Vesper

Release 0.4.14

May 11, 2023

Contents

Vesper is open source software for acoustic monitoring of nocturnal bird migration. Vesper aims to enable a wide range of users, including both bird enthusiasts and professional ornithologists, to collaboratively collect, view, and analyze nocturnal flight calls (NFCs) of migratory birds at a wide range of project scales.

Vesper has been used for several years by individuals and teams to operate acoustic monitoring networks ranging in size from a single recording station to more than 50 stations. In the latter case, a team of researchers at MPG Ranch used Vesper to collect and analyze several tens of thousands of hours of audio during each of two consecutive fall migration seasons. We intend that Vesper will eventually be able to support much larger projects, involving networks comprising hundreds or even thousands of monitoring stations operated by large teams of people. Even as the size of projects that Vesper can support grows, however, we will continue to improve its utility for smaller projects, including single-station ones.

Vesper is still very much a work in progress, and as such is not yet as full-featured and easy to use as it will be eventually. Nevertheless, experience has proved its utility for a range of users, and so we provide this documentation to help additional users adopt Vesper for their own monitoring efforts. We hope that Vesper will prove valuable to you, and welcome feedback that will help us improve it.

This documentation currently comprises installation instructions and a beginner's tutorial. Additional documentation is under development.

CHAPTER 1

Contents

1.1 Installation

1.1.1 Installing Miniconda or Anaconda

The Vesper installation instructions assume that you will use Vesper in conjunction with either the Miniconda or the Anaconda Python distribution. We recommend Miniconda and Anaconda due to their excellent package and environment management functionality, which makes it relatively easy to install and maintain any number of Vesper versions and their dependencies. You can read more about that functionality in the *Conda environments* section below.

Miniconda and Anaconda are both free and open source, and are offered by Anaconda, Inc., a software company that specializes in scientific applications of the Python programming language. Anaconda includes many packages, some of which Vesper uses but most of which it doesn't. Miniconda is much smaller than Anaconda, including only a minimal set of packages. Either Miniconda or Anaconda is fine for use with Vesper.

To install Miniconda, visit the Miniconda home page and follow the instructions there for your platform. To install Anaconda, visit the Anaconda home page and follow the instructions there.

1.1.2 Installing Vesper

Important: We strongly recommend installing each version of Vesper that you use into its own Conda environment (see the *Conda environments* section below for an introduction to Conda environments). This has several advantages, including allowing you to easily revert to a previously installed Vesper version if you encounter problems with a new one.

To install the most recent version of Vesper in a new Conda environment:

1. If you don't have either the Miniconda or the Anaconda Python distribution already, download and install one of them.
2. Open a Windows Anaconda Prompt or Unix terminal.

3. Create a new Conda environment for Vesper and install a Python interpreter in it by issuing the command:

```
conda create -n vesper-0.4.14 python=3.10
```

   Conda will display a list of packages that it proposes to install, including Python and some others. Press the `Return` key to accept.

4. Activate the environment you just created with:

```
conda activate vesper-0.4.14
```

5. Install Vesper and various dependencies into the environment with:

```
pip install vesper
```

   Here you must use pip rather than Conda since Vesper is distributed as a pip package. In addition to the Vesper package, pip will install several other packages on which Vesper depends, including, for example, ones for Django and NumPy.

1.1.3 Installing BirdVoxDetect (optional)

BirdVoxDetect is a nocturnal flight call detector developed by the BirdVox project. While Vesper and BirdVox are separate software development efforts, you can use BirdVoxDetect from within Vesper just like any other supported detector. For example, you can use Vesper to run BirdVoxDetect on your recordings, view the resulting clips in clip albums, annotate them with species classifications, etc.

If you would like to use BirdVoxDetect with Vesper, install BirdVoxDetect separately in its own Conda environment. The environment must have a name of the form:

> birdvoxdetect-<version number>

where `<version number>` is the number of the BirdVoxDetect version installed in the environment, for example `0.6.0`. The environment must also include the `vesper-birdvox` package.

To create a Conda environment to use, say, BirdVoxDetect version 0.6.0 with Vesper:

1. Open a Windows Anaconda Prompt or Unix terminal.
2. Create a new Conda environment and install a Python interpreter in it by issuing the command:

```
conda create -n birdvoxdetect-0.6.0 python=3.7
```

   Conda will display a list of packages that it proposes to install, including Python and some others. Press the `Return` key to accept.

4. Activate the environment you just created with:

```
conda activate birdvoxdetect-0.6.0
```

5. Install the `vesper-birdvox` and `birdvoxdetect` packages and their dependencies into the environment with:

```
pip install vesper-birdvox birdvoxdetect==0.6.0
```

To install a version of BirdVoxDetect other than 0.6.0, substitute the appropriate version number for 0.6.0 in the instructions above, and be sure to specify a Python version compatible with your BirdVoxDetect version in step 2. See the installation instructions for the specific BirdVoxDetect version you are installing for a list of compatible Python versions.

1.1.4 Conda environments

Miniconda and Anaconda both include a command line program called conda. You can use conda to manage multiple Python *environments* within your Miniconda or Anaconda installation, where each environment contains a set of software *packages*. For example, we strongly recommend installing each version of Vesper that you use in its own conda environment. Such an environment will include a Vesper package and several tens of other packages on which Vesper depends, including, for example, packages for Django, NumPy, and Python itself. Installing each version of Vesper in its own environment keeps the packages for those different versions from interfering with each other, and with other packages that you might want to install in other, non-Vesper environments.

Every Miniconda or Anaconda installation includes a default *base* environment that is created automatically on installation. We do *not* recommend installing Vesper in the base environment, but rather in its own environment, as discussed above.

Conda environments are fully documented in the Managing environments section of the conda documentation. We will describe only a few of the more common commands for managing conda environments here.

Conda environments are managed mainly using the conda command line program, which you can run from either the Windows Anaconda Prompt or a Unix terminal. The Windows Anaconda Prompt program comes with Miniconda and Anaconda, and is similar to the regular Command Prompt program, except that it is customized for use with Miniconda and Anaconda. The conda commands you type are the same on all platforms. (If you are using Linux, however, note that some shell initialization is required for the `conda activate` and `conda deactivate` commands to work. Issue the `conda init --help` command for more about this.)

To create a new conda environment, issue the command:

```
conda create -n <env> <package list>
```

where `<env>` is the name of the new environment (for example, `vesper-1.0.0`) and `<package list>` is a list of packages that you want to install. Conda will present you with a list of the Python packages it proposes to install, including the ones you listed and any other packages that they depend upon, and ask for confirmation before proceeding.

To remove an environment named `<env>`:

```
conda remove -n <env> --all
```

To see a list of your environments:

```
conda env list
```

To activate the environment named `<env>` in the current Windows Anaconda Prompt or Unix terminal, issue the command:

```
conda activate <env>
```

The name of the environment will subsequently appear at the beginning of each command prompt in the window.

If an environment is active in the current Windows Anaconda Prompt or Unix terminal, you can deactivate it with the command:

```
conda deactivate
```

1.2 Tutorial

Welcome to the Vesper tutorial! In this tutorial, you will create a new Vesper archive (don't worry if you don't know that that is: we'll explain in a moment), import an audio recording into it, and process the recording to find and classify some bird calls that are in it. The tutorial will introduce several Vesper concepts (like what an archive is) as needed, with just enough explanation for the tutorial to make sense. The concepts will be explained in more detail in other parts of the documentation.

1.2.1 Background

Before we begin the tutorial proper, this section will provide a little background about Vesper: how the Vesper application is structured, the kinds of data (audio and other) that it processes, and the kinds of processing that it can perform.

Vesper is a web application, and as such comprises two main components, a *server* and a *client*. The server provides access to a collection of data called a *Vesper archive* (or just *archive* for short) to one or more clients, and performs operations on the data on behalf of the clients. As a picture, this application architecture looks like this:

Fig. 1: The architecture of the Vesper web application.

The server typically runs on the computer that holds the archive. Each client runs in a web browser, either on the same computer as the server or on a different one. In this tutorial, we will run the server and a single client on the same computer.

A Vesper archive is a collection of audio recordings and related metadata. Each archive has its own directory on disk, called the *archive directory*. The archive directory always contains certain essential parts of an archive, and in many cases the entirety of the archive. One such part of the archive is a relational database called the *archive database*, which holds most of the metadata of the archive.

Vesper supports various operations on archive data. The operations that you will perform in this tutorial fall into four broad and common categories. These categories are illustrated in the following figure:

Fig. 2: Four common types of operations on Vesper archive data.

An *import* operation imports audio recordings and/or related metadata into an archive from an external source. For example, in this tutorial you'll exercise two different kinds of import operations, one for audio recordings and another for metadata pertaining to them.

A *view* operation creates some sort of graphical representation of data for you to view and in some cases interact with, for example a spectrogram or a chart.

A *process* operation processes data, for example by running an automatic detector or classifier, or by classifying a set of short audio clips according to a key that you type on your keyboard.

Finally, an *export* operation exports data from an archive to an external destination. For example, in this tutorial you'll export detected bird calls from your archive as audio files.

1.2.2 Getting started

Now that you're somewhat oriented to the Vesper web application and Vesper archives, let's get started with the actual tutorial! In this first part of the tutorial, you will create a new Vesper archive, add a user to it, start a Vesper server to serve the archive, and run the Vesper client in a web browser to view the archive.

