Niamoto core Documentation

Release 0.1

Dimitri Justeau

Jun 02, 2019

Contents:

Important: Niamoto is currently in active development and thus not ready for production use. If you are interested in the project and willing to contribute or collaborate feel free to contact.

Niamoto is an ecological data warehouse framework, it aims to provide a flexible and efficient platform for decision support in ecosystems and biodiversity conservation.

Niamoto is funded by the COGEFOR projet in New Caledonia, which is a partnership between the North Province of New Caledonia, the New Caledonian Agronomic Institute (IAC) and the CIRAD. The AMAP joint research unit and the IRD are also implied in the project.

Contents:

CHAPTER 1

Project overview

1.1 What is Niamoto?

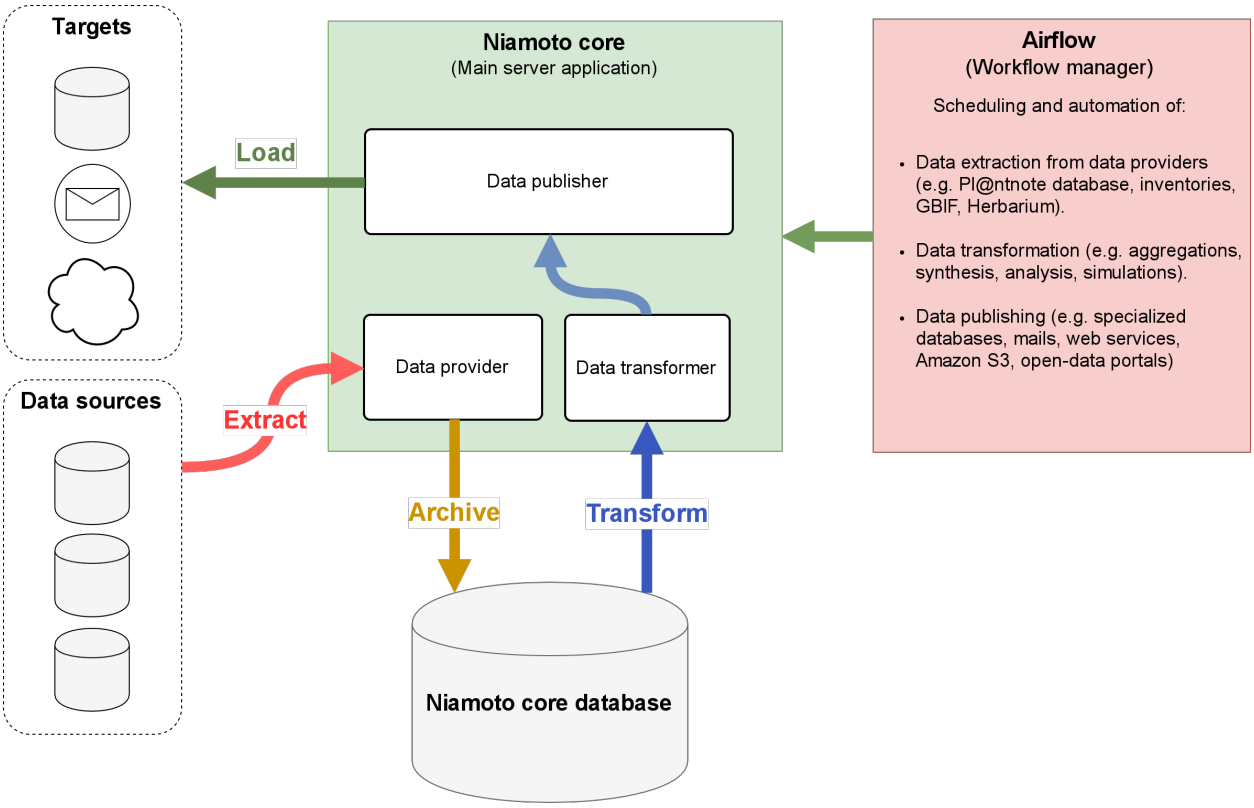
Niamoto is an ecological data warehouse framework designed to fill the gap between scientists and decision makers. It is currently being used and developed for tropical rainforest ecosystems in New Caledonia, but had been designed generic enough to be adapted for other ecosystems and contexts.

Niamoto's objective is to provide a platform for aggregating ecological data from several data sources, defining and running scientific data workflows and distributing value-added data in several formats, for several purposes. Niamoto provides ETL (Extract, Transform, Load) functionalities, a structured and persistent staging area and functionalities to publish data marts. Niamoto data marts can be seamlessly loaded into the **cubes** OLAP framework.

Niamoto comes with a command-line interface (CLI), and a Python API.

