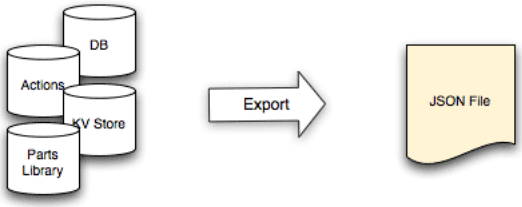
Documentation for Open JSON Format

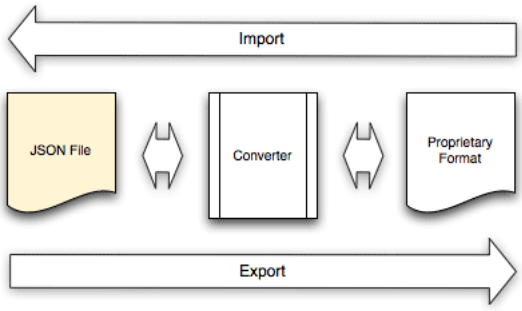
Last Updated: Nov 16, 2012 02:48PM EST

Import/Export Workflow

Inside the Upverter cloud, your designs and components get distributed and stored as a series of actions and tuples. When you export your design, we collate all of the distributed data into a single data file in JSON format.



To use this export in another piece of software, you will need to convert it into that software's proprietary format. Likewise, to get your data out of another piece of software and import it into Upverter, you will also need to run their export through the converter. We are working to make this easier, but we will largely depend on community contribution and the participation of the other vendors.



Index

File Version

Used to identify the source of the export and the file version begin used.

file_version (string)

Contact Us

Post a Public Question
Email Support

24 Phoebe Street
Toronto, Ontario
M5T 2Z3
Canada

A string representing the file format version used.

exporter (string)
The source of the file.

JSON Example

```
                                    "version": {
  "exporter": "Upverter",
  "file_version": "0.1.0"
}
```

This Gist brought to you by GitHub. Version_Snippet.json view raw

---

Component Instances (Array)

Used to store all of the component instances (think reference designators, or symbols) in the schematic. Each instance will reference back to a component in the library (also exported in the components section). An instance has symbol position information, annotations, and attributes.

instance_id (string)
The unique id for this instance.

library_id (string)
The id that corresponds to the component (of which this is an instance) in the components section.

symbol_index (string)
The index of the symbol variant used for this instance.

symbol_attributes (array)
A collection of attributes about each symbol instance in the schematic. The array index corresponds to the body index of the symbol variant. Normally this is an array of size one, but this allows for multi-body symbols.

x (integer)
The top left corner of the symbol, X-coordinate.

y (integer)
The top left corner of the symbol, Y-coordinate.

rotation (integer)
In pi radians, step size of 0.5 (nominally zero).

annotations (annotations)
The associated annotations.

attributes (attributes)
The instance attributes.

JSON Example

```
                                    "component_instances": [
  {
    "attributes": {
      "refdes": "U2"
    },
    "instance_id": "9848c1ff553f2849",
    "library_id": "0877aebec80a5c1a",
    "symbol_attributes": [
      {
        "annotations": [
          {
            "rotation": 0.0,
            "value": "refdes",
            "visible": "true",
            "x": 350,
            "y": 1040
          }
        ],
        "rotation": 0.0,
        "x": 330,
        "y": 1030
      }
    ],
    "symbol_index": 0
  }
]
```

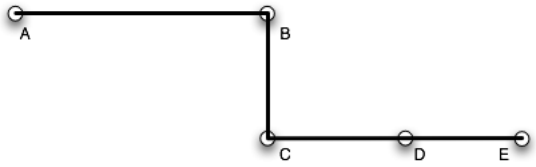
This Gist brought to you by GitHub. Component_Instances_Snippet.json view raw

---

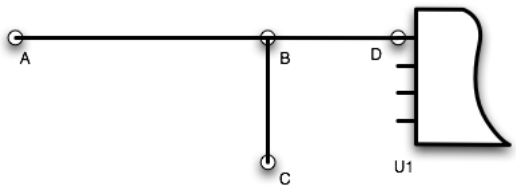
Nets (Array)

Used to store all of the nets in a schematic (also known as wires or connections). Each net has a unique id, a set of points (termination points or corners), attributes (such as a name), and annotations for displaying the name.

Net Point A net point is simply an (x, y) coordinate. In the following they are shown with circles and labeled with letters. A net point connects to any number of other net points and each connection will be drawn with a straight line. This data structure forms a graph that represents all of the interconnections in the design. It is stored directionally, for simplicity of import and export but should be implemented such that a connection in either direction from one point to another point implies a connection in the opposite direction. In this example, point A is connected to B, B is connected to A and C, C to B and D, and so on.



