

# Package ‘carnation’

May 7, 2026

**Title** Interactive Exploration & Management of RNA-Seq Analyses

**Version** 1.0.0

**Description** Highly interactive & modular shiny app to explore three facets of RNA-Seq analysis: differential expression (DE), functional enrichment and pattern analysis. Several visualizations are implemented to provide a wide-ranging view of data sets. For DE analysis, we provide PCA plot, MA plot, Upset plot & heatmaps, in addition to a highly customizable gene plot. Seven different visualizations are available for functional enrichment analysis, and we also support gene pattern analysis. Genes of interest can be tracked across all modules using the gene scratchpad. In addition, carnation provides an integrated platform to manage multiple projects and user access that can be run on a central server to share with collaborators.

**License** MIT + file LICENSE

**Encoding** UTF-8

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.3

**Depends** R (>= 4.6.0)

**Imports** BiocParallel, colorspace, ComplexUpset, dendextend, DESeq2, dplyr, DT, enrichplot, GeneTonic, ggplot2, ggrepel, heatmaply, htmltools, igraph, methods, MatrixGenerics, plotly, reticulate, RColorBrewer, rintrojs, scales, shiny, shinyBS, shinycssloaders, shinymanager, shinythemes, shinyWidgets, sortable, SummarizedExperiment, tools, utils, viridisLite, visNetwork, yaml

**Suggests** airway, BiocStyle, DEGREport, GenomicFeatures, goseq, knitr, org.Hs.eg.db, rmarkdown, testthat

**VignetteBuilder** knitr

**URL** <https://nichd-bspc.github.io/carnation/>

**BugReports** <https://github.com/NICHD-BSPC/carnation/issues>

**biocViews** GUI, GeneExpression, Software, ShinyApps, GO, Transcription, Transcriptomics, Visualization, DifferentialExpression, Pathways, GeneSetEnrichment

**git\_url** <https://git.bioconductor.org/packages/carnation>

**git\_branch** RELEASE\_3\_23

**git\_last\_commit** 2ff883e

**git\_last\_commit\_date** 2026-04-28

**Repository** Bioconductor 3.23

**Date/Publication** 2026-05-07

**Author** Apratim Mitra [aut, cre] (ORCID:

<https://orcid.org/0000-0003-3279-0054>),

Matthew Tyler Menold [ctb] (ORCID:

<https://orcid.org/0009-0007-4728-2470>),

Ryan Dale [fnd] (ORCID: <https://orcid.org/0000-0003-2664-3744>)

**Maintainer** Apratim Mitra <[apratim.mitra@gmail.com](mailto:apratim.mitra@gmail.com)>

## Contents

carnation-package . . . . .	4
add.set.column . . . . .	5
add_metadata . . . . .	5
alluvialmod . . . . .	6
check_user_access . . . . .	7
cnetmod . . . . .	8
create_access_yaml . . . . .	9
degmod . . . . .	10
degpatterns_dex . . . . .	12
dendromod . . . . .	13
distillmod . . . . .	14
dlmod . . . . .	15
dummy_genetonic . . . . .	16
emapmod . . . . .	17
enrich_to_genetonic . . . . .	18
eres_cell . . . . .	19
eres_dex . . . . .	19
format_genes . . . . .	20
fromList.with.names . . . . .	21
funenrichmod . . . . .	21
fuzzymod . . . . .	23
geneplotmod . . . . .	24
getcountplot . . . . .	26
get_access_path . . . . .	28
get_config . . . . .	28
get_config_path . . . . .	29
get_degplot . . . . .	29

get_gene_counts . . . . .	30
get_project_name_from_path . . . . .	31
get_upset_table . . . . .	32
get_y_init . . . . .	33
gs_radar . . . . .	33
heatmapmod . . . . .	35
helpmod . . . . .	36
helpModal . . . . .	37
horizonmod . . . . .	38
init_local_config . . . . .	39
install_carnation . . . . .	40
in_admin_group . . . . .	40
is_site_admin . . . . .	41
is_valid_pattern_obj . . . . .	42
loadmod . . . . .	42
makeEnrichResult . . . . .	43
make_example_carnation_object . . . . .	44
make_final_object . . . . .	45
maplotmod . . . . .	46
materialize_carnation_object . . . . .	47
metamod . . . . .	48
my.summary . . . . .	49
pcamod . . . . .	50
plotMA.label . . . . .	52
plotMA.label_ly . . . . .	53
plotPCA.ly . . . . .	54
plotPCA.san . . . . .	55
plotScatter.label . . . . .	56
plotScatter.label_ly . . . . .	58
radarmod . . . . .	61
read_access_yaml . . . . .	62
res_cell . . . . .	63
res_dex . . . . .	63
run_carnation . . . . .	64
savemod . . . . .	65
save_access_yaml . . . . .	66
scattermod . . . . .	67
settingsmod . . . . .	68
set_config . . . . .	70
summarize_res_list . . . . .	71
sumovmod . . . . .	73
top.genes . . . . .	74
upsetmod . . . . .	75
validate_carnation_object . . . . .	76

carnation-package      *carnation*

---

## Description

carnation is an interactive Shiny dashboard that makes complex bulk RNA-Seq data more accessible and intuitive, integrating all facets of bulk RNA-Seq analysis using three modules - differential expression analysis, functional enrichment and pattern analysis.

## Details

- Deeply explore analysis results from complex experiments using interactive plots.
- Easily keep track of genes of interest using the 'Gene scratchpad'.
- Use fuzzy search to filter and search functional enrichment results.
- Visualize complex patterns using highly customizable gene plot.
- Manage local data in single-user mode or deploy on a server to share with collaborators using in-built user management system.

Main function to run the app: `run_carnation()`

## Author(s)

**Maintainer:** Apratim Mitra <apratim.mitra@gmail.com> ([ORCID](#))

Other contributors:

- Matthew Tyler Menold <matthew.menold@gmail.com> ([ORCID](#)) [contributor]
- Ryan Dale <ryan.dale@nih.gov> ([ORCID](#)) [funder]

## See Also

Useful links:

- <https://nichd-bspc.github.io/carnation/>
- Report bugs at <https://github.com/NICHD-BSPC/carnation/issues>

---

add.set.column	<i>Add set column to UpSet plot matrix</i>
----------------	--

---

**Description**

This function adds a column denoting set number to a matrix generated for an upset plot with `fromList.with.names()`

**Usage**

```
add.set.column(df)
```

**Arguments**

`df` binary matrix where row = genes & columns are gene sets, with 1 indicating that a gene is present in that gene set and vice-versa

**Value**

data.frame with added set column

**Examples**

```
# list of genes
lst <- list(group1 = c(a = "gene1", b = "gene2", c = "gene3", d = "gene4"),
           group2 = c(c = "gene3", d = "gene4"))

# binarized matrix with group membership
df <- fromList.with.names(lst)

# matrix with added set column
ldf <- add.set.column(df)
```

---

add_metadata	<i>Add metadata to counts data frame</i>
--------------	--

---

**Description**

Add metadata to counts data frame

**Usage**

```
add_metadata(df, coldata, exclude.intgroups)
```

**Arguments**

df                    data.frame with gene counts  
 coldata             data.frame with metadata  
 exclude.intgroups   metadata columns to ignore

**Value**

counts data frame with added metadata

**Examples**

```
library(DESeq2)

# make example DESeq data set
dds <- makeExampleDESeqDataSet()

# extract counts and metadata
df <- assay(dds)
coldata <- colData(dds)

# get gene counts df
counts_df <- get_gene_counts(dds, paste0('gene', seq_len(10)))

# add metadata
counts_df <- add_metadata(counts_df, coldata, exclude.intgroups=NULL)
```

---

alluvialmod

*Alluvial plot module*


---

**Description**

UI & module to generate alluvial plots.