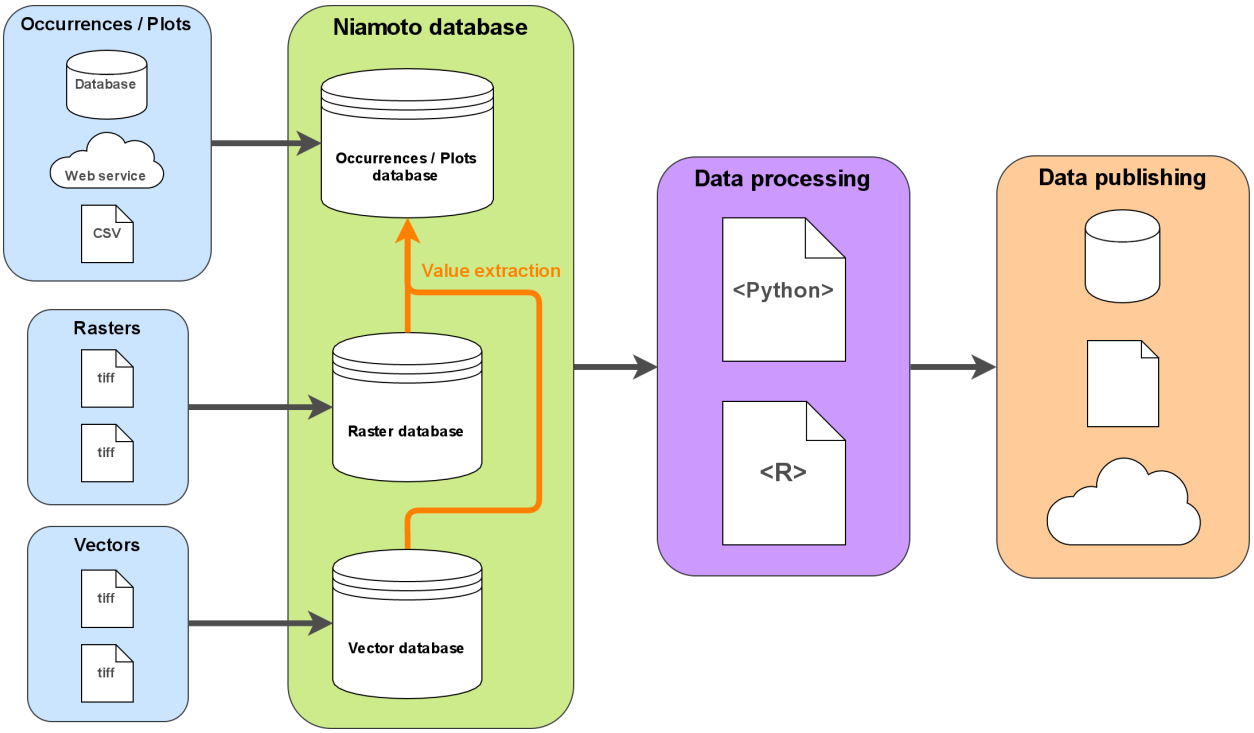
1.2 Niamoto structure

Bellow is a diagram of how Niamoto is structured in the main lines, and how it interacts with data sources and targets:



1.3 Niamoto data flow

Here a diagram describing the Niamoto data flow in the main lines:


Occurrences / Plots
Database
Web service
CSV
Rasters
tiff
tiff
Vectors
tiff
tiff
Niamoto database
Occurrences / Plots database
Value extraction
Raster database
Vector database
Data processing
<Python>
<R>
Data publishing

CHAPTER 2

Installation

2.1 Prerequisites

2.1.1 Operating System

Niamoto had been developed and tested to run under **Linux** systems. However, it had been developed with open-source and cross-platform technologies and it should then be possible to set up a Niamoto instance on **Mac** or **Windows**. If you succeeded to do so, feel free to contribute to this doc (see *Contributing*).

2.1.2 Python 3

Niamoto is written in Python 3. It is tested with Python 3.4, 3.5 and 3.6. You must have one of the following installed in your system, in addition to `pip`.

2.1.3 PostgreSQL / PostGIS

Niamoto uses PostgreSQL (>= 9.5) with its spatial extension PostGIS (>= 2.1) as a backend for constituting the data warehouse and storing the data. It implies that your Niamoto instance must be granted access to a PostgreSQL / PostGIS instance, whether it is distant or on the same system.

Important: PostgreSQL version must be **at least** 9.5, since Niamoto uses JSONB capabilities that had been released with PostgreSQL 9.5.

The installation procedure can change according to your system. Please refer to https://www.postgresql.org/download/ and http://postgis.net/install/ .

2.1.4 gdal-bin and libgdal-dev

Niamoto dependencies require that `libgdal-dev` and `gdal-bin` are installed in your system. The installation is straightforward:

```
sudo apt-get install -y libgdal-dev gdal-bin
```

2.1.5 Git

In order to clone the Niamoto repository, Git must be installed in your system:

```
sudo apt-get install -y git
```

2.2 Install Niamoto

First, clone the Niamoto repository in your system:

```
git clone https://github.com/dimitri-justeau/niamoto-core.git
```

Then, move into the project directory and install Niamoto using pip:

```
pip install .
```

2.3 Setting up the Niamoto environment

2.3.1 Setting up the Niamoto database

Note: For more options with the Niamoto database, please refer to *Configuration*.

1. Create Database and Database User

First, change the current Linux user to `postgres`:

```
sudo su postgres
```

Then, log into PostgreSQL:

```
psql
```

Create the Niamoto database (default name is `niamoto`, see *Configuration* for more details):

```
CREATE DATABASE niamoto;
```

Then, create the Niamoto user and grant full access to Niamoto database to it (to ensure a secure instance, you must change at least the default user password see *Configuration* for more details):

```
CREATE USER niamoto WITH PASSWORD niamoto;
GRANT ALL PRIVILEGES ON DATABASE niamoto TO niamoto;
```

Finally, logout with `\q`.

