DDFcsv datapackage

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.

DDFcsv datapackage is a structure which:

- packages a DDFcsv dataset
- describes the DDFcsv dataset's metadata
- describes the DDFcsv dataset's schema.

It is an extension of tabular datapackages (which extends datapackage) of the Frictionless Data project, run by the Open Knowledge Foundation.

A DDFcsv datapackage MUST have a `datapackage.json` file at the root of the DDFcsv dataset. The `datapackage.json` is the entry point to the dataset for automated readers and contains metadata for human readers.

This document goes on to describe `datapackage.json` for DDFcsv datapackages. Compared to tabular datapackages, DDFcsv datapackage adds stricter rules and some pre-defined fields. This means that a DDFcsv datapackage is always a tabular datapackage, but not vice-versa.
The added rules and fields are marked with a green highlight in this document. Unhighlighted rules are identical to those in tabular datapackage and added for clarity.

Implementations of this spec will first and foremost focus on the rules as described in this document but should aim to implement the complete spec where relevant, including tabular datapackages and datapackages.

Datapackage.json

Minimal version

An example of a minimal valid DDFcsv datapackage.json is the following:

```
{
    "name": "dataset-name",
    "resources": [
      {
        "path": "ddf--datapoints--population--by--geo--gender--year.csv",
        "name": "ddf--datapoints--population--by--geo--gender--year",
        "schema": {
          "fields": [
             { "name": "geo" },
             { "name": "year" },
             { "name": "gender" },
             { "name": "population" }
          ],
          "primaryKey": ["geo","gender","year"]
        }
      },
      ...
    ],
    "ddfSchema": {
      "datapoints": [
        {
          key: ["geo","year","gender"],
          value: "population",
          resources: ["ddf--datapoints--population--by--geo--gender--year"]
        }
      ],
      "entities": [ ... ],
      "concepts": [ ... ],
      "synonyms": [ ]
    }
}
```

A DDFcsv datapackage MUST contain the fields `name` and `resources`

Name

Name MUST be a string. Name MUST consist of solely lowercase alphanumeric characters, underscores, hyphens and periods. Name SHOULD be identical to the name of the folder or other structure (such as a github repo) it is stored in. Name SHOULD follow this naming guideline:

```
ddf--%dataset_provider%--%dataset_title%
```

Where %dataset_provider% SHOULD be the dataset provider and %dataset_title% SHOULD be similar to the dataset title. For example:

```
ddf--unpop--population
ddf--world_bank--world_development_indicators
ddf--gapminder--historical_life_expectancy
```

Resources

Resources MUST be an array containing resource objects. Each DDFcsv file in the dataset MUST be described by one resource object. Translation files MUST NOT be described in the `resources` array.

Resource object

Resource objects describe the resource and MUST contain the fields `path`, and `schema` and SHOULD contain the field `name`:

- `path` MUST be a string which contains the relative path to the DDFcsv file.
- `name` MUST be a string which MAY be the file name or file path of the resource, minus the extension. `name` MUST be unique to the datapackage, even when a dataset contains multiple files with the same filename (for example in different folders). `Name` MUST contain only lowercase alphanumeric characters, underscores (_), hyphens (-) or periods (.)
- `schema` MUST be an object which follows the DDFcsv JSON Table Schema spec as described in this document.

Conflicting `name`

The `name` field has to be unique across the dataset. But when it is solely the filename of the resource, identically named files can cause conflict. Thus, our simple guideline is to add a suffix `-#`, where # is a incrementing number, behind the filename for identically named files.
For example:

```
{
	"path": "ddf--entities--company.csv",
	"name": "ddf--entities--company",
	"schema": { ... }
},
{
	"path": "extra companies/ddf--entities--company.csv",
	"name": "ddf--entities--company-2",
	"schema": { ... }
}
```

DDF Schema

DDFcsv datapackage.json MUST contain a `ddfSchema` object, containing four fields: `datapoints`, `entities`, `concepts` and `synonyms`. Each of these fields has an array of key-value pair objects.
The `ddfSchema` SHOULD contain one object for each key-value pair in the dataset. The `ddfSchema` MUST NOT contain any key-value pairs not present in the dataset.

The ddfSchema is the entrypoint for any application to read the DDFcsv dataset.

