history

Description:

Spectral Time History data – a series of 1/3 octave band spectra vs. time. Used for inputs and outputs from many modules and processes – the basic data set processed by SuperFAR. Header includes info about generation and project and in some cases, time-averaging methodology. Data rows include a RecNum, which is the Record Number of the spectrum – used for internal synchronization of data sets, closely linked with index "k" in noise certification specifications and guidance material – followed by a timestamp, based on one of several methodologies, identified in the header, an optional relative time in seconds, and (currently defaulting to 24) one-third octave band SPLs representing a noise spectrum output from an analyzer, ar some stage in processing.

Header:

FileType**
FileName**
FileDateTime**
ProjectName**
MicrophoneID**
AveragingMethod**
TimeStampType**
AdjustmentCode**
StartTime**
ReferenceTime**
ReferenceTimeType**
GeneratedBy**
NumberOfGenerationFiles**
GenFileName1**
GenFileDateTime1**
GenFileName2**
GenFileDateTime2**
GenFileName3**
GenFileDateTime3**

GenFileName4**
GenFileDateTime4**
OtherRecords**
NumberOfCommentLines**

Column Labels:

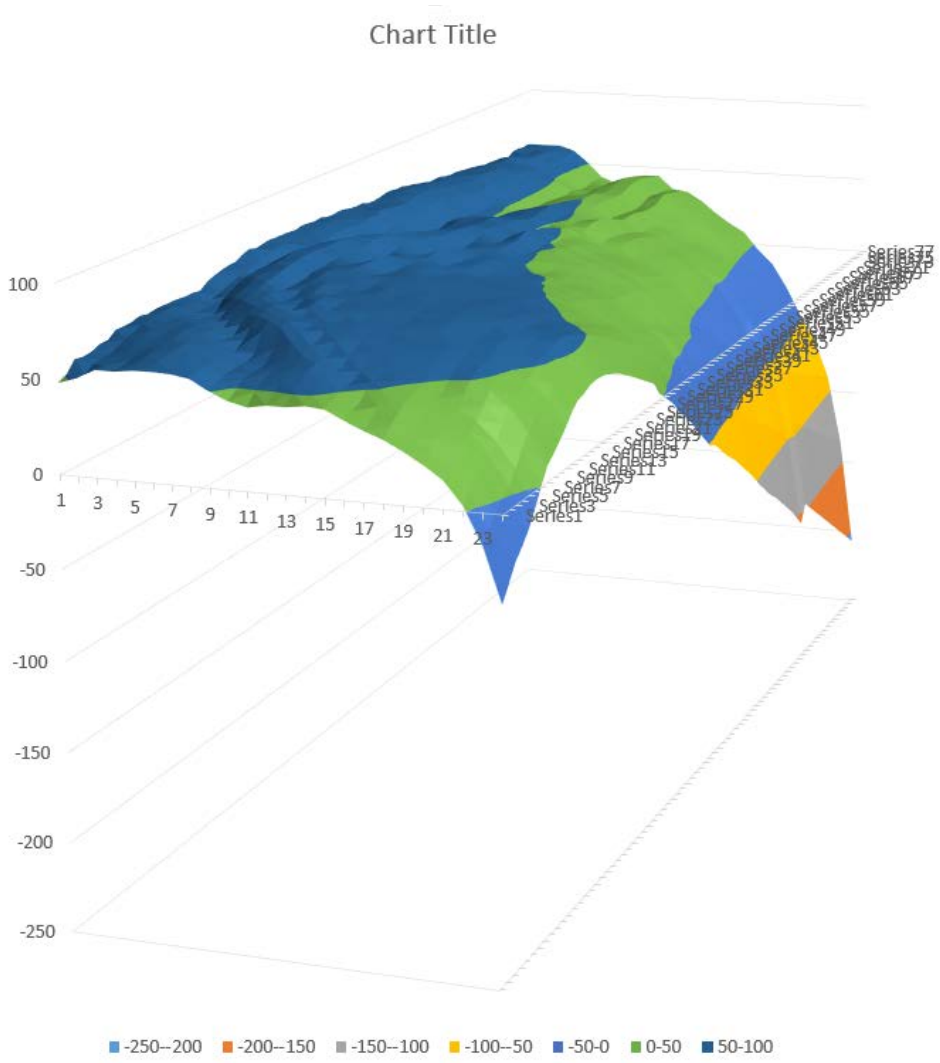
Rec# TODhh TODmm TODss RelTime B17/50Hz B18/63Hz B19/80Hz B20/100Hz B21/125Hz B22/160Hz

Data:

1	12	26	29.75	21.25	77.98	81.3	85.84	85.75	87.11	87.37
2	12	26	30.25	21.75	77.65	76.69	82.3	83.03	87.41	88.68
3	12	26	30.75	22.25	84.9	82.39	80.35	85.25	89.31	89.68
4	12	26	31.25	22.75	79.71	81.52	84.23	85.96	88.01	89.18
5	12	26	31.75	23.25	82.12	80.97	85.64	87.46	88.51	90.48
6	12	26	32.25	23.75	85.4	85.36	87.76	88.97	90.72	88.58
7	12	26	32.75	24.25	83.87	87.63	89.36	89.27	92.02	90.48

APPENDIX 3 – Data Analysis and Visualization

Output data files can be examined using text-editors or spreadsheet programs. It can be extremely useful when evaluating differing methodologies to generate a three-dimensional surface plot – similar to a waterfall plot - of spectral time-history data for an aircraft noise event. Such plots are available within Microsoft Excel. The simplest way to do so is to double-click on an STH file, which will open it in Excel (if Excel is set as the default software for the .csv file type within the operating system environment). Then the data values for the columns labelled with center frequencies and band numbers can be selected, followed by clicking on “Insert” in the main menu bar, then from the “Charts” section, select “Other Chart”, then the leftmost option under “Surface Charts”, which may have a tooltip popup saying “3D Surface Chart”. Once this is selected, a small chart appears in the middle of the selected data area. Right-click on the chart, then select “Move chart” from the pop-up menu. Select “New sheet:” and enter a new name or use the default, then click “OK”. With the new chart sheet displayed, right-click in the chart area again, and select “Select Data…”. In the pop-up that appears, click on the button labelled “Switch Row/Column”, then click “OK”. The chart should now have an appearance similar to this:



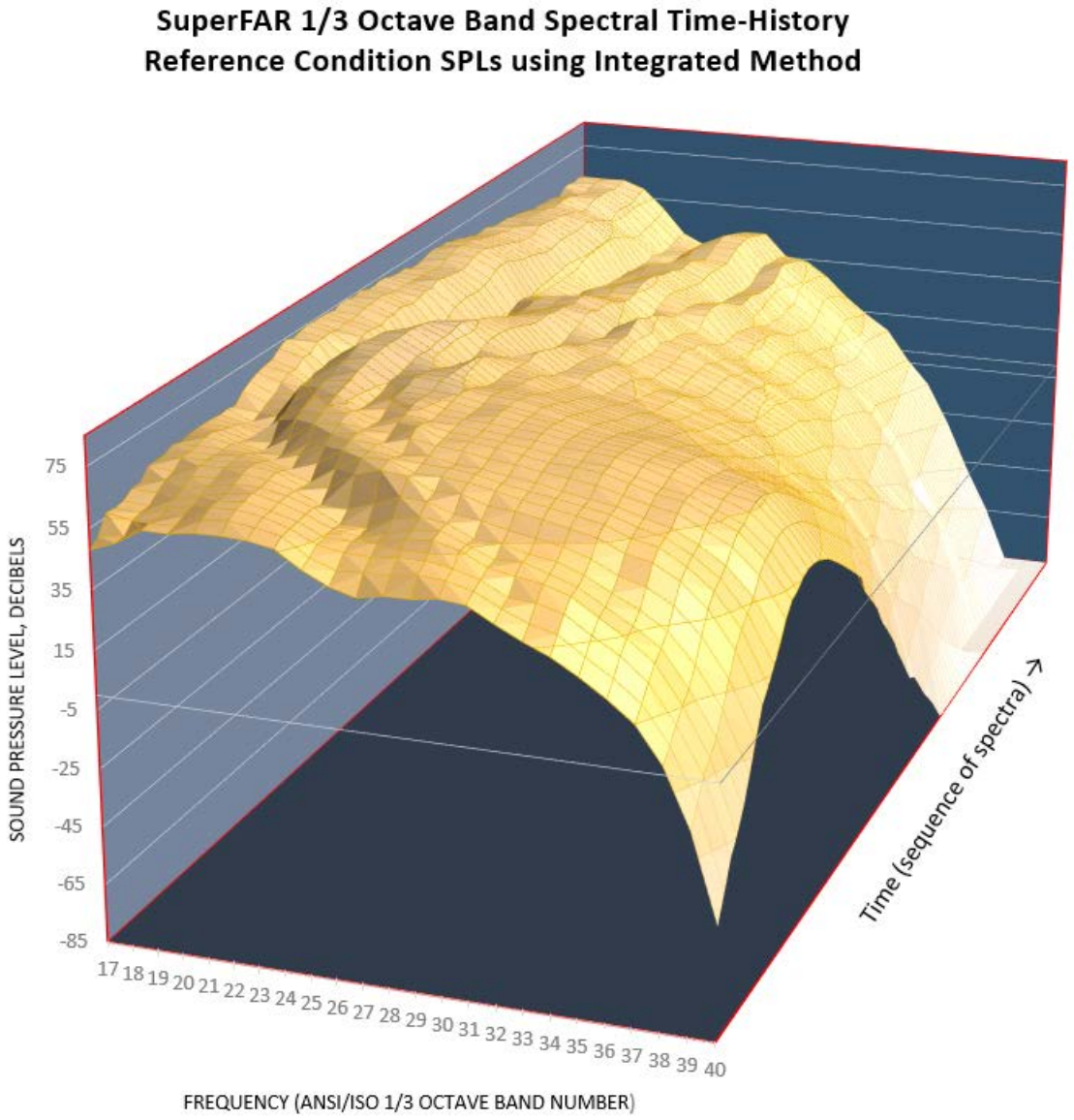
In this mode, Excel defines the axes as follows:

X: Horizontal – frequency (1/3 octave bands)

Y: Vertical – amplitude (dBSPL)

Z: Depth – time (spectrum record numbers)

There are many useful options for improving the displayed plot, including 3D rotations, scaling, and labelling info. These are beyond the scope of this User Guide, but the following is the same chart, modified for better presentation:



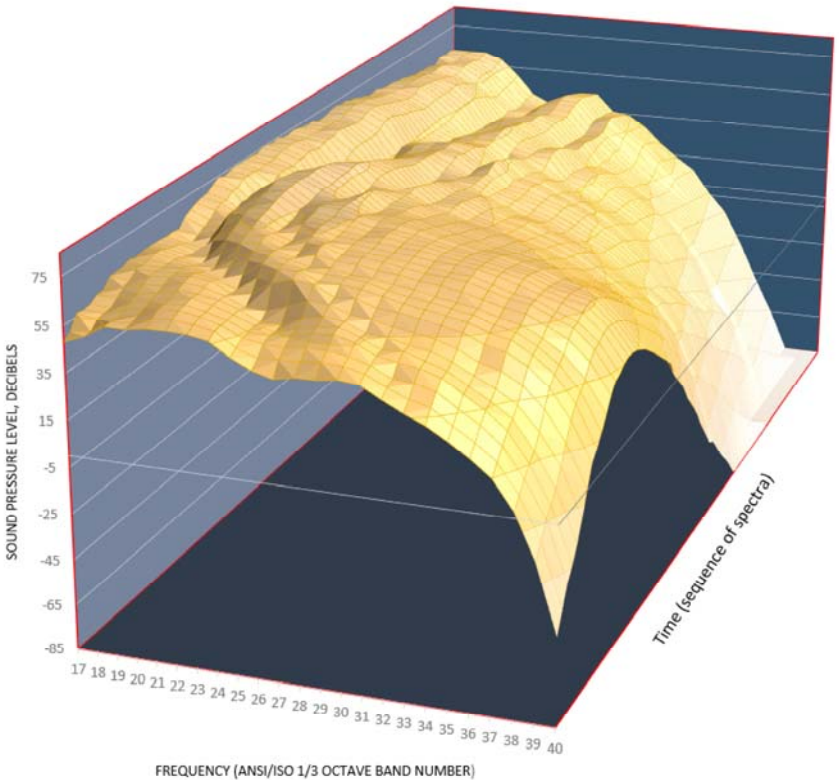
This version clearly illustrates the presence of low-frequency pseudotones caused by constructive and destructive interference between the direct and reflected sound paths from the aircraft to the microphone. Doppler-shifted tones exhibit a constant downward shift in frequency over time, while pseudotones exhibit a characteristic downward – then upward shift in frequency over time. This graphic also illustrates the effects of reconstructing masked high-frequency SPLs: in some cases, the reconstructed SPLs are extrapolated to levels below zero dB SPL.

—

U.S. Department of Transportation Volpe Center

29SEP2017

USDOT Volpe Center Acoustics
SuperFAR
(Spectral Data Processing Suite)
Developer's Manual



Dave Read & Chris Cutler, Environmental Measurement and Modeling Division, V-324

Eugene O'Neil (Contractor - Safety Management Systems)

SuperFAR 6.0 : 29 September 2017

Foreword

SuperFAR is a suite of Software modules developed by the USDOT Volpe Center Acoustics Facility (part of the Environmental Measurement & Modeling Division – V324 – of the Center for Policy, Planning and Environment – V320).

The software, funded by FAA's Office of Environment & Energy, Noise Division (AEE-100), has been developed under the Aircraft Noise Certification Support Interagency Agreements between AEE and Volpe, and is intended primarily for use in the Validation of Certification Applicants' Software and Methodology, as required by FAA Order 8110.4C.

SuperFAR is implemented in the Python programming language, and its operation is entirely script-based. This allows for readability of code and for simple text-editor coding and modification.

SuperFAR includes FAA-approved methodologies for all aspects of computing reference-condition EPNL noise levels from measured 1/3 octave band spectral time-histories of aircraft noise data. (Measured aircraft position and meteorological data are also required as inputs.) For each element of the process, SuperFAR includes at least one approved method, and in many cases, includes several alternate methods, which may or may not be considered "equivalent" under the specifications and requirements of Combined Federal Regulations, Title 14, part 36. Additionally, due to the flexibility of the scripting approach used, alternate sequences of operations may be implemented that may or may not meet the part 36 requirements. A preferred process script and example data set is provided with the distribution package. This preferred methodology is entirely in compliance with the part 36 requirements, if applied to appropriately-measured and collected input data sets.

This Developer's Manual is being provided for the first time with Version 6.0 of the SuperFAR software, which is the third version externally released to FAA. A Users' Guide, provided initially with Version 5.0 of the software, has been updated to Version 2 for this release. The Users' Guide provides practical information about the algorithms and processes available within SuperFAR, as well as guidance on development of scripts and data sets. This Developer's Manual provides detailed information on the software elements themselves, including specifics on each module, data object and structure. Additionally, this manual provides instruction on developing new modules and data structures. The bulk of this documentation is generated automatically with each new "build" of SuperFAR as an html document.

