This file provides an introductory tour of Onda.jl by generating, storing, and loading a toy Onda dataset. Run lines in the REPL to inspect output at each step! Tests are littered throughout to demonstrate functionality in a concrete manner, and so that we can ensure examples stay updated as the package evolves.

NOTE: You should read **https://github.com/beacon-biosignals/OndaFormat** before and/or alongside the completion of this tour; it explains the purpose/structure of the format.

```
using Onda, TimeSpans, DataFrames, Dates, UUIDs, Test, ConstructionBase
```

```
using TimeSpans: duration, translate, start, stop, index_from_time, time_from_index
```

```
using PlutoUI
```

generate some mock data

Let's kick off the tour by generating some mock data to play with in subsequent sections!

Onda is primarily concerned with manipulating 3 interrelated entities. Paraphrasing from the Onda specification, these entities are:

- "signals": A signal is the digitized output of a process, comprised of metadata (e.g. LPCM encoding, channel information, sample data path/format information, etc.) and associated multi-channel sample data.
- "recordings": A recording is a collection of one or more signals recorded simultaneously over some time period.
- "annotations": An annotation is a a piece of (meta)data associated with a specific time span within a specific recording.

Signals and annotations are serialized as Arrow tables, while each sample data file is serialized to the file format specified by its corresponding signal's metadata. A "recording" is simply the collection of signals and annotations that share a common `recording` field.

