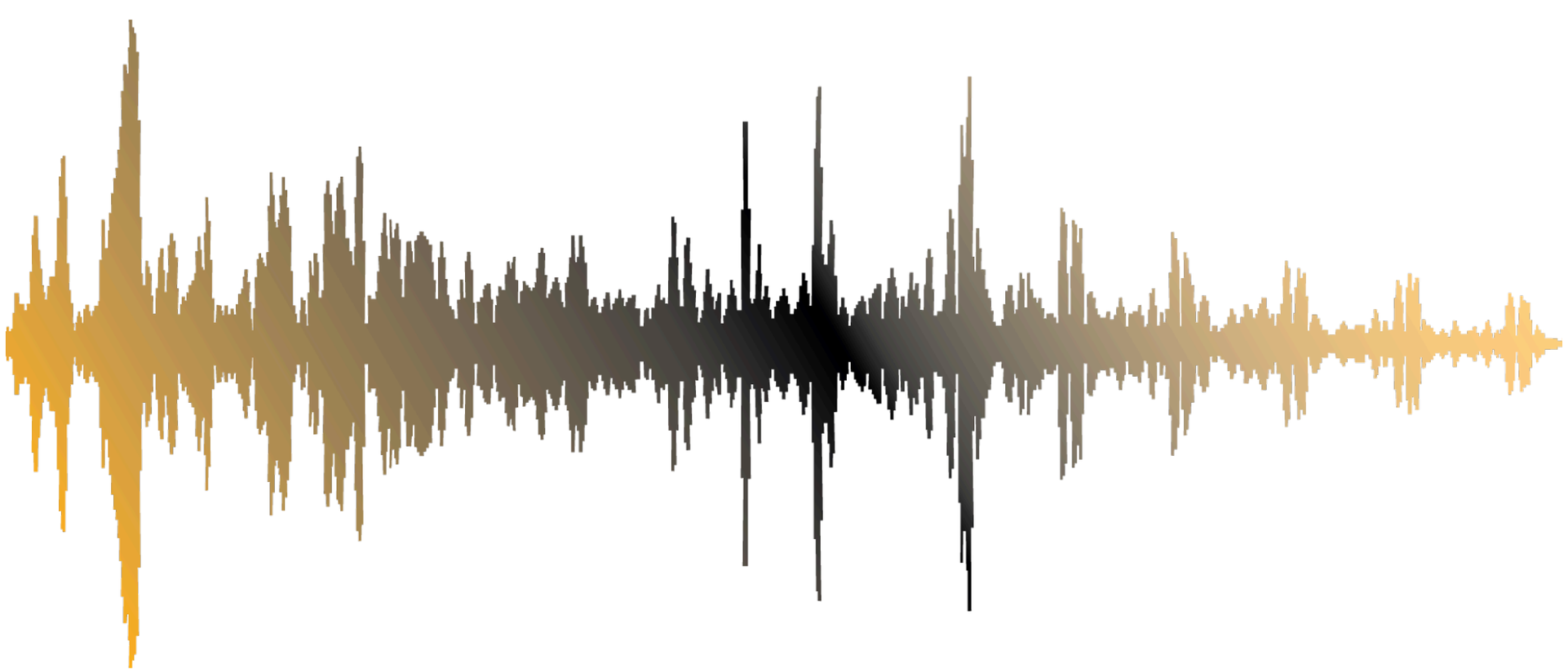
PrivacyStreams Walkthrough

https://privacystreams.github.io/

Running Example

Suppose developing a sleep monitor. The task is to get audio loudness every 10 minutes.



2 steps to access data

Step 1: Get data using UQI.

```
UQI uqi = new UQI(this.context);

uqi.getData(Audio.recordPeriodic(DURATION, INTERVAL), Purpose.HEALTH("monitoring sleep"))
        .debug()
```

Provider **Purpose**

Find a provider here*

*https://privacystreams.github.io/items.html

2 steps to access data

Step 2: Request permissions according to the provider

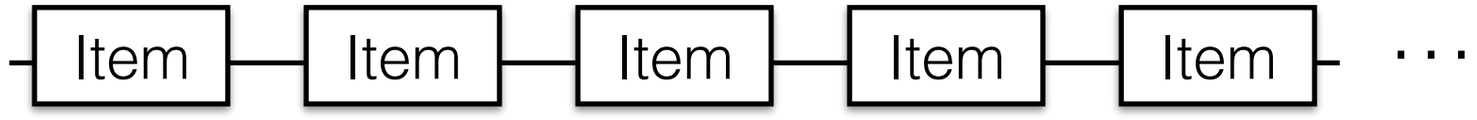
`MStreamProvider`	`Audio.recordPeriodic(long durationPerRecord, long interval)` Provide a live stream of Audio items. The audios are recorded from microphone periodically every certain time interval, and each Audio item is a certain duration of time long. For example, `recordPeriodic(1000, 4000)` will record audio from 0s-1s, 5s-6s, 10s-11s, ... This provider requires `Manifest.permission.RECORD_AUDIO` permission. - `durationPerRecord`: the time duration of each audio record, in milliseconds. - `interval`: the time interval between each two records, in milliseconds.

