geordi Documentation

Release 0.2

MetaBrainz Foundation

January 16, 2015

Contents

Geordi is a tool for storing and making available/usable general data dumps for use by the MusicBrainz community, for example data dumps from labels and production agencies, as well as public data sources such as Discogs and Jamendo.

CHAPTER 1

Contents

1.1 Geordi data structure

This document will provide a short conceptual overview of the various objects that make up Geordi, and their relations.

1.1.1 Geordi objects

Item

The core object of geordi is the 'item'. This is geordi's overarching container for entities of all sorts: releases, artists, labels, places, areas, whatever. Items have an *id*, a nominal *type* (roughly: "we think this is a release" or "we think this is an artist"), and a *map* – that is, data in a standard format defining the entity.

Data Item

Conceptually contained by items are what I'll call "data items": a bit of data from a specific source with a specific identifier relative to that source. This includes chunks of data derived from broader items, for example in a data source that only provides data files for releases, it is still often possible to extract information identifying and describing an artist, or a recording. Data items have an *id* composed of a data source (called 'index'), an item type, and an identifier (perhaps an integer, perhaps a catalog number, perhaps just a name – depends on the source), a link to an item, and the data itself.

Upon initial insertion to geordi, each item contains exactly one data item. Eventually it should be possible to merge items together (or, perhaps more accurately, match items within geordi to each other – for example, the same artist as represented in two different data sources). Data items cannot be directly linked to anything except items, since items represent the links between data items other than equivalence. However, data items carry most of the actual data in geordi, so many things are derived from them.

Editor

The third basic object that doesn't represent some sort of relation is an editor. Editors represent both human users who log in via MB and automatic processes indigenous to geordi.

1.1.2 Geordi relations

Item Links

Aside from the mapped data, another thing that can be extracted from data items is connections between pieces of data within a dataset. For example, a dataset with identifiers for both release and recording objects will probably include a list of references to recordings in its data for the release. Upon mapping, these are extracted into item links in geordi. Item links are composed of an item being linked from, a linked item (the other end of the link), and a type. The type is written as a path into the item's mapped data, separated by '%' characters, for example a type of release%artists%split%names%1 for the second artist listed at the release level.

Raw Matches

More complicated, but rather important, are links to entities within MusicBrainz. These present a lot of complication for the simple reason that they can often be represented, in various ways, from both geordi and from MusicBrainz. To simplify things somewhat, I'll first talk about *raw matches*: matches stored within geordi, that have no equivalent inverse way to be stored in MusicBrainz (perhaps a dataset derived from a private site that can't be linked to with URL relationships). These consist of an item, a set of mbid + type pairs, a user, and a timestamp. For the sake of storing history, raw matches can also be marked as 'superseded', when a new match is registered instead.

MusicBrainz Matches

(Note: not yet implemented). For data sources where relationships within MusicBrainz itself are present, it would be best not to duplicate that information and potentially let it get out of sync. So, ultimately, geordi should also be able to pull matches from MusicBrainz's data. This would presumably be done by way of either including a replicated MusicBrainz database alongside geordi, which it can then query, or as some sort of materialization process.

Automatic Matching

(Note: not yet implemented). Matches of various sorts may be suggested or created by automated process indigenous to geordi; for example, if a release is already matched to MusicBrainz and has only one artist in both databases, it's relatively likely the two artists are the same. Matches created by such a process are referred to as automatic matches.

1.1.3 Technical details

Geordi's data is stored in a PostgreSQL database. Further details of the schema and interconnections are documented in the *Database and Schema* document, or can be investigated by reading the SQLAlchemy model definitions or creating a geordi database and using PostgreSQL's built-in introspection tools.

1.2 Geordi data lifecycle/terminology

As data moves in and through geordi it goes through several stages. This document explains the various steps and introduces the terminology used to refer to each step.

1.2.1 Importing (to geordi)

The process of putting data into geordi. This step is done by an admin of the geordi installation, using commands provided by manager.py. In general, importing will use *importers*, which are implemented in the

geordi/data/importer/indexes directory. Should data be updated, or mappings be updated, reimporting the same data performs an update.

