

# **Bidirectional conversion to/from CSV for nested JSON data**

**Ben Webb**



**github.com/  
OpenDataServices/  
flatten-tool/**

Abstract:

A well defined nested format like JSON can be useful for defining a data standard. However, not everyone finds it easy to publish and consume JSON. For the Open Contracting and 360Giving data standards we've taken the hybrid approach of a canonical JSON representation with bidirectional conversion to/from spreadsheets. Since this involves converting between nested and flat representations we've called our software Flatten-Tool: <https://github.com/OpenDataServices/flatten-tool/>



A quick introduction to what I do, so it makes sense when I use these names later in the talk.

I work for a small workers co-op called Open Data Services

<http://opendataservices.coop/>

Part of what we do is provide technical help desk and software tool development for data standards including

The Open Contracting Data Standard (OCDS), which is for public disclosure on all stages of a contracting process:

<http://standard.open-contracting.org/latest/en/>

And 360Giving, which is a UK grants standard.

<http://www.threesixtygiving.org/>

# Why multiple formats?

In order to care about conversion, it means we care about multiple formats in the first place, why is this?



## Data on the Web Best Practices

### **Best Practice 15: Provide data in multiple formats**

*Data should be available in multiple data formats.*

---

#### **Why**

Providing data in more than one format reduces costs incurred in data transformation. It also minimizes the possibility of introducing errors in the process of transformation. If many users need to transform the data into a specific data format, publishing the data in that format from the beginning saves time and money and prevents errors many times over. Lastly it increases the number of tools and applications that can process the data.

#### **Intended Outcome**

It should be possible for data consumers to work with the data without transforming it.

#### **Possible Approach to Implementation**

Consider the data formats most likely to be needed by intended users, and consider alternatives that are likely to be useful in the future. Data publishers must balance the effort required to make the data available in many formats, but providing at least one alternative will greatly increase the usability of the data.

There's a few different reasons for this, but one comes from number #15 of the W3C's best practices for data on the web:

<https://www.w3.org/TR/dwbp/#dataFormats>

Basically, make open data available in a range of formats to meet the needs of different users. Developers may want JSON, researchers might prefer a spreadsheet format.

## Why not just CSV?

- “RFC 4180 proposes a specification for the CSV format, and this is the definition commonly used. However, in popular usage "CSV" is not a single, well-defined format.”
- Encodings differ between developer best practice (UTF-8) and Excel's export (Windows-1252)

Quote from wikipedia:

[https://en.wikipedia.org/wiki/Comma-separated\\_values#Specification](https://en.wikipedia.org/wiki/Comma-separated_values#Specification)

People sometimes claim CSV as both:

- \* A potentially well defined format that developers will like producing / consuming
- \* An easy format for less technical publishers to produce, by export from their existing software.

However these definitions of CSV are not consistent.

One particular problem is that CSV doesn't specify encoding information. Modern developers are often of the opinion “everything UTF-8”, but Excel doesn't play nicely for this!

# flatten-tool

<https://github.com/OpenDataServices/flatten-tool/>

Therefore for OCDS and 360Giving, we use a better defined, and more consistently understood format like JSON as our canonical representation.

We still want to provide spreadsheet users a means to publish and use this data, so we want conversion in both directions between spreadsheets and JSON.

This is what flatten-tool does. It flattens nested JSON into spreadsheet, and vice versa.

flatten-tool is an open source command line tool and Python library.

We also make use of it as a library in the web based validation and data quality tools for 360Giving and OCDS.

# XLSX Conf?

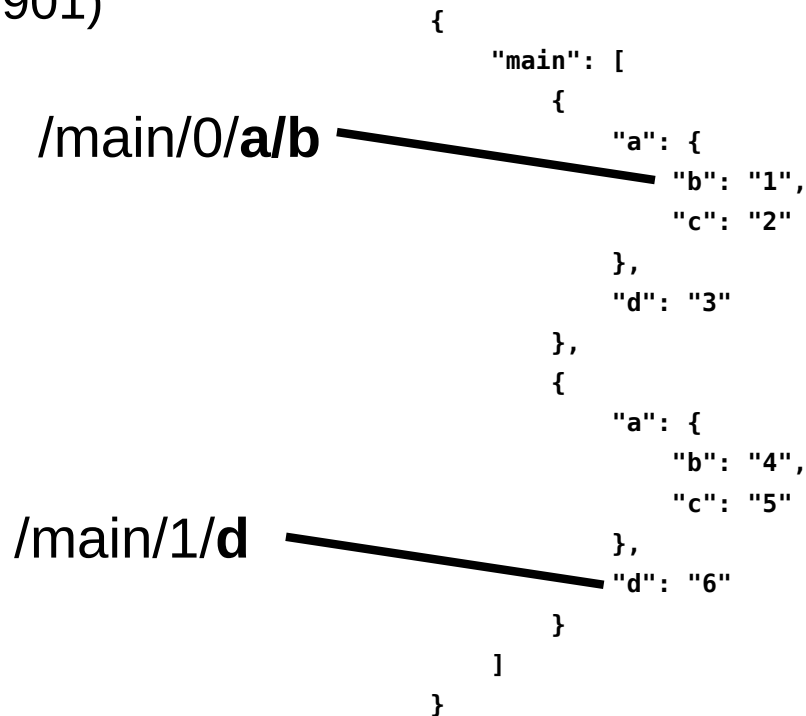
Part of our model with flatten-tool is to isolate dealing with the quirks of spreadsheets produced by less technical users in one piece of software.

Throughout this talk, I'm going to be saying "Spreadsheet", rather than CSV, because flatten-tool supports both CSV and XLSX.

Whilst XLSX has many bad properties as a data interchange format it gives us some extra metadata about what spreadsheet users were trying to do (encodings, number vs string formats etc.).

The flatten-tool model "lets us do this" because it isolates the pain of dealing with XLSX files in one tool, and everything else deals with nice JSON.

(JSON Pointer:  
RFC6901)



How does it work?

We've made some assumptions based on the data we're working with.

We assume we mostly care about a list of many similar JSON objects (with possibly some extra metadata at the top level).

We can identify each value in our JSON using **JSON pointer**, which looks like this...

We then remove the `/main/{number}` and use the remainder (in bold) as our column headings.



<b>a/b</b>	<b>a/c</b>	<b>d</b>
1	2	3
4	5	6

```

{
  "main": [
    {
      "a": {
        "b": "1",
        "c": "2"
      },
      "d": "3"
    },
    {
      "a": {
        "b": "4",
        "c": "5"
      },
      "d": "6"
    }
  ]
}

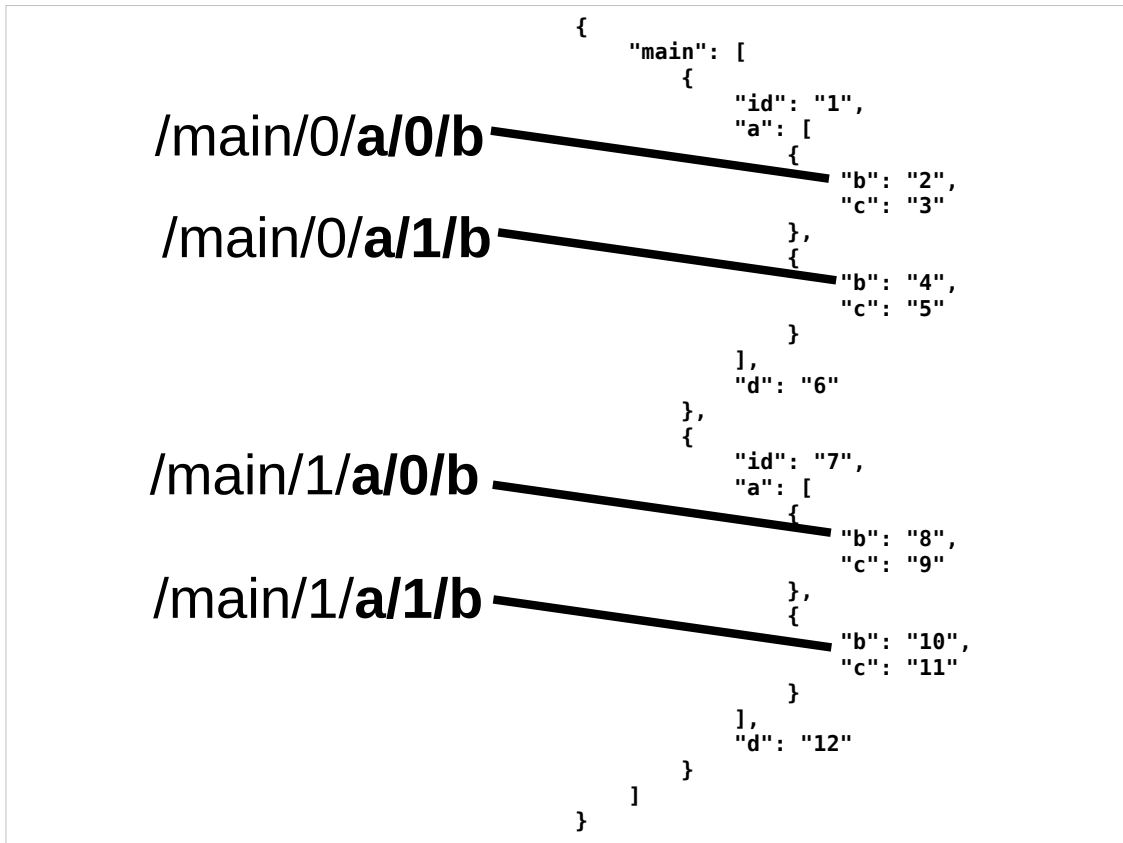
```

Here's what the spreadsheet looks like.

Flatten-tool can convert between these in either direction.

The top level key “main” is specified as a commandline argument.

This is somewhat “nested”, but what's more interesting is how to deal with arrays.



This is what the JSON pointer for items in arrays looks like.

Again the part we'd use for our spreadsheet headings is in bold.

There's a few different "shapes" of spreadsheet that you could use to represent this (so the next few slides will all have this same JSON on the right hand side):

