

JADN Sandbox

User Guide

High Level Table of Contents

JADN Sandbox Capabilities

[Schema Creation](#)

[How to get started...](#)

[Schema Visualization](#)

[Schema Translation](#)

[Data Creation](#)

[Data Validation](#)

[Example Data Generation](#)

[Schema Transformation](#)

[Other Features](#)

Docker Desktop

[How to Get the JADN Sandbox Image](#)

[How to Run the JADN Sandbox Image](#)

[How to Update the JADN Sandbox Image](#)

[How to Stop the JADN Sandbox Image](#)

[How to Learn More about Docker](#)

Low Level Table of Contents

Schema Creation

Feature: Editor Style

To Start

Add Data

Edit Data

Feature: Schema Tree

Feature: Types Outline

Feature: Hierarchy Viewer

Feature: Pattern Visualizer

More Functions and Features

Schema Transformation

Result: A Resolved Schema

Result: Strip Comments

Other Features

Theme Switcher

Add Custom Files

Remove Custom Files

Download Files

Schema Visualization

Results: Selecting One Language

Results: Selecting Multiple Languages

Schema Translation

Data Creation

View Data

Feature: Load/Save Builder

Feature: Magic Wand

Feature: Highlight Data

Data Validation

Invalid Results

Valid Results

Example Data Generation

Results

Schema Creation

Create valid JADN Schemas

JADN Sandbox Home Creation ▾ Visualization Translation ▾ Validation Transformation Generation About

Schema Creation

Home Schema Creation Data Creation

Creation Design and edit JADN schemas using guided forms, with the ability to view the underlying JSON representation. Create valid data instances that follow your schema definitions.

Schema Visualization Convert a JADN Schema into a variety of visual formats such as GraphViz, HTML, JIDL, Markdown, and PlantUML.

Schema Translation Convert JADN schemas to other formats (JSON Schema, XSD) or translate JSON/JIDL schemas into JADN format.

Data Translation Transform JSON data into CBOR or XML formats. View CBOR output as hexadecimal or annotated hex for debugging.

Schemas Data Instances GraphViz HTML JIDL MarkDown PlantUML JSON XSD | CBOR XML

Data Validation Validate data instances in various data language formats against a JADN Schema.

Schema Transformation Convert one or more JADN Schemas into a different but related Schema (resolve references, simplify by removing extensions, strip comments, etc).

Example Data Generation Generate various example data instances of a Schema.

CBOR JSON

localhost:3000/create/schema v0.16.0_1760539237340

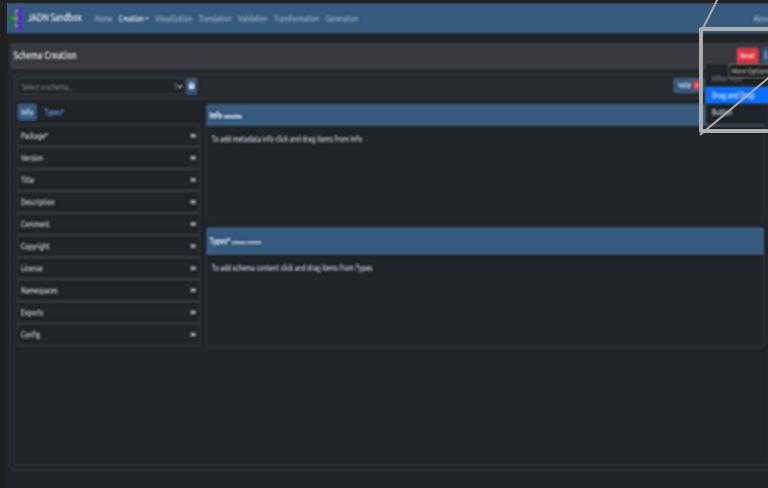
Go to: Creation

When the drop down menu appears, select Schema Creation

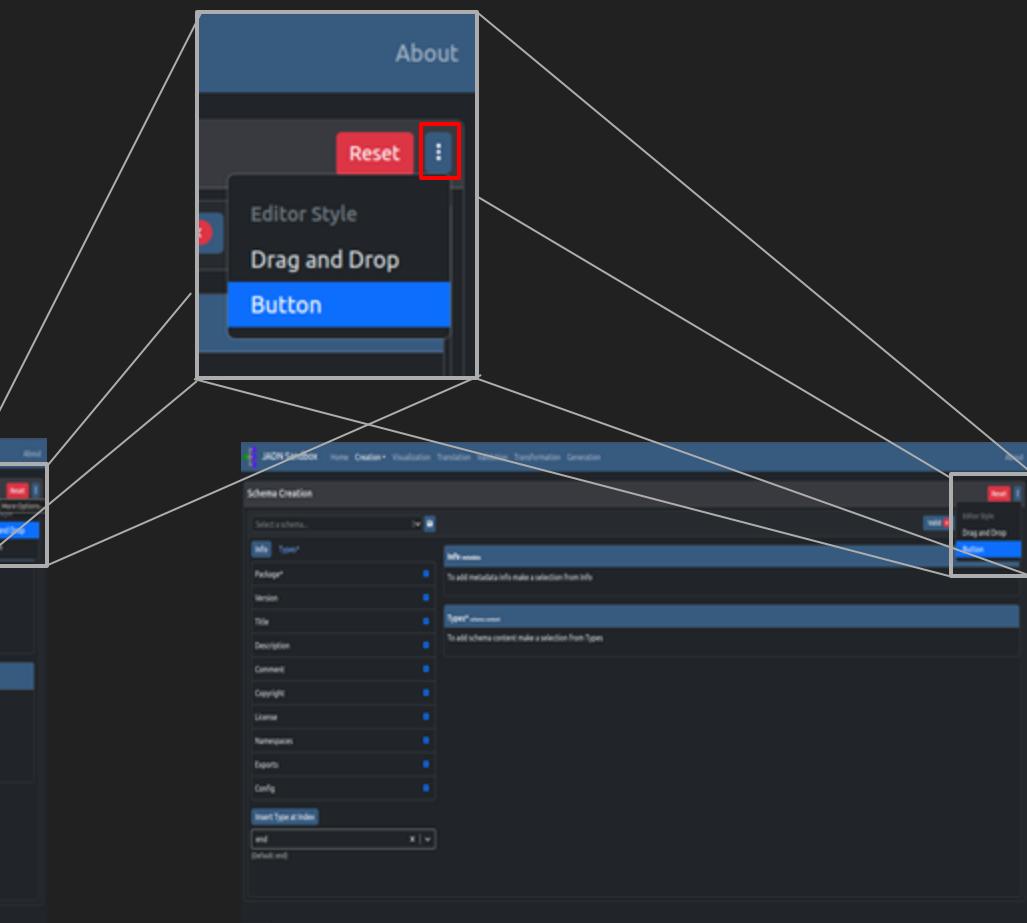
Feature: Editor Style

Choose your schema creation editor style:

- Drag and Drop (default)
- Button



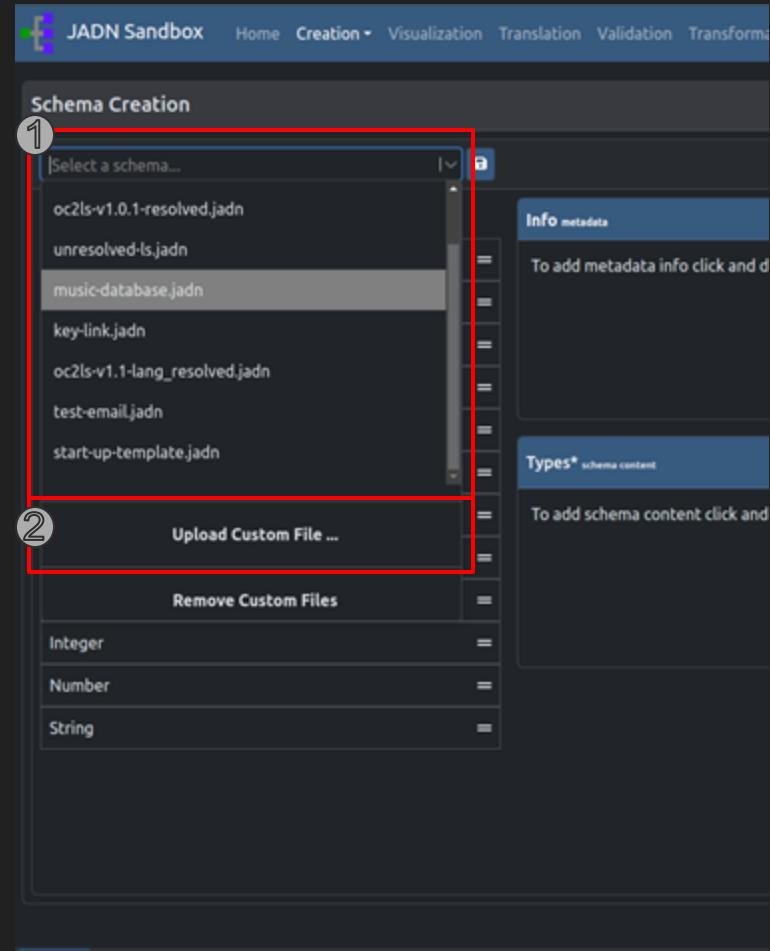
Drag and drop style (default)



Button style

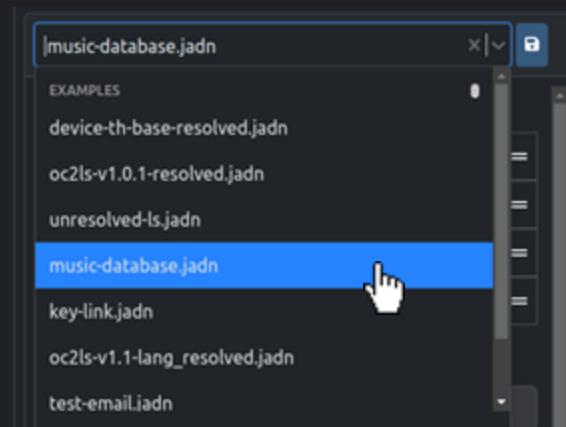
To start...

1. Select an example schema
or
2. Upload a syntactically valid JADN Schema
or
3. From scratch (blank schema)



Option ① to Start: Select a Schema

Selecting a preloaded Schema from the drop down menu will automatically generate the Schema in the Schema Creator.



The screenshot shows the 'Info' tab for the 'music-database.jadn' schema. The 'Info' section contains the following details:

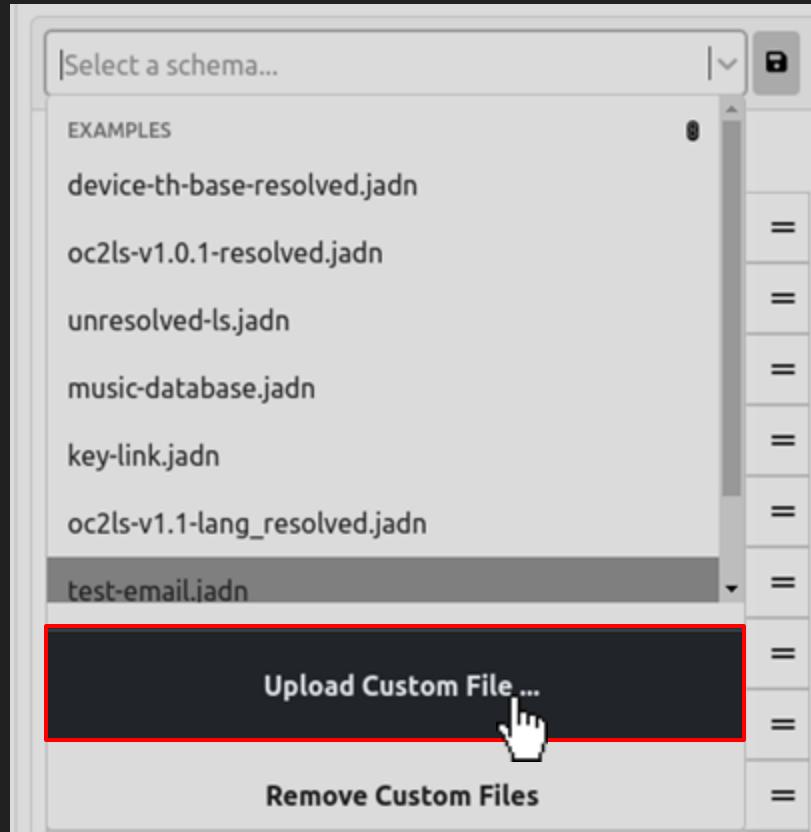
- Package*: <http://fake-audio.org/musicdb>
- Version: 1.0
- Title: Music Library
- Description: This information model defines a library of audio tracks, organized by album.
- License: CC-BY 4.0
- Exports: Two entities exported by this model

```
1: { "title": "Music Library",  
2: "package": "http://fake-audio.org/musicdb",  
3: "version": "1.0",  
4: "description": "This information model defines a library of audio tracks, organized by album.",  
5: "license": "CC-BY 4.0",  
6: "exports": ["Library", "Album", "Track"],  
7: },  
8: { "type": "Library",  
9: "name": "Library",  
10: "description": "A collection of albums.",  
11: "properties": { "Barcode": "string", "A UPC or barcode in 12 digits", 11},  
12: "relationships": { "Artist": "Artist", "An artist associated with this album", 12},  
13: "methods": { "addArtist": "void", "Add an artist associated with this album", 13},  
14: "events": { "trackAdded": "void", "An event fired when a track is added to this album", 14},  
15: "links": { "tracks": "Track", "A list of tracks for this album", 15},  
16: "methods": { "getTracks": "list", "Get all tracks for this album", 16},  
17: "events": { "trackAdded": "void", "An event fired when a track is added", 17},  
18: "links": { "Artist": "Artist", "Interesting information about the performer", 18},  
19: "methods": { "getArtist": "Artist", "Get an artist", 19},  
20: "events": { "Artist": "Artist", "An event fired when an artist is added", 20},  
21: "links": { "Artist": "Artist", "Who is this person?", 21},  
22: "methods": { "getArtist": "Artist", "Get an artist", 22},  
23: "events": { "Artist": "Artist", "An event fired when an artist is removed", 23},  
24: "links": { "Artist": "Artist", "What do they play?", 24},  
25: "methods": { "getArtist": "Artist", "Get an artist", 25}
```

Option ② to Start: Upload a Schema

Click ‘Upload Custom File...’ to upload a file using the computer’s file loader

This will upload any JADN file and then attempt to validate it.

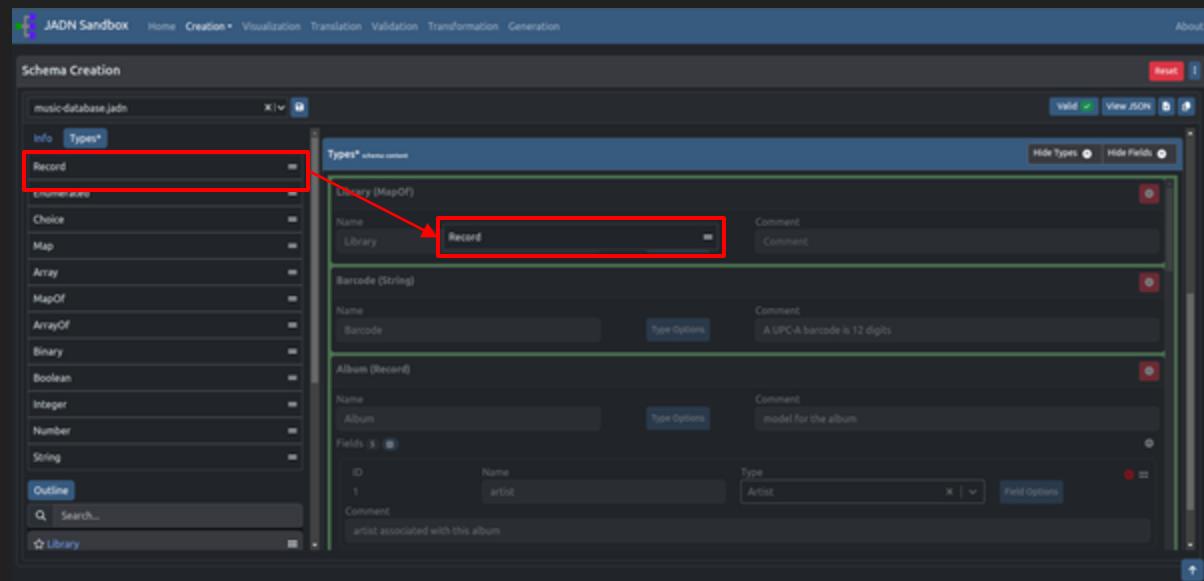


Add Data: Drag and Drop Style

Using the default drag and drop style creator, drag items from left to right.

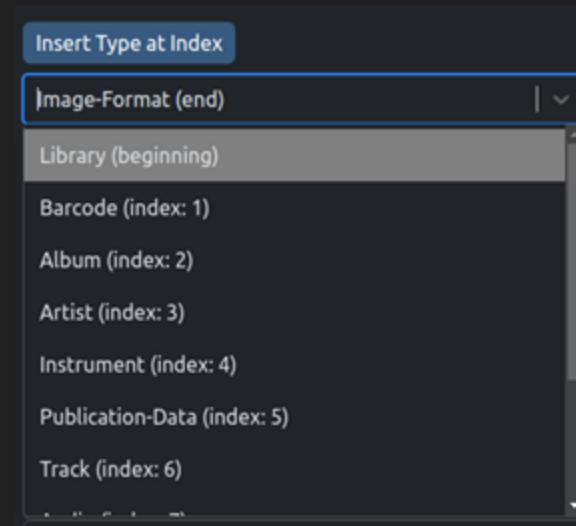
Drop the card when backgrounds are highlighted green.

When dragging, the background will change colors (in this case, to dark gray) to indicate where you can drop the card. It will then change to a green background when you are able to drop the card.