Below, we generate a bunch of signals/annotations across 10 recordings, writing the corresponding Arrow tables and sample data files to a temporary directory.

```
saws (generic function with 1 method)
```

```
saws(info, duration) = [(j + i) % 100 * info.sample_resolution_in_unit for
                        i in 1:channel_count(info), j in 1:sample_count(info,
duration)]
```

```
root = "/tmp/jl_Sflakp"
```

```
root = mktempdir()
```

```
signals_list = ▶Onda.Signal[]
```

```
signals_list = Signal[]
```

```
signals_recordings =
▶Base.UUID[UUID("41595379-0887-4205-bef2-b2c5470446ab"), UUID("95306252-5c4d-49ed-9da3-c2
```

```
signals_recordings = [uuid4() for _ in 1:10]
```

```
[ Info: 2021-02-16T14:49:13.434 | generating /tmp/jl_Sflakp/41595379-0887-4205-bef2-b2c5470
446ab_eeg.lpcm...
[ Info: 2021-02-16T14:49:14.916 | generating /tmp/jl_Sflakp/41595379-0887-4205-bef2-b2c5470
446ab_ecg.lpcm.zst...
[ Info: 2021-02-16T14:49:15.519 | generating /tmp/jl_Sflakp/41595379-0887-4205-bef2-b2c5470
446ab_spo2.lpcm.zst...
[ Info: 2021-02-16T14:49:15.711 | generating /tmp/jl_Sflakp/95306252-5c4d-49ed-9da3-c2ab6ae
fc482_eeg.lpcm.zst...
[ Info: 2021-02-16T14:49:16.113 | generating /tmp/jl_Sflakp/95306252-5c4d-49ed-9da3-c2ab6ae
fc482_ecg.lpcm...
[ Info: 2021-02-16T14:49:16.193 | generating /tmp/jl_Sflakp/95306252-5c4d-49ed-9da3-c2ab6ae
fc482_spo2.lpcm.zst...
[ Info: 2021-02-16T14:49:16.584 | generating /tmp/jl_Sflakp/897dcfd4-bf80-410b-926c-68103c3
57ccf_eeg.lpcm.zst...
[ Info: 2021-02-16T14:49:16.784 | generating /tmp/jl_Sflakp/897dcfd4-bf80-410b-926c-68103c3
57ccf_ecg.lpcm...
[ Info: 2021-02-16T14:49:16.845 | generating /tmp/jl_Sflakp/897dcfd4-bf80-410b-926c-68103c3
57ccf_spo2.lpcm.zst...
[ Info: 2021-02-16T14:49:17.121 | generating /tmp/jl_Sflakp/0e190006-5939-454d-8d57-6146fd7
d94ee_eeg.lpcm.zst...
[ Info: 2021-02-16T14:49:17.404 | generating /tmp/jl_Sflakp/0e190006-5939-454d-8d57-6146fd7
```

```
with_terminal() do # we'll use a PlutoUI terminal to view the logs
    for recording in signals_recordings
        for (kind, channels) in ("eeg" => ["fp1", "f3", "c3", "p3",
                                           "f7", "t3", "t5", "o1",
                                           "fz", "cz", "pz",
                                           "fp2", "f4", "c4", "p4",
                                           "f8", "t4", "t6", "o2"],
                                 "ecg" => ["avl", "avr"],
                                 "spo2" => ["spo2"])
            file_format = rand(("lpcm", "lpcm.zst"))
            file_path = joinpath(root, string(recording, "_", kind, ".",
file_format))
            Onda.log("generating $file_path...")
            info = SamplesInfo(; kind, channels,
                                 sample_unit="microvolt",
                                 sample_resolution_in_unit=rand((0.25, 1)),
                                 sample_offset_in_unit=rand((-1, 0, 1)),
                                 sample_type=rand((Float32, Int16, Int32)),
                                 sample_rate=rand((128, 256, 143.5)))
            data = saws(info, Minute(rand(1:10)))
            samples = Samples(data, info, false)
            start = Second(rand(0:30))
```

```julia
            signal = store(file_path, file_format, samples, recording, start)
            push!(signals_list, signal)
        end
    end
end
```

```
path_to_signals_file = "/tmp/jl_Sflakp/test.onda.signals.arrow"
```

```julia
path_to_signals_file = joinpath(root, "test.onda.signals.arrow")
```

```
Tables.CopiedColumns{NamedTuple{(:recording, :file_path, :file_format, :span, :kind, :chan
5])
```

```julia
write_signals(path_to_signals_file, signals_list)
```

```
[ Info: 2021-02-16T14:49:36.363 | wrote out /tmp/jl_Sflakp/test.onda.signals.arrow
```

```julia
with_terminal() do
    Onda.log("wrote out $path_to_signals_file")
end
```

```
annotations_list = ▸Onda.Annotation[]
```

```julia
annotations_list = Annotation[]
```

```
sources =
▸(UUID("2c64c36c-4060-4336-90a6-31e3eaf3bd43"),  UUID("5fac543c-b60e-4c1c-9465-213ffb82cf0
```

```julia
sources = (uuid4(), uuid4(), uuid4())
```

```
annotations_recordings =
▸Base.UUID[UUID("41595379-0887-4205-bef2-b2c5470446ab"),  UUID("95306252-5c4d-49ed-9da3-c2
```

```julia
annotations_recordings = vcat(signals_recordings[1:end-1], uuid4()) # overlapping but
not equal to signals_recordings
```

```julia
for recording in annotations_recordings
    for i in 1:rand(3:10)
        start = Second(rand(0:60))
        annotation = Annotation(recording, uuid4(), TimeSpan(start, start +
Second(rand(1:30)));
                                rating=rand(1:100), quality=rand(("good", "bad")),
source=rand(sources))
        push!(annotations_list,  annotation)
    end
end
```

```
path_to_annotations_file = "/tmp/jl_Sflakp/test.onda.annotations.arrow"
```

```julia
path_to_annotations_file = joinpath(root, "test.onda.annotations.arrow")
```

```
Tables.CopiedColumns{NamedTuple{(:recording, :id, :span, :rating, :quality, :source),Tuple
```

```julia
write_annotations(path_to_annotations_file, annotations_list)
```

```
[ Info: 2021-02-16T14:49:42.065 | wrote out /tmp/jl_Sflakp/test.onda.annotations.arrow
```

```julia
with_terminal() do
    Onda.log("wrote out $path_to_annotations_file")
end
```

basic Onda + DataFrames patterns

Since signals and annotations are represented tabularly, any package that supports the Tables.jl interface can be used to interact with them. Here, we show how you can use DataFrames.jl to perform a variety of common operations.

Note that most of these operations are only shown here on a single table to avoid redundancy, but these examples are generally applicable to both signals and annotations tables.

