FRICTIONLESS STANDARDS

Data Package

A simple container format for describing a coherent collection of data in a single 'package'. It provides the basis for convenient delivery, installation and management of datasets.

Author(s)	Paul Walsh, Rufus Pollock
Created	12 November 2007
Updated	2 May 2017
JSON Schema	data-package.json
Version	1

Language

The key words `MUST` , `MUST NOT` , `REQUIRED` , `SHALL` , `SHALL NOT` , `SHOULD` , `SHOULD NOT` , `RECOMMENDED` , `MAY` , and `OPTIONAL` in this document are to be interpreted as described in RFC 2119

Introduction

A Data Package consists of:

- Metadata that describes the structure and contents of the package
- Resources such as data files that form the contents of the package

The Data Package metadata is stored in a “descriptor”. This descriptor is what makes a collection of data a Data Package. The structure of this descriptor is the main content of the specification below.

In addition to this descriptor a data package will include other resources such as data files. The Data Package specification does NOT impose any requirements on their form or structure and can therefore be used for packaging **any kind of data**.

- Remote resources, referenced by URL
- "Inline" data (see below) which is included directly in the descriptor

Illustrative Structure

A minimal data package on disk would be a directory containing a single file:

```
datapackage.json  # (required) metadata and schemas for this data package
```

Lacking a single external source of data would make this of limited use. A slightly less minimal version would be:

```
datapackage.json
# a data file (CSV in this case)
data.csv
```

Additional files such as a README, scripts (for processing or analyzing the data) and other material may be provided. By convention scripts go in a scripts directory and thus, a more elaborate data package could look like this:

```
datapackage.json  # (required) metadata and schemas for this data package
README.md         # (optional) README in markdown format

# data files may go either in data subdirectory or in main directory
mydata.csv
data/otherdata.csv

# the directory for code scripts - again these can go in the base directory
scripts/my-preparation-script.py
```

Several example data packages can be found in the datasets organization on github , including:

- World GDP
- ISO 3166-2 country codes

Specification

Descriptor

The descriptor is the central file in a Data Package. It provides:

- General metadata such as the package's title, license, publisher etc
- A list of the data "resources" that make up the package including their location on disk or online and other relevant information (including, possibly, schema information about these data resources in a structured form)

A Data Package descriptor `MUST` be a valid JSON `object`. (JSON is defined in RFC 4627). When available as a file it `MUST` be named `datapackage.json` and it `MUST` be placed in the top-level directory (relative to any other resources provided as part of the data package).

The descriptor `MUST` contain a `resources` property describing the data resources.

All other properties are considered `metadata` properties. The descriptor `MAY` contain any number of other `metadata` properties. The following sections provides a description of required and optional metadata properties for a Data Package descriptor.

Adherence to the specification does not imply that additional, non-specified properties cannot be used: a descriptor `MAY` include any number of properties in additional to those described as required and optional properties. For example, if you were storing time series data and wanted to list the temporal coverage of the data in the Data Package you could add a property `temporal` (cf Dublin Core):

```js
"temporal": {
  "name": "19th Century",
  "start": "1800-01-01",
  "end": "1899-12-31"
}
```

This flexibility enables specific communities to extend Data Packages as appropriate for the data they manage. As an example, the Tabular Data Package specification extends

```js
{
  # general "metadata" like title, sources etc
  "name" : "a-unique-human-readable-and-url-usable-identifier",
  "title" : "A nice title",
  "licenses" : [ ... ],
  "sources" : [...],
  # list of the data resources in this data package
  "resources": [
    {
      ... resource info described below ...
    }
  ],
  # optional
  ... additional information ...
}
```

Resource Information

Packaged data resources are described in the `resources` property of the package descriptor. This property `MUST` be an array of `objects`. Each object `MUST` follow the Data Resource specification.

Metadata

Required Properties

The `resources` property is required, with at least one resource.

Recommended Properties

In addition to the required properties, the following properties `SHOULD` be included in every package descriptor:

`name`

A short url-usable (and preferably human-readable) name of the package. This `MUST` be

The name `SHOULD` be invariant, meaning that it `SHOULD NOT` change when a data package is updated, unless the new package version should be considered a distinct package, e.g. due to significant changes in structure or interpretation. Version distinction `SHOULD` be left to the version property. As a corollary, the name also `SHOULD NOT` include an indication of time range covered.