Create a new Vesper archive

1. Download the Vesper archive template to your computer.
2. Unzip the downloaded file. This should create an archive directory called `Archive Template` containing several files and directories.
3. Rename the archive directory whatever you want. In the following we will assume the name `Tutorial Archive`.
4. Move the directory wherever you want. For example, you might put it on your desktop or in a different directory that you reserve just for your Vesper archives.

Add a user to the archive

1. Open a Windows Anaconda Prompt or Unix terminal and activate your Vesper conda environment with a command like:

   ```
   conda activate vesper-x.y.z
   ```

   but with "vesper-x.y.z" replaced with the name of your Vesper environment. See the Installation section of this documentation for more about installation and conda environments.

2. In your Anaconda Prompt or terminal, change the current working directory to your archive directory by issuing an appropriate `cd` command. For example, if you're on Windows and you put your archive in `C:\Users\Nora\Desktop\Tutorial Archive`, the command is:

   ```
   cd "C:\Users\Nora\Desktop\Tutorial Archive"
   ```

   The analogous command for macOS or Linux is:

   ```
   cd "/Users/Nora/Desktop/Tutorial Archive"
   ```

   Note that if the name of your archive contains spaces you must enclose the directory path in double quotes in the command.

3. Vesper keeps track of who makes what changes in an archive via the notion of a *user*. You can add any number of users to an archive, and you must log in as one of those users to be able to modify the archive. Every archive should have at least one *superuser*, a user with certain administrative privileges. Add a superuser to your archive with the command:

   ```
   vesper_admin createsuperuser
   ```

   The command will prompt you for the superuser's name, email address, and password (twice). You can skip the email address if you wish. **Do not use a password that you want to keep secret.** Communication between the Vesper client and server is currently unencrypted, so it is possible for someone eavesdropping on your client/server network traffic to see your password.

Note: In some Vesper installations (such as ones including Python 3.10.11 and Django 4.2) the `vesper_admin createsuperuser` command will create the desired superuser but not terminate. If the command seems to hang, producing no output for at least thirty seconds, type `Ctrl-C` on your keyboard (you may have to do this several times on Windows) to terminate it.

Start the Vesper server

If you're using Windows, issue the following command in an Anaconda Prompt in which you've set the current working directory to the archive directory (for example, the Anacona Prompt of the last section):

```
python -m vesper.django.manage runserver
```

Or, if you're using macOS or Linux, issue the following command in a terminal:

```
vesper_admin runserver
```

Some output from the server should appear, indicating that the server started.

View the archive

To run a Vesper client to view the archive:

1. Start the Chrome web browser. We strongly recommend using Chrome over any other web browser since Vesper is tested and used most extensively with it.
2. Go to the URL:

   localhost:8000

   This should produce a page that looks something like this:

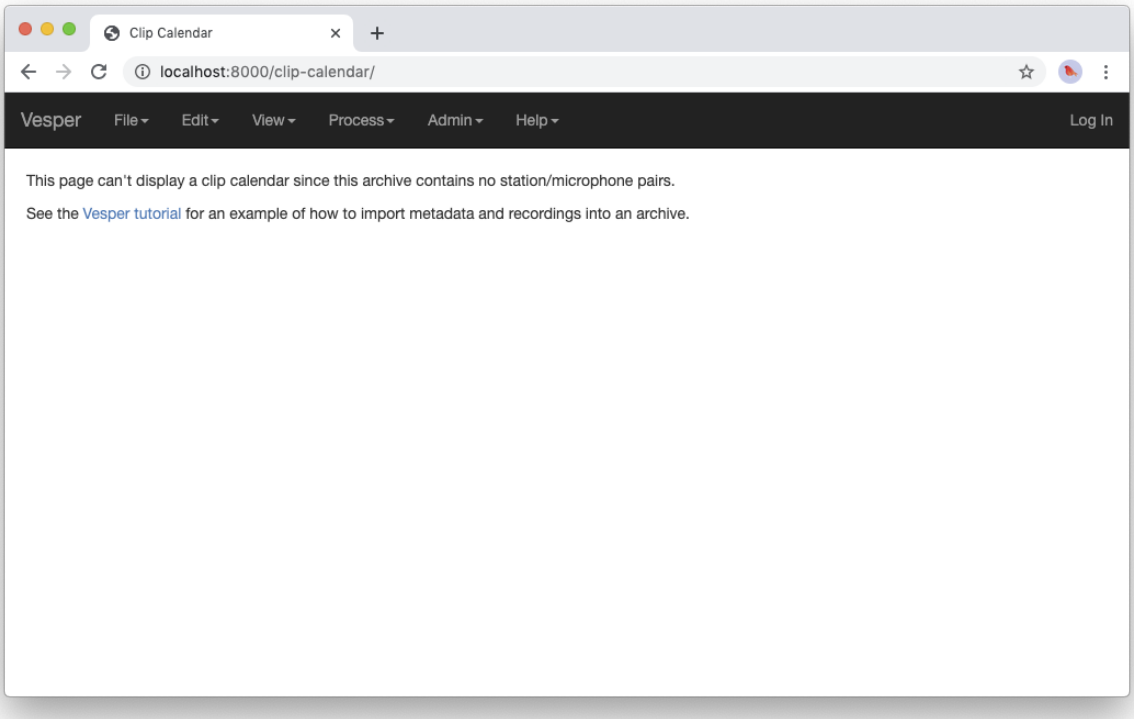

Fig. 3: An empty Vesper archive.

Congratulations: you've created, served, and viewed your very own Vesper archive! It doesn't contain any data yet, but you'll remedy that soon.

1.2.3 Archive data

As mentioned above, A Vesper archive is a collection of audio recordings and related metadata. In this section we will explain in a little more detail what that means, describing the main types of data in an archive and how they relate to each other.

Recordings, clips, annotations, and tags

First and most importantly, a *recording* is a continuous audio recording. A recording has one or more *channels*, each of which comprises a sequence of numbers called *samples*. Each sample represents an (almost) instantaneous air pressure measurement made with the aid of a microphone. In multichannel recordings (that is, recordings with more than one channel), the samples of different channels are most often recorded using different microphones.

The following figure illustrates the relationship between a recording, its channels, and their samples:

Fig. 4: A stereo recording, with two channels and their samples. The horizontal axis represents time, and each small blue box represents one sample. Note that the figure is only schematic, and that real recordings typically contain orders of magnitude more samples than shown.

The samples of a recording are collected at a fixed rate called the *sample rate*. The most common sample rates used for audio recording are in the tens of kilohertz. That is, samples are collected tens of thousands of times per second.

All of the channels of a recording have the same number of samples, and the corresponding samples of different channels (that is, corresponding in the sense that each sample has the same position in its channel's sample sequence) are collected at the same instant.

In addition to samples, a recording includes *metadata* that provide information *about* the recording (hence the "meta" prefix). For example, the metadata include a recording's start time and sample rate.

Aside from recordings, all of the other data in an archive help to describe the contents of the recordings, and hence are also metadata. In the remainder of this section, we will list and describe several types of such metadata.

A *clip* is a continuous portion of one recording channel. A clip is typically much shorter than the recording that contains it, and it typically contains a single acoustic event of interest, such as a bird call. The following figure illustrates the relationship between a recording and some clips within it:

Fig. 5: A stereo recording and some clips within it. The horizontal dimension represents time, and each small blue box represents one sample. Note that the figure is only schematic, and that real recordings and clips typically contain orders of magnitude more samples than shown.

Note that, as shown in the figure, different clips may have different durations, and each clip is confined to a single recording channel.

An *annotation* provides one piece of information about a particular clip, and has a name and a value. A *classification* is an annotation that classifies the contents of a clip. For example, classification annotation might have the name "Classification" and a value like "Call" or "Noise".

A *tag* marks a clip as belonging to a set of clips. A tag has a name that can be used to refer to the set. For example, you might use a tag named "Review" to mark clips that you want to review with a collaborator, and a tag named "Export" to mark clips for which you want to export audio files.

Stations and devices

A monitoring *station* is a location where recordings are made, with a name and a fixed latitude, longitude, and elevation. A station also has recording devices associated with it, as described below.

A recording *device* is a piece of recording hardware, either an audio *recorder* (for example, an autonomous recording unit, a manually-operated field recorder, or a general-purpose computer with audio recording capabilities) or a *microphone*. When you build an archive, you tell Vesper what devices you used to create the recordings of the archive, including which devices were used at which stations during which time periods, and which microphones were connected to which recorder inputs during which time periods. From this information Vesper infers which microphone was used to record each channel of each recording. This allows Vesper to support various useful archive queries, for example to retrieve for display all of the clips with a particular classification that were made with a particular microphone at a particular station on a particular night.

Processors

A *processor* is software that processes existing data to create new data. Vesper currently offers two types of processors: detectors and annotators. A *detector* processes each of the channels of one or more recordings to create clips, for example to mark portions of recordings that contain bird calls. An *annotator* processes clips to create annotations. For example, a *classifier* is a common type of annotator that creates annotations that classify the contents of clips.

