Escher Documentation

Release 1.5.0

Zachary King

August 10, 2016

Escher is a web-based tool for building, viewing, and sharing visualizations of biological pathways. These 'pathway maps' are a great way to contextualize biological datasets. The easiest way to use Escher is to browse and build maps on the Escher website. New users may be interested in the Getting started guide.

Escher in 3 minutes

Help! I just upgraded to v1.2 and my map and model caches are empty!

Starting with Escher v1.2, the maps and models available from the Escher website are versioned. Each time you upgrade Escher, you will have access to the new versions of the Escher maps and models. Any maps and models you were using before are still saved in the Escher cache in case you need them. For more details on finding, managing, and clearing the cache, see *the Python cache functions*.

Also starting with Escher v1.2, the maps and models in Escher are the same as those in the BiGG Database.

Features

1. View pathway maps in any modern web browser

2. *Build maps* using the content of genome-scale metabolic models

3. *Visualize data* on reactions, genes, and metabolites

4. Full text search

5. Detailed options for changing colors, sizes, and more, all from the web browser

6. View maps inside the IPython Notebook

7. Embed maps within any website, with minimal dependencies (escher.js, d3.js, and optionally Twitter Bootstrap)

Supported browsers

We recommend using Google Chrome for optimal performance, but Escher will also run in the latest versions of Firefox, Internet Explorer, and Safari (including mobile Safari).

Installation

Escher can be used without any installation by visiting the Escher website. However, you can install escher if you would like to (1) run Escher offline, (2) include your own maps and models in the launch page, (3) view Escher maps in an IPython Notebook, or (4) modify the source code.

To install the latest stable version of Escher, run:

```
pip install escher
```

For more information, see the documentation on Run Escher locally and in IPython and Developing with Escher.

Citing Escher

You can help support Escher by citing our publication when you use Escher or EscherConverter:

Zachary A. King, Andreas Dräger, Ali Ebrahim, Nikolaus Sonnenschein, Nathan E. Lewis, and Bernhard O. Palsson (2015) *Escher: A web application for building, sharing, and embedding data-rich visualizations of biological pathways*, PLOS Computational Biology 11(8): e1004321. doi:10.1371/journal.pcbi.1004321

Contents

7.1 Getting started

7.1.1 Introduction

Escher is here to help you visualize pathway maps. But, if you have never heard of a pathway map, you might appreciate a quick introduction.

What are pathway maps?

To understand pathway maps, it is useful to think about the general organization of a cell. At the smallest level, molecules in a cell are arranged in three-dimensional structures, and these structures determine many of the functions that take place in a cell. For example, the 3D structure of an enzyme determines the biochemical reactions that it can catalyze. These structures can be visualized in 3D using tools like Jmol (as in this example structure).

The DNA sequence is a second fundamental level of biological organization. DNA sequences are the blueprints for all the machinery of the cell, and they can be visualized as a one-dimensional series of bases (ATCG) using tools like the UCSC genome browser.

To use a football analogy, the 3D molecular structures are akin to the players on the field, and the information in the DNA sequence is like the playbook on the sidelines. But, football would not be very interesting if the players never took to the field and executed those plays. So, we are missing this level of detail: *the execution of biological plans by the molecular players*.

What we are missing is the biochemical reaction network. Proteins in the cell catalyze the conversion of substrate molecules into product molecules, and these *reactions* are responsible for generating energy, constructing cellular machinery and structures, detecting molecules in the environment, signaling, and more. Biochemical reactions can be grouped into pathways when they work in concert to carry out a function. (If a reaction is a football play, then the pathway is a drive). And Escher can be used to visualize these reactions and pathways. Together, we call these visualizations **pathway maps**.

Escher to the rescue

Many Escher maps represent *metabolic* pathways, and Escher was developed at the Systems Biology Research Group where we have been building genome-scale models of metabolism over the past fifteen years. However, Escher is not limited to metabolism: It can be used to visualize any collection of biochemical reactions.

Escher includes one more killer feature: The ability to visualize datasets on a pathway map. Many biological discoveries are enabled by collecting and analyzing enormous datsets, and so biologists are grappling with the challenges of *big data*. By visualizing data in the context of pathway maps, we can quickly spot trends which would not be apparent

with standard statistical tools. And Escher visualizations can be adapted and shared to demonstrate those biological discoveries.

The rest of this guide will introduce the Escher user interface and the major features of Escher.

7.1.2 The launch page

When you open the Escher website, you will see a launch page that looks like this:

The options on the launch page are:

- **Filter by organism:** Choose an organism to filter the Maps and Models.

- **Map:** Choose a pre-built map, or start from scratch with an empty builder by choosing **None**. In parentheses next to the map name, you will see the name of the model that was used to build this map.

- **Model:** (Optional) If you choose a COBRA model to load, then you can add new reactions to the pathway map. You can also load your own model later, after you launch the tool. For an explanation of maps, models, and COBRA, see Escher, COBRA, and COBRApy.

- **Tool:**

  - The **Viewer** allows you to pan and zoom the map, and to visualize data for reactions, genes, and metabolites.

  - The **Builder**, in addition to the Viewer features, allows you to add reactions, move and rotate existing reactions, add text annotations, and adjust the map canvas.

- **Options:**

  - **Scroll to zoom (instead of scroll to pan)**: Determines the effect of using the mouse's scroll wheel over the map.

  - **Never ask before reloading**: If this is checked, then you will not be warned before leaving the page, even if you have unsaved changes.

Choose **Load map** to open the Escher viewer or builder in a new tab, and prepare to be delighted by your very own pathway map.
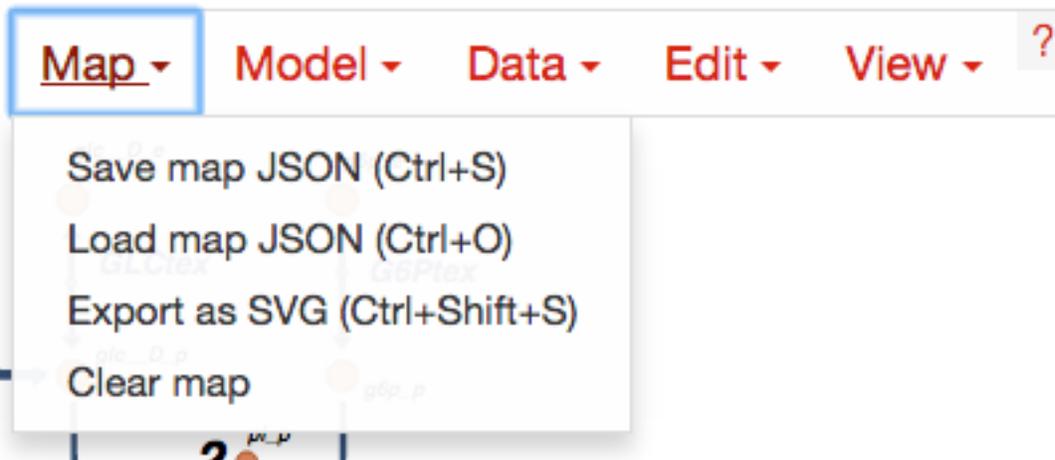
7.1.3 The menu bar

Once you have loaded an Escher map, you will see a menu bar along the top of the screen. Click the question mark to bring up the Escher documentation:

7.1.4 Loading and saving maps

Using the map menu, you can load and save maps at any time:

Click **Save map JSON** to save the Escher map as a JSON file, which is the standard file representing an Escher map.

NOTE: The JSON file does NOT save any datasets you have loaded. This may be changed in a future version of Escher.

NOTE 2: In Safari, saving files from Escher works a little differently than in the other major browsers. After clicking **Save map JSON**, Safari will load a new tab with the raw content of the Escher map file. To save the file, choose File>Save As... from the Safari menu. A new dialog appears; in the dropdown menu near the bottom, select Page Source (rather than Web Archive), give your file an appropriate name (e.g. `map.json` for an Escher map or `map.svg` for an SVG export), and click Save.

Later, you can can load a JSON file to view and edit a map by clicking **Load map JSON**.

Click **Export as SVG** to generate a SVG file for editing in tools like Adobe Illustrator and Inkscape. This is the best way to generate figures for presentations and publications. Unlike a JSON file, a SVG file maintains the data visualizations on the Escher map. However, you cannot load SVG files into Escher after you generate them.

Click **Clear Map** to empty the whole map, leaving a blank canvas. **NOTE**: You cannot undo **Clear Map**.

7.1.5 Loading models

Use the model menu to manage the COBRA model loaded in Escher:
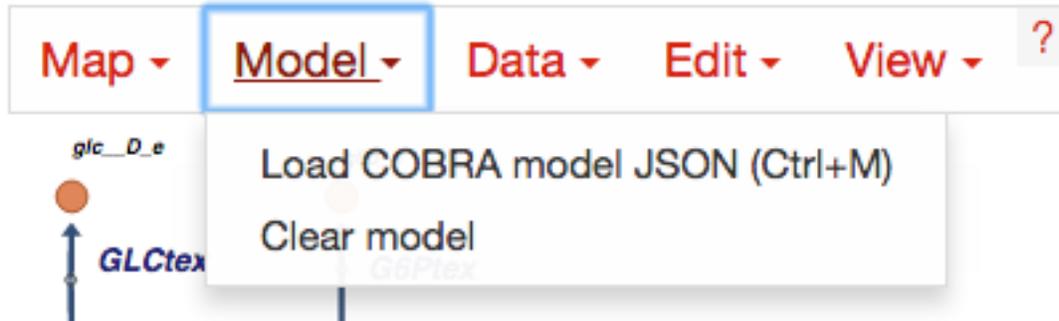
Choose **Load COBRA model JSON** to open a COBRA model. Read more about COBRA models in Escher, COBRA, and COBRApy. Once you have COBRApy v0.3.0 or later installed, then you can generate a JSON model by following this example code.