Read Onda Arrow files into `DataFrame`s:

```julia
signals =
```

	recording	file_path	file
1	UUID("41595379-0887-4205-bef2-b2c54704	"/tmp/jl_Sflakp/41595379-0887-4205-bef	"lp
2	UUID("41595379-0887-4205-bef2-b2c54704	"/tmp/jl_Sflakp/41595379-0887-4205-bef	"lp
3	UUID("41595379-0887-4205-bef2-b2c54704	"/tmp/jl_Sflakp/41595379-0887-4205-bef	"lp
4	UUID("95306252-5c4d-49ed-9da3-c2ab6aef	"/tmp/jl_Sflakp/95306252-5c4d-49ed-9da	"lp
5	UUID("95306252-5c4d-49ed-9da3-c2ab6aef	"/tmp/jl_Sflakp/95306252-5c4d-49ed-9da	"lp
6	UUID("95306252-5c4d-49ed-9da3-c2ab6aef	"/tmp/jl_Sflakp/95306252-5c4d-49ed-9da	"lp
7	UUID("897dcfd4-bf80-410b-926c-68103c35	"/tmp/jl_Sflakp/897dcfd4-bf80-410b-926	"lp
8	UUID("897dcfd4-bf80-410b-926c-68103c35	"/tmp/jl_Sflakp/897dcfd4-bf80-410b-926	"lp
9	UUID("897dcfd4-bf80-410b-926c-68103c35	"/tmp/jl_Sflakp/897dcfd4-bf80-410b-926	"lp
10	UUID("0e190006-5939-454d-8d57-6146fd7d	"/tmp/jl_Sflakp/0e190006-5939-454d-8d5	"lp

```
signals = DataFrame(read_signals(path_to_signals_file))
```

annotations =

	recording	id	
1	UUID("41595379-0887-4205-bef2-b2c54704	UUID("669f515a-c928-4556-9666-8f878f2d	Tim
2	UUID("41595379-0887-4205-bef2-b2c54704	UUID("96bb8090-7365-4765-8a88-c9702a57	Tim
3	UUID("41595379-0887-4205-bef2-b2c54704	UUID("642cb4bf-75e5-41c7-9630-ca8ffecf	Tim
4	UUID("41595379-0887-4205-bef2-b2c54704	UUID("603e1653-aa15-4c43-baff-2ab47a7b	Tim
5	UUID("95306252-5c4d-49ed-9da3-c2ab6aef	UUID("e2ccadf1-d7fb-426c-8a05-d7b79610	Tim
6	UUID("95306252-5c4d-49ed-9da3-c2ab6aef	UUID("cf1b7c39-d720-4308-bcb1-03d527b6	Tim
7	UUID("95306252-5c4d-49ed-9da3-c2ab6aef	UUID("3abd33ef-c289-4e49-9831-66dbd8ef	Tim
8	UUID("95306252-5c4d-49ed-9da3-c2ab6aef	UUID("f93052a2-0e2b-426e-84a4-1c8864b8	Tim
9	UUID("95306252-5c4d-49ed-9da3-c2ab6aef	UUID("26a71583-5764-4521-a8e2-40eedc28	Tim
10	UUID("95306252-5c4d-49ed-9da3-c2ab6aef	UUID("2c1f3c3d-71e6-406d-ba15-819415f0	Tim

```
annotations = DataFrame(read_annotations(path_to_annotations_file))
```

Grab all multichannel signals greater than 5 minutes long:

	recording	file_path	file_forr
1	UUID("41595379-0887-4205-bef2-b2c54704	"/tmp/jl_Sflakp/41595379-0887-4205-bef	"lpcm"
2	UUID("897dcfd4-bf80-410b-926c-68103c35	"/tmp/jl_Sflakp/897dcfd4-bf80-410b-926	"lpcm"
3	UUID("0d8711e8-0ba0-4009-8528-3c9a3d41	"/tmp/jl_Sflakp/0d8711e8-0ba0-4009-852	"lpcm.z
4	UUID("485f1628-ed5d-4732-8b15-65640329	"/tmp/jl_Sflakp/485f1628-ed5d-4732-8b1	"lpcm.z
5	UUID("485f1628-ed5d-4732-8b15-65640329	"/tmp/jl_Sflakp/485f1628-ed5d-4732-8b1	"lpcm.z

```
filter(s -> length(s.channels) > 1 && duration(s.span) > Minute(5), signals)
```

Get all signals from a given recording:

target = UUID("95306252-5c4d-49ed-9da3-c2ab6aefc482")

```
target = rand(signals.recording)
```

	recording	file_path	file_forn
1	UUID("95306252-5c4d-49ed-9da3-c2ab6aef	"/tmp/jl_Sflakp/95306252-5c4d-49ed-9da	"lpcm.zs
2	UUID("95306252-5c4d-49ed-9da3-c2ab6aef	"/tmp/jl_Sflakp/95306252-5c4d-49ed-9da	"lpcm"
3	UUID("95306252-5c4d-49ed-9da3-c2ab6aef	"/tmp/jl_Sflakp/95306252-5c4d-49ed-9da	"lpcm.zs

```
view(signals, findall(==(target), signals.recording), :)
```