Net points can also be connected to any number of component pins, exactly like other net points, but we store both the component id and the pin identifier. In this example, point A is connected to B; B is connected to A, C and D; and D is connected to B and U1, pin 1.



net_id (string)
The unique net id.

points (array)
A set of point objects in the net. A point is a single (x, y) location.

point_id (string)
The unique point id.

x (integer)
The absolute X-coordinate.

y (integer)
The absolute Y-coordinate.

connected_points (array)
An array of the point ids to which this point is connected.

connected_components (array)
An array of the component instance id and pin number pairs to which this point is also connected.

instance_id (string)
The component instance id.

pin_number (string)
The component pin number.

attributes (attributes)
The associated attributes.

annotations (annotations)
The associated annotations.

JSON Example

```
                                    "nets": [
{
  "annotations": [
    {
      "rotation": 0.0,
      "value": "name",
      "visible": "true",
      "x": 0,
      "y": 0
    }
  ],
  "attributes": {
    "name": "VCC"
  },
  "net_id": "bb545e4b779f527d",
```

```
    "points": [
      {
        "connected_components": [],
        "connected_points": [
          "ec1d007962d58cdf"
        ],
        "point_id": "de8ef2405ffe6a26",
        "x": 990,
        "y": 1000
      },
      {
        "connected_components": [
          {
            "instance_id": "9848c1ff553f2849",
            "pin_number": 3
          }
        ],
        "connected_points": [
          "de8ef2405ffe6a26"
        ],
        "point_id": "ec1d007962d58cdf",
        "x": 990,
        "y": 1020
      }
    ]
  }
]
```


Design Attributes (Object)

Used to store all of the design-level attributes for a schematic, e.g., creator, external links, the license, description, etc. Each attribute should show up exactly once.

annotations (annotations)
The associated annotations.

metadata (object)
The design metadata (essential attributes).

- name (string)
  The design name.
- description (string)
  The design description.
- license (string)
  The design license.
- owner (string)
  The Upverter username of the design owner.
- update_timestamp (string)
  The last update time of the design, as a UNIX epoch timestamp.
- design_id (string)
  The unique design id.
- attached_urls (array)
  An array of associated url strings.

attributes (attributes)
Freeform design-level attributes.

JSON Example


```
                                    "design_attributes": {
  "annotations": [],
  "attributes": {
    "_capacitor_counter": "2",
    "_integrated circuit_counter": "21",
    "_resistor_counter": "3",
    "_unknown_counter": "2"
  },
  "metadata": {
    "attached_urls": [
      "http://code.google.com/p/arduinoboy/"
    ],
    "description": "Based on the original ArduinoBoy (version 1.1.0) schematic by
Trash80 (http://trash80.net). Software available at:
http://code.google.com/p/arduinoboy/\n This version is designed to be built and
operated as a stand-alone device and eliminates extra parts n",
```

```
    "design_id": "0000000000003664",
    "license": "",
    "name": "Simplified ArduinoBoy",
    "owner": "Alex",
    "slug": "Simplified-ArduinoBoy",
    "updated_timestamp": 1317416102
  }
}
```

This Gist brought to you by GitHub. Design_Attributes_Snippet.json view raw

Components (Object)

Used to store components referenced by the component instances in the component instances array. It is stored as an object keyed by the library_id of the component.

Key

library_id (string)
The unique id of the component. This is referenced by the library id of a component instance.

Value Members

name (string)
The name of the component.

attributes (attributes)
The component attributes.

symbols (array)
An array of all possible symbol variant objects. Each symbol variant may have multiple bodies.

bodies (array)
An array of all bodies for a given symbol variant. A single-body symbol variant will have one body; a multi-body symbol variant will have multiple bodies.

pins (array)
An array of all pins attached to the body. Between all bodies, all pins must be assigned.

label (label)
A label shape used to specify and position/display the pin name.

p1 (object)
The location of the start end of the pin.

x (integer)
The X-coordinate referenced from 0,0 for the body.

y (integer)
The Y-coordinate referenced from 0,0 for the body.

p2 (object)
The location of the business end of the pin.

x (integer)
The X-coordinate referenced from 0,0 for the body.

y (integer)
The Y-coordinate referenced from 0,0 for the body.

pin_number (string)
The unique pin identifier. It does not have to be numeric.

shapes (array)
An array of shapes for drawing the non-pin portions of the body.