1.2.2 Mapping

The process of turning data in raw form into geordi's standard mapping format. This step is performed as part of importing, above, but is mentioned separately as it's separated in the codebase. Mappings are defined in the geordi/data/mapping/indexes, per index and item type. Mappings are defined declaratively using rules specifying a source in the raw JSON data and a destination, along with, potentially, item links, conditions, transformations, and so-called "blank node" destinations. Mapping is discussed separately in the *Mapping* document.

1.2.3 Matching

After importing, items that are already in MusicBrainz may be marked as such by way of matching. This step is performed by users, through the web interface. (not yet implemented)

1.2.4 Seeding (importing to MusicBrainz)

Geordi's mapped data can be converted into a format that allows adding it to MusicBrainz. Seeding is initiated by users who wish to add an item not already in MusicBrainz to it, or update some data in MusicBrainz. (not yet implemented)

1.2.5 Merging and Splitting

Since multiple items from several indexes can represent the same object, they can be marked as the same thing and thus merged, by way of an interface to specify their connection (potentially via intermediate objects such as release groups, areas, etc.). Likewise, it should be possible to split items by assigning their data items, links, and matches to two sides. (not yet implemented)

1.3 Database and Schema

1.3.1 Database and Helpers

geordi.data.model

Geordi model grouping module and general database-management tools.

`geordi.data.model.`**`create_tables`**`(`*app*`)`
Initialize tables in the database. Assumes the database already exists and a 'geordi' schema is created.

`geordi.data.model.`**`db = <SQLAlchemy engine=None>`**
A shared SQLAlchemy object for use across geordi. Intended to be imported from views, models, etc.

geordi.data.model.mixins

Helpful mixin classes for commonalities between models.

class `geordi.data.model.mixins.`**`DeleteMixin`**
Provides a 'delete' method deleting an object from the DB.

delete()
Delete this object from the DB.

1.3.2 Models

Core data

geordi.data.model.item

class geordi.data.model.item.**Item**(***kwargs*)
Bases: flask_sqlalchemy.Model, geordi.data.model.mixins.DeleteMixin

Model for the 'item' table, storing item type information and mapped data.

id
Integer ID of this item.

type
Nominal type guess for this item.

map
Mapped data for this item, JSON.

item_data
Property for data items linked to this item. Included by default when loading.

item_redirects
Property for redirects to this item.

item_links
Property for links from this item to other items. Included by default when loading.

items_linked
Property for links from other items to this item. Not loaded by default.

raw_matches
Property for matches of this item to MusicBrainz entities.

map_dict

to_dict()

classmethod get(*item_id*, ***kwargs*)

classmethod create(*type=None*, *map=None*)

geordi.data.model.item_data

class geordi.data.model.item_data.**ItemData**(***kwargs*)
Bases: flask_sqlalchemy.Model, geordi.data.model.mixins.DeleteMixin

Model for the 'item_data' table, storing index-specific raw data.

id
Data item identifier of the form (index)/(item type)/(identifier).

item_id
Item ID of the item to which this data item belongs.

data
Raw JSON data.

to_dict()

classmethod get(*id*, ***kwargs*)

classmethod get_by_item_id(*item_id*, ***kwargs*)

classmethod data_to_item(*data_id*)
Resolve a data ID to its associated item ID, if it has one (it should).

classmethod create(*item_id*, *data_json*, *data_id*)

classmethod update(*item_id*, *data*, *data_id*)

static get_indexes()

static get_item_types_by_index(*index*)

static get_item_ids(*index*, *item_type*)

static delete_data_item(*data_id*)

geordi.data.model.item_link

class geordi.data.model.item_link.**ItemLink**(***kwargs*)
Bases: flask_sqlalchemy.Model, geordi.data.model.mixins.DeleteMixin

Model for the 'item_link' table, storing automatically-extracted links between items.

type
Type of this link, expressed as a path into the mapped JSON data of the source item, joined by '%' characters.