In AndroidManifest.xml:

```
<uses-permission android:name="android.permission.RECORD_AUDIO" />
```

Understanding Stream, Item, and Field

MStream
(multi-item stream)



SStream
(single-item stream)

Item

```
// An example of an Audio Item.
{
    Long "time_created": 1489528276655,
    Long "timestamp": 1489528266640,
    AudioData "audio_data": <AudioData@12416728>
}
```

Field

The format of the data you got

With *uqi.getData(…)*, you get a **Stream**.

To know the **Item** format in the stream:

- Option 1. Read documentation*

Reference	Name	Type	Description
`Audio.TIME_CREATED`	`"time_created"`	`Long`	The timestamp of when this item is created. It is a general field for all items.
`Audio.TIMESTAMP`	`"timestamp"`	`Long`	The timestamp of when the audio/record was generated.
`Audio.AUDIO_DATA`	`"audio_data"`	`AudioData`	The abstraction of audio data. The value is an `AudioData` instance.

- Option 2. Use `debug()` method (item will be printed in logcat)

```
UQI uqi = new UQI(this.context);
uqi.getData(Audio.recordPeriodic(DURATION, INTERVAL), Purpose.HEALTH("monitoring sleep"))
        .debug()
```

*https://privacystreams.github.io/items.html

2 steps to process data

Step 1. Find proper Transformations here*

For example, `setField` can create a new field

`MStream->MStream`		`setField(String fieldToSet, Function<Item,TValue> functionToComputeValue)` Set a field to a new value for each item in the stream. The value is computed with a function that take the item as input. Eg. `setField("x", Comparators.gt("y", 10))` will set a new boolean field “x” to each item, which indicates whether the “y” field is greater than 10. - `fieldToSet`: the name of the field to set, it can be a new name or an existing name. - `functionToComputeValue`: the function to compute the new field value - `<TValue>`: the type of the new field value

How to use:

```
UQI uqi = new UQI(this.context);
uqi.getData(Audio.recordPeriodic(DURATION, INTERVAL), Purpose.HEALTH("monitoring sleep"))
        .setField("loudness", xxx)
        .debug()
```

*https://privacystreams.github.io/pipeline.html

2 steps to process data

Step 2. Find proper operators here**

For example, `calcLoudness` operator:

`Function<Item,Double>`	`AudioOperators.calcLoudness(String audioDataField)` Calculate the average (RMS) loudness of the audio specified by an AudioData field. The loudness is an double number indicating the sound pressure level in dB. - `audioDataField`: the name of the AudioData field.

To use it with `setField` transformation:

```
UQI uqi = new UQI(this.context);
uqi.getData(Audio.recordPeriodic(DURATION, INTERVAL), Purpose.HEALTH("monitoring sleep"))
        .setField("loudness", AudioOperators.calcLoudness("audio_data"))
        .debug()
```

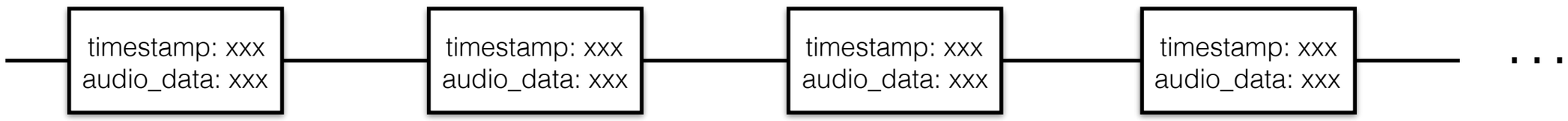
New field type: Double

Existing AudioData-type field

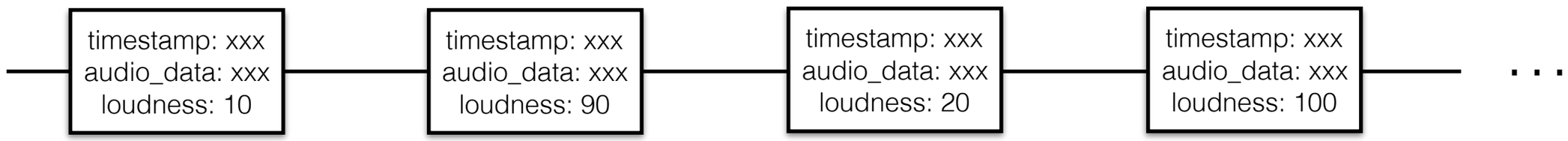
**https://privacystreams.github.io/operators.html

