

# Package ‘emln’

May 13, 2026

**Type** Package

**Title** Organize, Handle, and Explore Ecological Multilayer Networks

**Version** 1.2.0

**Date** 2026-05-13

**URL** <https://emln.ecomplab.com/>

**BugReports** <https://github.com/Ecological-Complexity-Lab/emln/issues>

## Description

Data and analysis of ecological multilayer networks, including standardization of data structures and functions to convert between them. Includes an interactive multilayer network visualizer (beta, paper forthcoming), and a collection of 78 empirical ecological multilayer network datasets. This work was supported by research grant ISF (Israel Science Foundation) 1281/20 to Shai Pilosof. Noa Frydman (2023) <[doi:10.1111/2041-210X.14225](https://doi.org/10.1111/2041-210X.14225)>.

**License** CC BY 4.0

**Encoding** UTF-8

**LazyData** true

**LazyDataCompression** xz

**Depends** R (>= 4.1.0)

**Imports** magrittr, stringr, igraph, tibble, dplyr, purrr, tidy,  
Matrix, DT, hablar, rlang

**Suggests** bipartite, httpuv, testthat (>= 3.0.0)

**RoxygenNote** 7.3.2

**Config/testthat/edition** 3

**NeedsCompilation** no

**Author** Noa Frydman [aut],  
Shai Pilosof [aut, fnd],  
Shirly Freilikhman [aut],  
Geut Galai [cre]

**Maintainer** Geut Galai <[geutg@post.bgu.ac.il](mailto:geutg@post.bgu.ac.il)>

**Repository** CRAN

**Date/Publication** 2026-05-13 11:00:02 UTC

## Contents

create_monolayer_network . . . . .	2
create_multilayer_network . . . . .	3
descriptions . . . . .	5
get_igraph . . . . .	6
get_sam . . . . .	7
list_to_matrix . . . . .	8
load_emln . . . . .	9
matrix_to_list_bipartite . . . . .	10
matrix_to_list_unipartite . . . . .	11
monolayer class . . . . .	12
multilayer class . . . . .	12
multilayer_to_csv . . . . .	13
multilayer_to_json . . . . .	15
plot_multilayer . . . . .	17
search_emln . . . . .	18
view_emln . . . . .	20
<b>Index</b>	<b>21</b>

---

create\_monolayer\_network

*Creates a monolayer network object.*

---

### Description

Automatically identifies if the input is a matrix or an edge list and creates a monolayer object.

### Usage

```
create_monolayer_network(
  x,
  directed = NULL,
  bipartite = NULL,
  group_names = c("set_cols", "set_rows"),
  node_metadata = NULL
)
```

### Arguments

x	input data in the format of a matrix, edge list or an igraph object.
directed	Is the network directed?
bipartite	Is the network bipartite? Ignored when the input is an igraph object.
group_names	For bipartite networks: name of the groups in the columns and rows, respectively (e.g., parasites and hosts).
node_metadata	Following the igraph method of graph_from_data_frame. First column must be have names matching those in x.

**Details**

Converts between edge list and matrix formats and creates a monolayer object. It a wrapper function for `list_to_matrix`, `matrix_to_list_unipartite`, `matrix_to_list_bipartite`. Node metadata can only be included with an edge list input.

**Value**

A monolayer object.

**See Also**

`list_to_matrix`, `matrix_to_list_unipartite`, `monolayer`.

**Examples**

```
library(igraph)

# A bipartite network from package bipartite
x <- create_monolayer_network(bipartite::memmott1999, bipartite = TRUE,
  directed = FALSE, group_names = c('Animals', 'Plants'))

# A bipartite network as an igraph object

# Generate a random bipartite network in igraph
g <- igraph::sample_bipartite(10,16,p=0.3, type = 'gnp', directed = TRUE, mode = 'in')
V(g)$name <- letters # name the nodes
V(g)$gender <- sample(c('F','M'), size = 26, replace = TRUE) # Add a node attribute
E(g)$weight=runif(ecount(g)) # Add edge weights
plot(g, layout=layout.bipartite)
create_monolayer_network(g, group_names = c('Parasites', 'Hosts'))
```

---

create\_multilayer\_network

*Create Multilayer Network*

---

**Description**

This function creates a multilayer network from multiple matrices / link lists.