**Usage**

```
alluvialUI(id, panel)

alluvialServer(id, obj, res_obj, config)
```

**Arguments**

id                    Module id  
 panel                string, can be 'sidebar' or 'main'  
 obj                  reactiveValues object containing GeneTonic object  
 res\_obj              reactive, dataframe containing enrichment results  
 config                reactive list with config settings

**Value**

UI returns tagList with plot UI server invisibly returns NULL (used for side effects)

**Examples**

```
library(shiny)

# get DESeqResults object
data(res_dex, package='carnation')

# get enrichResult object
data(eres_dex, package='carnation')

# convert to GeneTonic object

gt <- GeneTonic::shake_enrichResult(eres_dex)

obj <- reactive({
  list(l_gs = gt$l_gs,
       anno_df = gt$anno_df,
       label = 'comp1')
})

res_obj <- reactive({ res })

config <- reactiveVal(get_config())

# run simple shiny app with plot
if(interactive()){
  shinyApp(
    ui = fluidPage(
      sidebarPanel(alluvialUI('p', 'sidebar')),
      mainPanel(alluvialUI('p', 'main'))
    ),
    server = function(input, output, session){
      alluvialServer('p', obj, res_obj, config)
    }
  )
}
```

---

 check\_user\_access

*Get data areas a user has access to*


---

**Description**

This function takes a username and returns a list with two elements:

**Usage**

```
check_user_access(al, u, admin = "admin")
```

**Arguments**

al	list with access settings; should have two elements - user_group & data_area
u	user name
admin	Admin user group

**Details**

user\_group: one element vector data\_area: vector of data areas

**Value**

list of user groups and data areas

**Examples**

```
# save access details to file
home <- Sys.getenv('HOME')

# create carnation data area if it doesn't exist
carnation_home <- file.path(home, 'carnation/data')
if(!dir.exists(carnation_home)) dir.create(carnation_home)

create_access_yaml(user = 'admin',
                  user_group = 'admin',
                  data_area = carnation_home)

# get current user access details
al <- read_access_yaml()

lst <- check_user_access(al, u='admin')
```

---

cnetmod

*Cnetplot module*

---

**Description**

UI & module to generate Cnetplots.

**Usage**

```
cnetPlotUI(id, panel)
```

```
cnetPlotServer(id, obj, config)
```

**Arguments**

id	Module id
panel	string, can be 'sidebar' or 'main'
obj	reactive, dataframe containing enrichment results
config	reactive list with config settings

**Value**

UI returns tagList with plot UI server invisibly returns NULL (used for side effects)

**Examples**

```
library(shiny)

# get DESeqResults object
data(res_dex, package='carnation')

obj <- reactive({ res })

config <- reactiveVal(get_config())

# run simple shiny app with plot
if(interactive()){
  shinyApp(
    ui = fluidPage(
      sidebarPanel(cnetPlotUI('p', 'sidebar')),
      mainPanel(cnetPlotUI('p', 'main'))
    ),
    server = function(input, output, session){
      cnetPlotServer('p', obj, config)
    }
  )
}
```

---

create\_access\_yaml      *Create access yaml*

---

**Description**

This function creates an access yaml file. This is primarily intended for the first run.

**Usage**

```
create_access_yaml(user, user_group, data_area)
```

**Arguments**

user	User name
user_group	User group
data_area	Path to data area containing RDS files

**Value**

Invisibly returns NULL. This function is primarily used for its side effect of saving a yaml file with access settings

**Examples**

```
# save access details to file
home <- Sys.getenv('HOME')

# create carnation data area if it doesn't exist
carnation_home <- file.path(home, 'carnation/data')
if(!dir.exists(carnation_home)) dir.create(carnation_home)

create_access_yaml(user = 'admin',
                  user_group = 'admin',
                  data_area = carnation_home)
```

---

degmod

*Pattern plot module*


---

**Description**

Module UI & server to generate pattern plots.

**Usage**

```
patternPlotUI(id, panel, tab)
```

```
patternPlotServer(id, obj, coldata, gene_scratchpad, upset_data, config)
```

**Arguments**

id	Module id
panel	string, can be 'sidebar' or 'main'
tab	string, if 'plot' show plot settings, if 'table' show table settings; if 'both', show settings for both.
obj	reactiveValues object containing carnation object
coldata	reactiveValues object containing object metadata

```

gene_scratchpad
    reactive containing genes selected in scratchpad
upset_data
    reactive containing list with data from upset plot module
config
    reactive list with config settings

```

## Value

UI returns tagList with module UI server returns reactive with selected genes for scratchpad updates

## Examples

```

library(shiny)
library(DESeq2)

# Create reactive values to simulate app state
oobj <- make_example_carnation_object()

obj <- reactiveValues(
  dds = oobj$dds,
  rld = oobj$rld,
  res = oobj$res,
  all_dds = oobj$all_dds,
  all_rld = oobj$all_rld,
  dds_mapping = oobj$dds_mapping
)

cdata <- lapply(oobj$rld, function(x) colData(x))

coldata <- reactiveValues( all=cdata, curr=cdata )

gene_scratchpad <- reactive({ c('gene1', 'gene2') })
upset_data <- reactive({ list(genes=NULL, labels=NULL) })

config <- reactiveVal(get_config())

shinyApp(
  ui = fluidPage(
    sidebarPanel(
      patternPlotUI('p', 'sidebar', 'both'),
      conditionalPanel(condition = "input.pattern_mode == 'Plot'",
        patternPlotUI('p', 'sidebar', 'plot')
      ),
      conditionalPanel(condition = "input.pattern_mode == 'Table'",
        patternPlotUI('p', 'sidebar', 'table')
      )
    ),
    mainPanel(
      tabsetPanel(id='pattern_mode',
        tabPanel('Plot',
          patternPlotUI('p', 'plot')
        ), # tabPanel plot

```

```

        tabPanel('Cluster membership',
                patternPlotUI('p', 'table')
        ) # tabPanel cluster_membership

    ) # tabsetPanel pattern_mode
) # tabPanel pattern_analysis
),
server = function(input, output, session){
    patternPlotServer('deg_plot', obj, coldata,
                      gene_scratchpad, upset_data, config)
    }
)

```

---

degpatterns_dex	<i>A degPatterns object for differentially expressed genes in the dexamethasone treatment comparison.</i>
-----------------	---

---

## Description

A degPatterns object for differentially expressed genes in the dexamethasone treatment comparison.

## Format

A degPatterns object, generated with the degPatterns function from the DEGreport package.

## Details

This degPatterns object was created to test for groups of coexpressed genes in the top 100 differentially expressed genes from the dexamethasone treatment comparison.

Details on how this object has been created are included in the create\_carnation\_data.R script, included in the scripts folder of the Carnation package.

## References

Himes BE, Jiang X, Wagner P, Hu R, Wang Q, Klanderma B, Whitaker RM, Duan Q, Lasky-Su J, Nikolos C, Jester W, Johnson M, Panettieri R Jr, Tantisira KG, Weiss ST, Lu Q. "RNA-Seq Transcriptome Profiling Identifies CRISPLD2 as a Glucocorticoid Responsive Gene that Modulates Cytokine Function in Airway Smooth Muscle Cells." PLoS One. 2014 Jun 13;9(6):e99625. PMID: 24926665. GEO: GSE52778

---

dendromod	<i>Dendrogram module</i>
-----------	--------------------------

---

**Description**

UI & module to generate dendrograms.

**Usage**

```
dendrogramUI(id, panel)
```

```
dendrogramServer(id, obj, config)
```

**Arguments**

id	Module id
panel	string, can be 'sidebar' or 'main'
obj	reactiveValues object containing GeneTonic object
config	reactive list with config settings

**Value**

UI returns tagList with plot UI server invisibly returns NULL (used for side effects)

**Examples**

```
library(shiny)

# get enrichResult object
data(eres_dex, package='carnation')

# convert to GeneTonic object
gt <- GeneTonic::shake_enrichResult(eres_dex)

obj <- reactive({
  list(l_gs = gt$l_gs,
       anno_df = gt$anno_df,
       label = 'comp1')
})

config <- reactiveVal(get_config())

# run simple shiny app with plot
if(interactive()){
  shinyApp(
    ui = fluidPage(
      sidebarPanel(dendrogramUI('p', 'sidebar')),
      mainPanel(dendrogramUI('p', 'main'))
    )
  )
}
```

```

    ),
    server = function(input, output, session){
      dendrogramServer('p', obj, config)
    }
  )
}

```

---

distillmod

*Distilled enrichment map module*


---

### Description

UI & module to generate distill enrichment map plots.

