docs.audioblast.org

Ed Baker

2023-03-03

Contents

Chapter 1

About audioBlast

audioBlast is a project to make the world's collections of bioacoustic and ecoacoustic recordings findable, accessible, and searchable.

Chapter 2

Introduction

2.1 Design Philosophy

The technical design of audioBlast pushes computation down the stack as far as is conveniently possible. Ideally this will often be the database. This results in a reduction in network traffic, which can be significant if returning a large number of analyses.

Some core parts of the audioBlast system (i.e. the ingest process and analyse process) can communicate directly with the database, and all insertion or updating of records must be performed through one of these processes.

In general, and for all end-user cases, reading of data should be performed via the API (https://api.audioblast.org). The API is a lightweight database wrapper that automatically handles pagination of results and caching of slow queries. Various utilities provide access to the API in different environments, including the sonicscrewdriver package for the R environment. The API is also used to provide the data search and discovery tools at https://audioblast.org.

Table 2.1: Simplified illustration of audioBlast abstraction layers from bottom to top.

	Ingest	Analyse
MySQL	Triggers INSERT UPDATE	Stored Procedures INSERT U
R	r-dbi abdb	r-dbi abdb
API		
Users		

2.2 Analyse everything once, in advance

Many bioacoustic and ecoacoustic studies make use of a standard set of analyses. These are computed in advance by audioBlast, allowing the analyses to be instantly available and searchable. An additional advantage is that the computational resource needed to compute analyses need only be performed once.

Chapter 3

audioBlastIngest

What is it

3.1 Structure

Chapter 4

audioBlastAnalyse

audioBlastAnalyse is an R package that handles the analysis of recordings. While some analyses (e.g. annotations) can be imported during the ingest process this is not true for all sources of recordings.

4.1 Requirements

The analysis system requires a computer running the R environment for statistical computing. The package will import all dependencies on installation.

The package is tested on recent MacOS, Linux, and Windows systems.

If the analysis system is to write directly to the audioBlast database then it must additionally be on the NHM network (including via the VPN).

4.2 Development

The package is hosted on the audioBlast GitHub at https://github.com/audioblast/audioBlastAnalyse.

4.3 Installation

Installation of the package requires the devtools package.

```
install.packages("devtools")
```

4.3.1 From GitHub

```
devtools::install_github("audioblast/audioBlastAnalyse")
```

4.3.2 From R working directory

```
devtools::install()
```

Chapter 5

The Task Queue System

The Task Queue System maintains a list of outstanding analysis tasks in the database and is responsible for assigning tasks to analysis processes.

5.1 Implementation

The system is implemented as two tables in the database, `tasks` and `tasks-progress`. Tasks are assigned using the stored procedure `get-tasks()` for local files, or `get-tasks-by-file()` for web files. The `get-tasks()` procedure will assign `n` random tasks to the analysis process, whereas `get-tasks-by-file()` will return all outstanding tasks for a given file. The latter procedure removes the need for repeated downloads of the same file.

5.1.1 `tasks` table

The table `tasks` is a list of outstanding analysis tasks. It is populated automatically by triggers on the `recordings` and `recordings-calcuated` table. Analysis tasks are grouped into sections (recordings_calculated, sounscapes_minute and soundscpaes_second). The `recordings-calculaed` table has a TINYINT column for each of these, that is set to `1` once the task is completed.

5.1.2 `tasks-progress` table

The table `tasks-progress` is populated by the stored procedure `get-tasks()` when an analysis process requests outstanding tasks. It is used to prevent the same task being analysed to multiple analysis processes. Tasks from an unresponsive process (e.g. a process which has crashed) can be reassigned after an hour of inactivity by that process.

5.1.3 `get-tasks()` stored procedure

The stored procedure `get-tasks(process_id, n, source)` is used to assign `n` tasks to the analysis process `process_id`. Initially a call to `_quickMaintain()` is performed to deallocate time-expired tasks. It inserts `n` random unassigned tasks from the `tasks` table into the `tasks-progress` table along with the process_id and the current timestamp. It returns via a SELECT statement these tasks.

5.1.4 `get-tasks-by-file()` stored procedure

Initially a call to `_quickMaintain()` is performed to deallocate time-expired tasks. The stored procedure `get-tasks(process_id, source)` then assigns all outstanding tasks for a randomly chosen file to the analysis process `process_id`. It inserts these tasks from the `tasks` table into the `tasks-progress` table along with the process_id and the current timestamp. It returns via a SELECT statement these tasks.

5.1.5 `delete-task()` stored proecure

The stored procedure `delete-task(process_id, source, id, task)` marks a task as complete by deleting it from the `tasks` table. A trigger in the `tasks` table will delete the matching row from `tasks-progress`. The started time of any remaining tasks assigned to the same `process_id` in the `tasks-progess` table will be updated to the current time. A call is made to the stored procedure `_quickMaintain()` which will perform routine maintenance tasks, including removing any expired tasks in the `tasks-progress` table.