Once you have loaded a COBRA model, there may be inconsistencies between the content in the map and the model (e.g. reaction IDs, descriptive names and gene reaction rules). You click **Update names and gene reaction rules using model** to find matching reactions and metabolites between the map and the model (based on their IDs) and then apply the names and gene reaction rules from the model to the map. The reactions that do not match will be highlighted in red. (This can be turned off again in the settings menu by deselecting *Highlight reactions not in model*.) More advice on building maps is available in Building and contributing maps.

Click **Clear Model** to clear the current model.

7.1.6 Loading reaction, gene, and metabolite data

Datasets can be loaded as CSV files or JSON files, using the Data Menu.
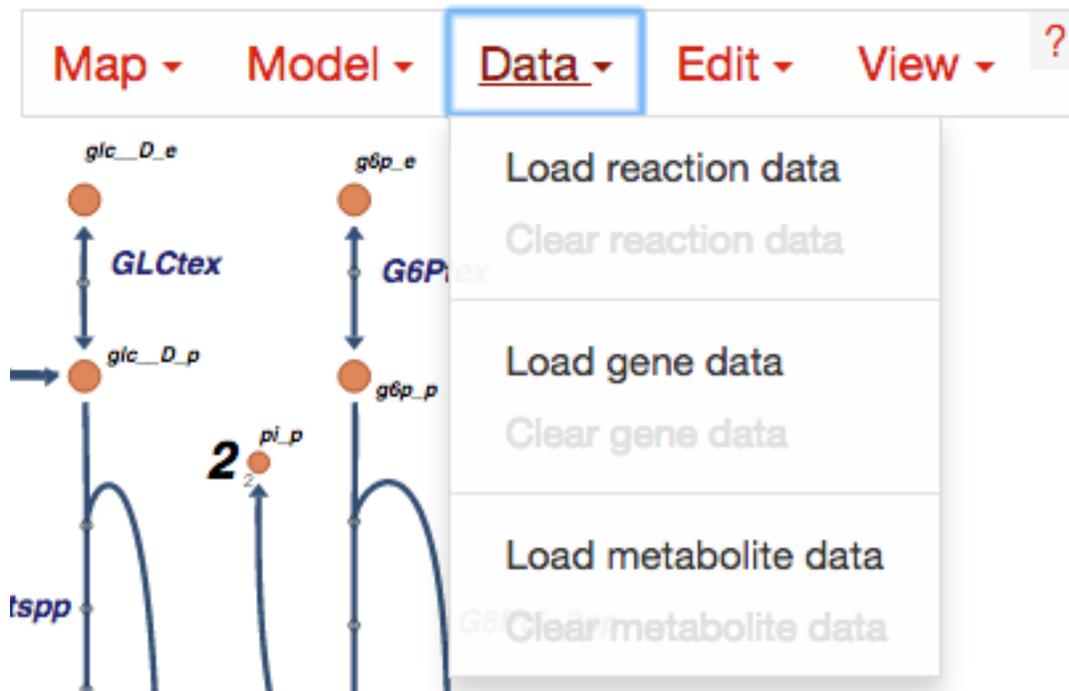
In Escher, reaction and gene datasets are visualized by changing the color, thickness, and labels of reaction arrows. Metabolite datasets are visualized by changing the color, size, and labels of metabolite circles. The specific visual styles can be modified in the *Settings* menu. When data is not present for a specific reaction, gene, or metabolite, then the text label will say 'nd' which means 'no data.'
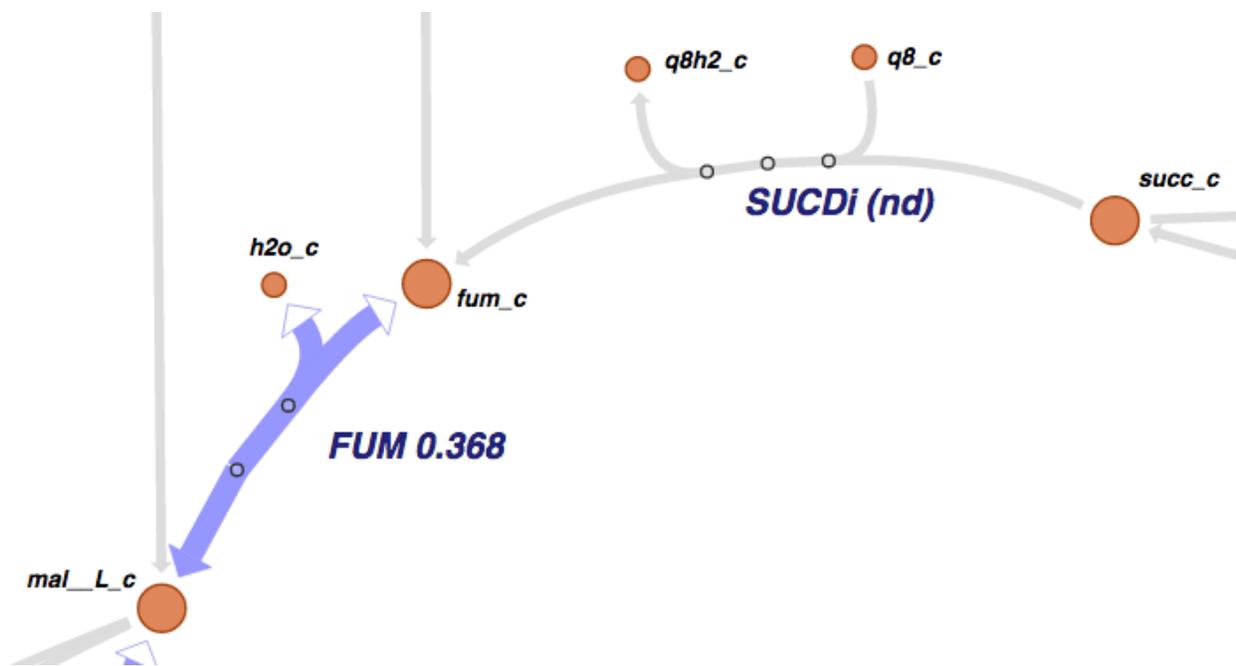
The structure of a CSV file

CSV files should have 1 header row, 1 ID column, and either 1 or 2 columns for data values. The ID column can contain BiGG IDs or descriptive names for the reactions, metabolites, or genes in the dataset. Here is an example with a single data value columns:

```
ID,time 0sec
glc__D_c,5.4
g6p__D_c,2.3
```

Which might look like this is Excel:

ID	time 0sec
glc__D_c	5.4
g6p_c	2.3

If two datasets are provided, then the Escher map will display the difference between the datasets. In the Settings menu, the **Comparison** setting allows you to choose between comparison functions (Fold Change, Log2(Fold Change), and Difference). With two datasets, the CSV file looks like this:

ID	time 0sec	time 5s
glc__D_c	5.4	10.2
g6p_c	2.3	8.1

Data can also be loaded from a JSON file. This Python code snippet provides an example of generating the proper format for single reaction data values and for reaction data comparisons:

```python
import json

# save a single flux vector as JSON
flux_dictionary = {'glc__D_c': 5.4, 'g6p_c': 2.3}
with open('out.json', 'w') as f:
    json.dump(flux_dictionary, f)

# save a flux comparison as JSON
```

```
flux_comp = [{'glc__D_c': 5.4, 'g6p_c': 2.3}, {'glc__D_c': 10.2, 'g6p_c': 8.1}]
with open('out_comp.json', 'w') as f:
    json.dump(flux_comp, f)
```

Gene data and gene reaction rules

Escher uses *gene reaction rules* to connect gene data to the reactions on a metabolic pathway. You can see these gene reaction rules on the map by selecting *Show gene reaction rules* in the *Settings* menu.

Gene reaction rules show the genes whose gene products are required to catalyze a reaction. Gene are connected using AND and OR rules. AND rules are used when two genes are required for enzymatic activity, e.g. they are members of a protein complex. OR rules are used when either gene can catalyze the enzymatic activity, e.g. they are isozymes.

With OR rules, Escher will take the sum of the data values for each gene. With AND rules, Escher will either take the mean (the default) or the minimum of the components. The AND behavior (mean vs. minimum) is defined in the *Settings* menu.