### Usage

```
distillPlotUI(id, panel)
```

```
distillPlotServer(id, obj, args, config)
```

### Arguments

id	Module id
panel	string, can be 'sidebar' or 'main'
obj	reactive containing 'distilled' enrichment results
args	reactive, list with plot arguments, 'numcat' (number of categories to plot)
config	reactive list with config settings

### Value

UI returns tagList with plot UI server returns reactive with number of plotted terms

### Examples

```

library(GeneTonic)
library(shiny)

# get DESeqResults object
data(res_dex, package='carnation')

# get enrichResult object
data(eres_dex, package='carnation')

# preprocess & convert to GeneTonic object
eres2 <- GeneTonic::shake_enrichResult(eres_dex)
gt <- enrich_to_genetonic(eres_dex, res_dex)

```

```

# get distilled results
df <- distill_enrichment(
  eres2,
  res_dex,
  gt$anno_df,
  n_gs = 10,
  cluster_fun = "cluster_markov"
)

# number of plotted terms
args <- reactive({ list(numcat=10) })

config <- reactiveVal(get_config())

# run simple shiny app with plot
if(interactive()){
  shinyApp(
    ui = fluidPage(
      sidebarPanel(distillPlotUI('p', 'sidebar')),
      mainPanel(distillPlotUI('p', 'main'))
    ),
    server = function(input, output, session){
      numcat <- observe({
        distillPlotServer('p',
                          reactive({ df })),
                          args,
                          config)
      })
    }
  )
}

```

---

dlmod

*Download button module*


---

## Description

Module UI & server for download buttons.

## Usage

```
downloadButtonUI(id)
```

```
downloadButtonServer(id, outplot, plot_type)
```

## Arguments

id	Module id
outplot	reactive plot handle
plot_type	reactive/static value used for output filename

**Value**

UI returns tagList with download button UI. Server invisibly returns NULL (used for side effects).

**Examples**

```
library(shiny)
library(ggplot2)

# get example object
obj <- make_example_carnation_object()
res <- as.data.frame(obj$res[[1]])

# make MA plot
p <- ggplot(res, aes(x=baseMean, y=log2foldChange)) +
  geom_point(color='black', alpha=0.5)

outplot <- reactive({ p })

# app with a single button to download a plot
if(interactive()){
  shinyApp(
    ui = fluidPage(
      downloadButtonUI('p')
    ),
    server = function(input, output, session){
      downloadButtonServer('p', outplot, 'maplot')
    }
  )
}
```

---

 dummy\_genetonic

*Make dummy GeneTonic object*


---

**Description**

Make dummy GeneTonic object

**Usage**

```
dummy_genetonic(eres)
```

**Arguments**

eres                    enrichResult object

**Value**

GeneTonic object

---

emapmod

*Enrichment map plot module*

---

## Description

UI & module to generate enrichment map plots.

## Usage

```
enrichmapUI(id, panel)
```

```
enrichmapServer(id, obj, res_obj, config)
```

## Arguments

id	Module id
panel	string, can be 'sidebar' or 'main'
obj	reactiveValues object containing GeneTonic object
res_obj	reactive, dataframe containing enrichment results
config	reactive list with config settings

## Value

UI returns tagList with plot UI server invisibly returns NULL (used for side effects)

## Examples

```
library(shiny)

# get DESeqResults object
data(res_dex, package='carnation')

# get enrichResult object
data(eres_dex, package='carnation')

# convert to GeneTonic object
gt <- GeneTonic::shake_enrichResult(eres_dex)

obj <- reactive({
  list(l_gs = gt$l_gs,
       anno_df = gt$anno_df,
       label = 'comp1')
})

res_obj <- reactive({ res })

config <- reactiveVal(get_config())
```

```
# run simple shiny app with plot
if(interactive()){
  shinyApp(
    ui = fluidPage(
      sidebarPanel(enrichmapUI('p', 'sidebar')),
      mainPanel(enrichmapUI('p', 'main'))
    ),
    server = function(input, output, session){
      enrichmapServer('p', obj, res_obj, config)
    }
  )
}
```

---

enrich\_to\_genetonic    *Convert enrichResult to GeneTonic object*

---

### Description

This function takes an enrichResult object and DE analysis results and creates a GeneTonic object.

### Usage

```
enrich_to_genetonic(enrich, res)
```

### Arguments

enrich	enrichResult object
res	data frame with DE analysis results

### Value

GeneTonic object

### Examples

```
# get enrich & res objects
data(res_dex, package="carnation")
data(eres_dex, package="carnation")

# convert to GeneTonic object
gt <- enrich_to_genetonic(eres_dex, res_dex)
```

---

eres_cell	<i>An enrichResult object for differentially expressed genes in the cell line comparison.</i>
-----------	---

---

**Description**

An `enrichResult` object for differentially expressed genes in the cell line comparison.

**Format**

An `enrichResult` object, generated with the `enrichGO` function from the `clusterProfiler` package.

**Details**

This `enrichResult` object was created to test for functional enrichment using the GO Biological Process (BP) ontology on the top 100 differentially expressed genes from the cell line comparison.

Details on how this object has been created are included in the `create_carnation_data.R` script, included in the `scripts` folder of the `Carnation` package.

**References**

Himes BE, Jiang X, Wagner P, Hu R, Wang Q, Klanderma B, Whitaker RM, Duan Q, Lasky-Su J, Nikolos C, Jester W, Johnson M, Panettieri R Jr, Tantisira KG, Weiss ST, Lu Q. "RNA-Seq Transcriptome Profiling Identifies CRISPLD2 as a Glucocorticoid Responsive Gene that Modulates Cytokine Function in Airway Smooth Muscle Cells." *PLoS One*. 2014 Jun 13;9(6):e99625. PMID: 24926665. GEO: GSE52778

---

eres_dex	<i>An enrichResult object for differentially expressed genes in the dexamethasone treatment comparison.</i>
----------	---

---

**Description**

An `enrichResult` object for differentially expressed genes in the dexamethasone treatment comparison.

**Format**

An `enrichResult` object, generated with the `enrichGO` function from the `clusterProfiler` package.

## Details

This enrichResult object was created to test for functional enrichment using the GO Biological Process (BP) ontology on the top 100 differentially expressed genes from the dexamethasone treatment comparison.

Details on how this object has been created are included in the create\_carnation\_data.R script, included in the scripts folder of the Carnation package.

## References

Himes BE, Jiang X, Wagner P, Hu R, Wang Q, Klanderman B, Whitaker RM, Duan Q, Lasky-Su J, Nikolos C, Jester W, Johnson M, Panettieri R Jr, Tantisira KG, Weiss ST, Lu Q. "RNA-Seq Transcriptome Profiling Identifies CRISPLD2 as a Glucocorticoid Responsive Gene that Modulates Cytokine Function in Airway Smooth Muscle Cells." PLoS One. 2014 Jun 13;9(6):e99625. PMID: 24926665. GEO: GSE52778

---

format_genes	<i>format gene names to look pretty in table output</i>
--------------	---

---

## Description

This function works by grouping long lists of genes into groups of a specified size. Each group is collapsed using commas, while groups are separated by spaces so that datatable formatting is tricked into separating space-separated groups and not comma-separated groups

## Usage

```
format_genes(g, sep = "\\/", genes.per.line = 6)
```

## Arguments

g	vector of gene names
sep	gene name separator
genes.per.line	number of genes to show in a line

## Value

vector of gene names prettified for data.table output

## Examples

```
# string with genes separated by '/'
g <- "gene1/gene2/gene3/gene4/gene5/gene6/gene7"

gg <- format_genes(g, genes.per.line=3)
```

---

fromList.with.names     *Prepare list for UpSet plots, but include rownames*

---

**Description**

Prepare list for UpSet plots, but include rownames

**Usage**

```
fromList.with.names(lst)
```

**Arguments**

lst                    List of sets to compare (same input as to UpSetR::fromList)

**Value**

data.frame of 1 and 0 showing which genes are in which sets

**Examples**

```
# list of genes
lst <- list(group1 = c(a = "gene1", b = "gene2", c = "gene3", d = "gene4"),
            group2 = c(c = "gene3", d = "gene4"))

# binarized matrix with group membership
df <- fromList.with.names(lst)
```

---

funenrichmod             *Functional enrichment module*

---

**Description**

UI & module to show functional enrichment tables & plots.