2. Create PostGIS extension and niamoto schema

Log into PostgreSQL, with `postgres` user and `niamoto` database:

```
psql -d niamoto
```

Create the PostGIS extension:

```
CREATE EXTENSION POSTGIS;
```

Logout with `\q`.

3. Create Database Schemas

Log into PostgreSQL, with `niamoto` user and `niamoto` database:

```
psql -U niamoto -d niamoto
```

Create the `niamoto`, `niamoto_raster`, `niamoto_vector`, `niamoto_dimensions`, `niamoto_fact_tables` schemas (see *Configuration* for more details and options):

```
CREATE SCHEMA niamoto;
CREATE SCHEMA niamoto_raster;
CREATE SCHEMA niamoto_vector;
CREATE SCHEMA niamoto_dimensions;
CREATE SCHEMA niamoto_fact_tables;
```

Logout with `\q`.

2.3.2 Initializing the Niamoto home directory

Note: For more options with the Niamoto home directory, please refer to *Configuration*.

Niamoto home is the place where configuration files, scripts and plugins will be stored. Niamoto comes with a handy command for initializing it:

```
niamoto init_niamoto_home
```

2.3.3 Initializing the Niamoto database

Initializing the Niamoto database means creating the tables, indexes, constraints and initializing basic data. The procedure is straightforward:

```
niamoto init_db
```

2.3.4 What's next?

At this point, you should have a working Niamoto environment. If you are ready to play, you can go to the *Quickstart* of the *Tutorial*!

CHAPTER 3

Quickstart

3.1 Setting the taxonomy

The Niamoto's taxonomic referential is set using the `set_taxonomy` command:

```
$ niamoto set_taxonomy taxonomy.csv
Setting the taxonomy...
The taxonomy had been successfully set!
    4 taxa inserted
    2 synonyms inserted: {'taxref', 'gbif'}
```

3.2 Adding data providers and importing data

Plots and occurrences data is imported registered data providers and syncing with them.

```
$ niamoto add_provider csv_gbif CSV gbif
  Registering the data provider in database...
  The data provider had been successfully registered to Niamoto!
```

It is possible to see the registered providers using the `niamoto providers` command:

```
$ niamoto providers
           name provider_type synonym_key
id
1      csv_gbif           CSV        gbif
```

Importing data using the csv data provider is done with three csv files:

- The **occurrences** csv file, containing the occurrence data.
- The **plots** csv file, containing the plot data.
- The **plots/occurrences** csv file, mapping plots with occurrences.

All of them are optional, you can import only occurrences, only plots or only map existing plots with existing occurrences. The command for importing data from a provider is `niamoto sync PROVIDER_NAME [PROVIDER_ARGS]`. With the csv data provider, three arguments are needed, corresponding to the csv files paths:

```
$ niamoto sync <csv_data_provider_name> <occurrences.csv> <plots.csv> <plots_
↪occurrences.csv>
```

Using `0` instead of a path means that no data is to be imported. For instance, importing only plot data can be achieved using:

```
$ niamoto sync <csv_data_provider_name> 0 <plots.csv> 0
```

Now let's import some data:

```
$ niamoto sync csv_gbif csv_niamoto_gbif_occurrences.csv csv_gbif_plots.csv csv_gbif_
↪plots_occurrences.csv
Syncing the Niamoto database with 'csv_gbif'...
[INFO] *** Data sync starting ('csv_gbif' - CSV)...
[INFO] ** Occurrence sync starting ('csv_gbif' - CSV)...
[INFO] ** Occurrence sync with 'csv_gbif' done (0.08 s)!
[INFO] ** Plot sync starting ('csv_gbif' - CSV)...
[INFO] ** Plot sync with 'csv_gbif' done (0.06 s)!
[INFO] *** Data sync with 'csv_gbif' done (total time: 0.08 s)!
The Niamoto database had been successfully synced with 'csv_gbif'!
Bellow is a summary of what had been done:
    Occurrences:
        432 inserted
        0 updated
        0 deleted
    Plots:
        34 inserted
        0 updated
        0 deleted
    Plots / Occurrences:
        432 inserted
        0 updated
        0 deleted
```

We can check the Niamoto database status with the `niamoto status` command:

```
$ niamoto status
    1 data providers are registered.
    123 taxa are stored.
    3 taxon synonym keys are registered.
    432 occurrences are stored.
    34 plots are stored.
    432 plots/occurrences are stored.
    0 rasters are stored.
    0 vectors are stored.
```

3.3 Importing rasters and vectors

3.4 Processing and publishing data

CHAPTER 4

Tutorial

4.1 Setting the taxonomy

Niamoto does not make any assumption on the taxonomic referential to be used, and therefore let you define it. Since the choice of a taxonomic referential should not be a blocking decision either a point of no return, Niamoto always store two taxon identifier values for an occurrence: the data provider's one, and it's correspondence in the Niamoto's referential. In order to be able to map a provider's taxon identifier with an internal taxon identifier, Niamoto needs a set of correspondences for each taxon. Those correspondences are called synonyms. When no synonym is known for a provider's taxon identifier, the Niamoto taxon identifier will be set null.