The Manual is divided up into six sections: Section 1 addresses the elements and structure of a SuperFAR process script. Section 2 provides specifics on SuperFAR Data Types. Section 3 is a listing of each SUperFAR module (or Application), and its Header Comments, Unit Parameters, Calls to other modules, Input Parameters, Output Parameters, and default settings/values, where applicable. Section 4 is a guide to writing SuperFAR files. Section 5 is a guide to reading SuperFAR files. Section 6 contains specifics on SuperFAR data file formats. In addition, there are three appendices: Appendix A1 provides information about SuperFAR's HTML Documentation Template Engine. Appendix A2 provides instructions for developing new SuperFAR applications, and Appendix A3 provides instructions for developing new SuperFAR data file formats.

Section 1: Basic Structure of a SuperFAR script

SuperFAR python scripts typically begin by importing the superfar library in the following manner:

```
from superfar import *
```

this instructs python to import all named symbols from the superfar package to be used in the script without qualification. An experienced python programmer is likely aware of other methods of importing a library, but the rest of this documentation assumes that superfar is imported in the above manner

The next step of a typical SuperFAR script involves setting the project name that the script is associated with, like so:

```
superfar.ProjectName = 'My Example Project'
```

other script level-settings can also be set in a similar manner. For example, to turn off verbose debugging output that superfar normally generates by default:

```
superfar.verbose = False
```

and to change the default values of one or more unit types (see Section 2.5: Measurable Quantities and Unit Parameters):

```
superfar.DistanceUnits = 'meters'
superfar.AngleUnits = 'radians'
```

the remainder of a typical SuperFAR script usually consists of a sequence of commands invoking SuperFAR applications, often in reference to SuperFAR data files. see Section 3: SuperFAR Applications and Section 6: SuperFAR File Formats.

Section 2: SuperFAR Data Types

This section describes the various data types implemented by the SuperFAR library.

Section 2.1.1: superfar, the Configuration Object

the superfar package contains a single BaseContainer object named 'superfar' that serves as a unified storage location for all configuration values that apply to an entire project, as opposed to individual data files or applications.

Unit Parameters:

Name	Type	Default
AlphaUnits	Alpha Unit	"dB100m"
AngleUnits	Angle Unit	"Degrees"
DistanceUnits	Distance Unit	"Feet"
FrequencyUnits	Frequency Unit	"Hz"
SpeedUnits	Speed Unit	"FPS"
TemperatureUnits	Temperature Unit	"Celsius"
HumidityUnits	Humidity Unit	"Humidity"
SPLUnits	SPL Unit	"dB"
TimeUnits	Time Unit	"HHMMSS"

Section 2.1: BaseContainer Objects

Many objects in superfar that contain data, such as Application Objects and TableData Objects are called container objects, and inherit many useful features by being decendants of a common base class named BaseContainer.

Every BaseContainer object contains a well-defined set of named items called parameters. For example, Application Objects have input and output parameters, and TableData Objects have parameters that correspond to file header annotations and column names.

the specific parameters provided by a given BaseContainer class should be described by a ContainerSchema object assigned to a variable named 'interface' in the definition of the class.

the parameters of a SuperFAR container object can be referenced using standard python array-index syntax:

```
container['parameter_name'] = value
```

by standard python object-attribute syntax:

```
container.parameter_name = value
```

and finally, multiple named parameeters of a container can be set using named-parameter function syntax:

```
container (
    parameter_name_1 = value1,
    parameter_name_2 = value2,
    parameter_name_3 = value3,
)
```

Parameter names are not case sensitive: container["Parameter_Name"] and container.PARAMETER_NAME both reference the same parameter. Some parameters have alternate names, called aliases, which are usually a synonym or abbreviation of the standard name. Aliases are also not case sensitive.

All container parameters have a data type that parameter values are expected to conform to. When a parameter is assigned a value not of the expected type, the value is converted to the expected type, usually by invoking the expected type in a functional manner on the value to be converted. This process is known as parameter coercion. For example, if a parameter with type str is assigned the numeric value of 5, it is coerced to the value str(5), which evaluates to '5'. Custom data types introduced by the SuperFAR library are usually designed with object constructor methods that provide sensible and intuitive results when coercing a value.

Section 2.1.2: Application Objects

SuperFAR Applications are python classes that descend from the Application class, which is in turn a subclass of the BaseContainer class.

Each application represents a computation involving input and output values, represented by input and output parameters of the application container object. The computation is performed when the application class is invoked in a function-like manner, usually with values provided for all applicable input parameters. The result returned by this invocation is a container object that contains all of the original input values as well as the resulting output values, all of which may be referenced later for subsequent computations.

for a further description of how Applications are used in practice, and a reference manual of individual SuperFAR applications, see Section 3: SuperFAR Applications

for instructions on how to write a new SuperFAR application, see Appendix A2: Adding New Applications to SuperFAR

Section 2.1.3: TableData Objects

SuperFAR file objects are python classes that descend from the TableData class, which is in turn a sublass of the BaseContainer class.

Some subclasses of TableData represent a single row or record of data, containing data values indexed by column name and/or band number. Named columns are implemented as a subtype of BaseContainer parameter, and thus are case insensitive, can be indexed by array-style or or object.attribute syntax, and so forth.

Other subclasses of TableData represent multiple rows of data, usually indexed by a record number. As record numbers are not strings, they may only be referenced by array-style subscripting. Each record stored in a multi-record TableData object is another TableData object containing a single row of table data, in the manner described in the previous paragraph. Since the most likely use of indexing a row of data is to then select a column of data, multi-row TableData object also allow indexing by row and column in a single operation:

```
multirow_data[row, col] == multirow_data[row][col]
```

There is a separate python class for each specific file format, but most file formats have a very similar layout: the first few lines of the file contain file annotations, followed by one row of column labels that serves as a column header for the final section, which is one or more rows of data.

When an input or output parameter of a SuperFAR application is a SuperFAR data file, the user is allowed to set that parameter to a string, which will be treated as a path to a data file.

for a description of individual SuperFAR file formats, see Section 6: SuperFAR File Formats

for instructions on how to write a new SuperFAR application, see Appendix A3: Adding New File Formats to SuperFAR

Section 2.2: ContainerSchema Objects

a ContainerSchema object is used to declare the interface of a BaseContainer object. A ContainerSchema is initialized with a list of Parameter objects and ContainerSchema objects to describe a set of attributes and properties for a BaseContainer class.

Section 2.2.1: The "stdapp" Object

stdapp is a ContainerSchema object that should be included as an element in the ContainerSchema interface of all Application classes, to define the following standard attributes common to all SuperFAR applications:

Object Properties:

Name	Type	Default	Description
recursion_level	Integer	None	counts the number of superfar applications invoking other superfar applications leading up to this application being invoked.

Virtual Parameters:

Name	Type	Description
GenerationFiles	list	generates list of input parameter values that are Data objects in this application invocation. Used to generate the default value of the GenerationFiles annotation of TableData objects.
GeneratedBy	list	returns a list containing the name of the application and the current version of superfar.

Section 2.2.2: The "stdfile" Object

stdfile is a ContainerSchema object that should be included as an element in the ContainerSchema interface of all TableData classes, to define the following standard attributes common to all TableData formats:

Object Properties:

Name	Type	Default	Description
path	String	None	full file system path to external csv file
header	ColumnHeader	* dynamic	A tuple of column keys that determines the presence and order of column values as they are written to a file in csv format. If the TableData class is row-oriented, the valid column keys that may be included in the header tuple are the names of the column parameters in the interface, plus any index key type accepted by the IndexedParameter of the interface, if present. If the TableData class is multi-row, which is to say it has an IndexedParameter with a TableData type, that TableData type is presumed to be row-oriented, and it is used to determine the valid column keys of the multi-row TableData class. When a csv output file is being written, the header is written to the csv file after the annotations, and then each row of data is written to the csv file as a comma separated sequence of column values corresponding to the column keys in the header. Conversely, when a csv input file is being read, the header is read from the file after the annotations, and then each subsequent line in the csv file is presumed to be the corresponding column values of a row of data.
disk_pending	Boolean	True	true if disk activity has not yet occurred, which is to say the contents of the TableData container has not yet been read from or written to a file.
app	BaseContainer	* dynamic	the application this file was used as a parameter for.
line_number	Integer	None	

Virtual Parameters:

Name	Type	Description
FileName	String	file name component of full file system path to file
FileDateTime	tuple	returns current date and time, in format used while writing file to disk.
FileNameDateTime	tuple	returns file name, date, and time, in format used while writing file to disk
ProjectName	String	a file Annotation that equals superfar.ProjectName
GeneratedBy	list	a file Annotation that equals app.GeneratedBy (if app exists)
GenerationFiles	list	a file Annotation that equals app.GenerationFiles (if app exists)

File Annotations:

Name	Type	Default	Description
ProjectName	String	None	a file Annotation that equals superfar.ProjectName
GeneratedBy	list	None	a file Annotation that equals app.GeneratedBy (if app exists)
GenerationFiles	list	None	a file Annotation that equals app.GenerationFiles (if app exists)
ScriptFile	String	None	a file Annotation that equals app.ScriptFile (if app exists)
ScriptLine	Integer	None	a file Annotation that equals app.ScriptLine (if app exists)
OtherRecords	list	None	an arbitrary list of user data that may be associated with a file
Comments	String	"Comments go here."	free-form comments

Section 2.3: Parameter Objects

objects of the Parameter class are used in a ContainerSchema object to define the name, type, and behavior of values stored in a container. Each parameter is declared with a set of named options with values that control the behavior of the parameter:

Parameter Objects Table

Option	Description
name	The name of a Parameter object is a string identifier used to access the parameter within the container. This can always be done by using python array-indexing notation on the container with the name of the parameter as a key. Optionally, if the name of a parameter would be a syntactically valid python object attribute, object attribute dot-notation can also be used.
suffix	A decomposed parameter may have a suffix instead of a name, specifying that the full name of the decomposed parameter will be derived from the original parameter's name, by appending the given suffix. for more context, see the description for the decomposition option.
index	A parameter may have an index type instead of a name, to control how the container manages key values that are not recognised as a standard parameter name. The value of the index parameter should be a python class, that unrecognized key values should be cast to before the container stores or retrieves a value using the standard python dictionary methods. The two most common index types are RowNum for TableData containers that contain multiple rows of data, and Band for TableData containers that have multiple columns identified by band number. Only one index parameter is allowed per container.
alias_of	The alias_of option specifies that this parameter is an alias, which is to say the name of this parameter is merely an alternate name for another "actual" parameter, that the alias_of option identifies by name. An alias parameter must have a name, and cannot have any other options aside from the alias_of option.
role	The role of a parameter describes the purpose of the parameter within the container, which in some cases influences the behavior of the container. For example, input and output parameters of an Application container have a role values of 'input' or 'output'.
type	The type of a parameter is a python class or SuperFAR unit describing the type of values the parameter is intended to store. Values assigned to a parameter or retrieved from the parameter will be coerced to the specified type. In most cases coercion simply involves invoking the type as a constructor on the value to be coerced: for example, int("3") coerces the string "3" to an integer 3. SuperFAR has special rules for coercing Boolean values: strings such as "T" and "yes" are coerced to True, and "F" and "no" are coerced to False, rather than simply calling bool(str). When the type option of a parameter is set to an abstract python unit such as "Distance", it stores values in an actual unit of the abstract type (ie, "Feet") as specified by a unit parameter. (see the next entry, 'unit_type') If a parameter has a constant value, (see the value option below) the type of the Parameter defaults to the type of the constant value.
unit_type	Unit parameters have a unit_type option instead of a standard type option, which should be an abstract unit such as "Distance", specifying that this parameter stores a specific unit of the abstract type (ie, "Feet"). The name of a unit parameter should always be the name of the abstract with "Units" appended (for example, "DistanceUnits").
value	Parameters that have a value option are read-only parameters. if the value option is set to a python function, the value of the parameter is dynamically computed every time it is accessed by invoking the function with the container as an argument, and additionally the index key value if the parameter is an index parameter. This is called a "dynamic value parameter". If the value option is not set to a python function, the value of the parameter is

Option	Description
	simply the value specified by the value argument. This is called a "constant value parameter". If a constant value parameter is not assigned an explicit type argument, the type of the parameter is inferred from the constant value. Parameters that have a dynamic value must always be given an explicit type.
default	The default option of the Parameter class behaves almost exactly like the value option, including the ability to compute dynamic values if given a python function. The crucial difference is that the default option does not prevent the user from assigning a non-default value to the parameter, after which the default value becomes irrelevant. Assigning the python value None to a parameter will re-enable default value behavior, if present.
minval	If a parameter has a minval argument, it will print a warning if it is ever assigned a value below the minval. Obviously the type of the minval argument must be compatible with the value type of the parameter for the purpose of comparison.
maxval	Like minval, a parameter with a maxval argument will print a warning if it is ever assigned a value below it's maxval. Similarly, the type of the maxval argument must be compatible with the value type of the parameter for the purpose of comparison.
keywords	If a parameter of type str is provided with a keywords argument containing a list or tuple of strings, it will print a warning message if it is ever assigned a string value that does not appear in the list of valid keywords: the valid keywords are reiterated in the warning message, for the convenience of the user.
comment	A string containing html formatted text, to document the purpose of the argument in dynamically generated HTML documentation. If the parameter has a keywords option, the comment value can be set to a python dictionary, mapping the keywords as keys of the dictionary to a specific html formatted comment string for each keyword.

decomposition	A decomposed parameter is a parameter that is not actually stored in the container as a single value, but instead is broken up into sub-parameters that each store a specific part of the original value, which can be dynamically re-composed as needed. This behavior is enabled by the existence of decomposition argument, which contains a tuple of Parameter objects that the decomposed parameter decomposes into.
	
	The decomposition option is most commonly set implicitly, when SuperFAR detects that the type of the parameter has an attribute named 'decomposition'. However, it is also possible to set the argument explicitly, either to override some aspect of the default decomposition associated with the class, or to assign a decomposition to a class that doesn't normally have one.
	
	To give a common, specific example of this abstract feature, consider a parameter named 'TOD' of type HHMMSS, which has the following default decomposition attribute attached to the class:
	
	`HHMMSS.decomposition = (`
	`  Parameter(suffix="hh", type=Hours),`
	`  Parameter(suffix="mm", type=Minutes),`
	`  Parameter(suffix="ss", type=Seconds, format='.2f'),`
	`)`
	
	Because the Parameter objects in this decomposition have suffixes instead of names, the final names of the sub-parameters are based on the original decomposed parameter, plus the given suffix. Thus in the case of 'TOD', the sub-parameters will be 'TODhh', 'TODmm', and 'TODss'.
	
