Tablo Documentation

Release 1.0.2

Conservation Biology Institute

Jul 07, 2017

Contents

Tablo is a lightweight Django application that creates a interface layer for interacting with spatial data stored within a PostGIS database.

CHAPTER 1

Tablo

What is Tablo?

Tablo is a lightweight Django application that creates a interface layer for interacting with spatial data stored within a PostGIS database.

Goals

The main goal of Tablo is to allow us to store data in a PostGIS database and be able to access it in the same way we access ArcGIS feature services.

We wanted to be able to query the data in the PostGIS database using a REST endpoint similar to this:

```
localhost:8383/tablo/arcgis/rest/services/905/FeatureServer/0/query?f=json
&returnGeometry=true&spatialRel=esriSpatialRelIntersects
&geometry={"xmin":-13697515.468700845,"ymin":5662246.477956772,
"xmax":-13619243.951736793,"ymax":5740517.994920822,
"spatialReference":{"wkid":3857}}
&geometryType=esriGeometryEnvelope&inSR=3857&outFields=*&outSR=3857
```

This allows us to take a CSV file like this:

Team Name	Name of the Stream	Longitude	Latitude	Temperature Reading
Fun Team	Columbia 'D' River	-122.2345	45.5483	60
Fun Team 2	Columbia River	-122.2869	45.5429	60
Fun Team	Columbia River	-122.4224	45.5655	65
Portlandias	Johnson Creek	-122.5849	45.4606	59
Portlandias	Johnson Creek	-122.5986	45.4551	61

And turn it into a map like this:

Documentation

Full documentation available here.

CHAPTER 2

Installing

To keep things separate, you will most likely want to work within a python virtual environment. This just makes things easier to manage in general. A good tool for managing virtual environments can be found at https://virtualenvwrapper.readthedocs.io/en/latest/.

Tablo has been written to support Python 3.5, so you will need to make sure your virtual environment is using Python 3.5.

Dependencies

Tablo is a Django application that relies on a PostGIS database. So, the first two things you'll need to work with Tablo are:

1. A PostGIS database (version 9.4 or above)
2. A Django Project

Setting up a PostGIS database

The scope of this document is not large enough to encapsulate all that is required in installing a PostGIS database. See http://postgis.net/install/ for information on how to install it on your own system (or on a remote server).

Setting up a Django Project

See https://docs.djangoproject.com/en/1.8/intro/tutorial01/#creating-a-project for instructions on how to set up a Django project. Tablo was tested using Django version 1.8.6.

For example, you could create a project called tablo_project by issuing the command: $ django-admin startproject tablo_project

Anywhere in this documentation where it refers to the Django project, it will be referenced as tablo_project.

Other Dependencies

Tablo relies on a few libraries that can often be more difficult to install. These include:

- pyproj (https://github.com/jswhit/pyproj)
- lxml (via messytables) (http://lxml.de/)
- GDAL (https://pypi.python.org/pypi/GDAL/)

See these websites to install these libraries first, as they may depend on other executables existing in your environment.

Installing Tablo

There are two main ways to install tablo; directly, or indirectly from a cloned repository.

To install directly, use the standard pip installation:

$ pip install git+https://github.com/consbio/tablo.git‘

To install indirectly from a cloned repository:

1. Clone this repository
2. **Install the app using pip and pointing to your local repository, but use the *–editable* flag, as in:** $ pip install –editable c:tablo‘

Modify Django Settings

In your tablo_project/tablo_project settings file, make the following modifications:

```
INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.auth',
    ...
    'tastypie',
    'tablo'
)

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': '{ database name }',
        'HOST': '{ database host }',
        'USER': '{ database user }',
        'PASSWORD': '{ database password}'
    }
}
```

Modify urls.py

In tablo_project/tablo_project/urls.py, make sure that the tablo.urls are included:

```
urlpatterns = [
    url(r'^', include('tablo.urls')),
    url(r'^admin/', include(admin.site.urls)),
]
```

Add API Key Authentication

To allow other applications to use Tablo using an API key rather than a full user login, add middleware.py file that looks like this:

```
from tastypie.authentication import ApiKeyAuthentication


class TastypieApiKeyMiddleware(object):
    """Middleware to authenticate users using API keys for regular Django views"""

    def process_request(self, request):
        ApiKeyAuthentication().is_authenticated(request)
```

Then add an additional INSTALLEDAPP referencing *tablo_project.middleware.TastypieApiKeyMiddleware*.