Summary

There are many kinds of archive data. For quick reference, here's a table summarizing those described above:

Data	Description
Recording	Continuous audio recording with one or more channels.
Channel	One channel of a recording, a sequence of samples.
Sample	Number representing an instantaneous air pressure measurement.
Clip	Continuous portion of one recording channel.
Annotation	One piece of information about a clip, with a name and a value.
Classification	Annotation that classifies a clip.
Tag	Name for a set of clips.
Station	A named monitoring location.
Device	Hardware recording device, either a recorder or a microphone.
Recorder	Device that records audio, creating recordings.
Microphone	Device that provides audio input to a recorder.
Processor	Software data processor, either a detector or an annotator.
Detector	Processor that creates clips from recordings.
Annotator	Processor that creates annotations for clips.
Classifier	Annotator that creates classifications.

1.2.4 Importing data

In this section of the tutorial, you will import a recording into your Vesper archive. Before you can do that, however, you must import some metadata that will allow Vesper to infer certain information about the recording, such as the station at which it was made and the microphone(s) that were used to make it. The availability of such metadata simplifies recording imports, and also helps Vesper to support powerful queries and data displays. Along with the metadata required for recording imports, you will also import metadata describing processors and annotations that Vesper will use when you work with your recording in the next section of the tutorial.

Import metadata

Vesper imports most metadata from text files that are in the YAML format. You can import metadata of various types from YAML files, including descriptions of stations, devices, processors, and annotations.

The archive template comes with several example YAML files in the `Metadata YAML` subdirectory of the archive directory. One of the files is named `One Station.yaml` and contains metadata for a monitoring setup with only one station. The metadata in this file will serve as the basis for your archive.

Note: Another of the files in the `Metadata YAML` directory is named `Two Stations.yaml` and describes a small monitoring network with two stations. We will not use that file in this tutorial, but it is provided as an example of how to specify metadata for more than one station.

To import metadata into your archive:

1. Make sure you have a Vesper server running in your archive directory, and point your browser to the archive. As at the end of the *Getting started* section, you should see a page much like this:

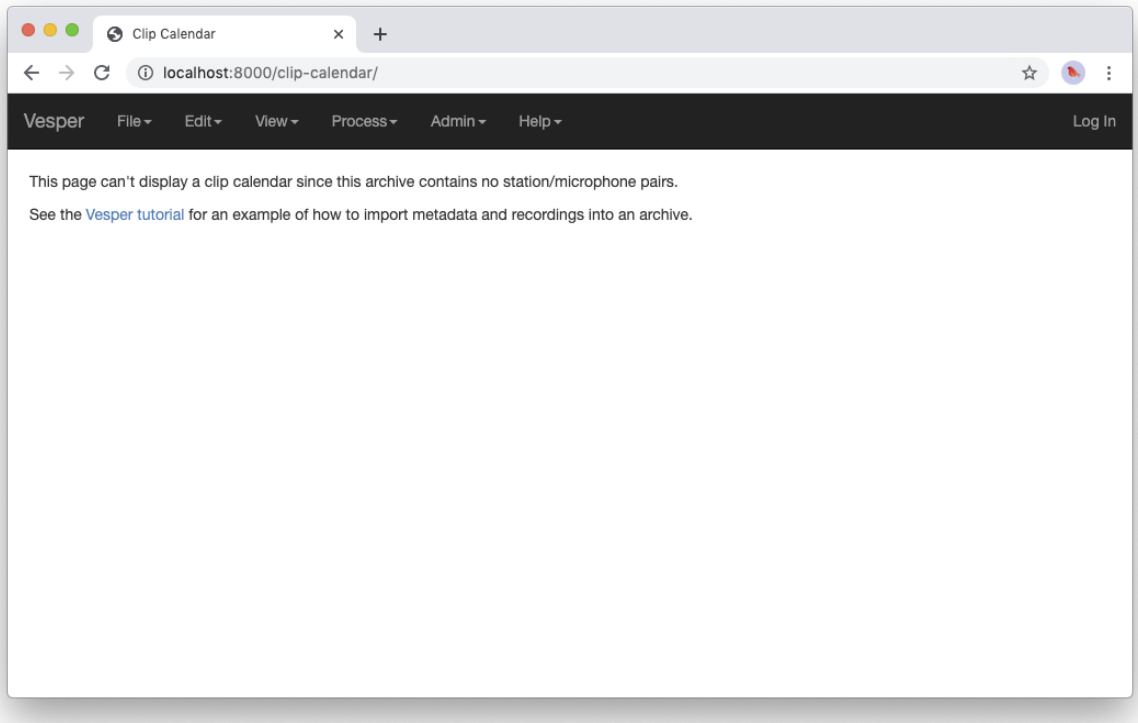

Fig. 6: An empty Vesper archive.

   The black bar near the top of the window is called the Vesper *navbar* (short for *navigation bar*), and the user interface elements with the little triangles at their right ends (named `File`, `Edit`, etc.) are called *dropdowns*.

2. Select `File->Import metadata` (that is, the `Import metadata` item within the `File` dropdown). This should take you to a login page, as shown in the following figure:

   Vesper requires that you be logged in as a specific user whenever you modify an archive, so it can keep track of who made the modifications. Enter the user name and password for the superuser you created in the *Create a*

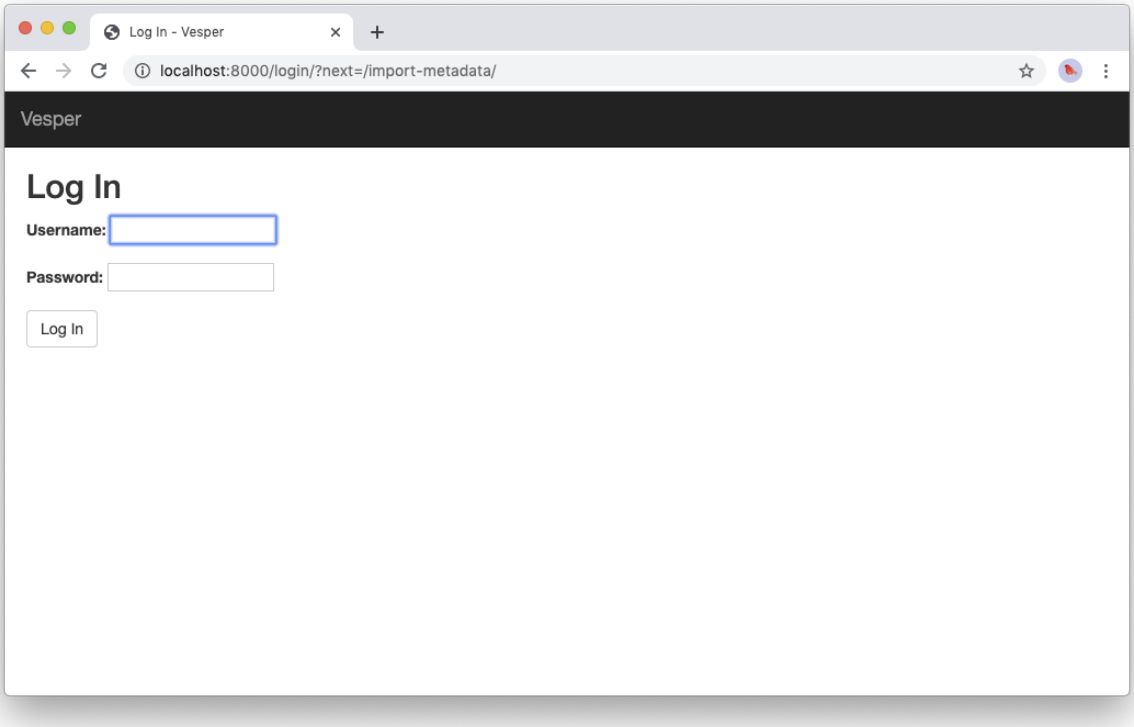

Fig. 7: The login page.

new Vesper archive section above, and press the `Log in` button. This should take you to a page that looks like this:

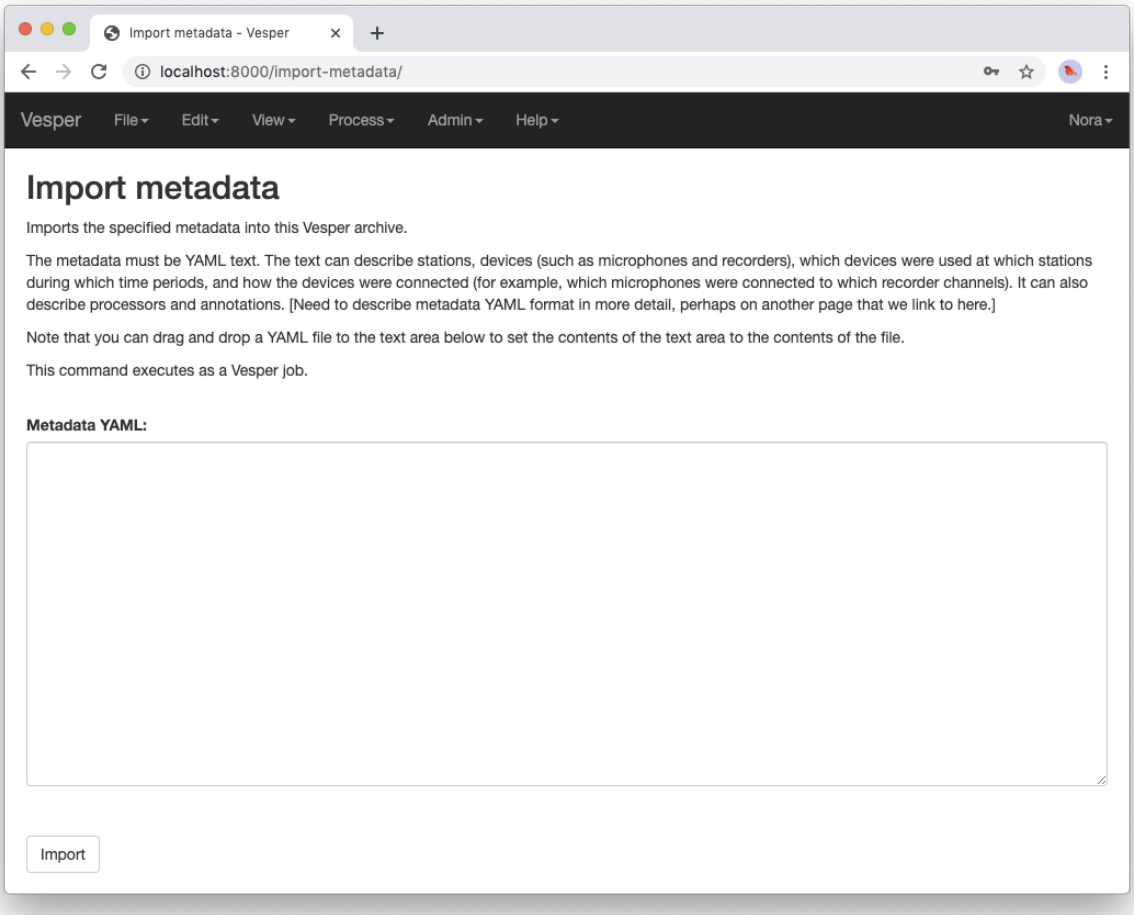

Fig. 8: The `Import metadata` page.

From a Windows Explorer or macOS Finder window, drag the `One Station.yaml` file and drop it onto the `Metadata YAML` text area on the `Import metadata` page. The contents of the file should appear in the text area, as shown in the following figure:

Look through the contents if you wish to see how they describe the station, devices, detectors, classifiers, annotations, etc. that you will add to your archive. Finally, press the `Import` button to import the data.

When you press the `Import` button, the Vesper client creates a textual *command* that describes the import operation you want to perform, including a copy of the text that you dropped onto the text area, and sends the command to the Vesper server for it to run. The server runs the command as a Vesper *job* and directs the client to a *job page* that provides information about the status of the job.

Note: While strictly speaking there is a difference between a *command*, which is a textual description of an operation, and the *execution* of that command as a job, the distinction is sometimes not important. In such situations we may ignore the distinction and speak of the command as an active entity, saying things like "the command imports data into the archive database", even though the active entity is really a job and not a com-

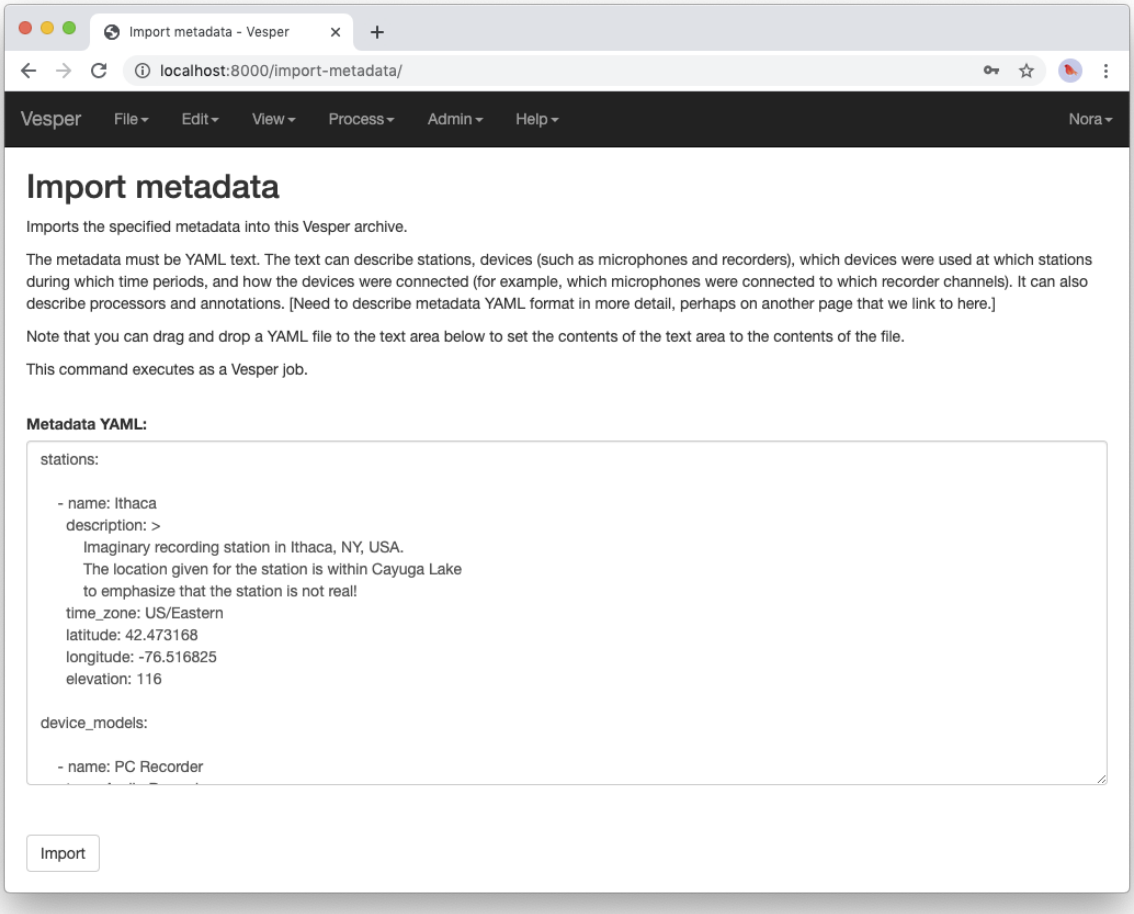

Fig. 9: The `Import metadata` page, including metadata.

mand. We will be careful to make the distinction when it is important.

The job page for your `Import metadata` command will initially look something like the following:

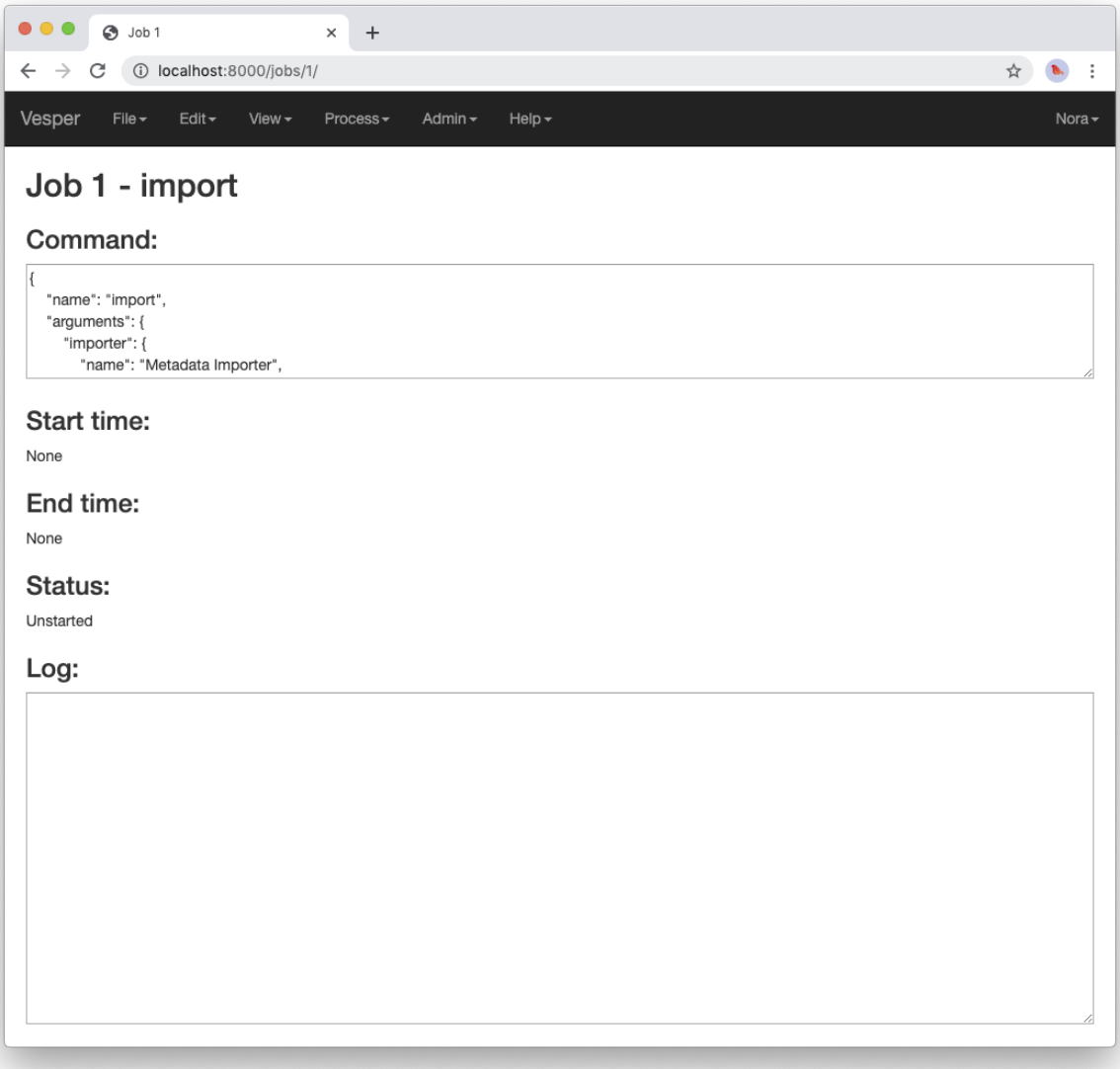

Fig. 10: A job page for an unstarted job.