	Any assignment to 'TOD' is intercepted by the container and translated into the appropriate sub-assignments to 'TODhh', 'TODmm', and 'TODss'. The sub-parameters can also be individually assigned, at any time and in any order. Any attempt to retrieve the value of 'TOD' results in an HHMMSS object reconstituted from the current values of 'TODhh', 'TODmm', and 'TODss'.

Option	Description

	In order for a class to be compatible with the decomposition option, it must have two crucial qualities: The constructor for the class must be able to create an instance of itself from a tuple of values, and instances of the class must be able to return the same elements as the original tuple when accessed in an array-like manner. The way the HHMMSS class satisfies these requirements are fairly complicated, but it is extremely useful to note that the standard python tuple class satisfies the requirements by default. Thus, declarations of custom decomposable data types based on the tuple class are trivial to implement:
	
	`class XYZ (tuple):`
	`  decomposition = (`
	`  Parameter(suffix="x", type=Distance),`
	`  Parameter(suffix="y", type=Distance),`
	`  Parameter(suffix="z", type=Distance),`
	`  )`
	
	Note that sub-parameters of a decomposed parameter fully implement type safety and unit awareness: thus each distance value in an XYZ vector use the same unit of distance as any other parameter of the container.

format	the format option controls the printed precision of unit-based parameters, when they are being written to a file. IE: format='.2f' limits the printed precision to two decimal places
deprecated	when a parameter has a deprecated argument set to True, that signifies that the parameter is not recommended for use in future development.

Parameter Roles

Role	Description
'input'	an input parameter for an Application container
'output'	an output parameter for an Application container
'attribute'	a generic "object attribute"
'annotation'	annotations are parameters of a TableData container that provide some sort of informative meta-data, and are printed at the top of an output csv file above the column header.
'column'	columns are parameters of a single-row TableData container that describe the name and type of a column, as it may be named in the column header of an output csv file.

Deprecated Compatibility Functions

The following functions re-implement the declarations of old Parameter classes, replacing them with the new, unified Parameter class. They are provided here only to help understand any code that has not yet been updated to reflect the new style:

Section 2.4: Band and Bands, RecNum and RecNums

SuperFAR provides a Band class to represent a single ANSI/ISO one-third octave band. For example, the expression

```
Band(17)
```

represents ANSI/ISO band number 17.

Band is a subclass of the python int class, and thus can be used in place of an int in any python expression that normally expects an int. Thus, all of the following python expressions are True:

```
Band(17) is not 17
Band(17) == 17
Band(17) + 1 is 18
{Band(17):'match'}[17] is 'match'
```

SuperFAR also provides a distinct but closely related class named Bands that represents a contiguous range of Band values. The following expressions all result in objects representing a range of bands from 17 to 40 inclusive:

```
Bands(17,40)
Bands((17, 40))
Bands([17, 40])
```

The class RecNum and RecNums behave in a similar manner to Band and Bands, but represent record numbers in a time history file. For example:

```
RecNum(1)
RecNums(1, 50)
```

represent record 1 and the range of records from 1 to 50 respectively.

Band and RecNum are subclasses of ExclusiveIntSubclass, which throws an an exception whenever any attempt is made to cast a Band into a RecNum or a RecNum into a band.

NOTE: Due to limitations in the specifications for Perceived Noise Level (PNL), Tone-Corrected Perceived Noise Level (PNLT), and Effective Perceived Noise Level (EPNL), the range of bands handled by SuperFAR defaults to band 17 through 40 (50 Hz through 10 kHz). While SuperFAR is capable of handling broader band ranges, these metrics will not be able to handle them without external development of the metrics themselves.

Section 2.5: Measurable Quantities and Unit Parameters

A SuperFAR quantity value is a python object that stores a real-world physical measurement (such as distance), in a form that is tightly bound to the physical unit of the measurement (such as feet). SuperFAR accomplishes this by having a different quantity class for every unit it can handle: for example, the Feet class in SuperFAR is used to store measurements in feet, and the expression Feet(3) creates a quantity value of 3 feet. " If a quantity class is invoked on an object that is already a quantity value, the value is converted to the unit of the new class: for example, the expression Feet(Meters(1)) evaluates to Feet(3.280839895013123). This is assuming of course that there is a sensible conversion method to apply between the two quantity classes: nonsensical conversions such as Feet (Seconds(1)) throw an exception. "

Quantity values also support several arithmetic operations. For example, addition and subtraction is supported between values of the same quantity type, and produce a result of the same unit type. Distance values can be divided by time values to get speed values, istance values can be divided by speed values to get time values, and speed values can be multiplied by time values to get distance values. Generally speaking, if an operation on two unit values obeys the rules of proper unit arithmetic, and the units of the result are supported by SuperFAR, the operation should be supported.

SuperFAR also has a Unit objects that identify units by name. For example, the expression Unit('Feet') returns an object that identifies Feet as a unit. A unit object can be used in the place of a directly-named quantity class to create a quantity object: Unit('Feet')(3) evaluates to Feet(3). The advantage of using Unit objects over directly named quantity classes is that Unit objects are not case sensitive, and also accept variety of aliases and abbreviations: Unit ('FEET'), Unit('foot'), and Unit('ft') are all equivalent to Unit('Feet').

Unit parameters control the unit of measurement used for values of the corresponding quantity type, within the same SuperFAR application or data object. For a concrete example of usage, see Applications.

Unit Parameters can be set to a string value containing the name of a unit, such as 'Feet' or 'Meters'. Each string name corresponds to a quantity unit in a fairly straightforward manner, but is not case sensitive, and also allows alternate names and aliases such as 'ft' or 'm'.

Units that are defined as one unit divided by another unit (such as feet per second) can be specified by a string containing *any* valid alias of the numerator and *any* valid alias of the denominator separated by a slash: ie 'feet/second', 'ft/sec', and so on.

The Feet class is a subclass of Distance, an abstract base class for all SuperFAR classes that represent concrete distance quantities. More generally, all python quantity classes that correspond to a named unit (such as Feet) are subclasses of one of the following abstract classes: Alpha, Angle, Distance, Frequency, Humidity, Ratio, Speed, Temperature, and Time.

When parameters that have an abstract quantity type are assigned a value, the value to be stored in the container is converted to concrete quantity class (such as Feet), based on the current value of the container's applicable unit parameter (ie, DistanceUnits for Distance parameters).

Table 2.5.1: Alpha Units

name of unit	aliases	quantity class	description
"dB100m"	"dbp100m"	dB100m	
"dB1kft"	"dbp1kft"	dB1kft	
"dBft"	dB/Feet	dBft	
"dBm"	dB/Meters	dBm	

Table 2.5.2: Angle Units

name of unit	aliases	quantity class	description
"Degrees"	"degree"	Degrees	Degrees
"Radians"	"rad", "radian"	Radians	Radians

Table 2.5.3: Distance Units

name of unit	aliases	quantity class	description
"Feet"	"foot", "ft"	Feet	feet
"Inches"	"in", "inch"	Inches	inches
"Meters"	"m", "meter"	Meters	meters
"Miles"	"mile"	Miles	miles
"Nautical Miles"	"nm"	Nautical_Miles	nautical miles
"Yards"	"yard", "yd"	Yards	yards
"kM"	"1000m", "kilometer", "kilometers"	kM	thousands of meters
"100m"	"hectometer", "hectometers", "hm", "x100m"	x100m	hundreds of meters
"1kft"	"x1kft"	x1kft	thousands of feet

Table 2.5.4: Frequency Units

name of unit	aliases	quantity class	description
"Hz"	"hertz"	Hz	Hertz
"kHz"	"kilohertz"	kHz	Kilohertz

Table 2.5.5: Humidity Units

name of unit	aliases	quantity class	description
"RH"		RH	Percent Relative Humidity

Table 2.5.6: SPL Units

name of unit	aliases	quantity class	description
"dB"	"decibel", "decibels"	dB	Decibels

Table 2.5.7: Speed Units

name of unit	aliases	quantity class	description
"FPS"	Feet/Seconds	FPS	feet per second
"KPH"	kM/Hours	KPH	kilometers per hour
"Knots"	Nautical Miles/Hours	Knots	nautical miles per hour
"MPH"	Miles/Hours	MPH	miles per hour
"MS"	Meters/Seconds, "metersps", "mps"	MS	meters per second

Table 2.5.8: Temperature Units

name of unit	aliases	quantity class	description
"Celsius"	"c"	Celsius	
"Fahrenheit"	"f"	Fahrenheit	
"Kelvin"	"k"	Kelvin	
"Rankine"	"r", "ra"	Rankine	

Table 2.5.9: Time Units

name of unit	aliases	quantity class	description
"HHMMSS"		HHMMSS	a time value represented by a string containing hours, minutes, and seconds separated by colons. For example: "3:22:05" means three hours, 22 minutes, and 5 seconds. Parameters of this type can also accept a simple numeric value as input, which is interpereted to mean a number of seconds.
"Hours"	"h", "hour", "hr", "hrs"	Hours	hours
"Minutes"		Minutes	minutes
"Seconds"	"s", "sec", "second"	Seconds	seconds

Section 3: SuperFAR Applications

Each SuperFAR application is a python function, or to be more precise a python class that is invoked in a function-like manner. The object created by invoking a SuperFAR application contains all of the output values of a computation. For example:

```
result = ARP866A (
  TemperatureUnits = "Celsius",
  temperature = 25,
  humidity = 50,
)
alpha = result.alpha
```

This script invokes the ARP866A application with a given temperature and humidity, then extracts the alpha value contained in the result. All SuperFAR functions follow this basic usage pattern: Input values are passed into the application using the "name = value" syntax, and output values can be retrieved from the result object using "object.name" syntax.

The names of input parameters and output object attributes are both not case sensitive.

All input and output parameters of a SuperFAR application have a parameter type, which in many cases is a unit type. In the above example, the TemperatureUnits parameter is set to "Celsius". This specifies that when the temperature parameter is set to the unitless numeric value of 25, it is internally "coerced" to the unit-aware value Celsius(25).

When a value that is already unit-aware is provided for a unit aware parameter, but the unit of the provided value differs from the desired unit of the parameter, it is coerced into the correct unit by unit conversion:

```
result = ARP866A (
  TemperatureUnits = "Celsius",
  temperature = Fahrenheit(77),
  humidity = 50,
)
alpha = result.alpha
```

In this case the value Fahrenheit(77) is converted to Celsius(25), and thus ultimately has the same result as the previous example.

If a parameter is assigned a value of a fundamentally inconvertible type, or a random python object that has no sensible coercion policy, SuperFAR raises a python runtime exception.

The final value an input parameter is converted to is always accessable in the result object using the "object.name" syntax. Thus, in either of the previous two cases, result.temperature would return Celsius(25)

Many SuperFAR applications read input files, or write output files. The path that input files should be read from, and the path that output files should be written to, can be specified as named parameters:

```
Adj2Ref (
    ProjectName = 'Regression Test',
    input_SSRData = 'Regression Data/PreDx.ssr.csv',
    input_TD_ALFData = 'Regression Data/CL_101_1.td.alf.csv',
    input_R_ALFData = 'Regression Data/CL_101_1.ff.alf.csv',
    input_SR = 10,
    input_SSR = 20,
    output_SSRData = 'Results/Adj2Ref.1.ssr.csv',
)
```

Section 3.1: ALF2ATH Application

generates ATH (Alpha Time-History) data from single ALF spectrum for each record in STH.

(Each record/row/spectrum in the alpha time-history is identical to the input ALF spectrum.)

Unit Parameters:

Name	Type	Default
AlphaUnits	Alpha Unit	"dB100m"

Input Parameters:

Name	Type	Default	Description
ALF	ALFData	None	input alpha spectrum, providing alpha data
STH	STHData	None	input spectrum time history, providing time history history RecNums and Timestamps

Output Parameters:

Name	Type	Description
ATH	ATHData	output alpha time history

Section 3.2: ARP866A Application

Calculates an atmospheric absorption coefficient (alpha) per SAE ARP866a for a specified temperature, humidity and frequency using the quadratic interpolation method.

Unit Parameters:

Name	Type	Default
TemperatureUnits	Temperature Unit	"Celsius"
AlphaUnits	Alpha Unit	"dB100m"

Input Parameters:

Name	Type	Default	Description
temperature	Temperature	None	
humidity	RH	None	
frequency	Hz	None	

Output Parameters:

Name	Type	Description
alpha	Alpha	

Section 3.3: ARP866A_spectrum Application

Calculates absorption coefficients (alphas) for a specified temperature and humidity pair over a user-selected range of frequency bands using ARP866A, and stores the result in an SSR file/object. If no user-selection for range, uses default of ANSI/ISO bands 17-40 (50 Hz – 10 kHz)

Unit Parameters:

Name	Type	Default
TemperatureUnits	Temperature Unit	"Celsius"
AlphaUnits	Alpha Unit	"dB100m"
DistanceUnits	Distance Unit	"Feet"

Input Parameters:

Name	Type	Default	Description
temperature	Temperature	None	
humidity	RH	None	

Output Parameters:

Name	Type	Description
ALF	ALFData	

Section 3.4: Adj2Ref Application

Adjusts a single 1/3 octave band spectrum to reference conditions, accounting for differences in test and reference propagation distance and test and reference atmospheric absorption coefficients. Currently works with ARP866A method. Will need to be modified to handle ARP5534.

Called By:

Integrated

Unit Parameters:

Name	Type	Default
DistanceUnits	Distance Unit	"Feet"

Input Parameters:

Name	Type	Default	Description
tdSSR	SSRData	None	test day SPL spectrum to be adjusted to reference conditions
tdALF	ALFData	None	A SPL spectrum of (typically 24) test-day atmospheric absorption coefficients (alphas).
refALF	ALFData	None	A spectrum of Reference Condition atmospheric absorption coefficients (alphas)
sr	Distance	None	Test-day propagation distance for the spectrum
srr	Distance	None	Slant Range Reference propagation distance for the spectrum

Output Parameters:

Name	Type	Description
refSSR	SSRData	spectrum adjusted to reference conditions

Section 3.5: Badjer Application

Background noise adjuster

- identifies valid pre-detection levels
- establishes masking criteria
- identifies masked aircraft levels
- de-combines valid pre-detection noise SPLs from valid aircraft noise SPLs by performing 'energy'-subtraction
- applies broadband and frequency-dependent system corrections (microphone & windscreen response, system frequency response, gain, calibration drift, etc.) to valid aircraft SPLs
- generates masking map for spectral-time-history

Calls:

MaskMan
ValidAdj

Input Parameters:

Name	Type	Default	Description
predetectSSR	SSRData	None	pre-detection background noise levels.
predetectWindow	dB	3	
postdetectSSR	SSRData	None	post-detection background noise levels.
postdetectWindow	dB	1	
STH	STHData	None	
correctionSSR	SSRData	None	
correctionValue	dB	* dynamic	

Output Parameters:

Name	Type	Description
validpreSSR	SSRData	adjusted data spectrum
maskcrtSSR	SSRData	
maskMap	MapData	
ambisubMap	MapData	
ambisubSTH	STHData	
gainadjSTH	STHData	
badjerMap	MapData	
badjerSTH	STHData	

Section 3.6: EPNLCalc Application

EPNL Calculator

- Used for test-day and reference condition EPNL (called by Integrated for Ref EPNL)
- Identifies Maximum record, first and last 10 dB-down records, and secondary peak records for each metric in the input .MTX file and inserts the appropriate labels in the xINT columns of the output metric time history
- Re-calculates effective time interval (EffInt) duration for each record based on input MTX timestamps
- Calculates Bandsharing adjustment for PNLT and applies it to PNLT Max level.
- Identifies Maximum record, first and last 10 dB-down records, and secondary peak records for each metric in the input .MTX file and inserts the appropriate labels in the xINT columns of the output metric time history
- Performs time-integration to obtain Time-Integrated LEvel or "TILE" (EPNL, SEL, etc.) Note that the reference duration is 10 seconds for EPNL (using PNLT or PNL only), and 1 second for SEL (using any other metric).
- Compares NVR data vs 10dB-down points, Max, etc. and tests for violations - generates warnings in EPNL.RPT file
- generates warnings in EPNL.RPT file
- Outputs a new xxxxEPNL.MTX.csv file, which has new columns inserted for EffInt, xINT labels for each metric type, and updated NVR data.