JSON Example

```
                                    "components": {
  "0877aebec80a5c1a": {
      "attributes": {
        "Current Ouput": "250 mA",
        "Min Operating Temperature": "-40 \u00b0C",
        "Voltage Input": "Up to 20 V",
        "Voltage Output": "5 V",
        "_datasheet": "http://datasheet.octopart.com/L4931CZ50-AP-
STMicroelectronics-datasheet-111644.pdf",
        "_imported_from_octopart": "yes",
        "_lead_free": "Yes",
        "_manufacturer": "STMicroelectronics",
        "_num_pins": "3",
        "_octopart_part_number": "52920711475",
        "_package": "TO-92",
```

```
        "_part_number": "L4931CZ50-AP",
        "_prefix": "U",
        "_rohs": "Yes",
        "_symbol_0_0": "Voltage Regulator",
        "_type": "integrated circuit",
        "max_operating_temperature": "125.00 \u00b0C",
        "mounting_type": "through hole"
      },
      "name": "L4931",
      "symbols": [
        {
          "bodies": [
            {
              "pins": [
                {
                  "p1": {
                    "x": 10,
                    "y": -10
                  },
                  "p2": {
                    "x": 0,
                    "y": -10
                  },
                  "pin_number": "1"
                },
                {
                  "p1": {
                    "x": 30,
                    "y": -20
                  },
                  "p2": {
                    "x": 30,
                    "y": -30
                  },
                  "pin_number": "2"
                },
                {
                  "p1": {
                    "x": 50,
                    "y": -10
                  },
                  "p2": {
                    "x": 60,
                    "y": -10
                  },
                  "pin_number": "3"
                }
              ],
              "shapes": [
                {
                  "height": 20,
                  "type": "rectangle",
                  "width": 40,
                  "x": 10,
                  "y": 0
                }
              ]
            }
          ]
        }
      ]
    }
}
```


Attributes (object)

Attributes can be assigned to most top level objects. These are keyed with the attribute name, and valued accordingly.

Key (string)
The attribute name.

Value (string|integer)
The attribute value.

JSON Example

```
"attributes": {
```

```
  "Current Ouput": "250 mA",
  "Min Operating Temperature": "-40 \u00b0C",
  "Voltage Input": "Up to 20 V",
  "Voltage Output": "5 V",
  "_datasheet": "http://datasheet.octopart.com/L4931CZ50-AP-STMicroelectronics-
datasheet-111644.pdf",
  "_imported_from_octopart": "yes",
  "_lead_free": "Yes",
  "_manufacturer": "STMicroelectronics",
  "_num_pins": "3",
  "_octopart_part_number": "52920711475",
  "_package": "TO-92",
  "_part_number": "L4931CZ50-AP",
  "_prefix": "U",
  "_rohs": "Yes",
  "_symbol_0_0": "Voltage Regulator",
  "_type": "integrated circuit",
  "max_operating_temperature": "125.00 \u00b0C",
  "mounting_type": "through hole"
}
```

This Gist brought to you by GitHub. Attributes_Snippet.json view raw

---

Annotations (array)

Annotations can be assigned to most top level displayed objects including the design itself. They are treated very similar to the label shape.

value (string)
The text to display.

x (integer)
The top left corner X-coordinate.

y (integer)
The top left corner Y-coordinate.

rotation (integer)
In pi radians, step size of 0.5 (nominally zero).

visible (string)
True if the annotation is displayed; false if it is hidden.

JSON Example

```
"annotations": [
  {
    "rotation": 0.0,
    "value": "refdes",
    "visible": "true",
    "x": 350,
    "y": 1040
  }
]
```

This Gist brought to you by GitHub. Annotations_Snippet.json view raw

---

Shape: Rectangle (object)

A rectangle is defined by the location of the upper left corner, a height, and a width.

type (string)
The type of shape, in this case "rectangle".

x (integer)
The upper left X-coordinate.

y (integer)
The upper left Y-coordinate.

width (integer)
Width of rectangle.

height (integer)
Height of rectangle.

JSON Example

```
"shapes": [
  {
    "height": 100,
```

```
    "type": "rectangle",
    "width": 110,
    "x": 5,
    "y": -5
  }
]
```

This Gist brought to you by GitHub. Rectangle_Snippet.json view raw

---

Shape: Rounded Rectangle (object)

A rounded rectangle is defined by the location of the upper left corner, a height, a width, and a radius.

type (string)
The type of shape, in this case "roundedrectangle".

x (integer)
The upper left X-coordinate.

y (integer)
The upper left Y-coordinate.

width (integer)
Width of rectangle.

height (integer)
Height of rectangle.

radius (integer)
Radius of the corners.