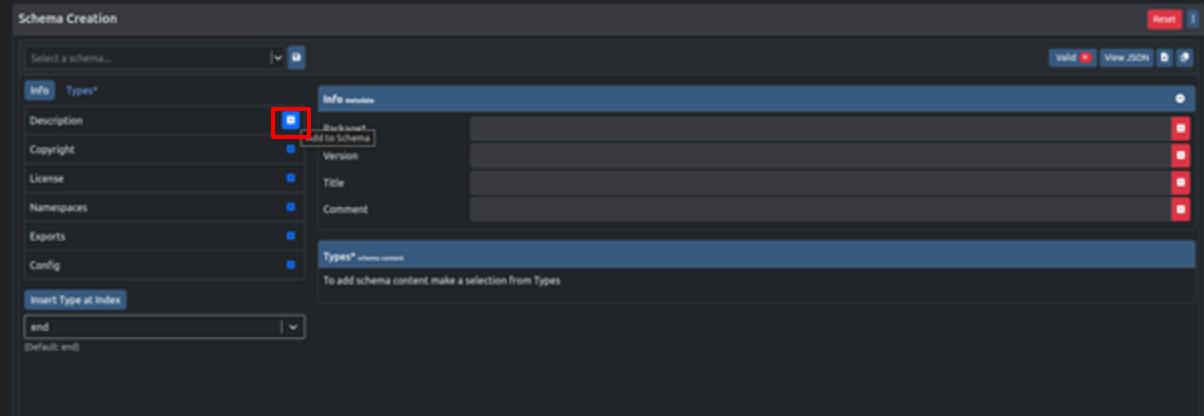


Add Data: Button Style

The 'Insert Type at Index' allows you to select where you would like to add your Type. Types are added to the end of the Schema by default.



Click to add Info or Type to the schema.



Edit Data: Add, change, or remove Types

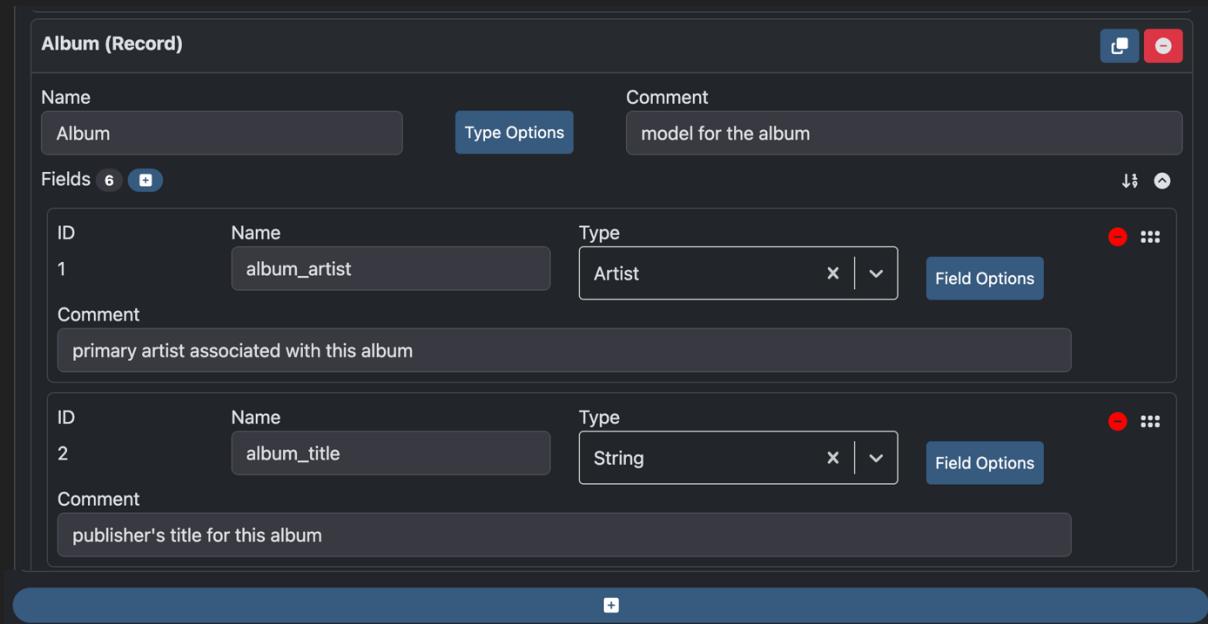
The cards dropped on the right can be edited by clicking within input fields or removed .

Fields, if applicable, can be:

-  - Added,
-  - removed,
-  - hidden,
-  - easily rearranged.

Types can be:

-  - Duplicated



The screenshot shows a data editing interface for an 'Album (Record)' card. The card has a 'Name' field set to 'Album', a 'Comment' field with the value 'model for the album', and a 'Type Options' button. Below the card, there are two fields listed:

ID	Name	Type	Actions
1	album_artist	Artist	  Field Options
2	album_title	String	  Field Options

Each field row includes a 'Comment' section with descriptive text and a 'Field Options' button. A blue '+' button is located at the bottom of the field list.

Feature: Schema Tree

You can easily navigate the entire schema tree, or search for types and/or fields.

The tree is recursive and will show a lot more depth than the Schema Type Outline shown on the next slide.

The current type will be highlighted in blue, while search term(s) found will be highlighted in yellow.

The screenshot shows the 'Schema Creation' interface with a search bar at the top containing 'music-database.jadn'. The search results for 'vocals' are displayed in a tree view on the left:

- > Library
- > Barcode
- > Album
- > Publication-Data (highlighted in blue)
- > Image
- > Image-Format
- < Artist
 - artist_name
- < Instrument
 - vocals (highlighted in yellow)
 - guitar
 - bass
 - drums
 - keyboards
 - percussion
 - brass
 - woodwinds

The right side of the interface shows the 'Publication-Data (Record)' schema definition. It includes a name 'Publication-Data', a comment 'who and when of publication', and two fields:

ID	Name	Type
1	publisher	String
2	release_date	String

Each field has a 'Type Options' button and a 'Field Options' button. A yellow box highlights the 'vocals' entry in the schema tree, and a yellow box highlights the 'vocals' entry in the list of instrument types.

Feature: Schema Types Outline

You can easily search or rearrange the Types within the Outline.



- ★ The cards within the outline can also be starred for future reference.

Clicking on the Type Name will take you to the card in the Types section of the schema on the right.

The screenshot displays two side-by-side views of a software interface for managing schema types. Both views feature a header with 'Tree' and 'Outline' tabs, with 'Outline' being active. Below the tabs is a search bar containing a magnifying glass icon and the placeholder text 'Search...'. The main area contains a list of ten schema types, each represented by a card with a star icon, the type name, and three vertical dots for more options. In the left view, all ten types are listed sequentially. In the right view, the same ten types are shown, but their positions are rearranged: 'Library' is at the top, followed by 'Barcode', 'Album', 'Publication-Data', 'Image', 'Image-Format', 'Artist', 'Instrument', 'Track', and 'Track-Info' at the bottom. Each type card includes a small blue square with a white checkmark icon in the top right corner, indicating it is starred. To the right of each type card are two small blue squares with white icons: an upward arrow and a downward arrow, used for rearranging the list.

Type	Starred	Reorder
Library	Yes	
Barcode	Yes	
Album	Yes	
Publication-Data	Yes	
Image	Yes	
Image-Format	Yes	
Artist	Yes	
Instrument	Yes	
Track	Yes	
Track-Info	Yes	
Audio-Format	Yes	

Feature: Inheritance Hierarchy Viewer

For types that inherit their fields, a hierarchy viewer is available to better visualize that relationship.

Example:

Employee extends Person.

Note: this feature is also present in Data Creation.

Employee (Record)

Name: Employee Comment: An employee instance of person.

Fields: 5

ID	Name	Type
1	employee_id	Integer

Comment: Comment

ID	Name	Type
6	department	String

Comment: Comment

Field Options

Field Options

Type Hierarchy Path

Employee (Current)

Fields: 1: employee_id, 6: department, 7: position, 8: hire_date, 9: salary

Extends ↴

Person (Root)

Fields: 1: id, 2: first_name, 3: last_name, 4: date_of_birth, 5: email

Close

Feature: Pattern Visualizer

For types that contain patterns, the user can visualize the pattern to get a better understanding of its restrictions.

Note: this is also a feature on Data Creation.

The screenshot shows the 'Barcode (String)' configuration screen. A red arrow points from the 'Type Options' button in the main header to a detailed 'Type Options' dialog window below. The dialog has fields for minLength, maxLength, minInclusive, maxInclusive, minExclusive, maxExclusive, format (a dropdown menu), default, const, and pattern (containing '^\\d{12}\$'). A question mark icon is highlighted with a red box. Another red arrow points from the 'pattern' field in the dialog to a visual pattern visualization at the bottom. The visualization consists of three nodes: 'Start of line' (grey circle), 'digit' (green rounded rectangle), and 'End of line' (grey circle). A curved arrow connects the 'digit' node to itself, labeled '11 times'. The 'Save' button in the dialog is highlighted with a red box.

Functions

As you build your schema, you can view the JSON code by clicking on 'View JSON'

[View JSON](#)

This code can be:

- Downloaded to your local storage
- Copied to the clipboard
- Saved onto the file uploader for easy selection

Other features include:

- Checked for validation
- Reset to start over
- Scroll to top (appears when you scroll down on window)

The screenshot shows the JADN Sandbox interface with the 'Schema Creation' tab selected. The main area displays a JSON schema for a 'music-database.jadn' file. The schema defines a 'meta' object with fields like title, package, version, description, license, and roots. It also defines a 'types' object containing various entities such as 'Barcode', 'Record', 'Image', and 'Publication-Data'. The 'Barcode' type is described as a string of 12 digits representing a UPC-A barcode. The 'Record' type is associated with 'Album', 'Artist', and 'Image'. The 'Image' type is associated with 'Image-Format' and 'Image-Content'. The 'Publication-Data' type is associated with 'publisher', 'release_date', and 'record_label'. The 'Image-Format' type is an enumerated list of values: 'PNG', 'JPEG', and 'GIF'. The 'Artist' type is described as a record for a performer. At the bottom right of the schema editor, there are buttons for 'Valid' (green checkmark), 'View Form' (blue button), and other controls.

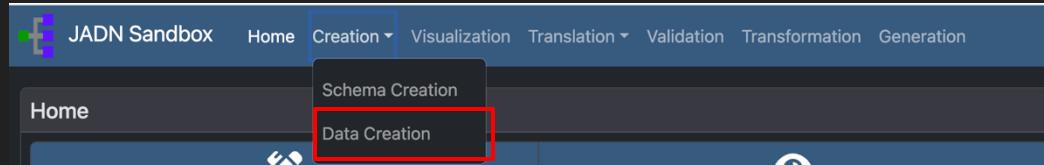
Click 'View Form' [View Form](#)

to return to the Schema Creator.

Data Creation

Create valid data instances from a JADN Schema

How to Navigate to Data Creation



or

The screenshot shows the JADN Sandbox Schema Creation interface. At the top right, there is a 'Data Creation' button, which is highlighted with a red box. The interface includes a sidebar for 'Meta' and 'Types*', and a main area for 'Meta metadata' and 'Types* schema content'.

Meta Types*

- Comment =
- Copyright =
- Namespaces =
- Config =

Tree Outline

Search... Library Barcode Album Publication-Data Image Image-Format Artist Instrument Track

Meta metadata

Package*	http://fake-audio.org/music-lib
Version	1.1
Title	Music Library
Description	This information model defines a library of audio tracks, organized by album, with associated metadata req
License	CC0-1.0

Roots Type definitions exported by this module

- Library

Types* schema content

Library (MapOf)

Name	Comment
------	---------

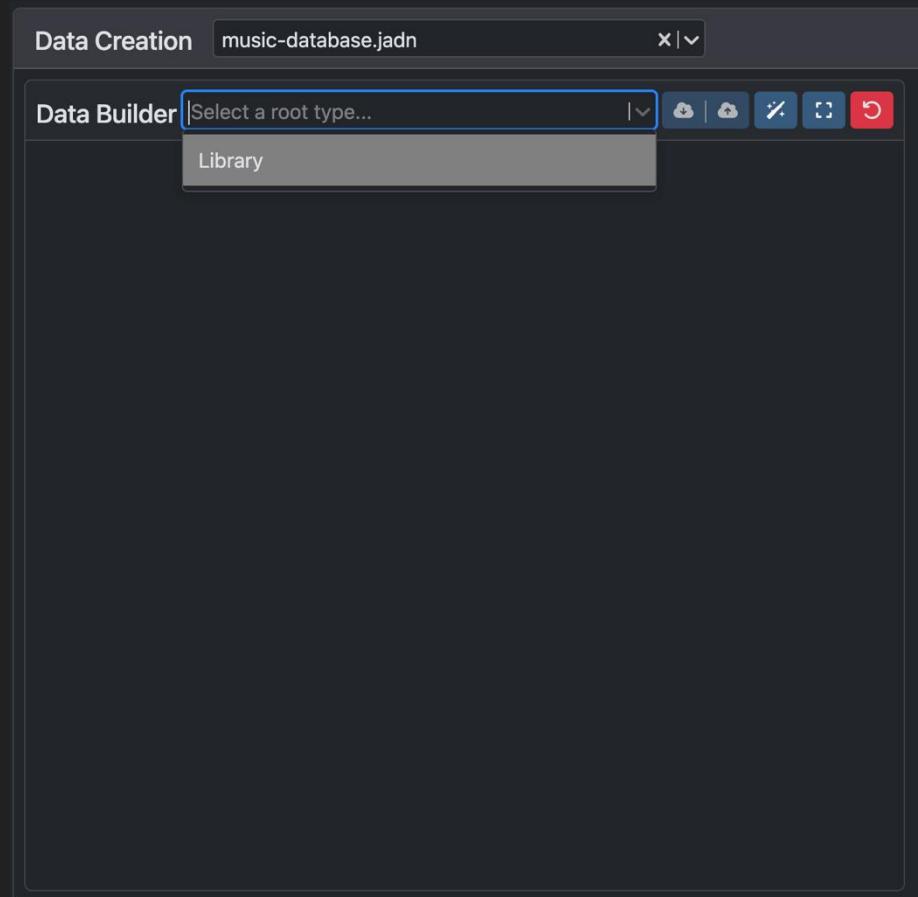
To Start : [click here](#)

Note: Only a JADN Schema can be uploaded.

After selecting a schema, you should be able to select a data type.

Note: This drop down menu is based on the Schema roots.

Once you have selected a data type, a form will appear to help guide you in building data that is valid against the selected schema.



View Data

While filling out the form, you can view the JSON you are creating

Other data formats can also be viewed in real time using the dropdown.

The data can also be:



- Downloaded to your local storage
- Copied to the clipboard

A screenshot of a software interface titled "Data Viewer". It has two tabs: "JSON" and "XML". The "JSON" tab is active, showing the following JSON code:

```
1 v [
2 v   "Library": {
3 v     "id": "123456789112",
4 v     "album_artist": {
5 v       "artist_name": "Artist",
6 v       "instruments": "harmonica"
7 v     },
8 v     "album_title": "Title",
9 v     "pub_data": {
10 v       "publisher": "Publisher",
11 v       "release_date": "2025-10-20"
12 v     },
13 v     "tracks": {
14 v       "location": "Desktop",
15 v       "metadata": {
16 v         "track_number": 1,
17 v         "title": "Title",
18 v         "length": 180,
19 v         "audio_format": "MP3",
20 v         "genre": "jazz"
21 v       }
22 v     },
23 v     "total_tracks": 5
24 v   }
25 v }
```

The "XML" tab is also visible, showing the corresponding XML code:

```
1 <xml version="1.0" >
2 <all>
3   <Library type="dict">
4     <id>123456789112</id>
5     <album_artist type="dict">
6       <artist_name type="str">Artist</artist_name>
7       <instruments type="str">harmonica</instruments>
8     </album_artist>
9     <album_title type="str">Title</album_title>
10    <pub_data type="dict">
11      <publisher type="str">Publisher</publisher>
12      <release_date type="str">2025-10-20</release_date>
13    </pub_data>
14    <tracks type="dict">
15      <location type="str">Desktop</location>
16      <metadata type="dict">
17        <track_number type="int">1</track_number>
18        <title type="str">Title</title>
19        <length type="int">180</length>
20        <audio_format type="str">MP3</audio_format>
21        <genre type="str">jazz</genre>
22      </metadata>
23    </tracks>
24    <total_tracks type="int">5</total_tracks>
25  </Library>
26 </all>
```

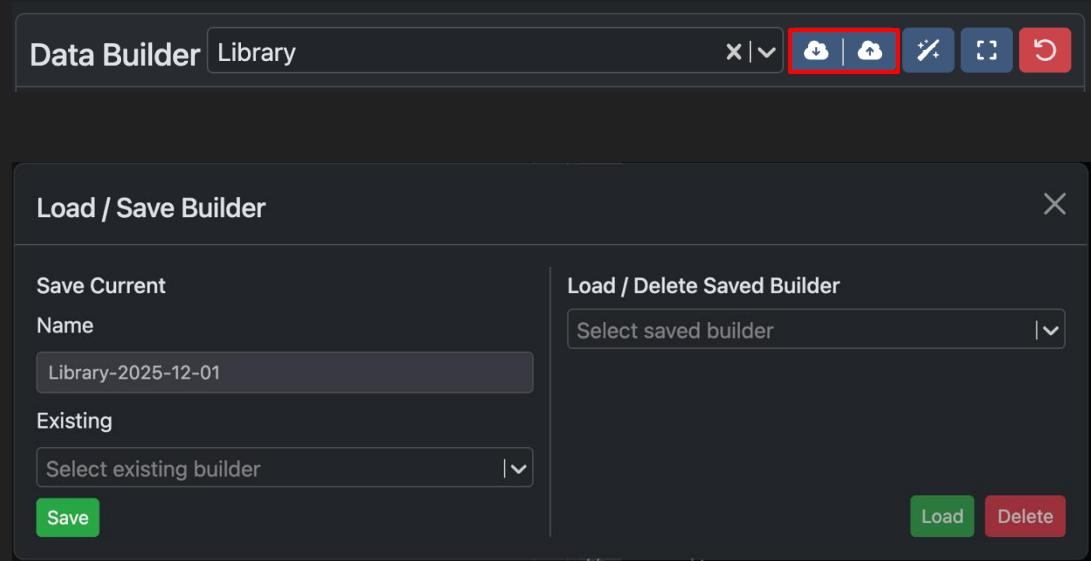


Feature: Load/Save Builder

While working on creating data from a large schema, you may need to take a break and come back later.

Load/Save builder allows you to save your current builder state and restore it at a later point.

Loading a saved state will populate the input fields and recreate the data.



Feature: Magic Wand

While testing out schemas in Data Creation, the individual inputs may not matter. To make testing quicker, the Magic Wand feature, when toggled on, populates all present input fields.

Fields that are hidden will not be populated.

Yellow indicates toggled on, blue indicates off.

Note: the wand takes patterns, options, type, etc into account to ensure valid data.

Data Builder Library

Barcode*

Album*

album_artist*

album_title*

pub_data*

tracks*

total_tracks*

cover_art

Data Builder Library

Barcode* 095318949805

Album* abcdefg

album_artist*

album_title*

pub_data*

tracks*

total_tracks*

cover_art

Feature: Highlight Data

While inputting data, you may want to reference previously completed data.