Called By:

Integrated

Input Parameters:

Name	Type	Default	Description
inputMTX	MTXData	None	input metric time history file
comments	String	None	comment to be inserted into output files

Output Parameters:

Name	Type	Description
epnlMTX	MTXData	output metric time history file
EPNL	EPNLData	output EPNL report file

Section 3.7: GeoCalc Application

Builds a Geometry Time History that contains data (T, X, Y, Z, sound propagation distance, and sound emission angle) for aircraft geometry at time of emission for each measured acoustic data record relative to a specific microphone. Uses the approximate straight-line flight path methodology provided in the ETM.

- Requires that single-point straight-line flight path descriptors be available (either from externally-provided source such as photographic positioning methodology, or by running SPoinTrkOut on Position Time History (PTH) data set.)
- Also requires that PTH data set be available (either directly from measured TSPI data, or by running SPoinTrkIn using externally-provided single-point straight line flight path descriptors.)
- Requires RecNums and Timestamps from test-day aircraft noise data spectral time-history file (.STH).

Unit Parameters:

Name	Type	Default
AngleUnits	Angle Unit	"Degrees"
DistanceUnits	Distance Unit	"Feet"
SpeedUnits	Speed Unit	"FPS"
TemperatureUnits	Temperature Unit	"Celsius"

Input Parameters:

Name	Type	Default	Description
STH	STHData	None	aircraft acoustics data
PTH	PTHData	None	A series of aircraft positions vs time (usually TSPI data)
MIC	MICData	None	Microphone coordinates & height above ground.
SPO	SPOData	None	Single-Point straight-line flight path descriptors
Temperature	Temperature	None	Average air temperature used in calculation of soundspeed, (written to GTH file header)
SoundSpeed	Speed	None	speed of sound

Output Parameters:

Name	Type	Description
GTH	GTHData	geometric time history of emission data for each input record

Section 3.8: Integrated Application

Integrated Procedure for Reference Condition EPNL

Calculates the Reference Condition EPNL using the Integrated Method (See ICAO Annex 16 Vol. I, Appendix 2, Section 8.4 and ICAO ETM Vol. I, Chapter 4, Section 4.3.1.4) Assumes that test-day processing has been completed through EPNLCalc and that RefGeo has been run.

Calls Metrix and EPNLCalc modules. Reads Reference Reception Times (TR) from REF.GTH file and assigns as Reference Condition Timestamps in Metrix. EPNLCalc then re-calculates Effective Interval (EffInt) for each spectrum.

Calls:

Adj2Ref
Metrix
EPNLCalc

Unit Parameters:

Name	Type	Default
DistanceUnits	Distance Unit	"Feet"

Input Parameters:

Name	Type	Default	Description
bugfix	Boolean	False	if set to True, EPNLCalc parameters inputMTX and epnlMTX are not both assigned to the same object
tdSTH	STHData	None	A test day time-history of one-third octave band SPLs. For normal operation, use the most recently generated .STH.CSV file for a particular run.
tdMap	MapData	None	A test day time-history of masking and adjustment codes for each 1/3 octave band SPL in the tdSTH, plus a history of the NVR codes for each spectrum. (Used by Metrix app.
tdATH	ATHData	None	A test day time-history of 1/3 octave band alphas (atmospheric absorption coefficients) corresponding to each 1/3 octave band SPL in the tdSTH. (Used by Adj2Ref function)
tdALF	ALFData	None	A single spectrum of test day alpha values, as an alternative to tdATH.
refALF	ALFData	None	A single spectrum of reference-day atmospheric absorption coefficient constants. (Used by Adj2Ref function)
refGTH	REFGTHData	None	provides Slant Range (Test-Day propagation distance) for each record in the “SR” column, the Reference Slant Range (Ref propagation distance) for each record in the “SRR” column, and provides reference reception time in the "TR" column for use in re-computing the, effective duration for each record in the “EffInt” column.
AFlag	Boolean	True	include ANSI A-weighted metrics if true
CFlag	Boolean	True	include ANSI C-weighted metrics if true
DFlag	Boolean	True	include ANSI D-weighted metrics if true
OAFlag	Boolean	True	include unweighted metrics if true
PFlag	Boolean	True	include PNL metrics if true
TFlag	Boolean	True	include PNLT metrics if true
TCB40Flag	Boolean	False	
NoRoundFlag	Boolean	False	
HelicopterFlag	Boolean	False	
TCLowBand	Band	None	

Output Parameters:

Name	Type	Description
EPNL	EPNLData	output EPNL report file
epnlMTX	MTXData	output metric time history file
refSTH	STHData	

Section 3.9: MASKCRT Application

Determines masking criterion for each one-third octave band.

Called By:

MaskMan

Input Parameters:

Name	Type	Default	Description
validpreSSR	SSRData	None	
postdetectSSR	SSRData	None	
predetectWindow	dB	3	
postdetectWindow	dB	1	

Output Parameters:

Name	Type	Description
maskcrtSSR	SSRData	

Section 3.10: MASKMODE_MAP Application

Identifies masking mode for each 1/3 octave band in each spectrum record in the input SF_SpectralTimeHistory ("sAircraft.sth") from loRec to hiRec and stores masking codes for each calculated SF_Maskrec into the resulting SF_MaskMap ("sMapOut.map").

Called By:

MaskMan

Input Parameters:

Name	Type	Default	Description
STH	STHData	None	
predetectSSR	SSRData	None	
postdetectSSR	SSRData	None	
maskcrtSSR	SSRData	None	
lowRec	Integer	* dynamic	
highRec	Integer	* dynamic	

Output Parameters:

Name	Type	Description
maskMap	MapData	

Section 3.11: MaskMan Application

Identifies valid pre-detection levels, establishes masking criteria, identifies masked aircraft levels, and generates masking map for spectral-time-history.

Calls:

PREDETEST
MASKCRT
MASKMODE_MAP

Called By:

Badjer

Input Parameters:

Name	Type	Default	Description
predetectSSR	SSRData	None	pre-detection background noise levels.
postdetectSSR	SSRData	None	post-detection background noise levels.
predetectWindow	dB	3	
postdetectWindow	dB	1	
STH	STHData	None	
lowRec	Integer	* dynamic	
highRec	Integer	* dynamic	

Output Parameters:

Name	Type	Description
validpreSSR	SSRData	adjusted data spectrum
maskcrtSSR	SSRData	
maskMap	MapData	

Section 3.12: Metrix Application

Applies selected frequency-weighting to each one-third octave spectrum, and calculates the broadband level for each selected weighting; Also, if selected, calculates Perceived Noise Level (PNL), Tone-Correction (TC), and PNLT, and saves these values to a metrics time-history (.MTX) file.

Note that PNL and selected frequency-weighted metrics are computed by virtual parameters within the SuperFAR SSRData object specification (See SRRData.py for PNL code and frequency-weighting constants).

Calculates metric values (specified in the flags input property) for each spectrum in a spectral time history object, Generates a metrics time-history which may be saved by the calling software.

Compares Masking and adjustment codes from .MAP file for individual bands in each spectrum to criteria provided in the Background Noise Adjustment Procedure Volpe developed for AC36-4C and the ETM. Generates NVR (Non Valid Record) code for each spectrum, and saves them in the .MTX file.

Note: This module is the only one that accesses NVR data from both the .MAP and the .MTX files.

Called By:

Integrated

Input Parameters:

Name	Type	Default	Description
STH	STHData	None	
Map	MapData	None	
AFlag	Boolean	True	include ANSI A-weighted metrics if true
CFlag	Boolean	True	include ANSI C-weighted metrics if true
DFlag	Boolean	True	include ANSI D-weighted metrics if true
OAFlag	Boolean	True	include unweighted metrics if true
PFlag	Boolean	True	include PNL metrics if true
TFlag	Boolean	True	include PNLT metrics if true
TCB40Flag	Boolean	False	
HelicopterFlag	Boolean	False	
NoRoundFlag	Boolean	False	
TCLowBand	Band	None	

Output Parameters:

Name	Type	Description
MTX	MTXData	

Section 3.13: PNLTCalc Application

Computes TC and PNLT values for a given spectrum. Runs the ToneCorrection (previously TCCALC_1) function. Obtains PNL from the SSRData object specification.

Input Parameters:

Name	Type	Default	Description
SSR	SSRData	None	
HelicopterFlag	Boolean	False	
NoRoundFlag	Boolean	False	
LGB	Integer	40	
TCLowBand	Band	None	

Output Parameters:

Name	Type	Description
TC	float	
TCBand	Band	
PNL	float	
PNLT	float	

Section 3.14: PREDETEST Application

Tests pre-detection background noise levels against post-detection background noise levels, and sets invalid pre-detection levels to SF_INVALID_LEVEL

Called By:

MaskMan

Input Parameters:

Name	Type	Default	Description
predetectSSR	SSRData	None	pre-detection background noise levels.
postdetectSSR	SSRData	None	post-detection background noise levels.

Output Parameters:

Name	Type	Description
validpreSSR	SSRData	adjusted data spectrum

Section 3.15: ReAvg Application

Performs simulated Slow time-averaging (using continuous or x-sample functions) on a spectral time-history that was analyzed using half-second linear averaging (integration), and assigns slow timestamps.

Input Parameters:

Name	Type	Default	Description
inputSTH	STHData	None	The spectral time-history of linear averaged aircraft noise data.
inputMap	MapData	None	optional map file that matches input linear sth
AveragingMethod	String	"CONTEXPO"	Accepts the following keywords: "CONTEXPO" "4S100" "4S95"
TimeStamp	String	"ICAOSLO"	Accepts the following keywords: "ICAOSLO" "OLDSLO" "START" "MIDPT" "END"
low_rec	Integer	* dynamic	
high_rec	Integer	* dynamic	

Output Parameters:

Name	Type	Description
outputSTH	STHData	The output spectral time-history of Slow time-averaged aircraft
outputMap	MapData	The output map file corresponding to the Slow time-history.

Section 3.16: Reconstruct Application

The concepts, processes and algorithms in this Application are based on Guidance Materials developed by the Volpe Center Acoustics Facility, and accepted by domestic and international aircraft noise certificating bodies in 2003, published as "Appendix 3" to both the ICAO Environmental Technical Manual - Third Edition, and FAA's Advisory Circular AC36-4C: "Guidelines for Adjustment of Aircraft Noise Levels for the Effects of Background Noise". These guidelines (to be referred to as "Appendix 3") included a complete, step-by-step procedure (referred to as "VCAF03" methodology) includes including the "LGB" concept, which was accepted by the authorities as an officially approved procedure. Appendix 3 also included a section outlining limits and requirements applicable to ANY methodology developed to deal with background noise (to be referred to as "Section 4" limitations and requirements.) To further complicate things, FAA has since determined, under advisement by Volpe, that the simpler procedure provided in earlier version of Advisory Circular AC36-4B, should be allowed and approved as equivalent to the VCAF03 methodology. The primary difference, being that while the VCAF03 methodology adapted the earlier ICAO ETM assumption of a flat spectral shape at a distance of 60 meters from the aircraft under standard atmospheric conditions (ISA+10 degrees C: 25C/70%RH or 77F/ 70%RH), the older AC method assumes flat spectral shaping at a distance of ZERO from the aircraft. This simpler methodology will be referred to as the "AC36-4B" variant of the VCAF03 methodology.

'Time-extrapolation section ----------
This section handles the set-up and control of data passed to the TimeX function, which works on a single SPL at a time, unlike the AvAdj and FreqX functions, which handle an entire spectral record at a time. Time extrapolation is performed per Appendix 3 of FAA's AC36-4C and ICAO's ETM Third Edition, reconstructing the value for a masked SPL by starting with a valid SPL in the same band, but from a different spectral record (or point in time). The criterion for Time- extrapolation is that when any of the bands between band 29 and 33 (800Hz to 2kHz) in a spectrum are masked, time-extrapolation is performed on all HF bands (Band 29 thru 40; 800Hz through 10kHz) in that spectrum. (Note: Code has been set up to allow an override of this when Time- and Freq- extrapolation are mixed within a single spectrum. This special condition was an option in the DOSFAR software, and may be implemented in future in SuperFAR for research purposes.) When using the LGB methodology, Time extrapolation is performed on all masked HF bands in a spectrum when that spectrum's LGB is between Bands 28 and 32 (630 Hz to 1.6kHz) inclusive.

NOTE: This version of Reconstruct uses the LGB Method exclusively. Two schema are provided for identifying center_rec (the source of time-extrapolation): "Slice" and "Spectrum"; The "Spectrum" scheme is used when information is available regarding the "center" of the Spectral Time-History: That is, the record of PNLTm and/ or the Record closest to TOH (Time at OverHead). If either of these data are 'known, then it is preferred to perform time- extrapolation outward from one of these points for every band that is being time-extrapolated. The "Slice" scheme is used when this information is not available; instead, for each one-third octave band to be time-extrapolated, the maximum SPL in the band is identified and the spectrum containing that Max SPL is used as the point from which to extrapolate outward. Note that a third scheme, where the nearest valid SPL in a band in used as the source for Time-extrapolation has not been codified yet, even though the simplistic language in Appendix 3 regarding time- extrapolation would seem to indicate that this is the preferred method. Such a scheme could potentially extrapolate an SPL "inward" from one that is valid, but has been propagated from farther away, and therefore less accurate than one which is farther away in time, but propagated over a shorter distance. Future revision of Reconstruct may incorporate this scheme.