Note that the status of the job is "Unstarted", indicating that when the page was sent from the server to the client the job had not yet started running.

You can refresh a job page in your browser to monitor the progress of the job. In Chrome, for example, you can do this by pressing the small circular arrow button just to the left of the address bar. (Yes, it's a little clunky for you to have to refresh the page yourself. A future version of Vesper will update job pages automatically to display progress.) While a job is running, its status is displayed as "Running", and when a job completes, its status changes to "Completed". For example, after the job pictured above completed its job page looked like this:

Every job has a *log* to which it writes messages as it runs to document its progress. The log of a job is displayed

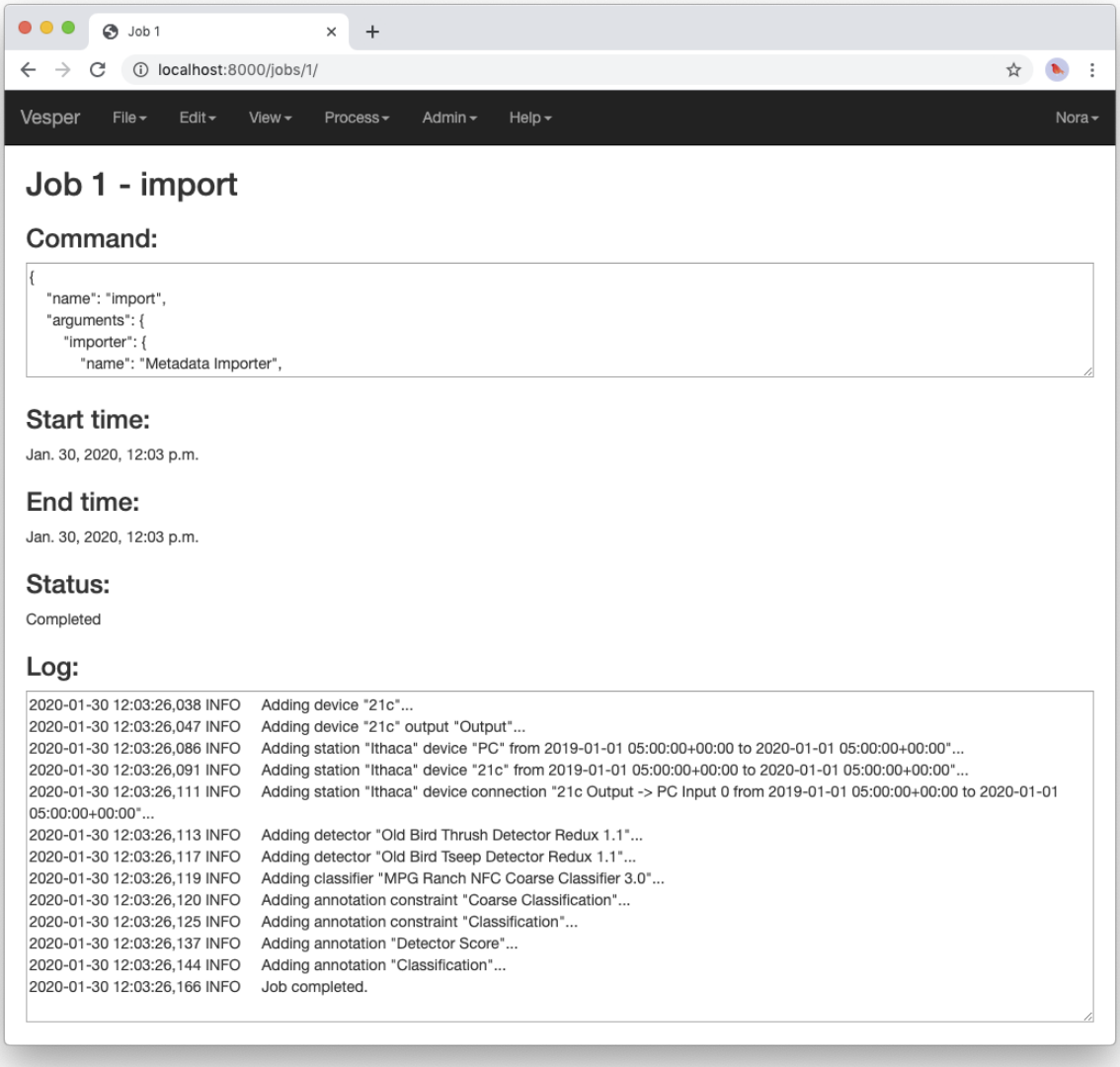

Fig. 11: A job page for a completed job.

at the bottom of the job page. In the log pictured above, note the messages that indicate the various objects that the job added to the archive.

Jobs sometimes fail to complete, for example if information required for the job is unavailable or because of a software bug. In such cases, the status of the job changes to "Failed", indicating that the job failed due to an error. For example, if you run the import command you ran above a second time it fails, as shown in the resulting job page:

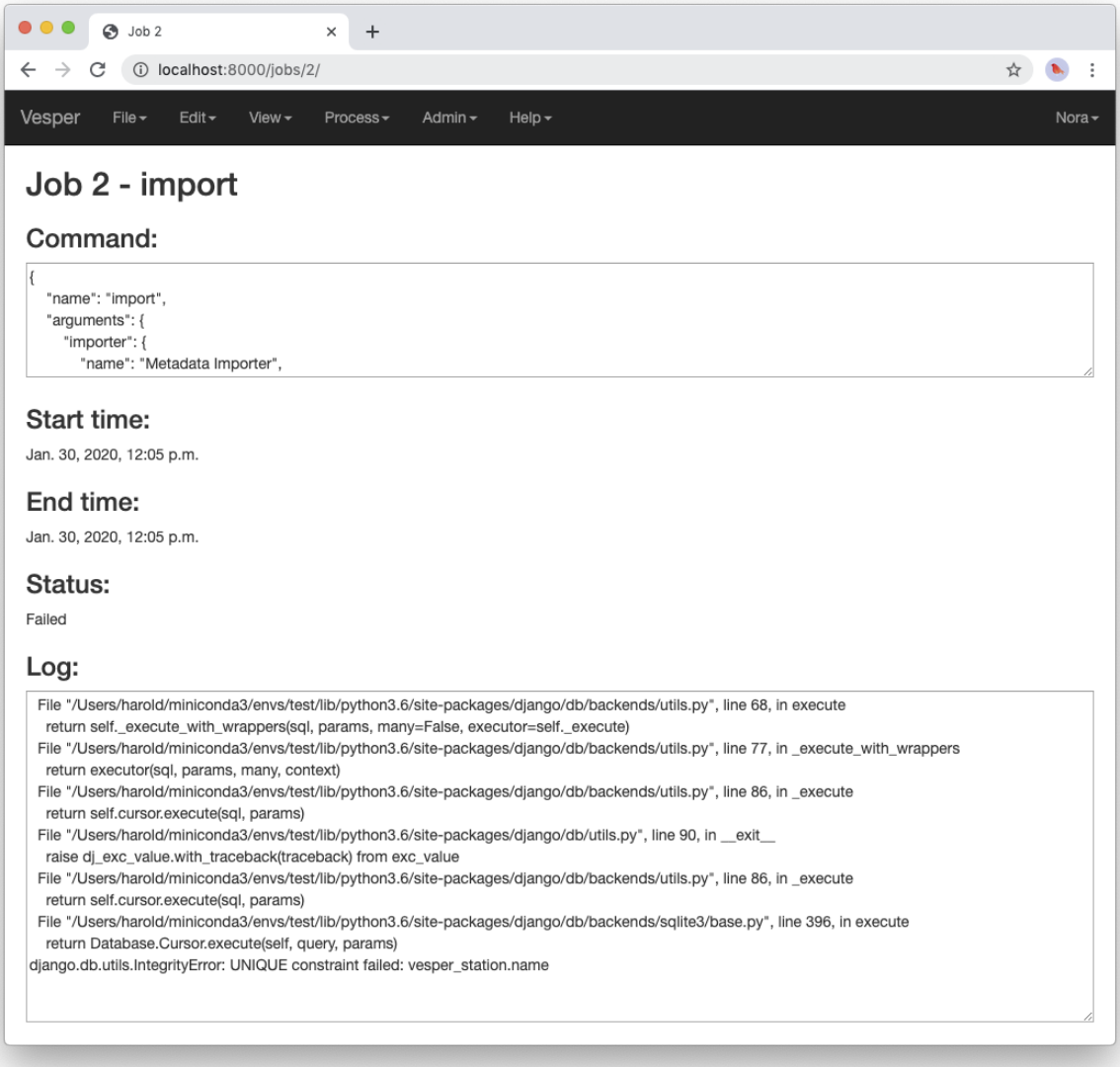

Fig. 12: A job page for a failed job.

