FRICTIONLESS STANDARDS

Data Resource

A simple format to describe and package a single data resource such as a individual table or file.

Author(s)	Paul Walsh, Rufus Pollock
Created	11 December 2016
Updated	17 April 2018
JSON Schema	data-resource.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

The **Data Resource** format describes a data resource such as an individual file or table.
The essence of a Data Resource is a locator for the data it describes.
A range of other properties can be declared to provide a richer set of metadata.

Examples

A minimal Data Resource looks as follows:

With data accessible via the local filesystem.

```js
{
```

With data accessible via http.

```js
{
  "name": "resource-name",
  "path": "http://example.com/resource-path.csv"
}
```

A minimal Data Resource pointing to some inline data looks as follows.

```js
{
  "name": "resource-name",
  "data": {
    "resource-name-data": [
      {"a": 1, "b": 2}
    ]
  },
}
```

A comprehensive Data Resource example with all required, recommended and optional properties looks as follows.

```js
{
  "name": "solar-system",
  "path": "http://example.com/solar-system.csv",
  "title": "The Solar System",
  "description": "My favourite data about the solar system.",
  "format": "csv",
  "mediatype": "text/csv",
  "encoding": "utf-8",
  "bytes": 1,
  "hash": "",
  "schema": "",
  "sources": "",
  "licenses": ""
}
```

FRICTIONLESS
STANDARDS

Descriptor

A Data Resource descriptor MUST be a valid JSON `object`. (JSON is defined in RFC 4627).

Key properties of the descriptor are described below. A descriptor MAY include any number of properties in additional to those described below as required and optional properties.

Data Location

A resource MUST contain a property describing the location of the data associated to the resource. The location of resource data MUST be specified by the presence of one (and only one) of these two properties:

- `path` : for data in files located online or locally on disk.
- `data` : for data inline in the descriptor itself.

`path` Data in Files

`path` MUST be a string – or an array of strings (see “Data in Multiple Files”). Each string MUST be a “url-or-path” as defined in the next section.

URL or Path

A “url-or-path” is a `string` with the following additional constraints:

- `MUST` either be a URL or a POSIX path
- URLs MUST be fully qualified. MUST be using either http or https scheme. (Absence of a scheme indicates `MUST` be a POSIX path)
- POSIX paths (unix-style with `/` as separator) are supported for referencing local files, with the security restraint that they `MUST` be relative siblings or children of the descriptor. Absolute paths (/) and relative parent paths (…/) MUST NOT be used, and implementations SHOULD NOT support these path types.

Examples:

```
# relative path
# note: this will work both as a relative path on disk and on online
"path": "my-data-directory/my-csv.csv"
```

WARNING

`/` (absolute path) and `../` (relative parent path) are forbidden to avoid security vulnerabilities when implementing data package software. These limitations on resource `path` ensure that resource paths only point to files within the data package directory and its subdirectories. This prevents data package software being exploited by a malicious user to gain unintended access to sensitive information.

For example, suppose a data package hosting service stores packages on disk and allows access via an API. A malicious user uploads a data package with a resource path like `/etc/passwd` . The user then requests the data for that resource and the server naively opens `/etc/passwd` and returns that data to the caller.

Prior to release 1.0.0-beta.18 (Nov 17 2016) there was a `url` property distinct from `path` . In order to support backwards compatibility, implementors MAY want to automatically convert a `url` property to a `path` property and issue a warning.

Data in Multiple Files

Usually, a resource will have only a single file associated to it. However, sometimes it may be convenient to have a single resource whose data is split across multiple files – perhaps the data is large and having it in one file would be inconvenient.

To support this use case the `path` property MAY be an array of strings rather than a single string:

```
"path": [ "myfile1.csv", "myfile2.csv" ]
```

It is NOT permitted to mix fully qualified URLs and relative paths in a `path` array: strings

Implementors MUST be able to concatenate together the files in the simplest way and treat the result as one large file. For tabular data there is the issue of header rows. See the Tabular Data Package spec for more on this.

`data` Inline Data

Resource data rather than being stored in external files can be shipped `inline` on a Resource using the `data` property.

The value of the data property can be any type of data. However, restrictions of JSON require that the value be a string so for binary data you will need to encode (e.g. to Base64). Information on the type and encoding of the value of the data property SHOULD be provided by the format (or mediatype) property and the encoding property.

Specifically: the value of the data property MUST be:

- EITHER: a **JSON** array or **Object**- the data is then assumed to be JSON data and SHOULD be processed as such
- OR: a **JSON** string - in this case the format or mediatype properties MUST be provided.

Thus, a consumer of resource object MAY assume if no format or mediatype property is provided that the data is JSON and attempt to process it as such.

Examples 1 - inline JSON:

```
{
   ...
   "resources": [
     {
        "format": "json",
        # some json data e.g.
        "data": [
           { "a": 1, "b": 2 },
           { .... }
        ]
     }
   ]
}
```

```
{
  ...
  "resources": [
    {
      "format": "csv",
      "data": "A,B,C\n1,2,3\n4,5,6"
    }
  ]
}
```

Metadata Properties

Required Properties

A descriptor MUST contain the following properties:

`name`

A resource MUST contain a `name` property. The name is a simple name or identifier to be used for this resource.

- If present, the name MUST be unique amongst all resources in this data package.
- It MUST consist only of lowercase alphanumeric characters plus ".", "-" and "_".
- It would be usual for the name to correspond to the file name (minus the extension) of the data file the resource describes.

Recommended Properties

`profile`

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

Examples:

```
}
```

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

Optional Properties

A descriptor MAY contain any number of additional properties. Common properties include:

- `title` : a title or label for the resource.
- `description` : a description of the resource.
- `format` : 'csv', 'xls', 'json' etc. Would be expected to be the standard file extension for this type of resource.
- `mediatype` : the mediatype/mimetype of the resource e.g. "text/csv", or "application/vnd.ms-excel". Mediatypes are maintained by the Internet Assigned Numbers Authority (IANA) in a media type registry .
- `encoding` : specify the character encoding of the resource's data file. The values should be one of
  the "Preferred MIME Names" for a character encoding registered with IANA . If no value for this key is specified then the default is UTF-8.
- `bytes` : size of the file in bytes.
- `hash` : the MD5 hash for this resource. Other algorithms can be indicated by prefixing the hash's value with the algorithm name in lower-case. For example:

  ```
  "hash": "sha1:8843d7f92416211de9ebb963ff4ce28125932878"
  ```

- `sources` : as for Data Package metadata.
- `licenses` : as for Data Package metadata. If not specified the resource

Resource Schemas

A Data Resource MAY have a `schema` property to describe the schema of the resource data.

The value for the `schema` property on a `resource` MUST be an `object` representing the schema OR a `string` that identifies the location of the schema.

If a `string` it must be a url-or-path as defined above, that is a fully qualified http URL or a relative POSIX path. The file at the location specified by this url-or-path string `MUST` be a JSON document containing the schema.

NOTE: the Data Package specification places no restrictions on the form of the schema Object. This flexibility enables specific communities to define schemas appropriate for the data they manage. As an example, the Tabular Data Package specification requires the schema to conform to Table Schema.

Edit this page

Last Updated: 12/7/2021, 9:10:42 PM