Group/index signals by recording:

```
grouped =
```

GroupedDataFrame with 10 groups based on key: recording

First Group (3 rows): recording = UUID("41595379-0887-4205-bef2-b2c5470446ab")

	recording	file_path
	UUID...	String
1	41595379-0887-4205-bef2-b2c5470446ab	/tmp/jl_Sflakp/41595379-0887-4205-bef2-b2c5470446ab_eeg.lpcm
2	41595379-0887-4205-bef2-b2c5470446ab	/tmp/jl_Sflakp/41595379-0887-4205-bef2-b2c5470446ab_ecg.lpcm.zst
3	41595379-0887-4205-bef2-b2c5470446ab	/tmp/jl_Sflakp/41595379-0887-4205-bef2-b2c5470446ab_spo2.lpcm.zst

⋮

Last Group (3 rows): recording = UUID("485f1628-ed5d-4732-8b15-65640329e302")

	recording	file_path
	UUID...	String
1	485f1628-ed5d-4732-8b15-65640329e302	/tmp/jl_Sflakp/485f1628-ed5d-4732-8b15-65640329e302_eeg.lpcm.zst
2	485f1628-ed5d-4732-8b15-65640329e302	/tmp/jl_Sflakp/485f1628-ed5d-4732-8b15-65640329e302_ecg.lpcm.zst
3	485f1628-ed5d-4732-8b15-65640329e302	/tmp/jl_Sflakp/485f1628-ed5d-4732-8b15-65640329e302_spo2.lpcm

```
grouped = groupby(signals, :recording)
```

recording	file_path	file_forn

	recording	file_path	file_forn
1	UUID("95306252-5c4d-49ed-9da3-c2ab6aef	"/tmp/jl_Sflakp/95306252-5c4d-49ed-9da	"lpcm.z
2	UUID("95306252-5c4d-49ed-9da3-c2ab6aef	"/tmp/jl_Sflakp/95306252-5c4d-49ed-9da	"lpcm"
3	UUID("95306252-5c4d-49ed-9da3-c2ab6aef	"/tmp/jl_Sflakp/95306252-5c4d-49ed-9da	"lpcm.z

```
grouped[(; recording=target)]
```

Group/index signals + annotations by recording together:

```
dict =
▶Dict(UUID("897dcfd4-bf80-410b-926c-68103c357ccf") ⇒ (
```

	recording
1	UUID("897dcfd4-bf80-410b-926c-
2	UUID("897dcfd4-bf80-410b-926c-
3	UUID("897dcfd4-bf80-410b-926c-

```
dict = Onda.gather(:recording, signals, annotations)
```

▸ (

	recording	file_path	file_
1	UUID("95306252-5c4d-49ed-9da3-c2ab6aef	"/tmp/jl_Sflakp/95306252-5c4d-49ed-9da	"lpcm
2	UUID("95306252-5c4d-49ed-9da3-c2ab6aef	"/tmp/jl_Sflakp/95306252-5c4d-49ed-9da	"lpcm
3	UUID("95306252-5c4d-49ed-9da3-c2ab6aef	"/tmp/jl_Sflakp/95306252-5c4d-49ed-9da	"lpcm

```
dict[target]
```

Count number of signals in each recording:

	recording	nrow
1	UUID("41595379-0887-4205-bef2-b2c54704	3
2	UUID("95306252-5c4d-49ed-9da3-c2ab6aef	3
3	UUID("897dcfd4-bf80-410b-926c-68103c35	3
4	UUID("0e190006-5939-454d-8d57-6146fd7d	3
5	UUID("3ae32bc3-b5d3-4ef0-b9c0-69780a07	3
6	UUID("2de1371b-2d2d-418b-a97c-c169b75a	3
7	UUID("19ce2220-00fd-49a6-951c-918f493b	3
8	UUID("dd4935a2-4719-423f-b900-340016dc	3
9	UUID("0d8711e8-0ba0-4009-8528-3c9a3d41	3
10	UUID("485f1628-ed5d-4732-8b15-65640329	3

```
combine(groupby(signals, :recording), nrow)
```