The resource schemas described in the resources section are not adequate for efficient reading of a DDFcsv dataset. In DDF, an entity can be referenced by multiple concepts, because it can belong to multiple entity sets. Therefore, it is possible that data might be found under a different header in the csv file than described in the resource schema.
For example the entity `sgp`, representing Singapore, could be both in the entity sets `country` and `united_nations_state`, in the entity domain `geo`. Therefore it may be found in `ddf--entities--geo--country.csv`, `ddf--entities--geo.csv` and `ddf--entities--geo--united_nations_state.csv` to declare its properties. And datapoints describing data concerning Singapore can also refer to `country`, `united_nations_state` or `geo`:
`ddf--datapoints--population--by--geo--year.csv`,
`ddf--datapoints--population--by--country--year.csv`,
`ddf--datapoints--population--by--united_nations_state--year.csv`.
The above means that when an application wants to read a certain key-value pair in the DDF dataset, it has to look in files that contain other concepts in the same entity domain. `Country` data might be found in `united_nation_state` files or even `city` files (when Singapore is both a city and country).
To minimize the files needed to read, the ddfSchema enumerates per key-value pair all resources containing relevant data.

Unexpected key-value pairs in ddfSchema
Certain key-value pairs may be unexpected side-effects of entities' multi-set memberships. For example, Singapore might have a `national_anthem` property because it's a country. Additionally, it's also a city, leading to the key-value pair `<city>,national_anthem` being present in the dataset. However, a schema query returning that a `city` has a `national_anthem` might be unwanted.
A way to indicate this is to add an `expected` field to the key-value pair, with a value of false, to indicate the key-value pair may be unexpected.
Another option is to not include the key-value pair in the `ddfSchema` array at all. Then it will seem to not be part of the dataset at all. However you SHOULD NOT keep key-value pairs out of the `ddfSchema`. What might be unwanted in one use-case, can be interesting in others.

Since DDFcsv datasets are meant for open data and collaboration, you don't know yet what use cases your data might be used for.

Key-value pair object

A key-value pair object describes a key-value pair in the dataset and MUST include the fields `primaryKey`, `value` and `resources`.

- `primaryKey` must be an array of strings in which each string is a concept in the key of the key-value pair.
- `value` must be a string describing the value of the key-value pair or `null`
    - The key-value pair object with `value null` designates that for that key there is data but no key-value pair. In other words, there is only a list of keys. Only key-value pair objects for entities can have `value null`.
      Entities can exist in a dataset without any properties. In that case there are no key-value pairs, but there are keys: entities. The `value null` key-value pair object is needed to reflect the availability of these entities and help dataset readers to find these entities in the ddfcsv dataset.

frequency
1yearly
5yearly

ddf--entities--frequency.csv

frequency	**name**
10yearly	Every decade
20yearly	Every 20 years

more/ddf--entities--frequency.csv

```
{
  key: ["frequency"],
  value: null,
  resources: ["ddf--entities--frequency"]
},{
  key: ["frequency"],
  value: name,
  resources: ["more-ddf--entities--frequency"]
}
```

Note that the second file/table is not in the resources list for the `value null` key-value object because the table contains a key-value pair, so doesn't need a `value null` key-value object to describe it.

Concepts MUST have a concept_type property and thus each concept in the dataset is featured in at least one key-value pair (with value concept_type). Therefore, no `value null` key-value pair object is needed for concepts. Datapoints MUST have an indicator and thus each datapoint in the dataset is featured in at least one key-value pair. Therefore, no `value null` key-value pair object is needed for datapoints.

- `resources` must be an array of strings in which each string is a resource-name in the resources section of the datapackage.json. The resource MUST contain at least one row conforming the key-value pair described in this key-value pair object.
  You SHOULD NOT have redundant resources in your dataset. A redundant resource is a resource which contains only redundant data (i.e. all data also occurs in other files in the dataset). Because of the possibility to have redundant or duplicate data in DDFcsv datasets, a resource in the resources array might also be redundant. Redundant resources serve no purpose and are thus NOT RECOMMENDED.
  Redundant data and resources are only useful for readers when your reader is smart enough to use the redundancy to its benefit by selecting a set of resources which cover the query data but read the least amount of non-query data. For example, a query for cities will only read the cities resource and not the country resource (even though there is city data on singapore/hong kong) because it knows that all city data queried for in the country resource is also in the city resource.

The key-value pair object MAY include an `expected` field.