The job fails because it attempts to create a new station whose name is the same as that of an existing station. That's not allowed, however: Vesper requires that the names of stations be unique. When a job fails, it usually raises an *exception* that includes information about the failure. The log for the failed job includes this information in the form of one or more error messages and a *stack trace* indicating exactly what parts of Vesper were running when the failure occurred. The portion of the log visible in the above figure shows the tail end of the stack trace, and the final line of the log indicates that the job failed because it violated a database uniqueness constraint concerning the station name. Both the error messages and the stack trace are useful for diagnosing

why a job failed, so that you can, say, fix a problem with your command or archive if that caused the failure, or report a problem with Vesper.

An earlier message in the log that is not visible in the figure (if you run the command yourself, you can scroll up in the log to see it) indicates that because the command failed, the archive database was restored to its state before the import. This is an important property of Vesper jobs that import metadata or recordings: when such a job fails, it leaves the archive database exactly as it was before the job started, preserving the integrity of the database and allowing you to resume work from the point just before you ran the failed job.

3. Go to the URL:

   localhost:8000

Previously, when you visited this URL, you saw a page that indicated that your archive was empty. Now, however, you see something slightly different, because of your metadata import:

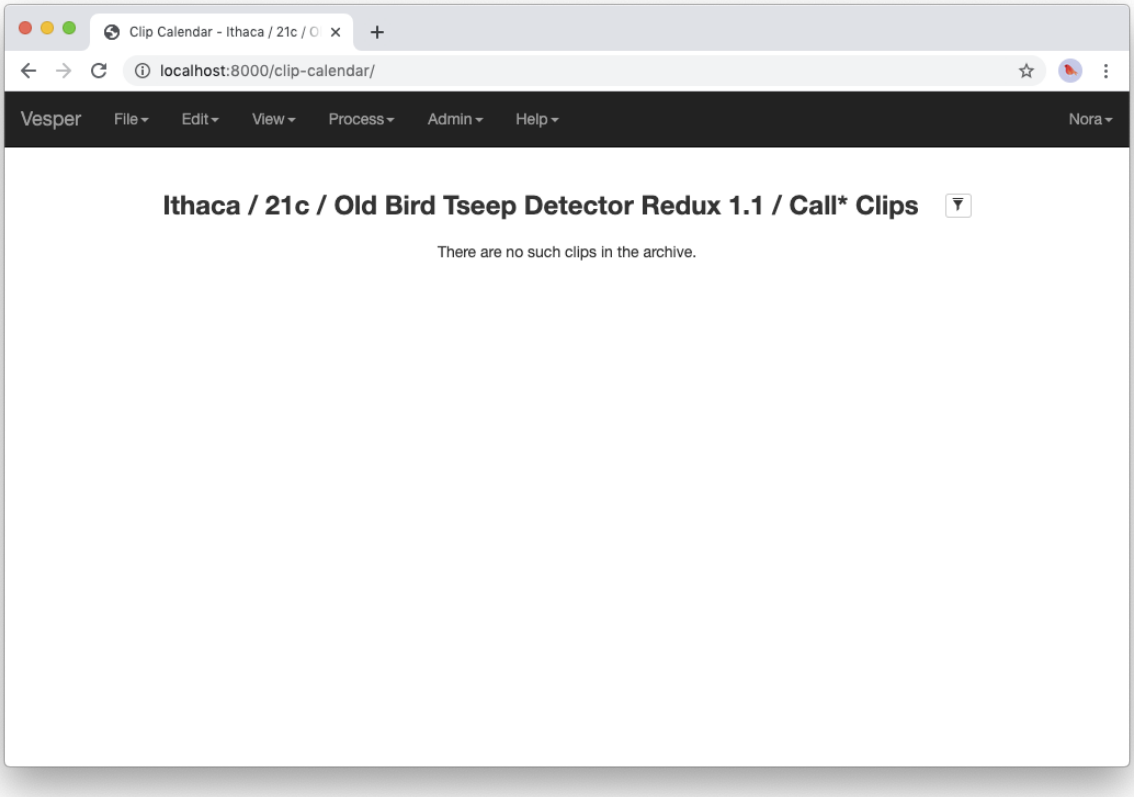

Fig. 13: An empty clip calendar.

This page displays a *clip calendar* for the station and microphone whose metadata you imported in the last step. That's progress over a message about a totally empty archive, but it's still not very interesting since, as the message in the calendar reflects, the archive does not yet contain any clips. Next, though, you'll import a recording and run some automatic detectors on it to create some clips to look at. You'll learn more about the contents and use of the clip calendar then.

Import a recording

Now you're ready to import some actual audio data into your Vesper archive! For the purpose of this tutorial, the Vesper project provides a short recording for you to import, but of course when you create your own archive you can import your own recordings into it.

To import a recording into your archive:

1. Download the recording file for this tutorial and put it in the `Recordings` subdirectory of your archive.
2. Select `File->Import recordings`. This should take you to a page that looks like this:

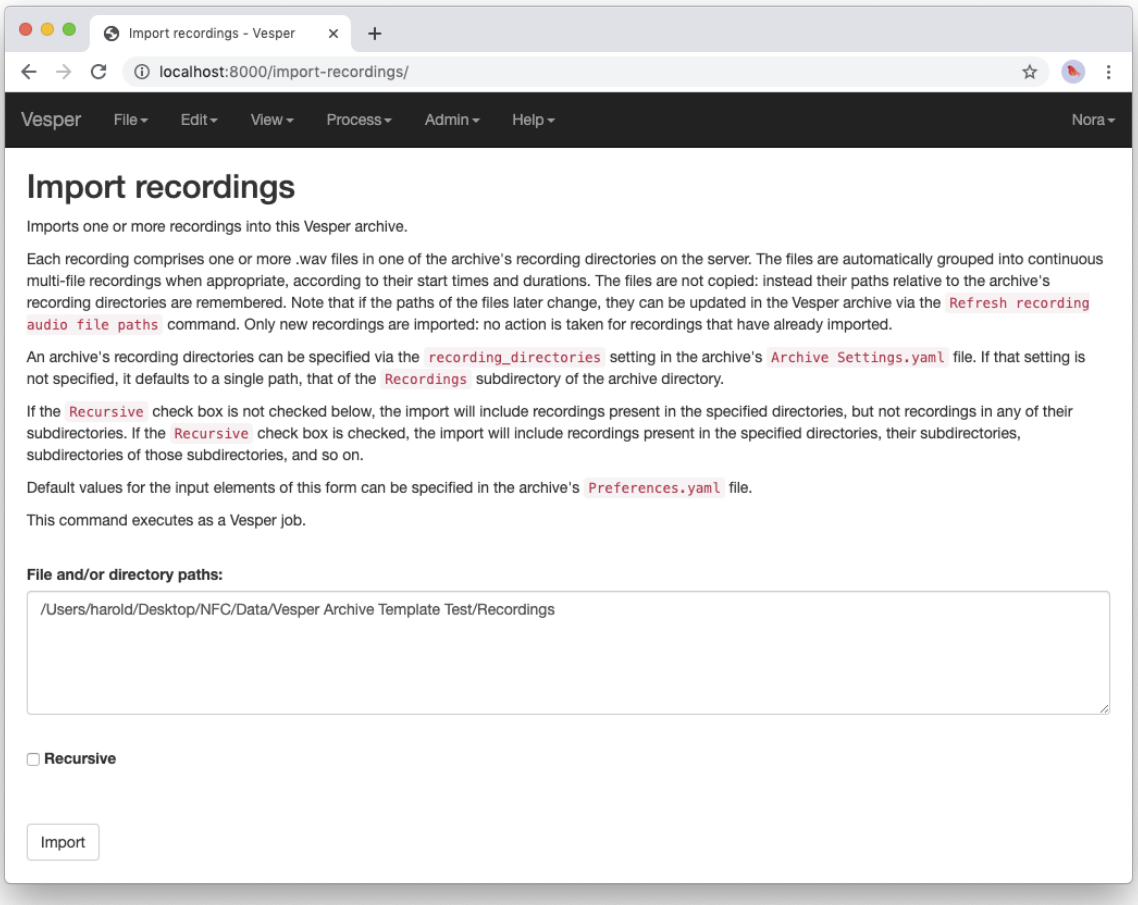

Fig. 14: The `Import recordings` page.

A Vesper archive stores metadata for each recording in the archive database, and audio data in one or more .wav audio files. All of the audio files are located in file system directories designated as the archive's *recording directories*. By default, an archive has a single recording directory, the `Recordings` subdirectory of the archive directory. The path of this directory for your archive should appear in the `File and/or directory paths` text area on the `Import recordings` page. We will use this default recording directory for the archive of this tutorial.