Demonstration

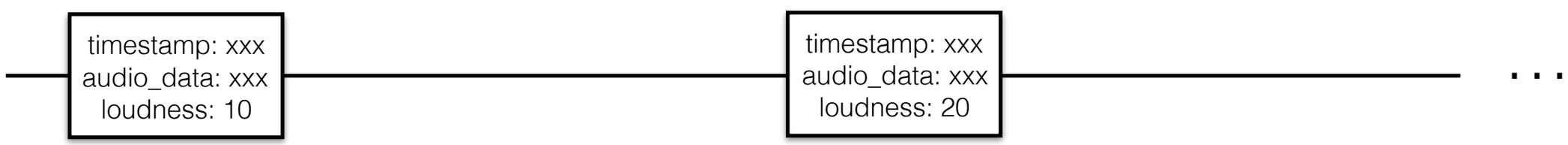
```
uqi.getData(Audio.recordPeriodic(DURATION, INTERVAL), Purpose.HEALTH("monitoring sleep"))
```




```
    .setField("loudness", AudioOperators.calcLoudness("audio_data"))
```




```
    .filter(Comparators.lt("loudness", 80))
```



2 steps to output data

Step 1. Find a proper Action here*

For example, the `forEach` action:

`MStream->void`	Non-blocking	`forEach(String fieldToSelect, Callback<TValue> callback)` Callback with a certain field of each item. - `fieldToSelect`: the name of the field to callback with - `callback`: the callback to invoke for each item field - `<TValue>`: the type of the field

How to use:

```
UQI uqi = new UQI(this.context);
uqi.getData(Audio.recordPeriodic(DURATION, INTERVAL), Purpose.HEALTH("monitoring sleep"))
        .setField("loudness", AudioOperators.calcLoudness("audio_data"))
        .forEach("loudness", xxx)
```

*https://privacystreams.github.io/pipeline.html

2 steps to output data

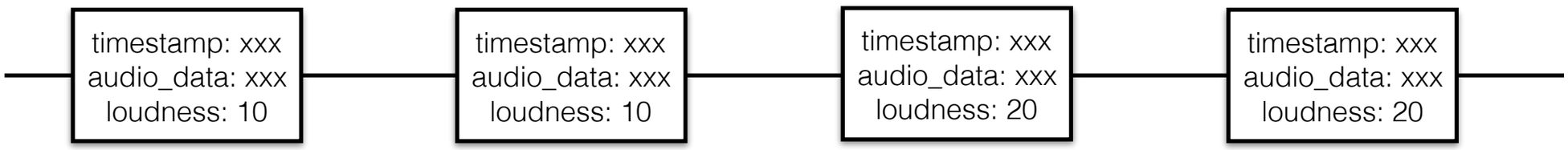
Step 2. Handle result.

- In return value (blocking).
  - Eg. asList(), asList("fieldName"), getField("fieldName")
- In callback (non-blocking).
  - Eg. forEach(…), ifPresent("fieldName", …), onChange("fieldName", …)

```
UQI uqi = new UQI(this.context);
uqi.getData(Audio.recordPeriodic(DURATION, INTERVAL), Purpose.HEALTH("monitoring sleep"))
        .setField("loudness", AudioOperators.calcLoudness("audio_data"))
        .forEach("loudness", new Callback<Double>() {
            @Override
            protected void onInput(Double loudness) {
                System.out.println("Current loudness is " + loudness + "dB.");
            }
        });
```

Demonstration

```
uqi.getData(Audio.recordPeriodic(DURATION, INTERVAL), Purpose.HEALTH("monitoring sleep"))
        .setField("loudness", AudioOperators.calcLoudness("audio_data"))
        .limit(4)
```




```
.forEach("loudness", callback)
      callback(10),       callback(10),       callback(20),       callback(20)

.ifPresent("loudness", callback)
      callback(10)

.onChange("loudness", callback)
      callback(10),                           callback(20)

.asList(“loudness") // blocking
      [10, 10, 20, 20]
```

Summary

- 2 steps for accessing data:
  - Get data using UQI and a Provider
  - Request permissions according to the provider
- 2 steps for processing data:
  - Find proper Transformations
  - Find Operators as the parameters of Transformations
- 2 steps for outputting data:
  - Find a proper Action (blocking or non-blocking)
  - Handle result in the return value or in a callback
- More examples can be found on our webpage.