7.1.7 Editing and building

The Edit menu gives you access to function for editing the map:
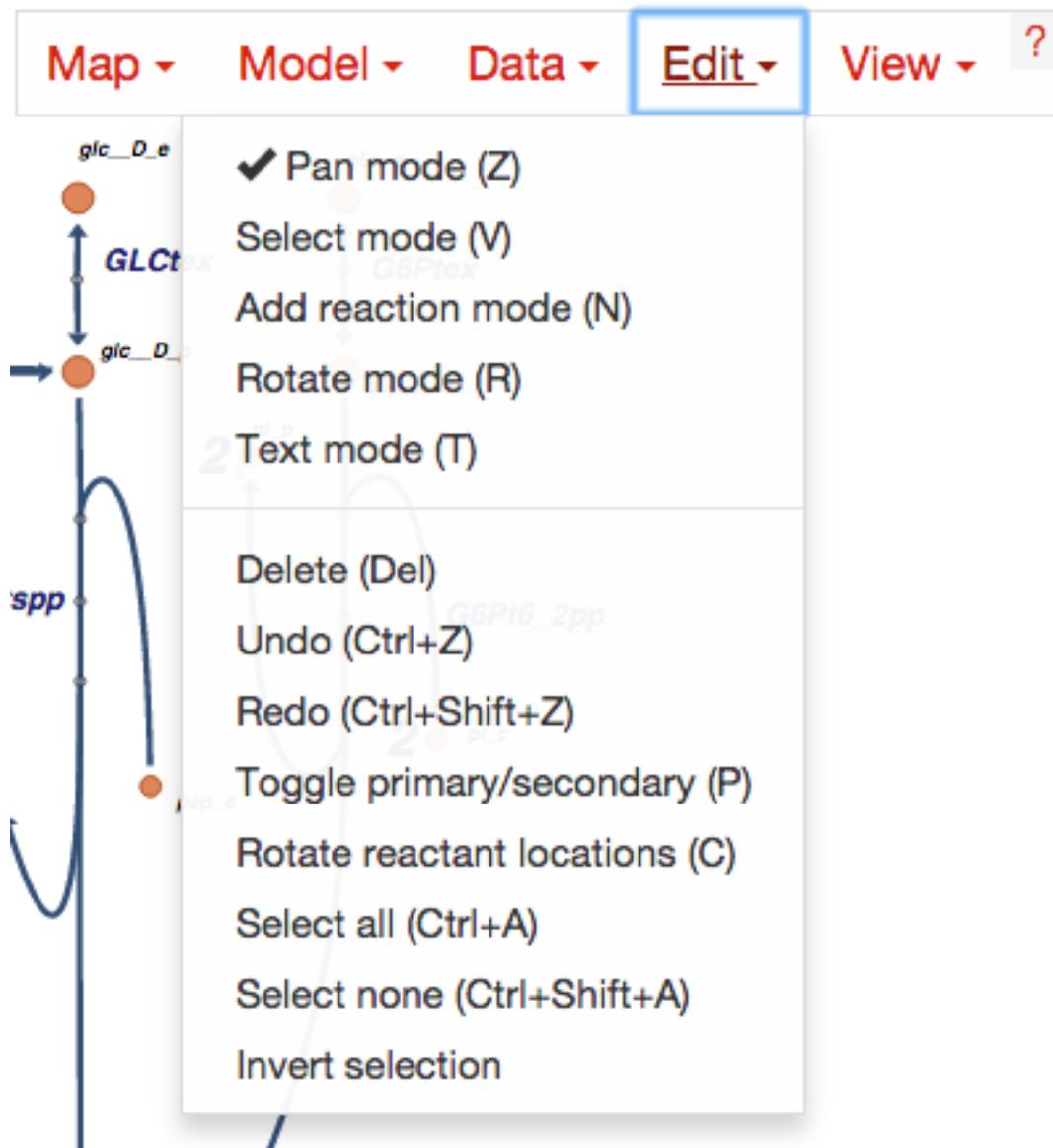
Escher has five major modes, and you can switch between those modes using the buttons in the Edit menu, or using the buttons in the *button bar* on the left of the screen.

1. **Pan mode:** Drag the canvas to pan the map. You can also use the mouse scroll wheel or trackpad scroll function (drag with 2 fingers) to pan the map (or to zoom if you selected **Scroll to zoom** in the settings).

2. **Select mode:** Select nodes by clicking on them. Shift-click to select multiple nodes, or drag across the canvas to select multiple nodes using the selection brush.

3. **Add reaction mode**: If you have loaded a Model, then click on the canvas to see a list of reactions that you can draw on the map. Click on a node to see reactions that connect to that node. In the input box, you can search by reaction ID, metabolite ID, or gene ID (locus tag).

4. **Rotate model**: Before entering rotate mode, be sure to select one or more nodes in select mode. Then, after entering rotate mode, drag anywhere on the canvas to rotate the selection. You can also drag the red crosshairs to change the center of the rotation.

5. **Text mode**: Use text mode to add text annotations to the map. Click on the canvas to add a new text annotation, or click an existing annotation to edit it. When you are finished, click Enter or Escape to save the changes.

In addition to the editing modes, the Edit menu gives you access to the following commands:

- **Delete:** Delete the currently selected node(s).

- **Undo**: Undo the last action. NOTE: Certain actions, such as editing the canvas, cannot be undone in the current version of Escher.

- **Redo**: Redo the last action that was undone.

- **Toggle primary/secondary node**: In Escher, each metabolite node is either a primary node or a secondary node. Primary nodes are larger, and secondary nodes can be hidden in the Settings menu. Use this command to toggle the currently selected node(s) between primary and secondary.

- **Rotate reactant locations**: When you draw a new reaction in Escher, this command will rotate the new reactants so that a new reactant is primary and centered. This command is extremely useful when you are drawing a long pathway and you want to quickly switch which metabolite to "follow", e.g. make sure you are following the carbon-containing metabolites.

  If you are confused, then try drawing a new pathway and hitting the "c" key to see the reactants rotate.

- **Select all:** Select all nodes.

- **Select none:** Unselect all nodes.

- **Invert selection**: Select all the nodes that are currently unselected. This feature is very useful when you want to keep just one part of the map. Simply drag to select the part you want to keep, call the **Invert selection** command, then call the **Delete** command.

7.1.8 Editing the canvas

A somewhat non-obvious feature of Escher is that you can edit the canvas by dragging the canvas edges. This is possible in Pan mode and Select mode.

7.1.9 View options
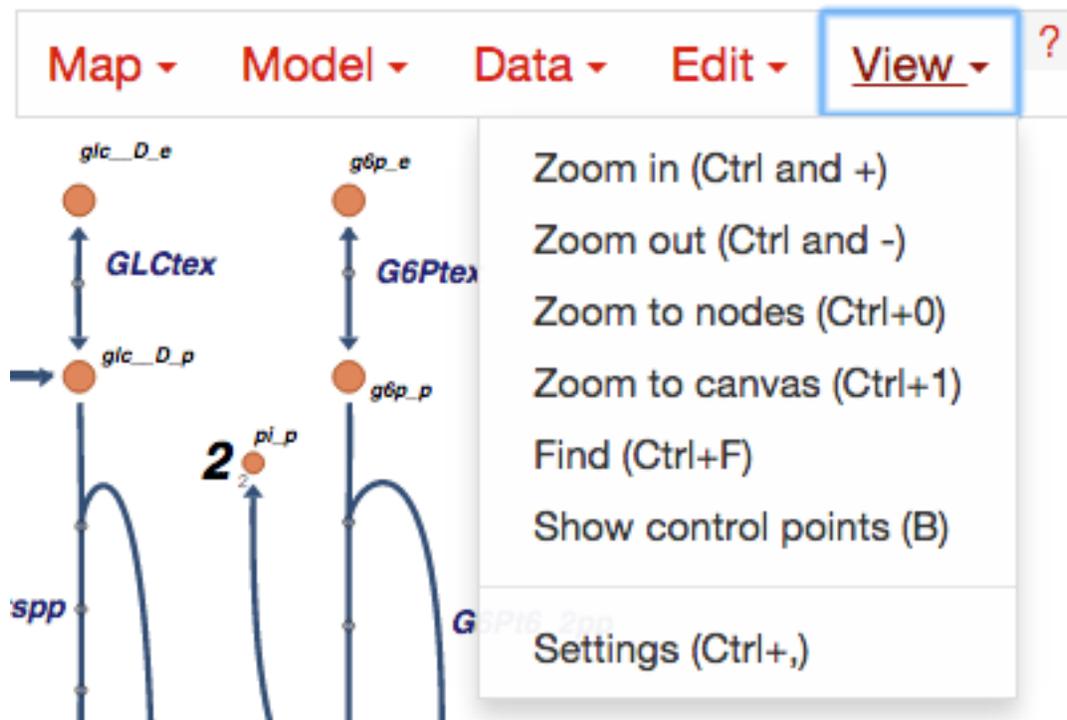
- **Zoom in:** Zoom in to the map.
- **Zoom out:** Zoom out of the map.
- **Zoom to nodes:** Zoom to see all the nodes on the map.
- **Zoom to canvas:** Zoom to see the entire canvas.
- **Find:** Search for a reaction, metabolite, or gene on the map.
- **Show control points:** Show the control points; you can drag these red and blue circle to adjust the shapes of the reactions curves.
- **Settings:** Open the *Settings* menu.