**Usage**

```
enrichUI(id, panel, tab = "none")
```

```
enrichServer(id, obj, upset_table, gene_scratchpad, reset_genes, config)
```

**Arguments**

id	ID string used to match the ID used to call the module UI function
panel	string, can be 'sidebar' or 'main'
tab	string, if 'table' show table settings, if 'plots' show plot settings; if 'compare_results', show comparison settings.
obj	reactiveValues object containing carnation object
upset_table	reactive, data from upset plot module
gene_scratchpad	
	reactive, genes selected in gene scratchpad
reset_genes	reactive to reset genes in scratchpad
config	reactive list with config settings

**Value**

UI returns tagList with plot UI server returns reactive with gene selected from functional enrichment tables.

**Examples**

```
library(shiny)
library(DESeq2)

# Create reactive values to simulate app state
oobj <- make_example_carnation_object()

obj <- reactiveValues(
  dds = oobj$dds,
  rld = oobj$rld,
  res = oobj$res,
  all_dds = oobj$all_dds,
  all_rld = oobj$all_rld,
  dds_mapping = oobj$dds_mapping
)

upset_table <- reactiveValues(tbl=NULL, intersections=NULL, set_labels=NULL)

gene_scratchpad <- reactive({ c('gene1', 'gene2') })

config <- reactiveVal(get_config())

shinyApp(
  ui = fluidPage(
    sidebarPanel(
      conditionalPanel(condition = "input.func == 'Table'",
        enrichUI('p', 'sidebar', 'table')
      ),
      conditionalPanel(condition = "input.func == 'Plot'",
        enrichUI('p', 'sidebar', 'plot')
      ),
    ),

```

```

        conditionalPanel(condition = "input.func == 'Compare results'",
          enrichUI('p', 'sidebar', 'compare_results')
        )
      ),
      mainPanel(
        tabsetPanel(id='func',
          tabPanel('Table',
            enrichUI('p', 'main', 'table')
          ), # tabPanel table

          tabPanel('Plot',
            enrichUI('p', 'main', 'plot')
          ), # tabPanel plot

          tabPanel('Compare results',
            enrichUI('p', 'main', 'compare_results')
          ) # tabPanel compare_results

        ) # tabsetPanel func
      ) # tabPanel
    ),
    server = function(input, output, session){
      enrich_data <- enrichServer('p', obj,
        upset_table,
        gene_scratchpad,
        reactive({ FALSE }),
        config)
    }
  )
)

```

---

fuzzymod

*Fuzzy enrichment map module*


---

## Description

UI & module to generate fuzzy enrichment map plots.

## Usage

```
fuzzyPlotUI(id, panel)
```

```
fuzzyPlotServer(id, obj, args, config)
```

## Arguments

id	Module id
panel	string, can be 'sidebar' or 'main'
obj	reactive containing 'distilled' enrichment results

args                reactive, list with plot arguments, 'numcat' (number of categories to plot)  
 config             reactive list with config settings

### Value

UI returns tagList with plot UI server returns reactive with number of plotted terms

### Examples

```
library(shiny)

# get enrichResult object
data(eres_dex, package='carnation')

# preprocess & convert to GeneTonic object
gt <- GeneTonic::shake_enrichResult(eres_dex)

# get distilled results
df <- GeneTonic::gs_fuzzyclustering(gt[seq_len(10)],,
  similarity_threshold = 0.35,
  fuzzy_seeding_initial_neighbors = 3,
  fuzzy_multilinkage_rule = 0.5)

# number of plotted terms
args <- reactive({ list(numcat=10) })

config <- reactiveVal(get_config())

# run simple shiny app with plot
if(interactive()){
  shinyApp(
    ui = fluidPage(
      sidebarPanel(fuzzyPlotUI('p', 'sidebar')),
      mainPanel(fuzzyPlotUI('p', 'main'))
    ),
    server = function(input, output, session){
      numcat <- observe({
        fuzzyPlotServer('p',
          reactive({ df }),
          args,
          config)
      })
    }
  )
}
```

**Description**

UI & server for module to create gene plot

**Usage**

```
genePlotUI(id, panel)
```

```
genePlotServer(id, obj, coldata, plot_args, config)
```

**Arguments**

id	Module id
panel	string, can be 'sidebar' or 'main'
obj	reactiveValues object containing carnation object
coldata	reactiveValues object containing object metadata
plot_args	reactive list with 3 elements: 'gene.id' (all gene IDs) & 'gene_scratchpad' (genes selected in scratchpad) & 'comp_all' (selected comparison)
config	reactive list with config settings

**Value**

UI returns tagList with gene plot UI. Server invisibly returns NULL (used for side effects).

**Examples**

```
library(shiny)
library(DESeq2)

# Create reactive values to simulate app state
oobj <- make_example_carnation_object()

obj <- reactiveValues(
  dds = oobj$dds,
  rld = oobj$rld,
  res = oobj$res,
  all_dds = oobj$all_dds,
  all_rld = oobj$all_rld,
  dds_mapping = oobj$dds_mapping
)

# Set up coldata structure that the module expects
coldata <- reactiveValues(
  curr = list(
    all_samples = colData(oobj$dds$main),
    main = colData(oobj$dds$main)
  )
)

plot_args <- reactive({
```

```

list(
  gene.to.plot = c("gene1", "gene2"),
  gene.id = rownames(obj$dds$main),
  comp_all = "comp1"
)
})

config <- reactiveVal(get_config())

shinyApp(
  ui = fluidPage(
    sidebarPanel(genePlotUI('p', 'sidebar')),
    mainPanel(genePlotUI('p', 'main'))
  ),
  server = function(input, output, session){
    genePlotServer('p', obj, coldata, plot_args, config)
  }
)

```

---

getcountplot

*Create gene plot*


---

## Description

This function creates the gene plot.

## Usage

```

getcountplot(
  df,
  intgroup = "group",
  factor.levels,
  title = NULL,
  ylab = "Normalized counts",
  color = "gene",
  nrow = 2,
  ymin = NULL,
  ymax = NULL,
  log = TRUE,
  freey = FALSE,
  trendline = "smooth",
  facet = NULL,
  legend = TRUE,
  boxes = TRUE,
  rotate_x_labels = 30
)

```

**Arguments**

<code>df</code>	data.frame with gene counts
<code>intgroup</code>	metadata variable to plot on x-axis
<code>factor.levels</code>	levels of intgroup to show on x-axis
<code>title</code>	title of plot
<code>ylab</code>	y-axis label
<code>color</code>	metadata variable to color by
<code>nrow</code>	number of rows to plot if faceting
<code>ymin</code>	y-axis lower limit
<code>ymax</code>	y-axis upper limit
<code>log</code>	should y-axis be log10-transformed?
<code>freey</code>	should y-axes of faceted plots have independent scales?
<code>trendline</code>	type of trendline to draw
<code>facet</code>	metadata variable to facet by
<code>legend</code>	show legend?
<code>boxes</code>	show boxes?
<code>rotate_x_labels</code>	angle to rotate x-axis labels (default=30)

**Value**

ggplot handle

**Examples**

```
# make example DESeq dataset
dds <- DESeq2::makeExampleDESeqDataSet()

# get gene counts
df <- get_gene_counts(dds, gene = c('gene1', 'gene2'))

# standard gene plot
p <- getcountplot(df, intgroup = "condition", factor.levels = c("A", "B"))

# with genes faceted
p1 <- getcountplot(df, intgroup = "condition", factor.levels = c("A", "B"), facet = "gene")
```

---

get_access_path	<i>Get path to access yaml file</i>
-----------------	-------------------------------------

---

**Description**

This function checks for an environment variable 'CARNATION\_ACCESS\_YAML' to specify directory to save access yaml. If env variable does not exist uses home directory as save location.

**Usage**

```
get_access_path()
```

**Value**

path to access yaml

**Examples**

```
p <- get_access_path()
```

---

get_config	<i>Get config</i>
------------	-------------------

---

**Description**

This function reads the bundled package config and returns it. If a local config yaml exists, only supported user-editable settings are merged into the returned config.

**Usage**

```
get_config(config_path = NULL)
```

**Arguments**

config\_path      optional path to a local config yaml. If NULL, uses the path returned by get\_config\_path().