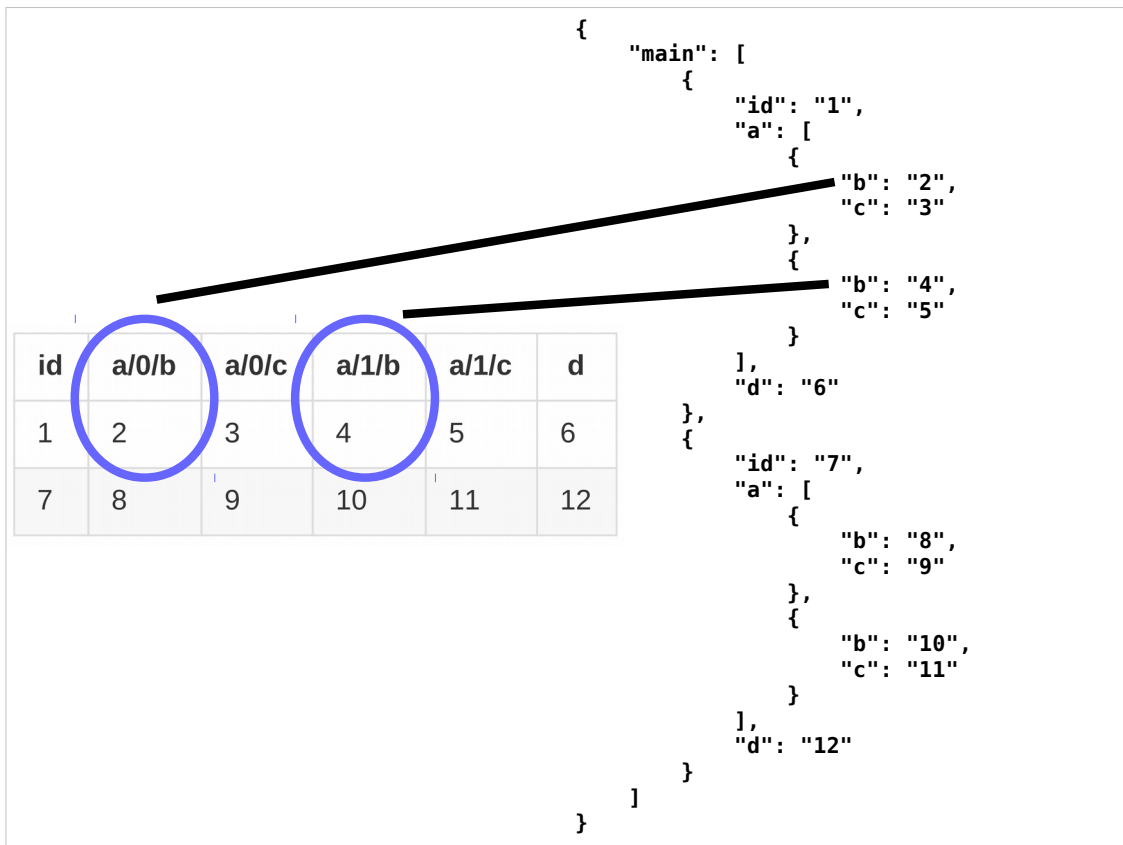
id	a/0/b	a/0/c	a/1/b	a/1/c	d
1	2	3	4	5	6
7	8	9	10	11	12

```

{
  "main": [
    {
      "id": "1",
      "a": [
        {
          "b": "2",
          "c": "3"
        },
        {
          "b": "4",
          "c": "5"
        }
      ],
      "d": "6"
    },
    {
      "id": "7",
      "a": [
        {
          "b": "8",
          "c": "9"
        },
        {
          "b": "10",
          "c": "11"
        }
      ],
      "d": "12"
    }
  ]
}

```

... the simplest of these is to have one column for each JSON pointer.



This means there's one column for entry in the array.

Flatten-tool supports this for unflattening (spreadsheet → JSON) but not the other way round atm.

This is not very useful for large arrays, but can be useful for smaller cases. e.g. useful when the standard has an array for additional Classifications, and people have two additional classifications they want to add, they can easily just double up the classification columns in their spreadsheet).

id	d
1	6
7	12

id	a/0/b	a/0/c
1	2	3
1	4	5
7	8	9
7	10	11

```

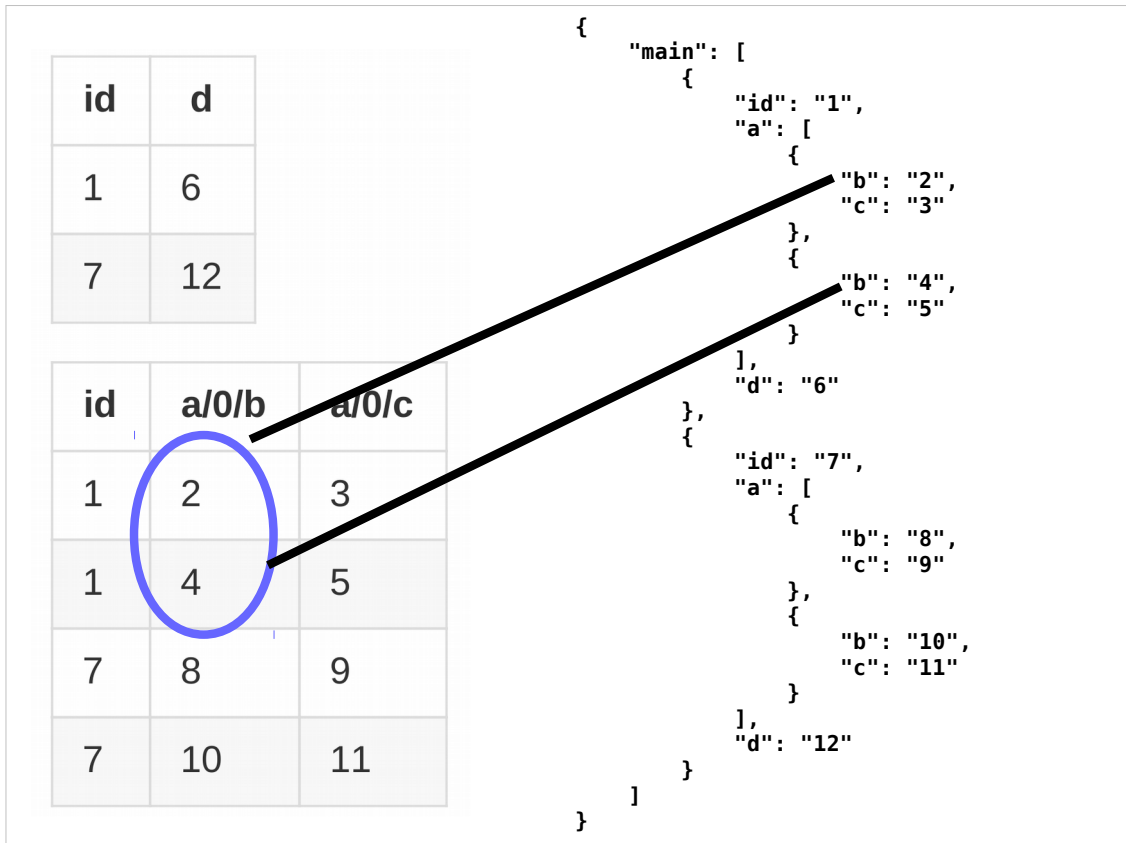
{
  "main": [
    {
      "id": "1",
      "a": [
        {
          "b": "2",
          "c": "3"
        },
        {
          "b": "4",
          "c": "5"
        }
      ],
      "d": "6"
    },
    {
      "id": "7",
      "a": [
        {
          "b": "8",
          "c": "9"
        },
        {
          "b": "10",
          "c": "11"
        }
      ],
      "d": "12"
    }
  ]
}

```

For larger arrays what we really want is some sort of relational structure.

flatten-tool implements this by having multiple sheets (or multiple CSVs).

This is the shape of spreadsheet that we support for conversion in both directions.



So you can see that consecutive array items are in different rows in one column of the related sheet.

id	d
1	6
7	12

id	a/0/b	a/0/c
1	2	3
1	4	5
7	8	9
7	10	11

```
{
  "main": [
    {
      "id": "1",
      "a": [
        {
          "b": "2",
          "c": "3"
        },
        {
          "b": "4",
          "c": "5"
        }
      ],
      "d": "6"
    },
    {
      "id": "7",
      "a": [
        {
          "b": "8",
          "c": "9"
        },
        {
          "b": "10",
          "c": "11"
        }
      ],
      "d": "12"
    }
  ]
}
```

In order to relate the two sheets, we need a common id column for both.

The 360Giving and OCDS standards mandate these ids because they're useful for relational data models like these.

id	a/0/b	a/0/c	d
1	2	3	6
1	4	5	6
7	8	9	12
7	10	11	12

```

{
  "main": [
    {
      "id": "1",
      "a": [
        {
          "b": "2",
          "c": "3"
        },
        {
          "b": "4",
          "c": "5"
        }
      ],
      "d": "6"
    },
    {
      "id": "7",
      "a": [
        {
          "b": "8",
          "c": "9"
        },
        {
          "b": "10",
          "c": "11"
        }
      ],
      "d": "12"
    }
  ]
}

```

We also support conversion **//from//** a third shape which is single sheet format with repeated rows with duplicate values for items not in the array.

((((((I think this format is more confusing than multiple sheets, but some existing tools use it, and it was easy for us to implement.))))))



id	a/0/id
1	2
1	4
7	8
7	10

id	d
1	6
7	12

id	a/0/id	a/0/b/0/c
1	2	3
1	2	3a
1	4	5
1	4	5a
7	8	9
7	8	9a
7	10	11
7	10	11a

```

{
  "main": [
    {
      "id": "1",
      "d": "6",
      "a": [
        {
          "id": "2",
          "b": [
            {
              "c": "3"
            },
            {
              "c": "3a"
            }
          ]
        },
        {
          "id": "4",
          "b": [
            {
              "c": "5"
            },
            {
              "c": "5a"
            }
          ]
        }
      ]
    }
  ]
},

```

Back to multisheet...

We can describe arrays nested within arrays by having all the necessary sheets....

id	a/0/id
1	2
1	4
7	8
7	10

id	d
1	6
7	12

id	a/0/id	a/0/b/0/c
1	2	3
1	2	3a
1	4	5
1	4	5a
7	8	9
7	8	9a
7	10	11
7	10	11a

```

{
  "id": "7",
  "a": [
    {
      "id": "8",
      "b": [
        { "c": "9" },
        { "c": "9a" }
      ]
    },
    {
      "id": "10",
      "b": [
        { "c": "11" },
        { "c": "11a" }
      ]
    }
  ]
}

```

... and id fields for every parent we need to identify.

## Using the JSON Schema

- Types
- Templates
- Titles
- Automagic arrays

So, the basic examples we've just seen demonstrate what flatten-tool can do without any schema.

360Giving and OCDS both have a JSON schema which flatten-tool can use to infer extra information (although the non-schema behaviour is also important because in both people can specify extra fields not in the schema).

Types:

We can use the types from the schema, e.g. to determine whether a number in a CSV should actually be a JSON number or a string.

Templates: Next slide

```

1  {
2  "id": "http://standard.open-contracting.org/schema/1_0_1/release-schema.json",
3  "$schema": "http://json-schema.org/draft-04/schema#",
4  "title": "Schema for an Open Contracting Release",
5  "type": "object",
6  "properties": {
7    "ocid": {
8      "title": "Open Contracting ID",
9      "description": "A globally unique identifier for this Open Contracting Proc
10     "type": "string",
11     "mergeStrategy": "ocdsOmit"
12   },
13   "id": {
14     "title": "Release ID",
15     "description": "A unique identifier that identifies this release. A release
16     "type": "string",
17     "mergeStrategy": "ocdsOmit"
18   },
19   "date": {
20     "title": "Release Date",
21     "description": "The date this information is released, it may well be the s
22     "type": "string",
23     "format": "date-time",
24     "mergeStrategy": "ocdsOmit"
25   },
26   "tag": {
27     "title": "Release Tag",

```

In order to create their spreadsheet data, publishers often want a “template” of the correct column headers to fill in.

flatten-tool can generate these.

So from the OCDS JSON schema, we can produce...

1	ocid	id	date	tag	initiationType	planning/budget/source	planning/budget/id
1	ocid	id	tender/id	tender/items/0/id	tender/items/0/description	tender/items/0/classification/scheme	tender/items/0/classification/id
1	ocid	id	tender/id	tender/items/0/id	tender/items/0/additionalClassifications/0/scheme	tender/items/0/additionalClassifications/0/id	

<a href="#">awa_ame_changes.csv</a>
<a href="#">awa_documents.csv</a>
<a href="#">awa_ite_additionalClassificatio.csv</a>
<a href="#">awa_items.csv</a>
<a href="#">awa_sup_additionalIdentifiers.csv</a>
<a href="#">awa_suppliers.csv</a>
<a href="#">awards.csv</a>
<a href="#">buy_additionalIdentifiers.csv</a>
<a href="#">con_ame_changes.csv</a>
<a href="#">con_documents.csv</a>

Multiple csvs in the multisheet shapes or one multisheet xlsx.