Create a tablo user

$ python manage.py createsuperuser –username=tablo –email=your.email@somewhere.com

Supply a password for the tablo user.

Run the server

Make sure that your path to osgeo is in your PATH variable.

Run the server by issuing the *manage.py runserver* command.

Setup the API Key

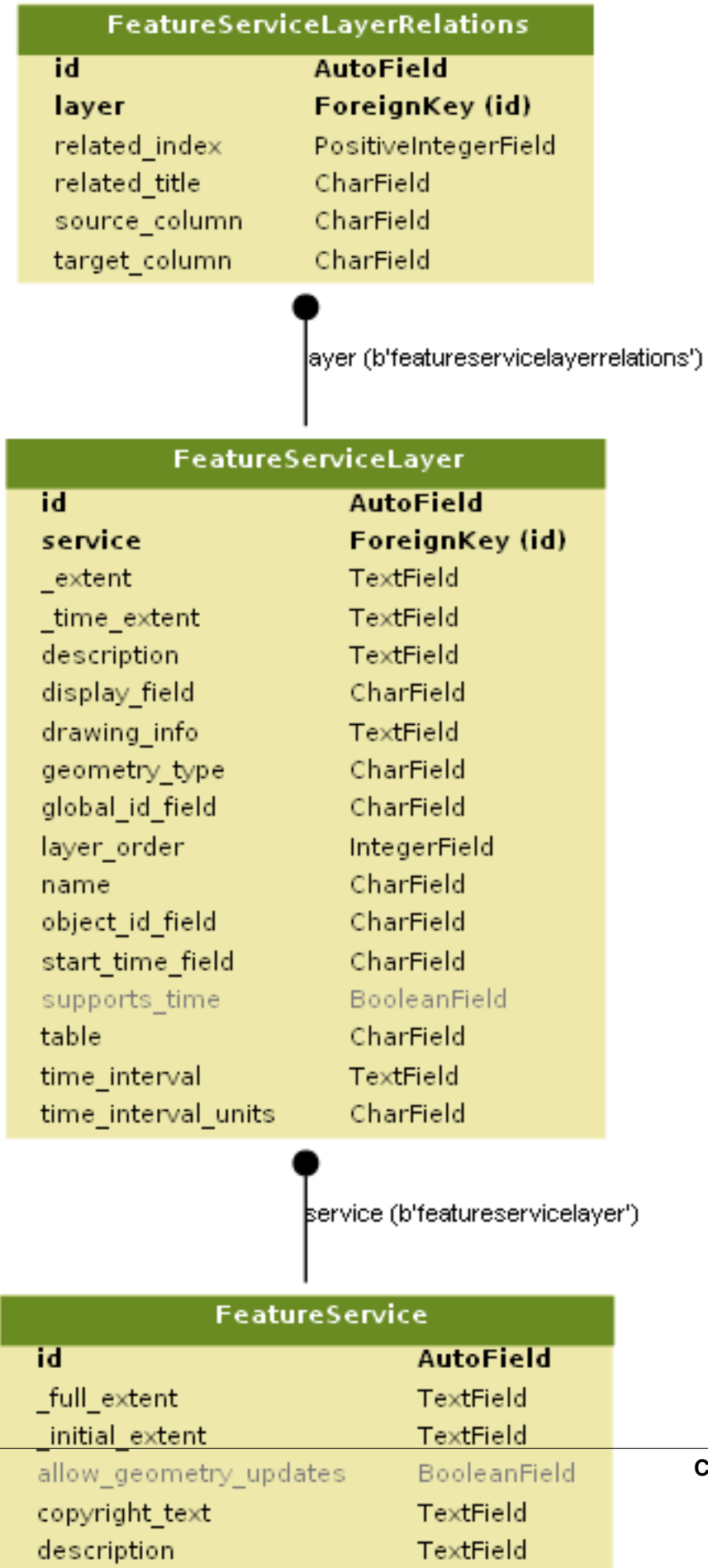
Go to http://localhost/admin and login as tablo, using the password you supplied above.