JSON Example

```
"shapes": [
  {
    "height": 100,
    "radius": 3,
    "type": "rounded_rectangle",
    "width": 110,
    "x": 5,
    "y": -5,
  }
]
```

This Gist brought to you by GitHub. Rounded_Rectangle_Snippet.json view raw

---

Shape: Arc (object)

Creates an arc.

type (string)
The type of shape, in this case "arc".

x (integer)
The centre X-coordinate.

y (integer)
The centre Y-coordinate.

start angle (integer)
Angle of the arc meeting the start point.

end angle (integer)
Angle of the arc meeting the end point.

radius (integer)
Radius of the arc.

JSON Example

```
"shapes": [
  {
    "start_angle": 0.5,
    "end_angle": 1.5,
    "type": "arc",
    "radius": 10,
    "x": 10,
    "y": -10
  }
]
```

Shape: Circle (object)

Creates a circle

type (string)
The type of shape, in this case "circle".

x (integer)
The centre X-coordinate.

y (integer)
The centre Y-coordinate.

radius (integer)
The radius of the circle.

JSON Example

```
                                    "shapes": [
  {
    "radius": 3,
    "type": "circle",
    "x": 10,
    "y": -27
  }
]
```


Shape: Label (object)

Creates a label

type (string)
The type of shape, in this case "label".

x (integer)
The X-coordinate.

y (integer)
The Y-coordinate.

text (string)
The text value of the label.

align (string)
Possible values include "left", "right", and "center".

rotation (integer)
In pi radians, step size of 0.5.

JSON Example

```
                                    "shapes": [
  {
    "align": "left",
    "rotation": 0.0,
    "text": "1",
    "x": 9,
    "y": -14,
    "type": "label"
  }
]
```


Shape: Line (object)

Creates a line

type (string)
The type of shape, in this case "line".

p1 (object)
The starting point.

x (integer)

The X-coordinate of the starting point.

y (integer)
The Y-coordinate of the starting point.

p2 (object)
The ending point.

x (integer)
The X-coordinate of the ending point.

y (integer)
The Y-coordinate of the ending point.

JSON Example

```
                                    "shapes": [
  {
    "p1": {
      "x": 13,
      "y": -5
    },
    "p2": {
      "x": 18,
      "y": -15
    },
    "type": "line"
  }
]
```

This Gist brought to you by GitHub. Line_Snippet.json view raw

---

Shape: Polygon (object)

Creates a polygon

type (string)
The type of shape, in this case "polygon".

points (array)
A list of coordinates.

x (integer)
The X-coordinate.

y (integer)
The Y-coordinate.

JSON Example

```
                                    "shapes": [
  {
    "points": [
      {
        "x": 13,
        "y": -5
      },
      {
        "x": 45,
        "y": -10
      },
      {
        "x": 45,
        "y": -30
      },
      {
        "x": 13,
        "y": -15
      }
    ],
    "type": "polygon"
  }
]
```

This Gist brought to you by GitHub. Polygon_Snippet.json view raw

---

Shape: Bezier Curve (object)

Creates a bezier curve.

type (string)
The type of shape, in this case "bezier".

control1 (object)
Control point 1.

x (integer)
The X-coordinate.

y (integer)
The Y-coordinate.

control2 (object)
Control point 2.

x (integer)
The X-coordinate.

y (integer)
The Y-coordinate.

point1 (object)
Point 1.

x (integer)
The X-coordinate.

y (integer)
The Y-coordinate.

point2 (object)
Point 2.

x (integer)
The X-coordinate.

y (integer)
The Y-coordinate.

JSON Example

```
                                    "shapes": [
  {
   "control1": {
     "x": 9,
     "y": -10
   },
   "control2": {
     "x": 11,
     "y": -10
   },
   "p1": {
     "x": 3,
     "y": -12
    },
    "p2": {
     "x": 17,
     "y": -12
    },
    "type": "bezier"
  }
]
```

This Gist brought to you by GitHub. Bezier_Snippet.json view raw

Constraints

There are a few contraints that will be enforced on the Upverter import when given a file formatted in this way. They are as follows.

Grid Spacing
We will enforce a grid spacing of 10 pixels. In other words, pins will be shuffled to sit on the intersections of a 10x10 grid. We do this to ease net routing and improve readability.

Pin Count
We expect components to have each of their pins defined. Non-connected pins should be defined and labeled NC.

I found this article helpful
I did not find this article helpful