Unit Parameters:

Name	Type	Default
DistanceUnits	Distance Unit	"Feet"

Input Parameters:

Name	Type	Default	Description
inputSTH	STHData	None	Background-adjusted Aircraft Spectral Time-History data (gen by BADJER tool or MaskMan/ValAdjust)
inputMap	MapData	None	Associated MapData / adjustment code data (gen by BADJER tool or MaskMan/ValAdjust)
inputGTH	GTHData	None	Associated GTHData emission TXYZ and SR info (gen by GEOCALC) not required for fixed-sloping option.
tdATH	ATHData	None	Time history of Test-Day propagation path atmospheric absorption coefficients. Either this or TDAlfa must be input. If neither is set, alphas default to zero.
tdALF	ALFData	None	Test-Day propagation path atmospheric absorption coefficients (SF_MetAlfa) (gen by PROPALFAS or other means) not required for fixed-sloping option. Either this or TDAlfaTimeHistory must be set, if neither are set then TDAlfaTimeHistory defaults to zero.
ffALF	ALFData	None	Free-Field Alphas (SF_MetAlfa) for empirically-derived source spectral shaping data (Obtained by various means) - not required for fixed-sloping option, or for AC36-4B Method: Flat (or shaped) at ZERO distance from source. (xxxxFF.ALF) (Optional input, default all zeros)
ffSSR	SSRData	None	[optional] Empirically-derived Free-Field Spectrum Shape data (SF_Spectrum), either absolute SPLs or relative values. Default to all zeroes for flat spectrum at "source" or Free-Field Distance, which defaults to 60 meters. NOTE: Possibility of handling multiple FF spectra based on emission angle or other geometrical variable) (or allow user to set all zeroes, or fixed slope, i.e., -3dB/third-oct band) (xxxxFF.SSR)
FFMR	dB	None	Free-Field Spectrum - Masked Record: The 1/3 octave band SPL value of the band of interest in the Free-Field shaping Spectrum (determined via angular relationship of the Masked record when the aircraft spectral shaping is considered to be non-omnidirectional); used in determining the spectral shaping to be applied in addition to the time-extrapolation. (Default: spectral values equal to zero - assuming an omnidirectional flat spectral shape under free-field conditions).
FFGR	dB	None	Free-Field Spectrum - Good Record: The 1/3 octave band SPL value of the band of interest in the Free-Field shaping Spectrum (determined via angular relationship of the Good record when the aircraft spectral shaping is considered to be non-omnidirectional); used in determining the spectral shaping to be applied in addition to the time-extrapolation. (Default: spectral values equal to zero - assuming an omnidirectional flat spectral shape under free-field conditions).
LowBand	Band	17	(50Hz); If not specified elsewhere, the Approved lowest freq ANSI/ISO band of interest: 50Hz.
HighBand	Band	40	(10kHz); If not specified elsewhere, the Approved highest frequency ANSI/ISO band of interest is 10kHz.
TopLFBand	Band	28	(630Hz); For the VCAF03 / App3 BG Noise Procedure, the highest-frequency "Low Frequency" Band is 630Hz.
BtmHFBand	Band	29	(800Hz); For the VCAF03 / App3 BG Noise Procedure, the lowest-frequency "High Frequency" Band is 800Hz.
FXCritBand	Band	33	(2kHz); Criterion: ANSI/ISO Band number for lowest allowed value of GB for Frequency-extrapolation. (Can be superseded for research) Freq-Extrapolation is ONLY APPROVED from band 34 (2.5kHz) upward.

Name	Type	Default	Description
TXCritBand	Band	28	(630Hz); Criterion: ANSI/ISO Band number for lowest allowed value of GB for Time-extrapolation. (Can be superseded for research) Time-Extrap is ONLY APPROVED only from band 29 (800Hz) upward.
StopBand	Band	28	ANSI/ISO Band number for the upper limit of AvAdj processing; Since AvAdj normally used only for LF bands, deafaults to ANSI/ISO 28.
FFSR	Distance	60	FFSR is "Free-Field Slant Range", The Free-Field Acoustic Propagation distance from the Aircraft to the theoretical microphone position for the Empirically-derived source spectral shaping. By Default, this distance is 60 meters (Based on Approved VCAF2003 Appendix 3 Guidance).
TXCRec	Integer	1	Time-Extrapolation record number. Record number of the spectrum/maskrow to be used as the center point for time-extrapolation of the current band
TXTop	Integer	40	For future expansion - TXTop is used as a means to stop TimeX at band other than 40, such as in cases where the user selects to combine time and frequency extrapolation methods whithin a single spectrum. (Default = 40). May be set to 32 for Timex only if Bands not eligible for Freq-extrap.
TXCType	String	"Slice"	Accepts the following keywords: "Slice" "Spectrum"
AC364Bflag	Boolean	False	sets the FFSR parameter to zero, reproducing the "AC36-4B" variant of the VCAF03 methodology which assumes flat spectral shaping at a distance of zero meters from the aircraft.
LGBFlag	Boolean	True	
MultiGBFlag	Boolean	False	
VCAF03Flag	Boolean	True	
FixedSlopeFlag	Boolean	False	
BirdBugFlag	Boolean	False	
MixTmFrqFlag	Boolean	False	
HFAvAjdFlag	Boolean	False	
LFTimeAvgFlag	Boolean	False	
SegDFlag	Boolean	False	
TXOutwardFlag	Boolean	True	
AvAdjFlag	Boolean	True	
TimeXFlag	Boolean	True	
FreqXFlag	Boolean	True	
UseMaskedFlag	Boolean	False	

Output Parameters:

Name	Type	Description
AvgAdjSTH	STHData	
AvgAdjMap	MapData	
FreqSTH	STHData	
FreqMap	MapData	
TimeSTH	STHData	
TimeMap	MapData	
OutputSTH	STHData	
OutputMap	MapData	

Section 3.17: RefGeo Application

Calculates the Reference Condition emission coordinates, sound propagation distances, reception times, and effective intervals* from the test-day noise geometry time history, Test-day spectral time-history timestamps, and single-point tracking data values for the reference track. (See ICAO ETM Vol. I, Chapter 4, Section 4.3.1.2) Assumes that test-day noise geometry has already been obtained by running GeoCalc App.

*NOTE: EffInt is re-computed within EPNLCalc and Integrated modules using Reference timestamps taken from Reference Reception times (TR) in the REF.GTH output from this module.

Unit Parameters:

Name	Type	Default
DistanceUnits	Distance Unit	"Feet"
AngleUnits	Angle Unit	"Degrees"
SpeedUnits	Speed Unit	"FPS"

Input Parameters:

Name	Type	Default	Description
SPO	SPOData	None	
tdSTH	STHData	None	
tdGTH	GTHData	None	
TOHR	HHMMSS	None	The time at which the aircraft is overhead or abeam the test microphone on the Reference flight path. Typically ZERO or alternately, equal to TOH.
ZOHR	Distance	None	The vertical distance of the reference flight path above the ground at the reference microphone location.
YMICR	Distance	None	The lateral coordinate for the reference Microphone location: typically zero for centerline, and either +/-150m or +/-450m for sideline.
VGR	Speed	None	The Reference ground speed of the aircraft. Horizontal component of the reference speed along the reference flight path.) Units: "KTS", "MPH", "KMH", "FPS","M/S"
RGAMMA	Angle	None	The reference climb/descent angle
TOH	HHMMSS	* dynamic	The time at which the aircraft is overhead or abeam the test microphone on the test flight path.
ZOH	Distance	* dynamic	The vertical distance of the test-day straight-line flight path above the ground at the microphone location.
VG	Speed	* dynamic	The average test-day ground speed of the aircraft, based on the noise duration (10-dB-down points)
GAMMA	Angle	* dynamic	The average climb/descent angle of the straight-line test-day flight path, based on the noise duration (10-dB-down points)
CPA	Distance	* dynamic	Closest Point of Approach (minimum distance between test microphone and the straight-line test-day flight path).

Output Parameters:

Name	Type	Description
refGTH	REFGTHData	

Section 3.18: SPoinTrkIn Application

Takes as input single-point tracking data from user prompts or an .INI file, and generates a Position Time History (.PTH) file that can be read by GeoCalc.

Unit Parameters:

Name	Type	Default
DistanceUnits	Distance Unit	"Feet"
SpeedUnits	Speed Unit	"FPS"
AngleUnits	Angle Unit	"Degrees"

Input Parameters:

Name	Type	Default	Description
StartTOD	HHMMSS	None	
TOH	HHMMSS	None	
EndTOD	HHMMSS	None	
TInt	Seconds	* dynamic	
AltOH	Distance	None	
YOff	Distance	None	
GSPD	Speed	None	
VSPD	Speed	None	
PHI	Angle	None	
CDSlope	float	None	
CDRise	Distance	None	
CDRun	Distance	None	
CYAng	Angle	None	
CYSlope	float	None	
CYY	Distance	None	
CYX	Distance	None	

Output Parameters:

Name	Type	Description
PTHData	PTHData	

Section 3.19: SPoinTrkOut Application

Generates Single-Point Tracking info from a .GTH (Emission coordinate Geometry Time History re microphone) or a .PTH (Raw aircraft TXYZ Position Time History) file. If an .MTX (Metrics Time History file is provided, it will display and prompt the First and last 10 dB- down point selections for each metric type in the file. The user can select from any of these (default: PNLT F10 and L10), or can manually input the limits of the averaging process (AvStart and AvEnd) via prompts or .INI file. Usage note, accepted data: - Accepts either PTH OR GTH data. - If PTH data then no MTX data. - MIC data is optional

Unit Parameters:

Name	Type	Default
DistanceUnits	Distance Unit	"Feet"
AngleUnits	Angle Unit	"Degrees"
SpeedUnits	Speed Unit	"FPS"

Input Parameters:

Name	Type	Default	Description
GTH	GTHData	None	
PTH	PTHData	None	A series of aircraft position vs time (usually TSPI data)
MTX	MTXData	None	
MIC	MICData	None	
AvStartTime	HHMMSS	None	
AvEndTime	HHMMSS	None	
XOH	Distance	None	

Output Parameters:

Name	Type	Description
SPO	SPOData	

Section 3.20: SSPDCalc Application

compute speed of sound from temperature

Unit Parameters:

Name	Type	Default
SpeedUnits	Speed Unit	"FPS"
TemperatureUnits	Temperature Unit	"Celsius"

Input Parameters:

Name	Type	Default	Description
Temperature	Temperature	None	air temperature
SSPDCalcType	String	"ICAO_FIXED"	Accepts the following keywords: "ICAO_FIXED" "RICKLEY" "ICAO_TM" "SUPR_EZ" "BERANEK_EZ"

Output Parameters:

Name	Type	Description
SoundSpeed	Speed	speed of sound

Section 3.21: SSPDTemp Application

This module calculates average temperature to use as input to soundspeed computation in a simplified manner. Since SSPD is needed prior to determining aircraft noise geometry, it is impossible to know the time of PNLTM, of the aircraft height at the emission point for the noise measured at that time. Additionally, there is not a large sensitivity of soundspeed to temperature – about 1 FPS for each degree Fahrenheit over the range of permissible temperatures for aircraft noise certification (-14F to 95F) - which does not vary rapidly over time or with increasing heights typical for certification in relationship to its effect on soundspeed. Therefore several approximations are made in determining the input temperature to use for determination of event soundspeed: First, it is not necessary to interpolate temperature measurements at a particular height over time, since temperature measurements must occur in close proximity to aircraft noise events – at least one of the preceding or succeeding met temperature measurements must be within 30 minutes of the aircraft noise event. Because of this, it is considered to be sufficient (by Dave Read) to simply average the temperature measurement values at a particular height for the closest-in-time meteorological measurements prior to and subsequent to the aircraft noise event. Second, while the official guidance documents for noise certification (ICAO ETM and FAA AC36-4) specify that the average of temperatures at the ground and at aircraft height is to be used, it is considered to be sufficient (again by Dave Read) to use the 10M temperature to represent the temperature at the ground, and to use the temperature at the closest met measurement height (increments are limited to 100 feet or 30 meters) to that of the aircraft height at overhead, which is easily obtainable without performing any noise geometry calculations. These elements reduce the calculation of approximate event temperature to a very simple process that can be easily performed by hand or with a simple software function.

Unit Parameters:

Name	Type	Default
TemperatureUnits	Temperature Unit	"Celsius"
DistanceUnits	Distance Unit	"Feet"

Input Parameters:

Name	Type	Default	Description
mode	Integer	4	1. Preferred 2. ETOD Approx. 3. Preferred Alternate 4. Averaged approx. 5. ETOD 10m 6. ETOD 10m Alternate 7. Average 10m
preMET	METData	None	
postMET	METData	None	
etodMET	METData	None	
eTOD	HHMMSS	None	
height	Distance	None	

Output Parameters:

Name	Type	Description
ACMetHeight	Distance	
Avg10MTemp	Temperature	
AvgACMetHeightTemp	Temperature	
SSPDAvgTemp	Temperature	

Section 3.22: SimpleStats Application

This module performs the simple statistical computations on multiple event EPNLs required for aircraft noise certification (for which Guidance is provided in the ETM). It works on a single composite input file that (currently) the user must assemble from the results of running SuperFAR on a series of separate events in separate data folders. Future development may provide for automated routines to identify and select data from various types of report files from other SuperFAR modules.

Input Parameters:

Name	Type	Default	Description
StatMethod	String	"Clustered"	Accepts the following keywords: "Clustered" "Regressed"
K	PolynomialOrder	None	
data	StatsData	None	

Output Parameters:

Name	Type	Description
N	Integer	number of values
DOF	Integer	Degrees Of Freedom
EPNLsum	float	sum of EPNL values
EPNLavg	float	average EPNL value
T	float	Student's T value
SumDS	float	sum of delta squares
StdDev	float	standard deviation
CI90	float	90% confidence interval
report	StatsData	

Section 3.23: Simplified Application

SIMPLIFIED PROCEDURE for Reference Condition EPNL - Calculates the Reference Condition EPNL using the Simplified Method (See ICAO Annex 16 Vol. I, Appendix 2, Section 8.3 and ICAO ETM Vol. I, Chapter 4, Section 4.3.1.3) Assumes that test-day processing has been completed through EPNLCalc and that RefGeo has been run. Adjusts all spectra in TD STH input file to reference conditions, calculates PNLTR values and determines Delta Peak and other Simplified Deltas to be added to Test-Day EPNL to obtain Simplified EPNLR.