The highlight data function makes that easy, and works if there are multiple equivalent data.

The screenshot shows the JADN Sandbox application interface. The top navigation bar includes Home, Creation, Visualization, Translation, Validation, Transformation, Generation, and About. Below the navigation is a toolbar with icons for schema creation, reset, and other functions. The main area is divided into two panels: Data Creation and Data Viewer.

Data Creation: This panel is titled "Data Creation" and has a file path "music-database.jadn". It contains a "Data Builder" section with a "Library" dropdown and a "Barcode" field containing "123456789112". Below this are fields for "Album": "album_artist" (highlighted in blue), "album_title" (containing "Title"), "pub_data", "tracks", "total_tracks" (set to 5), and "cover_art".

Data Viewer: This panel is titled "Data Viewer" and displays the JSON representation of the data being created. The JSON code is as follows:

```
1 {  
2   "Library": {  
3     "123456789112": {  
4       "album_artist": {  
5         "artist_name": "Artist",  
6         "Instruments": "harmonica"  
7       },  
8       "album_title": "Title",  
9       "pub_data": {  
10         "publisher": "Publisher",  
11         "release_date": "2025-10-20"  
12       },  
13       "tracks": {  
14         "location": "Desktop",  
15         "metadata": {  
16           "track_number": 1,  
17           "title": "Title",  
18           "length": 180,  
19           "audio_format": "MP3",  
20           "genre": "jazz"  
21         }  
22       },  
23       "total_tracks": 5  
24     }  
25   }  
26 }
```

The bottom right corner of the interface shows the version "v0.16.0_1760539237340".

Schema Visualization

View JADN schema in another language format:
JIDL¹, HTML, Markdown, PlantUML, GraphViz

1. [JADN Interface Definition Language \(IDL\)](#) is a textual representation of JADN type definitions



Home



Creation

Create and edit JADN schemas using forms, view JADN schemas in JSON format.
Create schema compliant data instances (documents, messages).



Schema Visualization

Convert a JADN Schema into different visual representations.



Schema Translation

Translate a JADN Schema to another Schema format.
Translate a JSON Schema to a JADN Schema.

[Schemas](#) [Data Instances](#)[GraphViz](#) [HTML](#) [JDL](#) [MarkDown](#) [PlantUML](#)[JSON](#) [Relax \(XML\)](#) [XSD](#)

Data Validation

Validate data instances in various data language formats against a JADN Schema.



Schema Transformation

Convert one or more JADN Schemas into a different but related Schema (resolve references, simplify by removing extensions, strip comments, etc).



Example Data Generation

Generate various example data instances based off of a Schema.

[CBOR](#) [JSON](#) [Relax \(XML\)](#)

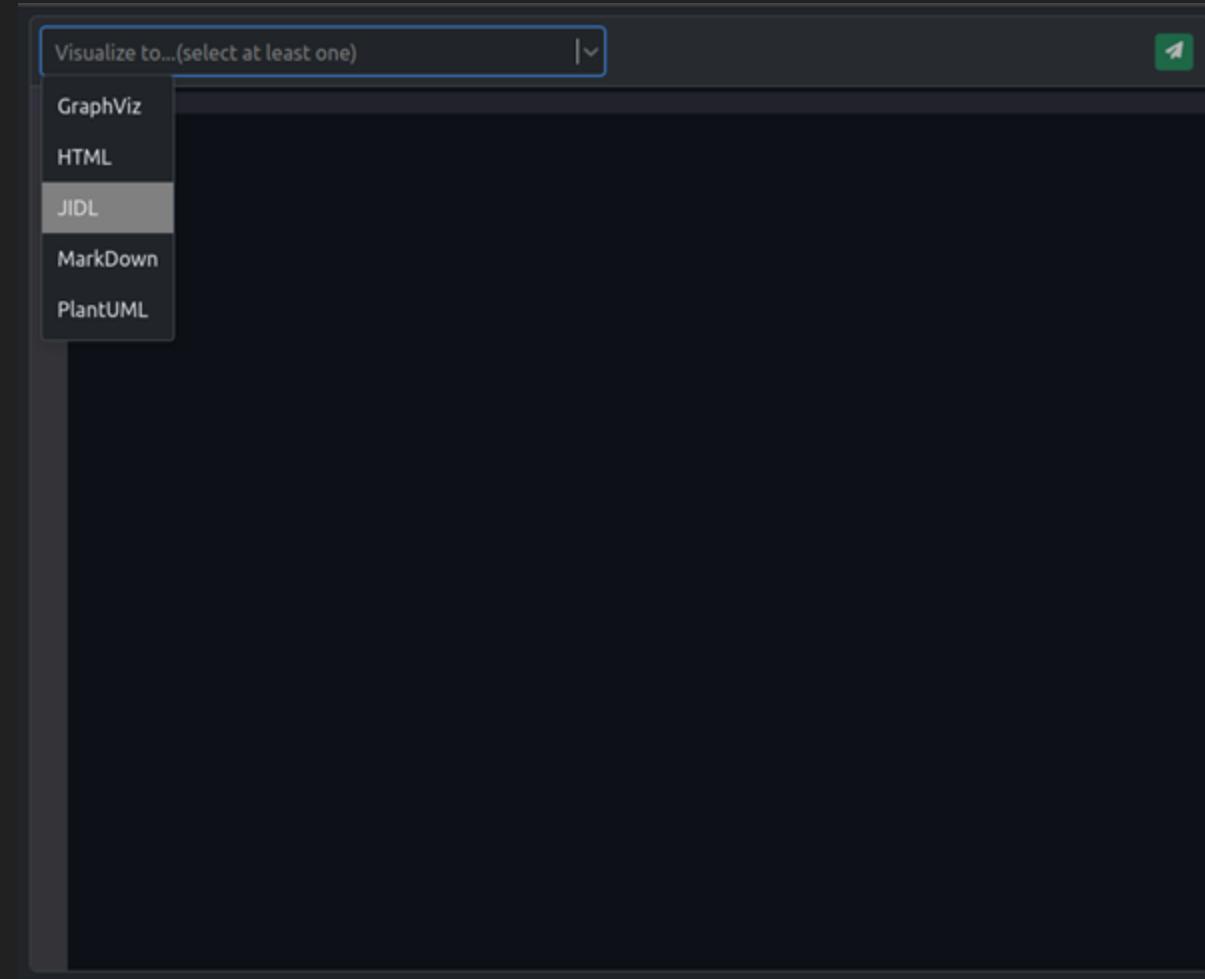
Go to: Visualization

To Start : [click here](#)

Note: Only a JADN Schema can be uploaded.

After selecting a valid schema, select the desired languages you would like to convert the schema to.

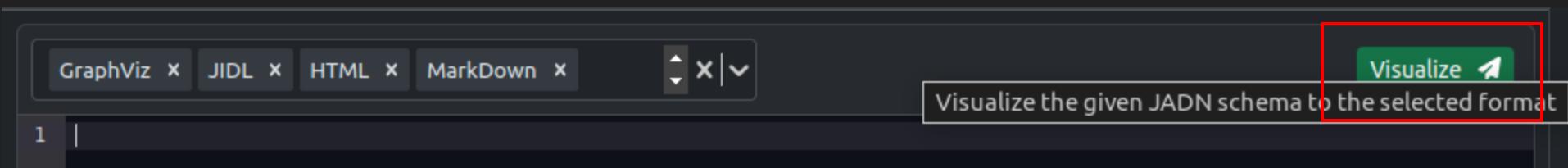
Options on this page are for visual representation of JADN data. For Schema Translations (like to JSON or XML) [click here](#)



Visualize

Once you have selected the languages, the green paper airplane button should light up allowing you to proceed to visualize the schema to the selected messages.

Click the Visualize button to proceed.



Results: Selecting one language

The result will show you the code in the selected language.

The following features allow you to interact with the visualization as you need.



- Split view (currently shown) : To view the code and visualization simultaneously



- Pop out : See the visualization in a new tab window



- Download visualization as PDF



- Download visualization as an image



- Download code



- Copy code to Clipboard

The screenshot shows a software interface with a dark theme. On the left, there is a vertical toolbar with several icons: split view, pop out, download PDF, download image, download code, and copy code to clipboard. The main area has two tabs: "HTML" (selected) and "Schema". The "HTML" tab displays the following code:

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8">
5     <title>Music Library</title>
6     <style type="text/css"/> /* Theme Colors - CC3322 as base using Adobe Kuler */
7   /* PDF Styles */
8   @page {
9     size: letter portrait;
10    @frame content_frame {
11      top: 16pt;
12      left: 16pt;
13    }
14  }
15  @media print {
16    #schema {
17      max-width: auto;
18    }
19  }
20  /* Standard Styles */
21  dim:hide;
```

The "Schema" tab displays the following JSON schema:

```
package: "http://fake-audio.org/music-lib"
version: "1.0"
title: "Music Library"
description: "This information model defines a library of audio tracks, organized by album"
license: "CC0-1.0"
exports: ["Library", "Album", "Track"]
config: {}
```

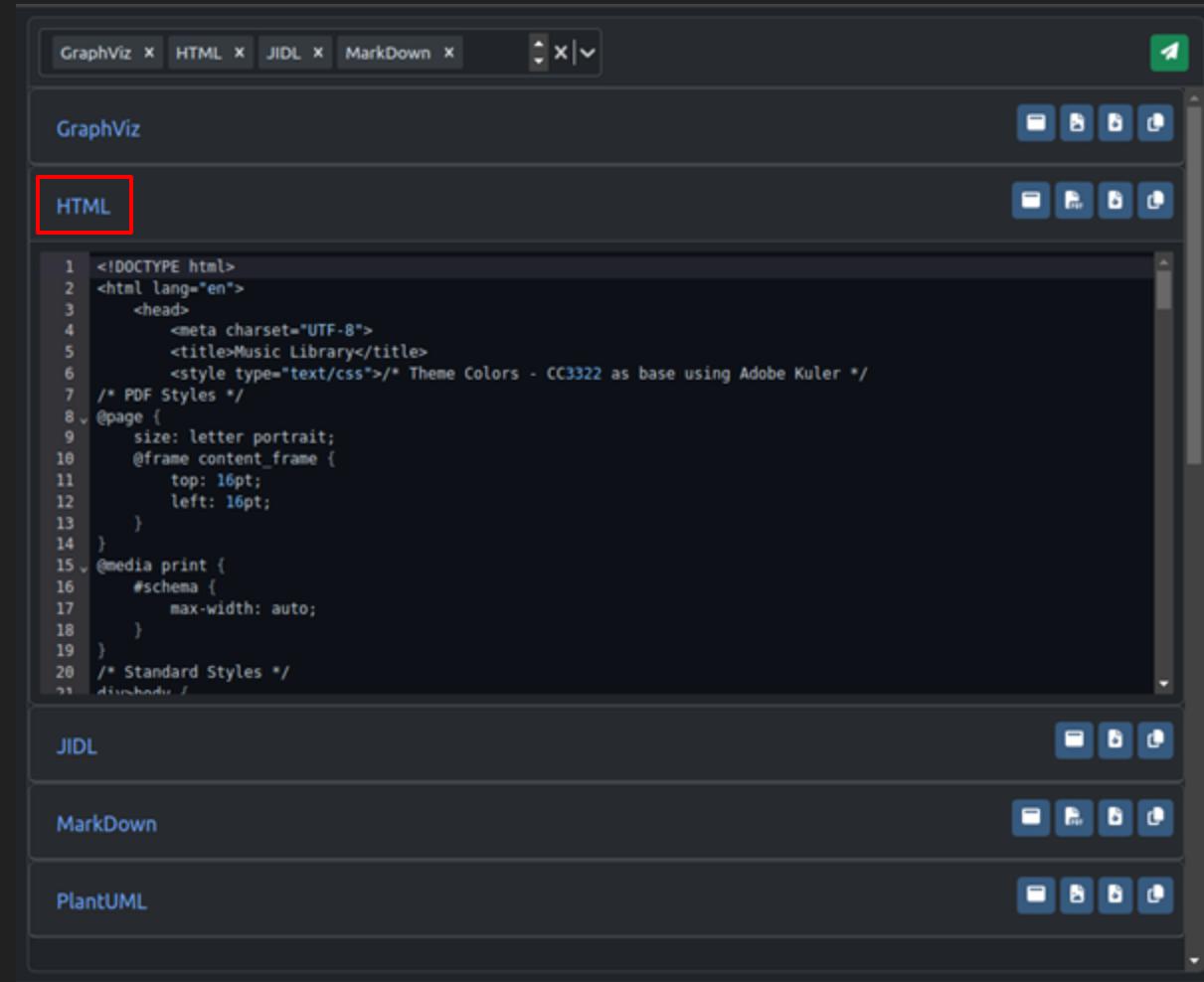
Below the schema, there are sections for "Compound Types", "Library", and "Album".

Results: Selecting multiple languages

You also have the ability to visualize multiple languages at once.

You can view the code for one or multiple languages at once by clicking on the header of each section.

Each language has its own applicable features as seen in each section.



The screenshot shows a code editor interface with four tabs at the top: GraphViz, HTML, JIDL, and MarkDown. Below the tabs, there are four sections labeled GraphViz, HTML, JIDL, and MarkDown, each containing code snippets. The 'HTML' section is currently selected, as indicated by a red rectangular box around its tab and the active code preview. The code in the HTML section is a CSS stylesheet for a 'Music Library' page, defining styles for the document, page, and media print. The other sections (GraphViz, JIDL, and MarkDown) are also visible but inactive.

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8">
5     <title>Music Library</title>
6     <style type="text/css"> /* Theme Colors - CC3322 as base using Adobe Kuler */
7   /* PDF Styles */
8   @page {
9     size: letter portrait;
10    @frame content_frame {
11      top: 16pt;
12      left: 16pt;
13    }
14  }
15  @media print {
16    #schema {
17      max-width: auto;
18    }
19  }
20  /* Standard Styles */
21  diashade /
```

Schema Translation

Convert JADN Schema to JSON , Relax XML, or XSD.
Convert JSON Schema to JADN.



Home



Creation

Create and edit JADN schemas using forms, view JADN schemas in JSON format.
Create schema compliant data instances (documents, messages).



Schema Visualization

Convert a JADN Schema into different visual representations.



Schema Translation

Translate a JADN Schema to another Schema format.
Translate a JSON Schema to a JADN Schema.

Schemas Data Instances

GraphViz HTML JDL MarkDown PlantUML

JSON Relax (XML) XSD



Data Validation

Validate data instances in various data language formats against a JADN Schema.



Schema Transformation

Convert one or more JADN Schemas into a different but related Schema (resolve references, simplify by removing extensions, strip comments, etc).



Example Data Generation

Generate various example data instances based off of a Schema.

CBOR JSON Relax (XML)

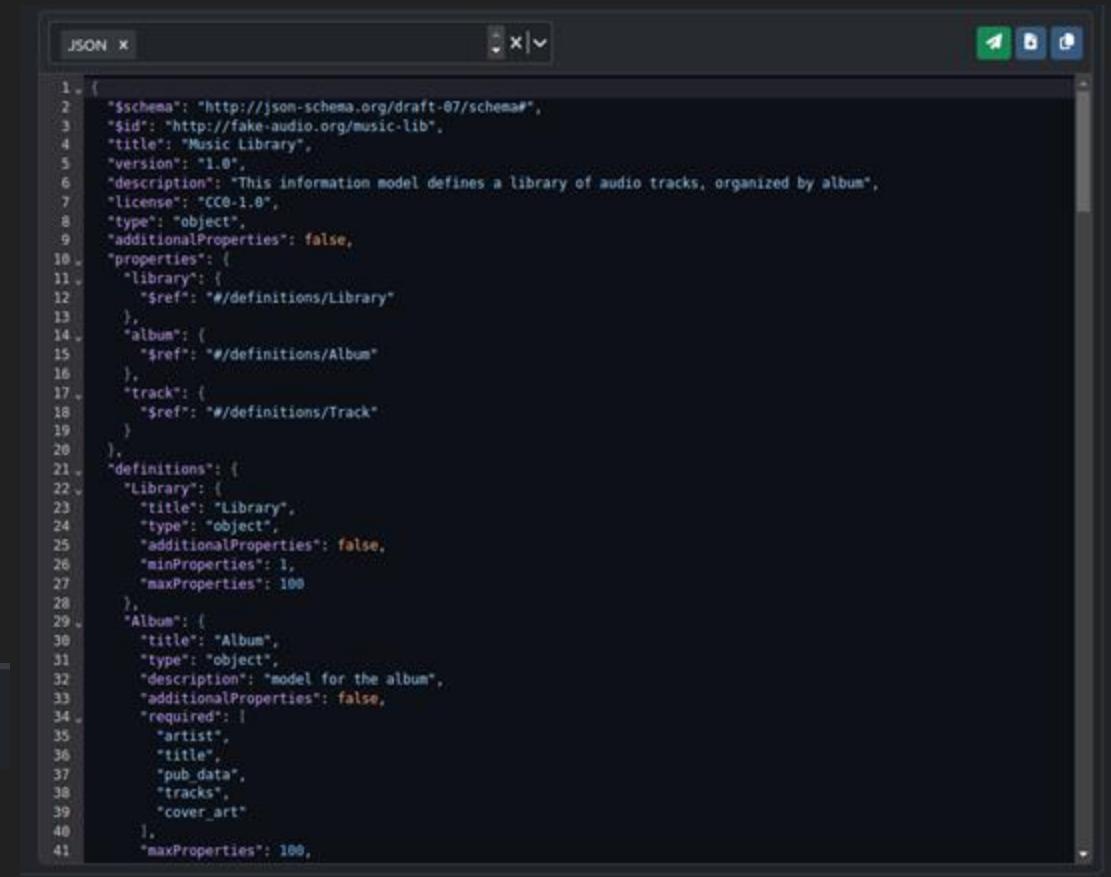
Go to: Translation

To Start : [click here](#)

Note: A JSON or JADN Schema can be uploaded.

Similar to Schema Visualization, Schema Translation allows you to translate your schema into your desired language.

When selecting a schema, make sure to select the type of schema you want to translate from: JADN or JSON.