## Using the schema

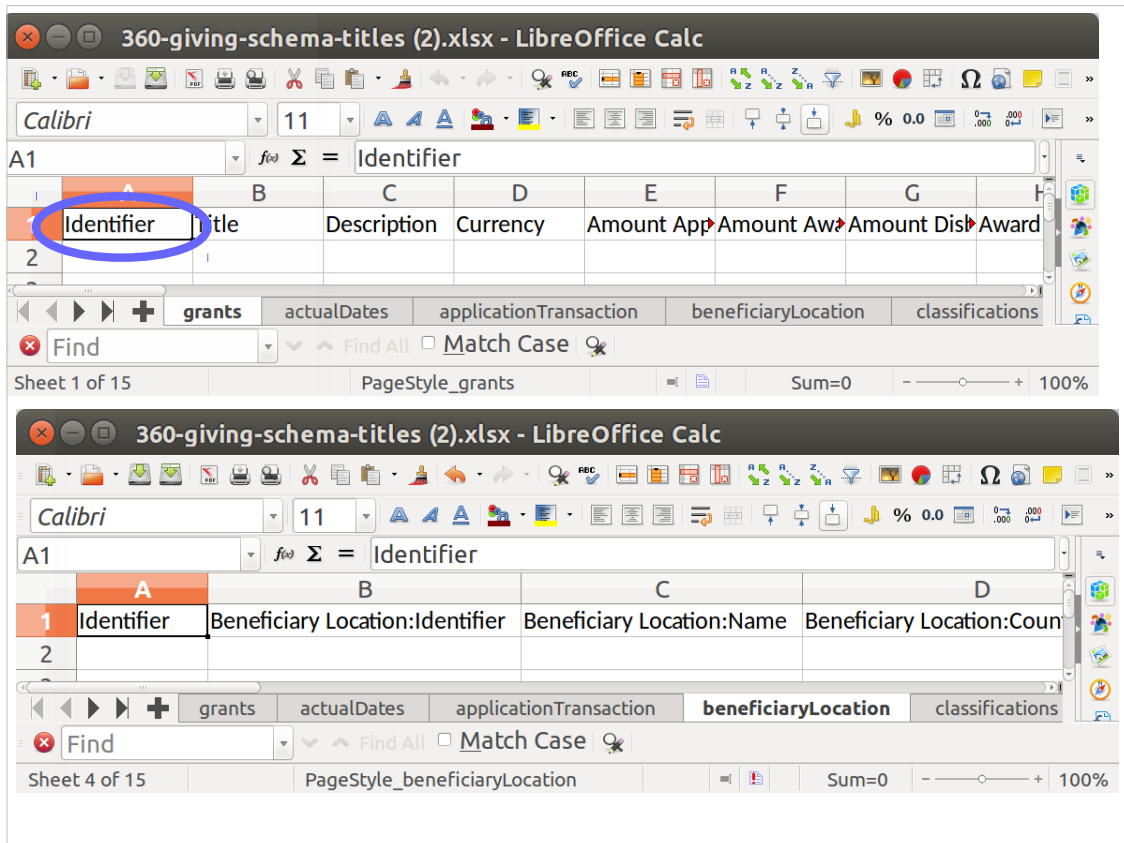
- Types
- Templates
- Titles
- Automagic arrays

**Titles:**

```
431 "properties": {
432   "id": {
433     "type": "string",
434     "description": "The unique identifier for this grant. Made up of youi
435     "weight": 0.001,
436     "title": "Identifier"
437   },
438   "title": {
439     "type": "string",
440     "description": "A title for this grant activity. This should be unde
441     "weight": 1.05,
442     "title": "Title"
443   },
444   "description": {
445     "type": "string",
446     "description": "A short description of this grant activity.",
447     "weight": 9,
448     "title": "Description"
449   },
450   "currency": {
451     "$ref": "#/definitions/currency",
452     "description": "The currency used in grant amounts and transactions u
453   },
454   "amountAppliedFor": {
```

For 360Giving, where we're trying to get small UK charitable trusts to publish data about their grants, the JSON pointer style headings are a bit too unfriendly.

Therefore, we make use of the more human readable titles from the JSON schema.

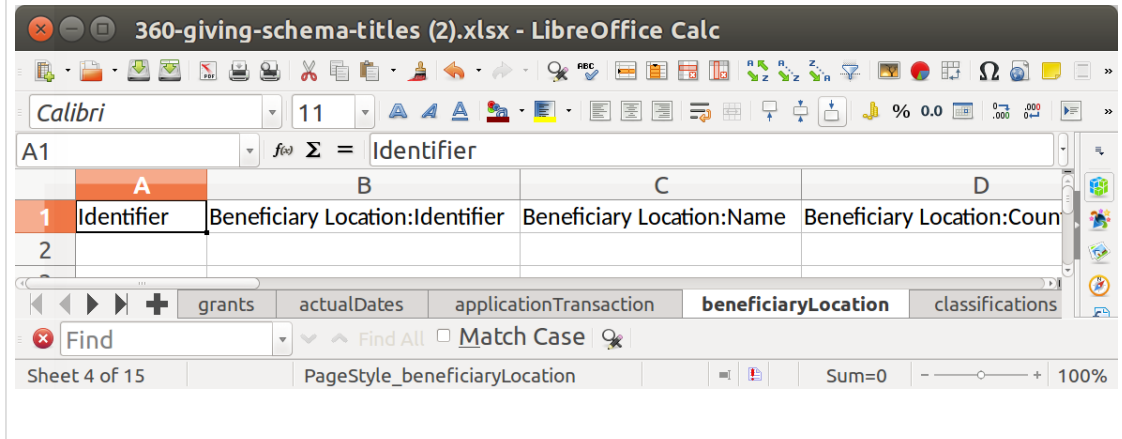


So we can generate a template with these titles, and flatten-tool also supports these for conversion.

Additionally these titles are case and space insensitive, to make it harder for publishers to “break” the template.



Beneficiary Location:Identifier → beneficiaryLocation/0/id  
Beneficiary Location:Name → beneficiaryLocation/0/name



As part of this titles support, we're using colons instead of slashes, as we think this is more intuitive to many spreadsheet users who would read '/' as either or.

Finally, we drop the numbers used to indicate arrays from the JSON pointers, as it “looks confusing”, and we're able to infer it from the schema instead.

## What's next?

- Continuing to use to help publishers and users of the 360Giving and OCDS standards
- Continuing to tidy up the code
- Sharing our work with the community.

Possibly:

- Supporting more standards (e.g. XML for IATI)
- Support for more spreadsheet formats (e.g. ODS, data packages)

(((Continuing to use to help publishers and users of the 360Giving and OCDS standards)))

Fixing bugs, making it easier to use  
Produce better warnings and error messages

(((Continuing to tidy up the code)))

The projects about a year and a half old now, and we've recently made some changes to many-to-one relationships based problems we found with the previous approach.

These changes are done now, but there's still plenty of refactoring & cleanup to be done as a result.

## What's next?

- Continuing to use to help publishers and users of the 360Giving and OCDS standards
- Continuing to tidy up the code
- Sharing our work with the community.

Possibly:

- Supporting more standards (e.g. XML for IATI)
- Support for more spreadsheet formats (e.g. ODS, data packages)

- (((Sharing our work with the community.))
  - We've been keeping fairly quiet about flatten-tool until we've made those changes. Now that we have, we're interested in trying to share it more widely, as it may be useful to other people (hence this talk).

....Possibly:

- Supporting more standards (e.g. we've discussed the possibility of adding XML support to flatten IATI data)
- Support for more spreadsheet formats (e.g. ODS (open office's spreadsheet format), Open Knowledge's data packages)

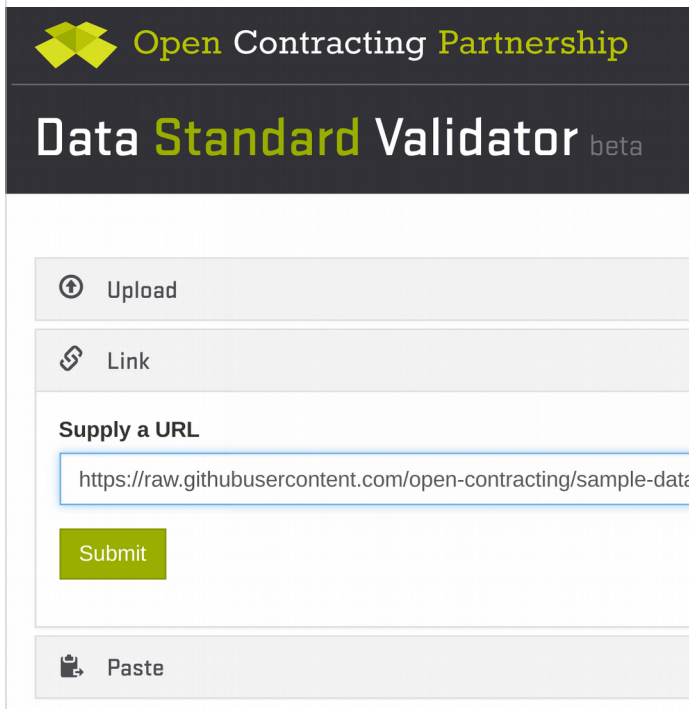
**Ben Webb**



**ben.webb@  
opendataservices.coop**

**github.com/  
OpenDataServices/  
flatten-tool/**

# CoVE - Convert, Validate, Explore



The screenshot shows the CoVE Data Standard Validator interface. At the top, there is a dark header with the Open Contracting Partnership logo and the text "Data Standard Validator beta". Below the header, there are four main sections: "Upload" with a plus icon, "Link" with a chain icon, "Supply a URL" with a text input field containing "https://raw.githubusercontent.com/open-contracting/sample-data" and a green "Submit" button, and "Paste" with a clipboard icon.

Cove - <http://cove.opendataservices.coop/>

Powers the OCDS Validator and the 360Giving Data Quality Tool.

Amongst other features, provides a web interface to flatten-tool for these standards.

Open Contracting Partnership

# Data Standard Validator beta

Download Files

JSON (Original) 5.5 KB

### Key Facts

Data downloaded from <https://raw.githubusercontent.com/open-contracting/san>

This package contains 1 release(s) describing 1 contracting process(es).

The publisher named in the file is Open Data Services Co-operative Limited. TI

You get a button to click to convert.

If you supply a spreadsheet we do the conversion automatically, because Cove's other functionality e.g. validation works on the JSON.



# Data Standard Validator beta

## Download Files

JSON (Original) 5.5 KB

Excel Spreadsheet (.xlsx) (Converted from Original) 22.8 KB

## Conversion Warnings

Skipping field "tender/amendment/changes/0/former\_value/", because it has no properties.

Skipping field "awards/0/amendment/changes/0/former\_value/", because it has no properties.

flattened (5).xlsx - LibreOffice Calc

Calibri 11 % 0.0

A1  $f_{\text{ed}} \Sigma = \text{ocid}$

	A	B	C	D	E	F	G	H
1	ocid	id	date	tag	initiationType	planning/bu	planning/bu	planning/bu
2	ocds-213czf	ocds-213czf	2010-03-15	tender	tender			
3								

releases additionalClassifications awa\_ame\_changes awa\_documents awa\_ite\_addit

Find Find All Match Case

Sheet 1 of 31 PageStyle\_releases Sum=0 100%

flattened (5).xlsx - LibreOffice Calc

Calibri 11 % 0.0

E2  $f_{\text{ed}} \Sigma = \text{CPV}$

	A	B	C	D	E	F
1	ocid	id	tender/id	tender/items/0/id	tender/items/0/additionalClassifications/0/scheme	tender/i
2	ocds-213czf	ocds-213czf	ocds-213czf	0001	CPV	4523316
3						

releases additionalClassifications awa\_ame\_changes awa\_documents awa\_ite\_additionalClassificati

Find Find All Match Case

Sheet 2 of 31 PageStyle\_additionalClassifications Sum=0 100%



**Ben Webb**



**ben.webb@  
opendataservices.coop**

**github.com/  
OpenDataServices/  
flatten-tool/**

# **Bidirectional conversion to/from CSV for nested JSON data**

**Ben Webb**



open data  
services

**github.com/  
OpenDataServices/  
flatten-tool/**



open data  
services



v1.0

**Open Contracting  
Data Standard**

**Why multiple  
formats?**



## Data on the Web Best Practices

### **Best Practice 15: Provide data in multiple formats**

*Data should be available in multiple data formats.*

---

#### **Why**

Providing data in more than one format reduces costs incurred in data transformation. It also minimizes the possibility of introducing errors in the process of transformation. If many users need to transform the data into a specific data format, publishing the data in that format from the beginning saves time and money and prevents errors many times over. Lastly it increases the number of tools and applications that can process the data.