Add an API key for the tablo user. This is the API key you will want to set for any applications that communicate with the tablo server.

CHAPTER 3

Tablo Data Model

The Tablo data model is structured around three tables, shown in the diagram below:


FeatureServiceLayerRelations
id AutoField
layer ForeignKey (id)
related_index PositiveIntegerField
related_title CharField
source_column CharField
target_column CharField
ayer (b'featureservicelayerrelations')
FeatureServiceLayer
id AutoField
service ForeignKey (id)
_extent TextField
_time_extent TextField
description TextField
display_field CharField
drawing_info TextField
geometry_type CharField
global_id_field CharField
layer_order IntegerField
name CharField
object_id_field CharField
start_time_field CharField
supports_time BooleanField
table CharField
time_interval TextField
time_interval_units CharField
service (b'featureservicelayer')
FeatureService
id AutoField
_full_extent TextField
_initial_extent TextField
allow_geometry_updates BooleanField
copyright_text TextField
description TextField

The main table, **FeatureService**, contains the information about the feature service itself, its capabilities, extents, etc. The ID for the FeatureService will be used in any URLs used to reference the data.

The **FeatureServiceLayer** table associates an individual layer with a **FeatureService**. Currently, this is a one to one relationship, but could be one to many in the future. The *table* property within the *FeatureServiceLayer* identifies the table where this FeatureServiceLayer will retrieve it's data from.

The **FeatureServiceLayerRelations** table allows the association of related tables to the main layer table. Associated data will be pulled from a table named with the same name as the **FeatureServiceLayer's** table, but with a suffix of _{related_index}. So, if the original layer pulled data from a table called `db_12345`, the first related table would be called `db_12345_0`.

Data Tables

Tables containing data can have any structure, but must contain the following a `db_id` field that is the primary key for the table, and a `dbasin_geom` field that contains the geometry for the given row.

Related tables do not need these fields, but must have foreign keys that match the `source_column` and `target_column` in the FeatureServiceLayerRelations table.

CHAPTER 4

Data Upload

Tablo is built around the idea of being able to upload a CSV file and then consuming that data through one or more generic interfaces that abstract out the data storage layer. Before you can consume the data, you must upload it into Tablo. This document details the steps involved in the upload process.

There are multiple steps to the process, allowing for minor changes in data along the way. The minimum set of steps is as follows:

1. *Upload CSV File*
2. *Describe the CSV File*
3. *Deploy the CSV File*
4. *Create the Feature Service*
5. *Finalize the Feature Service*

Upload

`TemporaryFileUploadUrlView.`**`dispatch`**(*request*, **args*, ***kwargs*)

Uploads a CSV file, based on URL and stores it under a UUID, allowing it to be referenced later. Send a POST message to `{tablo-server}/tablo/admin/upload-by-url`, with a parameter of **url** specifying where your file resides.

Keyword Arguments

url The URL of the file you wish to upload-by-url

Returns

A JSON object in the following format:

```
{
    "uuid": "uniqueIdentifierForTheFile"
}
```

Describe

`TemporaryFileResource.`**`describe`**(*request*, ***kwargs*)

Describe, located at the `{tablo-server}/api/v1/temporary-files/{uuid}/describe` endpoint, will describe the uploaded CSV file. This allows you to know the column names and data types that were found within the file.

Returns

A JSON object in the following format:

```
{
    "fieldNames": ["field one", "field two", "latitude", "longitude"],
    "dataTypes": ["String", "Integer", "Double", "Double"],
    "optionalFields": ["field one"],
    "xColumn": "longitude",
    "yColumn": "latitude",
    "filename": "uploaded.csv"
}
```

fieldNames A list of field (column) names within the CSV

dataTypes A list of data types for each of the columns. The index of this list will match the index of the fieldNames list.

optionalFields A list of fields that had empty values, and are taken to be optional.

xColumn The best guess at which column contains X spatial coordinates.

yColumn The best guess at which column contains Y spatial coordinates.

filename The name of the file being described

Deploy

`TemporaryFileResource.`**`deploy`**(*request*, ***kwargs*)

The deploy endpoint, at `{tablo_server}/api/v1/temporary-files/{uuid}/{dataset_id}/deploy/` deploys the file specified by {uuid} into a database table named after the {dataset_id}. The {dataset_id} must be unique for the instance of Tablo.