- `expected` MUST be a boolean which indicates if the key-value pair is expected in the dataset.

```
"ddfSchema": {
	"concepts": [{
		"primaryKey": ["concept"],
		"value": "name",
		"resources": ["ddf--concepts"]
	}, {
		"primaryKey": ["concept"],
		"value": "concept_type",
		"resources": ["ddf--concepts"]
	}],
	"entities": [{
		"primaryKey": ["geo"],
		"value": "name",
		"resources": ["ddf--entities--geo"]
	},{
```

```
            "primaryKey": ["geo"],
            "value": null,
            "resources": ["ddf--entities--geo"]
        },{
            "primaryKey": ["country"],
            "value": "name",
            "resources": ["ddf--entities--geo"],
            "expected": false
        }],
        "datapoints": [{
            "primaryKey": ["country","year"],
            "value": "population",
            "resources": ["ddf--datapoints--population--by--year--geo",
                          "ddf--datapoints--population--by--year--country"]
        }, {
            "primaryKey": ["geo","year"],
            "value": "population",
            "resources": ["ddf--datapoints--population--by--year--geo",
                          "ddf--datapoints--population--by--year--country"]
        }],
        "synonyms": []
}
```

Recommended fields

The fields `title`, `description`, `author` and `license` SHOULD be fields in datapackage.json. For more explanation about what information they must contain, see the tabular datapackage spec.

Language and translations

A DDFcsv dataset SHOULD contain strings in only one language. When a DDFcsv datapackage contains strings in only one language, that language SHOULD be described using the `language` field.
A DDFcsv dataset MAY contain data in multiple languages. When it does, the original language and translations MUST be described using the `language` and `translations` fields.
The `language` and `translations` fields MUST be at the root of `datapackage.json`.
`language` MUST be one language object while `translations` MUST be an array containing one or more language objects.

Language object

A language object MUST have the field `id` and MAY have the field `name`.

- `id` MUST be a string with the language tag as described in the language and translation section of the DDF specs.
  If the Language Object is part of the `translations` field, a folder with the same name as `id` must exist in the `/lang/` folder of the dataset.
- `name` SHOULD be the native name of the language (i.e. as written in the language itself)

Look at the typical datapackage.json below for an example of translation and language fields.

DDFcsv JSON Table Schema

DDFcsv JSON Table Schema (DJTS) is an extension of the JSON Table Schema spec (JTS). DDFcsv datapackages use DJTS to describe the schema of DDFcsv files.

Schema object

The schema object MUST contain the fields `fields` and `primaryKey`.

Fields

Fields MUST be an array of Field Objects. Each Field Object describes one field in the DDFcsv file. The order of Field Objects in the array must be identical to the order of fields in the DDFcsv file.

Field Object

A field object MUST contain the field `name` which MUST be a string identical to the field name in the csv.

Constraints

When the DDFcsv file is a datapoint file, which follows the split files by dimension value guideline, the Field Object SHOULD contain the field `constraints`, with subfield `enum`.

For example:

```
{
  "path": "ddf--datapoints--population--by--geo-usa--gender--year.csv",
  "name": "ddf--datapoints--population--by--geo-usa--gender--year",
  "schema": {
    "fields": [
       { "name": "geo"
         "constraints": {
           "enum": ["usa"]
         }
```

```
        },
        { "name": "year",  },
        { "name": "gender" },
        { "name": "population" }
      ],
      "primaryKey": ["geo","gender","year"]
    }
}
```

PrimaryKey

`primaryKey` MUST be a string or array describing the fields which form the primary key in the DDFcsv file. For details, please refer to the primaryKey in the JTS spec.
Fields mentioned in the `fields` array which are not in the `primaryKey` form the values in that table. In other words, they are indicators, entity properties or concept properties.

Typical datapackage.json example without ddfSchema

Below is an example of typical content of datapackage.json.

```
{
  "name": "ddf--unpop--population",
  "title": "UNPOP Population"
  "description": "Population statistics from the UN Population Division"
  "language": [
    { "id": "en",
      "name": "English"
    }
  ],
  "translations": [
    { "id": "nl-NL",
      "name": "Nederlands (Nederland)"
    },
    { "id": "ru",
      "name": "русский"
    }
  ],
  "license": "BSD-3-Clause"
  "author": "Gapminder <info@gapminder.org> (http://gapminder.org)",
  "resources": [
      {
        "path": "ddf--datapoints--population--by--geo-usa--gender--year.csv",
        "name": "ddf--datapoints--population--by--geo-usa--gender--year",
        "schema": {
          "fields": [
             { "name": "geo"
               "constraints": {
                 "enum": ["usa"]
               }
             },
             { "name": "year" },
             { "name": "gender" },
             { "name": "population" }
         ],
         "primaryKey": ["geo","gender","year"]
      },
      {
        "path": "ddf--datapoints--population--by--geo-swe--gender--year.csv",
        "name": "ddf--datapoints--population--by--geo-swe--gender--year",
        "schema": {
          "fields": [
             { "name": "geo"
               "constraints": {
                 "enum": ["swe"]
               }
             },
             { "name": "year" },
             { "name": "gender" },
             { "name": "population" }
         ],
         "primaryKey": ["geo","gender","year"]
      },
      { ... },
      ...
    ]
}
```