**Value**

list containing config items

**Examples**

```
cfg <- get_config()
```

---

get_config_path	<i>Get path to local config yaml file</i>
-----------------	---

---

**Description**

This function checks for an environment variable CARNATION\_CONFIG\_YAML to specify the local config yaml path. If the variable is not set, a default path in the home directory is used.

**Usage**

```
get_config_path()
```

**Value**

path to local config yaml

**Examples**

```
p <- get_config_path()
```

---

get_degplot	<i>Plot a degPatterns object</i>
-------------	----------------------------------

---

**Description**

This function plots a degPatterns object.

**Usage**

```
get_degplot(  
  obj,  
  time,  
  color = NULL,  
  cluster_column = "cluster",  
  cluster_to_show,  
  x_order,  
  points = TRUE,  
  boxes = TRUE,  
  smooth = "smooth",  
  lines = TRUE,  
  facet = TRUE,  
  prefix_title = "Cluster ",  
  genes_to_label = NULL  
)
```

**Arguments**

obj	degPatterns object
time	metadata variable to plot on x-axis
color	variable to color plot
cluster_column	column to use for grouping genes
cluster_to_show	which clusters to show in plot
x_order	order of x-axis values
points	boolean, show samples on plot? Default: TRUE
boxes	boolean, show boxes on plot? Default: TRUE
smooth	what type of trendline to use? can be 'smooth' (default) or 'line'.
lines	show lines joining samples? Default: TRUE
facet	boolean, should plot be faceted? Default: TRUE
prefix_title	string, prefix for facet titles
genes_to_label	genes to label on plot

**Value**

ggplot handle

**Examples**

```
# get degpatterns object
data(degpatterns_dex, package = 'carnation')

# get pattern plot
all_clusters <- unique(degpatterns_dex$normalized$cluster)

dp <- get_degplot(degpatterns_dex, time='dex',
                  cluster_to_show=all_clusters,
                  x_order=c('untrt','trt'))
```

---

get\_gene\_counts      *Get read counts for gene*

---

**Description**

This is a simple function to obtain read counts for a specified gene, based on the DESeq2::plotCounts function.

**Usage**

```
get_gene_counts(dds, gene, intgroup = "condition", norm_method = "libsize")
```

**Arguments**

dds	DESeqDataSet object
gene	gene name vector
intgroup	metadata variable to attach to counts
norm_method	normalization method, can be 'libsize' (default) or 'vst'

**Value**

data.frame with gene counts

**Examples**

```
# make example DESeq data set
dds <- DESeq2::makeExampleDESeqDataSet()

# get counts for gene1
gg <- get_gene_counts(dds, 'gene1')
```

---

get\_project\_name\_from\_path  
*Get project name from path*

---

**Description**

This function takes in a path to an RDS file and returns a string to be used as project name

**Usage**

```
get_project_name_from_path(
  x,
  depth = 2,
  end_offset = 0,
  staging_dir = "dev",
  fsep = .Platform$file.sep
)
```

**Arguments**

x	character path to RDS file
depth	integer how many levels below path to look?
end_offset	integer how far from the end of path to end?
staging_dir	name of staging directory
fsep	file separator to split path with

**Value**

project name parsed from path to object

**Examples**

```
# path to carnation object
obj_path <- "/path/to/project/test/main.rnaseq.rds"

# parsed project name
get_project_name_from_path(obj_path, depth = 2, end_offset = 0)
```

---

get_upset_table	<i>Generate upset plot table</i>
-----------------	----------------------------------

---

**Description**

Generate upset plot table

**Usage**

```
get_upset_table(gene.lists, comp_split_pattern = ";")
```

**Arguments**

gene.lists      list with character vectors of gene names  
comp\_split\_pattern  
                 character used to separate gene set names

**Value**

list with upset table elements

**Examples**

```
lst <- list(group1 = c(a = "gene1", b = "gene2", c = "gene3", d = "gene4"),
           group2 = c(b = "gene2", d = "gene4", e = "gene5"),
           group3 = c(d = "gene4", e = "gene5", f = "gene6"))

df <- get_upset_table(lst)
str(df)
```

---

get_y_init	<i>Get initial y-axis limits</i>
------------	----------------------------------

---

**Description**

Get initial y-axis limits

**Usage**

```
get_y_init(df, y_delta, pseudocount)
```

**Arguments**

df	data.frame with counts. Must have column 'count'
y_delta	y-axis padding for visualization, must be between 0 and 1
pseudocount	pseudo-count to add to the data.frame

**Value**

min and max limits for count column, padded for visualization

**Examples**

```
# make example DESeq dataset
dds <- DESeq2::makeExampleDESeqDataSet()

# get gene counts
df <- get_gene_counts(dds, gene = c('gene1', 'gene2'))

# get y axis limits
get_y_init(df, y_delta = 0.01, pseudocount = 1)
```

---

gs_radar	<i>Radar plot</i>
----------	-------------------

---

**Description**

This is a copy of gs\_radar from GeneTonic where the labels of gene sets are converted to parameters

**Usage**

```
gs_radar(
  res_enrich,
  res_enrich2 = NULL,
  label1 = "scenario 1",
  label2 = "scenario 2",
  n_gs = 20,
  p_value_column = "gs_pvalue"
)
```

**Arguments**

res_enrich	GeneTonic object for comparison 1
res_enrich2	GeneTonic object for comparison 2 (default = NULL)
label1	label for comparison 1
label2	label for comparison 2
n_gs	number of gene sets (default = 20)
p_value_column	column to use as p-value (default = 'gs_pvalue')

**Value**

ggplot handle

**Examples**

```
library(GeneTonic)

# get DESeqResults object
data(res_dex, package='carnation')

# get enrichResult object
data(eres_dex, package='carnation')

# convert to GeneTonic object
gt <- shake_enrichResult(eres_dex)

# get annotation df
idx <- match(c('gene','symbol'), tolower(colnames(res_dex)))
anno_df <- res_dex[,idx]
colnames(anno_df) <- c('gene_id', 'gene_name')

# add aggregate score columns
gt <- get_aggrscores(gt, res_dex, anno_df)

# make radar plot
p <- gs_radar(gt)
```

---

heatmapmod	<i>Heatmap module</i>
------------	-----------------------

---

## Description

Module UI & server to generate heatmap.

## Usage

```
heatmapUI(id, panel)
```

```
heatmapServer(id, obj, coldata, plot_args, gene_scratchpad, config)
```

## Arguments

id	Module id
panel	string, can be 'sidebar' or 'main'
obj	reactiveValues object containing carnation object
coldata	reactiveValues object containing object metadata
plot_args	reactive containing 'fdr.thres' (padj threshold), 'fc.thres' (log2FC) & 'upset_data' (list containing data from upset plot module)
gene_scratchpad	reactiveValues object containing genes selected in scratchpad which will be labeled
config	reactive list with config settings

## Value

UI returns tagList with heatmap UI. Server invisibly returns NULL (used for side effects).

## Examples

```
library(shiny)
library(DESeq2)

# Create reactive values to simulate app state
oobj <- make_example_carnation_object()

obj <- reactiveValues(
  dds = oobj$dds,
  rld = oobj$rld,
  res = oobj$res,
  all_dds = oobj$all_dds,
  all_rld = oobj$all_rld,
  dds_mapping = oobj$dds_mapping
)
```

```

cdata <- lapply(oobj$rld, function(x) colData(x))

coldata <- reactiveValues( all=cdata, curr=cdata )

plot_args <- reactive({
  list(
    fdr.thres=0.1,
    fc.thres=0,
    upset_data=list(genes=NULL, labels=NULL)
  )
})

gene_scratchpad <- reactive({ c('gene1', 'gene2') })

config <- reactiveVal(get_config())

shinyApp(
  ui = fluidPage(
    sidebarPanel(heatmapUI('p', 'sidebar')),
    mainPanel(heatmapUI('p', 'sidebar'))
  ),
  server = function(input, output, session){
    heatmapServer('p', obj, coldata,
                  plot_args, gene_scratchpad, config)
  }
)

```

---

 helpmod

*Help button module*


---

## Description

Module UI & server for help buttons.

## Usage

```
helpButtonUI(id)
```

```
helpButtonServer(id, ...)
```

## Arguments

<code>id</code>	Module id. This also doubles as prefixes for help text files.
<code>...</code>	other params passed to helpModal()

## Value

UI returns tagList with help button UI. Server invisibly returns NULL (used for side effects).