With the deploy endpoint, this is the start of what Tablo considers an import. The data will be temporarily stored in an import table until the finalize endpoint for the dataset_id is called.

POST messages to the deploy endpoint should include the following data:

csv_info Information about the CSV file. This is generally that information obtained through the describe endpoint, but can be modified to send additional information or modify it.

fields A list of field JSON objects in the following format:

```
{
    "name": "field_name",
    "type": "text",
    "value": "optional value",
    "required": true
}
```

The value can be specified if the field is a constant value throughout the table. This can be use for adding audit information.

Returns An empty HTTP 200 response if the deploy was successful. An error response if otherwise.

Create a Feature Service

class `tablo.api.`**`FeatureServiceResource`**(*api_name=None*)

The FeatureService resource, located at `{tablo_server}/api/v1/featureservice`, allows you to create, edit and delete feature services within Tablo.

Once your data has been deployed through the deploy endpoint, you can create a feature service by sending a POST message to the above endpoint with data structured as:

```
{
    "description": "",
    "copyright_text": "",
    "spatial_reference": "{'wkid': 3857}",
    "units": "esriMeters",
    "allow_geometry_updates": false,
    "layers": [
        {
            "layer_order": 0,
            "table": "db_dataset_id_import",
            "name": "layername",
            "description": null,
            "geometry_type": "esriGeometryPoint",
            "supports_time": false,
            "start_time_field": null,
            "time_interval": 0,
            "time_interval_units": null,
            "drawing_info": {
                "renderer": {
                    "description": "",
                    "label": "",
                    "symbol": {
                        "angle": 0,
                        "color": [255, 0, 0, 255],
                        "size": 5,
                        "style": "esriSMSCircle",
                        "type": "esriSMS",
                        "xoffset": 0,
                        "yoffset": 0
                    },
                    "type": "simple"
                }
            }
        }
    ]
}
```

drawing_info contains ESRI styling for the FeatureService geometry type. The creation will return a URL that will contain the Service ID of your newly created feature service.

Finalize the Service

`FeatureServiceResource.`**`finalize`**(*request*, ***kwargs*)

The finalize endpoint, located at `{tablo_host}/api/v1/featureservice/{service_id}/finalize` allows you to finalize the feature service and mark it as available for use. This endpoint moves the data from the temporary database table and into its permanent one. This endpoint whould be accessed with a **service_id** parameter specifying the Service ID of the service you want to finalize.

CHAPTER 5

Interfaces

Tablo provides a partial implementation of the *ArcGIS Feature Service API* (http://resources.arcgis.com/en/help/rest/apiref/featureserver.html).

ArcGIS Interface

This interface layer provides a lightweight interface to the underlying PostGIS data stored in Tablo similar to that provided by an ArcGIS feature service. The services stored in Tablo can be accessed using endpoints such as `{tablo_server}/rest/services/{service_id}/FeatureServer`, where tablo_server is the host name of the server and service_id is the ID of the service within Tablo.

Accessing the FeatureServer endpoint will provide data about the Feature Service itself, such as version and layer information. Since these represent feature services, they are limited to a single layer, so the `{tablo_server}/rest/services/{service_id}/FeatureServer/0` endpoint will give more information about the specific layer properties, such as attributes, extent and related tables (if they exist).

This is the main interface layer that most external applications will use to access and query the spatial data stored within Tablo. The access points here are open and not restricted by any authentication layers, so authentication would need to happen outside of Tablo.

The Feature Server Layer will have more information provided by the layer specific endpoints.

Query

class `tablo.interfaces.arcgis.views.`**`QueryView`**(**args*, ***kwargs*)

Query is the main way data is retrieved from a feature service. This implements an api similar to ArcGIS http://resources.arcgis.com/en/help/rest/apiref/fsquery.html, but limited to the following parameters listed below. It can be accessed using the endpoint at `{tablo_server}/rest/services/{service_id}/FeatureServer/0/query`.