`id`

A property reserved for globally unique identifiers. Examples of identifiers that are unique include UUIDs and DOIs.

A common usage pattern for Data Packages is as a packaging format within the bounds of a system or platform. In these cases, a unique identifier for a package is desired for common data handling workflows, such as updating an existing package. While at the level of the specification, global uniqueness cannot be validated, consumers using the `id` property `MUST` ensure identifiers are globally unique.

Examples:

```js
{
  "id": "b03ec84-77fd-4270-813b-0c698943f7ce"
}
```

```js
{
  "id": "https://doi.org/10.1594/PANGAEA.726855"
}
```

`licenses`

The license(s) under which the package is provided.

This property is not legally binding and does not guarantee the package is licensed under the terms defined in this property.

`licenses` `MUST` be an array. Each item in the array is a License. Each `MUST` be an

Here is an example:

```js
"licenses": [{
  "name": "ODC-PDDL-1.0",
  "path": "http://opendatacommons.org/licenses/pddl/",
  "title": "Open Data Commons Public Domain Dedication and License v1.0"
}]
```

- `name` : The `name` `MUST` be an Open Definition license ID
- `path` : A url-or-path string, that is a fully qualified HTTP address, or a relative POSIX path (see the url-or-path definition in Data Resource for details).
- `title` : A human-readable title.

`profile`

A string identifying the profile of this descriptor as per the profiles specification.

Examples:

```js
{
  "profile": "tabular-data-package"
}
```

```js
{
  "profile": "http://example.com/my-profiles-json-schema.json"
}
```

Optional Properties

The following are commonly used properties that the package descriptor `MAY` contain:

`title`

A `string` providing a title or one sentence description for this package

`description`

A description of the package. The description `MUST` be markdown formatted – this also allows for simple plain text as plain text is itself valid markdown. The first paragraph (up to the first double line break) should be usable as summary information for the package.

`homepage`

A URL for the home on the web that is related to this data package.

`version`

A version string identifying the version of the package. It should conform to the Semantic Versioning requirements and should follow the Data Package Version pattern.

`sources`

The raw sources for this data package. It `MUST` be an array of Source objects. Each Source object `MUST` have a `title` and `MAY` have `path` and/or `email` properties. Example:

```js
"sources": [{
  "title": "World Bank and OECD",
  "path": "http://data.worldbank.org/indicator/NY.GDP.MKTP.CD"
}]
```

- `title` : title of the source (e.g. document or organization name)
- `path` : A url-or-path string, that is a fully qualified HTTP address, or a relative POSIX path (see the url-or-path definition in Data Resource for details).
- `email` : An email address

`contributors`

The people or organizations who contributed to this Data Package. It `MUST` be an array. Each entry is a Contributor and `MUST` be an `object` . A Contributor `MUST` have a `title` property and MAY contain `path` , `email` , `role` and `organization` properties. An example of the object structure is as follows:

```
    "email": "joe@bloggs.com",
    "path": "http://www.bloggs.com",
    "role": "author"
  }]
```

- `title` : name/title of the contributor (name for person, name/title of organization)
- `path` : a fully qualified http URL pointing to a relevant location online for the contributor
- `email` : An email address
- `role` : a string describing the role of the contributor. It's `RECOMMENDED` to be one of: `author` , `publisher` , `maintainer` , `wrangler` , and `contributor` . Defaults to `contributor` .
  - Note on semantics: use of the "author" property does not imply that that person was the original creator of the data in the data package - merely that they created and/or maintain the data package. It is common for data packages to "package" up data from elsewhere. The original origin of the data can be indicated with the `sources` property - see above.
- `organization` : a string describing the organization this contributor is affiliated to.

`keywords`

An Array of string keywords to assist users searching for the package in catalogs.

`image`

An image to use for this data package. For example, when showing the package in a listing.

The value of the image property `MUST` be a string pointing to the location of the image. The string must be a url-or-path, that is a fully qualified HTTP address, or a relative POSIX path (see the url-or-path definition in Data Resource for details).

`created`

The datetime on which this was created.

Note: semantics may vary between publishers – for some this is the datetime the data was created, for others the datetime the package was created.

FRICTIONLESS STANDARDS

```js
{
  "created": "1985-04-12T23:20:50.52Z"
}
```

Edit this page

Last Updated: 10/11/2022, 4:12:07 PM