The definition of the Niamoto referential is done using a csv file. This csv file must have a header defining at least the following columns:

- **id**: The unique identifier of the taxon, in the provider's referential.
- **parent_id**: The parent's id of the taxon. If the taxon is a root, let the value blank.
- **rank**: The rank of the taxon, can be a value among: 'REGNUM', 'PHYLUM', 'CLASSIS', 'ORDO', 'FAMILIA', 'GENUS', 'SPECIES', 'INFRASPECIES'.
- **full_name**: The full name of the taxon.
- **rank_name**: The rank name of the taxon.

All the additional columns will be considered as synonyms.

Let's consider the following example:

id	parent_id	rank	full_name	rank_name	gbif	taxref
0		FAMILIA	A	A	10	1
1	0	GENUS	A a	a	20	2
2	1	SPECIES	A a 1	1	30	3
3	1	SPECIES	A a 2	2	40	4

We set this table as the Niamoto's taxonomic referential using the `set_taxonomy` command:

```
$ niamoto set_taxonomy taxonomy.csv
Setting the taxonomy...
The taxonomy had been successfully set!
    4 taxa inserted
    2 synonyms inserted: {'taxref', 'gbif'}
```

We can see that Niamoto found the two following synonym keys: **'taxref'** and **'gbif'**. Those keys are the one that we will use later to tell Niamoto how to map the data provider's taxon identifier. Note that there is also a special synonym key, **'niamoto'**, that is used when a data provider uses the same taxon identifiers as Niamoto.

4.2 Managing data providers

Now that we have set the taxonomic referential, we would like to import some data within Niamoto. But before being able to do so, we need to define data providers.

Using the command `niamoto providers`, we can see that there are not registered providers in the database:

```
$ niamoto providers
There are no registered data providers in the database.
```

The simplest data provider type implemented in Niamoto is the csv data provider, which enables us to import occurrence and plot data from plain csv files. All the available provider_types can be obtained using the `niamoto provider_types` command.

Adding a data provider can achieved using the `niamoto add_provider` command, which have the following usage:

```
$ niamoto add_provider --help
Usage: niamoto add_provider [OPTIONS] NAME PROVIDER_TYPE [SYNONYM_KEY]

  Register a data provider. The name of the data provider must be unique.
  The available provider types can be obtained using the 'niamoto
  provider_types' command. The available synonym keys can be obtained using
  the 'niamoto synonym_keys" command.

Options:
  --help  Show this message and exit.
```

Let's add three data providers: **csv_niamoto**, **csv_taxref** and **csv_gbif**:

```
$ niamoto add_provider csv_niamoto CSV niamoto
  Registering the data provider in database...
  The data provider had been successfully registered to Niamoto!
```

```
$ niamoto add_provider csv_taxref CSV taxref
  Registering the data provider in database...
  The data provider had been successfully registered to Niamoto!
```

```
$ niamoto add_provider csv_gbif CSV gbif
  Registering the data provider in database...
  The data provider had been successfully registered to Niamoto!
```

They are now available with the `niamoto providers` command:

```
$ niamoto providers
           name provider_type synonym_key
id
2   csv_niamoto           CSV     niamoto
3    csv_taxref           CSV      taxref
4      csv_gbif           CSV        gbif
```

In the next section, we will see how to import data with these data providers.

4.3 Importing occurrence, plot and plot/occurrence data

Importing data using the csv data provider is done with three csv files:

- The **occurrences** csv file, containing the occurrence data.
- The **plots** csv file, containing the plot data.
- The **plots/occurrences** csv file, mapping plots with occurrences.

All of them are optional, you can import only occurrences, only plots or only map existing plots with existing occurrences. The command for importing data from a provider is `niamoto sync PROVIDER_NAME [PROVIDER_ARGS]`. With the csv data provider, three arguments are needed, corresponding to the csv files paths:

```
$ niamoto sync <csv_data_provider_name> <occurrences.csv> <plots.csv> <plots_
↪occurrences.csv>
```

Using `0` instead of a path means that no data is to be imported. For instance, importing only plot data can be achieved using:

```
$ niamoto sync <csv_data_provider_name> 0 <plots.csv> 0
```

In this tutorial, we will import occurrence data for the three previously registered data providers. We will also import plot and plot/occurrence data, only for the first provider.

4.3.1 1. Importing occurrence data

The occurrences csv file must have a header and contain at least the following columns:

- **id**: The provider's unique identifier for the occurrence.
- **taxon_id**: The provider's taxon id for the occurrence.
- **x**: The longitude of the occurrence (WGS84).
- **y**: The latitude of the occurrence (WGS84).

All the remaining column will be stored as properties.