#### **Intended Outcome**

It should be possible for data consumers to work with the data without transforming it.

#### **Possible Approach to Implementation**

Consider the data formats most likely to be needed by intended users, and consider alternatives that are likely to be useful in the future. Data publishers must balance the effort required to make the data available in many formats, but providing at least one alternative will greatly increase the usability of the data.

## Why not just CSV?

- “RFC 4180 proposes a specification for the CSV format, and this is the definition commonly used. However, in popular usage "CSV" is not a single, well-defined format.”
- Encodings differ between developer best practice (UTF-8) and Excel's export (Windows-1252)

# flatten-tool

<https://github.com/OpenDataServices/flatten-tool/>

**XLSX Conf?**



(JSON Pointer:  
RFC6901)

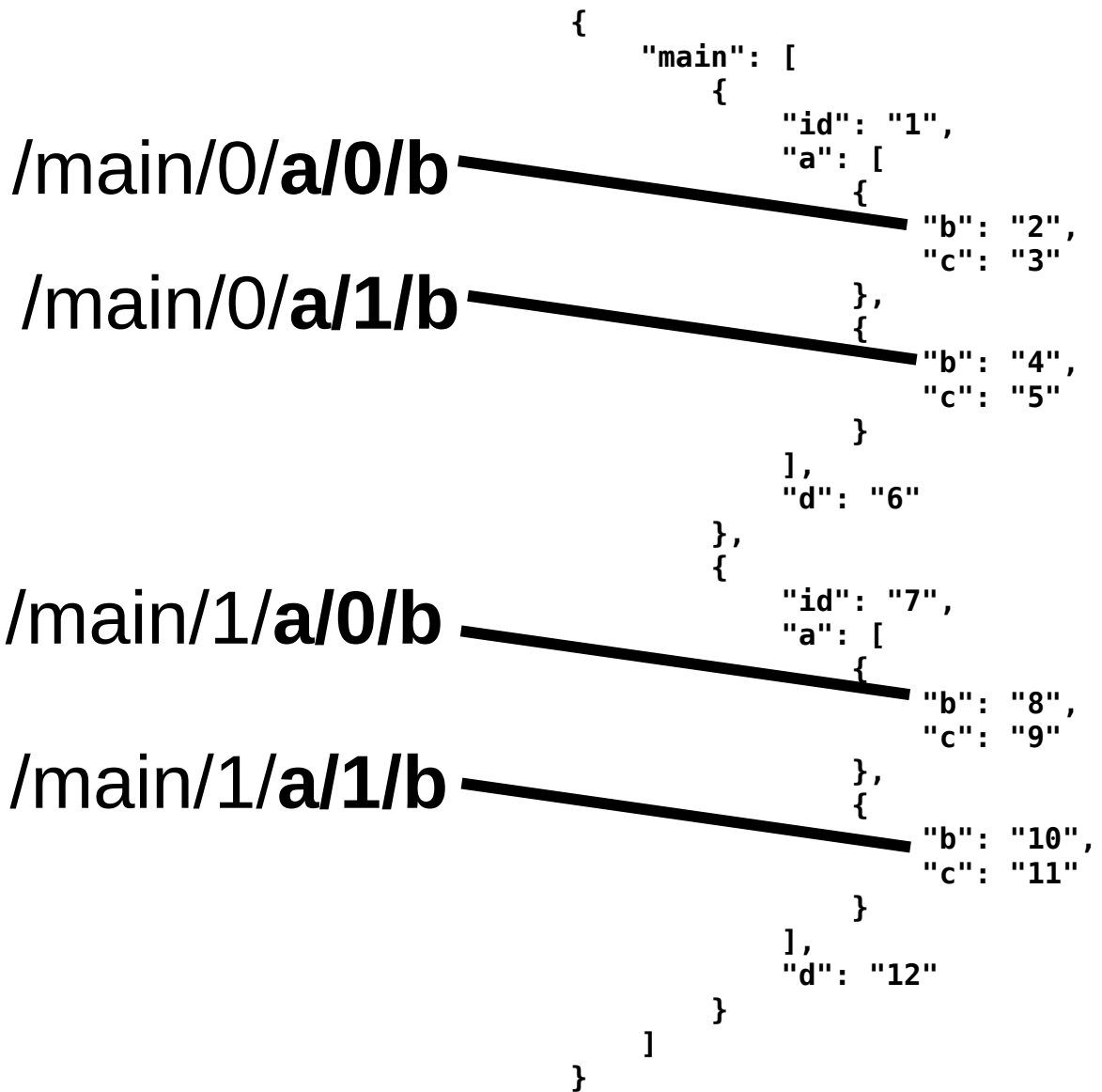
```
    {
      "main": [
        {
          "a": {
            "b": "1",
            "c": "2"
          },
          "d": "3"
        },
        {
          "a": {
            "b": "4",
            "c": "5"
          },
          "d": "6"
        }
      ]
    }
```

**/main/0/a/b**

**/main/1/d**

<b>a/b</b>	<b>a/c</b>	<b>d</b>
1	2	3
4	5	6

```
{  
  "main": [  
    {  
      "a": {  
        "b": "1",  
        "c": "2"  
      },  
      "d": "3"  
    },  
    {  
      "a": {  
        "b": "4",  
        "c": "5"  
      },  
      "d": "6"  
    }  
  ]  
}
```



```

{
  "main": [
    {
      "id": "1",
      "a": [
        {
          "b": "2",
          "c": "3"
        },
        {
          "b": "4",
          "c": "5"
        }
      ],
      "d": "6"
    },
    {
      "id": "7",
      "a": [
        {
          "b": "8",
          "c": "9"
        },
        {
          "b": "10",
          "c": "11"
        }
      ],
      "d": "12"
    }
  ]
}

```

id	a/0/b	a/0/c	a/1/b	a/1/c	d
1	2	3	4	5	6
7	8	9	10	11	12

```

{
  "main": [
    {
      "id": "1",
      "a": [
        {
          "b": "2",
          "c": "3"
        },
        {
          "b": "4",
          "c": "5"
        }
      ]
    },
    {
      "id": "7",
      "a": [
        {
          "b": "8",
          "c": "9"
        },
        {
          "b": "10",
          "c": "11"
        }
      ]
    }
  ],
  "d": "6"
},
{
  "id": "7",
  "a": [
    {
      "b": "8",
      "c": "9"
    },
    {
      "b": "10",
      "c": "11"
    }
  ],
  "d": "12"
}
]
}

```



id	a/0/b	a/0/c	a/1/b	a/1/c	d
1	2	3	4	5	6
7	8	9	10	11	12

id	d
1	6
7	12

id	a/0/b	a/0/c
1	2	3
1	4	5
7	8	9
7	10	11

```

{
  "main": [
    {
      "id": "1",
      "a": [
        {
          "b": "2",
          "c": "3"
        },
        {
          "b": "4",
          "c": "5"
        }
      ],
      "d": "6"
    },
    {
      "id": "7",
      "a": [
        {
          "b": "8",
          "c": "9"
        },
        {
          "b": "10",
          "c": "11"
        }
      ],
      "d": "12"
    }
  ]
}

```

id	d
1	6
7	12

id	a/0/b	a/0/c
1	2	3
1	4	5
7	8	9
7	10	11

```

{
  "main": [
    {
      "id": "1",
      "a": [
        {
          "b": "2",
          "c": "3"
        },
        {
          "b": "4",
          "c": "5"
        }
      ],
      "d": "6"
    },
    {
      "id": "7",
      "a": [
        {
          "b": "8",
          "c": "9"
        },
        {
          "b": "10",
          "c": "11"
        }
      ],
      "d": "12"
    }
  ]
}

```



id	d
1	6
7	12

id	a/0/b	a/0/c
1	2	3
1	4	5
7	8	9
7	10	11

```
{  
  "main": [  
    {  
      "id": "1",  
      "a": [  
        {  
          "b": "2",  
          "c": "3"  
        },  
        {  
          "b": "4",  
          "c": "5"  
        }  
      ],  
      "d": "6"  
    },  
    {  
      "id": "7",  
      "a": [  
        {  
          "b": "8",  
          "c": "9"  
        },  
        {  
          "b": "10",  
          "c": "11"  
        }  
      ],  
      "d": "12"  
    }  
  ]  
}
```



id	a/0/b	a/0/c	d
1	2	3	6
1	4	5	6
7	8	9	12
7	10	11	12

```

{
  "main": [
    {
      "id": "1",
      "a": [
        {
          "b": "2",
          "c": "3"
        },
        {
          "b": "4",
          "c": "5"
        }
      ],
      "d": "6"
    },
    {
      "id": "7",
      "a": [
        {
          "b": "8",
          "c": "9"
        },
        {
          "b": "10",
          "c": "11"
        }
      ],
      "d": "12"
    }
  ]
}

```

id	a/0/id
1	2
1	4
7	8
7	10

id	d
1	6
7	12

id	a/0/id	a/0/b/0/c
1	2	3
1	2	3a
1	4	5
1	4	5a
7	8	9
7	8	9a
7	10	11
7	10	11a

```

{
  "main": [
    {
      "id": "1",
      "d": "6",
      "a": [
        {
          "id": "2",
          "b": [
            {
              "c": "3"
            },
            {
              "c": "3a"
            }
          ]
        },
        {
          "id": "4",
          "b": [
            {
              "c": "5"
            },
            {
              "c": "5a"
            }
          ]
        }
      ]
    }
  ],
  }

```

id	a/0/id
1	2
1	4
7	8
7	10

id	d
1	6
7	12

id	a/0/id	a/0/b/0/c
1	2	3
1	2	3a
1	4	5
1	4	5a
7	8	9
7	8	9a
7	10	11
7	10	11a

```

{
  "id": "7",
  "d": "12",
  "a": [
    {
      "id": "8",
      "b": [
        {
          "c": "9"
        },
        {
          "c": "9a"
        }
      ]
    },
    {
      "id": "10",
      "b": [
        {
          "c": "11"
        },
        {
          "c": "11a"
        }
      ]
    }
  ]
}

```

# Using the JSON Schema

- Types
- Templates
- Titles
- Automagic arrays

```
1 {
2   "id": "http://standard.open-contracting.org/schema/1__0__1/release-schema.json",
3   "$schema": "http://json-schema.org/draft-04/schema#",
4   "title": "Schema for an Open Contracting Release",
5   "type": "object",
6   "properties": {
7     "ocid": {
8       "title": "Open Contracting ID",
9       "description": "A globally unique identifier for this Open Contracting Proc
10      "type": "string",
11      "mergeStrategy": "ocdsOmit"
12    },
13    "id": {
14      "title": "Release ID",
15      "description": "A unique identifier that identifies this release. A release
16      "type": "string",
17      "mergeStrategy": "ocdsOmit"
18    },
19    "date": {
20      "title": "Release Date",
21      "description": "The date this information is released, it may well be the s
22      "type": "string",
23      "format": "date-time",
24      "mergeStrategy": "ocdsOmit"
25    },
26    "tag": {
27      "title": "Release Tag",
```