## Examples

```
library(shiny)

# app with a single help button to show DE summary table details
if(interactive()){
  shinyApp(
    ui = fluidPage(
      helpButtonUI('de_summary_help')
    ),
    server = function(input, output, session){
      helpButtonServer('de_summary_help')
    }
  )
}
```

---

helpModal

*Help modal*

---

## Description

This generates a modal dialog that includes text from a markdown file.

## Usage

```
helpModal(mdfile, title = NULL, ...)
```

## Arguments

mdfile	path to markdown file
title	Title of modal dialog
...	other params passed to modalDialog()

## Value

Modal dialog with help documentation.

---

horizonmod	<i>Horizon plot module</i>
------------	----------------------------

---

## Description

UI & module to generate horizon plots.

## Usage

```
horizonUI(id, panel)
```

```
horizonServer(id, obj, config)
```

## Arguments

id	Module id
panel	string, can be 'sidebar' or 'main'
obj	reactiveValues object containing two GeneTonic objects
config	reactive list with config settings

## Value

UI returns tagList with plot UI server invisibly returns NULL (used for side effects)

## Examples

```
library(shiny)

# get enrichResult object
data(eres_dex, package='carnation')

# convert to GeneTonic object
gt <- GeneTonic::shake_enrichResult(eres_dex)

# get second enrichResult object
data(eres_cell, package='carnation')

# convert to GeneTonic object
gt1 <- GeneTonic::shake_enrichResult(eres_cell)

obj <- reactive({
  list(
    obj1 = list(l_gs = gt$l_gs,
               anno_df = gt$anno_df,
               label = 'comp1'),
    obj2 = list(l_gs = gt1$l_gs,
               anno_df = gt1$anno_df,
               label = 'comp2')
```

```
    )
  })

  config <- reactiveVal(get_config())

  # run simple shiny app with plot
  if(interactive()){
    shinyApp(
      ui = fluidPage(
        sidebarPanel(horizonUI('p', 'sidebar')),
        mainPanel(horizonUI('p', 'main'))
      ),
      server = function(input, output, session){
        horizonServer('p', obj, config)
      }
    )
  }
}
```

---

init_local_config	<i>Initialize local config</i>
-------------------	--------------------------------

---

## Description

This function copies the bundled package config to a user-writable local config yaml. This is intended for users who want to customize the supported config settings without editing the installed package.

## Usage

```
init_local_config(config_path = get_config_path(), overwrite = FALSE)
```

## Arguments

config_path	path to the local config yaml to create. Defaults to get_config_path().
overwrite	logical indicating whether to overwrite an existing file.

## Value

Path to the local config yaml, invisibly.

## Examples

```
cfg_out <- tempfile(fileext = ".yaml")
init_local_config(cfg_out)
```

---

install_carnation	<i>Create carnation python environment</i>
-------------------	--

---

**Description**

This function installs 'plotly' and 'kaleido' python packages in an environment to allow PDF downloads from plotly plots.

**Usage**

```
install_carnation(envname, ...)
```

**Arguments**

envname	name of the python environment
...	parameters passed to reticulate::py_install

**Value**

NULL, invisibly. The function is called for its side effects.

**Examples**

```
if(interactive()){  
  install_carnation()  
}
```

---

in_admin_group	<i>is user is in admin group?</i>
----------------	-----------------------------------

---

**Description**

is user is in admin group?

**Usage**

```
in_admin_group(u)
```

**Arguments**

u	username
---	----------

**Value**

TRUE/FALSE to indicate if the user is part of the admin group

**Examples**

```
# save access details to file
home <- Sys.getenv('HOME')

# create carnation data area if it doesn't exist
carnation_home <- file.path(home, 'carnation/data')
if(!dir.exists(carnation_home)) dir.create(carnation_home)

create_access_yaml(user = 'admin',
                  user_group = 'admin',
                  data_area = carnation_home)

check <- in_admin_group('user')
```

---

is_site_admin	<i>is user an admin?</i>
---------------	--------------------------

---

**Description**

is user an admin?

**Usage**

```
is_site_admin(u)
```

**Arguments**

u	username
---	----------

**Value**

boolean to indicate is user is in admin group

**Examples**

```
# check if default user is admin
yy <- is_site_admin(u='admin')
```

`is_valid_pattern_obj` *Validate Pattern Analysis Object Schema*

---

### Description

Validate the schema for a single degpatterns analysis element used by the pattern analysis module.

### Usage

```
is_valid_pattern_obj(pattern_obj, require_symbol = FALSE)
```

### Arguments

`pattern_obj` A single pattern analysis element. Must be either a `data.frame` or a list containing a normalized `data.frame`.

`require_symbol` Logical, if TRUE require a `symbol` column in the analysis table.

### Value

Returns TRUE when validation succeeds, otherwise returns FALSE after emitting a message describing the issue.

### Examples

```
data(degpatterns_dex, package = "carnation")  
  
is_valid_pattern_obj(degpatterns_dex)
```

---

`loadmod` *Load data module*

---

### Description

Module UI & server to load new data

### Usage

```
loadDataUI(id)  
  
loadDataServer(id, username, config, rds = NULL)
```

**Arguments**

id	Module id
username	user name
config	reactive list with config settings
rds	Object to be edited

**Value**

UI returns tagList with module UI Server returns reactive with app reload trigger

**Examples**

```
library(shiny)

username <- 'admin'

config <- reactiveVal(get_config())

obj <- make_example_carnation_object()

rds <- reactive({ obj=obj })

shinyApp(
  ui = fluidPage(
    loadDataUI('p')
  ),
  server = function(input, output, session){
    loadDataServer('p', username=username, config, rds)
  }
)
```

---

makeEnrichResult      *Make an enrichResult obj from a data frame*

---

**Description**

Most of the parameters are just placeholders and the dataframe must contain the columns 'ID' and 'geneID'

**Usage**

```
makeEnrichResult(
  df,
  split = "/",
  keytype = "UNKNOWN",
  ontology = "UNKNOWN",
```

```
    type = "enrichResult"  
  )
```

### Arguments

df	data frame with functional enrichment results
split	string, character used to split gene IDs
keytype	type of gene ID
ontology	ontology database being used
type	string, can be 'enrichResult' or 'gseaResult'

### Value

enrichResult object

### Examples

```
# get enrichResult object  
data(eres_dex, package='carnation')  
  
# extract the results  
df <- as.data.frame(eres_dex)  
  
# convert to a stripped down enrichResult object  
eres2 <- makeEnrichResult(df)
```

---

```
make_example_carnation_object  
    Make example carnation object
```

---

### Description

Returns example carnation object used in examples & testing

### Usage

```
make_example_carnation_object()
```

### Value

reactiveValues object containing carnation object

### Examples

```
obj <- make_example_carnation_object()
```

---

make_final_object	<i>Make final object for internal use by the app</i>
-------------------	--

---

## Description

This function takes an uploaded object and sanitizes it to make sure it is suitable for internal use along with other additions:

- adds a 'dds\_mapping' element that maps dds\_list keys to res\_list objects.
- if there are multiple dds\_list objects, it adds a 'all\_dds' element combining all samples.

## Usage

```
make_final_object(obj)
```

## Arguments

obj                    list object containing lists of DE analysis results, functional enrichment objects, pattern analysis objects & raw and normalized counts objects.

## Value

final carnation object with additional pre-processing

## Examples

```
library(DESeq2)

# make example DESeq dataset
dds <- makeExampleDESeqDataSet()

# run DE analysis
dds <- DESeq(dds)

# extract comparison of interest
res <- results(dds, contrast = c("condition", "A", "B"))

# perform VST normalization
rld <- varianceStabilizingTransformation(dds, blind = TRUE)

# build minimal object
obj <- list(
  res_list = list(
    comp = list(
      res = res,
      dds = "main",
      label = "A vs B"
    )
  ),
)
```

```

        dds_list = list(main = dds),
        rld_list = list(main = rld)
    )

# final object
final_obj <- make_final_object(obj)

```

---

maplotmod

*MA plot module*


---

## Description

UI & server for module to create MA plot

## Usage

```
maPlotUI(id, panel)
```

```
maPlotServer(id, obj, plot_args, config)
```

## Arguments

id	Module id
panel	string, can be 'sidebar' or 'main'
obj	reactiveValues object containing carnation object
plot_args	reactive containing 'fdr.thres' (padj threshold), 'fc.thres' (log2FC threshold) & 'gene.to.plot' (genes selected in scratchpad)
config	reactive list with config settings

## Value

UI returns tagList with MA plot UI. Server invisibly returns NULL (used for side effects).