item_id
Item ID of the source side of this link.

linked_id
Item ID of the target side of this link.

value

to_dict()

classmethod get(*type*, *item_id*, *linked_id*, ***kwargs*)

classmethod get_by_item_id(*item_id*, ***kwargs*)

classmethod find_or_insert(*node_item_id*, *target_item_id*, *link_type*)

classmethod delete_by_item_id(*item_id*, ***kwargs*)

geordi.data.model.item_redirect

class geordi.data.model.item_redirect.**ItemRedirect**(***kwargs*)
Bases: flask_sqlalchemy.Model, geordi.data.model.mixins.DeleteMixin

Model for the 'item_redirect' table, storing the old IDs of items that have been merged.

old_id
The obsolete item ID which should be redirected.

new_id
The item ID to which it should be redirected.

Matches and entities

geordi.data.model.raw_match

class geordi.data.model.raw_match.**RawMatch**(***kwargs*)
Bases: flask_sqlalchemy.Model, geordi.data.model.mixins.DeleteMixin

Model for the 'raw_match' table, storing matches between items and MusicBrainz entities.

id
A unique ID for the match.

item_id
The item ID for the item this match is for.

editor_name
The editor name for the editor who created this match.

timestamp
The time when the match was made.

superseded
Boolean, false if this match is current, true if it should be considered historical only.

entities
Property for raw match entities linking this match to MusicBrainz entities.

classmethod get_by_item(*item_id*, ***kwargs*)

classmethod match_item(*item_id*, *editor_name*, *entities*)

to_dict()

geordi.data.model.raw_match_entity

class geordi.data.model.raw_match_entity.**RawMatchEntity**(***kwargs*)
Bases: flask_sqlalchemy.Model, geordi.data.model.mixins.DeleteMixin

Model for the 'raw_match_entity' table, storing the link between matches and materialized entity data.

raw_match_id
The ID of the match.

entity_mbid
The MBID of the entity.

geordi.data.model.entity

class geordi.data.model.entity.**Entity**(***kwargs*)
Bases: flask_sqlalchemy.Model, geordi.data.model.mixins.DeleteMixin

Model for the 'entity' table, storing materialized information about entities in MusicBrainz.

mbid
The entity's MBID.

type
The type of entity (e.g. 'release', 'release_group', 'artist', etc.

data
Materialized JSON data used to display a link (e.g. the name or title).

raw_match_entities
Property for matches using this entity.

classmethod get(*mbid*, ***kwargs*)

classmethod get_remote(*mbid*, ***kwargs*)

merge_into(*target*)

to_dict()

Editors and login

geordi.data.model.csrf

class geordi.data.model.csrf.**CSRF**(***kwargs*)
Bases: flask_sqlalchemy.Model, geordi.data.model.mixins.DeleteMixin

Model for the 'csrf' table, storing information about users who may be logging in via MusicBrainz OAuth.

ip
The user's IP, after stripping away known/trusted proxies.

csrf
The random csrf value passed as the 'state' to MusicBrainz's OAuth.

opts
JSON representation of user options, e.g. 'remember me' and the URL to return to.

timestamp
Timestamp for when this was allocated, used for automatic removal.

classmethod get(*csrf*, ***kwargs*)

classmethod update_csrf(*ip*, *rand*)
Called with an ip and a random value to be used as a csrf. Inserts this into the DB and automatically deletes other values for this IP older than 1 hour.

geordi.data.model.editor

class geordi.data.model.editor.**Editor**(***kwargs*)
Bases: flask_sqlalchemy.Model, geordi.data.model.mixins.DeleteMixin

Model for the 'editor' table, storing information about both human users and automated processes which match items.

name
Editor name from MusicBrainz, or a descriptive name for an automated process.

tz
For human editors, timezone preference.

internal
Boolean; True for automated processes, otherwise False.

matches
Property for matches entered by this user. Not loaded by default.

classmethod get(*name*, ***kwargs*)

classmethod add_or_update(*name*, *tz=None*)
Given a name and optionally a timezone, either insert a new row, update the timezone preference, or do nothing, depending on what's already stored.