1	ocid	id	date	tag	initiationType	planning/budget/source	planning/budget/id
---	------	----	------	-----	----------------	------------------------	--------------------

1	ocid	id	tender/id	tender/items/0/id	tender/items/0/description	tender/items/0/classification/scheme	tender/items/0/classification/id
---	------	----	-----------	-------------------	----------------------------	--------------------------------------	----------------------------------

1	ocid	id	tender/id	tender/items/0/id	tender/items/0/additionalClassifications/0/scheme	tender/items/0/additionalClassifications/0/id
---	------	----	-----------	-------------------	---	---

 [awa\\_ame\\_changes.csv](#)

 [awa\\_documents.csv](#)


 [awa\\_ite\\_additionalClassificatio.csv](#)

 [awa\\_items.csv](#)

 [awa\\_sup\\_additionalIdentifiers.csv](#)

 [awa\\_suppliers.csv](#)

 [awards.csv](#)

 [buy\\_additionalIdentifiers.csv](#)

 [con\\_ame\\_changes.csv](#)

 [con\\_documents.csv](#)

# Using the schema

- Types
- Templates
- Titles
- Automagic arrays

```
431 "properties": {
432   "id": {
433     "type": "string",
434     "description": "The unique identifier for this grant. Made up of your",
435     "weight": 0.001,
436     "title": "Identifier"
437   },
438   "title": {
439     "type": "string",
440     "description": "A title for this grant activity. This should be under",
441     "weight": 1.05,
442     "title": "Title"
443   },
444   "description": {
445     "type": "string",
446     "description": "A short description of this grant activity.",
447     "weight": 9,
448     "title": "Description"
449   },
450   "currency": {
451     "$ref": "#/definitions/currency",
452     "description": "The currency used in grant amounts and transactions us",
453   },
454   "amountAppliedFor": {
```



360-giving-schema-titles (2).xlsx - LibreOffice Calc

Calibri 11

A1 f(x) Σ = Identifier

1	Identifier	Title	Description	Currency	Amount App	Amount Aw	Amount Dis	Award
2								

grants actualDates applicationTransaction beneficiaryLocation classifications

Find Find All Match Case

Sheet 1 of 15 PageStyle\_grants Sum=0 100%

360-giving-schema-titles (2).xlsx - LibreOffice Calc

Calibri 11

A1 f(x) Σ = Identifier

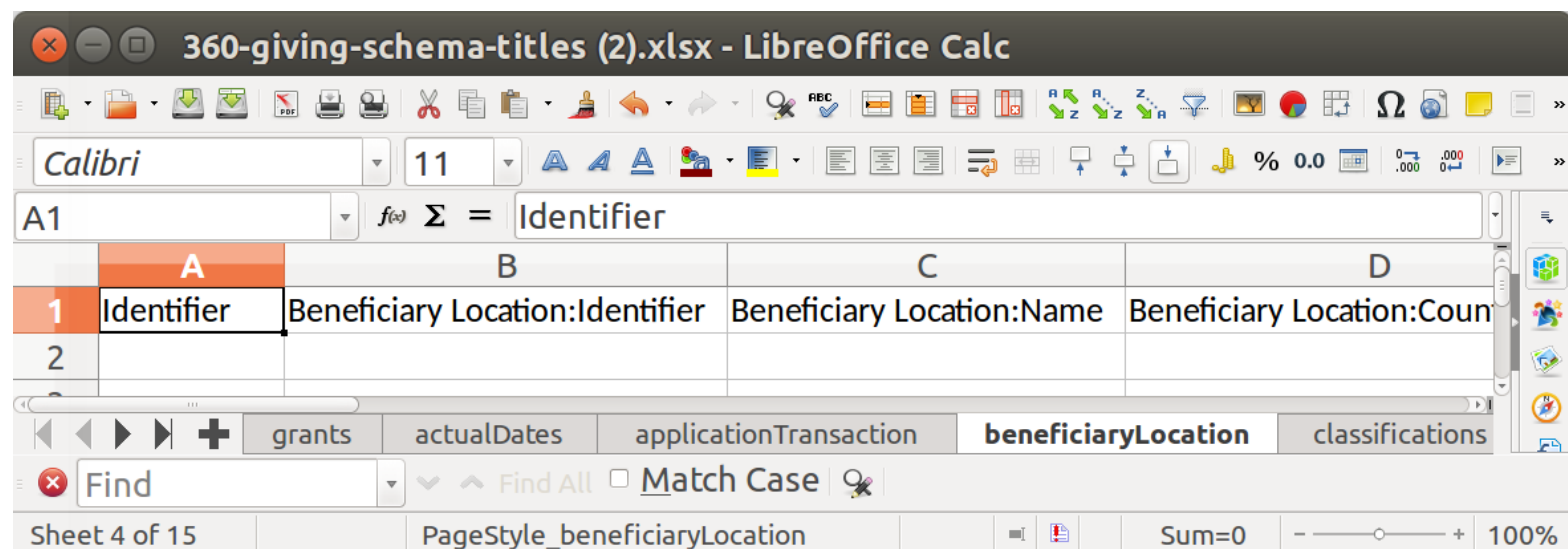
	A	B	C	D
1	Identifier	Beneficiary Location:Identifier	Beneficiary Location:Name	Beneficiary Location:Count
2				

grants actualDates applicationTransaction beneficiaryLocation classifications

Find Find All Match Case

Sheet 4 of 15 PageStyle\_beneficiaryLocation Sum=0 100%

Beneficiary Location:Identifier → beneficiaryLocation/0/id  
Beneficiary Location:Name → beneficiaryLocation/0/name



# What's next?

- Continuing to use to help publishers and users of the 360Giving and OCDS standards
- Continuing to tidy up the code
- Sharing our work with the community.

Possibly:

- Supporting more standards (e.g. XML for IATI)
- Support for more spreadsheet formats (e.g. ODS, data packages)

# What's next?

- Continuing to use to help publishers and users of the 360Giving and OCDS standards
- Continuing to tidy up the code
- Sharing our work with the community.

Possibly:

- Supporting more standards (e.g. XML for IATI)
- Support for more spreadsheet formats (e.g. ODS, data packages)

**Ben Webb**



open data  
services

**ben.webb@  
opendataservices.coop**

**github.com/  
OpenDataServices/  
flatten-tool/**

# CoVE - Convert, Validate, Explore



Open Contracting Partnership

Data **Standard** Validator beta

 Upload

 Link

Supply a URL

Submit

 Paste



Open Contracting Partnership

# Data Standard Validator beta



## Download Files

JSON (Original) 5.5 KB

[Convert to Spreadsheet](#)

## Key Facts

Data downloaded from <https://raw.githubusercontent.com/open-contracting/san>

This package contains 1 release(s) describing 1 contracting process(es).

The publisher named in the file is Open Data Services Co-operative Limited. Tl



Open Contracting Partnership

# Data Standard Validator beta

## Download Files

JSON (Original) 5.5 KB

Excel Spreadsheet (.xlsx) (Converted from Original) 22.8 KB

## Conversion Warnings

Skipping field "tender/amendment/changes/0/former\_value/", because it has no properties.

Skipping field "awards/0/amendment/changes/0/former\_value/", because it has no properties.



flattened (5).xlsx - LibreOffice Calc

Calibri 11

A1 f(x) Σ = ocid

	A	B	C	D	E	F	G	H
1	ocid	id	date	tag	initiationTyp	planning/bu	planning/bu	planning/bu
2	ocds-213czf	ocds-213czf	2010-03-15	tender	tender			
3								

releases additionalClassifications awa\_ame\_changes awa\_documents awa\_ite\_addit

Find Find All Match Case

Sheet 1 of 31 PageStyle\_releases Sum=0 100%

flattened (5).xlsx - LibreOffice Calc

Calibri 11

E2 f(x) Σ = CPV

	A	B	C	D	E	F
1	ocid	id	tender/id	tender/items/0/id	tender/items/0/additionalClassifications/0/scheme	tender/i
2	ocds-213czf	ocds-213czf	ocds-213czf	0001	CPV	4523316
3						

releases additionalClassifications awa\_ame\_changes awa\_documents awa\_ite\_additionalClassificati

Find Find All Match Case

Sheet 2 of 31 PageStyle\_additionalClassifications Sum=0 100%

**Ben Webb**



open data  
services

**ben.webb@  
opendataservices.coop**

**github.com/  
OpenDataServices/  
flatten-tool/**

# **Bidirectional conversion to/from CSV for nested JSON data**

**Ben Webb**



**github.com/  
OpenDataServices/  
flatten-tool/**

Abstract:

A well defined nested format like JSON can be useful for defining a data standard. However, not everyone finds it easy to publish and consume JSON. For the Open Contracting and 360Giving data standards we've taken the hybrid approach of a canonical JSON representation with bidirectional conversion to/from spreadsheets. Since this involves converting between nested and flat representations we've called our software Flatten-Tool: <https://github.com/OpenDataServices/flatten-tool/>



A quick introduction to what I do, so it makes sense when I use these names later in the talk.

I work for a small workers co-op called Open Data Services

<http://opendataservices.coop/>

Part of what we do is provide technical help desk and software tool development for data standards including

The Open Contracting Data Standard (OCDS), which is for public disclosure on all stages of a contracting process:

<http://standard.open-contracting.org/latest/en/>

And 360Giving, which is a UK grants standard.

<http://www.threesixtygiving.org/>

# Why multiple formats?

In order to care about conversion, it means we care about multiple formats in the first place, why is this?



## Data on the Web Best Practices

### **Best Practice 15: Provide data in multiple formats**

*Data should be available in multiple data formats.*

---

#### **Why**

Providing data in more than one format reduces costs incurred in data transformation. It also minimizes the possibility of introducing errors in the process of transformation. If many users need to transform the data into a specific data format, publishing the data in that format from the beginning saves time and money and prevents errors many times over. Lastly it increases the number of tools and applications that can process the data.

#### **Intended Outcome**

It should be possible for data consumers to work with the data without transforming it.

#### **Possible Approach to Implementation**

Consider the data formats most likely to be needed by intended users, and consider alternatives that are likely to be useful in the future. Data publishers must balance the effort required to make the data available in many formats, but providing at least one alternative will greatly increase the usability of the data.

There's a few different reasons for this, but one comes from number #15 of the W3C's best practices for data on the web:

<https://www.w3.org/TR/dwbp/#dataFormats>

Basically, make open data available in a range of formats to meet the needs of different users. Developers may want JSON, researchers might prefer a spreadsheet format.

## Why not just CSV?

- “RFC 4180 proposes a specification for the CSV format, and this is the definition commonly used. However, in popular usage "CSV" is not a single, well-defined format.”
- Encodings differ between developer best practice (UTF-8) and Excel's export (Windows-1252)

Quote from wikipedia:

[https://en.wikipedia.org/wiki/Comma-separated\\_values#Specification](https://en.wikipedia.org/wiki/Comma-separated_values#Specification)

People sometimes claim CSV as both:

- \* A potentially well defined format that developers will like producing / consuming
- \* An easy format for less technical publishers to produce, by export from their existing software.

However these definitions of CSV are not consistent.

One particular problem is that CSV doesn't specify encoding information. Modern developers are often of the opinion “everything UTF-8”, but Excel doesn't play nicely for this!

# flatten-tool

<https://github.com/OpenDataServices/flatten-tool/>

Therefore for OCDS and 360Giving, we use a better defined, and more consistently understood format like JSON as our canonical representation.