## Examples

```

library(shiny)
library(DESeq2)

# Create reactive values to simulate app state
oobj <- make_example_carnation_object()

obj <- reactiveValues(
  dds = oobj$dds,
  rld = oobj$rld,
  res = oobj$res,
  all_dds = oobj$all_dds,
  all_rld = oobj$all_rld,

```

```

    dds_mapping = oobj$dds_mapping
  )

# Set up coldata structure that the module expects
coldata <- reactiveValues(
  curr = list(
    all_samples = colData(oobj$dds$main),
    main = colData(oobj$dds$main)
  )
)

plot_args <- reactive({
  list(
    fdr.thres=0.1,
    fc.thres=0,
    gene.to.plot=c('gene1', 'gene2')
  )
})

config <- reactiveVal(get_config())

shinyApp(
  ui = fluidPage(
    sidebarPanel(maPlotUI('p', 'sidebar')),
    mainPanel(maPlotUI('p', 'main'))
  ),
  server = function(input, output, session){
    maPlotServer('p', obj, plot_args, config)
  }
)

```

---

materialize\_carnation\_object

*Materialize expensive carnation object components*

---

## Description

This function materializes expensive derived pieces for a validated carnation object, including DE-SeqDataSet creation from raw count matrices, variance-stabilized counts, and GeneTonic conversions.

## Usage

```
materialize_carnation_object(obj, config = NULL, cores = NULL)
```

## Arguments

obj	A validated object returned by <code>validate_carnation_object()</code> or <code>validate_loaded_carnation_obje</code>
config	Optional config list. If NULL, will use <code>get_config()</code> .
cores	Optional number of worker processes. If NULL, uses <code>config\$server\$cores</code> .

**Value**

The input object with materialized `dds_list`, `rld_list`, and optional genenetic slots.

**Examples**

```
# Minimal example with DE results and counts
library(DESeq2)

# Create example data
dds <- makeExampleDESeqDataSet()
dds <- DESeq(dds)
res <- results(dds, contrast = c("condition", "A", "B"))
rld <- varianceStabilizingTransformation(dds, blind = TRUE)

# Validate object inputs
obj <- validate_carnation_object(
  res_list = list(
    comp1 = list(
      res = as.data.frame(res),
      dds = "main",
      label = "A vs B"
    )
  ),
  dds_list = list(main = dds),
  rld_list = list(main = rld)
)

materialized <- materialize_carnation_object(obj, cores = 1)
```

---

metamod

*Metadata module*


---

**Description**

This module generates the metadata tab that allows users to view the metadata associated with the loaded carnation object.

**Usage**

```
metadataUI(id, panel)

metadataServer(id, obj, cols.to.drop)
```

**Arguments**

<code>id</code>	Module id
<code>panel</code>	context for generating ui elements ('sidebar' or 'main')
<code>obj</code>	reactiveValues object containing carnation object
<code>cols.to.drop</code>	columns to hide from table

**Value**

UI returns tagList with metadata UI. Server returns reactive object with metadata.

**Examples**

```
library(shiny)

# Create reactive values to simulate app state
oobj <- make_example_carnation_object()

obj <- reactiveValues(
  dds = oobj$dds,
  rld = oobj$rld,
  res = oobj$res,
  all_dds = oobj$all_dds,
  all_rld = oobj$all_rld,
  dds_mapping = oobj$dds_mapping
)

config <- get_config()
cols.to.drop <- config$server$cols.to.drop

shinyApp(
  ui = fluidPage(
    sidebarPanel(metadataUI('p', 'sidebar')),
    mainPanel(metadataUI('p', 'main'))
  ),
  server = function(input, output, session){
    # reactiveVal to save updates
    saved_data <- reactiveVal()

    cdata <- metadataServer('p', obj, cols.to.drop)

    observeEvent(cdata(), {
      saved_data(cdata())
    })
  }
)
```

---

my.summary

*Summarize DESeq2 results into a dataframe*


---

**Description**

summary(res) prints out info; this function captures it into a dataframe

**Usage**

```
my.summary(res, dds, alpha, lfc.thresh = 0)
```

**Arguments**

res	DESeq2 results object
dds	DESeq2 object
alpha	Alpha level at which to call significantly changing genes
lfc.thresh	log2FoldChange threshold

**Value**

Dataframe of summarized results

**Examples**

```
n_genes <- 100

# make mock dds list
dds <- DESeq2::makeExampleDESeqDataSet(n=n_genes)

# make mock results df
res <- data.frame(
  baseMean = runif(n_genes, 10, 1000),
  log2FoldChange = rnorm(n_genes, 0, 2),
  lfcSE = runif(n_genes, 0.1, 0.5),
  stat = rnorm(n_genes, 0, 3),
  pvalue = runif(n_genes, 0, 1),
  padj = runif(n_genes, 0, 1),
  symbol = paste0("GENE", 1:n_genes),
  row.names = paste0("gene", 1:n_genes)
)

# get summary
df <- my.summary(res, dds, alpha=0.1)
```

---

pcamod

*PCA plot module*

---

**Description**

Module UI + server to generate a pca plot.

**Usage**

```
pcaPlotUI(id, panel)
```

```
pcaPlotServer(id, obj, coldata, config)
```

**Arguments**

id	Module id
panel	string, can be 'sidebar' or 'main'
obj	reactiveValues object containing carnation object
coldata	reactiveValues object containing object metadata
config	reactive list with config settings

**Value**

UI returns tagList with PCA plot UI. Server invisibly returns NULL (used for side effects).

**Examples**

```
library(shiny)
library(DESeq2)

# Create reactive values to simulate app state
oobj <- make_example_carnation_object()

obj <- reactiveValues(
  dds = oobj$dds,
  rld = oobj$rld,
  res = oobj$res,
  all_dds = oobj$all_dds,
  all_rld = oobj$all_rld,
  dds_mapping = oobj$dds_mapping
)

# Set up coldata structure that the module expects
coldata <- reactiveValues(
  curr = list(
    all_samples = colData(oobj$dds$main),
    main = colData(oobj$dds$main)
  )
)

config <- reactiveVal(get_config())

shinyApp(
  ui = fluidPage(
    sidebarPanel(pcaPlotUI('p', 'sidebar')),
    mainPanel(pcaPlotUI('p', 'main'))
  ),
  server = function(input, output, session){
    pcaPlotServer('p', obj, coldata, config)
  }
)
```

---

plotMA.label	<i>Create a labeled MA plot</i>
--------------	---------------------------------

---

### Description

This function creates an MA plot from a data.frame containing DE analysis results.

### Usage

```
plotMA.label(  
  res,  
  fdr.thres = 0.01,  
  fc.thres = 0,  
  fc.lim = NULL,  
  lab.genes = NULL,  
  tolower.cols = c("SYMBOL", "ALIAS")  
)
```

### Arguments

res	data.frame with DE analysis results. Must contain "padj" & "log2FoldChange" columns
fdr.thres	False discovery rate (FDR) threshold
fc.thres	log2FoldChange threshold
fc.lim	y-axis limits
lab.genes	genes to label on MA plot
tolower.cols	column names that will be converted to lower case

### Value

ggplot handle

### Examples

```
# make mock results df  
n_genes <- 100  
res <- data.frame(  
  baseMean = runif(n_genes, 10, 1000),  
  log2FoldChange = rnorm(n_genes, 0, 2),  
  lfcSE = runif(n_genes, 0.1, 0.5),  
  stat = rnorm(n_genes, 0, 3),  
  pvalue = runif(n_genes, 0, 1),  
  padj = runif(n_genes, 0, 1),  
  symbol = paste0("GENE", 1:n_genes),  
  row.names = paste0("gene", 1:n_genes)  
)
```

```
plotMA.label(res, lab.genes = c("gene1", "gene2"))
```

---

plotMA.label_ly	<i>Create an interactive labeled MA plot</i>
-----------------	--