**Usage**

```
create_multilayer_network(  
  list_of_layers,  
  bipartite,
```

```

    directed,
    interlayer_links = NULL,
    layer_attributes = NULL,
    state_node_attributes = NULL,
    physical_node_attributes = NULL
)

```

### Arguments

`list_of_layers` List of matrices or list of link lists (format from to weight; see description). These are the intralayer links.

`bipartite` Is the network bipartite? Must be provided (no default value).

`directed` Is the network directed? Must be provided (no default value).

`interlayer_links` Interlayer extended edge list of the format layer\_from node\_from layer\_to node\_to weight. Default is NULL, assuming no interlayer links.

`layer_attributes` Optional. A data frame with layer attributes. The first column must be called `layer_id`, and it must have unique values (one row per layer). Layer name column, if exists must be called `layer_name`, and its values must be unique. The order of the rows and `layer_id` should be the same as in `list_of_layers`. No column is allowed to be named `name`.

`state_node_attributes` Optional. Additional information on physical nodes in layers. Must contain columns `layer_name` and `node_name`.

`physical_node_attributes` Optional. Additional information on physical nodes. Must contain column `node_name`.

### Details

Input of `list_of_layers` is handled by the function `create_monolayer_network`; see its description for details on input. If using matrices as input, they should contain row and column names. If link lists are provided as input column names should be from to weight.

When using link lists, it is possible to include link attributes. In that case the headers of all the link lists and, if included, the interlayer links, must be the same (after the weight), even if some attributes are included in only some layers (see detailed example in the code accompanying the package). Missing link attributes can be set to NA.

If interlayer links are provided, then `layer_attributes` must also be provided. Specify layer and nodes by UNIQUE names (not IDs). Layer names should correspond to those provided in `layer_attributes`.

Attributes of state nodes can be provided with `state_node_attributes`. If provided, then columns `layer_name` and `node_name` must be included.

For compatibility with function `load_eml` the output contains description and references. The value of any missing data frames in the multilayer object will be set to NULL.

See multiple example in the code accompanying the package

**Value**

A multilayer object (see ?multilayer).

**See Also**

multilayer, create\_monolayer\_network

**Examples**

```
library(tibble)
pond_1 <- matrix(c(0,1,1,0,0,1,0,0,0), byrow = TRUE,
  nrow = 3, ncol = 3, dimnames = list(c('pelican', 'fish', 'crab'),
  c('pelican', 'fish', 'crab')))
pond_2 <- matrix(c(0,1,0,0,0,1,0,0,0), byrow = TRUE,
  nrow = 3, ncol = 3, dimnames = list(c('pelican', 'fish', 'crab'),
  c('pelican', 'fish', 'crab')))
pond_3 <- matrix(c(0,1,1,0,0,1,0,0,0), byrow = TRUE,
  nrow = 3, ncol = 3, dimnames = list(c('pelican', 'fish', 'tadpole'),
  c('pelican', 'fish', 'tadpole')))

layer_attrib <- tibble(layer_id=1:3,
  layer_name=c('pond_1', 'pond_2', 'pond_3'),
  location=c('valley', 'mid-elevation', 'uphill'))

# Create the ELL tibble with interlayer links.
interlayer <- tibble(layer_from=c('pond_1', 'pond_1', 'pond_1'),
  node_from=c('pelican', 'crab', 'pelican'),
  layer_to=c('pond_2', 'pond_2', 'pond_3'),
  node_to=c('pelican', 'crab', 'pelican'),
  weight=1)

# This is a directed network so the links should go both ways,
# even though they are symmetric.
interlayer_2 <- interlayer[,c(3,4,1,2,5)]
names(interlayer_2) <- names(interlayer)
interlayer <- rbind(interlayer, interlayer_2)

multilayer <- create_multilayer_network(list_of_layers =
  list(pond_1, pond_2, pond_3), layer_attributes = layer_attrib,
  interlayer_links = interlayer, bipartite = FALSE, directed = TRUE)
```

---

descriptions

*View the data sets*

---

**Description**

@format Data set of multilayer networks

**Usage**

descriptions

**Format**

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 89 rows and 10 columns.