Unit Parameters:

Name	Type	Default
SpeedUnits	Speed Unit	"FPS"
DistanceUnits	Distance Unit	"Feet"

Input Parameters:

Name	Type	Default	Description
tdSTH	STHData	None	
tdMap	MapData	None	
tdATH	ATHData	None	A test day time-history of 1/3 octave band alphas (atmospheric absorption coefficients) corresponding to each 1/3 octave band SPL in the tdSTH. (Used by Adj2Ref function)
tdALF	ALFData	None	A single spectrum of test day alpha values, as an alternative to tdATH.
refALF	ALFData	None	
tdEPNL	EPNLData	None	
refGTH	REFGTHData	None	
tdMTX	MTXData	None	
VG	Speed	0	
VGR	Speed	0	
DEL3	dB	* dynamic	
AFlag	Boolean	True	include ANSI A-weighted metrics if true
CFlag	Boolean	True	include ANSI C-weighted metrics if true
DFlag	Boolean	True	include ANSI D-weighted metrics if true
OAFlag	Boolean	True	include unweighted metrics if true
PFlag	Boolean	True	include PNL metrics if true
TFlag	Boolean	True	include PNLT metrics if true
TCB40Flag	Boolean	False	
NoRoundFlag	Boolean	False	
HelicopterFlag	Boolean	False	
TCLowBand	Band	None	

Output Parameters:

Name	Type	Description
refSTH	STHData	
refMTX	MTXData	
refEPNL	REFEPNLData	

Section 3.24: TDMet Application

Determines cumulative and layered 1/3 octave band atmospheric absorption coefficients (alphas) using ARP866A methodology from inputs of Event Time (ETOD) and T&H vs. height profiles before and after ETOD.

Unit Parameters:

Name	Type	Default
DistanceUnits	Distance Unit	"Feet"
TemperatureUnits	Temperature Unit	"Celsius"
AlphaUnits	Alpha Unit	"dB100m"
AngleUnits	Angle Unit	"Degrees"

Input Parameters:

Name	Type	Default	Description
preMET	METData	None	T&H profile before ETOD
postMET	METData	None	T&H profile after ETOD
ETOD	HHMMSS	None	Aircraft Event Time-of-day
ETODType	String	"TOH"	Accepts the following keywords: "TOH" Time aircraft is overhead or abeam of the microphone "TCPA" Time at geometrical closest-point-of-approach, or minimum distance to microphone of interest "TMAX" Time at measurement of maximum PNLT level "OTHER" any other TOD assigned to the aircraft event
LayerCrit	Alpha	* dynamic	Layering Criterion variation window for 3150Hz band in dB100M or db1KFT
ACHeight	Distance	None	Aircraft height above ground at ETOD
MIC	MICData	None	Microphone data file providing ZMic and HMic
ZMic	Distance	* dynamic	Z offset (ie, altitude) of microphone site
HMic	Distance	* dynamic	Height of microphone reciever above site
ForceLay	Boolean	False	force Layer processing
ForceFull	Boolean	False	force full layer processing
SPFlag	Boolean	False	when layering is not required, force single-point (height) methodology

Output Parameters:

Name	Type	Description
etodMET	METData	
etodALF	METALFData	
LayrAlf	METALFData	
MaxDev	Alpha	Maximum deviation in 3150 Hz band alpha from ground layer
LayFlag	Boolean	
fullALF	ALFData	AvgAlpha computed by full layered method
simpleALF	ALFData	AvgAlpha computed by simple layered method
twoptALF	ALFData	AvgAlpha computed by two point method
oneptALF	ALFData	AvgAlpha computed by one point method
avgALF	ALFData	AvgAlpha computed by selected or default method

Section 3.25: ValidAdj Application

Decombines valid pre-detection noise levels from valid aircraft SPLs and adds measurement system corrections to valid aircraft SPLs.

Called By:

Badjer

Input Parameters:

Name	Type	Default	Description
validpreSSR	SSRData	None	
maskMap	MapData	None	
STH	STHData	None	
correctionSSR	SSRData	None	
correctionValue	dB	* dynamic	
lowRec	Integer	* dynamic	
highRec	Integer	* dynamic	

Output Parameters:

Name	Type	Description
ambisubSTH	STHData	
ambisubMap	MapData	
gainadjSTH	STHData	
badjerSTH	STHData	
badjerMap	MapData	

Section 4: Writing SuperFAR Files

All Superfar Data files can be read and written in a simple CSV ("Comma Separated Value") format. Each file format typically breaks down int three sections: annotations, followed by a one-line column header, followed by one or more rows of tabular data.

The first section of all data files contains annotations. At the most literal level, an annotation is any interface parameter that has a role option set to 'annotation'. At a more conceptual level, they are usually values associated with the file as a whole rather than any particular row of data. For example, unit parameters are annotations, as they define the units used by the file as a whole. Another example is the filename, which obviously applies to the entire file, and so forth.

Each annotation parameter is written to an output CSV file in the same order that they are are defined in the TableData subclass interface. Most annotations occupy one row of the CSV file: the first value in the row is annotation is the parameter name, with two asterisks appended. After the name comes the value of the parameter, or if the parameter has the deconstruction option, the deconstructed form of the value is appended to the row as a sequence of deconstructed values.

Some annotation parameters have a special multi-line format. All such cases begin in the same way, with a label (usually beginning with "NumberOf" and ending in two asterisks) followed by a number of output values to follow.

Parameter Name	Annotation Label	Data Values
GenerationFiles	NumberOfGenerationFiles**	two lines per generation file: the first line is a filename, the second line is a date and time.
OtherRecords	OtherRecords**	one line per 'other record' (meaning left to user)
Comments	NumberOfCommentLines**	one line per each line of comment string, split by newline character.

after all annotations are written to the output file, the header is written to the output file as a single row of column names. For header columns that are internally stored as Band objects, the band number is translated to an appropriate string formatted column name.

after the header row is written to the output file, each row of data is written to the output file in the same column order defined by the header. What "each row of data" means depends on whether or not the current TableData class is considered to be a single-row class or a multi-row class.

A TableData subclass is single row by default, unless it qualifies as multi-row. To qualify as multi-row, it must have an index parameter (where the index type is usually RowNum) and the value type stored for each key of the index type must be a subclass of DataTable, which is the type of each row of data in the multi-row container.

Thus if the container is a single-row class, "each row of data" simply refers to the container itself as one row of data. If the container is a multi-row class, "each row of data" is retrieved as the values stored in the container indexed by keys of the index type (usually RowNum), sorted in ascending order. In either case, each row of data is written to the output file as a sequence of values corresponding to the sequence of column names specified by the header. A proper row class for a multi-row container should also define a column parameter or alias named 'primary_key', which is important for parsing files (as explained in the next section).

Section 5: Reading SuperFAR Files

Reading a SuperFAR file mostly involves doing the reverse of writing the file, with some caveats and special cases. See the previous section for more context.

The first caveat about reading files is that all annotations are optional on input, and they are allowed to appear in any order. Superfar will even defer resolving the units of annotations that depend on a unit parameter, in case the unit parameter appears after the parameter that depends on it.

Annotations that have a constant or computed value, (ie, parameters that have a value option set) are parsed but ignored. The only exception is FileType, which will print a warning if the parsed FileType does not match the expected FileType.

When the header row of a data file is parsed, columns corresponding to band numbers are recognized and converted to Band objects at the point where the container object's header attribute is assigned the value of a tuple containing the string column names.

The SuperFAR file parser tests whether the class being parsed is single-row or multi-row, to decide how the remaining rows of data should be parsed. If it is a single-row class, a single row of data is parsed from the file and the column values are assigned directly to the container. If it's a multi-row class, each line of data is stored in a new instance of the row class type, and then the row is stored in the multi-row container using the 'primary_key' of the row, which should be a parameter or alias defined by the row class that determines what key value is used to store the row in a multi-row container.

Section 6: SuperFAR File Formats

This section describes individual SuperFAR file formats, as implemented by corresponding subclasses of the TableData class. See the previous two sections for a more general introduction to how SuperFAR files are read or written.

Section 6.1: ALFData Class / File Format

A spectrum of Alpha values indexed by 1/3 octave bands

Unit Parameters:

Name	Type	Default
AlphaUnits	Alpha Unit	"dB100m"

Annotations:

Name	Type	Default	Description
ProjectName	String	None	a file Annotation that equals superfar.ProjectName
GeneratedBy	list	None	a file Annotation that equals app.GeneratedBy (if app exists)
GenerationFiles	list	None	a file Annotation that equals app.GenerationFiles (if app exists)
ScriptFile	String	None	a file Annotation that equals app.ScriptFile (if app exists)
ScriptLine	Integer	None	a file Annotation that equals app.ScriptLine (if app exists)
OtherRecords	list	None	an arbitrary list of user data that may be associated with a file
Comments	String	"Comments go here."	free-form comments
AlphaUnits	Alpha Unit	"dB100m"	Unit of Alpha
MicrophoneID	String	None	
LAYERFLAG	Boolean	None	

Data Columns:

Name	Type	Description
Rec#	RecNum	
TOD	HHMMSS	

Section 6.2: ATHData Class / File Format

ATHData: Alpha Time-History - a time-history of alpha rows, each containing 1/3 octave band atmospheric absorption coefficients.

Unit Parameters:

Name	Type	Default
AlphaUnits	Alpha Unit	"dB100m"

Annotations:

Name	Type	Default	Description
ProjectName	String	None	a file Annotation that equals superfar.ProjectName
GeneratedBy	list	None	a file Annotation that equals app.GeneratedBy (if app exists)
GenerationFiles	list	None	a file Annotation that equals app.GenerationFiles (if app exists)
ScriptFile	String	None	a file Annotation that equals app.ScriptFile (if app exists)
ScriptLine	Integer	None	a file Annotation that equals app.ScriptLine (if app exists)
OtherRecords	list	None	an arbitrary list of user data that may be associated with a file
Comments	String	"Comments go here."	free-form comments
AlphaUnits	Alpha Unit	"dB100m"	Unit of Alpha
MicrophoneID	String	None	
LAYERFLAG	Boolean	None	

Section 6.3: EPNLData Class / File Format

EPNLData: Contains dB levels, record numbers and timestamps for Maximum, First & Last 10 dB-down points and Secondary Peaks (within 2 dB of max) for each metric listed in the input MTX (Metric time-history) file. Also provides "TILE": Time-Integrated LEvels (Including EPNL for PNLT) for each metric.

Annotations:

Name	Type	Default	Description
ProjectName	String	None	a file Annotation that equals superfar.ProjectName
GeneratedBy	list	None	a file Annotation that equals app.GeneratedBy (if app exists)
GenerationFiles	list	None	a file Annotation that equals app.GenerationFiles (if app exists)
ScriptFile	String	None	a file Annotation that equals app.ScriptFile (if app exists)
ScriptLine	Integer	None	a file Annotation that equals app.ScriptLine (if app exists)
OtherRecords	list	None	an arbitrary list of user data that may be associated with a file
Comments	String	"Comments go here."	free-form comments
TCLowBand	Band	None	
PNLTM	dB	None	
DeltaBndShr	dB	None	
ToneCorr(max-2)	dB	None	
ToneCorr(max-1)	dB	None	
ToneCorr(max)	dB	None	
ToneCorr (max+1)	dB	None	
ToneCorr (max+2)	dB	None	
AverageTC	dB	None	

Section 6.4: GTHData Class / File Format

GTHData: Geometry Time-History Data - history of emisson TXYZ, propagation distance (SR) and sound emission angle (THETA)

Unit Parameters:

Name	Type	Default
AngleUnits	Angle Unit	"Degrees"
DistanceUnits	Distance Unit	"Feet"
TemperatureUnits	Temperature Unit	"Celsius"
SpeedUnits	Speed Unit	"FPS"

Annotations:

Name	Type	Default	Description
ProjectName	String	None	a file Annotation that equals superfar.ProjectName
GeneratedBy	list	None	a file Annotation that equals app.GeneratedBy (if app exists)
GenerationFiles	list	None	a file Annotation that equals app.GenerationFiles (if app exists)
ScriptFile	String	None	a file Annotation that equals app.ScriptFile (if app exists)
ScriptLine	Integer	None	a file Annotation that equals app.ScriptLine (if app exists)
OtherRecords	list	None	an arbitrary list of user data that may be associated with a file
Comments	String	"Comments go here."	free-form comments
AngleUnits	Angle Unit	"Degrees"	Unit of Angle
DistanceUnits	Distance Unit	"Feet"	Unit of Distance
TemperatureUnits	Temperature Unit	"Celsius"	Unit of Temperature
SpeedUnits	Speed Unit	"FPS"	Unit of Speed
MicrophoneID	String	None	
Microphone(x y z h)	XYZH	None	
Microphone Horizontal Angle	Angle	None	
Microphone Vertical Angle	Angle	None	
SoundSpeed	Speed	None	
PromptTemperature	Temperature	None	

Section 6.5: METALFData Class / File Format

METALF: Meteorological measurement height alphas

Unit Parameters:

Name	Type	Default
AlphaUnits	Alpha Unit	"dB100m"
DistanceUnits	Distance Unit	"Feet"

Annotations:

Name	Type	Default	Description
ProjectName	String	None	a file Annotation that equals superfar.ProjectName
GeneratedBy	list	None	a file Annotation that equals app.GeneratedBy (if app exists)
GenerationFiles	list	None	a file Annotation that equals app.GenerationFiles (if app exists)
ScriptFile	String	None	a file Annotation that equals app.ScriptFile (if app exists)
ScriptLine	Integer	None	a file Annotation that equals app.ScriptLine (if app exists)
OtherRecords	list	None	an arbitrary list of user data that may be associated with a file
Comments	String	"Comments go here."	free-form comments
AlphaUnits	Alpha Unit	"dB100m"	Unit of Alpha
DistanceUnits	Distance Unit	"Feet"	Unit of Distance
ETOD	HHMMSS	None	Aircraft Event Time-of-day
ETODType	String	"TOH"	Accepts the following keywords: "TOH" Time aircraft is overhead or abeam of the microphone "TCPA" Time at geometrical closest-point-of-approach, or minimum distance to microphone of interest "TMAX" Time at measurement of maximum PNLT level "OTHER" any other TOD assigned to the aircraft event

Section 6.6: METData Class / File Format

METData: Meteorological data (Temp and RH%) vs. height and time

Unit Parameters:

Name	Type	Default
DistanceUnits	Distance Unit	"Feet"
TemperatureUnits	Temperature Unit	"Celsius"

Annotations:

Name	Type	Default	Description
ProjectName	String	None	a file Annotation that equals superfar.ProjectName
GeneratedBy	list	None	a file Annotation that equals app.GeneratedBy (if app exists)
GenerationFiles	list	None	a file Annotation that equals app.GenerationFiles (if app exists)
ScriptFile	String	None	a file Annotation that equals app.ScriptFile (if app exists)
ScriptLine	Integer	None	a file Annotation that equals app.ScriptLine (if app exists)
OtherRecords	list	None	an arbitrary list of user data that may be associated with a file
Comments	String	"Comments go here."	free-form comments
DistanceUnits	Distance Unit	"Feet"	Unit of Distance
TemperatureUnits	Temperature Unit	"Celsius"	Unit of Temperature
ETOD	HHMMSS	None	Aircraft Event Time-of-day
ETODType	String	"TOH"	Accepts the following keywords: "TOH" Time aircraft is overhead or abeam of the microphone "TCPA" Time at geometrical closest-point-of-approach, or minimum distance to microphone of interest "TMAX" Time at measurement of maximum PNLT level "OTHER" any other TOD assigned to the aircraft event