For the `csv_niamoto` provider, let's consider the following dataset:

id	taxon_id	x	y	dbh	height
0	3	165.321	-21.47	21	18
1	2	165.321	-21.47	20.5	14
2	2	165.321	-21.47	22.5	16
3	3	165.125	-21.54	18	12
4	3	165.125	-21.54	19	18
5	2	162.001	-18.11	11	15
6	2	162.001	-18.11	24	20
7	2	162.001	-18.11	25	22

For the `csv_taxref` provider, let's consider the following dataset:

id	taxon_id	x	y	status
0	4	92.321	42.40	alive
1	4	91.224	41.56	alive
2	4	91.015	41.11	dead
3	4	92.221	42.10	alive
4	4	92.221	42.10	dead
5	4	92.221	42.10	alive
6	4	92.221	42.10	alive

For the `csv_gbif` provider, let's consider the following dataset:

id	taxon_id	x	y
0	20	11.921	11.47
1	30	16.120	21.54
2	30	61.045	18.12
3	20	16.001	8.11

Now let's import the data:

```
$ niamoto sync csv_niamoto csv_niamoto_occurrences.csv 0 0
Syncing the Niamoto database with 'csv_niamoto'...
[INFO] *** Data sync starting ('csv_niamoto' - CSV)...
[INFO] ** Occurrence sync starting ('csv_niamoto' - CSV)...
[INFO] ** Occurrence sync with 'csv_niamoto' done (0.08 s)!
[INFO] *** Data sync with 'csv_niamoto' done (total time: 0.08 s)!
The Niamoto database had been successfully synced with 'csv_niamoto'!
Bellow is a summary of what had been done:
    Occurrences:
        8 inserted
        0 updated
        0 deleted
```

```
$ niamoto sync csv_taxref csv_niamoto_taxref_occurrences.csv 0 0
Syncing the Niamoto database with 'csv_taxref'...
[INFO] *** Data sync starting ('csv_taxref' - CSV)...
[INFO] ** Occurrence sync starting ('csv_taxref' - CSV)...
[INFO] ** Occurrence sync with 'csv_taxref' done (0.08 s)!
[INFO] *** Data sync with 'csv_taxref' done (total time: 0.08 s)!
The Niamoto database had been successfully synced with 'csv_taxref'!
Bellow is a summary of what had been done:
```

(continues on next page)

(continued from previous page)

```
    Occurrences:
        7 inserted
        0 updated
        0 deleted
```

```
$ niamoto sync csv_gbif csv_niamoto_gbif_occurrences.csv 0 0
Syncing the Niamoto database with 'csv_gbif'...
[INFO] *** Data sync starting ('csv_gbif' - CSV)...
[INFO] ** Occurrence sync starting ('csv_gbif' - CSV)...
[INFO] ** Occurrence sync with 'csv_gbif' done (0.08 s)!
[INFO] *** Data sync with 'csv_gbif' done (total time: 0.08 s)!
The Niamoto database had been successfully synced with 'csv_gbif'!
Bellow is a summary of what had been done:
    Occurrences:
        4 inserted
        0 updated
        0 deleted
```

We now have 19 occurrences coming from 3 data providers in our Niamoto database, as we can see using the following command:

```
$ niamoto status
    3 data providers are registered.
    4 taxa are stored.
    3 taxon synonym keys are registered.
    19 occurrences are stored.
    0 plots are stored.
    0 plots/occurrences are stored.
    0 rasters are stored.
    0 vectors are stored.
```

4.3.2 2. Importing plot data

The plot csv file must have a header and contain at least the following columns:

- **id**: The provider's identifier for the plot.
- **name**: The name of the plot.
- **x**: The longitude of the plot (WGS84).
- **y**: The latitude of the plot (WGS84).

All the remaining column will be stored as properties.

Let's consider the following dataset for the `csv_niamoto` provider:

id	name	x	y	width	height
0	plot_1	165.321	-21.47	100	100
1	plot_2	165.125	-21.54	100	100
2	plot_3	162.001	-18.11	100	100

We import the plot data using the following command:

```
$ niamoto sync csv_niamoto 0 csv_niamoto_plots.csv 0
    Syncing the Niamoto database with 'csv_niamoto'...
    [INFO] *** Data sync starting ('csv_niamoto' - CSV)...
    [INFO] ** Plot sync starting ('csv_niamoto' - CSV)...
    [INFO] ** Plot sync with 'csv_niamoto' done (0.06 s)!
    [INFO] *** Data sync with 'csv_niamoto' done (total time: 0.07 s)!
    The Niamoto database had been successfully synced with 'csv_niamoto'!
    Bellow is a summary of what had been done:
        Plots:
            3 inserted
            0 updated
            0 deleted
```

4.3.3 3. Importing plot/occurrence data

The plot/occurrence data is a many to many relationship between occurrences and plots. A plot can contains several occurrences and an occurrence can be contained by several plots. The plot/occurrence csv file must have a header and contain at least the following columns:

- **plot_id**: The provider's id for the plot.
- **occurrence_id**: The provider's id for the occurrence.
- **occurrence_identifier**: The occurrence identifier in the plot.

The additional columns will be ignored.