Note: If you want to store the audio data of an archive's recordings in one or more directories other than the default recording directory, you can explicitly list the recording directories in the `recording_directories`

setting of the `Archive Settings.yaml` file of the archive directory. This feature provides a lot of flexibility regarding where you can store your recordings' audio files. For example, you can store them outside of the archive directory, and even across multiple disks. You can read more about such possibilities in the example `Archive Settings.yaml` file provided with the archive template.

3. Press the `Import` button at the bottom of the `Import recordings` page to import your recording into the archive. This should take you to a job page that (after the job completes) looks like this:

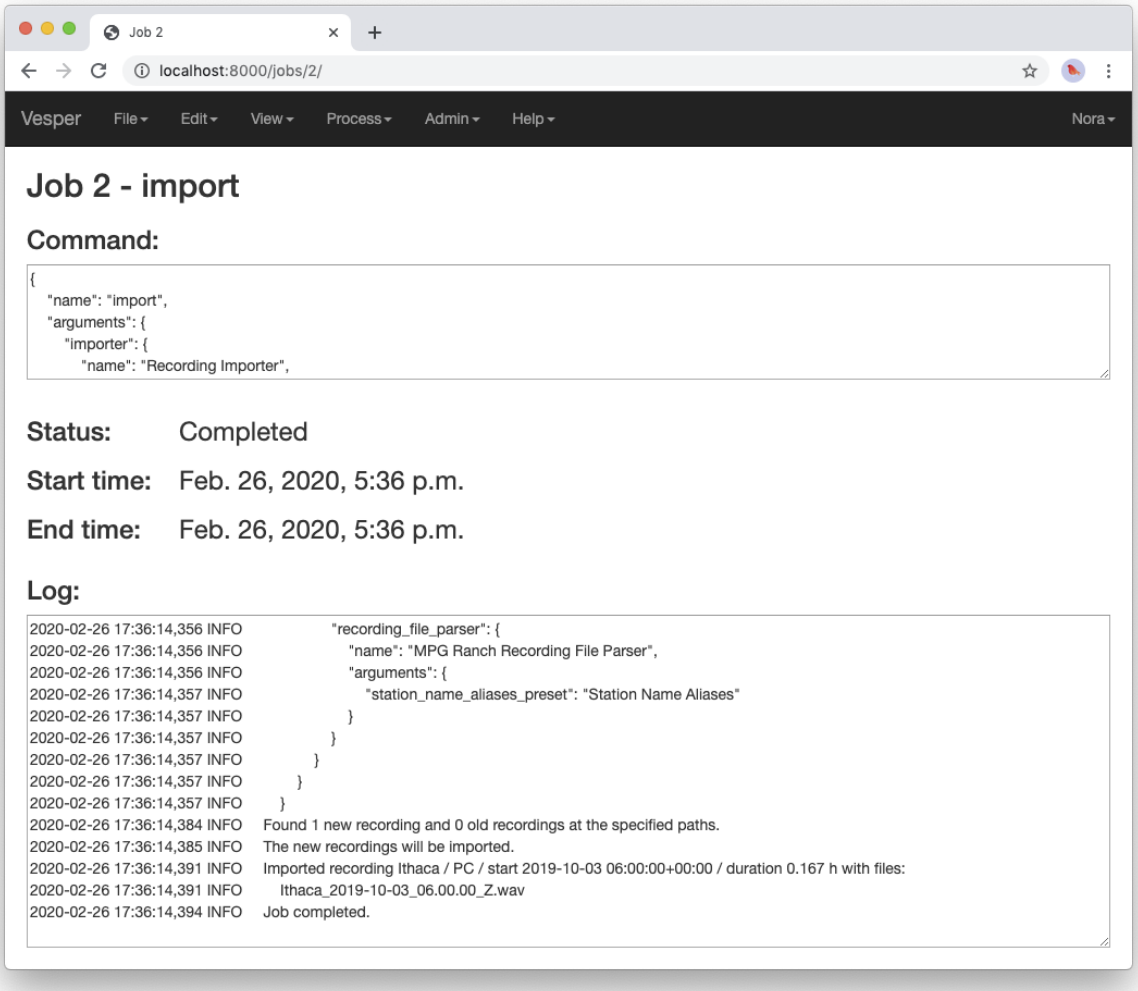

Fig. 15: `Import recordings` job page.

The log indicates that one recording file was imported into your archive from the archive directory.

When an `Import recordings` command imports a recording, it adds metadata for the recording to the archive database. The metadata include the station at which the recording was made, the number of channels of the recording, the microphone used to record each channel, and the locations on disk of the recording's audio files. The metadata are derived from the names and contents of the audio files as well as from other metadata already in the database, such as the metadata you imported with the `Import metadata` command.

When you run an `Import recordings` command, it analyzes the audio files in your archive's recording directories to determine which have already been imported and which are new, and imports only the new ones.

Thus you can run the command any number of times for an archive to import new recordings as they become available. For example, during a migration season you might run the command once each morning to import the previous night's recording.

Note that the `Import recordings` command does not move or copy the audio files of the recordings it imports: it leaves the files where they are, and simply notes their locations in the archive database. So *do not delete the files after import*: if you do and you don't have copies of the files elsewhere you will have lost them!

Note: The name of a recording file must be in one of several formats for Vesper to be able to parse certain metadata from it. These metadata include the name of the station at which the file was recorded and the file's start time. For example, the name of the recording file you imported in this section was:

Ithaca_2019-10-03_06.00.00_Z.wav

which specifies that the file was recorded at the Ithaca station beginning at 6:00 am on October 3, 2019 UTC time. The "Z" towards the end of the file name indicates that the time is UTC.

We recommend using UTC times in your recording file names, explicitly marked as such as in the example above. UTC times take some getting used to, but since they conform to an international standard they will be clearly interpretable all over the world for many years to come. If you use them you will necessarily avoid various possible pitfalls of local times, giving your recordings greater value, especially in the long term.

That said, Vesper can parse some files names that specify local start times. For example, it can parse a name like:

Ithaca_20191003_020000.wav

Since this file name is not explicitly specified as a UTC time (currently the only way to do that is with a name like the other example above, with the dashes in the dates and the dots in the times), Vesper assumes that it is a local time. It uses the time zone of the recording's station to convert that local time to the equivalent UTC time, since Vesper uses only UTC time internally. In this case, the Ithaca station is in the US/Eastern time zone, which was four hours behind UTC on the night of the recording. Thus the UTC start time for this file is 6:00 am on October 3, 2019 UTC time, the same as that specified by the first file name above.

If needed, you can specify station name aliases for use in recording file names. For example, if your recording files use "ITH" as an abbreviation for the Ithaca station, an appropriate station name alias would enable you to import files with names like:

ITH_2019-10-03_06.00.00_Z.wav

Station name aliases are specified via the `Station Name Aliases` preset, in the file `Presets/Station Name Aliases/Station Name Aliases.yaml` in your archive directory. See the example preset in the archive template for more documentation regarding this feature.

---

4. Select `View->View clip calendar`, which should take you to a page like the following:

   Now that there's a recording in your archive, the clip calendar looks more like an actual calendar. The gray bubble on the October 2 date indicates that the archive contains a recording for that date (the one you just imported), but the gray color indicates that there are no clips for that recording. In the next section of the tutorial you will create some clips by running a detector on your recording.

1.2.5 Processing data

Note: This section of the tutorial is coming soon!

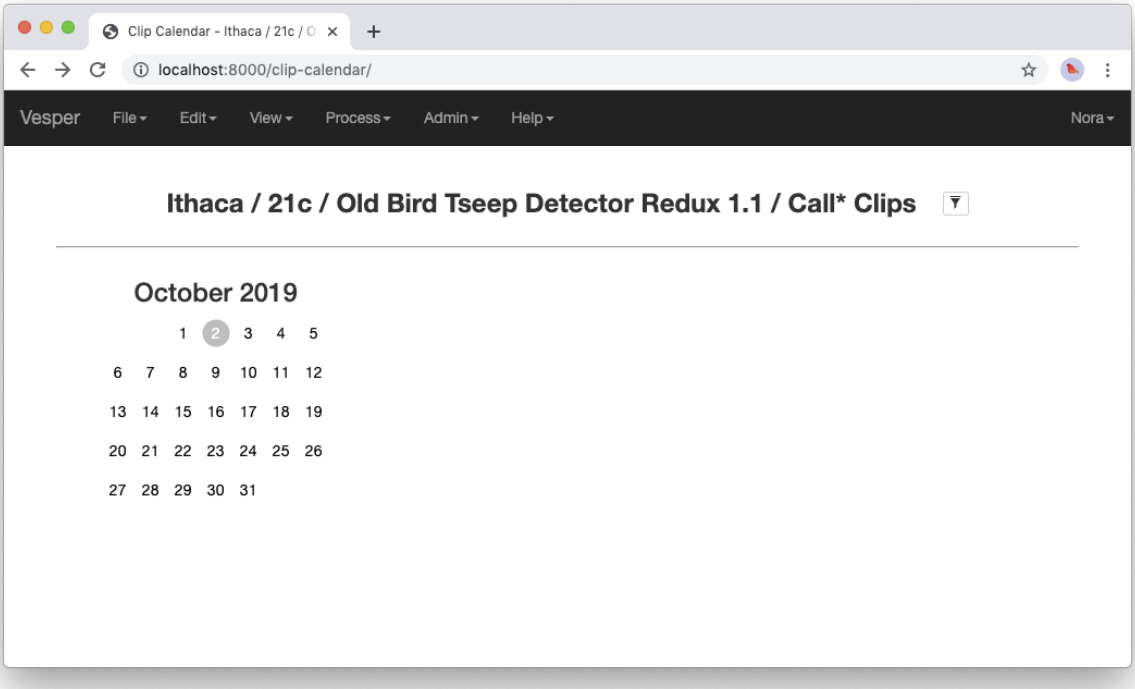

Fig. 16: A nonempty clip calendar.