**Details**

@source create

@examples data(descriptions)

---

get\_igraph

*Convert a multilayer class to a list of igraph objects*

---

**Description**

This function creates a list of `igraph` objects from the network layers. Interlayer links are not included.

**Usage**

```
get_igraph(multilayer, bipartite, directed)
```

**Arguments**

<code>multilayer</code>	The multilayer object.
<code>bipartite</code>	Is the network bipartite? Must be provided (no default value).
<code>directed</code>	Is the network directed? Must be provided (no default value).

**Details**

If link, physical node or state node attributes are included in the multilayer network they are passed to the `igraph` objects.

**Value**

A list:

- `layers_igraph` A list of `igraph` objects, one per layer
- `nodes` A table of physical nodes, as in the multilayer object.
- `state_nodes_map` A table with state nodes to map the nodes to the row and columns of a SAM.

**Examples**

```
# See examples in:
# https://emln.ecomplab.com/multilayer.html#To_igraph_objects
```

---

`get_sam`*Convert a multilayer class to a supra-adjacency matrix*

---

### Description

This function creates a supra-adjacency matrix from a multilayer object.

### Usage

```
get_sam(  
  multilayer,  
  bipartite,  
  directed,  
  sparse = FALSE,  
  remove_zero_rows_cols = FALSE  
)
```

### Arguments

<code>multilayer</code>	The multilayer object.
<code>bipartite</code>	Is the network bipartite? Must be provided (no default value).
<code>directed</code>	Is the network directed? Must be provided (no default value).
<code>sparse</code>	Should the returned matrix be a sparse matrix (class Matrix)? Defaults to false.
<code>remove_zero_rows_cols</code>	Should rows and columns that sum to 0 be removed? Defaults to false.

### Details

By default, a SAM contains all the nodes in all the layers. However, not all nodes always occur in all layers. This will be reflected in the `state_nodes_map`: When layer and node ids are NA that means that the node did not occur in the layer.

A node may not have links across all the layers. In that case, its row or column sum will be zero. This can happen in a directed matrix (see toy example for unipartite network), or in diagonally-coupled networks (see toy bipartite example). The user can choose to remove rows and columns that sum to zero by setting `remove_zero_rows_cols` to TRUE.

### Value

A list:

- `M` The SAM. Can also be sparse.
- `nodes` A table of physical nodes, as in the multilayer object.
- `state_nodes_map` A table with state nodes to map the nodes to the row and columns of the SAM.

**Examples**

```
# See examples in:
# https://emln.ecomplab.com/multilayer.html#To_supra-adjacency_matrices
```

---

list_to_matrix	<i>Edge list to matrix conversion</i>
----------------	---------------------------------------

---

**Description**

Convert an edge list to a matrix. Can handle unipartite and bipartite networks.

**Usage**

```
list_to_matrix(
  x,
  directed = FALSE,
  bipartite = TRUE,
  group_names = c("set_cols", "set_rows"),
  node_metadata = NULL
)
```

**Arguments**

x	An edge list of the format from to weight.
directed	Is the network directed?
bipartite	Is network bipartite?
group_names	For bipartite networks: name of the groups in the columns and rows, respectively (e.g., parasites and hosts).
node_metadata	Following the igraph method of graph_from_data_frame. Must have a column called node_name with names matching those in x.

**Details**

It is also possible to add node attributes. For this, it uses igraph. The first column in the node\_metadata data frame should have the names of the nodes in the edge list.

**Value**

A monolayer object.

**See Also**

Functions create\_monolayer\_network, monolayer and graph\_from\_data\_frame from igraph.



**Examples**

```
library(dplyr)
network <- load_emln(17)
nodes <- network$nodes
links<-subset(network$extended_ids %>% filter(layer_from==1), select=c(node_from, node_to, weight))

list_to_matrix(links, directed = TRUE, bipartite = FALSE, node_metadata = nodes)

# See examples in:
# https://emln.ecomplab.com/monolayer.html#Unipartite
```

---

load_emln	<i>Load networks include din the package</i>
-----------	----------------------------------------------

---

**Description**

Load a specific network by specifying its ID.