Let's consider the following data, for linking `csv_niamoto`'s occurrences with `csv_niamoto`'s plots:

plot_id	occurrence_id	occurrence_identifier
0	0	PLOT_1__OCC_1
0	1	PLOT_1__OCC_2
0	2	PLOT_1__OCC_3
1	3	PLOT_2__OCC_1
1	4	PLOT_2__OCC_2
2	5	PLOT_3__OCC_1
2	6	PLOT_3__OCC_2
2	7	PLOT_3__OCC_3

We import the plot/occurrence data using the following command:

```
$ niamoto sync csv_niamoto 0 0 csv_niamoto_plots_occurrences.csv
Syncing the Niamoto database with 'csv_niamoto'...
[INFO] *** Data sync starting ('csv_niamoto' - CSV)...
[INFO] ** Plot-occurrence sync starting ('csv_niamoto' - CSV)...
[INFO] ** Plot-occurrence sync with 'csv_niamoto' done (0.05 s)!
[INFO] *** Data sync with 'csv_niamoto' done (total time: 0.06 s)!
The Niamoto database had been successfully synced with 'csv_niamoto'!
Bellow is a summary of what had been done:
    Plots / Occurrences:
        8 inserted
        0 updated
        0 deleted
```

We can check the Niamoto database status with the `niamoto status` command:

```
$ niamoto status
    3 data providers are registered.
    4 taxa are stored.
    3 taxon synonym keys are registered.
    19 occurrences are stored.
    3 plots are stored.
    8 plots/occurrences are stored.
    0 rasters are stored.
    0 vectors are stored.
```

4.4 Importing rasters

Niamoto provides functionalities to import and manage raster within the Niamoto database, these functionalities rely on the PostGIS raster functionalities. The main advantage of storing rasters inside a PostGIS database is to benefit from the power of the SQL language, and the PostGIS spatial functions. It is also a convenient way for having all the data stored at the same place and for using the same system for querying.

Importing a raster in Niamoto is straightforward using the `niamoto add_raster` command:

```
$ niamoto add_raster --help
Usage: niamoto add_raster [OPTIONS] NAME RASTER_FILE_PATH

  Add a raster in Niamoto's raster database.

Options:
  -t, --tile_dimension TEXT  Tile dimension <width>x<height>
  -R, --register             Register the raster as a filesystem (out-db)
                             raster. (-R option of raster2pgsql).
  --help                     Show this message and exit.
```

Now let's import a rainfall raster in our Niamoto database:

```
$ niamoto add_raster rainfall rainfall.tif
Registering the raster in database...
The raster had been successfully registered to the Niamoto raster database!
```

We can see the registered rasters with the `niamoto rasters` command:

```
$ niamoto rasters
        name date_create date_update
id
1   rainfall  2017/06/08        None
```

4.5 Importing vectors

Niamoto rely on the *ogr2ogr* utility to import vector layers in the Niamoto vector database. To import a vector layer, we use the *niamoto add_vector* command:

```
$ niamoto add_vector boundaries boundaries.shp
Registering the vector in database...
The vector had been successfully registered to the Niamoto vector database!
```

We can see the registered rasters with the `niamoto vectors` command:

```
$ niamoto vectors
          name date_create date_update
id
2   boundaries  2017/06/23        None
```

4.6 Extracting raster values to occurrences and plot properties

Niamoto provides utilities for extracting raster values directly into occurrences or plots properties.

```
$ niamoto raster_to_occurrences --help
Usage: niamoto raster_to_occurrences [OPTIONS] RASTER_NAME

  Extract raster values to occurrences properties.

Options:
  --help  Show this message and exit
```

```
$ niamoto raster_to_plots --help
Usage: niamoto raster_to_plots [OPTIONS] RASTER_NAME

  Extract raster values to plots properties.

Options:
  --help  Show this message and exit.
```

```
$ niamoto all_rasters_to_occurrences --help
Usage: niamoto all_rasters_to_occurrences [OPTIONS]

  Extract raster values to occurrences properties for all registered
  rasters.

Options:
  --help  Show this message and exit.
```

```
$ niamoto all_rasters_to_plots --help
Usage: niamoto all_rasters_to_plots [OPTIONS]

  Extract raster values to plots properties for all registered rasters.

Options:
  --help  Show this message and exit.
```

For instance, let's extract the values of the previously registered raster, `rainfall` to the occurrences properties:

```
$ niamoto raster_to_occurrences rainfall
Extracting 'rainfall' raster values to occurrences...
The raster values had been successfully extracted!
```

4.7 Processing and publishing data

The list of available data publishers can be displayed using the `niamoto publishers` command:

```
$ niamoto publishers
    occurrences       :   Publish the occurrence dataframe with properties as columns.
    plots             :   Publish the plot dataframe with properties as columns.
    taxa              :   Publish the taxa dataframe.
    plots_occurrences :   Publish the plots/occurrences dataframe.
    raster            :   Publish a raster from the niamoto raster database.
```

For a given publisher, the available publish formats can be displayed using the `niamoto publish_formats` command:

```
$ niamoto publish_formats occurrences
    csv :    Publish the data using the csv format.
    sql :    Publish the data as a table to a SQL database
```

For each publisher, it is possible to get the list options using the `niamoto publish <publisher> --help` command:

```
$ niamoto publish occurrences --help
Usage: niamoto publish occurrences [OPTIONS] COMMAND [ARGS]...

  Publish the occurrence dataframe with properties as columns.

Options:
  --drop_null_properties
  --properties TEXT       List of properties to retain. Can be a python list
                          or a comma (',') separated string.
  --help                  Show this message and exit.

Commands:
  csv  Publish the data in a csv file.
  sql  Publish a DataFrame as a table to a SQL...
```