Section 6.7: MICData Class / File Format

MICData: Microphone site XYZ coordinates, Microphone height and angles

Unit Parameters:

Name	Type	Default
AngleUnits	Angle Unit	"Degrees"
DistanceUnits	Distance Unit	"Feet"

Annotations:

Name	Type	Default	Description
ProjectName	String	None	a file Annotation that equals superfar.ProjectName
GeneratedBy	list	None	a file Annotation that equals app.GeneratedBy (if app exists)
GenerationFiles	list	None	a file Annotation that equals app.GenerationFiles (if app exists)
ScriptFile	String	None	a file Annotation that equals app.ScriptFile (if app exists)
ScriptLine	Integer	None	a file Annotation that equals app.ScriptLine (if app exists)
OtherRecords	list	None	an arbitrary list of user data that may be associated with a file
Comments	String	"Comments go here."	free-form comments
AngleUnits	Angle Unit	"Degrees"	Unit of Angle
DistanceUnits	Distance Unit	"Feet"	Unit of Distance
MicrophoneID	String	None	

Data Columns:

Name	Type	Description
X	Distance	
Y	Distance	
Z	Distance	
Height	Distance	
MVertAng	Angle	
MHorzAng	Angle	

Section 6.8: MTXData Class / File Format

#MTXData: Metrics Time-History

Annotations:

Name	Type	Default	Description
ProjectName	String	None	a file Annotation that equals superfar.ProjectName
GeneratedBy	list	None	a file Annotation that equals app.GeneratedBy (if app exists)
GenerationFiles	list	None	a file Annotation that equals app.GenerationFiles (if app exists)
ScriptFile	String	None	a file Annotation that equals app.ScriptFile (if app exists)
ScriptLine	Integer	None	a file Annotation that equals app.ScriptLine (if app exists)
OtherRecords	list	None	an arbitrary list of user data that may be associated with a file
Comments	String	"Comments go here."	free-form comments
MicrophoneID	String	None	
AveragingMethod	String	None	
TimeStampType	String	None	
AdjustmentCode	String	None	
StartTime	HHMMSS	None	
ReferenceTime	HHMMSS	None	
ReferenceTimeType	String	None	
TCLowBand	Band	None	
NoRndFlag	Boolean	None	
TCB40Flag	Boolean	None	
HeliTCFlag	Boolean	None	
PNLTM + Delta Bandshare	dB	None	
PNLTMR + Delta Bandshare	dB	None	
PNLTMR + Delta BandshareR	dB	None	
Delta Bandshare	dB	None	
PNLTM Recnum	Integer	None	
Bands	Bands	None	

Section 6.9: MapData Class / File Format

MAPData: Masking and adjustment code map

Annotations:

Name	Type	Default	Description
ProjectName	String	None	a file Annotation that equals superfar.ProjectName
GeneratedBy	list	None	a file Annotation that equals app.GeneratedBy (if app exists)
GenerationFiles	list	None	a file Annotation that equals app.GenerationFiles (if app exists)
ScriptFile	String	None	a file Annotation that equals app.ScriptFile (if app exists)
ScriptLine	Integer	None	a file Annotation that equals app.ScriptLine (if app exists)
OtherRecords	list	None	an arbitrary list of user data that may be associated with a file
Comments	String	"Comments go here."	free-form comments

Section 6.10: PTHData Class / File Format

PTHData: Position Time-History - measured aircraft position vs. time

Unit Parameters:

Name	Type	Default
DistanceUnits	Distance Unit	"Feet"
SpeedUnits	Speed Unit	"FPS"
AngleUnits	Angle Unit	"Degrees"

Annotations:

Name	Type	Default	Description
ProjectName	String	None	a file Annotation that equals superfar.ProjectName
GeneratedBy	list	None	a file Annotation that equals app.GeneratedBy (if app exists)
GenerationFiles	list	None	a file Annotation that equals app.GenerationFiles (if app exists)
ScriptFile	String	None	a file Annotation that equals app.ScriptFile (if app exists)
ScriptLine	Integer	None	a file Annotation that equals app.ScriptLine (if app exists)
OtherRecords	list	None	an arbitrary list of user data that may be associated with a file
Comments	String	"Comments go here."	free-form comments
DistanceUnits	Distance Unit	"Feet"	Unit of Distance
SpeedUnits	Speed Unit	"FPS"	Unit of Speed
AngleUnits	Angle Unit	"Degrees"	Unit of Angle
Start Time	HHMMSS	None	
Overhead Time	HHMMSS	None	
End Time	HHMMSS	None	
Position Time Interval	Seconds	None	
Overhead Altitude	Distance	None	
Lateral Y Offset	Distance	None	
Ground Speed	Speed	None	
Climb/Descent Angle	Angle	None	
Lateral Cross Track Angle	Angle	None	

Section 6.11: REFEPNLData Class / File Format

REFEPNLData: Report format for Simplified Reference EPNL, Reference PNLTM spectrum SPLRs, and Simplified Deltas

Annotations:

Name	Type	Default	Description
ProjectName	String	None	a file Annotation that equals superfar.ProjectName
GeneratedBy	list	None	a file Annotation that equals app.GeneratedBy (if app exists)
GenerationFiles	list	None	a file Annotation that equals app.GenerationFiles (if app exists)
ScriptFile	String	None	a file Annotation that equals app.ScriptFile (if app exists)
ScriptLine	Integer	None	a file Annotation that equals app.ScriptLine (if app exists)
OtherRecords	list	None	an arbitrary list of user data that may be associated with a file
Comments	String	"Comments go here."	free-form comments
EPNLRSimp	float	None	
PNLTMRSimp	float	None	
DEL1	dB	None	
DELPeak	dB	None	
DEL2	dB	None	
DEL2D	dB	None	
DEL2S	dB	None	
DEL3	dB	None	
Max2Peak	dB	None	
Max2PeakK	Integer	None	

Section 6.12: REFGTHData Class / File Format

REFGTHData: Reference Condition Geometry Time-History

Unit Parameters:

Name	Type	Default
DistanceUnits	Distance Unit	"Feet"
AngleUnits	Angle Unit	"Degrees"
SpeedUnits	Speed Unit	"FPS"

Annotations:

Name	Type	Default	Description
ProjectName	String	None	a file Annotation that equals superfar.ProjectName
GeneratedBy	list	None	a file Annotation that equals app.GeneratedBy (if app exists)
GenerationFiles	list	None	a file Annotation that equals app.GenerationFiles (if app exists)
ScriptFile	String	None	a file Annotation that equals app.ScriptFile (if app exists)
ScriptLine	Integer	None	a file Annotation that equals app.ScriptLine (if app exists)
OtherRecords	list	None	an arbitrary list of user data that may be associated with a file
Comments	String	"Comments go here."	free-form comments
DistanceUnits	Distance Unit	"Feet"	Unit of Distance
AngleUnits	Angle Unit	"Degrees"	Unit of Angle
SpeedUnits	Speed Unit	"FPS"	Unit of Speed
Test TOH (Time at OverHead)	HHMMSS	None	
Reference TOH (Time at OverHead)	HHMMSS	None	
Test ZOH (Height above ground at OverHead)	Distance	None	
Ref. ZOH (Ref. Height above ground at OverHead)	Distance	None	
Average Test Groundspeed	Speed	None	
Reference Groundspeed	Speed	None	
Average test climb/descent angle	Angle	None	
Ref. climb/descent angle	Angle	None	
Test soundspeed	Speed	None	
Ref. soundspeed	Speed	None	
Test microphone X Y Z coordinates	XYZ	None	
Test microphone height above local ground	Distance	None	
Ref. microphone Y coordinate	Distance	None	
Ref. microphone height above local ground	Distance	None	
Test flight path CPA (Closest Point of Approach)	Distance	None	
Ref. flight path CPAR (Closest Point of Approach)	Distance	None	
Ref flight path CPAOHR (Closest Point of Approach for centerline location)	Distance	None	

Section 6.13: SPOData Class / File Format

A data file for output of Single Point Track definition containing only Annotation values

Unit Parameters:

Name	Type	Default
DistanceUnits	Distance Unit	"Feet"
AngleUnits	Angle Unit	"Degrees"
SpeedUnits	Speed Unit	"FPS"

Annotations:

Name	Type	Default	Description
ProjectName	String	None	a file Annotation that equals superfar.ProjectName
GeneratedBy	list	None	a file Annotation that equals app.GeneratedBy (if app exists)
GenerationFiles	list	None	a file Annotation that equals app.GenerationFiles (if app exists)
ScriptFile	String	None	a file Annotation that equals app.ScriptFile (if app exists)
ScriptLine	Integer	None	a file Annotation that equals app.ScriptLine (if app exists)
OtherRecords	list	None	an arbitrary list of user data that may be associated with a file
Comments	String	"Comments go here."	free-form comments
DistanceUnits	Distance Unit	"Feet"	Unit of Distance
AngleUnits	Angle Unit	"Degrees"	Unit of Angle
SpeedUnits	Speed Unit	"FPS"	Unit of Speed
TOH	HHMMSS	None	
XOH	Distance	None	
YOH	Distance	None	
ZOH	Distance	None	
MicX	Distance	None	
MicY	Distance	None	
MicZ	Distance	None	
MicHeight	Distance	None	
SPCPA	Distance	None	
THCPA	Distance	None	
AvStartTime	HHMMSS	None	
AvEndTime	HHMMSS	None	
SPHI	Angle	None	
SPCY	Angle	None	
SPGSPD	Speed	None	
SPVSPD	Speed	None	
AvStart	Integer	None	
AvEnd	Integer	None	

Section 6.14: SSRData Class / File Format

A spectrum of dB values indexed by 1/3 octave bands. This class also implements PNL, OASPL, AWT, and DWT as virtual parameters. Code and constants for PNL and frequency-weighted metrics are implemented here.

Annotations:

Name	Type	Default	Description
ProjectName	String	None	a file Annotation that equals superfar.ProjectName
GeneratedBy	list	None	a file Annotation that equals app.GeneratedBy (if app exists)
GenerationFiles	list	None	a file Annotation that equals app.GenerationFiles (if app exists)
ScriptFile	String	None	a file Annotation that equals app.ScriptFile (if app exists)
ScriptLine	Integer	None	a file Annotation that equals app.ScriptLine (if app exists)
OtherRecords	list	None	an arbitrary list of user data that may be associated with a file
Comments	String	"Comments go here."	free-form comments

Data Columns:

Name	Type	Description
Rec#	RecNum	
TmTOD	HHMMSS	
TOD	HHMMSS	
RelTime	Seconds	
PNL	float	a virtual parameter that computes the Perceived Noise Level of the spectrum
OASPL	float	a virtual parameter that computes the Overall Sound Pressure Level of the spectrum
AWT	float	a virtual parameter that computes the A-Weighted sound level of the spectrum
CWT	float	a virtual parameter that computes the C-Weighted sound level of the spectrum
DWT	float	a virtual parameter that computes the D-Weighted sound level of the spectrum

Section 6.15: STHData Class / File Format

A Spectral Time-History table of 1/3 octave band SPL records indexed by Rec#

Annotations:

Name	Type	Default	Description
ProjectName	String	None	a file Annotation that equals superfar.ProjectName
GeneratedBy	list	None	a file Annotation that equals app.GeneratedBy (if app exists)
GenerationFiles	list	None	a file Annotation that equals app.GenerationFiles (if app exists)
ScriptFile	String	None	a file Annotation that equals app.ScriptFile (if app exists)
ScriptLine	Integer	None	a file Annotation that equals app.ScriptLine (if app exists)
OtherRecords	list	None	an arbitrary list of user data that may be associated with a file
Comments	String	"Comments go here."	free-form comments
MicrophoneID	String	None	
AveragingMethod	String	None	
TimeStampType	String	None	
AdjustmentCode	String	None	
StartTime	HHMMSS	None	
ReferenceTime	HHMMSS	None	
ReferenceTimeType	String	None	

Section 6.16: StatsData Class / File Format

Output from SimpleStats Module

Annotations:

Name	Type	Default	Description
ProjectName	String	None	a file Annotation that equals superfar.ProjectName
GeneratedBy	list	None	a file Annotation that equals app.GeneratedBy (if app exists)
GenerationFiles	list	None	a file Annotation that equals app.GenerationFiles (if app exists)
ScriptFile	String	None	a file Annotation that equals app.ScriptFile (if app exists)
ScriptLine	Integer	None	a file Annotation that equals app.ScriptLine (if app exists)
OtherRecords	list	None	an arbitrary list of user data that may be associated with a file
Comments	String	"Comments go here."	free-form comments
Average	float	None	
StdDev	float	None	
90% Confidence Interval	float	None	
Number of Values	Integer	None	
Degrees of Freedom	Integer	None	
Student's T	float	None	
Sum of Deltas Squared	float	None	
Data Set Type	String	"Clustered"	Accepts the following keywords: "Clustered" "Regressed"
K	PolynomialOrder	None	

Appendix A1: HTML Documentation Template Engine

This documentation is automatically generated by a python script named autodoc.py, with the help of a SuperFAR class named HTMLDOC. This class implements an html documentation template engine for writing html doctumentation. It is initialized with a series of template parameters, as follows:

```
doc = HTMLDOC (
    FILENAME            = "ZIPME/$(TITLE).html",
    TITLE               = "SuperFAR Version $(VERSION) Documentation",
    VERSION             = superfar.version,
    DATA_SECTION        = '1',
    CONTAINER_SECTION   = '$(DATA_SECTION).1',
    BAND_SECTION        = '$(DATA_SECTION).2',
    UNIT_SECTION        = '$(DATA_SECTION).3',
    APP_SECTION         = '2',
    FILE_SECTION        = '3',
    TEMPLATE_SECTION    = 'A1',
)
```

Note that several parameters reference the name of another parameter, enclosed in parenthesis and prefixed with a dollar sign. These are called template expressions, and the template engine replaces them with the value of the referenced variable as needed. Thus the title of the document will include the current SuperFAR version number, and the filename includes the title of the document, and so on.

the rest of the autodoc.py script is taken up by a single invocation of the doc object's write method, on a multi-line string that contains the template used to generate the the final html documentation, including some more advanced examples of template expressions, such as this:

```
$(FILE_CLASSES) {
    <div class="section">
        <a name="$(NAME)"></a>
        <h2>Section $(FILE_SECTION).$(NUMBER): $(NAME) File Format</h2>
        <div class="indent">
            $(DESCRIPTION)
            $(UNITS?) {
                <h3>Unit Annotations:</h3>
                <div class="indent">
                    $(UNITS)
                </div>
            }
            $(ANNOTATIONS?) {
                <h3>Other Annotations:</h3>
                <div class="indent">
                    $(ANNOTATIONS)
                </div>
            }
            $(COLUMNS?) {
                <h3>Data Columns:</h3>
                <div class="indent">
                    $(COLUMNS)
                </div>
            }
        </div>
    </div>
}
```

When template expressions are followed by text enclosed by a matching pair of curly braces, that is considered part of the template expression, and the enclosed text is called a "sub-template". Each sub-template may contain other template expressions, which may in turn have their own nested sub-templates.