**Usage**

```
load_emln(network_id)
```

**Arguments**

network\_id      The network ID.

**Details**

Use the search\_emln and view\_emln functions to obtain network IDs.

**Value**

A multilayer object (see ?multilayer).

**See Also**

multilayer

**Examples**

```
load_emln(network_id = 38)
```

---

`matrix_to_list_bipartite`*Incidence matrix to edge list conversion*

---

**Description**

Convert an incidence matrix to an edge list.

**Usage**

```
matrix_to_list_bipartite(x, group_names = c("set_cols", "set_rows"))
```

**Arguments**

<code>x</code>	An incidence matrix.
<code>group_names</code>	Name of the groups in the columns and rows, respectively (e.g., parasites and hosts).

**Details**

Assigns column and row names if those do not exist. Cannot handle node attributes/metadata. Always assumes an undirected network.

**Value**

A monolayer object.

**See Also**

`create_monolayer_network`, `monolayer`

**Examples**

```
# See examples in: https://emln.ecomplab.com/monolayer.html#Bipartite  
matrix_to_list_bipartite(bipartite::memmott1999, group_names = c('Animals', 'Plants'))
```

---

`matrix_to_list_unipartite`*Adjacency matrix to edge list conversion*

---

**Description**

Convert an adjacency matrix to an edge list.

**Usage**

```
matrix_to_list_unipartite(x, directed)
```

**Arguments**

<code>x</code>	An adjacency or incidence matrix.
<code>directed</code>	Is the network directed? TRUE/FALSE

**Details**

Assigns column and row names if those do not exist. Cannot handle node attributes/metadata.

**Value**

A monolayer object

**See Also**

`create_monolayer_network`, `monolayer`

**Examples**

```
# See examples in:  
# https://emln.ecomplab.com/monolayer.html#Unipartite  
  
# Generate a matrix with random weighted interactions  
x <- matrix(rbinom(100,1,0.6),10,10)  
x <- x*round(runif(100,1,5),0)  
# run  
matrix_to_list_unipartite(x, directed = TRUE)
```

---

monolayer class      *An object of class monolayer*

---

### Description

A network object of class monolayer contains all the necessary information and R objects to perform future analyses on a monolayer network, and in particular analyses using Infomap.

### Value

A list with:

- mode: bipartite (B) or unipartite (U)
- directed: TRUE/FALSE
- nodes: a tibble with information on the nodes (name, id, type, other attributes)
- mat: the network in a matrix format
- edge\_list: the network in an edge list format
- igraph\_object: the igraph object of the network

### See Also

`list_to_matrix`, `matrix_to_list_unipartite`, `matrix_to_list_bipartite`, `create_network_object`

---

multilayer class      *An object of class multilayer*

---

### Description

A network object of class multilayer contains all the necessary information and R objects that define a multilayer network.

### Value

A list with:

- nodes Physical nodes. First column is a unique node id. Node attributes are included if provided as input.
- layers Information on layers.
- extended An extended link list of the format layer\_from node\_from layer\_to node\_to weight. All nodes and layers are identified by names.
- extended\_ids An extended link list of the format layer\_from node\_from layer\_to node\_to weight. All nodes and layers are identified by unique IDs automatically generated.
- state\_nodes List of state nodes of the format layer\_id node\_id layer\_name node\_name. Also includes state node attributes if provided as input.
- description For compatibility with `load_emln`.
- references For compatibility with `load_emln`.

**See Also**

create\_multilayer\_network

---

multilayer\_to\_csv      *Export a multilayer network to CSV files*

---

**Description**

Writes an EMLN multilayer object as a set of CSV files consumed by MiRA's CSV importer. Three files are always written; a fourth (<prefix>\_state\_nodes.csv) is added when per-(layer, node) attributes are present.