The same is possible for each publisher's publish format:

```
$ niamoto publish occurrences csv --help
Usage: niamoto publish occurrences csv [OPTIONS]

  Publish the data in a csv file.

Options:
  --index_label TEXT
  -d, --destination TEXT
  --help                  Show this message and exit.
```

Let's publish the occurrence dataframe in a csv file:

```
$ niamoto publish occurrences csv -d occurrences.csv
```

CHAPTER 5

CLI reference

5.1 General commands

5.1.1 init_niamoto_home

```
Usage: niamoto init_niamoto_home [OPTIONS]

  Initialize the Niamoto home directory.

Options:
  --niamoto_home_path TEXT
  --help                    Show this message and exit.
```

5.1.2 init_db

```
Usage: niamoto init_db [OPTIONS]

  Initialize the Niamoto database.

Options:
  --help  Show this message and exit.
```

5.1.3 status

```
Usage: niamoto status [OPTIONS]

  Show the status of the Niamoto database.

Options:
  --help  Show this message and exit.
```

5.2 Taxonomy commands

5.2.1 set_taxonomy

```
Usage: niamoto set_taxonomy [OPTIONS] CSV_FILE_PATH

  Set the taxonomy.

Options:
  --no_mapping BOOLEAN
  --help                Show this message and exit.
```

5.2.2 map_all_synonyms

```
Usage: niamoto map_all_synonyms [OPTIONS]

  Update the synonym mapping for every data provider registered in the
  database.

Options:
  --help  Show this message and exit.
```

5.2.3 synonym_keys

```
Usage: niamoto synonym_keys [OPTIONS]

  List the registered synonym keys.

Options:
  --help  Show this message and exit.
```

5.3 Data providers commands

5.3.1 provider_types

```
Usage: niamoto provider_types [OPTIONS]

  List registered data provider types.

Options:
  --help  Show this message and exit.
```

5.3.2 providers

```
Usage: niamoto providers [OPTIONS]

  List registered data providers.
```

(continues on next page)

(continued from previous page)

```
Options:
```

5.3.3 add_provider

```
Usage: niamoto add_provider [OPTIONS] NAME PROVIDER_TYPE [SYNONYM_KEY]

  Register a data provider. The name of the data provider must be unique.
  The available provider types can be obtained using the 'niamoto
  provider_types' command. The available synonym keys can be obtained using
  the 'niamoto synonym_keys" command.

Options:
  --help  Show this message and exit.
```

5.3.4 update_provider

```
Usage: niamoto update_provider [OPTIONS] CURRENT_NAME

  Update a data provider.

Options:
  --new_name TEXT
  --synonym_key TEXT
  --help              Show this message and exit.
```

5.3.5 delete_provider

```
Usage: niamoto delete_provider [OPTIONS] NAME

  Delete a data provider.

Options:
  -y TEXT
  --help   Show this message and exit.
```

5.3.6 sync

```
Usage: niamoto sync [OPTIONS] PROVIDER_NAME [PROVIDER_ARGS]...

  Sync the Niamoto database with a data provider.

Options:
  --help  Show this message and exit.
```

5.4 Raster commands

5.4.1 rasters

```
Usage: niamoto rasters [OPTIONS]

  List registered rasters.

Options:
  --help  Show this message and exit.
```

5.4.2 add_raster

```
Usage: niamoto add_raster [OPTIONS] NAME RASTER_FILE_PATH

  Add a raster in Niamoto's raster database.

Options:
  -t, --tile_dimension TEXT  Tile dimension <width>x<height>
  --help                     Show this message and exit.
```

5.4.3 update_raster

```
Usage: niamoto update_raster [OPTIONS] NAME RASTER_FILE_PATH

  Update an existing raster in Niamoto's raster database.

Options:
  -t, --tile_dimension TEXT  Tile dimension <width>x<height>
  --new_name TEXT            The new name of the raster
  --help                     Show this message and exit.
```

5.4.4 delete_raster

```
Usage: niamoto delete_raster [OPTIONS] NAME

  Delete an existing raster from Niamoto's raster database.

Options:
  --help   Show this message and exit.
```

5.4.5 raster_to_occurrences

```
Usage: niamoto raster_to_occurrences [OPTIONS] RASTER_NAME

  Extract raster values to occurrences properties.

Options:
  --help  Show this message and exit.
```

5.4.6 raster_to_plots

```
Usage: niamoto raster_to_plots [OPTIONS] RASTER_NAME

  Extract raster values to plots properties.

Options:
  --help  Show this message and exit.
```

5.4.7 all_rasters_to_occurrences

```
Usage: niamoto all_rasters_to_occurrences [OPTIONS]

  Extract raster values to occurrences properties for all registered
  rasters.

Options:
  --help  Show this message and exit.
```

5.4.8 all_rasters_to_plots

```
Usage: niamoto all_rasters_to_plots [OPTIONS]

  Extract raster values to plots properties for all registered rasters.

Options:
  --help  Show this message and exit.
```

5.5 Vector commands

5.5.1 vectors

```
Usage: niamoto vectors [OPTIONS]

  List the registered vectors.

Options:
  --help  Show this message and exit.
```