Grab the longest signal in each recording:

	recording	file_path	file_

	recording	file_path	file_
1	UUID("41595379-0887-4205-bef2-b2c54704	"/tmp/jl_Sflakp/41595379-0887-4205-bef	"lp
2	UUID("95306252-5c4d-49ed-9da3-c2ab6aef	"/tmp/jl_Sflakp/95306252-5c4d-49ed-9da	"lp
3	UUID("897dcfd4-bf80-410b-926c-68103c35	"/tmp/jl_Sflakp/897dcfd4-bf80-410b-926	"lp
4	UUID("0e190006-5939-454d-8d57-6146fd7d	"/tmp/jl_Sflakp/0e190006-5939-454d-8d5	"lp
5	UUID("3ae32bc3-b5d3-4ef0-b9c0-69780a07	"/tmp/jl_Sflakp/3ae32bc3-b5d3-4ef0-b9c	"lp
6	UUID("2de1371b-2d2d-418b-a97c-c169b75a	"/tmp/jl_Sflakp/2de1371b-2d2d-418b-a97	"lp
7	UUID("19ce2220-00fd-49a6-951c-918f493b	"/tmp/jl_Sflakp/19ce2220-00fd-49a6-951	"lp
8	UUID("dd4935a2-4719-423f-b900-340016dc	"/tmp/jl_Sflakp/dd4935a2-4719-423f-b90	"lp

```
combine(s -> s[argmax(duration.(s.span)), :], groupby(signals, :recording))
```

Load all sample data for a given recording:

	recording	file_path	file_form
1	UUID("95306252-5c4d-49ed-9da3-c2ab6aef	"/tmp/jl_Sflakp/95306252-5c4d-49ed-9da	"lpcm.z
2	UUID("95306252-5c4d-49ed-9da3-c2ab6aef	"/tmp/jl_Sflakp/95306252-5c4d-49ed-9da	"lpcm"
3	UUID("95306252-5c4d-49ed-9da3-c2ab6aef	"/tmp/jl_Sflakp/95306252-5c4d-49ed-9da	"lpcm.z

```
transform(view(signals, findall(==(target), signals.recording), :),
          AsTable(:) => ByRow(load) => :samples)
```

Delete all sample data for a given recording (uncomment the inline-commented section to actual delete filtered signals' sample data!):

```
signals_copy =
```

	recording	file_path	file
1	UUID("41595379-0887-4205-bef2-b2c54704	"/tmp/jl_Sflakp/41595379-0887-4205-bef	"lp
2	UUID("41595379-0887-4205-bef2-b2c54704	"/tmp/jl_Sflakp/41595379-0887-4205-bef	"lp
3	UUID("41595379-0887-4205-bef2-b2c54704	"/tmp/jl_Sflakp/41595379-0887-4205-bef	"lp
4	UUID("95306252-5c4d-49ed-9da3-c2ab6aef	"/tmp/jl_Sflakp/95306252-5c4d-49ed-9da	"lp
5	UUID("95306252-5c4d-49ed-9da3-c2ab6aef	"/tmp/jl_Sflakp/95306252-5c4d-49ed-9da	"lp

	recording	file_path	file
6	UUID("95306252-5c4d-49ed-9da3-c2ab6aef	"/tmp/jl_Sflakp/95306252-5c4d-49ed-9da	"lp
7	UUID("897dcfd4-bf80-410b-926c-68103c35	"/tmp/jl_Sflakp/897dcfd4-bf80-410b-926	"lp
8	UUID("897dcfd4-bf80-410b-926c-68103c35	"/tmp/jl_Sflakp/897dcfd4-bf80-410b-926	"lp

```
signals_copy = copy(signals) # we're gonna keep using `signals` afterwards, so let's
work with a copy
```