We still want to provide spreadsheet users a means to publish and use this data, so we want conversion in both directions between spreadsheets and JSON.

This is what flatten-tool does. It flattens nested JSON into spreadsheet, and vice versa.

flatten-tool is an open source command line tool and Python library.

We also make use of it as a library in the web based validation and data quality tools for 360Giving and OCDS.



# XLSX Conf?

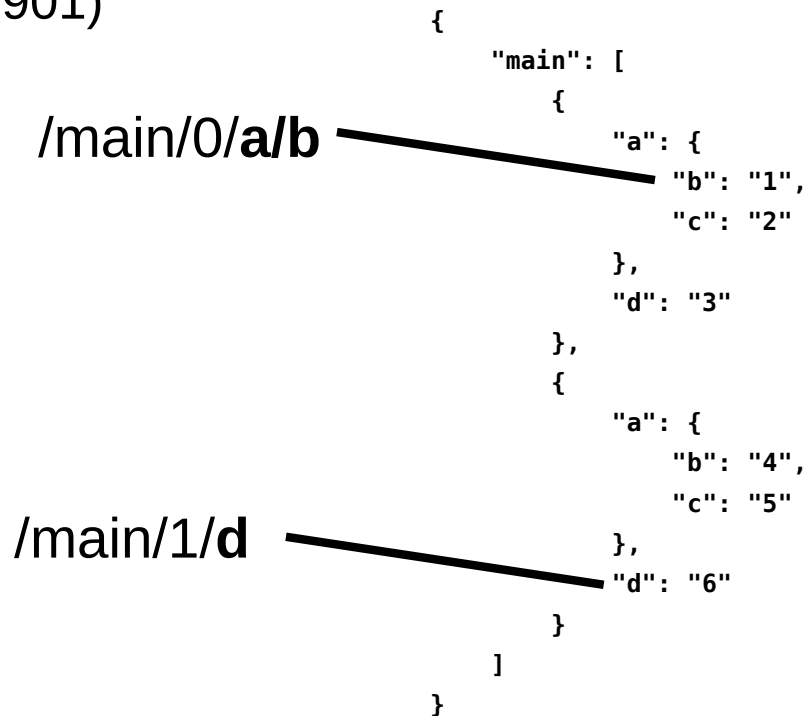
Part of our model with flatten-tool is to isolate dealing with the quirks of spreadsheets produced by less technical users in one piece of software.

Throughout this talk, I'm going to be saying "Spreadsheet", rather than CSV, because flatten-tool supports both CSV and XLSX.

Whilst XLSX has many bad properties as a data interchange format it gives us some extra metadata about what spreadsheet users were trying to do (encodings, number vs string formats etc.).

The flatten-tool model "lets us do this" because it isolates the pain of dealing with XLSX files in one tool, and everything else deals with nice JSON.

(JSON Pointer:  
RFC6901)



How does it work?

We've made some assumptions based on the data we're working with.

We assume we mostly care about a list of many similar JSON objects (with possibly some extra metadata at the top level).

We can identify each value in our JSON using **JSON pointer**, which looks like this...

We then remove the /main/{number} and use the remainder (in bold) as our column headings.

<b>a/b</b>	<b>a/c</b>	<b>d</b>
1	2	3
4	5	6

```

{
  "main": [
    {
      "a": {
        "b": "1",
        "c": "2"
      },
      "d": "3"
    },
    {
      "a": {
        "b": "4",
        "c": "5"
      },
      "d": "6"
    }
  ]
}

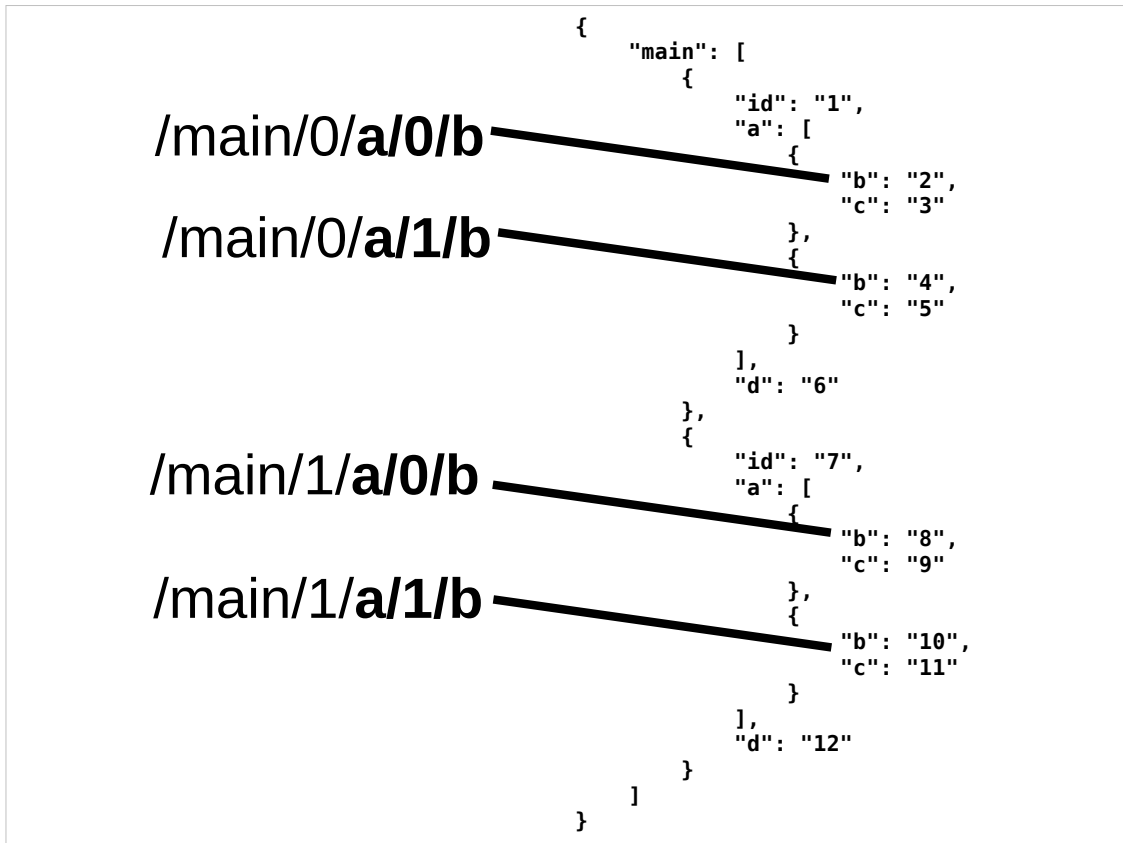
```

Here's what the spreadsheet looks like.

Flatten-tool can convert between these in either direction.

The top level key “main” is specified as a commandline argument.

This is somewhat “nested”, but what's more interesting is how to deal with arrays.



This is what the JSON pointer for items in arrays looks like.

Again the part we'd use for our spreadsheet headings is in bold.

There's a few different "shapes" of spreadsheet that you could use to represent this (so the next few slides will all have this same JSON on the right hand side):

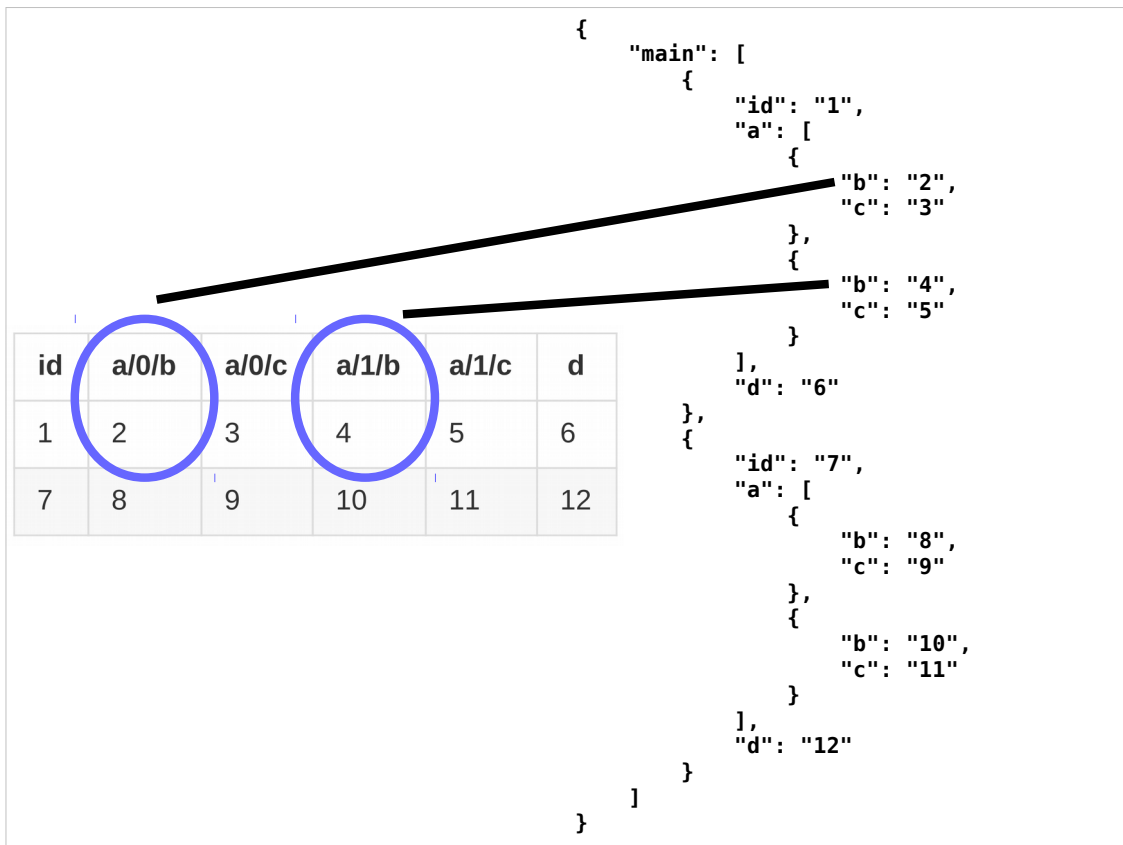
id	a/0/b	a/0/c	a/1/b	a/1/c	d
1	2	3	4	5	6
7	8	9	10	11	12

```

{
  "main": [
    {
      "id": "1",
      "a": [
        {
          "b": "2",
          "c": "3"
        },
        {
          "b": "4",
          "c": "5"
        }
      ],
      "d": "6"
    },
    {
      "id": "7",
      "a": [
        {
          "b": "8",
          "c": "9"
        },
        {
          "b": "10",
          "c": "11"
        }
      ],
      "d": "12"
    }
  ]
}

```

... the simplest of these is to have one column for each JSON pointer.



This means there's one column for entry in the array.

Flatten-tool supports this for unflattening (spreadsheet → JSON) but not the other way round atm.