7.1.10 The button bar

The button bar give you quick access to many of the common Escher functions:

— Zoom in (Ctrl and +)

— Zoom out (Ctrl and -)

— Zoom to canvas (Ctrl+0)

— Pan mode (Z)

— Select mode (V)

— Add reaction mode (N)

— Rotate mode (R)

— Text mode (T)

7.1.11 Settings

View and build options

- **Identifiers:** Choose whether to show BiGG IDs or descriptive names for reactions, metabolites, and genes.

- **Hide secondary metabolites:** This will simplify the map by hiding all secondary metabolites and the paths connected to them.

- **Show gene reaction rules:** Show the gene reaction rules below the reaction labels, even gene data is not loaded.

- **Hide reaction, gene, and metabolite labels:** Another option to visually simplify the map, this will hide all labels that are not text annotations.

- **Allow duplicate reactions:** By default, duplicate reactions are hidden in the add reaction dropdown menu. However, you can turn this option on to show the duplicate reactions.

- **Highlight reactions not in model:** Highlight in red any reactions that are on the map but are not in the model. This is useful when you are adapting a map from one model for use with another model

Reaction data settings

When reaction or gene data is loaded, this section can be used to change visual settings for reactions.

The color bar has individual *control points*, and you can drag the control points (execpt min and max) left and right to change their values. Alternatively, you can use the dropdown menu (next to the word *median* in the figure above), to attach a control point to a statistical measure (mean, median, first quartile (Q1), or third quartile (Q3)). This lets you choose a color scale that will adapt to your dataset.

For each control point, you can choose a color by entering a CSS-style color (e.g. red, #ff0000, rgba(20, 20, 255, 0.5), and you can choose a size that will scale the thickness of reactions.

There are also color and size options for reactions that do not have any data value.

Finally, there are a few on/off settings for the loaded reaction or gene dataset:

- **Options:** These determine how to visualize the datasets. Check *Absolute value* to color and size reactions by the absolute value of each data value. The *Size*, *Color*, and *Text* options can be unselected to turn off sizing, coloring, and data values in reaction labels individually.

- **Comparison:** Determines the comparison algorithm to use when two datasets are loaded.

- **Method for evaluating AND:** Determines the method that will be used to reconcile AND statements in gene reaction rules when there is gene data loaded. See *Gene data and gene reaction rules* for more details.

Metabolite data settings

The data settings for metabolite data are analagous to those for reaction data. The only difference is that *size* now refers to the size of the metabolite circles.

7.2 Escher, COBRA, and COBRApy

Escher can be used as an independent application, but it draws heavily on the information in COBRA models. A COBRA model is a collection of all the reactions, metabolites, and genes known to exist in an organism. (In the literature, these are generally called genome-scale models (GEMs), but we refer to them as COBRA models here to emphasize that Escher interoperates with models that are exported from COBRApy.)

By loading a COBRA model into the Escher interface, you have access to every reaction and metabolite in that model. You also have the *gene reaction rules* for the reactions in the network, which allow you to connect gene data to reactions and metabolites.

COBRApy is a software package for COBRA modeling written in Python. The Escher Python package uses CO-BRApy package for reading and writing COBRA models.

7.2.1 Maps and models

In Escher, you will see references to *maps* and *models*.

A map contains the reactions and metabolites that you see in the Escher builder, including their locations, text annotations, and the canvas.

A model (a COBRA model) contains reactions and metabolites that you have not drawn yet. Thus, you can load a COBRA model when you want to draw new reactions on the map.

7.2.2 What is JSON and why do we use it?

Both Escher maps and COBRA models are stored as JSON files. JSON is a useful, plain-text format for storing nested data structures. We use JSON much like the SBML community uses XML. You may notice that SBML files have a .xml extension, and Escher maps and COBRA models have a .json extension.

You can use Python to explore a JSON file like this:

```
import json

with open('map.json', 'r') as f:
    map_object = json.load(f)

print map_object[0]
print map_object[1]['reactions'].values()[0]
```

7.2.3 Escher, SBML, and SBGN

A tool has been developed for converting Escher maps to SBML Layout and SBGN, and it will be released soon.

COBRA models can be converted to SBML using COBRApy.

7.3 Run Escher locally and in IPython

To run Escher on a local computer or to use Escher in Python/IPython, first install it. The Python package for Escher can be installed using pip:

```
pip install escher
```

Depending on your installation of Python, you may need sudo:

```
sudo pip install escher
```

Alternatively, one can download the source files and install the package directly:

```
python setup.py install
```

7.3.1 Escher in the IPython Notebook

Once you have installed Escher locally, you can interact with Escher maps in an IPython Notebook.

Here are example notebooks to get started with:

- COBRApy and Escher
- JavaScript development and offline maps
- Generate JSON models in COBRApy

Dependencies:

- Jinja2
- Tornado
- COBRApy, 0.3.0 or later

7.3.2 Running the local server

You can run your own local server if you want to use Escher offline or explore your own maps with the homepage browser. To get started, install the Python package and run from any directory by calling:

```
python -m escher.server
```

This starts a server at http://localhost:7778. You can also choose another port:

```
python -m escher.server --port=8005
```

7.4 Validate and convert maps

7.4.1 Validate an Escher map

Escher maps follow a specification using the JSON Schema format. Therefore, any JSON Schema validator can be used to validate an Escher map by comparing it to the latest schema file. For example, the current schema is located here:

https://github.com/zakandrewking/escher/blob/master/escher/jsonschema/1-0-0

To make this easier, the Escher Python package includes a validation script. To validate a map, first install Escher:

```
pip install escher
```

Then call the validator:

```
python -m escher.validate my_map.json
```

Any errors in the map will print to the console.

7.4.2 Convert or upgrade an Escher map

Any Escher maps built with pre-release versions of Escher will not load right away in the stable v1.0 release. To convert pre-release maps to the new format, follow these steps:

1. Install Escher:

```
pip install escher
```

2. Find a COBRA model for your maps. This COBRA model will be used to update the content of the map in order to support all the new Escher features. You can use a COBRA model encoded as SBML or JSON (generated with COBRApy v0.3.0b4 or later). The COBRA models currently available on the Escher website can be downloaded from the BiGG Models website:

   http://bigg.ucsd.edu

   For a refresher on the distinction between Escher maps, COBRA models, and their file types (SBML, JSON, SBML Layout), see Escher, COBRA, and COBRApy.

3. Run the `convert_map` script to convert your existing Escher map (my_old_map.json) to the new format, using a COBRA model (model_file.json or model_file.xml in these examples):

```
# With a JSON file model
python -m escher.convert_map my_old_map.json path/to/model_file.json

# With an SBML model
python -m escher.convert_map my_old_map.json path/to/model_file.xml
```

Those commands will generate a new map called `my_old_map_converted.json` that will load in Escher v1.0 and later.

7.5 Building and contributing maps

We are excited to collect pathway maps for every organism with a well characterized metabolic network. This section describes the process of building a new map, either from scratch or using a COBRA model with BiGG IDs.

7.5.1 Building from scratch

To build a map from scratch, you will first need a COBRA model for your map. See the section Escher, COBRA, and COBRApy for some background information on COBRA models.

If you would like to eventually contribute your map to the Escher website, it is important that your COBRA model adheres to the identifiers in the BiGG Database. Escher and BiGG are being developed together, and we want to maintain consistency and interoperability between them.

Once you have a COBRA model, you can follow these steps:

1. Load your model in the Escher Builder.

2. Begin building new reactions. If you are familiar with the genes in your organism, then try search for new reactions by their gene IDs.

3. Limit each map to ~200 reactions. Maps larger than this will slow down the Escher viewer, especially on old browsers. Rather than building one giant map, Escher is designed for building many, smaller subsystem maps.

4. When you have built a map for your a subsystem, save the map as JSON with a name that includes the model ID, followed by a period, followed by the name of the subsystem. For example:

```
iMM904.Amino acid biosynthesis.json
```

5. (Optional) Once you have a set of subsystem maps, you can set up a local Escher server so that subsystem maps appear in the "quick jump" menu in the bottom right corner of the screen (as seen here for iJO1366). To set this up, you will need to start a local server as describe in *Running the local server*. Next, find your local cache directory by running this command in a terminal:

```
python -c "import escher; print(escher.get_cache_dir(name='maps'))"
```

This will print the location of the local maps cache. Add your new subsystem maps to cache folder. Now, when you run the server (described in *Running the local server*), you should see that quick jump menu appear.

NOTE: The cache directory is organized into folders for organisms. You can use these folder for filtering by organism on the local launch page, or you can place the maps in the top directory.

NOTE 2: A similar approach can be used to access your models from the local launch page. Place maps in the folder indicated by:

```
python -c "import escher; print(escher.get_cache_dir(name='models'))"
```

7.5.2 Building from an existing map for a similar organism

Follow the instruction above, except, rather than starting from scratch, load an existing Escher map for a different organism.

Once you have the new model loaded, use the **Update names and gene reaction rules using model** button in the Model menu to convert all descriptive names and gene reaction rules in the model to those in the map. Reactions that do not match the model will be highlighted in red. (This can be turned off again in the settings menu by deselecting *Highlight reactions not in model*.)

Now, visit each highlighted reaction and see if you can replace it with an equivalent biochemical pathway from the model. If not, then delete the reaction and move on.