	recording	file_path	file
1	UUID("41595379-0887-4205-bef2-b2c54704	"/tmp/jl_Sflakp/41595379-0887-4205-bef	"lp
2	UUID("41595379-0887-4205-bef2-b2c54704	"/tmp/jl_Sflakp/41595379-0887-4205-bef	"lp
3	UUID("41595379-0887-4205-bef2-b2c54704	"/tmp/jl_Sflakp/41595379-0887-4205-bef	"lp
4	UUID("897dcfd4-bf80-410b-926c-68103c35	"/tmp/jl_Sflakp/897dcfd4-bf80-410b-926	"lp
5	UUID("897dcfd4-bf80-410b-926c-68103c35	"/tmp/jl_Sflakp/897dcfd4-bf80-410b-926	"lp
6	UUID("897dcfd4-bf80-410b-926c-68103c35	"/tmp/jl_Sflakp/897dcfd4-bf80-410b-926	"lp
7	UUID("0e190006-5939-454d-8d57-6146fd7d	"/tmp/jl_Sflakp/0e190006-5939-454d-8d5	"lp
8	UUID("0e190006-5939-454d-8d57-6146fd7d	"/tmp/jl_Sflakp/0e190006-5939-454d-8d5	"lp
9	UUID("0e190006-5939-454d-8d57-6146fd7d	"/tmp/jl_Sflakp/0e190006-5939-454d-8d5	"lp
10	UUID("3ae32bc3-b5d3-4ef0-b9c0-69780a07	"/tmp/jl_Sflakp/3ae32bc3-b5d3-4ef0-b9c	"lp

⋮ more

```
filter!(s -> s.recording != target #=|| (rm(s.file_path); false)=#, signals_copy)
```

Merge overlapping annotations of the same `quality` in the same recording. `merged` is an annotations table with a custom column of merged ids:

```
merged =
```

	recording	id	
1	UUID("0d8711e8-0ba0-4009-8528-3c9a3d41	UUID("ef009223-3360-4fcc-811d-15c42e9f	Tim
2	UUID("0d8711e8-0ba0-4009-8528-3c9a3d41	UUID("74ed5e4a-638f-4c2e-852e-1787aa99	Tim
3	UUID("2de1371b-2d2d-418b-a97c-c169b75a	UUID("84d87ecf-2e91-4fa1-a3c6-267542ea	Tim
4	UUID("2de1371b-2d2d-418b-a97c-c169b75a	UUID("0a4a38f5-4058-48d3-931c-918d7bdc	Tim
5	UUID("95306252-5c4d-49ed-9da3-c2ab6aef	UUID("772c5c44-c927-46e0-b0bc-9b0f2e05	Tim

	recording	id	
6	UUID("41595379-0887-4205-bef2-b2c54704	UUID("b821461d-bcc4-4f41-bbb2-fc3913e1	Tim
7	UUID("897dcfd4-bf80-410b-926c-68103c35	UUID("df53333f-9bdc-4ca2-bfc1-f738f733	Tim
8	UUID("dd4935a2-4719-423f-b900-340016dc	UUID("4224ab4d-bd5d-4cf9-aea7-8b0d3880	Tim

```
merged = DataFrame(mapreduce(merge_overlapping_annotations, vcat,
groupby(annotations, :quality)))
```

let's get the original annotation(s) from this merged annotation:

```
m =
```

DataFrameRow (4 columns)

	recording	id	span
	UUID...	UUID...	TimeSpa...
22	2de1371b-2d2d-418b-a97c-c169b75a0b6d	1bff464e-b7e4-4fd8-913a-a3a2128f8e5a	TimeSpan(00:00:44.000000000, 00:00:49.000000000)

```
m = rand(eachrow(merged))
```

	recording	id	
1	UUID("2de1371b-2d2d-418b-a97c-c169b75a	UUID("fafb16c6-1f49-4302-a910-b25b268d	TimeSpar

```
view(annotations, findall(in(m.from), annotations.id), :)
```

Load all the annotated segments that fall within a given signal's timespan:

```
within_signal (generic function with 1 method)
```

```
within_signal(ann, sig) = ann.recording == sig.recording &&
TimeSpans.contains(sig.span, ann.span)
```

```
sig =
```

DataFrameRow (11 columns)

recording	file_path
UUID...	String

	recording UUID...	file_path String
1	41595379-0887-4205-bef2-b2c5470446ab	/tmp/jl_Sflakp/41595379-0887-4205-bef2-b2c5470446ab_eeg.lpcm

```
sig = first(sig for sig in eachrow(signals) if any(within_signal(ann, sig) for ann in eachrow(annotations)))
```

	recording	id	
1	UUID("41595379-0887-4205-bef2-b2c54704	UUID("669f515a-c928-4556-9666-8f878f2d	TimeSpa
2	UUID("41595379-0887-4205-bef2-b2c54704	UUID("96bb8090-7365-4765-8a88-c9702a57	TimeSpa

```
transform(filter(ann -> within_signal(ann, sig), annotations),
          :span => ByRow(span -> load(sig, translate(span, -start(sig.span)))) => :samples)
```