The screenshot shows a software interface for schema translation. On the left, there is a code editor window titled "music-database.jadn" containing JADN schema code. On the right, there is another code editor window titled "JSON X" containing JSON schema code. Between the two windows are several tabs and buttons: "jadn" (highlighted with a red box), "Valid", and "X|v". The JSON schema in the "JSON X" window is as follows:

```
1. {
2.   "$schema": "http://json-schema.org/draft-07/schema#",
3.   "$id": "http://fake-audio.org/music-lib",
4.   "title": "Music Library",
5.   "version": "1.0",
6.   "description": "This information model defines a library of audio tracks, organized by album",
7.   "license": "CC0-1.0",
8.   "type": "object",
9.   "additionalProperties": false,
10.  "properties": {
11.    "library": {
12.      "$ref": "#/definitions/Library"
13.    },
14.    "album": {
15.      "$ref": "#/definitions/Album"
16.    },
17.    "track": {
18.      "$ref": "#/definitions/Track"
19.    }
20.  },
21.  "definitions": {
22.    "Library": {
23.      "title": "Library",
24.      "type": "object",
25.      "additionalProperties": false,
26.      "minProperties": 1,
27.      "maxProperties": 100
28.    },
29.    "Album": {
30.      "title": "Album",
31.      "type": "object",
32.      "description": "model for the album",
33.      "additionalProperties": false,
34.      "required": [
35.        "artist",
36.        "title",
37.        "pub_data",
38.        "tracks",
39.        "cover_art"
40.      ],
41.      "maxProperties": 100
42.    }
43.  }
44. }
```

Result: Translated Schema to one language

Result: Translated Schema to multiple languages

You can translate from JADN to JSON and XML (XSD or Relax) or from JSON to JADN.

Schema Translation

music-database.jadn

```
1 v {
2 v   "meta": {
3 v     "title": "Music Library",
4 v     "package": "http://fake-audio.org/music-lib",
5 v     "version": "1.1",
6 v     "description": "This information model defines a library of audio tracks, organized by
7 v     license: \"CC0-1.0\",
8 v     roots: [\"Library\"]
9 v   },
10 v   "types": [
11 v     ["Library", "MapOf", ["+Barcode", "*Album", "[1]", "Top level of the library is a map
12 v       of barcode", "String", "[%\\d{12}"]", "A UPC-A barcode is 12 digits", ],
13 v     ["Album", "Record", [], "model for the album", [
14 v       [1, "album_artist", "Artist", [], "primary artist associated with this album"],
15 v       [2, "album_title", "String", [], "publisher's title for this album"],
16 v       [3, "pub_data", "Publication-Data", [], "metadata about the album's publication"],
17 v       [4, "tracks", "Track", ["J0"], "individual track descriptions and content"],
18 v       [5, "total_tracks", "Integer", ["W1"], "total track count"],
19 v       [6, "cover_art", "Image", ["[0]", " cover art image for this album"]
20 v     ],
21 v     ["Publication-Data", "Record", [], "who and when of publication", [
22 v       [1, "publisher", "String", [], "record label that released this album"],
23 v       [2, "release_date", "String", ["/date"], "and when did they let this drop"]
24 v     ],
25 v     ["Image", "Record", [], "pretty picture for the album or track", [
26 v       [1, "image_format", "Image-Format", [], "what type of image file?"],
27 v       [2, "image_content", "Binary", [], "the image data in the identified format"]
28 v     ],
29 v     ["Image-Format", "Enumerated", [], "can only be one, but can extend list", [
30 v       [1, "PNG", ""],
31 v       [2, "JPEG", ""],
32 v       [3, "GIF", ""]
33 v     ],
34 v     ["Artist", "Record", [], "interesting information about a performer" ]
35 v   ]
36 v }
```

JSON XSD

```
1 v {
2 v   "$schema": "http://json-schema.org/draft-07/schema#",
3 v   "$id": "http://fake-audio.org/music-lib",
4 v   "title": "Music Library",
5 v   "version": "1.1",
6 v   "description": "This information model defines a library of audio tracks, organized by
7 v     license: \"CC0-1.0\",
8 v     $ref": "#/definitions/Library",
9 v   "definitions": {
10 v     "Library": {
11 v       "title": "Library",
12 v       "type": "object",
13 v       "description": "Top level of the library is a map of CDs by barcode",
14 v       "additionalProperties": false,
15 v       "minProperties": 1,
16 v       "maxProperties": 255,
17 v       "properties": {
18 v         "Barcode": {
19 v           "type": "string"
20 v         }
21 v       }
22 v     }
23 v   }
24 v }
```

XSD

```
1 v <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:jadn="jadn_base_types">
2 v   <xs:import schemaLocation=".//data/xsd/jadn_base_types.xsd" namespace="jadn_base_t
3 v     <xs:annotation>
4 v       <xs:documentation>Source: https://raw.githubusercontent.com/ScreamBun/sb-jadn-base-types/0.1.0/jadn_base_types.xsd</xs:documentation>
5 v     <xs:annotation>
6 v       <xs:import>
7 v         <xs:complexType name="Library">
8 v           <xs:annotation>
9 v             <xs:documentation>Top level of the library is a map of CDs by barcode</xs:documentation>
10 v           </xs:annotation>
11 v         </xs:complexType>
12 v       </xs:import>
13 v     </xs:annotation>
14 v   </xs:import>
15 v </xs:schema>
```

Data Validation

Validate data instances in JSON, XML, CBOR, or JSON
Compact against a JADN Schema



Home



Creation

Create and edit JADN schemas using forms, view JADN schemas in JSON format.
Create schema compliant data instances (documents, messages).



Schema Visualization

Convert a JADN Schema into different visual representations.



Schema Translation

Translate a JADN Schema to another Schema format.
Translate a JSON Schema to a JADN Schema.

Schemas Data Instances

GraphViz HTML JDL Markdown PlantUML

JSON Relax (XML) XSD



Data Validation

Validate data instances in various data language formats against a JADN Schema.



Schema Transformation

Convert one or more JADN Schemas into a different but related Schema (resolve references, simplify by removing extensions, strip comments, etc).



Example Data Generation

Generate various example data instances based off of a Schema.

CBOR JSON Relax (XML)

Go to: Validation

To Start : [click here](#)

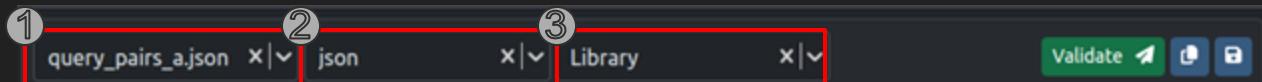
Note: Only a JADN Schema can be uploaded.

① Select data to validate.

② Make sure the correct data format is selected.

③ Make sure the correct data type being validated is selected.

In this example, the data is a json file being validated against the Library type from the music-database.jadn



```
1 v {  
2   "action": "query",  
3   "target": {  
4     "features": [  
5       "pairs"  
6     ]  
7   }  
8 }
```

Select data to validate

Starting by uploading data:

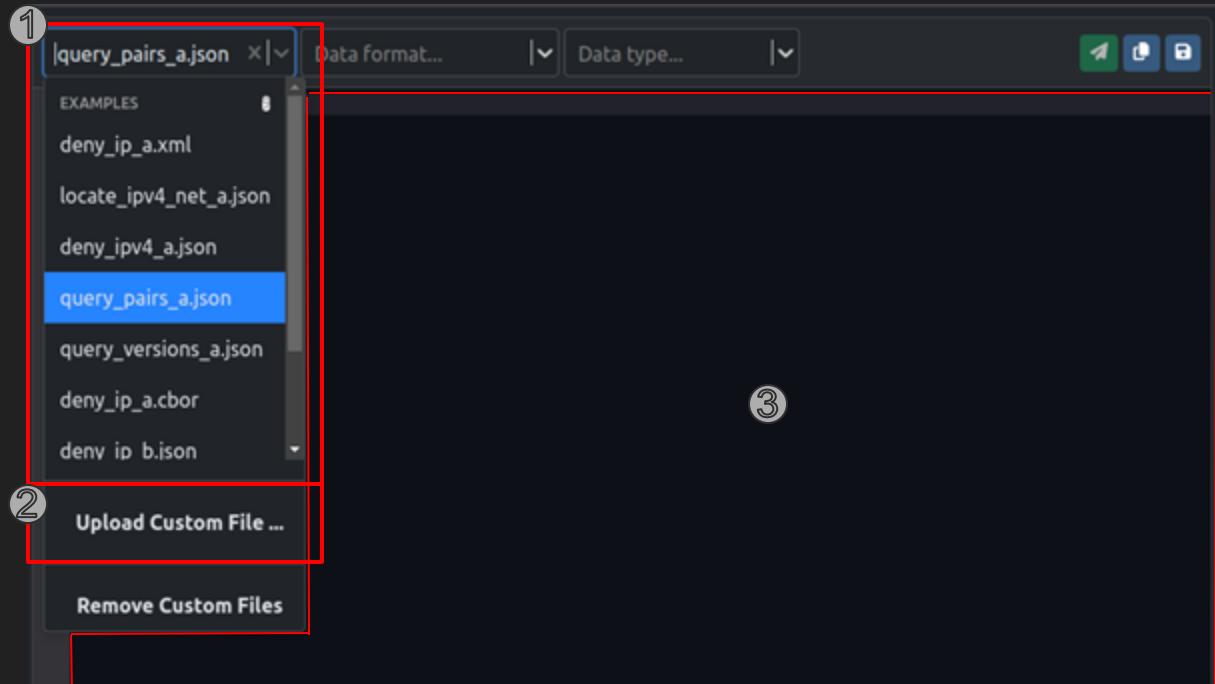
- ① Select example data

or

- ② Upload a custom file

or

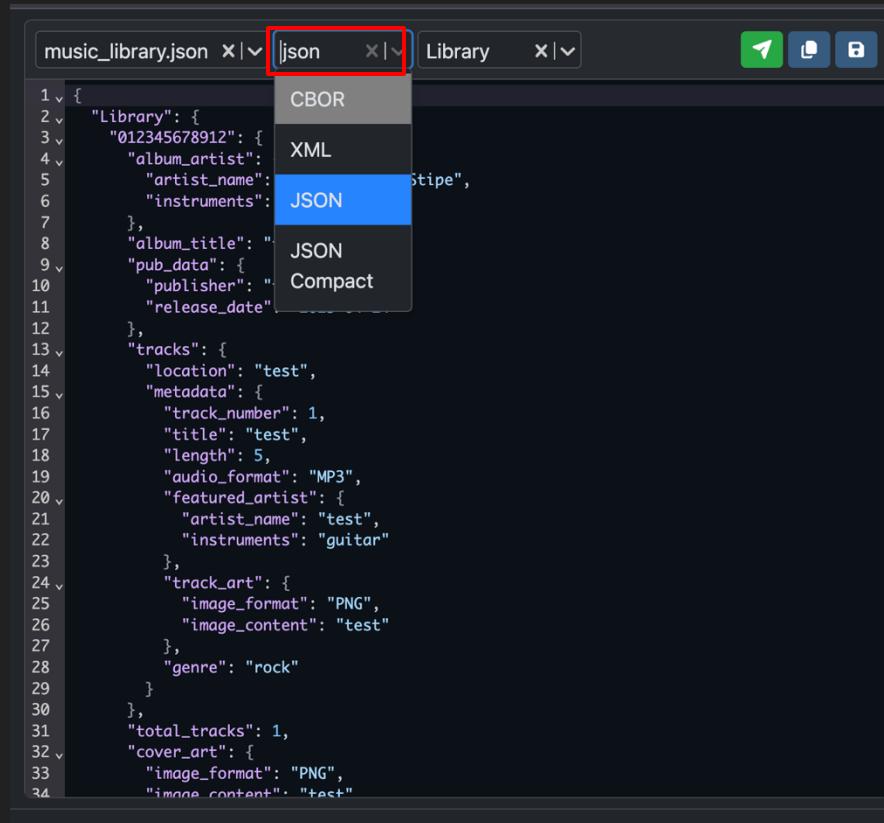
- ③ type/paste directly into code editor.



Make sure to select the correct data format

The JADN Sandbox can currently validate JSON, CBOR, XML, and JSON Compact data against a JADN Schema.

If using a data file, the data format drop down may be pre-selected.



The screenshot shows the JADN Sandbox interface with a code editor window titled "music_library.json". The code is a JSON schema definition. Above the code, there is a dropdown menu with five options: "CBOR", "XML", "JSON", and "Compact". The "JSON" option is highlighted with a blue background, indicating it is selected. The "CBOR", "XML", and "Compact" options are shown in grey. The "Library" tab is also visible in the top bar.

```
1 {  
2   "Library": {  
3     "012345678912": {  
4       "album_artist": "Stipe",  
5       "artist_name": "Bono",  
6       "instruments": "Guitar",  
7     },  
8     "album_title": "Rattle and Hum",  
9     "pub_data": {  
10       "publisher": "Columbia",  
11       "release_date": "1985-05-01T00:00:00Z"  
12     },  
13     "tracks": {  
14       "location": "test",  
15       "metadata": {  
16         "track_number": 1,  
17         "title": "test",  
18         "length": 5,  
19         "audio_format": "MP3",  
20         "featured_artist": {  
21           "artist_name": "test",  
22           "instruments": "guitar"  
23         },  
24         "track_art": {  
25           "image_format": "PNG",  
26           "image_content": "test"  
27         },  
28         "genre": "rock"  
29       }  
30     },  
31     "total_tracks": 1,  
32     "cover_art": {  
33       "image_format": "PNG",  
34       "image_content": "test"  
35     }  
36   }  
37 }
```

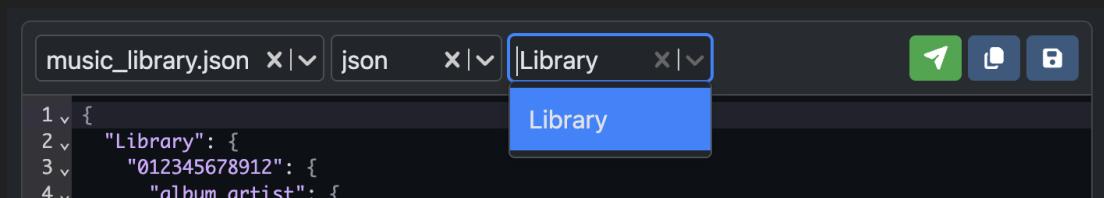
Make sure the correct data type is selected

The Data types available to validate against are based off of the roots from the JADN Schema file.

In this example, notice that the uploaded schema music-database.jadn has exports that correspond to the valid datatypes:

Library

```
{  
    "meta": {  
        "title": "Music Library",  
        "package": "http://fake-audio.org/music-lib",  
        "version": "1.1",  
        "description": "This information model defines a library of audio tracks, organized by  
        "license": "CC0-1.0",  
        "roots": ["Library"]  
    },  
}
```



The screenshot shows a JSON editor interface. At the top, there are three tabs: 'music_library.json' (selected), 'json' (disabled), and 'Library' (selected). Below the tabs, the JSON schema is displayed. The root node 'Library' is highlighted with a blue background. The schema content is as follows:

```
1 <pre>{</pre>  
2 <pre>    "Library": {</pre>  
3 <pre>        "012345678912": {</pre>  
4 <pre>            "album_artist": {</pre>
```

Validate

Once you have selected:

- A Schema,
- Data,
- Data format, and
- Data type

Click the bright, green “Validate” button.

The screenshot shows a user interface for validating JSON data against a schema. At the top, there are tabs for "query_pairs_a.json" (selected), "json" (disabled), and "Library" (disabled). On the right, there is a green "Validate" button with a white play icon, which is highlighted with a red box. Below the tabs, there is a status bar with the text "Validate the data against the given schema". The main area contains a JSON code editor with the following content:

```
1 {
2   "action": "query",
3   "target": [
4     "features": [
5       "pairs"
6     ]
7   ]
}
```

Invalid Results

An invalid data file will return errors to help fix the data.

The screenshot shows a code editor interface with a dark theme. At the top, there are tabs for "query_pairs_a.json", "json", and "Library". On the right side of the header are icons for saving, opening, and closing. The main area contains the following JSON code:

```
1 {  
2   "action": "query",  
3   "target": (  
4     "features": [  
5       "pairs"  
6     ]  
7   )  
8 }
```

Below the code, a series of red error messages are displayed in a sidebar:

- ! field required at artist
- ! object of type
! 'builtin_function_or_method' has no len() at title
- ! field required at pub_data
- ! field required at tracks
- ! field required at cover_art

At the bottom right of the sidebar, there is a "Clear All" button.

Valid Results

A valid data file will return a green success notification.

The screenshot shows a software interface for validating JSON data. At the top, there are three dropdown menus: "Select a data file...", "json", and "Album". To the right of these are three small icons: a green arrow, a square, and a document. Below the menu bar is a code editor window displaying a JSON object. The code is numbered from 1 to 27. The object describes an album track, including details like artist, title, publication date, tracks, and cover art. The code editor has a dark background with light-colored text. In the bottom right corner of the application window, there is a green notification bar with a white checkmark icon and the text "Validation success".

```
1v {
2v   "artist": {
3v     "artist_name": "the Beatles",
4v     "instruments": ["vocals"]
5v   },
6v   "title": "Hey Jude",
7v   "pub_data": {
8v     "label": "Capital Records",
9v     "rel_date": "1979-05-11"
10v   },
11v   "tracks": [
12v     {
13v       "t_number": 1,
14v       "title": "Can't Buy Me Love",
15v       "length": "00:02:19",
16v       "featured": [],
17v       "audio": {
18v         "a_format": "MP3",
19v         "a_content": "dGhlIEJlYXRsZXMu"
20v       }
21v     }
22v   ],
23v   "cover_art": {
24v     "i_format": "JPG",
25v     "i_content": "dGhlIEJlYXRsZXMu"
26v   }
27v }
```

Validation success

Schema Transformation

Strip comments from Schemas
or Resolve a schema with references

JADN Sandbox Home Creation Visualization Translation Validation Transformation Generation About

Home

 Creation Create and edit JADN schemas using forms, view JADN schemas in JSON format. Create schema compliant data instances (documents, messages).	 Schema Visualization Convert a JADN Schema into different visual representations.	 Schema Translation Translate a JADN Schema to another Schema format. Translate a JSON Schema to a JADN Schema.
Schemas Data Instances	GraphViz HTML JIDL MarkDown PlantUML	JSON Relax (XML) XSD
 Data Validation Validate data instances in various data language formats against a JADN Schema.	 Schema Transformation Convert one or more JADN Schemas into a different but related Schema (resolve references, simplify by removing extensions, strip comments, etc).	 Example Data Generation Generate various example data instances based off of a Schema.
CBOR JSON Relax (XML)		

Theme: v0.10.0_1705587211962

Go to: Transformation

To Start :

Select at least one schema to upload.

Notice that there is no code editor. Therefore, schemas must be uploaded.