Finally, when there are no highlighted reactions left, you can repeat this for other subsystems.

7.5.3 Submitting maps to the Escher website

If you would like to contribute maps to Escher, you can make a Pull Request to the GitHub repository escher.github.io. Make sure there is a folder with the name of the organism in `1-0-0/maps`. For example, a new yeast map goes in the folder:

```
1-0-0/maps/Saccharomyces cerevisiae/
```

Then, name your map by concatenating the model ID and the map name, separated by a period. For example, a yeast map built with the genome-scale model iMM904 could be named:

```
iMM904.Amino acid biosynthesis.json
```

Then, add the JSON file for the model to the Pull Request *if that model is not already available*. As before, make a folder for your organism within `1-0-0/models/`. The model filename is just the model ID.

In this example, a correct Pull Request would include the following files:

```
1-0-0/maps/Saccharomyces cerevisiae/iMM904.Amino acid biosynthesis.json
1-0-0/models/Saccharomyces cerevisiae/iMM904.json
```

7.6 Developing with Escher

7.6.1 Using the static JavaScript and CSS files

You can include the compiled Escher JavaScript and CSS files in any HTML document. The only dependencies are d3.js, and optionally Twitter Bootstrap if you are using the option menu='all'.

The compiled files are available form npmcdn:

```
https://npmcdn.com/escher-vis/js/dist/escher.js
https://npmcdn.com/escher-vis/js/dist/escher.min.js
https://npmcdn.com/escher-vis/css/dist/builder.css
https://npmcdn.com/escher-vis/css/dist/builder.min.css
```

Source maps are also hosted there:

```
https://npmcdn.com/escher-vis/js/dist/escher.js.map
https://npmcdn.com/escher-vis/js/dist/escher.min.js.map
https://npmcdn.com/escher-vis/css/dist/builder.min.css.map
```

If you want a particular version of escher, add a version tag like this:

```
https://npmcdn.com/escher-vis@1.4.0-beta.3/js/dist/escher.js
```

For an example of the boilerplate code that is required to begin developing with Escher, have a look at the escher-demo repository. For projects built with npm, use the escher-test repository as a guide.

7.6.2 Building and testing Escher

First, install dependencies with npm:

```
npm install
```

Escher uses grunt to manage the build process. To run typical build steps, just run:

```
npm run compile
```

To test the JavaScript files, run:

```
npm run test
```

For Python testing, run this in the py directory:

```
python setup.py test
```

Build the static website:

```
python setup.py build_gh
```

Clear static website files:

```
python setup.py clean
```

Build and run the docs:

```
cd docs
make html
cd _build/html
```

```
python -m SimpleHTTPServer # python 2
python -m http.server # python 3
```

7.6.3 Generating and reading Escher and COBRA files

The Escher file format

Escher layouts are defined by JSON files that follow a specific schema, using json schema. The latest schema for Escher JSON files is here. The Escher schemas are versioned, with inspiration from SchemaVer. The `escher.validate` module can be used to validate models against the schema.

The Escher layout schema is designed to be as simple as possible. For example, the core metabolism map of Escherichia coli is layed out like this:

```
[
    {
        "map_name": "E coli core.Core metabolism",
        "map_id": "2938hoq32a1",
        "map_description": "E. coli core metabolic network\nLast Modified Fri Dec 05 2014 16:39:44 GM
        "homepage": "https://escher.github.io",
        "schema": "https://escher.github.io/escher/jsonschema/1-0-0#"
    },
    {
        "reactions": { ... },
        "nodes": { ... },
        "text_label": { ... },
        "canvas": {
            "x": 7.857062530517567,
            "y": 314.36893920898433,
            "width": 5894.515691375733,
            "height": 4860.457037353515
        }
    },
]
```

The `map_name` includes the model that was used to build this layout, followed by a period and then a readable name for the map. The `map_id` is a unique identifier for this map. The `map_description` describes the map and the last time it was modified. Both the `homepage` and the `schema` entries must have exactly these values for the Escher map to be valid.

In the next section, the reactions, nodes, labels, and canvas are all defined. For reactions, nodes, and text labels, each element has a key that is an arbitrary integer. As long as there are no repeated IDs (e.g. no 2 segments with the ID 517), then everything should work fine.

Read through the schema (here) for more detail on the format.

The COBRA file format

COBRA models are also saved as JSON files. This format has not been documented with a schema, but you can browse through the core metabolism model as a guide to generating valid COBRA models.

I still need help!

If you are interested in developing with Escher and you need more information than what is provided in the documentation, please contact Zachary King (zaking-AT-ucsd-DOT-edu).

7.7 EscherConverter

EscherConverter is a standalone program that reads files created with the graphical network editor Escher and converts them to files in community standard formats.

Download EscherConverter 0.5 (13.7 MB).

7.7.1 Using EscherConverter

Escher uses a JSON file format to represent its networks. This format has been developed because of its very small file size and its compatibility to online programs that are written in JavaScript. In particular, JSON is a JavaScript Object Notation, or in other words, a JSON file directly represents components of JavaScript programs. This makes parsing very simple and allows direct use of those files in web-based programs.

However, in systems biology, specific file formats have been developed with the aim to be easily exchangeable between software implemented in diverse programming languages.

To this end, these formats support semantically clear annotations and are maintained by a large community of scientists. EscherConverter supports export to two particularly important XML-based community file formats SBML with layout extension and SBGN-ML.

While SBML has been mainly developed for dynamic simulation of biological networks, it is nowadays also usable for diverse other purposes. Its layout extension facilitates the encoding the display of biological networks.

SBGN-ML has been directly developed as a language for the display of biological pathway maps of diverse kinds. It stores the position and connection of entities, similar to what is shown in Escher networks.

EscherConverter takes Escher's JSON files as input and generates equivalent SBML Level 3 Version 1 files with layout extension or SBGN-ML files.

In order to ensure that the conversion is correct, EscherConverter provides its own display that gives users a preview of how the export data format will be rendered by other tools. In this preview display, you can zoom in and out, move arcs and node positions. However, it is important to know that none of the changes made in this preview are stored in the export file.

Download and Installation

You can obtain local copy of EscherConverter by clicking here.

As a Java™ application, no specific installation is required for EscherConverter.

However, make sure you have a recent Java™ Runtime Environment (JRE) installed on your computer (at least JRE 8). You can obtain Java™ from the Oracle website. There you can also find installation instructions for your respective operating system.

Once Java™ has been installed, you can simply place the EscherConverter JAR file somewhere on your local computer, for instance in the folder

- `/Applications/` if you are working under Mac OS

- `/opt/` for Linux computers
- `C:\Program Files\` if you are using Windows

Launching the program

You can launch EscherConverter simply by double-clicking on the application JAR file. This will open a following graphical user interface as described in the following figures.

Figure 1 | The graphical user interface of EscherConverter.

- Preferences: opens a settings dialog (see next screenshot).
- Open: displays a file chooser to select an input file in JSON format.
- Save: export the map that is displayed in the current tab to SBML or SBGN-ML.
- Discard: closes the current tab without saving.
- Help: opens the online help that displays all command-line options.
- License: displays the license under which this software is distributed.
- About: shows information about the authors of this software.
- Main: the main panel of the software, in which converted Escher maps will be displayed, organized in tabs.
- Status: the status bar shows information and logging messages during the conversion.
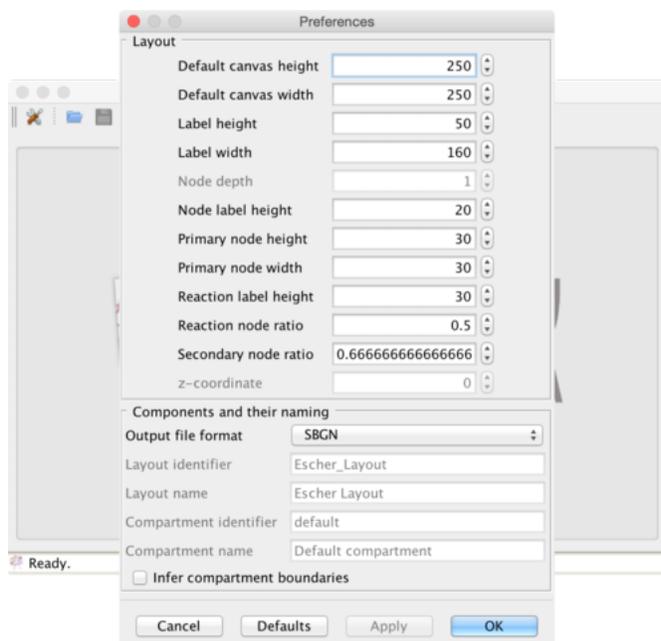
Figure 2 | The preferences dialog. All settings in this dialog are also available through EscherConverter's command-line interface.

Layout section: several properties that are required in SBML and SBGN-ML are not explicitly specified in Escher's JSON format and therefore need to be adjusted by the user. These are the size of the drawing canvas, the default box size for text labels, the size of primary and secondary nodes. In contrast to SBGN-ML and Escher's JSON format, the SBML layout extension supports three-dimensional displays. When selecting this export file format, it is therefore necessary to also define the z-coordinate and the depth of all objects.