Keyword arguments can be passed as query arguments such as `{tablo_server}/rest/services/{service_id}/FeatureServer/0/query?f=json&offset=2&returnIdsOnly=true`

Keyword Arguments

- **geometryType** (*string*) The type of geometry sent in the geometry argument. Valid values are *esriGeometryEnvelope* and *esriGeometryPolygon*
- **limit** (*int*) The maximum number of features returned by the query
- **objectIds** (*comma-separated list*) A comma-separated list of ObjectIDs for the features in the table that you want to query
- **offset** (*int*) The starting record number for the query, often used in concert with the limit to paginate groups of responses
- **orderByFields** (*string List*) The names of the attributes to order the response by. Optional ASC and DESC flags can be used here to specify ascending or descending order. The default order is ASC.
- **outFields** (*string List*) The names of the attributes to include in the response
- **outSR** (*spatial reference*) The spatial reference for the returned geometry.
- **returnCountOnly** (*boolean*) Returns only the count of the matching features
- **returnGeometry** (*boolean*) Whether or not to return the geometry of the feature in the query response.
- **returnIdsOnly** (*boolean*) Returns only the ObjectIDs of the matching features
- **time** (*[float, float]*) The start_time and end_time for the query (in seconds since the epoch)
- **where** (*string*) A where clause for the query

Returns A JSON response, with slightly different syntax depending on the information requested with the keyword arguments. Example responses can be seen at http://resources.arcgis.com/en/help/rest/apiref/fsquery.html

TimeQuery

class `tablo.interfaces.arcgis.views.`**`TimeQueryView`**(**args*, ***kwargs*)

TimeQuery is a way to get back consolidated time data about a time-enabled feature service in Tablo. It is an extension to the base ArcGIS interface, but is built on top of it. It can be accessed at the `{tablo_server}/rest/services/{service_id}/FeatureServer/0/query` endpoint. It does not take any keyword arguments but returns a list of features that specify the location and count of feature occurrences over the full time of the feature service's time extent.

Response A JSON object similar to the following:

```
{
    "fields":[
        {
            "type":"esriFieldTypeInteger",
            "name":"count",
            "alias":"count"
        }
    ],
    "geometryType":"esriGeometryPoint",
    "count":3,
    "features":[
        {
            "attributes":{
                "count":90
```

```
            },
            "geometry":{
                "y":4020267.35731412,
                "x":-12974132.1182389
            }
        },
        {
            "attributes":{
                "count":173
            },
            "geometry":{
                "y":4021352.9816459,
                "x":-12978661.118326
            }
        },
        {
            "attributes":{
                "count":1572
            },
            "geometry":{
                "y":4020153.56924836,
                "x":-12978123.2336784
            }
        }
    ]
}
```


GenerateRenderer

class `tablo.interfaces.arcgis.views.`**`GenerateRendererView`**(**args*, ***kwargs*)

GenerateRenderer operation groups data using the supplied classificationDef to create a renderer object. This renderer object can then be used as a style or to create a legend. It can be accessed using the endpoint at `{tablo_server}/rest/services/{service_id}/FeatureServer/0/generateRenderer`.

Keyword Arguments

- **classificationDef** A *classificationDef* object with a JSON syntax like:


```
{
    "type": "classBreaksDef",
    "classificationField": "POP2010",
    "classificationMethod": "esriClassifyNaturalBreaks",
    "breakCount": 5,
    "normalizationType": "esriNormalizeByField",
    "normalizationField": "Area"
}
```


Returns

An ArcGIS renderer JSON object, like this:


```
{
    "type": "simple",
    "symbol":
    {
        "type": "esriSMS",
```

```
        "style": "esriSMSCircle",
        "color": [255,0,0,255],
        "size": 5,
        "angle": 0,
        "xoffset": 0,
        "yoffset": 0,
        "outline":
        {
         "color": [0,0,0,255],
         "width": 1
        }
    },
    "label": "",
    "description": ""
}
```

Python Module Index

t

Index