5.5.2 add_vector

```
Usage: niamoto add_vector [OPTIONS] NAME VECTOR_FILE_PATH

  Add a raster in Niamoto's vector database.

Options:
  --help  Show this message and exit.
```

5.5.3 update_vector

```
Usage: niamoto add_vector [OPTIONS] NAME VECTOR_FILE_PATH

  Add a raster in Niamoto's vector database.

Options:
  --help  Show this message and exit.
```

5.5.4 delete_vector

```
Usage: niamoto delete_vector [OPTIONS] NAME

  Delete an existing vector from Niamoto's vector database.

Options:
  --help   Show this message and exit.
```

5.6 Data publisher commands

5.6.1 publish_formats

```
Usage: niamoto publish_formats [OPTIONS] PUBLISHER_KEY

  Display the list of available publish formats for a given publisher.

Options:
  --help  Show this message and exit.
```

5.6.2 publishers

```
Usage: niamoto publishers [OPTIONS]

  Display the list of available data publishers.

Options:
  --help  Show this message and exit.
```

5.6.3 publish

Note: Please refer to *Data publishers reference* for details specific to each available data publisher.

```
Usage: niamoto publish [OPTIONS] PUBLISHER_KEY PUBLISH_FORMAT [ARGS]...

  Process and publish data.

```

(continues on next page)

(continued from previous page)

```
Options:
  -d, --destination TEXT
  --help                  Show this message and exit.
```

5.7 Data marts commands

5.7.1 dimension_types

```
Usage: niamoto dimension_types [OPTIONS]

  List the available dimension types.

Options:
  --help  Show this message and exit.
```

5.7.2 dimensions

```
Usage: niamoto dimensions [OPTIONS]

  List the registered dimensions.

Options:
  --help  Show this message and exit.
```

5.7.3 fact_tables

```
Usage: niamoto fact_tables [OPTIONS]

  List the registered fact tables.

Options:
  --help  Show this message and exit.
```

5.7.4 create_taxon_dimension

```
Usage: niamoto create_taxon_dimension [OPTIONS]

  Create the taxon dimension.

Options:
  --populate  Populate the dimension
  --help      Show this message and exit.
```

5.7.5 create_vector_dimension

```
Usage: niamoto create_vector_dimension [OPTIONS] VECTOR_NAME

  Create a vector dimension from a registered vector.

Options:
  --label_col TEXT  The label column name of the dimension
  --populate        Populate the dimension
  --help            Show this message and exit.
```

5.7.6 create_fact_table

```
Usage: niamoto create_fact_table [OPTIONS] NAME

  Create and register a fact table from existing dimensions. Use -d
  <dimension_name> for each dimension, and -m <measure_name> for each
  measure.

Options:
  -d, --dimension TEXT  The fact table's dimension names  [required]
  -m, --measure TEXT    The fact table's measures names  [required]
  --help                Show this message and exit.
```

5.7.7 delete_dimension

```
Usage: niamoto delete_dimension [OPTIONS] DIMENSION_NAME

  Delete a registered dimension.

Options:
  --help  Show this message and exit.
```

5.7.8 delete_fact_table

```
Usage: niamoto delete_fact_table [OPTIONS] FACT_TABLE_NAME

  Delete a registered fact table.

Options:
  --help  Show this message and exit.
```

5.7.9 populate_fact_table

```
Usage: niamoto populate_fact_table [OPTIONS] FACT_TABLE_NAME PUBLISHER_KEY

  Populate a registered fact table using an available publisher.

Options:
  --help  Show this message and exit.
```

CHAPTER 6

Data publishers reference

6.1 Occurrence publisher

Publish the occurrence dataframe with properties as columns.

key `occurrences`

formats `csv`

options

`--properties <properties>` List of properties to extract as columns. If not specified, retrieve all the existing properties. Exemple: `--properties height dbh`.

`--drop_null_properties` Flag indicating that record with null values for properties must be dropped.

6.2 Plot publisher

Publish the plot dataframe with properties as columns.

key `plots`

formats `csv`

options

`--properties <properties>` List of properties to extract as columns. If not specified, retrieve all the existing properties. Exemple: `--properties width height`.

6.3 Plot/Occurrence publisher

Publish the plot/occurrence dataframe.

key `plots_occurrences`

formats `csv`

options

6.4 Taxon publisher

Publish the taxa dataframe.

key `taxa`

formats `csv`

options

`--include_mptt` Flag indicating that the MPTT (Modified Pre-ordered Traversal Tree) columns must be included

CHAPTER 7

Configuration

(Available soon)

CHAPTER 8

Contributing

8.1 Setting up a developpment environment

Prerequesite: You need to have `PostgreSQL` and `PostGIS` installed and running in your system, or have access to a distant instance of it. Check the test settings in the `tests/test_data/test_niamoto_home/settings.py` file.

It is recommended to use `virtualenv` to setup a development environment with python 3.4, 3.5 or 3.6. Please refer to https://virtualenv.pypa.io/en/stable/

First, clone the repository in your system using `git`:

```
git clone https://github.com/dimitri-justeau/niamoto-core.git
```

Move in the cloned repository and install the dependencies using `pip`:

```
pip install -r requirements.txt
```

Finally download the test data:

```
sh tests/download_test_data.sh
```

You can check the tests using:

```
python run_tests.py
```