Components and naming section: Most settings in this section are specific for the SBML layout extension and are only active if this output format is selected. These are names and identifiers of the layout component and the, in case of SBML, mandatory default compartment (a compartment without physical meaning, a reaction space in which all metabolites are located). EscherConverter can infer compartment bounds from metabolites, but this is an experimental feature. Just try it and see how the preview changes.
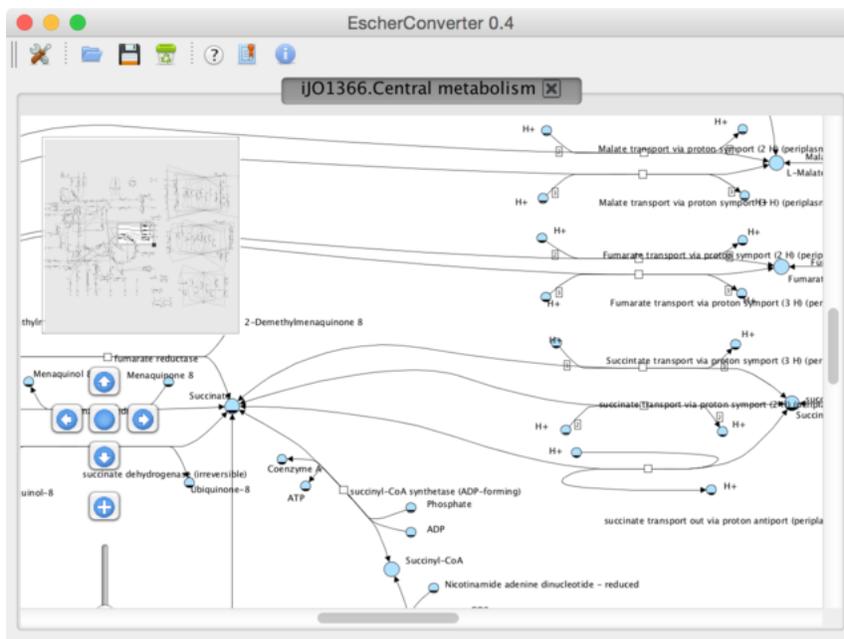
Figure 3 | A layout preview.

The birds-eye view and the navigation component on the left help you navigate through the network display. This graph shows you how the converted pathway map will be rendered by software that supports SBGN-ML or the SBML layout extension. This preview can be modified by moving arcs or nodes, however, none of those changes are stored when exporting the file. In this current version, EscherConver only exports layouts as given in the original JSON file.

Included third-party software

EscherConverter includes several third-party libraries, which we here list and acknowledge:

- ArgParser
- JSBML
- libSBGN
- Pixel-Mixer icons
- yFiles

7.7.2 Command-line interface API

You can launch EscherConverter from the command-line. On a Unix system (such as Linux, MacOS, or Solaris, etc.), use a command like:

```
bash$ java -jar -Xms8G -Xmx8G -Duser.language=en ./EscherConverter-0.5.jar [options]
```

Under Window, use a command like:

```
C:\> javaw -jar -Xms8G -Xmx8G -Duser.language=en EscherConverter-0.5.jar [options]
```

Escher has a large collection of command-line options (see below), which can be useful if you want to launch the program with specific settings or if multiple files are to be converted in a batch mode. It is even possible to completely disable the graphical user interface.

Input and output files

Definition of input and output data files as well as the format for the output.

```
--input=<File>
```

Specifies the JSON input file. If a directory is given, the conversion will be recursively performed. Accepts JSON.

```
--output=<File>
```

The path to the file into which the output should be written. If the input is a directory, this must also be a directory in order to perform a recursive conversion. Accepts SBML, SBGN.

Escher layout

The options in this group allow you to influence how large certain elements should be displayed.

```
--canvas-default-height=<Double>
```

Just as in the case of the width of the canvas, this value needs to be specified for cases where the JSON input file lacks an explicit specification of the canvas height. Arguments must fit into the range {[1,1E9]}. (Default value: 250.0)

```
--canvas-default-width=<Double>
```

This value is used when no width has been defined for the canvas. Since the width attribute is mandatory for the layout, a default value must be provided in these cases. Arguments must fit into the range {[1,1E9]}. (Default value: 250.0)

```
--label-height=<Double>
```

With this option you can specify the height of the bounding box of text labels. Arguments must fit into the range {[1,1E9]}. (Default value: 50.0)

```
--label-width=<Double>
```

This option defines the width of bounding boxes for text labels. Arguments must fit into the range {[1,1E9]}. (Default value: 160.0)

```
--node-depth=<Double>
```

The length of nodes along z-coordinate. Escher maps are actually two-dimensional, but in general, a layout can be three-dimensional. This value should be an arbitrary value greater than zero, because some rendering engines might not display the node if its depth is zero. Arguments must fit into the range {[1,1E9]}. (Default value: 1.0)

```
--node-label-height=<Double>
```

Node labels can have a size different from general labels in the graph. Here you can specify how height the bounding box of the labels for nodes should be. Arguments must fit into the range {[1,1E9]}. (Default value: 20.0)

```
--primary-node-height=<Double>
```

The primary node should be bigger than the secondary node. With this option you can specify the height of this type of nodes. Arguments must fit into the range {[1,1E9]}. (Default value: 30.0)

```
--primary-node-width=<Double>
```

Escher maps distinguish between primary and secondary nodes. Primary nodes should be larger than secondary nodes and display the main flow of matter through the network. This option allows you to specify the width of primary nodes. Arguments must fit into the range {[1,1E9]}. (Default value: 30.0)

```
--reaction-label-height=<Double>
```

Reaction label heightArguments must fit into the range {[1,1E9]}. (Default value: 30.0)

```
--reaction-node-ratio=<Double>
```

This value is used as a conversion factor to determine the size of the reaction display box depending on the size of primary nodes. Height and width of reaction nodes are determined by dividing the corresponding values from the primary node size by this factor. Arguments must fit into the range {[0,1]}. (Default value: 0.5)

```
--secondary-node-ratio=<Double>
```

Similar to the reaction node ratio, the size of secondary nodes (width and height) is determined by dividing the corresponding values from the primary nodes by this value. Arguments must fit into the range {[0,1]}. (Default value: 0.6666666666666666)

```
--z=<Double>
```

The position on the z-axis where the entire two-dimensional graph should be drawn. Arguments must fit into the range {[-1E9,1E9]}. (Default value: 0.0)

Escher Components and their naming

Here you can influence, which elements are to be included in the layout and how elements in the layout are to be called or identified.

```
--format=<OutputFormat>
```

The desired format for the conversion, e.g., SBML. All possible values for type <OutputFormat> are: SBGN and SBML. (Default value: SBML)

```
--layout-id=<String>
```

In contrast to the name, this identifier does not have to be human-readable. This is a machine identifier, which must start with a letter or underscore and can only contain ASCII characters. (Default value: Escher_Layout)

```
--layout-name=<String>
```

This should be a human-readable name for the layout that is to be created. This name might be displayed to describe the figure and should therefore be explanatory. (Default value: Escher Layout)

```
--compartment-id=<String>
```

A compartment needs to have a unique identifier, which needs to be a machine-readable Sting that must start with a letter or underscore and can only contain ASCII characters. Since the JSON file does not provide this information, this option allows you to specify the required identifier. (Default value: default)

```
--compartment-name=<String>
```

With this option it is possible to define a name for the default compartment can be that needs to be generated for the conversion to SBML. The name does not have any restrictions, i.e., any UTF-8 character can be used. (Default value: Default compartment)

```
--infer-compartment-bounds
```

This converter can infer where the boundaries of compartments could be drawn. To this end, it uses each node's BiGG ids to identify the compartment of all metabolites. Assuming that compartments have rectangular shapes, the algorithm can find the outermost node on each side of the box and hence obtain the boundaries of the compartment. However,

this methods will fail when metabolites are drawn inside of such a box that belong to a different compartment that is actually further outside. For this reason, this option is deactivated by default. (Default value: false)

Options for the graphical user interface

```
--gui
```

If this option is given, the program will display its graphical user interface. (Default value: false)

```
--log-level=<String>
```

Change the log-level of this application. This option will influence how fine-grained error and other log messages will be that you receive while executing this program. All possible values for type <String> are: ALL, CONFIG, FINE, FINER, FINEST, INFO, OFF, SEVERE, and WARNING. (Default value: INFO)

```
--log-file=<File>
```

This option allows you to specify a log file to which all information of the program will be written. Accepts (*.bp2, *.bp3, *.log, *.owl, *.xml).

7.8 JavaScript API

class escher.**Builder**(*map_data*, *model_data*, *embedded_css*, *selection*, *options*)
    A Builder object contains all the UI and logic to generate a map builder or viewer.

        Arguments

- **map_data** (*object*) – The data for an Escher map layout. Optional: Pass null to load an empty Builder.

- **model_data** (*object*) – The data for a Cobra model that will be used with the Add Reaction tool to build a layout. Optional: Pass null to load the Builder without a model.

- **embedded_css** (*string*) – The stylesheet for the SVG elements in the Escher map. Optional: Pass null to use the default style.