**Usage**

```
multilayer_to_csv(
  multilayer,
  dir,
  prefix = "network",
  bipartite = FALSE,
  setA_type = NULL,
  directed = NULL
)
```

**Arguments**

multilayer	A multilayer object (created by create_multilayer_network or load_emln).
dir	Directory to write the CSV files to. Created if it does not already exist.
prefix	Character. File name prefix. Defaults to "network". Three files are always written: <prefix>_edges.csv, <prefix>_layers.csv, <prefix>_nodes.csv. A fourth file <prefix>_state_nodes.csv is written when the multilayer object contains per-(layer, node) attributes beyond layer_name and node_name (e.g. abundance, module).
bipartite	Logical. Whether the network is bipartite. Defaults to FALSE. When TRUE, the nodes table must contain a node_type (or legacy node_group) column with exactly two distinct values across the network, and a bipartite column is written to the layers CSV.
setA_type	Optional character. The node_type value to render as Set A (top row) in MiRA's bipartite layout. By ecological convention this is the higher trophic level (e.g. "pollinator", "parasite", "disperser"). Only used when bipartite = TRUE; written as a setA_type column in the layers CSV. If NULL (default), MiRA falls back to alphabetical ordering of the two types.
directed	Logical or NULL. Whether the network is directed. If NULL (default), auto-detected by checking intralayer edge symmetry.

## Details

The CSV files follow the schema accepted by MiRA's CSV importer:

- **edges:** layer\_from, node\_from, layer\_to, node\_to, weight (+ any extra link attributes).
- **layers:** layer\_id, layer\_name (+ latitude, longitude, bipartite and any extra layer attributes).
- **nodes:** node\_name (+ node\_type and any extra physical node attributes, same across all layers). node\_group is renamed to node\_type.
- **state\_nodes** (written only when extra per-(layer, node) attributes exist): layer\_name, node\_name + extra columns. Use this when the same node has different attribute values in different layers (e.g. abundance, module membership).

The directed flag is not written to the CSV files; it is a property you specify in MiRA's CSV import dialog at load time.

## Value

Invisibly returns a named character vector of file paths written (edges, layers, nodes, and optionally state\_nodes).

## Note

MiRA (Multilayer Interactive Rendering Application) is a browser-based interactive visualization tool for multilayer networks, available at <https://mira.ecomplab.com/>.

If you use MiRA in published research, please cite:

Nehoray SM, Bloch Y, Pilosof S (2026). Interactively visualizing biological multilayer networks using MiRA. *arXiv:2605.09597* [cs.SI]. doi:10.48550/arXiv.2605.09597

Feedback and bug reports: <https://github.com/Ecological-Complexity-Lab/MiRA/issues>.

## See Also

[multilayer\\_to\\_json](#), [plot\\_multilayer](#)

## Examples

```
net <- load_emln(14)
multilayer_to_csv(net, dir = "tests/out/", prefix = "kefi", bipartite = TRUE)
```

---

multilayer\_to\_json      *Export a multilayer network to JSON*

---

### Description

Converts an EMLN multilayer object to JSON format compatible with MiRA, the multilayer interactive R-network app.

### Usage

```
multilayer_to_json(
  multilayer,
  file = NULL,
  bipartite = FALSE,
  setA_type = NULL,
  directed = NULL,
  directed_interlayer = NULL
)
```

### Arguments

multilayer	A multilayer object (created by <code>create_multilayer_network</code> or <code>load_emln</code> ).
file	Optional file path to write the JSON to. If NULL, returns the JSON string.
bipartite	Logical. Whether the network is bipartite. Defaults to FALSE. MiRA does not auto-detect bipartite structure — you must set this to TRUE explicitly. When TRUE, the nodes table must contain a <code>node_type</code> (or legacy <code>node_group</code> ) column with exactly two distinct values across the network. Any <code>node_group</code> column is renamed to <code>node_type</code> in the output regardless of the bipartite flag (MiRA's canonical name).
setA_type	Optional character. The <code>node_type</code> value to render as Set A (top row) in MiRA's bipartite layout. By ecological convention this is the higher trophic level (e.g. "pollinator", "parasite", "disperser"). Only used when <code>bipartite = TRUE</code> . If NULL (default), MiRA falls back to alphabetical ordering of the two types.
directed	Logical or NULL. Whether intralayer links are directed. If NULL (default), auto-detected by checking intralayer edge symmetry.
directed_interlayer	Logical or NULL. Whether interlayer links are directed. If NULL (default), inherits from <code>directed</code> (matching MiRA's parser behavior). Set explicitly when intra- and interlayer directionality differ.