Introduction

Run an automatic detector and classifier

Classify clips manually

1.2.6 Exporting data

Note: This section of the tutorial is coming soon!

Export clip metadata to a CSV file

Export clip audio files

1.3 FAQ

This section of the Vesper documentation answers frequently asked questions (FAQs) regarding Vesper.

1.3.1 When I start the Vesper server, it tells me I have "unapplied migrations". What does that mean, and what can I do about it?

It means that the relational database of your Vesper archive has an older and different structure than that expected by your version of the Vesper software. This kind of version mismatch can happen when you update Vesper to a new version, or if somebody supplies you with an archive created with an older version of Vesper than yours. The mismatch may (or may not, depending on the particular differences) make it impossible for Vesper to work properly with the database.

The process of updating the structure of your database is called *migration*, and it is typically straightforward. To migrate your database:

1. Make a backup copy of the `Archive Database.sqlite` database file that's in your archive directory.

> **Warning:** Always make a copy of your archive database before migrating it. Then if something goes wrong with the migration, you can always recover by restoring the original version of the database from your copy. If you don't make a copy of your database and the migration fails, you may lose some or all of your data!

2. Run the command:

   vesper_admin migrate

   in your archive directory from an Anaconda Prompt (on Windows) or terminal (on macOS or Linux). This should apply any needed migrations to your database to bring it up to date.

Note: In some Vesper installations (such as ones including Python 3.9.6 and Django 3.2.6) the `vesper_admin migrate` command will migrate your database correctly but not terminate. If the command seems to hang, producing no output for at least thirty seconds, type `Ctrl-C` on your keyboard to terminate it.

1.3.2 I ran the `vesper_admin` command and it hung. What do I do?

In some Vesper installations (such as ones including Python 3.9.6 and Django 3.2.6) the `vesper_admin` command will function properly, for example by creating a superuser or migrating a database, except that it won't terminate. If the command seems to hang, producing no output for at least thirty seconds, type `Ctrl-C` on your keyboard to terminate it.

1.3.3 How do I use BirdVoxDetect with Vesper?

BirdVoxDetect is a nocturnal flight call detector developed by the BirdVox project. While Vesper and BirdVox are separate software development efforts, you can use BirdVoxDetect from within Vesper just like any other supported detector. For example, you can use Vesper to run BirdVoxDetect on your recordings, view the resulting clips in clip albums, annotate them with species classifications, etc.

To use BirdVoxDetect with Vesper, first install it according to Vesper's BirdVoxDetect installation instructions. Then, add one or more BirdVoxDetect instances to your Vesper archive as described in the answer to this question. You can then run any of the BirdVoxDetect instances you added using Vesper's `Process->Detect` command.

1.3.4 How do I add a new detector to a Vesper archive?

First, a note about terminology. Within Vesper, the term *detector* is used in three different senses, and it is important to understand how these senses differ to avoid confusion. The three senses are termed *detector series*, *detector ver-*

sion, and *detector instance*, and they differ in their level of specificity. A *detector series* is a sequence of *detector versions*, which correspond to the usual notion of software versions. For example, the `BirdVoxDetect` detector series has to date included several versions, such as `BirdVoxDetect 0.4.0`, `BirdVoxDetect 0.4.1`, and `BirdVoxDetect 0.5.0`. Each detector version typically has one or more *settings*, such as a detection threshold, whose values must be specified before the detector can actually run. A detector version plus a set of values for its settings is a *detector instance*. Examples of `BirdVoxDetect` instances are `BirdVoxDetect 0.5.0 FT 50`, which is version `BirdVoxDetect 0.5.0` with a fixed threshold of 50, and `BirdVoxDetect 0.5.0 AT 40`, which is the same version with an adaptive threshold of 40.

Currently, when you add a new detector to a Vesper archive, you add a detector instance. There's no such thing (yet, at least) as adding the detector series `BirdVoxDetect` to an archive, or even the detector version `BirdVoxDetect 0.5.0`. You always add a detector instance, like `BirdVoxDetect 0.5.0 FT 50`. With that understood, in what follows we will use the term *detector* as a shorthand for *detector instance*.

Vesper's `Process->Detect` command allows you to run one or more detectors on a set of recordings. For a detector to appear in the form for that command, it must first be added to the archive database. You can do this with the `File->Import metadata` command.

It is common to import YAML metadata for one or more detectors when you create an archive, as described in the Importing data section of the Vesper tutorial. See the example YAML metadata files of the tutorial, particularly the `Processors` sections of those files, for examples of detector metadata.

It often happens, however, that after creating a Vesper archive and using it for awhile, you decide that you would like to add a new detector to the archive. This can happen for a variety of reasons. A new detector version might appear in a detector series you've been using, or you might want to try a new instance of a detector version you've been using, say with a different detection threshold.

Suppose, for example, that you've been using the detector `BirdVoxDetect 0.5.0 FT 50`, but would like to try `BirdVoxDetect 0.5.0 FT 40`, i.e. the same detector version with a lower threshold. To do that, you can create a YAML metadata file for just the new detector and import it. Specifically, you can:

1. Create a text file (`BirdVoxDetect 0.5.0 FT 40.yaml`, say, but you can call it anything you'd like since Vesper doesn't care about the file name) with the following contents. Note that the first line should not have any leading space, and the other lines should be indented relative to the first exactly as shown:

```
detectors:
    - name: BirdVoxDetect 0.5.0 FT 40
      description: BirdVoxDetect 0.5.0 NFC detector with a fixed threshold of 40.
```

2. Select `File->Import metadata` in Vesper to display the `Import metadata` form.
3. Drag your text file into the `Metadata YAML` text area of the form. The contents of the file should appear in the form.
4. Press the `Import` button to run the command.
5. After the command completes, restart the Vesper server for your archive to ensure that the server recognizes the new detector.

After these steps, the new detector should appear in all the appropriate places in the Vesper user interface, for example in the `Filter clips` clip album modal and the `Detect` form.

Note that in the case of BirdVoxDetect, you must also make sure that the appropriate version of BirdVoxDetect is installed on your system in an appropriately-named Conda environment. See Vesper's BirdVoxDetect installation instructions for how to create such an environment.

1.3.5 How do I modify the classification options displayed in Vesper?

The Vesper user interface (UI) presents classification options in several places, for example to let you specify which clips should be displayed in a clip album or which clips you would like to export to audio files. The options presented are determined by *annotation constraints* specified in the archive database. When you create a new archive, you typically create annotation constraints by importing a metadata YAML file using the `File->Import metadata` command, as in the Vesper tutorial.

If you later decide you would like to modify an annotation constraint, for example to add or remove a species from it, you can do so using the *Django admin interface*. Django is a third-party web framework used by Vesper, and the admin interface is a set of web pages provided with Django that allow you to edit your Vesper archive database. Eventually, when the Vesper UI is more complete, it will never be necessary to use the Django admin interface to edit Vesper archives, but as of this writing it is still needed for some tasks.

> **Warning:** We strongly recommend that you make a copy of your archive database before you edit it with the Django admin interface. The archive database is contained in the file `Archive Database.sqlite` in your archive directory. If you make a copy of this file before editing it, then if you make a mistake in your editing, you can always recover from the mistake by restoring the original version of the database from your copy. If you don't make a copy of your database and accidentally mangle it with your edits, you may be very sorry!

To edit an annotation constraint with the Django admin interface, first point your browser to the URL `localhost:8000/admin`. Within that interface, select `Annotation constraints` and then the name of the constraint you want to edit, for example `Classification`. This will display a form that you can use to edit the selected constraint. The constraint contains a YAML text field called `Text`. The value of that field is a YAML mapping that includes an item named `values` that you can edit to add or remove classification values.

Once you have edited an annotation constraint, you should restart your Vesper server to be sure to pick up your changes.

1.3.6 How do I modify the key bindings of a clip album?

Vesper clip albums allow you to type keys on your keyboard to invoke a variety of *clip album commands*. For example, you might type ">" to invoke a command to navigate to the next clip album page, "n" to annotate the selected clip as a "Noise", or "/" to play the selected clip. You can configure which keys (or key sequences) invoke which commands using *clip album command presets*. Each of these presets is a YAML file in the `Presets/Clip Album Commands` subdirectory of your archive directory. To modify any of the presets, just use your favorite text editor. After you edit a preset, you should restart your Vesper server to be sure to pick up your changes.

Only one clip album command preset can be active at a time in a given clip album. To choose the active preset for a clip album, select `Choose presets...` from the rightmost button to the right of the album's title. To choose the default preset for your clip albums, edit the `default_presets` item in your archive preferences, stored in the file `Preferences.yaml` in your archive directory. After you edit this file, you should restart your Vesper server to be sure to pick up your changes.