Here you can see 2 schemas selected to start:

- unresolved-ls.jadn
- ocls-v1.1-lang_resolved.jadn

The screenshot shows a user interface for selecting schemas. At the top, there are tabs for 'unresolved-ls.jadn' and 'oc2ls-v1.1-lang_resolved.jadn'. Below each tab is a code editor window displaying JSON-like schema definitions.

unresolved-ls.jadn

```
1 v {
2 v   "info": {
3 v     "package": "https://github.com/oasis-tcs/openc2-ap-hunt",
4 v     "version": "0-wd01",
5 v     "title": "Unresolved ls schema",
6 v     "description": "Data definitions for Threat Hunting (TH) functions",
7 v     "namespaces": {
8 v       "ls": "http://oasis-open.org/openc2/oc2ls/v1.1"
9 v     },
10 v     "exports": ["Language-Spec-Types"]
11 v   },
12 v   "types": [
13 v     {"$ref": "#/Language-Spec-Types", "Record": [], "description": "for each track there's a file with the audio and a metadata record", "examples": [
14 v       {"$ref": "#/ls:Artifact", "ls:Artifact": "[0]", "description": "An array of bytes representing a file-like object or a link to that object."}
15 v     ], "title": "Command", "type": "Object", "x-alias": "ls:Command-ID", "x-type": "String", "x-value": "[0]"}, {"$ref": "#/Language-Spec-Types", "Record": [], "description": "The properties of a hardware device.", "examples": [
16 v       {"$ref": "#/ls:Device", "ls:Device": "[0]", "description": "The properties of a hardware device."}
17 v     ], "title": "Device", "type": "Object", "x-alias": "ls:Device", "x-type": "String", "x-value": "[0]"}
18 v   ]
19 v }
```

oc2ls-v1.1-lang_resolved.jadn

```
1 v {
2 v   "info": {
3 v     "package": "http://oasis-open.org/openc2/oc2ls/v1.1",
4 v     "title": "OpenC2 Language Profile",
5 v     "description": "Language Profile from the OpenC2 Language Specification version 1.1",
6 v     "exports": ["OpenC2-Command", "OpenC2-Response"]
7 v   },
8 v   "types": [
9 v     {"$ref": "#/OpenC2-Command", "Record": [], "description": "The Command defines an Action to be performed on a Target", "examples": [
10 v       {"$ref": "#/action", "action": "[0]", "description": "The task or activity to be performed (i.e., the 'verb')."}, {"$ref": "#/target", "target": "[0]", "description": "The object of the Action. The Action is performed on the Target."}, {"$ref": "#/args", "args": "[0]", "description": "Additional information that applies to the Command."}
11 v     ], "title": "Command", "type": "Object", "x-alias": "OpenC2-Command", "x-type": "String", "x-value": "[0]"}, {"$ref": "#/OpenC2-Response", "Record": [], "description": "The Response is the result of a Command being executed on a Target", "examples": [
12 v       {"$ref": "#/status", "status": "[0]", "description": "The status of the response."}
13 v     ], "title": "Response", "type": "Object", "x-alias": "OpenC2-Response", "x-type": "String", "x-value": "[0]"}
14 v   ]
15 v }
```

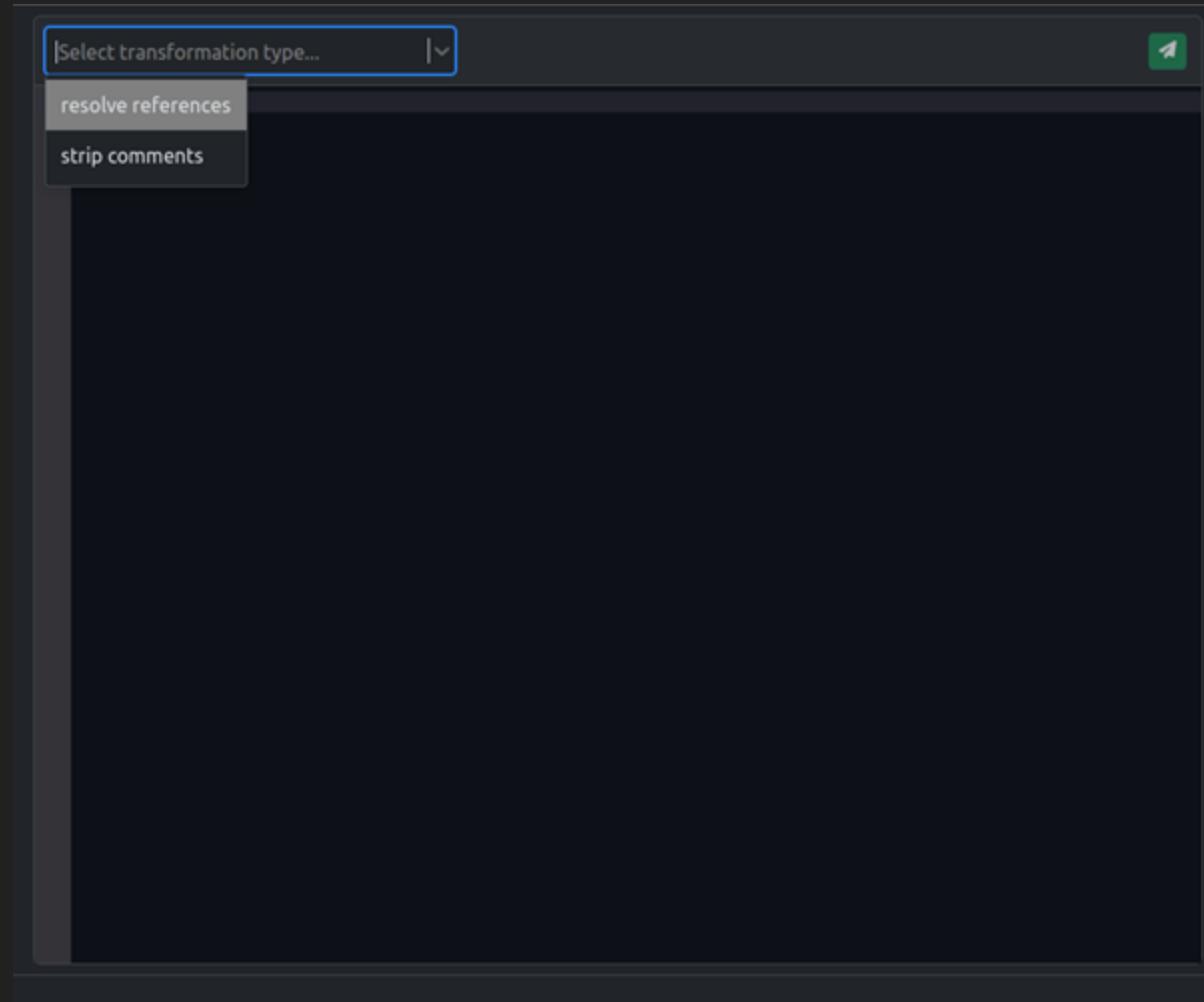
After selecting the desired schemas, you can select your desired transformation:

Resolve references

- Create a single JADN schema from an unresolved schema and its imports

Strip comments

- Remove comments from a JADN schema to reduce size or visual clutter

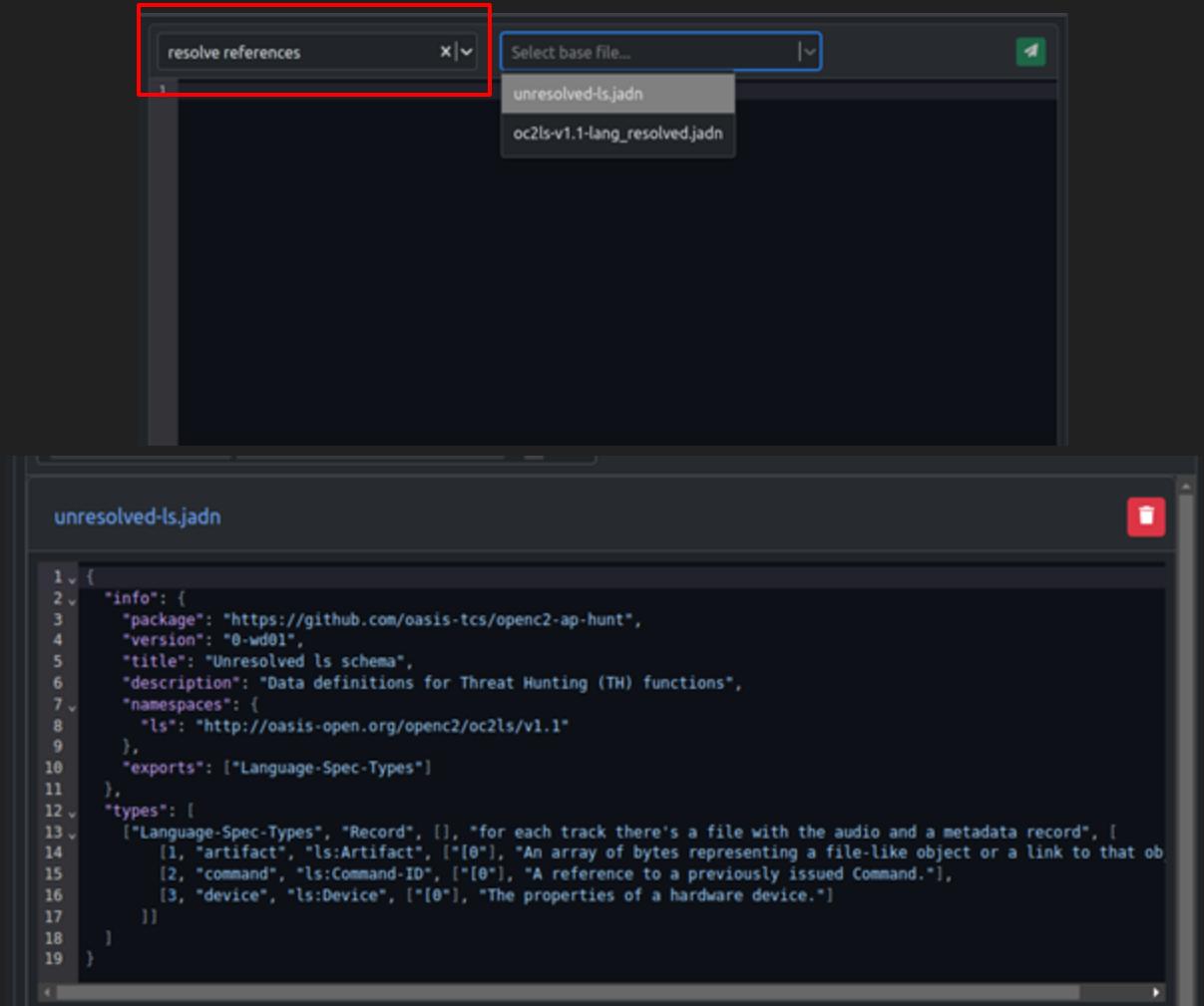


Resolve References ¹

For resolving references, you will need to select a base file. This file is the file you would like to be resolved.

The other uploaded files will be used to help resolve the chosen base file.

In this example, the base file is :
unresolved-ls.jadn



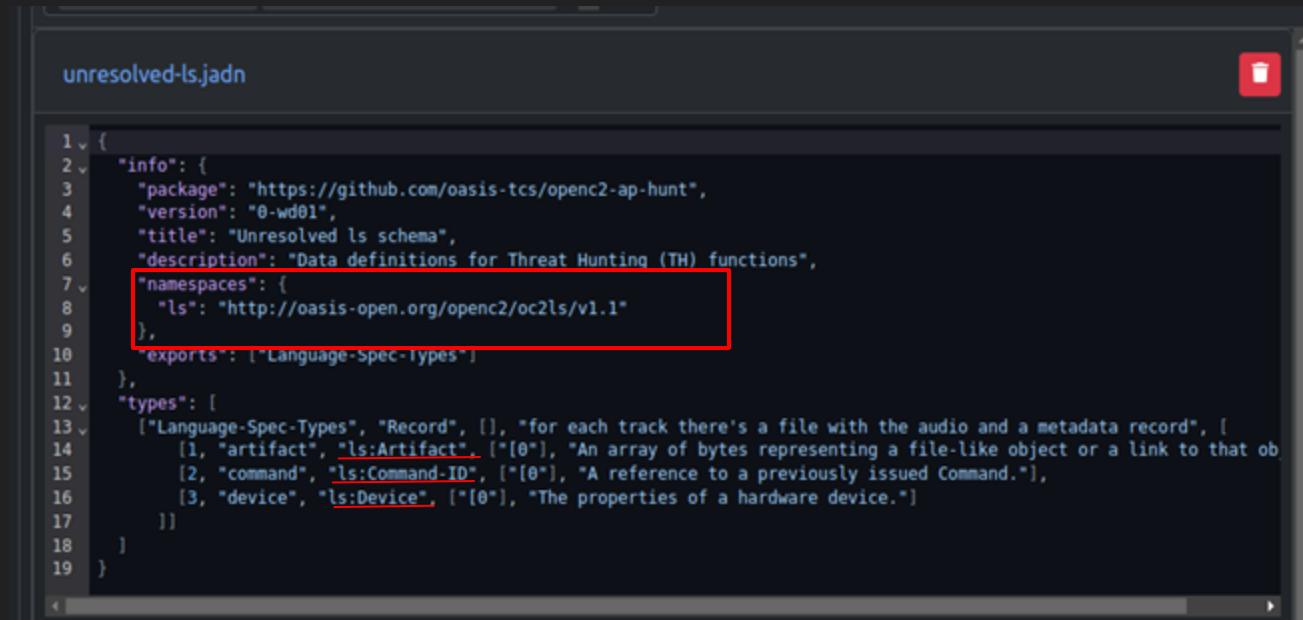
The screenshot shows a software interface for resolving schema references. At the top, there is a toolbar with a red box around the 'resolve references' button. To its right is a dropdown menu labeled 'Select base file...' with a dropdown arrow. Below the toolbar, two files are listed in a dropdown menu: 'unresolved-ls.jadn' (which is highlighted) and 'oc2ls-v1.1-lang_resolved.jadn'. The main window below shows the 'unresolved-ls.jadn' schema file. The file content is as follows:

```
1 v {
2 v   "info": {
3 v     "package": "https://github.com/oasis-tcs/openc2-ap-hunt",
4 v     "version": "0-wd01",
5 v     "title": "Unresolved ls schema",
6 v     "description": "Data definitions for Threat Hunting (TH) functions",
7 v     "namespaces": {
8 v       "ls": "http://oasis-open.org/openc2/oc2ls/v1.1"
9 v     },
10 v     "exports": ["Language-Spec-Types"]
11 v   },
12 v   "types": [
13 v     ["Language-Spec-Types", "Record", [], "for each track there's a file with the audio and a metadata record", [
14 v       [1, "artifact", "ls:Artifact", "[0]", "An array of bytes representing a file-like object or a link to that ob
15 v       [2, "command", "ls:Command-ID", "[0]", "A reference to a previously issued Command."],
16 v       [3, "device", "ls:Device", "[0]", "The properties of a hardware device."]
17 v     ]]
18 v   ]
19 v }
```

1. A resolved Schema is a schema without references.

What is an unresolved Schema?

A Schema is unresolved if it has namespaces (references to types defined in other schemas).



```
unresolved-ls.jadn

1 v {
2 v   "info": {
3 v     "package": "https://github.com/oasis-tcs/openc2-ap-hunt",
4 v     "version": "0-wd01",
5 v     "title": "Unresolved ls schema",
6 v     "description": "Data definitions for Threat Hunting (TH) functions",
7 v     "namespaces": {
8 v       "ls": "http://oasis-open.org/openc2/oc2ls/v1.1"
9 v     },
10 v     "exports": ["Language-Spec-Types"]
11 },
12 v   "types": [
13 v     ["Language-Spec-Types", "Record", [], "for each track there's a file with the audio and a metadata record", [
14 v       [1, "artifact", "ls:Artifact", "[[0]", "An array of bytes representing a file-like object or a link to that ob
15 v         [2, "command", "ls:Command-ID", "[[0]", "A reference to a previously issued Command."],
16 v           [3, "device", "ls:Device", "[[0]", "The properties of a hardware device."]
17 v         ]]
18   ]
19 }
```

Click Transform to resolve the base file.



Result: A resolved Schema

Notice there is no namespace and that the types from the unresolved schema can be referenced within the resolved schema.

| s: Artifact => Artifact

Ls: Command-ID → Command-ID

Ls: Device → Device

```
resolve references x | v unresolved-ls.jadn x | v

1 v {
2   "info": {
3     "package": "https://github.com/oasis-tcs/openc2-ap-hunt",
4     "version": "0-wd01",
5     "title": "Unresolved ls schema",
6     "description": "Data definitions for Threat Hunting (TH) functions",
7     "exports": ["Language-Spec-Types"]
8   },
9
10  "types": [
11    {"Language-Spec-Types", "Record", [], "for each track there's a file with the audio and a metadata record", [
12      {1, "artifact", "Artifact", ["[0]", "An array of bytes representing a file-like object or a link to that object."]},
13      {2, "command", "Command-ID", ["[0]", "A reference to a previously issued Command."]},
14      {3, "device", "Device", ["[0]", "The properties of a hardware device."]}
15    ]},
16
17    {"Artifact", "Record", [{"1": ""}, {
18      {1, "mime_type", "String", ["[0]", "Permitted values specified in the IANA Media Types registry, [RFC6838]"]},
19      {2, "payload", "Payload", ["[0]", "Choice of literal content or URL"]},
20      {3, "hashes", "Hashes", ["[0]", "Hashes of the payload content"]}
21    ]},
22
23    {"Device", "Map", [{"1": ""}, {
24      {1, "hostname", "Hostname", ["[0]", "A hostname that can be used to connect to this device over a network"]},
25      {2, "idn_hostname", "IDN-Hostname", ["[0]", "An internationalized hostname that can be used to connect to this devi
26      {3, "device_id", "String", ["[0]", "An identifier that refers to this device within an inventory or management syst
27    ]}},
28
29    {"URI", "String", ["uri"], "Uniform Resource Identifier, [RFC3986]", []},
30
31    {"Hashes", "Map", [{"1": "Cryptographic hash values", [
32      {1, "md5", "Binary", ["x", "[16", "]16", "[0]", "MD5 hash as defined in [RFC1321]"],
33      {2, "sha1", "Binary", ["x", "[20", "]20", "[0]", "SHA-1 hash as defined in [RFC3174]"],
34      {3, "sha256", "Binary", ["x", "[32", "]32", "[0]", "SHA-256 hash as defined in [RFC4634]"],
35      {4, "sha512", "Binary", ["x", "[64", "]64", "[0]", "SHA-512 hash as defined in [RFC4634]"]
36    ]}}]
37  ]}}
```

Result: Strip Comments

When stripping comments, results will return all schemas without comments.