In the above, we called `load(sig, span)` for each `span`. This invocation attempts to load *only* the sample data corresponding to `span`, which can be very efficient if the sample data file format + storage system supports random access and the full sample data file is very large. However, if random access isn't supported, or the sample data file is relatively small, or the requested set of `span`s heavily overlap, this approach may be less efficient than simply loading the whole file upfront. Here we demonstrate the latter as an alternative (note: in the future, we want to support an optimal batch loader):

```
samples =
Samples (00:09:00.000000000):
  info.kind: "eeg"
  info.channels: ["fp1", "f3", "c3", "p3", "f7", "t3", "t5", "o1", "fz", "cz", "pz", "f
  info.sample_unit: "microvolt"
  info.sample_resolution_in_unit: 1.0
  info.sample_offset_in_unit: 1
  info.sample_type: Int16
  info.sample_rate: 143.5 Hz
  encoded: false
  data:
19×77490 Array{Float64,2}:
  2.0   3.0   4.0   5.0   6.0   7.0   8.0  …  86.0  87.0  88.0  89.0  90.0  91.0
  3.0   4.0   5.0   6.0   7.0   8.0   9.0     87.0  88.0  89.0  90.0  91.0  92.0
  4.0   5.0   6.0   7.0   8.0   9.0  10.0     88.0  89.0  90.0  91.0  92.0  93.0
  5.0   6.0   7.0   8.0   9.0  10.0  11.0     89.0  90.0  91.0  92.0  93.0  94.0
  6.0   7.0   8.0   9.0  10.0  11.0  12.0     90.0  91.0  92.0  93.0  94.0  95.0
  7.0   8.0   9.0  10.0  11.0  12.0  13.0  …  91.0  92.0  93.0  94.0  95.0  96.0
  8.0   9.0  10.0  11.0  12.0  13.0  14.0     92.0  93.0  94.0  95.0  96.0  97.0
  ⋮                             ⋮          ⋱         ⋮
 15.0  16.0  17.0  18.0  19.0  20.0  21.0     99.0   0.0   1.0   2.0   3.0   4.0
 16.0  17.0  18.0  19.0  20.0  21.0  22.0      0.0   1.0   2.0   3.0   4.0   5.0
```

```
17.0  18.0  19.0  20.0  21.0  22.0  23.0  …   1.0   2.0   3.0   4.0   5.0   6.0
18.0  19.0  20.0  21.0  22.0  23.0  24.0      2.0   3.0   4.0   5.0   6.0   7.0
```

```
samples = load(sig)
```

	recording	id	
1	UUID("41595379-0887-4205-bef2-b2c54704	UUID("669f515a-c928-4556-9666-8f878f2d	TimeSpa
2	UUID("41595379-0887-4205-bef2-b2c54704	UUID("96bb8090-7365-4765-8a88-c9702a57	TimeSpa

```
transform(filter(ann -> within_signal(ann, sig), annotations),
          :span => ByRow(span -> view(samples, :, translate(span, -start(sig.span))))
=> :samples)
```

working with `Samples`

A `Samples` struct wraps a matrix of interleaved LPCM-encoded (or decoded) sample data, along with a `SamplesInfo` instance that allows this matrix to be encoded/decoded. In this matrix, the rows correspond to channels and the columns correspond to timesteps.

Let's grab a `Samples` instance for one of our mock EEG signals:

eeg_signal =
DataFrameRow (11 columns)