- **selection** (*object*) – The D3 selection of a HTML element that will hold the Builder. (The selection cannot be a SVG element.) Optional: Pass null to load the Builder in the HTML body.

- **options** (*object*) –

  An object defining any of the following options. Optional: Pass null to use all default options.

  options.**menu**
      (Default: 'all') The type of menu that will be displayed. Can be 'all' for the full menu or 'zoom' for just zoom buttons. The 'all' option requires the full set of Escher dependencies (D3.js, JQuery, and Bootstrap), while the 'zoom' option requires only D3.js. For more details, see Developing with Escher.

  options.**scroll_behavior**
      (Default: 'pan') This option determines the effect that the scroll wheel will have on an Escher map. Can be 'pan' to pan the map or 'zoom' to zoom the map when the user moves the scroll wheel.

options.**use_3d_transform**
   (Default: Chooses a good option by testing the browser) If true, then use CSS3 3D transforms to speed up panning and zooming. This feature will only work on browsers that support the 3D transforms. It works best in the latest versions of Chrome, Firefox and Internet Explorer. Safari works better with this turned off.

options.**enable_editing**
   (Default: `true`) If true then display the map editing functions. If false, then hide them and only allow the user to view the map.

option.**enable_keys**
   (Default: `true`) If true then enable keyboard shortcuts.

options.**enable_search**
   (Default: `true`) If true, then enable indexing of the map for search. Use false to disable searching and potentially improve the map performance.

options.**fill_screen**
   (Default: `false`) Use true to fill the screen when an Escher Builder is placed in a top-level container (e.g. a div in the body element).

options.**full_screen_button**
   (Default: `false`) Include a button in the user interface for entering full screen mode.

options.**ignore_bootstrap**
   (Default: `false`) Do not use Bootstrap for buttons, even if it available. This is used to embed Escher in a Jupyter notebook where it conflicts with the Jupyter Boostrap installation.

Map, model, and styles

options.**starting_reaction**
   (Default: `null`) The ID (as a string) of a reaction to draw when the Builder loads.

options.**never_ask_before_quit**
   (Default: `false`) If false, then display a warning before the user closes an Escher map. If true, then never display the warning. This options is only respected if options.enable_editing == true. If enable_editing is false, then the warnings are not displayed.

options.**unique_map_id**
   (Default: `null`) A unique ID that will be used to UI elements don't interfere when multiple maps are in the same HTML document.

options.**primary_metabolite_radius**
   (Default: `15`) The radius of primary metabolites, in px.

options.**secondary_metabolite_radius**
   (Default: `10`) The radius of secondary metabolites, in px.

options.**marker_radius**
   (Default: `5`) The radius of marker nodes, in px.

options.**gene_font_size**
   (Default: `18`) The font size of the gene reaction rules, in px.

options.**hide_secondary_metabolites**
   (Default: `false`) If true, then secondary nodes and segments are hidden. This is convenient for generating simplified map figures.

options.**show_gene_reaction_rules**
   (Default: `false`) If true, then show the gene reaction rules, even without gene data.

options.**hide_all_labels**
  (Default: `false`) If checked, hide all reaction, gene, and metabolite labels

Applied data

options.**reaction_data**
  (Default: `null`) An object with reaction ids for keys and reaction data points for values.

options.**reaction_styles**
  Default: `['color', 'size', 'text']`

  An array of style types. The array can contain any of the following: 'color', 'size', 'text', 'abs'. The 'color' style means that the reactions will be colored according to the loaded dataset. The 'size' style means that the reactions will be sized according to the loaded dataset. The 'text' style means that the data values will be displayed in the reaction labels. The 'abs' style means the the absolute values of reaction values will be used for data visualization.

options.**reaction_compare_style**
  (Default: `'diff'`) How to compare to datasets. Can be either 'fold, 'log2_fold', or 'diff'.

options.**reaction_scale**
  Default:

```
[ { type: 'min', color: '#c8c8c8', size: 12 },
    { type: 'median', color: '#9696ff', size: 20 },
    { type: 'max', color: '#ff0000', size: 25 } ]
```

  An array of objects that define points on the data scale.

  Each point is an object with a type attribute. Types can be 'min', 'max', 'mean', 'Q1' (first quartile), 'median', 'Q3' (third quartile), or 'value'. Each point can have a color attribute that specifies a color with a string (any CSS color specification is allowed, including hex, rgb, and rgba). Each point can have a size attirbute that specifies a reaction thickness as a number. Finally, points with type 'value' can have a value attribute that specifies an exact number for pont in the scale.

  NOTE: If 'min' or 'max' is not provided, Escher automatically adds them. To be completely clear about what you expect to see on the map, it is best to provide 'min' and 'max' in addition to other scale points.

  Here are examples of each type:

  `{ type: 'min', color: 'red', size: 12 }` Specifies that reactions near the minimum value are red and have thickness 12.

  `{ type: 'Q1', color: 'rgba(100, 100, 50, 0.5)', size: 12 }` Specifies that reactions near the first quartile have the given color, opacity, and thickness.

  `{ type: 'mean', color: 'rgb(100, 100, 50)', size: 50 }` Specifies that reactions near the mean value have the given color and thickness.

  `{ type: 'value', value: 8.5, color: '#333', size: 50 }` Specifies that reactions near 8.5 value have the given color and size.

options.**reaction_no_data_color**
  (Default: `'#dcdcdc'`) The color of reactions with no data value.

options.**reaction_no_data_size**
  (Default: 8) The size of reactions with no data value.

options.**gene_data**
  (Default: `null`) An object with Gene ids for keys and gene data points for values.

options.**and_method_in_gene_reaction_rule**
  (Default: `mean`) When evaluating a gene reaction rule, use this function to evaluate AND rules. Can be 'mean' or 'min'.

options.**metabolite_data**
  (Default: `null`) An object with metabolite ids for keys and metabolite data points for values.

options.**metabolite_styles**
  Default: `['color', 'size', 'text']`

  An array of style types. The array can contain any of the following: 'color', 'size', 'text', 'abs'. The 'color' style means that the metabolites will be colored according to the loaded dataset. The 'size' style means that the metabolites will be sized according to the loaded dataset. The 'text' style means that the data values will be displayed in the metabolite labels. The 'abs' style means the the absolute values of metabolite values will be used for data visualization.

options.**metabolite_compare_style**
  (Default: `'diff'`) How to compare to datasets. Can be either 'fold', 'log2_fold' or 'diff'.

options.**metabolite_scale**
  Default:

```
[ { type: 'min', color: '#fffaf0', size: 20 },
  { type: 'median', color: '#f1c470', size: 30 },
  { type: 'max', color: '#800000', size: 40 } ]
```

  An array of objects that define points on the data scale. See the description of **options.reaction_scale** for an explanation of the format.

options.**metabolite_no_data_color**
  (Default: `'#ffffff'`) The color of metabolites with no data value.

options.**metabolite_no_data_size**
  (Default: `10`) The size of metabolites with no data value.

View and build options

options.**identifiers_on_map**
  (Default: `'bigg_id'`) The identifiers that will be displayed in reaction, metabolite, and gene labels. Can be 'bigg_id' or 'name'.

options.**highlight_missing**
  (Default: `false`) If true, then highlight in red reactions that are not in the loaded COBRA model.

options.**allow_building_duplicate_reactions**
  (Default: `true`) If true, then building duplicate reactions is allowed. If false, then duplicate reactions are hidden in *Add reaction mode*.

options.**cofactors**
  (Default: `['atp', 'adp', 'nad', 'nadh', 'nadp', 'nadph', 'gtp', 'gdp', 'h', 'coa', 'ump', 'h20', 'ppi']`) A list of metabolite IDs to treat as cofactors. These will be secondary metabolites in new reactions.

Callbacks

options.**first_load_callback**
> A function to run after loading the Builder.

load_map(*map_data*[, *should_update_data*])
> Load a map for the loaded data. Also reloads most of the Builder content.
>
> **Arguments**
>
> - **map_data** – The data for a map.
> - **should_update_data** (*Boolean*) – (Default: true) Whether data should be applied to the map.

load_model(*model_data*[, *should_update_data*])
> Load the cobra model from model data.
>
> **Arguments**
>
> - **model_data** – The data for a Cobra model. (Parsing in done by escher.CobraModel).
> - **should_update_data** (*Boolean*) – (Default: true) Whether data should be applied to the model.

view_mode()
> Enter view mode.

build_mode()
> Enter build mode.

brush_mode()
> Enter brush mode.

zoom_mode()
> Enter zoom mode.

rotate_mode()
> Enter rotate mode.

text_mode()
> Enter text mode.

set_reaction_data(*data*)
>
> **Arguments**
>
> - **data** (*array*) – An array of 1 or 2 objects, where each object has keys that are reaction ID's and values that are data points (numbers).

set_metabolite_data(*data*)
>
> **Arguments**
>
> - **data** (*array*) – An array of 1 or 2 objects, where each object has keys that are metabolite ID's and values that are data points (numbers).

set_gene_data(*data*, *clear_gene_reaction_rules*)
>
> **Arguments**
>
> - **data** (*array*) – An array of 1 or 2 objects, where each object has keys that are gene ID's and values that are data points (numbers).
>
> **Arguments**
>
> - **clear_gene_reaction_rules** (*Boolean*) – (Optional, Default: false) In addition to setting the data, also turn of the gene_reaction_rules.