The screenshot displays a code editor interface with two tabs. The top bar has a search field containing "strip comments" with a red border around it, and a green arrow icon in the top right corner.

unresolved-ls

```
1 v {
2 v   "info": {
3 v     "package": "https://github.com/oasis-tcs/openc2-ap-hunt",
4 v     "version": "0-wd01",
5 v     "title": "Unresolved ls schema",
6 v     "description": "Data definitions for Threat Hunting (TH) functions",
7 v     "namespaces": {
8 v       "ls": "http://oasis-open.org/openc2/oc2ls/v1.1"
9 v     },
10 v     "exports": ["Language-Spec-Types"]
11 v   },
12 v   "types": [
13 v     ["Language-Spec-Types", "Record", [], "", [
14 v       [1, "artifact", "ls_Artifact", "[{0}], ""],
15 v       [2, "command", "ls_Command_I0", "[{0}], ""],
16 v       [3, "device", "ls_Device", "[{0}], ""]
17 v     ]]
18 v   ]
19 v }
```

oc2ls-v1.1-lang_resolved

```
1 v {
2 v   "info": {
3 v     "package": "http://oasis-open.org/openc2/oc2ls/v1.1",
4 v     "title": "OpenC2 Language Profile",
5 v     "description": "Language Profile from the OpenC2 Language Specification version 1.1",
6 v     "exports": ["OpenC2-Command", "OpenC2-Response"]
7 v   },
8 v   "types": [
9 v     ["OpenC2-Command", "Record", [], "", [
10 v       [1, "action", "Action", [], ""],
11 v       [2, "target", "Target", [], ""],
12 v       [3, "args", "Args", "[{0}], ""]
13 v     ]]
14 v   ]
15 v }
```

Example Data Generation

Automatically create valid data instances given the JADN Schema



Home



Creation

Create and edit JADN schemas using forms, view JADN schemas in JSON format.

Create schema compliant data instances (documents, messages).



Schema Visualization

Convert a JADN Schema into different visual representations.



Schema Translation

Translate a JADN Schema to another Schema format.

Translate a JSON Schema to a JADN Schema.

Schemas Data Instances

GraphViz HTML JDL Markdown PlantUML

JSON Relax (XML) XSD



Data Validation

Validate data instances in various data language formats against a JADN Schema.



Schema Transformation

Convert one or more JADN Schemas into a different but related Schema (resolve references, simplify by removing extensions, strip comments, etc).



Example Data Generation

Generate various example data instances based off of a Schema.

CBOR JSON Relax (XML)

Go to: Generation

To Start : [click here](#)

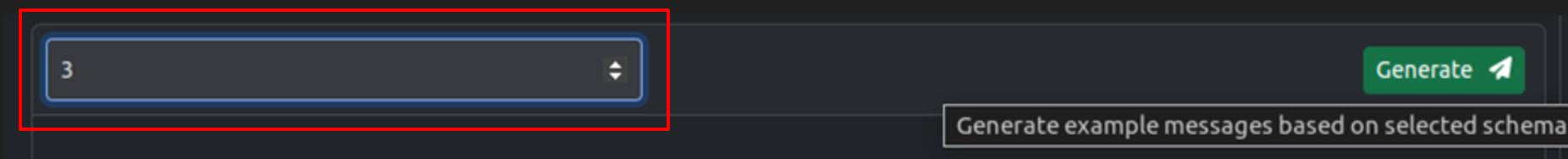
Note: Only a JADN Schema can be uploaded.

Enter the desired number of examples you would like generated. This number should be between 1 - 10.

(In this example, we have selected 3.)

After you input a number, click the green “Generate” button.

Note: The button will only be clickable if you have a valid JADN schema and the number of examples desired.



Warning: Example Generation may fail due to large Schema data. As a result, the application may lock itself. In this case, please restart docker container.

Results

If successfully generated, you will be able to view and hide the example data generated.

For each example, you can:



- Download to local storage
- Save the file to data uploader
(The file can then be found for easy access in the Data Validation page)



- Copy to clipboard

The screenshot shows a data visualization interface with three examples of generated data. Each example is a JSON object with numbered lines for readability.

Data Example #1

```
1 v {  
2 v   "artist": {  
3 v     "artist_name": "consectetur magna dolor",  
4 v     "instruments": [  
5 v       [  
6 v         "harmonica",  
7 v         "brass",  
8 v         "keyboards",  
9 v         "guitar",  
10 v        "woodwinds",  
11 v        "guitar",  
12 v        "harmonica",  
13 v        "bass",  
14 v        "harmonica",  
15 v        "drums",  
16 v        "percussion",  
17 v        "drums",  
18 v        "vocals",  
19 v        "brass",  
20 v        "bass",  
21 v        "harmonica",  
22 v     ]  
23 v   }  
24 v }
```

Data Example #2

```
1 v {  
2 v   "t_number": -4919358.7109660655,  
3 v   "title": "in",  
4 v   "length": "21:54:30.163Z",  
5 v   "featured": [  
6 v     {  
7 v       "artist_name": "magna nisi minim est",  
8 v       "instruments": [  
9 v         "harmonica",  
10 v        "brass",  
11 v        "keyboards",  
12 v        "guitar",  
13 v        "woodwinds",  
14 v        "guitar",  
15 v        "harmonica",  
16 v        "bass",  
17 v        "harmonica",  
18 v        "drums",  
19 v        "percussion",  
20 v        "drums",  
21 v        "vocals",  
22 v        "brass",  
23 v        "bass",  
24 v        "harmonica",  
25 v     ]  
26 v   }  
27 v }
```

Data Example #3

```
1 v {  
2 v   "t_number": -4919358.7109660655,  
3 v   "title": "in",  
4 v   "length": "21:54:30.163Z",  
5 v   "featured": [  
6 v     {  
7 v       "artist_name": "magna nisi minim est",  
8 v       "instruments": [  
9 v         "harmonica",  
10 v        "brass",  
11 v        "keyboards",  
12 v        "guitar",  
13 v        "woodwinds",  
14 v        "guitar",  
15 v        "harmonica",  
16 v        "bass",  
17 v        "harmonica",  
18 v        "drums",  
19 v        "percussion",  
20 v        "drums",  
21 v        "vocals",  
22 v        "brass",  
23 v        "bass",  
24 v        "harmonica",  
25 v     ]  
26 v   }  
27 v }
```

Other Features

The screenshot shows the JADN Sandbox application interface. At the top, there's a navigation bar with links for Home, Creation+, Visualization, Translation, Validation, Transformation, Generation, and About. Below the navigation bar is a main content area divided into several sections:

- Creation:** "Create and edit JADN schemas using forms, view JADN schemas in JSON format." Below it is a sub-section for "Data Instances".
- Schema Visualization:** "Convert a JADN Schema into different visual representations." Below it is a sub-section for "Schemas".
- Schema Translation:** "Translate a JADN Schema to another Schema format." Below it is a sub-section for "Relax (XML)".

On the left side, there's a sidebar with a "Data Validation" section ("Validate data instances in various data language formats against a schema") and a "CBOR JSON Relax (XML)" section. At the bottom of the sidebar, there's a "Theme" button with a dropdown menu.

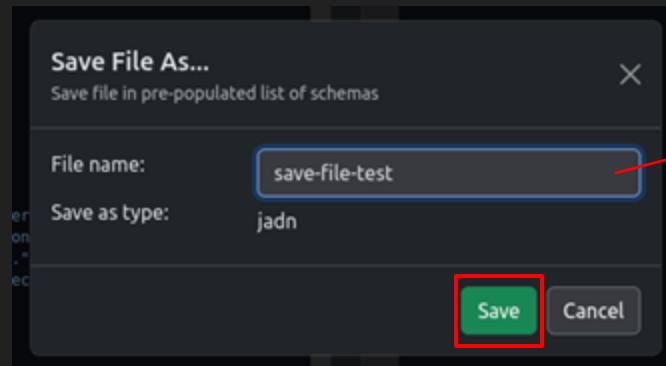
The "Theme" dropdown menu is highlighted with a red box. It contains two options: "Light" (which is selected) and "Dark".

Theme Switcher: Choose between light and dark mode.

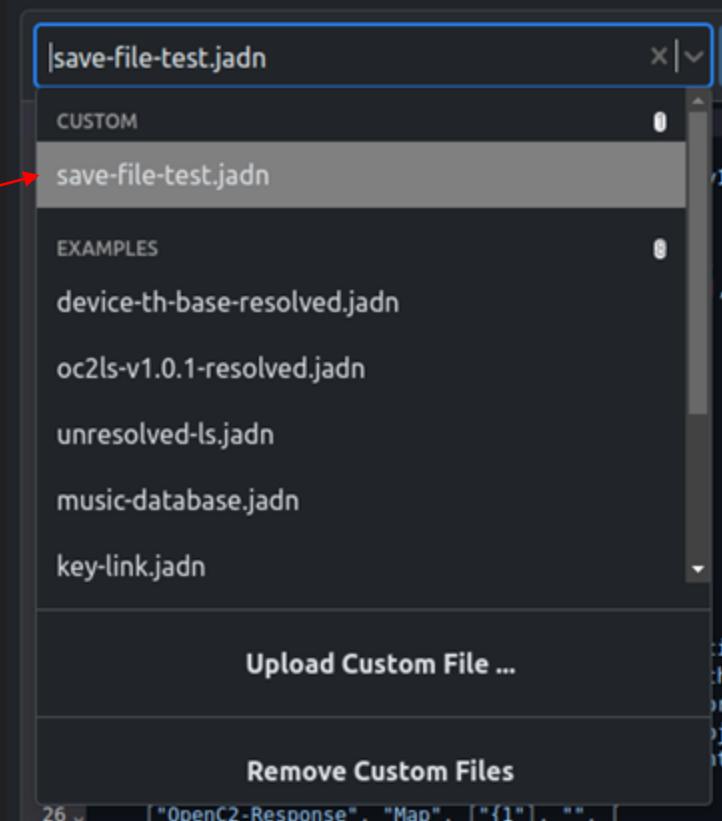
Note: Dark is the default theme.



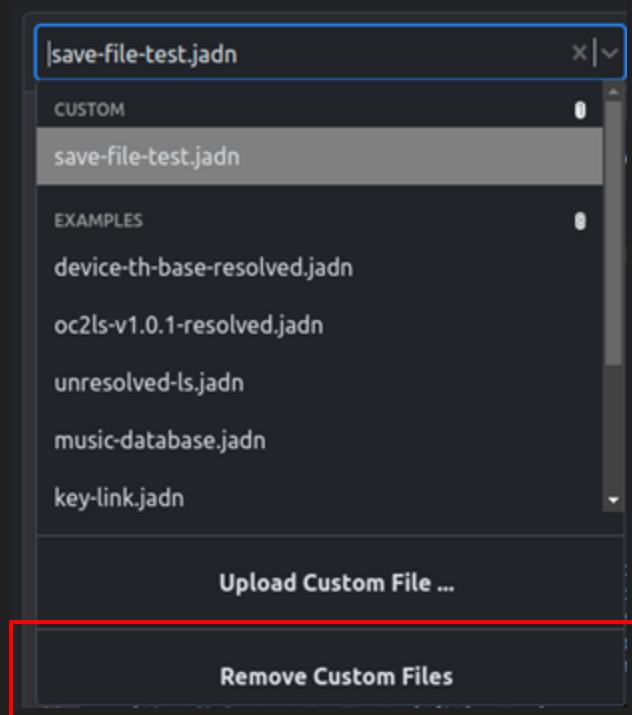
Add custom files (Save as...)



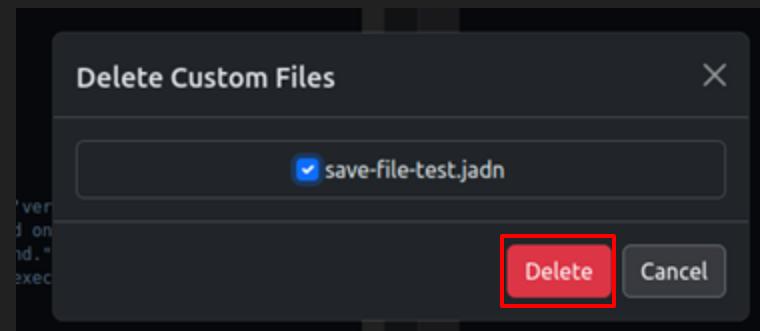
The file saver will allow you to name and save data into a file to be found in the file loader for future use.



Delete custom files

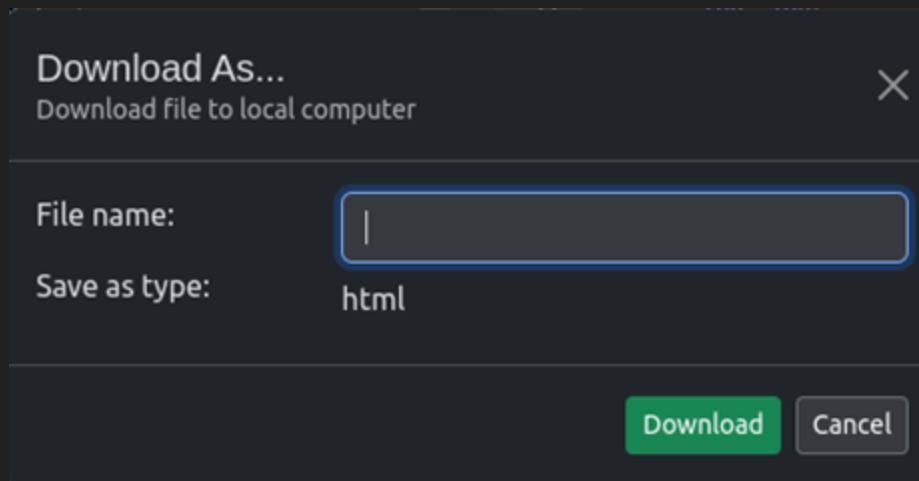


Custom files can be removed from the file loader if no longer needed.





Download files to local storage (Download As..)



Files can be named and saved to your Downloads Folder.



JADN Sandbox

Docker Desktop: Startup Guide

Docker Table of Contents

[How to Get the JADN Sandbox Image](#)

[How to Run the JADN Sandbox Image](#)

- [From Images](#)¹
- [From Containers](#)²
- [From Search Bar](#)

[How to Update the JADN Sandbox Image](#)

- [From Images](#)
- [From Search Bar](#)

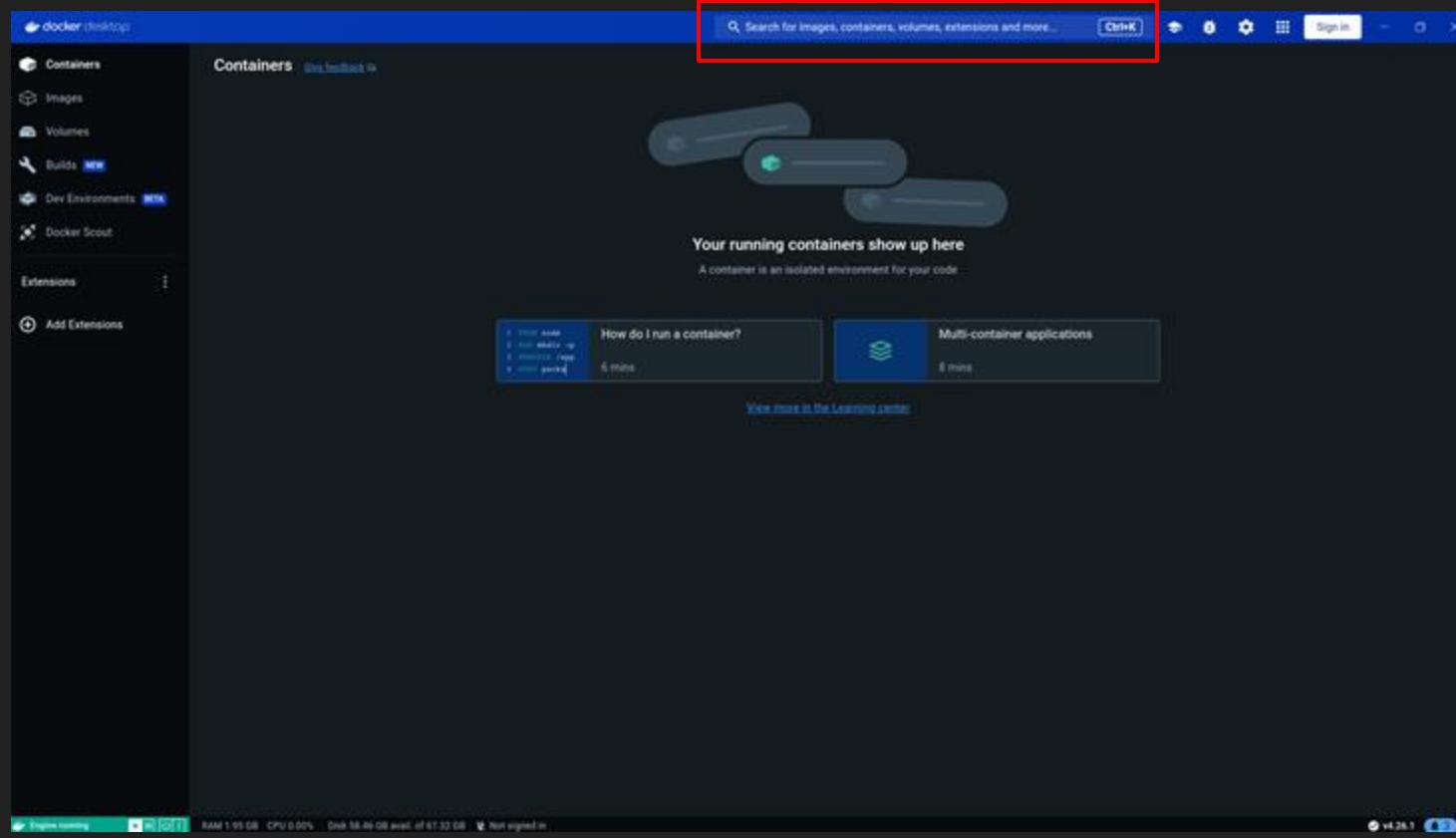
[How to Stop the JADN Sandbox Image](#)

[How to Learn More about Docker](#)

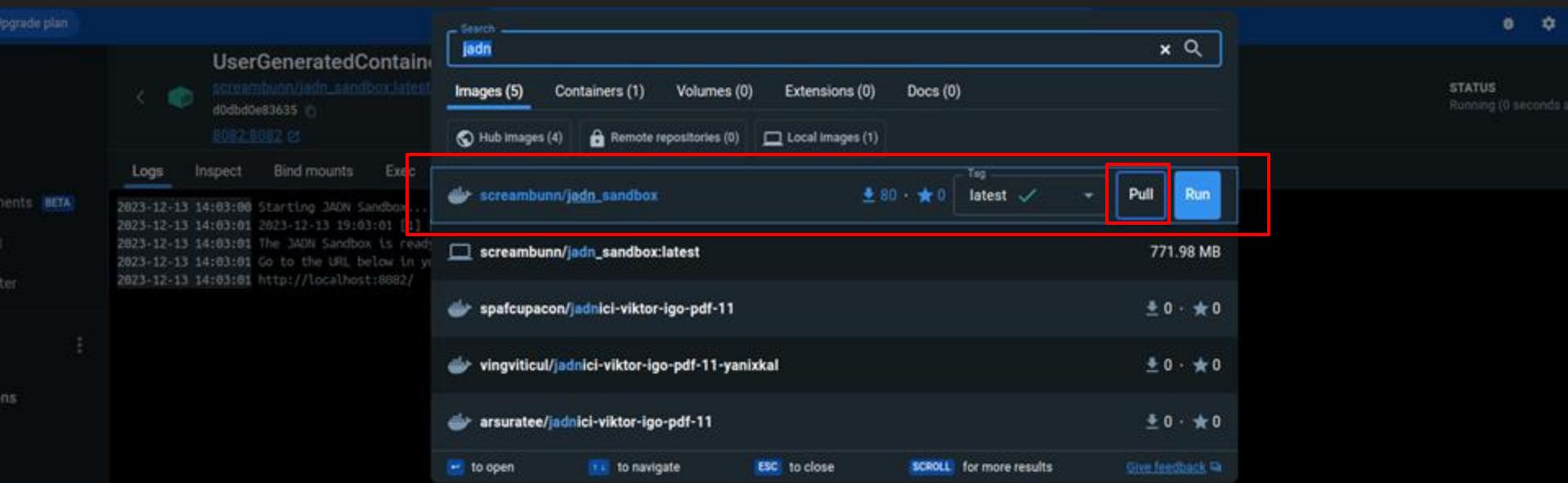
Tips:

1. A Docker image is like a set of instructions. It is the template loaded onto the container to run it.
2. A Docker container is a self-contained, runnable software application or service created from a docker image.