	recording	file_path
	UUID…	String
1	41595379-0887-4205-bef2-b2c5470446ab	/tmp/jl_Sflakp/41595379-0887-4205-bef2-b2c5470446ab_eeg.lpcm

```
eeg_signal = signals[findfirst(==("eeg"), signals.kind), :]
```

```
eeg =
Samples (00:09:00.000000000):
  info.kind: "eeg"
  info.channels: ["fp1", "f3", "c3", "p3", "f7", "t3", "t5", "o1", "fz", "cz", "pz", "f
  info.sample_unit: "microvolt"
  info.sample_resolution_in_unit: 1.0
  info.sample_offset_in_unit: 1
  info.sample_type: Int16
  info.sample_rate: 143.5 Hz
  encoded: false
  data:
19×77490 Array{Float64,2}:
  2.0   3.0   4.0   5.0   6.0   7.0   8.0  …  86.0  87.0  88.0  89.0  90.0  91.0
  3.0   4.0   5.0   6.0   7.0   8.0   9.0     87.0  88.0  89.0  90.0  91.0  92.0
  4.0   5.0   6.0   7.0   8.0   9.0  10.0     88.0  89.0  90.0  91.0  92.0  93.0
  5.0   6.0   7.0   8.0   9.0  10.0  11.0     89.0  90.0  91.0  92.0  93.0  94.0
  6.0   7.0   8.0   9.0  10.0  11.0  12.0     90.0  91.0  92.0  93.0  94.0  95.0
  7.0   8.0   9.0  10.0  11.0  12.0  13.0  …  91.0  92.0  93.0  94.0  95.0  96.0
  8.0   9.0  10.0  11.0  12.0  13.0  14.0     92.0  93.0  94.0  95.0  96.0  97.0
  ⋮                             ⋮          ⋱         ⋮
 15.0  16.0  17.0  18.0  19.0  20.0  21.0     99.0   0.0   1.0   2.0   3.0   4.0
 16.0  17.0  18.0  19.0  20.0  21.0  22.0      0.0   1.0   2.0   3.0   4.0   5.0
 17.0  18.0  19.0  20.0  21.0  22.0  23.0  …   1.0   2.0   3.0   4.0   5.0   6.0
 18.0  19.0  20.0  21.0  22.0  23.0  24.0      2.0   3.0   4.0   5.0   6.0   7.0
```

```
eeg = load(eeg_signal)
```

Here are some basic functions for examining `Samples` instances:

```
Test Passed
```

```
@test eeg isa Samples && !eeg.encoded
```

```
Test Passed
```

```
@test sample_count(eeg) == sample_count(eeg_signal, duration(eeg)) ==
index_from_time(eeg.info.sample_rate, duration(eeg)) - 1
```

```
Test Passed
```

```
@test channel_count(eeg) == channel_count(eeg_signal) == length(eeg.info.channels)
```

```
Test Passed
```

```
@test channel(eeg, "f3") == channel(eeg_signal, "f3") == findfirst(==("f3"),
eeg.info.channels)
```

```
Test Passed
```

```
@test channel(eeg, 2) == channel(eeg_signal, 2) == eeg.info.channels[2]
```

```
Test Passed
```

```
@test duration(eeg) == duration(eeg_signal.span)
```

Here are some basic indexing examples using `getindex` and `view` wherein channel names and sample-rate-agnostic `TimeSpan`s are employed as indices:

```
span = TimeSpan(00:00:03.000000000, 00:00:09.000000000)
```

```
span = TimeSpan(Second(3), Second(9))
```

```
span_range = 431:1291
```

```
span_range = index_from_time(eeg.info.sample_rate, span)
```

```
Test Passed
```

```julia
@test eeg[:, span].data == view(eeg, :, span_range).data
```

```
Test Passed
```

```julia
@test eeg["f3", :].data == view(eeg, channel(eeg, "f3"), :).data
```

```
Test Passed
```

```julia
@test eeg["f3", 1:10].data == view(eeg, channel(eeg, "f3"), 1:10).data
```

```
Test Passed
```

```julia
@test eeg["f3", span].data == view(eeg, channel(eeg, "f3"), span_range).data
```

```
rows_1 = ▶String["f3",  "c3",  "p3"]
```

```julia
rows_1 = ["f3", "c3", "p3"]
```

```
Test Passed
```

```julia
@test eeg[rows_1, 1:10].data == view(eeg, channel.(Ref(eeg), rows_1), 1:10).data
```

```
rows_2 = ▶Any["c3",  4,  "f3"]
```

```julia
rows_2 = ["c3", 4, "f3"]
```

```
Test Passed
```

```julia
@test eeg[rows_2, span].data == view(eeg, channel.(Ref(eeg), rows_2),
span_range).data
```

Note that `Samples` is not an `AbstractArray` subtype; the special indexing behavior above is only defined for convenient data manipulation. It is fine to access the sample data matrix directly via the `data` field if you need to manipulate the matrix directly or pass it to downstream computations.