This is not very useful for large arrays, but can be useful for smaller cases. e.g. useful when the standard has an array for additional Classifications, and people have two additional classifications they want to add, they can easily just double up the classification columns in their spreadsheet).

id	d
1	6
7	12

id	a/0/b	a/0/c
1	2	3
1	4	5
7	8	9
7	10	11

```

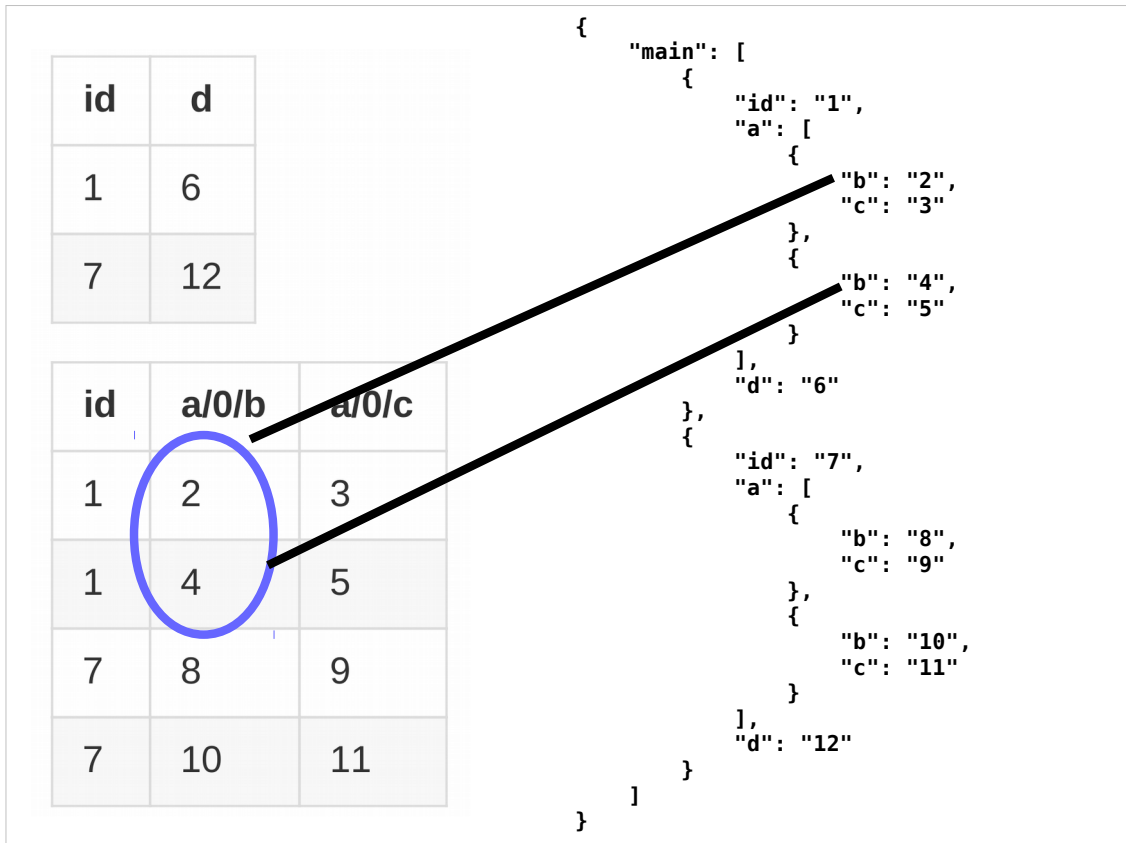
{
  "main": [
    {
      "id": "1",
      "a": [
        {
          "b": "2",
          "c": "3"
        },
        {
          "b": "4",
          "c": "5"
        }
      ],
      "d": "6"
    },
    {
      "id": "7",
      "a": [
        {
          "b": "8",
          "c": "9"
        },
        {
          "b": "10",
          "c": "11"
        }
      ],
      "d": "12"
    }
  ]
}

```

For larger arrays what we really want is some sort of relational structure.

flatten-tool implements this by having multiple sheets (or multiple CSVs).

This is the shape of spreadsheet that we support for conversion in both directions.



So you can see that consecutive array items are in different rows in one column of the related sheet.



id	d
1	6
7	12

id	a/0/b	a/0/c
1	2	3
1	4	5
7	8	9
7	10	11

```

{
  "main": [
    {
      "id": "1",
      "a": [
        {
          "b": "2",
          "c": "3"
        },
        {
          "b": "4",
          "c": "5"
        }
      ],
      "d": "6"
    },
    {
      "id": "7",
      "a": [
        {
          "b": "8",
          "c": "9"
        },
        {
          "b": "10",
          "c": "11"
        }
      ],
      "d": "12"
    }
  ]
}

```

In order to relate the two sheets, we need a common id column for both.

The 360Giving and OCDS standards mandate these ids because they're useful for relational data models like these.

id	a/0/b	a/0/c	d
1	2	3	6
1	4	5	6
7	8	9	12
7	10	11	12

```

{
  "main": [
    {
      "id": "1",
      "a": [
        {
          "b": "2",
          "c": "3"
        },
        {
          "b": "4",
          "c": "5"
        }
      ],
      "d": "6"
    },
    {
      "id": "7",
      "a": [
        {
          "b": "8",
          "c": "9"
        },
        {
          "b": "10",
          "c": "11"
        }
      ],
      "d": "12"
    }
  ]
}

```

We also support conversion **//from//** a third shape which is single sheet format with repeated rows with duplicate values for items not in the array.

((((((I think this format is more confusing than multiple sheets, but some existing tools use it, and it was easy for us to implement.))))))

id	a/0/id
1	2
1	4
7	8
7	10

id	d
1	6
7	12

id	a/0/id	a/0/b/0/c
1	2	3
1	2	3a
1	4	5
1	4	5a
7	8	9
7	8	9a
7	10	11
7	10	11a

```

{
  "main": [
    {
      "id": "1",
      "d": "6",
      "a": [
        {
          "id": "2",
          "b": [
            {
              "c": "3"
            },
            {
              "c": "3a"
            }
          ]
        },
        {
          "id": "4",
          "b": [
            {
              "c": "5"
            },
            {
              "c": "5a"
            }
          ]
        }
      ]
    }
  ]
},

```

Back to multisheet...

We can describe arrays nested within arrays by having all the necessary sheets....

id	a/0/id
1	2
1	4
7	8
7	10

id	d
1	6
7	12

id	a/0/id	a/0/b/0/c
1	2	3
1	2	3a
1	4	5
1	4	5a
7	8	9
7	8	9a
7	10	11
7	10	11a

```

{
  "id": "7",
  "a": [
    {
      "id": "8",
      "b": [
        { "c": "9" },
        { "c": "9a" }
      ]
    },
    {
      "id": "10",
      "b": [
        { "c": "11" },
        { "c": "11a" }
      ]
    }
  ]
}

```

... and id fields for every parent we need to identify.

## Using the JSON Schema

- Types
- Templates
- Titles
- Automagic arrays

So, the basic examples we've just seen demonstrate what flatten-tool can do without any schema.

360Giving and OCDS both have a JSON schema which flatten-tool can use to infer extra information (although the non-schema behaviour is also important because in both people can specify extra fields not in the schema).

Types:

We can use the types from the schema, e.g. to determine whether a number in a CSV should actually be a JSON number or a string.

Templates: Next slide

```

1  {
2  "id": "http://standard.open-contracting.org/schema/1_0_1/release-schema.json",
3  "$schema": "http://json-schema.org/draft-04/schema#",
4  "title": "Schema for an Open Contracting Release",
5  "type": "object",
6  "properties": {
7    "ocid": {
8      "title": "Open Contracting ID",
9      "description": "A globally unique identifier for this Open Contracting Proc
10     "type": "string",
11     "mergeStrategy": "ocdsOmit"
12   },
13   "id": {
14     "title": "Release ID",
15     "description": "A unique identifier that identifies this release. A release
16     "type": "string",
17     "mergeStrategy": "ocdsOmit"
18   },
19   "date": {
20     "title": "Release Date",
21     "description": "The date this information is released, it may well be the s
22     "type": "string",
23     "format": "date-time",
24     "mergeStrategy": "ocdsOmit"
25   },
26   "tag": {
27     "title": "Release Tag",

```

In order to create their spreadsheet data, publishers often want a “template” of the correct column headers to fill in.

flatten-tool can generate these.

So from the OCDS JSON schema, we can produce...

1	ocid	id	date	tag	initiationType	planning/budget/source	planning/budget/id
1	ocid	id	tender/id	tender/items/0/id	tender/items/0/description	tender/items/0/classification/scheme	tender/items/0/classification/id
1	ocid	id	tender/id	tender/items/0/id	tender/items/0/additionalClassifications/0/scheme	tender/items/0/additionalClassifications/0/id	

<a href="#">awa_ame_changes.csv</a>
<a href="#">awa_documents.csv</a>
<a href="#">awa_ite_additionalClassificatio.csv</a>
<a href="#">awa_items.csv</a>
<a href="#">awa_sup_additionalIdentifiers.csv</a>
<a href="#">awa_suppliers.csv</a>
<a href="#">awards.csv</a>
<a href="#">buy_additionalIdentifiers.csv</a>
<a href="#">con_ame_changes.csv</a>
<a href="#">con_documents.csv</a>

Multiple csvs in the multisheet shapes or one multisheet xlsx.

## Using the schema

- Types
- Templates
- Titles
- Automagic arrays

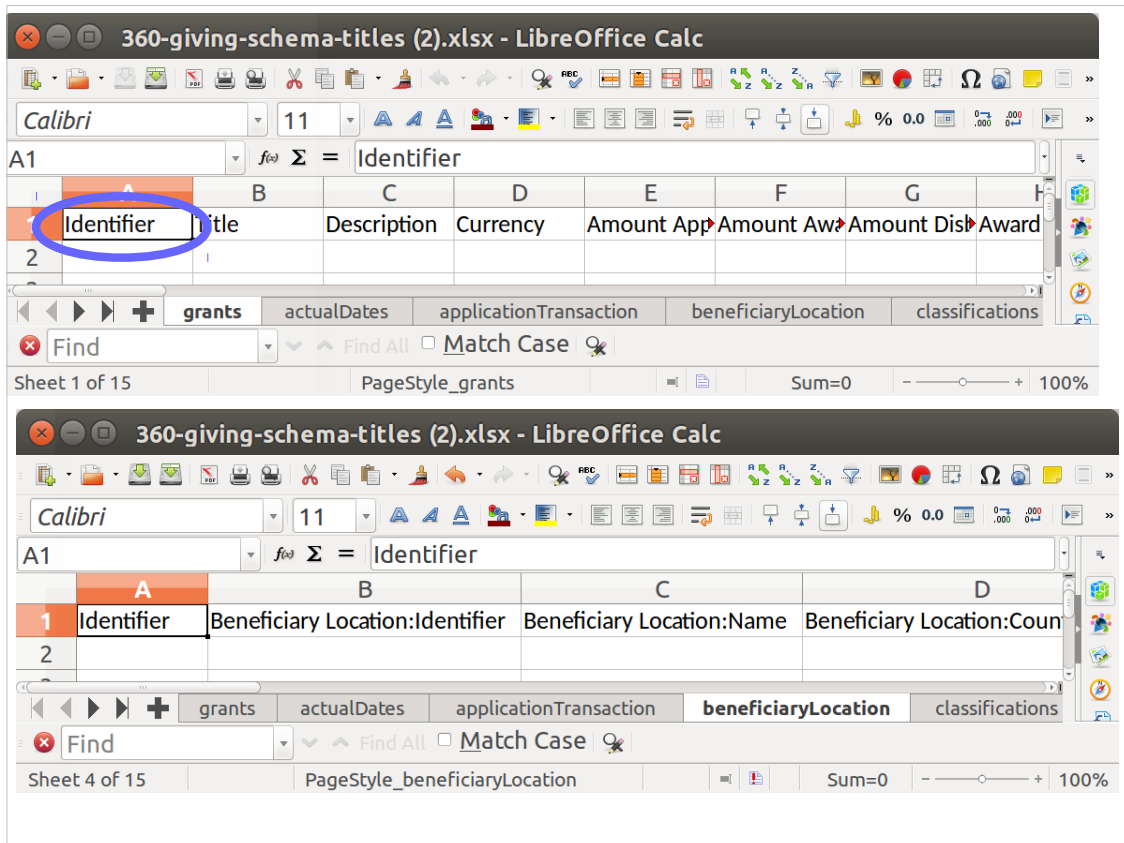
**Titles:**



```
431 "properties": {
432   "id": {
433     "type": "string",
434     "description": "The unique identifier for this grant. Made up of youi
435     "weight": 0.001,
436     "title": "Identifier"
437   },
438   "title": {
439     "type": "string",
440     "description": "A title for this grant activity. This should be unde
441     "weight": 1.05,
442     "title": "Title"
443   },
444   "description": {
445     "type": "string",
446     "description": "A short description of this grant activity.",
447     "weight": 9,
448     "title": "Description"
449   },
450   "currency": {
451     "$ref": "#/definitions/currency",
452     "description": "The currency used in grant amounts and transactions u
453   },
454   "amountAppliedFor": {
```

For 360Giving, where we're trying to get small UK charitable trusts to publish data about their grants, the JSON pointer style headings are a bit too unfriendly.