1.4 Mapping

Mapping in geordi is done by way of an array of rule definitions, looked up by the data source name (i.e. index name) and item type (for example, ninjatune/release, or test_index/artist). The rules are within geordi/data/mapping/indexes/, in a file by index name, which exports a dictionary mapping item type to a list of rules.

The rule language is defined by a set of other files within geordi/data/mapping, especially geordi/data/mapping/rule.py and geordi/data/mapping/extract.py; in general, rules specify a path within the source data (potentially including some choices to be made by the extractor, e.g. to continue within every element of an array, or to continue along some keys of a dictionary but not others) and a destination path to which the data should be mapped. Though there are defaults, rules are generally expected to provide a condition (i.e., which values to accept, defaulting to all) and an ordering value (i.e., where in the result array to put the value, defaulting to None to specify the order doesn't matter).

Rules are also permitted to provide information about links (to other data items; the types of these links will be determined by the destination path of the rule), a function to transform the data before insertion, and may specify a node to which the destination applies (defaulting to "the node corresponding to the data item being mapped"; this functionality is to be used when an index provides information about linked entities within the data for another entity, but does not provide separate information about the linked entity, e.g. provides artist type information in a release listing).

For more information, it's recommended to look at the various indexes' mapping rules.

1.4.1 Weaknesses

- It's not currently possible to have a rule depend on multiple locations in the source data – while choices can be made, if e.g. it's possible for the index to provide more than one track title and more than one track title version for each track, a cross product can't be generated to e.g. have the track name suggestions be ["title 1 (version 1)", "title 2 (version 1)", "title 1 (version 2)", "title 2 (version 2)"]. This could be fixed by making it possible for rules to generate multiple values, and using a 'transform' function at a higher level in the tree.

1.5 Mapping Format

The mapping format is described by a JSON Schema document in the repository.

One bit of note is that some places MusicBrainz has joint fields that aren't joint in other data sources. For a relatively complete example, take artists: depending on the source, they might be provided as MusicBrainz-type artist + credit + join phrase sets, or as artists independently, or as a complete rendered credit without splitting, or a combination of the above (a list of artists plus a complete rendered credit is especially common). Other cases include mediums and their contained tracks/formats, labels and catalog numbers, and release events/dates/countries of release. For these cases, the mapping format includes 'combined', 'split', and 'unsplit' formats: 'combined' is the target format (like MusicBrainz), 'split' is for separated lists (such as when a list of labels and a list of catalog numbers are provided, but they aren't paired), and 'unsplit' is when the data is a single list, but more joined together than the final form (such as a list of fully-rendered artist credits).

1.6 API

GET /api/1/data
Get list of indexes.

Response Headers

- **Content-Type** – *application/json*

GET /api/1/data/ (*index*) **/** *item_type***/path:** *data_id* Get item ID based on specified index, item type, and data ID.

Response Headers

- **Content-Type** – *application/json*

GET /api/1/item/ (**int:** *item_id*) **/matches**

GET /api/1/item/ (**int:** *item_id*) **/links**
Get links to specified item.

Response Headers

- **Content-Type** – *application/json*

GET /api/1/data/ (*index*) **/**
item_type Get list of items for a specified index and item type.

Response Headers

- **Content-Type** – *application/json*

GET /api/1/item/ (**int:** *item_id*)
Get item's data, links, and map.

Response Headers

- **Content-Type** – *application/json*

GET /api/1/data/ (*index*)
Get list of item types for a specified index.

Response Headers

- **Content-Type** – *application/json*

/api

GET /api/1/data, 11
GET /api/1/data/(index), 11
GET /api/1/data/(index)/(item_type), 11
GET /api/1/data/(index)/(item_type)/(path:data_id),
    11
GET /api/1/item/(int:item_id), 11
GET /api/1/item/(int:item_id)/links, 11
GET /api/1/item/(int:item_id)/matches,
    11

g

A

C

D

E

F

G

I