How to Get the JADN Sandbox Image



Click in the search bar, search: jadn



Then, click Pull: screambunn/jadn_sandbox

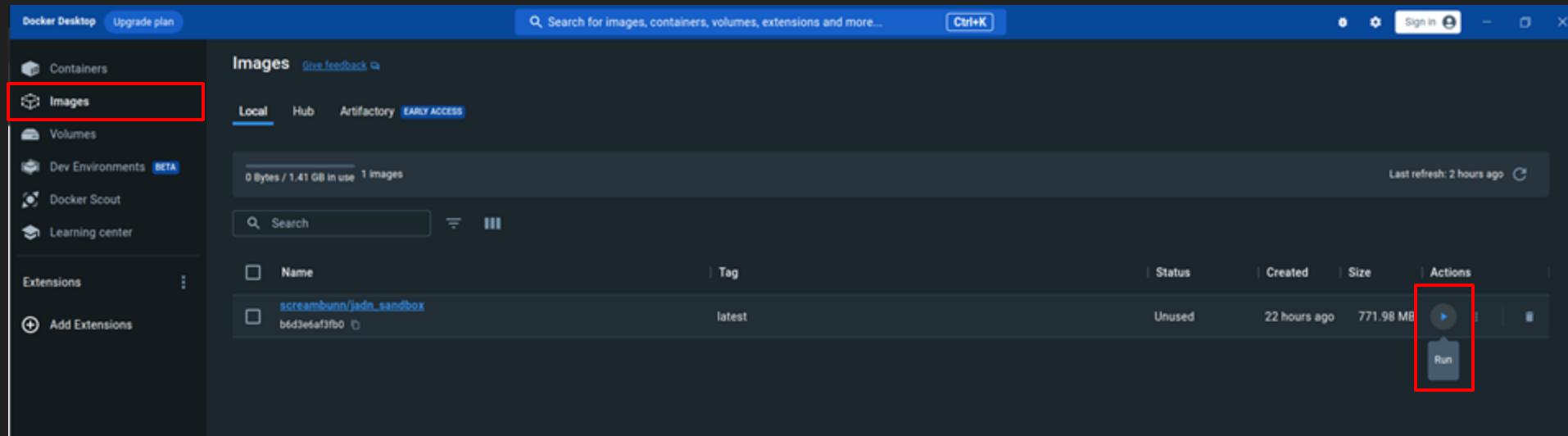
How to Run the JADN Sandbox Image

[From Images](#)

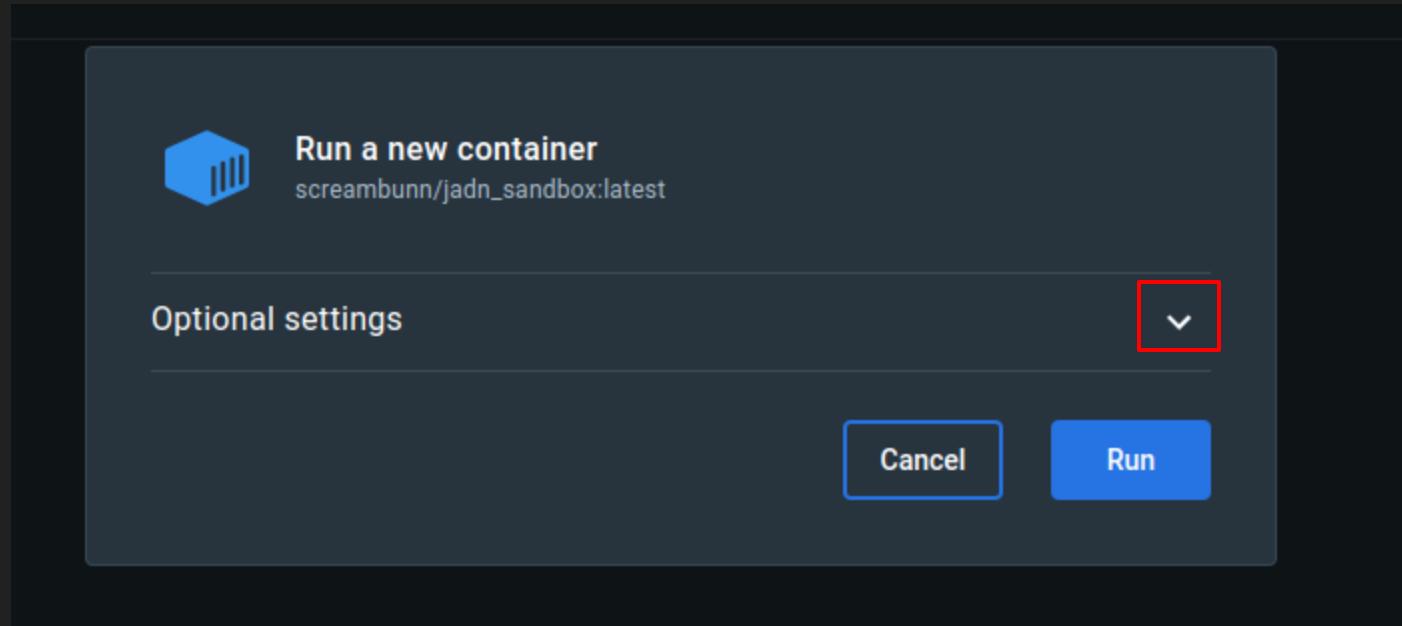
[From Containers](#)

[From Search Bar](#)

From Images



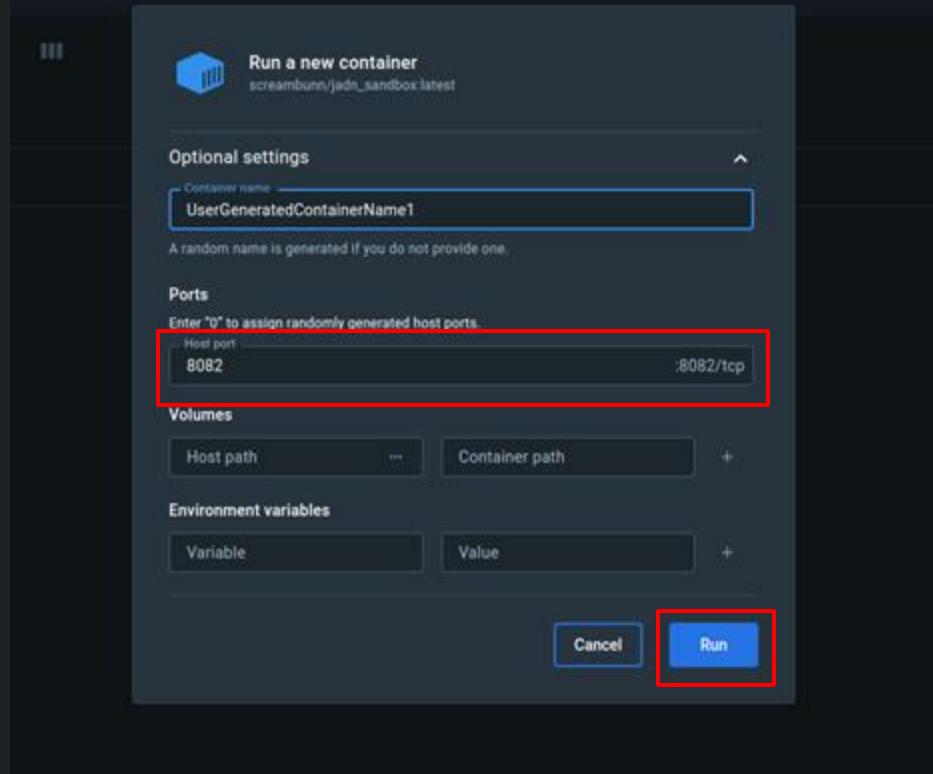
Under Images, find the desired image and click the play button to run the image.



After clicking the play button, this window will appear.

You will need to enter information on the container to be able to run the image.

Click the carrot icon (v) to expand the Optional settings.



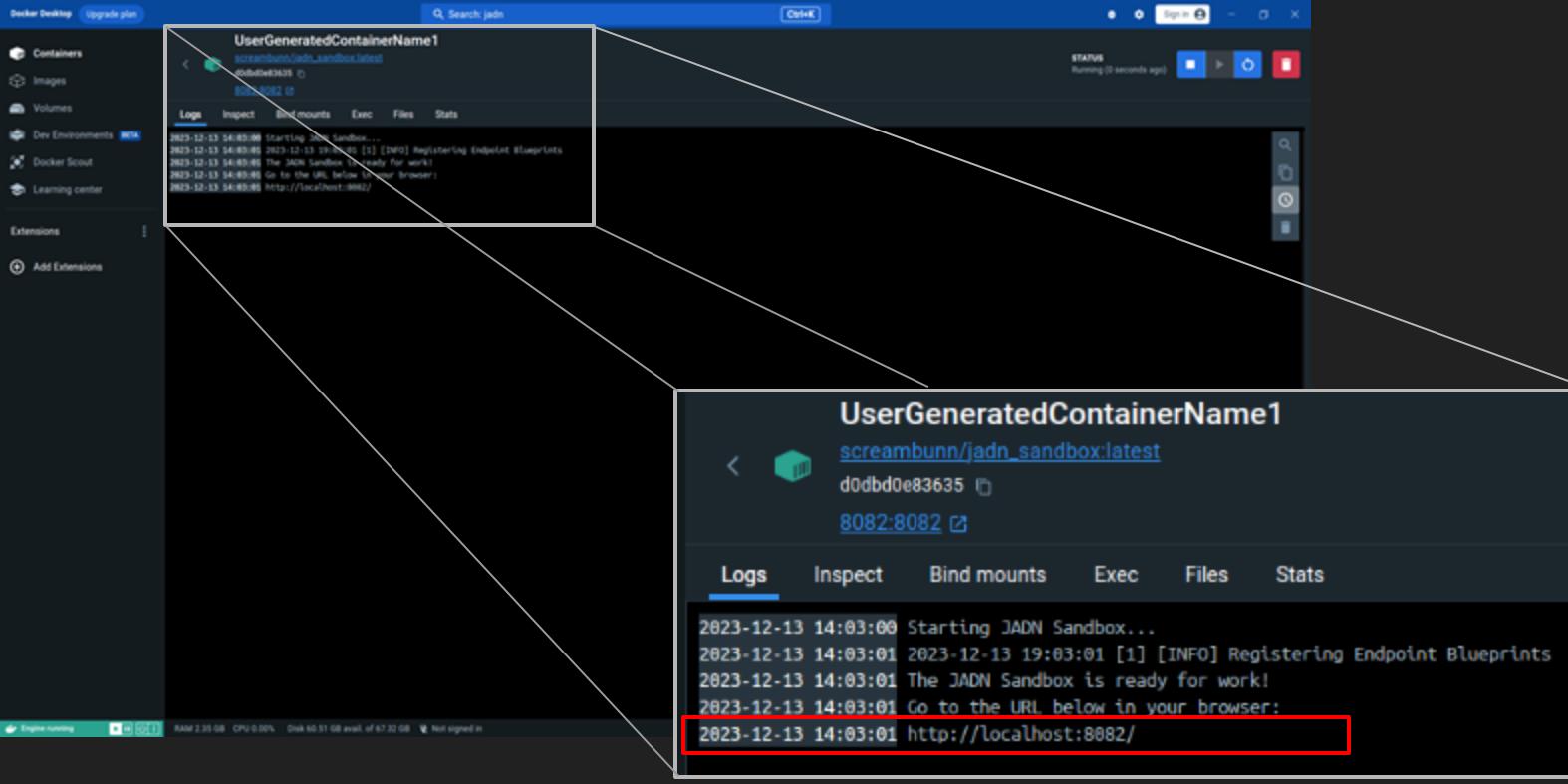
After clicking the carrot icon, you will see the settings as shown.

The Container Name is optional, but has been provided as an example (*UserGeneratedContainerName1*).

Note: no spaces allowed in the container name.

REQUIRED INFORMATION: Host Port - Enter: **8082**

Click Run.



After clicking on Run, a container will appear.

Click on the link provided to open the image in a browser window.

You can also enter the link itself in the browser:

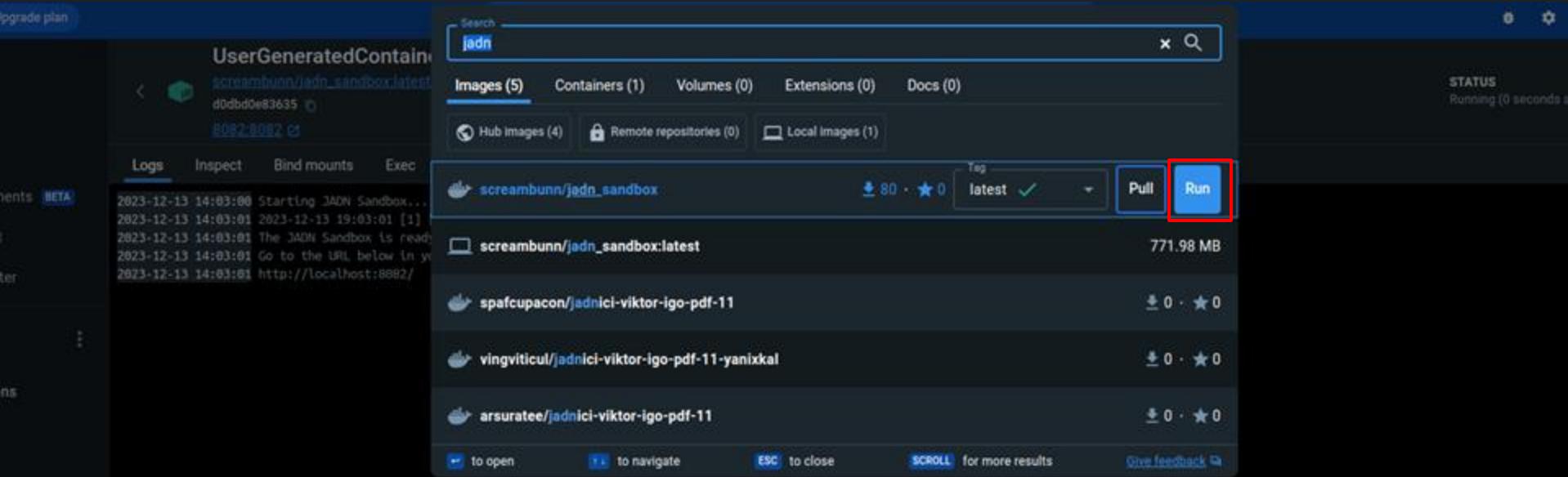
<http://localhost:8082/>

The screenshot shows the Docker Desktop application window. On the left sidebar, there are icons for Containers, Images, Volumes, Dev Environments (BETA), Docker Scout (EARLY ACCESS), Learning center, Extensions, and a button to Add Extensions. The main area displays a container named 'amazing_brahmagupta' with the image 'screambunn/jadn_sandbox:latest'. The container ID is 'a3868fda2581' and its port is '8082:8082'. The status is 'Running (39 minutes ago)'. A red box highlights the 'Stop' button in the top right corner of the container card. Below the container card, there are tabs for Logs, Inspect, Bind mounts, Terminal, Files, and Stats. The Logs tab is selected, showing the following log entries:

```
2024-02-07 14:28:23 Starting JADN Sandbox...
2024-02-07 14:28:23 2024-02-07 19:28:23 [1] [INFO] Registering Endpoint Blueprints
2024-02-07 14:28:24 The JADN Sandbox is ready for work!
2024-02-07 14:28:24 Go to the URL below in your browser:
2024-02-07 14:28:24 http://localhost:8082/
```

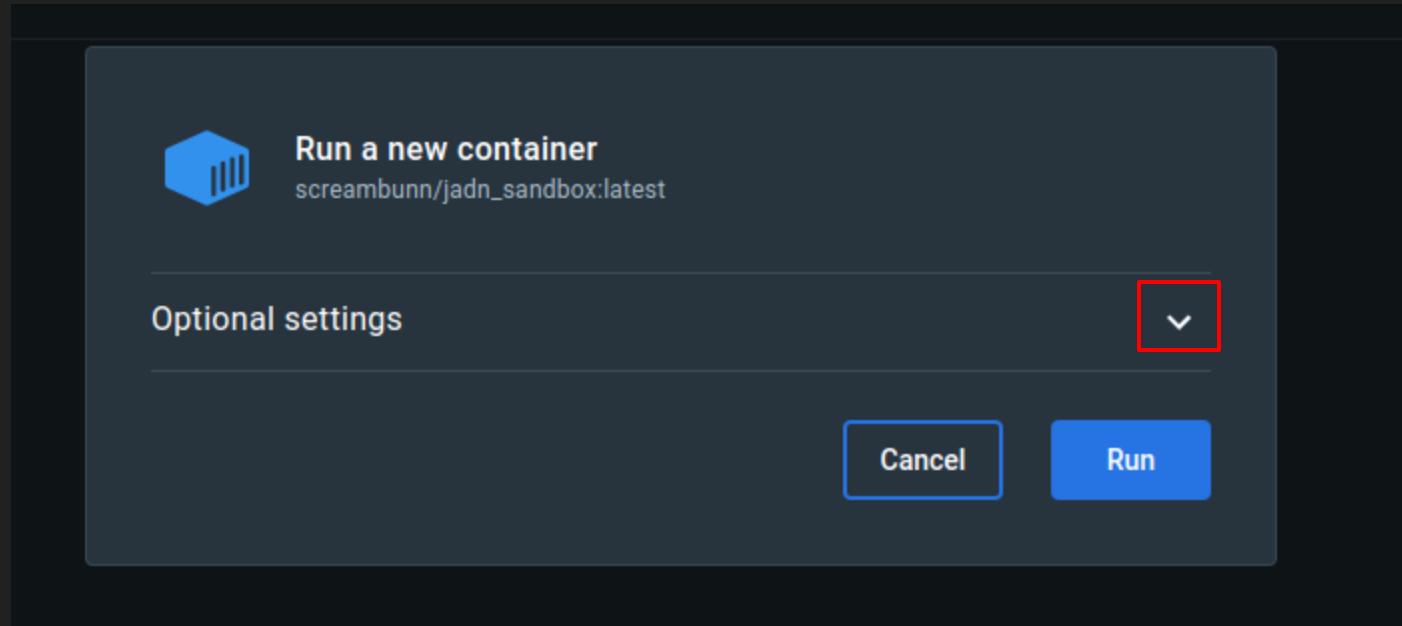
To stop the container, click the stop button.