In most cases, if a template expression has a sub-template, the value of the variable is presumed to be a list of python dictionaries, and the sub-template is applied to each dictionary with the key-value pairs of the dictionary being treated as template variables, within the scope of the sub-template. For example, the FILE_CLASSES template parameter contains a list of dictionaries that each describe one file class, and each dictionary has a NAME key that can be referenced in the sub-template.

As a special exception to the previous general rule, if a template expression has a question mark after the variable name, then the expression represents a test of whether the named variable is empty. To be more precise, the value is presumed to be a list or some other list-like container object, and the test is literally whether len(val) > 0. If the test returns true, the sub-template is included in the final result; if not, it is excluded. This is the only flow control costruct in the template engine, but it is sufficient for the needs of the project so far.

FILE_CLASSES is a built-in variable proveded by the HTMLDOC object, and does not have to be initialized by the user of the library. Here is a full list of built in template variables provided:

- APP_CLASSES - a list of dictionaries containing the following key-value pairs to describe an application class:
  - NAME - the name of the application class
  - NUMBER - the number of the application class, as it appears in a list sorted in alphabetical order, starting at 1
  - DESCRIPTION - the description of the application class
  - CALLS - a list of dictionaries, one per app called, each with one key named 'LINK' containing the name of the called app
  - CALLED_BY - a list of dictionaries, one per app calling this app, each with one key named 'LINK' containing the name of the app
  - UNITS - an html table object describing the unit parameters of this application
  - INPUTS - an html table object describing the input parameters of this app
  - OUTPUTS - an html table object describing the output parameters of this app
- FILE_CLASSES - a list of dictionaries containing the following key-value pairs to describe a file class:
  - NAME - the name of the file class
  - NUMBER - the number of the file class, as it appears in a list sorted in alphabetical order, starting at 1
  - DESCRIPTION - the description of the file class
  - UNITS - an html table object describing the unit parameters of the file format
  - ANNOTATIONS - an html table object describing file annotation/parameters
  - COLUMNS - an html table object describing file data columns
- UNIT_TABLES - a list of dictionaries containing the following key-value pairs to describe a general type of unit:
  - NAME - the name of the general type of unit (Distance, Time, etc...)
  - NUMBER - the number of the general type of unit, as it appears in a list sorted in alphabetical order, starting at 1
  - ANCHORS - a string containging html link anchors, so that individual unit names such as "Feet" will link to their general unit type entry
  - TABLE - a html table object describing the specific units available for this general unit type.

Appendix A2: Adding New Applications to SuperFAR

To illustrate how to add a new Application to SuperFAR, let us examine a real world example of ALF2ATH.

To make ALF2ATH available to a user of the superfar package, it must be defined or imported by the __init__.py script in the superfar directory. This is the script that is imported when a user begins a SuperFAR script with the statement "from superfar import *". Here is a small exerpt of __init__.py where ALF2ATH is imported:

```
...
from .Adj2Ref           import Adj2Ref
from .ALF2ATH           import ALF2ATH
from .ARP866A           import ARP866A
...
```

The statement "from .ALF2ATH import ALF2ATH" tells python to look for a module named ALF2ATH.py in the current package (ie, in the same directory as __init__.py), and imports the symbol ALF2ATH from it. By convention, all SuperFAR applications are defined in a module file of the same name as the application, and imported in __init__.py in this manner. Each application module can define any number of classes, variables, or other symbols in it's local scope as needed to implement the desired functionality of the application, but only the application is made visible to the user by the import statement in the __init__.py module. This keeps the name-space imported by the statement "from superfar import *" clean and well-organized, containing only the named symbols meant to be visible to the user.

So with that in mind, here are the contents of ALF2ATH.py module that implement the ALF2ATH application class:

```
from .Applications import *
from .ALFData import ALFData
from .STHData import STHData
from .ATHData import ATHData

class ALF2ATH (Application):

    description = """
        generates ATH (Alpha Time-History) data from single ALF spectrum for each record in STH
    """
    interface = ContainerSchema ([
        stdapp,
        UnitParameter("AlphaUnits"),

        Input('ALF',
              type = ALFData,
              comment = "input alpha spectrum, providing alpha data"),
        Input('STH',
              type = STHData,
              comment = "input spectrum time history, providing time history"),
        Output('ATH',
              type = ATHData,
              comment = "output alpha time history"),
    ])

    def invoke(app):

        ALF = app.ALF
        STH = app.STH
        ATH = app.ATH

        bands = ALF.index_keys()
        ATH.header = ['rec#', 'TODHH', 'TODMM', 'TODSS'] + bands

        for rec in STH.index_keys():
            STHrec = STH[rec]
            ATHrec = ATH[rec]
            ATHrec.TOD = STHrec.TOD
            for band in bands:
                ATHrec[band] = ALF[band]
```

The first step of writing a new module for SuperFAR is importing other symbols we need from the superfar package, from inside of the package rather than outside of it. We cannot begin a SuperFAR module with the statement "from superfar import *": the closest equivalent statement using the correct relative syntax would be "from . import *". However, even this has a subtly hidden pitfall: "from . import *" would try to import the __init__.py module, but if __init__.py imports anything the module we are currently implementing, we create a circular dependancy loop. How can we finish importing __init__.py at the beginning of ALF2ATH.py, and yet also finish importing ALF2ATH.py before we finish importing __init__.py?

To get around this chicken-and-egg problem, we instead begin ALF2ATH.py with "from .Applications import *", which includes most of the core features of SuperFAR minus individual applications and TableData classes, and then explicitly import any applications or TableData classes we need: in the case of ALF2ATH, that would be ALFData, STHData, and ATHData.

With import statements taken care of, we can begin our class definition of the ALF2ATH class, as a subclass of the Application class. Every sublcass of Application is expected to have at least three attributes: a description, an interface, and an invoke method.

the description of an application is simply a string that will be included in the auto-generated documentation for SuperFAR. It can include HTML markup.

the interface of an application should be an instance of the ContainerSchema class that defines the names and types of this application's paremeters. the constructor of the ContainerSchema class accepts one parameter, that is a list of values used to construct the ContainerSchema.

By convention, the first value used to construct the ContainerSchema of a SuperFAR application shoulod be the symbol stdapp, which is a ContainerSchema object defining the standard parameters shared by all SuperFAR applications.

After stdapp, a typical SuperFAR application ContainerSchema should include any relevant unit parameters, by constructing instances of the UnitParameter class, which is a subclass of the abstract Parameter class. Like most subclasses of the Parameter classes, UnitParameter expects the name of the desired parameter to be provided as the first argument to the constructor: UnitParameter is simpler than some other Parameter types by not requiring any other constructor arguments.

Next, a typical SuperFAR application defines all of it's input and output parameters, by constructing instances of the Input and Output classes. Input and Output are also sublcasses of the Parameter class, and follows the convention of expecting the name of the parameter as the first argument to the constructor. Unlike UnitParameter, Input and Output also requires a type argument, and a comment argument (technically the comment is optional, but it's strongly recommended).

Appendix A3: Adding New File Formats to SuperFAR

File formats supported by SuperFAR are read or written by instances of the TableData class, which serves as a container for data that is read from a file or destined to be written to one. To illustrate how to add a new TableData class to SuperFAR, and thus a new file format, let us examine a real world example of MTXData.

Just as in the case of an application, to make a TableData class like MTXData available to a user of the superfar package, it must be defined or imported by the __init__.py script in the superfar directory. This is the script that is imported when a user begins a SuperFAR script with the statement "from superfar import *". Here is a small exerpt of __init__.py where MTXData is imported:

```
...
from .METData       import METData
from .MTXData       import MTXData
from .MapData       import MapData
...
```

The statement "from .MTXData import MTXData" tells python to look for a module named MTXData.py in the current package (ie, in the same directory as __init__.py), and imports the symbol MTXData from it. By convention, all SuperFAR TableData classes are defined in a module file of the same name as the TableData class, and imported in __init__.py in this manner. Each of these modules can define any number of classes, variables, or other symbols in it's local scope as needed to implement the desired file format, but only MTXData is made visible to the user by the import statement in the __init__.py module. This keeps the name-space imported by the statement "from superfar import *" clean and well-organized, containing only the named symbols meant to be visible to the user.

So with that in mind, here are the contents of MTXData.py module that implement the MTXData file format:

```
from .DataStructures import *

epnl_row_keywords = ['', 'MAX', '2ND', 'F10', 'L10']

class MetricRowData (TableData):
    interface = ContainerSchema ([
        stdfile,
        Column('Rec#',      type = RecNum),
        Column('TOD',       type = HHMMSS),
        Column('RelTime',   type = Seconds),
        Column('EFFINT',    type = Seconds),
        Column('AWT',       type = dB),
        Column('AINT',      type = str, keywords = epnl_row_keywords),
        Column('CWT',       type = dB),
        Column('CINT',      type = str, keywords = epnl_row_keywords),
        Column('DWT',       type = dB),
        Column('DINT',      type = str, keywords = epnl_row_keywords),
        Column('OASPL',     type = dB),
        Column('OINT',      type = str, keywords = epnl_row_keywords),
        Column('PNL',       type = dB),
        Column('PINT',      type = str, keywords = epnl_row_keywords),
        Column('PNLT',      type = dB),
        Column('TINT',      type = str, keywords = epnl_row_keywords),
        Column('TONECOR',   type = dB),
        Column('TONEBND',   type = Band),
        Column('LGB',       type = int),
        Column('NVR',       type = str),

        Alias('primary_key', 'Rec#'),
    ])


class MTXData (TableData):
    interface = ContainerSchema ([
        FileType('Metrics Time-History'),
        Extension('.mtx.csv'),
        stdfile,

        Annotation('MicrophoneID', type = str),
```

```
        Annotation('AveragingMethod', type = str),
        Annotation('TimeStampType', type = str),
        Annotation('AdjustmentCode', type = str),
        Annotation('StartTime', type = HHMMSS),
        Annotation('ReferenceTime', type = HHMMSS),
        Annotation('ReferenceTimeType', type = str),
        Annotation('TC1kFlag', type = bool),
        Annotation('NoRndFlag', type = bool),
        Annotation('TCB40Flag', type = bool),
        Annotation('HeliTCFlag', type = bool),
        Annotation('PNLTM + Delta Bandshare', type = dB),
        Annotation('PNLTMR + Delta Bandshare', type = dB),
        Annotation('PNLTMR + Delta BandshareR', type = dB),
        Annotation('Delta Bandshare', type = dB),
        Annotation('PNLTM Recnum', type = int),
        Annotation('Bands', type = Bands),

        IndexedParameter(index = RecNum, type = MetricRowData),
    ])
```

As explained in the section on applications, the first step of writing a new module for superfar is importing other symbols we need from the superfar package, from inside of the package rather than outside of it. We cannot begin a superfar module with the statement "from superfar import *": the closest equivalent statement using the correct relative syntax would be "from . import *". However, even this has a subtly hidden pitfall: "from . import *" would try to import the __init__.py module, but if __init__.py imports anything the module we are currently implementing, we create a circular dependancy loop. How can we finish importing __init__.py at the beginning of MTXData.py, and yet also finish importing MTXData.py before we finish importing __init__.py?

To get around this chicken-and-egg problem, we instead begin MTXData.py with "from .DataStructures import *", which includes most of the core features of SuperFAR needed to implement a TableData class. This is sufficient for the purpose of MTXData, but if other symbols are needed they could be imported as needed from the modules that define them on a case by case basis.

With import statements taken care of, we can begin defining the TableData classes that represent the MTXData file format. There are actually two classes that do this: the MetricRowData class represents a single row of data, and the MTXData class (the actual class visible to the user) which is a multi-row data format.

the first element of the MetricRowData container class is stdfile. Just as stdapp is a ContainerSchema defining the standard parameters common to all applications, stdfile is a ContainerSchema defining the standard parameters common to all TableData classes.

The bulk of the remaining parameters defined by the MetricRowData interface are Column parameters, which are in practice what makes MetricRowData behave as a "row of data", by defining what type of value is stored in the row for each column name.

Finally, the MetricRowData interface contains an Alias object, which instructs the interface to treat 'primary_key' as an alias for the 'Rec#' column. Aliases are a general-purpose mechanism for giving alternative names to any named parameter in a superfar container object, but the alias name 'primary_key' has a special meaning for TableData objects that represent a single row of data: it specifies which column is used to index rows of this type, in a multi-row context.

Parameters that contain string values can have an optional keywords property, listing the valid string values accepted by the parameter. All string columns in MetricRowData have the same set of acceptable keywords, so the list of keywords is defined in one place by the name of epnl_row_keywords.

The MTXData interface begins with a FileType and an Extension, which is necissary for TableData classes intended to be written to external csv files, in addition to the standard inclusion of stdfile.

Next, we have a series of Annotation parameters, which are named attributes associated with the data file as a whole, rather than individual rows of data. When a TableData object is read from or written to an external csv file, Annotation parameters correspond to rows of data in the csv file that appear above the main column header line.

If a data file requires Unit parameters, they behave much like Annotations for the purpose of being read or written to a file.

The final expression used to initialize the MTXData interface, "IndexedParameter(index = RecNum, type = MetricRowData)", is the key expression that makes MTXData behave as a multi-row TableData class. An IndexedParameter object with an index option set to an integer subtype (which RecNum is) declares that this superfar containar can contain values keyed by RecNum (or any integer, which is converted to RecNum). the type of these values indexed by RecNum is MetricRowData, the previously defined single-row TableData class. The fact that the type of the IndexedParameter is another TableData class is the definitive criteria that superfar uses to determine that MTXData is intended to be a multi-row TableData class.

Another use for IndexedParameter is to declare a single-row TableData class that has multiple columns that map to an integer type, usually Band. For example, here is the declaration of the ALFData class:

```
from .DataStructures import *

class ALFData(TableData):
    interface = ContainerSchema ([
        FileType('ALF'),
        Extension('.alf.csv'),
        stdfile,
        UnitParameter("AlphaUnits"),

        Annotation('MicrophoneID', type = str),
        Annotation('LAYERFLAG', type = bool),

        Column('Rec#',  type = RecNum),
        Column('TOD',   type = HHMMSS),

        IndexedParameter(index=Band, type = Alpha, default = dBm(0)),

        Alias('primary_key', 'Rec#'),

        Alias('AlphaType', 'AlphaUnits'),
    ])
```

in this example, the "index=Band" option says that integer index values should be treated as band numbers, and the type = Alpha which is a simple Quantity type, not a subclass of TableData. thus ALFData represents a single row of data, with some columns indexed by band number.