7.9 Python API

class `escher.`**`Builder`**(*map_name=None,   map_json=None,   model=None,   model_name=None,   model_json=None,   embedded_css=None,   reaction_data=None,   metabolite_data=None,   gene_data=None,   local_host=None,   id=None,   safe=False,   **kwargs*)

A metabolic map that can be viewed, edited, and used to visualize data.

This map will also show metabolic fluxes passed in during consruction. It can be viewed as a standalone html inside a browswer. Alternately, the respresentation inside an IPython notebook will also display the map.

Maps are stored in json files and are stored in a cache directory. Maps which are not found will be downloaded from a map repository if found.

> **Parameters**
>
> - **`map_name`** – A string specifying a map to be downloaded from the Escher web server, or loaded from the cache.
>
> - **`map_json`** – A JSON string, or a file path to a JSON file, or a URL specifying a JSON file to be downloaded.
>
> - **`model`** – A Cobra model.
>
> - **`model_name`** – A string specifying a model to be downloaded from the Escher web server, or loaded from the cache.
>
> - **`model_json`** – A JSON string, or a file path to a JSON file, or a URL specifying a JSON file to be downloaded.
>
> - **`embedded_css`** – The CSS (as a string) to be embedded with the Escher SVG.
>
> - **`reaction_data`** – A dictionary with keys that correspond to reaction ids and values that will be mapped to reaction arrows and labels.
>
> - **`metabolite_data`** – A dictionary with keys that correspond to metabolite ids and values that will be mapped to metabolite nodes and labels.
>
> - **`gene_data`** – A dictionary with keys that correspond to gene ids and values that will be mapped to corresponding reactions.
>
> - **`local_host`** – A hostname that will be used for any local files. This is generally used for using the notebook offline and for testing in the IPython Notebook with modified Escher code. An example value for local_host is 'http://localhost:7778/'.
>
> - **`id`** – Specify an id to make the javascript data definitions unique. A random id is chosen by default.
>
> - **`safe`** – If True, then loading files from the filesytem is not allowed. This is to ensure the safety of using Builder within a web server.

Keyword Arguments

These are defined in the Javascript API:

> •use_3d_transform
>
> •enable_search
>
> •fill_screen
>
> •zoom_to_element
>
> •full_screen_button

- •starting_reaction

- •unique_map_id

- •primary_metabolite_radius

- •secondary_metabolite_radius

- •marker_radius

- •gene_font_size

- •hide_secondary_metabolites

- •show_gene_reaction_rules

- •hide_all_labels

- •reaction_styles

- •reaction_compare_style

- •reaction_scale

- •reaction_no_data_color

- •reaction_no_data_size

- •and_method_in_gene_reaction_rule

- •metabolite_styles

- •metabolite_compare_style

- •metabolite_scale

- •metabolite_no_data_color

- •metabolite_no_data_size

- •identifiers_on_map

- •highlight_missing

- •allow_building_duplicate_reactions

- •cofactors

All keyword arguments can also be set on an existing Builder object using setter functions, e.g.:

```
my_builder.set_reaction_styles(new_styles)
```

display_in_browser(*ip='127.0.0.1'*, *port=7655*, *n_retries=50*, *js_source='web'*, *menu='all'*, *scroll_behavior='pan'*, *enable_editing=True*, *enable_keys=True*, *minified_js=True*, *never_ask_before_quit=False*)
  Launch a web browser to view the map.

  Parameters

  - **ip** – The IP address to serve the map on.

  - **port** – The port to serve the map on. If specified the port is occupied, then a random free port will be used.

  - **n_retries** (*int*) – The number of times the server will try to find a port before quitting.

  - **js_source** (*string*) – Can be one of the following:

    - *web* (Default) - Use JavaScript files from escher.github.io.

- *local* - Use compiled JavaScript files in the local Escher installation. Works offline.
- ***dev* - No longer necessary with source maps. This now gives the** same behavior as 'local'.

- **menu** (`string`) – Menu bar options include:

  - *none* - No menu or buttons.
  - *zoom* - Just zoom buttons.
  - *all* (Default) - Menu and button bar (requires Bootstrap).

- **scroll_behavior** (`string`) – Scroll behavior options:

  - *pan* - Pan the map.
  - *zoom* - Zoom the map.
  - *none* (Default) - No scroll events.

- **enable_editing** (`Boolean`) – Enable the map editing modes.

- **enable_keys** (`Boolean`) – Enable keyboard shortcuts.

- **minified_js** (`Boolean`) – If True, use the minified version of JavaScript and CSS files.

- **never_ask_before_quit** (`Boolean`) – Never display an alert asking if you want to leave the page. By default, this message is displayed if enable_editing is True.

display_in_notebook (*js_source='web'*, *menu='zoom'*, *scroll_behavior='none'*, *minified_js=True*, *height=500*, *enable_editing=False*)

Embed the Map within the current IPython Notebook.

Parameters

- **js_source** (`string`) – Can be one of the following:

  - *web* (Default) - Use JavaScript files from escher.github.io.
  - *local* - Use compiled JavaScript files in the local Escher installation. Works offline.
  - ***dev* - No longer necessary with source maps. This now gives the** same behavior as 'local'.

- **menu** (`string`) – Menu bar options include:

  - *none* - No menu or buttons.
  - *zoom* - Just zoom buttons.
  - Note: The *all* menu option does not work in an IPython notebook.

- **scroll_behavior** (`string`) – Scroll behavior options:

  - *pan* - Pan the map.
  - *zoom* - Zoom the map.
  - *none* - (Default) No scroll events.

- **minified_js** (`Boolean`) – If True, use the minified version of JavaScript and CSS files.

- **height** – Height of the HTML container.

- **enable_editing** (`Boolean`) – Enable the map editing modes.

save_html (*filepath=None*, *overwrite=False*, *js_source='web'*, *protocol='https'*, *menu='all'*, *scroll_behavior='pan'*, *enable_editing=True*, *enable_keys=True*, *minified_js=True*, *never_ask_before_quit=False*, *static_site_index_json=None*)
>   Save an HTML file containing the map.

>   **Parameters**

>   - **filepath** (*string*) – The HTML file will be saved to this location. When js_source is 'local', then a new directory will be created with this name.

>   - **overwrite** (*Boolean*) – Overwrite existing files.

>   - **js_source** (*string*) – Can be one of the following:

>   – *web* (Default) - Use JavaScript files from escher.github.io.

>   – *local* - **Use compiled JavaScript files in the local Escher** installation. Works offline. To make the dependencies available to the downloaded file, a new directory will be made with the name specified by filepath.

>   – *dev* - **No longer necessary with source maps. This now gives the** same behavior as 'local'.

>   - **protocol** (*string*) – The protocol can be 'http', 'https', or None which indicates a 'protocol relative URL', as in //escher.github.io. Ignored if source is local.

>   - **menu** (*string*) – Menu bar options include:

>   – *none* - No menu or buttons.

>   – *zoom* - Just zoom buttons.

>   – *all* (Default) - Menu and button bar (requires Bootstrap).

>   - **scroll_behavior** (*string*) – Scroll behavior options:

>   – *pan* - Pan the map.

>   – *zoom* - Zoom the map.

>   – *none* (Default) - No scroll events.

>   - **enable_editing** (*Boolean*) – Enable the map editing modes.

>   - **enable_keys** (*Boolean*) – Enable keyboard shortcuts.

>   - **minified_js** (*Boolean*) – If True, use the minified version of JavaScript and CSS files.

>   - **height** (*number*) – Height of the HTML container.

>   - **never_ask_before_quit** (*Boolean*) – Never display an alert asking if you want to leave the page. By default, this message is displayed if enable_editing is True.

>   - **static_site_index_json** (*string*) – The index, as a JSON string, for the static site. Use javascript to parse the URL options. Used for generating static pages (see static_site.py).

7.9.1 Cache

escher.**get_cache_dir** (*versioned=True*, *name=None*)
>   Get the cache dir as a string, and make the directory if it does not already exist.

>   **Parameters**

- **versioned** (`Boolean`) – Whether to return the versioned path in the cache. Escher maps for the latest version of Escher are found in the versioned directory (versioned = True), but maps for previous versions of Escher can be found by visiting the parent directory (versioned = False).

- **name** (`string`) – An optional subdirectory within the cache. If versioned is False, then name is ignored.

escher.**clear_cache**(*different_cache_dir=None*, *ask=True*)
Empty the contents of the cache directory, including all versions of all maps and models.

> **Parameters**
>
> - **different_cache_dir** (`string`) – (Optional) The directory of another cache. This is mainly for testing.
>
> - **ask** (`Boolean`) – Whether to ask before deleting.

escher.**list_cached_maps**()
Return a list of all cached maps.

escher.**list_cached_models**()
Return a list of all cached models.

escher.**list_available_maps**()
Return a list of all maps available on the server

escher.**list_available_models**()
Return a list of all models available on the server

7.10 License

Escher and EscherConverter are distributed under the MIT license.

- genindex

e

T

text_mode() (built-in function),

V

view_mode() (built-in function),

Z

zoom_mode() (built-in function),