
pydatajson Documentation

Release 0.1.3

Datos Argentina

December 19, 2016

1	pydatajson	1
1.1	Indice	1
1.2	Instalación	1
1.3	Uso	2
1.4	Tests	4
1.5	Créditos	4
2	History	5
2.1	0.1.4 (2016-12-16)	5
2.2	0.1.2 (2016-12-14)	5
2.3	0.1.0 (2016-12-01)	5
2.4	0.0.13 (2016-11-25)	5
2.5	0.0.12 (2016-11-25)	5
3	Indices y tablas	7

pydatajson

Paquete en python con herramientas para manipular y validar metadatos de catálogos de datos en formato data.json.

- Licencia: MIT license
- Documentación: <https://pydatajson.readthedocs.io>

1.1 Indice

- *Instalación*
- *Uso*
 - *Setup*
 - *Posibles validaciones de catálogos*
 - *Ubicación del catálogo a validar*
 - *Ejemplos*
 - * *Archivo data.json local*
 - * *Archivo data.json remoto*
 - * *Diccionario (data.json deserializado)*
- *Tests*
- *Créditos*

1.2 Instalación

- **Producción:** Desde cualquier parte

```
$ pip install pydatajson
```

- **Desarrollo:** Clonar este repositorio, y desde su raíz, ejecutar:

```
$ pip install -e .
```

1.3 Uso

La librería implementa el objeto `DataJson` con varios métodos para verificar la integridad de archivos de metadatos `data.json` (locales o remotos) y manipular su contenido.

1.3.1 Setup

`DataJson` utiliza un esquema default que cumple con el perfil de metadatos recomendado en la [Guía para el uso y la publicación de metadatos \(v0.1\)](#) del [Paquete de Apertura de Datos](#).

```
from pydatajson import DataJson

dj = DataJson()
```

Si se desea utilizar un esquema alternativo, se debe especificar un **directorio absoluto** donde se almacenan los esquemas (`schema_dir`) y un nombre de esquema de validación (`schema_filename`), relativo al directorio de los esquemas. Por ejemplo, si nuestro esquema alternativo se encuentra en `/home/datosgobar/metadatos-portal/esquema_de_validacion.json`, especificaremos:

```
from pydatajson import DataJson

dj = DataJson(schema_filename="esquema_de_validacion.json",
              schema_dir="/home/datosgobar/metadatos-portal")
```

1.3.2 Posibles validaciones de catálogos

- Si se desea un **resultado sencillo** (V o F) sobre la validez de la estructura del catálogo, se utilizará `is_valid_catalog(datajson_path_or_url)`.
- Si se desea un **mensaje de error detallado**, se utilizará `validate_catalog(datajson_path_or_url)`.

1.3.3 Ubicación del catálogo a validar

Ambos métodos mencionados de `DataJson()` son capaces de validar archivos `data.json` locales o remotos:

- Para validar un **archivo local**, `datajson_path_or_url` deberá ser el **path absoluto** a él.
- Para validar un **archivo remoto**, `datajson_path_or_url` deberá ser una **URL que comience con ‘http’ o ‘https’**.

Alternativamente, también se pueden validar **diccionarios**, es decir, el resultado de deserializar un archivo `data.json` en una variable.

Por conveniencia, la carpeta `tests/samples/` contiene varios ejemplos de `data.json`s bien y mal formados con distintos tipos de errores.

1.3.4 Ejemplos

Archivo `data.json` local

```

from pydatajson import DataJson

dj = DataJson()
datajson_path = "tests/samples/full_data.json"
validation_result = dj.is_valid_catalog(datajson_path)
validation_report = dj.validate_catalog(datajson_path)

print validation_result
True

print validation_report
{
    "status": "OK",
    "error": {
        "catalog": {
            "status": "OK",
            "errors": [],
            "title": "Datos Argentina"
        },
        "dataset": [
            {
                "status": "OK",
                "errors": [],
                "title": "Sistema de contrataciones electrónicas"
            }
        ]
    }
}

```

Archivo data.json remoto

```

datajson_url = "http://181.209.63.71/data.json"
validation_result = dj.is_valid_catalog(datajson_url)
validation_report = dj.validate_catalog(datajson_url)

print validation_result
False

print validation_report
{
    "status": "ERROR",
    "error": {
        "catalog": {
            "status": "ERROR",
            "errors": [
                {
                    "instance": "",
                    "validator": "format",
                    "path": [
                        "publisher",
                        "mbox"
                    ],
                    "message": "u'' is not a u'email'",
                    "error_code": 2,
                    "validator_value": "email"
                }
            ]
        }
    }
}

```

```
        "instance": "",
        "validator": "minLength",
        "path": [
            "publisher",
            "name"
        ],
        "message": "'u' is too short",
        "error_code": 2,
        "validator_value": 1
    }
],
"title": "Andino"
},
"dataset": [
    {
        "status": "OK",
        "errors": [],
        "title": "Dataset Demo"
    }
]
}
```

Diccionario (data.json deserializado)

El siguiente fragmento de código tendrá resultados idénticos al primero:

```
import json
datajson_path = "tests/samples/full_data.json"

datajson = json.load(datajson_path)

validation_result = dj.is_valid_catalog(datajson)
validation_report = dj.validate_catalog(datajson)
(...)
```

1.4 Tests

Los tests se corren con `nose`. Desde la raíz del repositorio:

```
$ pip install nose # Sólo la primera vez
$ nosetests
```

1.5 Créditos

El validador de archivos `data.json` desarrollado es mayormente un envoltorio (*wrapper*) alrededor de la librería `jenschema`, que implementa el vocabulario definido por [JSONSchema.org](https://jsonschema.org/) para anotar y validar archivos JSON.

History

2.1 0.1.4 (2016-12-16)

- Al resultado de `DataJson.validate_catalog()` se le incorpora una lista (`"errors"`) con información de los errores encontrados durante la validación en cada nivel de jerarquía (“catalog” y cada elemento de “dataset”)

2.2 0.1.2 (2016-12-14)

- Se incorpora validación de tipo y formato de campo
- Los métodos `DataJson.is_valid_catalog()` y `DataJson.validate_catalog()` ahora aceptan un dict además de un `path/to/data.json` o una url a un `data.json`.

2.3 0.1.0 (2016-12-01)

Primera versión para uso productivo del paquete.

- La instalación via `pip install` debería reconocer correctamente la ubicación de los validadores por default.
- El manejo de `data.json`'s ubicados remotamente se hace en función del resultado de `urlparse.urlparse`
- El formato de respuesta de `validate_catalog` se adecuó a la última especificación (ver `samples/validate_catalog_returns.json`).

2.4 0.0.13 (2016-11-25)

- Intentar que la instalación del paquete sepa donde están instalados los schemas por default

2.5 0.0.12 (2016-11-25)

- Primera versión propuesta para v0.1.0

Indices y tablas

- `genindex`
- `modindex`
- `search`