### Details

The JSON output contains four arrays matching MiRA's expected format:

- `nodes`: Physical nodes with `node_id`, `node_name`, and any extra attributes. For bipartite networks, `node_group` is mapped to `node_type`.

- `layers`: Layer metadata with `layer_id`, `layer_name`, and extra attributes. For bipartite networks, a `bipartite: true` flag is added; if `setA_type` is given it is also written per layer.
- `extended`: The extended edge list with `layer_from`, `node_from`, `layer_to`, `node_to`, `weight`, and any link attributes.
- `state_nodes`: State node map with `layer_id`, `node_id`, `layer_name`, `node_name`, plus any extra per-(layer, node) attributes (e.g. `abundance`, `module`).

Top-level flags `directed` and (when set) `directed_interlayer` indicate edge directionality globally; they are not repeated per link.

### Value

If `file` is `NULL`, returns the JSON string invisibly. If `file` is specified, writes JSON to disk and returns the file path invisibly.

### Note

MiRA (Multilayer Interactive Rendering Application) is a browser-based interactive visualization tool for multilayer networks, available at <https://mira.ecomplab.com/>.

If you use MiRA in published research, please cite:

Nehoray SM, Bloch Y, Pilosof S (2026). Interactively visualizing biological multilayer networks using MiRA. *arXiv:2605.09597* [cs.SI]. doi:10.48550/arXiv.2605.09597

Feedback and bug reports: <https://github.com/Ecological-Complexity-Lab/MiRA/issues>.

### See Also

`plot_multilayer`, `create_multilayer_network`, `load_emln`

### Examples

```
# Export to file
net <- load_emln(14)
multilayer_to_json(net, file = "tests/my_network.json", bipartite = TRUE)

# Bipartite with explicit Set A ordering
multilayer_to_json(net, bipartite = TRUE, setA_type = "pollinator")

# Get JSON string
json_str <- multilayer_to_json(net)
```



---

plot\_multilayer      *Plot a multilayer network in the browser*

---

## Description

Converts an EMLN multilayer object to JSON and opens MiRA, the multilayer interactive R-network app, in the default web browser.

## Usage

```
plot_multilayer(
  multilayer,
  bipartite = FALSE,
  setA_type = NULL,
  directed = NULL,
  directed_interlayer = NULL,
  port = 8080,
  mira_path = NULL,
  browser = getOption("browser")
)
```

## Arguments

multilayer	A multilayer object (created by <code>create_multilayer_network</code> or <code>load_emln</code> ).
bipartite	Logical. Whether the network is bipartite. Defaults to <code>FALSE</code> . Bipartite is not auto-detected. When <code>TRUE</code> , the nodes table must contain a <code>node_type</code> (or legacy <code>node_group</code> ) column with exactly two distinct values across the network.
setA_type	Optional character. The <code>node_type</code> value to render as Set A (top row) in MiRA's bipartite layout. By ecological convention this is the higher trophic level (e.g. "pollinator", "parasite", "disperser"). Only used when <code>bipartite = TRUE</code> . Forwarded to <code>multilayer_to_json</code> .
directed	Logical or <code>NULL</code> . Whether intralayer links are directed. If <code>NULL</code> (default), auto-detected by checking intralayer edge symmetry.
directed_interlayer	Logical or <code>NULL</code> . Whether interlayer links are directed. If <code>NULL</code> (default), inherits from <code>directed</code> .
port	Integer. Port for the local HTTP server. Default is 8080.
mira_path	Character. Path to the MiRA web-app directory. If <code>NULL</code> (default), uses the copy bundled with the package ( <code>system.file("MiRA", package = "emln")</code> ).
browser	Choose the specific browser to open MiRA in. Defaults to the system default via <code>getOption("browser")</code> . Can be a browser name (e.g. "chrome", "firefox") or a command (e.g. "open -a 'Google Chrome'").