From the Search Bar



In the search bar, search: jadn

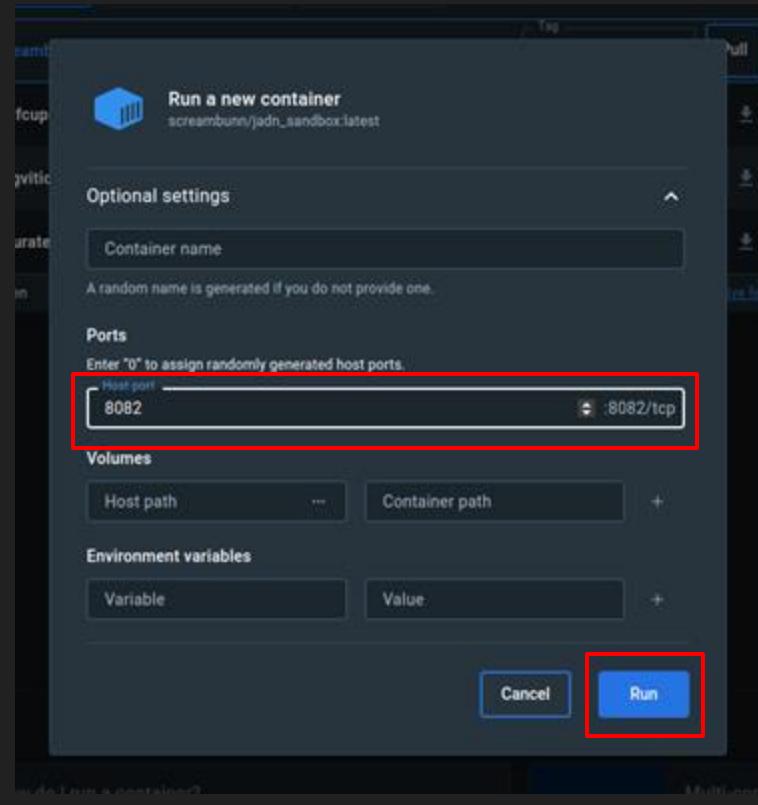
Then, click Run: screambunn/jadn_sandbox



After clicking the play button, this window will appear.

You will need to enter information on the container to be able to run the image.

Click the carrot icon (v) to expand the Optional settings.



After clicking the carrot icon, you will see the settings as shown.

The Container Name is optional.

Note: no spaces allowed in the container name.

REQUIRED INFORMATION: Host Port - Enter: **8082**

Click Run.

The screenshot shows the Docker Desktop interface. On the left sidebar, there are icons for Containers, Images, Volumes, Dev Environments (BETA), Docker Scout (EARLY ACCESS), Learning center, Extensions, and a button to Add Extensions. The main area displays a container named 'amazing_brahmagupta' with the image 'screambunn/jdn_sandbox:latest'. The container ID is 'a3868fda2581'. It is currently 'Running (39 minutes ago)'. The status bar at the top says 'Search for images, containers, volumes, extensions and more...' and shows '2K' results. Below the container details, there are tabs for Logs, Inspect, Bind mounts, Terminal, Files, and Stats. The Logs tab is selected, showing the following log entries:

```
2024-02-07 14:28:23 Starting JADN Sandbox...
2024-02-07 14:28:23 2024-02-07 19:28:23 [1] [INFO] Registering Endpoint Blueprints
2024-02-07 14:28:24 The JADN Sandbox is ready for work!
2024-02-07 14:28:24 Go to the URL below in your browser:
2024-02-07 14:28:24 http://localhost:8082/
```

The link 'http://localhost:8082/' is highlighted with a red box.

After clicking on Run, a container will appear.

Click on the link provided to open the image in a browser window.

You can also enter the link itself in the browser:

<http://localhost:8082/>

From Containers

Docker Desktop Upgrade plan

Q Search: jadn Ctrl+K

Containers Give feedback

Container CPU usage 0.04% / 800% (8 cores allocated)

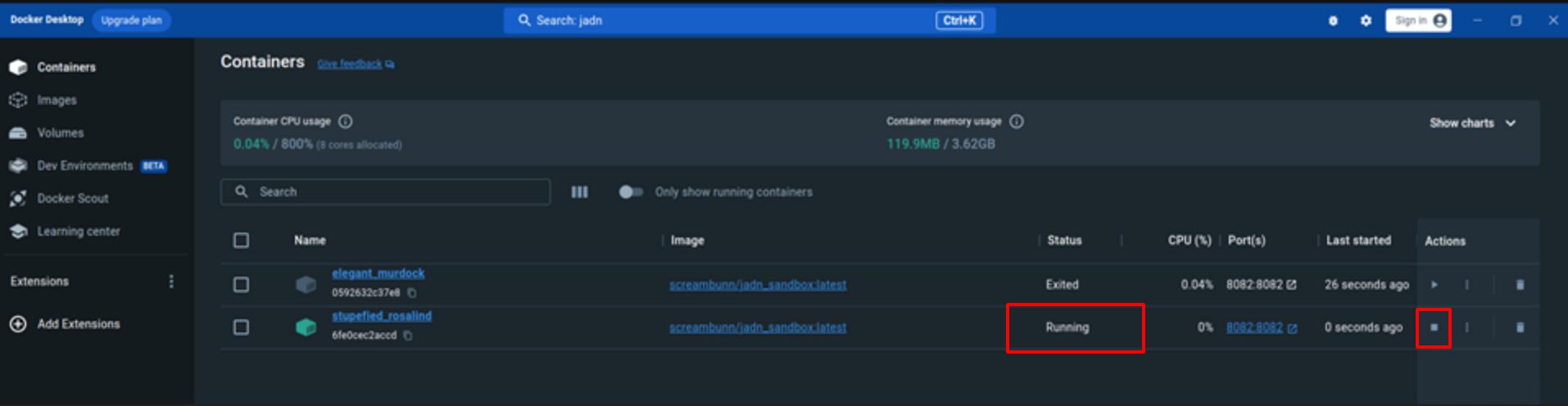
Container memory usage 119.9MB / 3.62GB

Show charts

Search Only show running containers

Name	Image	Status	CPU (%)	Port(s)	Last started	Actions
elegant_murdock 059262c37e8	screambunn/jadn_sandbox:latest	Exited	0.04%	8082:8082	26 seconds ago	[More]
stupefied_rosalind 6fe0dec2accd	screambunn/jadn_sandbox:latest	Running	0%	8082:8082	0 seconds ago	[More]

Extensions Add Extensions



First, make sure to stop any running containers.

Docker Desktop Upgrade plan

Q Search: jadn Ctrl+K

Containers Give feedback

Container CPU usage ⓘ Container memory usage ⓘ Show charts ▾

No containers are running.

No containers are running.

Search Only show running containers

Name	Image	Status	CPU (%)	Port(s)	Last started	Actions
elegant_murdock 0592632c37e8	screambunn/jadn_sandbox:latest	Exited	N/A	8082:8082	2 minutes ago	[▶]
stupefied_rosalind 6fe0dec2accd	screambunn/jadn_sandbox:latest	Exited	N/A	8082:8082	4 seconds ago	[▶]

Containers Images Volumes Dev Environments BETA Docker Scout Learning center Extensions Add Extensions

The screenshot shows the Docker Desktop interface with the 'Containers' tab selected. At the top, there are sections for 'Container CPU usage' and 'Container memory usage', both stating 'No containers are running.' Below these is a search bar and a filter option 'Only show running containers'. The main area is a table listing containers. The first container, named 'elegant_murdock' with the ID '0592632c37e8', is highlighted with a red box. The 'Actions' column for this container also has a red box around its '▶' icon, which typically represents a terminal or play button. The second container, 'stupefied_rosalind' with the ID '6fe0dec2accd', is also listed. The table columns are: Name, Image, Status, CPU (%), Port(s), Last started, and Actions.

Click play on the desired container and then click on link to see terminal of running container.

elegant_murdock

Containers Images Volumes Dev Environments Docker Scout Learning center Extensions Add Extensions

Logs Inspect Bind mounts Exec Files Stats

0592632c3748 08082:8082

STATUS Running (32 seconds ago)

Logs

2023-12-21 09:36:10 Starting JADN Sandbox...
2023-12-21 09:36:10 2023-12-21 14:36:10 [1] [INFO] Registering Endpoint Blueprints
2023-12-21 09:36:10 The JADN Sandbox is ready for work!
2023-12-21 09:36:10 Go to the URL below in your browser:
2023-12-21 09:36:10 http://localhost:8082/
2023-12-21 09:38:58 Starting JADN Sandbox...
2023-12-21 09:38:58 2023-12-21 14:38:58 [1] [INFO] Registering Endpoint Blueprints
2023-12-21 09:38:58 The JADN Sandbox is ready for work!
2023-12-21 09:38:58 Go to the URL below in your browser:
2023-12-21 09:38:58 http://localhost:8082/
2023-12-21 09:41:35 Starting JADN Sandbox...
2023-12-21 09:41:35 2023-12-21 14:41:35 [1] [INFO] Registering Endpoint Blueprints
2023-12-21 09:41:35 The JADN Sandbox is ready for work!
2023-12-21 09:41:35 Go to the URL below in your browser:
2023-12-21 09:41:35 http://localhost:8082/

Click on the link provided to open the image in a browser window.

You can also enter the link itself in the browser:

<http://localhost:8082/>

How to Update the JADN Sandbox Image

[From Images](#)

[From Search Bar](#)

From Images

Docker Desktop Upgrade plan

Search: jadn Ctrl+K

Containers

Container CPU usage: No containers are running.

Container memory usage: No containers are running.

Show charts

Container ID: b6d3ebaf3fb0

Only show running containers

Name	Image	Status	CPU (%)	Port(s)	Last started	Actions
kind_coredump f5070dd74397	screambunn/jadn_sandbox:latest	Exited	N/A	8082:8082	31 seconds ago	Delete

OPTIONAL: Delete all containers related to the image (screambunn/jadn_sandbox:latest) by clicking the delete button (trash can icon) under Action.

Note: Containers use the image that it was created with. Even though the image is named the same, its versioning is different. Therefore older containers will be out of date.

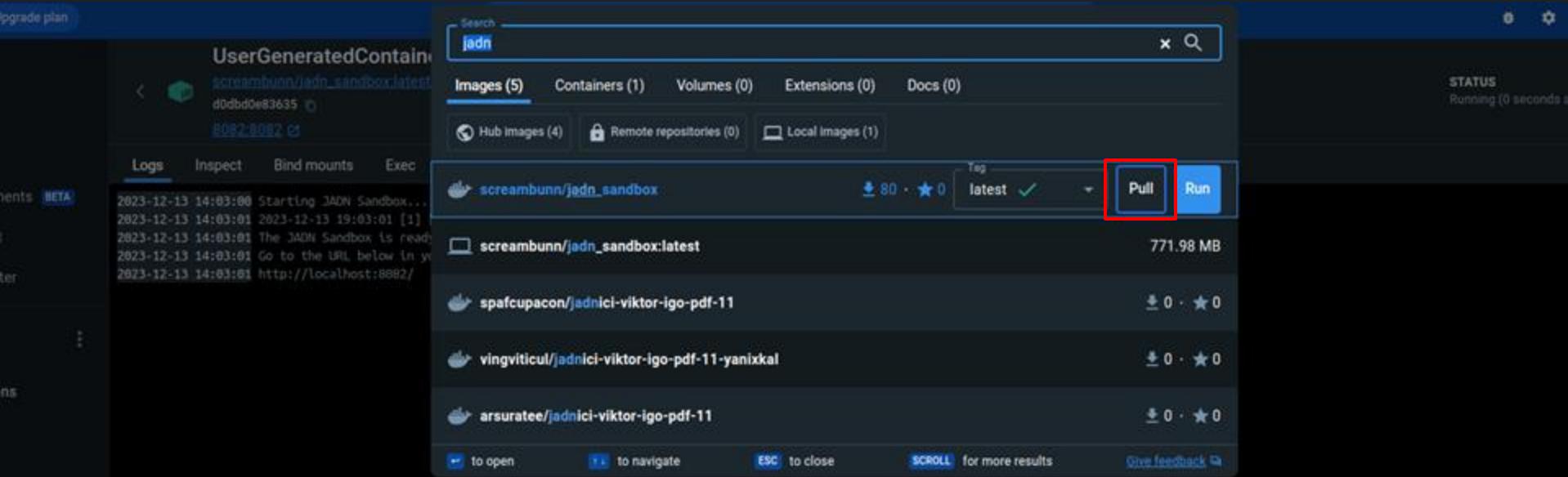
The screenshot shows the Docker Desktop interface. On the left sidebar, the 'Images' option is selected and highlighted with a red box. The main area displays the 'Images' tab with tabs for Local, Hub, Artifactory, and EARLY ACCESS. A search bar at the top says 'Search: jadn'. Below it, it shows '0 Bytes / 1.41 GB in use' and '1 Images'. A message 'Last refresh: 2 hours ago' is on the right. The table lists one image: 'screambunn/jadn_sandbox' with tag 'latest', status 'Unused', created '22 hours ago', and size '771.93 MB'. A context menu is open over this image, also highlighted with a red box. The menu items are: 'View packages and CVEs' (disabled), 'Pull' (selected and highlighted with a red box), and 'Push to Hub'.

Name	Tag	Status	Created	Size	Actions
screambunn/jadn_sandbox	latest	Unused	22 hours ago	771.93 MB	<ul style="list-style-type: none">View packages and CVEsPullPush to Hub

Pulling image..

Under Images, Go to Actions and click the 3 dotted icon.
A menu drop down will appear. Click Pull.
Then go through the [steps](#) to run the image.

From the Search Bar



In the search bar, search: jadn

Then, click Pull: screambunn/jadn_sandbox

How to Stop the JADN Sandbox Image

Docker Desktop Upgrade plan

Search for images, containers, volumes, extensions and more... Ctrl+K

Containers

Container CPU usage ⓘ Container memory usage ⓘ

No containers are running.

Show charts ▾

b6d3e6af3fb0

Only show running containers

Name	Image	Status	CPU (%)	Port(s)	Last started	Actions
kind_crd f5070dd74397 ⓘ	screambahn/adv_sandbox:latest	Running	N/A	8082:8082 ⓘ	0 seconds ago	Stop

Click the stop button under Action in Containers.

Containers

Images

Volumes

Dev Environments BETADocker Scout EARLY ACCESS

Learning center

Extensions

+ Add Extensions

sad_ritchie

screambunn/jadn_sandbox:latest

00bd8f6577d3

STATUS

Running (0 seconds ago)



Logs

Inspect

Bind mounts

Terminal

Files

Stats

```
2024-02-07 15:21:48 Starting JADN Sandbox...
2024-02-07 15:21:41 2024-02-07 20:21:41 [1] [INFO] Registering Endpoint Blueprints
2024-02-07 15:21:41 The JADN Sandbox is ready for work!
2024-02-07 15:21:41 Go to the URL below in your browser:
2024-02-07 15:21:41 http://localhost:8082/
2024-02-07 15:21:48 Starting JADN Sandbox...
2024-02-07 15:21:48 2024-02-07 20:21:48 [1] [INFO] Registering Endpoint Blueprints
2024-02-07 15:21:48 The JADN Sandbox is ready for work!
2024-02-07 15:21:48 Go to the URL below in your browser:
2024-02-07 15:21:48 http://localhost:8082/
2024-02-07 15:21:52 Starting JADN Sandbox...
2024-02-07 15:21:52 2024-02-07 20:21:52 [1] [INFO] Registering Endpoint Blueprints
2024-02-07 15:21:52 The JADN Sandbox is ready for work!
2024-02-07 15:21:52 Go to the URL below in your browser:
2024-02-07 15:21:52 http://localhost:8082/
```



Alternatively, Click the stop button within the running container

How to Learn More about Docker

- Containers
- Images
- Volumes
- Dev Environments BETA
- Docker Scout
- Learning center**
- Extensions
- [Add Extensions](#)

Learning center Give feedback

Walkthroughs

Quick hands-on guides to show you around

How do I run a container?
6 mins

Multi-container applications
6 mins

[View all](#)

AI/ML guides

Get started with AI/ML using Docker

GenAI Stack
Start your GenAI application using Neo4j, Langchain, Ollama, Python, and Docker Compose

Docker for beginners by language

45 min guides written for different programming languages

NodeJS

Python

Go

Java

C# (.NET)

Rust

[Request a guide](#)

Samples

Go to the Learning center to learn more about Docker.