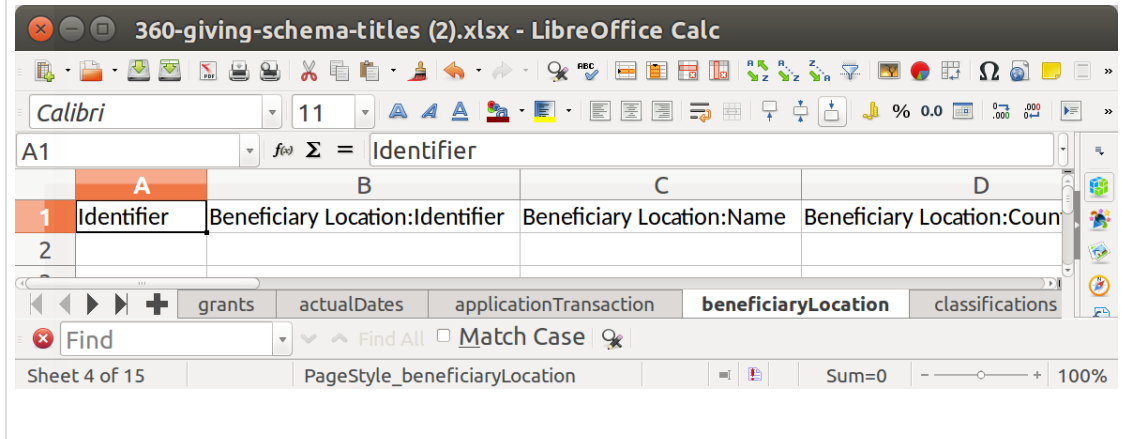
Therefore, we make use of the more human readable titles from the JSON schema.



So we can generate a template with these titles, and flatten-tool also supports these for conversion.

Additionally these titles are case and space insensitive, to make it harder for publishers to “break” the template.

Beneficiary Location:Identifier → beneficiaryLocation/0/id  
Beneficiary Location:Name → beneficiaryLocation/0/name



As part of this titles support, we're using colons instead of slashes, as we think this is more intuitive to many spreadsheet users who would read '/' as either or.

Finally, we drop the numbers used to indicate arrays from the JSON pointers, as it “looks confusing”, and we're able to infer it from the schema instead.

## What's next?

- Continuing to use to help publishers and users of the 360Giving and OCDS standards
- Continuing to tidy up the code
- Sharing our work with the community.

Possibly:

- Supporting more standards (e.g. XML for IATI)
- Support for more spreadsheet formats (e.g. ODS, data packages)

(((Continuing to use to help publishers and users of the 360Giving and OCDS standards)))

Fixing bugs, making it easier to use  
Produce better warnings and error messages

(((Continuing to tidy up the code)))

The projects about a year and a half old now, and we've recently made some changes to many-to-one relationships based problems we found with the previous approach.

These changes are done now, but there's still plenty of refactoring & cleanup to be done as a result.

## What's next?

- Continuing to use to help publishers and users of the 360Giving and OCDS standards
- Continuing to tidy up the code
- Sharing our work with the community.

Possibly:

- Supporting more standards (e.g. XML for IATI)
- Support for more spreadsheet formats (e.g. ODS, data packages)

- (((Sharing our work with the community.))
  - We've been keeping fairly quiet about flatten-tool until we've made those changes. Now that we have, we're interested in trying to share it more widely, as it may be useful to other people (hence this talk).

....Possibly:

- Supporting more standards (e.g. we've discussed the possibility of adding XML support to flatten IATI data)
- Support for more spreadsheet formats (e.g. ODS (open office's spreadsheet format), Open Knowledge's data packages)

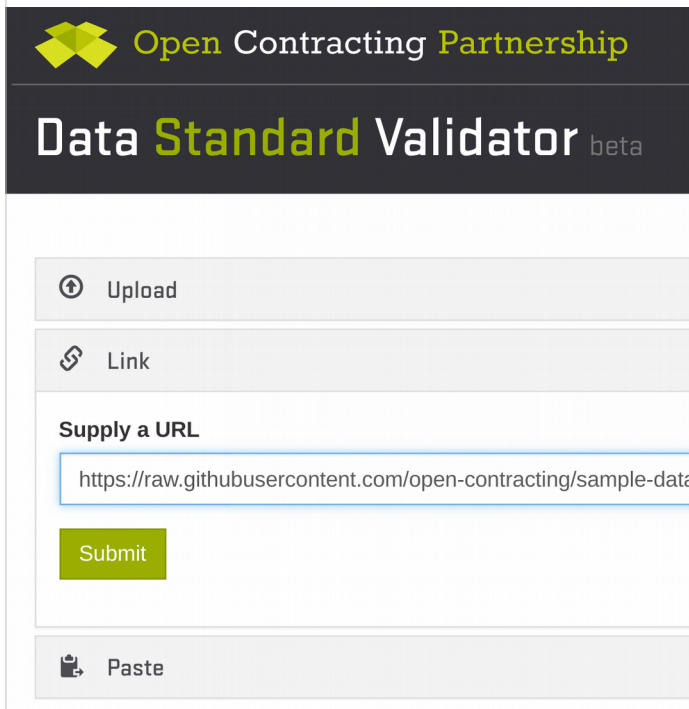
**Ben Webb**



**ben.webb@  
opendataservices.coop**

**github.com/  
OpenDataServices/  
flatten-tool/**

# CoVE - Convert, Validate, Explore

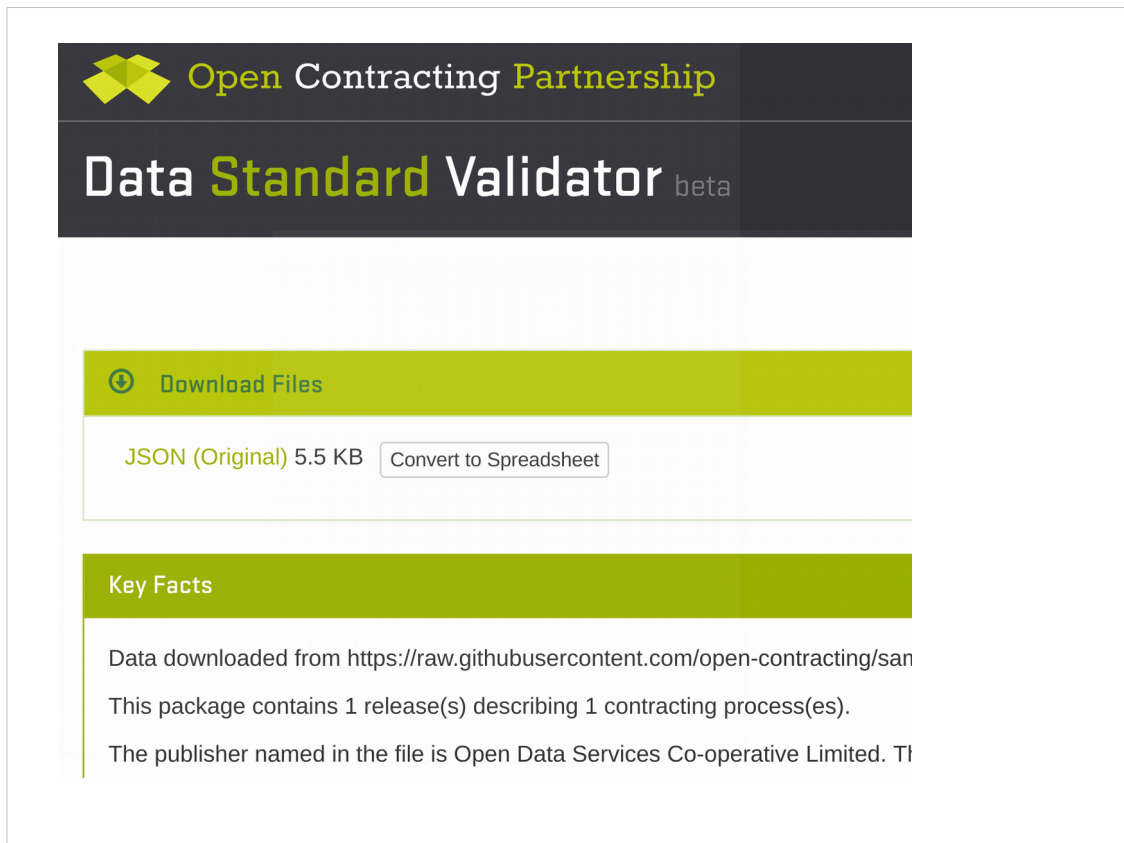


The screenshot shows the CoVE Data Standard Validator interface. At the top, it features the Open Contracting Partnership logo and the text "Data Standard Validator beta". Below this, there are four main sections: "Upload" with a plus icon, "Link" with a chain icon, "Supply a URL" with a text input field containing the URL "https://raw.githubusercontent.com/open-contracting/sample-data" and a green "Submit" button, and "Paste" with a clipboard icon.

Cove - <http://cove.opendataservices.coop/>

Powers the OCDS Validator and the 360Giving Data Quality Tool.

Amongst other features, provides a web interface to flatten-tool for these standards.



You get a button to click to convert.

If you supply a spreadsheet we do the conversion automatically, because Cove's other functionality e.g. validation works on the JSON.





# Data Standard Validator beta

## Download Files

JSON (Original) 5.5 KB

Excel Spreadsheet (.xlsx) (Converted from Original) 22.8 KB

## Conversion Warnings

Skipping field "tender/amendment/changes/0/former\_value/", because it has no properties.

Skipping field "awards/0/amendment/changes/0/former\_value/", because it has no properties.

flattened (5).xlsx - LibreOffice Calc

Calibri 11 % 0.0

A1  $f_{\text{ed}} \Sigma = \text{ocid}$

	A	B	C	D	E	F	G	H
1	ocid	id	date	tag	initiationType	planning/bu	planning/bu	planning/bu
2	ocds-213czf	ocds-213czf	2010-03-15	tender	tender			
3								

releases additionalClassifications awa\_ame\_changes awa\_documents awa\_ite\_addit

Find Find All Match Case

Sheet 1 of 31 PageStyle\_releases Sum=0 100%

flattened (5).xlsx - LibreOffice Calc

Calibri 11 % 0.0

E2  $f_{\text{ed}} \Sigma = \text{CPV}$

	A	B	C	D	E	F
1	ocid	id	tender/id	tender/items/0/id	tender/items/0/additionalClassifications/0/scheme	tender/i
2	ocds-213czf	ocds-213czf	ocds-213czf	0001	CPV	4523316
3						

releases additionalClassifications awa\_ame\_changes awa\_documents awa\_ite\_additionalClassificati

Find Find All Match Case

Sheet 2 of 31 PageStyle\_additionalClassifications Sum=0 100%

**Ben Webb**



**ben.webb@  
opendataservices.coop**

**github.com/  
OpenDataServices/  
flatten-tool/**