## Details

The function:

1. Converts the multilayer object to JSON via `multilayer_to_json`
2. Starts a local HTTP server (using `httpuv`) that serves the MiRA app and the network JSON
3. Opens the browser with auto-load enabled

The server runs in the background. Call `httpuv::stopServer(handle)` or close R to stop it. The JSON data is kept in memory and never written to disk.

## Value

Invisibly returns the server handle. Use `httpuv::stopServer(handle)` to stop the server when done.

## Note

MiRA (Multilayer Interactive Rendering Application) is a browser-based interactive visualization tool for multilayer networks, available at <https://mira.ecomplab.com/>.

If you use MiRA in published research, please cite:

Nehoray SM, Bloch Y, Pilosof S (2026). Interactively visualizing biological multilayer networks using MiRA. *arXiv:2605.09597* [cs.SI]. doi:10.48550/arXiv.2605.09597

Feedback and bug reports: <https://github.com/Ecological-Complexity-Lab/MiRA/issues>.

## See Also

`multilayer_to_json`, `create_multilayer_network`, `load_emln`

## Examples

```
if (interactive()){
net <- load_emln(60)
srv <- plot_multilayer(net, bipartite = TRUE, setA_type = "pollinator")

# When done:
httpuv::stopServer(srv)
}
```

## Description

Search the database for networks depending on search parameters specified below.

**Usage**

```
search_emln(
  ecological_network_type = NULL,
  multilayer_network_type = NULL,
  state_nodes = NULL,
  node_number_minimum = NULL,
  layer_number_minimum = NULL,
  weighted = NULL,
  interlayer = NULL,
  directed = NULL
)
```

**Arguments**

ecological_network_type	One of: Seed-Dispersal, Pollination, Food-Web, Host-Parasite, Multiple, Plant-Herbivore, Anemone-Fish, Plant-Ant
multilayer_network_type	Environment, Temporal, Multiplex, Perturbation, Spatial
state_nodes	TRUE/FALSE
node_number_minimum	The minimum number of layers wanted in the network
layer_number_minimum	The minimum number of layers wanted in the network
weighted	TRUE/FALSE
interlayer	TRUE/FALSE
directed	TRUE/FALSE

**Value**

A tibble containing the network id, and all the search arguments for each of the networks that are compatible with the search.

**Examples**

```
# See examples in:
# https://emln.ecomplab.com/data.html#Browsing_the_data

# Generates a tibble of all the networks in the package that are pollination networks
search_emln(ecological_network_type = 'Plant-Ant')

# Generates a tibble of all the networks in the package that have state nodes and are temporal
search_emln(multilayer_network_type = 'Temporal', state_node = TRUE)
```

---

`view_emln`*View networks*

---

**Description**

View the database interactively.

**Usage**

```
view_emln()
```

**Value**

An interactive GUI for browsing through networks.

# Index

## \* datasets

- descriptions, [5](#)
  
- [create\\_monolayer\\_network](#), [2](#)
- [create\\_multilayer\\_network](#), [3](#)
  
- descriptions, [5](#)
  
- [get\\_igraph](#), [6](#)
- [get\\_sam](#), [7](#)
  
- [list\\_to\\_matrix](#), [8](#)
- [load\\_emln](#), [9](#)
  
- [matrix\\_to\\_list\\_bipartite](#), [10](#)
- [matrix\\_to\\_list\\_unipartite](#), [11](#)
- [monolayer](#) (monolayer class), [12](#)
- [monolayer class](#), [12](#)
- [multilayer](#) (multilayer class), [12](#)
- [multilayer class](#), [12](#)
- [multilayer\\_to\\_csv](#), [13](#)
- [multilayer\\_to\\_json](#), [14](#), [15](#)
  
- [plot\\_multilayer](#), [14](#), [17](#)
  
- [search\\_emln](#), [18](#)
  
- [view\\_emln](#), [20](#)