---

## Description

This function creates an MA plot from a data.frame containing DE analysis results using plot\_ly

## Usage

```
plotMA.label_ly(  
  res,  
  fdr.thres = 0.01,  
  fc.thres = 0,  
  fc.lim = NULL,  
  lab.genes = NULL,  
  tolower.cols = c("SYMBOL", "ALIAS")  
)
```

## Arguments

res	data.frame with DE analysis results. Must contain "padj" & "log2FoldChange" columns
fdr.thres	False discovery rate (FDR) threshold
fc.thres	log2FoldChange threshold
fc.lim	y-axis limits
lab.genes	genes to label on MA plot
tolower.cols	column names that will be converted to lower case

## Value

plotly handle

## Examples

```
# make mock results df  
n_genes <- 100  
res <- data.frame(  
  baseMean = runif(n_genes, 10, 1000),  
  log2FoldChange = rnorm(n_genes, 0, 2),  
  lfcSE = runif(n_genes, 0.1, 0.5),  
  stat = rnorm(n_genes, 0, 3),  
  pvalue = runif(n_genes, 0, 1),  
  padj = runif(n_genes, 0, 1),  
  symbol = paste0("GENE", 1:n_genes),
```

```
        row.names = paste0("gene", 1:n_genes)
      )
plotMA.label_ly(res, lab.genes = c("gene1", "gene2"))
```

---

plotPCA.ly

*Plot an interactive PCA plot*

---

### Description

Plot an interactive PCA plot

### Usage

```
plotPCA.ly(rld, intgroup)
```

### Arguments

rld	DESeqTransform object output by <code>varianceStabilizingTransformation()</code> or <code>rlog()</code>
intgroup	character vector of names in <code>colData(x)</code> to use for grouping

### Value

Handle to `ggplot` with added label field in `aes_string()` for plotting with `ggplotly()`

### Examples

```
# make example dds object
dds <- DESeq2::makeExampleDESeqDataSet()

# normalize
rld <- DESeq2::varianceStabilizingTransformation(dds, blind=TRUE)

# make pca plot
p <- plotPCA.ly(rld, intgroup='condition')
```

---

plotPCA.san	<i>Adjustable PCA plot</i>
-------------	----------------------------

---

### Description

Create a PCA plot with specified PCs on x- and y-axis

### Usage

```
plotPCA.san(  
  object,  
  intgroup = "group",  
  pcx,  
  pcy,  
  pcz = NULL,  
  ntop = 500,  
  samples = NULL,  
  loadings = FALSE,  
  loadings_ngenes = 10  
)
```

### Arguments

object	normalized DESeqDataSet object
intgroup	metadata variable to use for grouping samples
pcx	principal component to plot on x-axis
pcy	principal component to plot on y-axis
pcz	principal component to plot on z-axis. If not NULL, function returns a 3-D PCA plot.
ntop	number of most-variable genes to use
samples	vector of sample names to show on plot
loadings	boolean, show gene loadings? Default is FALSE.
loadings_ngenes	integer, # genes to show loadings for (default=10)

### Value

ggplot handle

### Examples

```
# make example dds object  
dds <- DESeq2::makeExampleDESeqDataSet()  
  
# normalize
```

```
rld <- DESeq2::varianceStabilizingTransformation(dds, blind=TRUE)

# make pca plot
p <- plotPCA.san(rld, intgroup='condition', pcx='PC1', pcy='PC2')
```

---

plotScatter.label      *Plot a scatterplot to compare two contrasts*

---

### Description

Plot a scatterplot to compare two contrasts

### Usage

```
plotScatter.label(
  compare,
  df,
  label_x,
  label_y,
  lim.x,
  lim.y,
  color.palette,
  lab.genes = NULL,
  plot_all = "no",
  name.col = "geneid",
  lines = c("yes", "yes", "yes"),
  alpha = 1,
  size = 4,
  show.grid = "yes"
)
```

### Arguments

compare	string, what values to plot? can be 'log2FoldChange' or 'P-adj'
df	data frame with log2FoldChange & padj values to plot from 2 contrasts
label_x	string, label for x-axis
label_y	string, label for y-axis
lim.x	x-axis limits
lim.y	y-axis limits
color.palette	character vector of colors to use for significance categories 'Both - same LFC sign', 'Both - opposite LFC sign', 'None', label_x, label_y
lab.genes	genes to label (default=NULL)
plot_all	string, can be 'yes' or 'no'. if 'yes', points outside axis limits are plotted along x/y axis lines (default='no').

name.col	gene name column to merge the 2 results, also used for labeling points
lines	3-element character vector to plot gridlines in the order (x=0, y=0, x=y), with 'yes' or 'no' values. E.g. ('yes', 'yes', 'no') will plot dotted lines for x = 0 & y = 0, but not the x = y diagonal.
alpha	float, marker opacity (default=1).
size	float, marker size (default=4).
show.grid	string, can be 'yes' (default) or 'no'.

**Value**

ggplot handle

**Examples**

```
# make mock results df
n_genes <- 100
res1 <- data.frame(
  baseMean = runif(n_genes, 10, 1000),
  log2FoldChange = rnorm(n_genes, 0, 2),
  lfcSE = runif(n_genes, 0.1, 0.5),
  stat = rnorm(n_genes, 0, 3),
  pvalue = runif(n_genes, 0, 1),
  padj = runif(n_genes, 0, 1),
  symbol = paste0("GENE", 1:n_genes),
  row.names = paste0("gene", 1:n_genes)
)

res2 <- data.frame(
  baseMean = runif(n_genes, 10, 1000),
  log2FoldChange = rnorm(n_genes, 0, 2),
  lfcSE = runif(n_genes, 0.1, 0.5),
  stat = rnorm(n_genes, 0, 3),
  pvalue = runif(n_genes, 0, 1),
  padj = runif(n_genes, 0, 1),
  symbol = paste0("GENE", 1:n_genes),
  row.names = paste0("gene", 1:n_genes)
)

# add geneid column
res1 <- cbind(geneid=row.names(res1), res1)
res2 <- cbind(geneid=row.names(res2), res2)

# make merged df from the two comparisons
cols.sub <- c('log2FoldChange', 'padj', 'geneid')
df_full <- dplyr::inner_join(
  dplyr::select(as.data.frame(res1), all_of(cols.sub)),
  dplyr::select(as.data.frame(res2), all_of(cols.sub)),
  by = 'geneid',
  suffix = c('.x', '.y')
)
```

```

# calculate x & y limits for log2FoldChange
xlim <- range(df_full[[ 'log2FoldChange.x' ]])
ylim <- range(df_full[[ 'log2FoldChange.y' ]])

# get color palette
color.palette <- RColorBrewer::brewer.pal(n=5, name='Set2')

# add significance column
sig.x <- df_full$padj.x < 0.1 & !is.na(df_full$padj.x)
sig.y <- df_full$padj.y < 0.1 & !is.na(df_full$padj.y)
up.x <- df_full$log2FoldChange.x >= 0
up.y <- df_full$log2FoldChange.y >= 0
significance <- rep('None', nrow(df_full))
significance[ sig.x & sig.y & ((up.x & up.y) | (!up.x & !up.y)) ] <- 'Both - same LFC sign'
significance[ sig.x & sig.y & ((up.x & !up.y) | (!up.x & up.y)) ] <- 'Both - opposite LFC sign'
significance[ sig.x & !sig.y ] <- 'A vs B'
significance[ !sig.x & sig.y ] <- 'B vs A'
df_full$significance <- significance

# generate scatter plot
p <- plotScatter.label(compare = 'log2FoldChange',
                       df = df_full,
                       label_x = 'A vs B',
                       label_y = 'B vs A',
                       lim.x = xlim,
                       lim.y = ylim,
                       color.palette = color.palette)

```

---

plotScatter.label\_ly *Plot an interactive scatterplot to compare two contrasts*

---

## Description

Plot an interactive scatterplot to compare two contrasts

## Usage

```

plotScatter.label_ly(
  compare,
  df,
  label_x,
  label_y,
  lim.x,
  lim.y,
  color.palette,
  lab.genes = NULL,
  name.col = "geneid",
  lines = c("yes", "yes", "yes"),

```

```

    alpha = 1,
    size = 4,
    show.grid = "yes",
    source = "A"
  )

```

### Arguments

compare	string, what values to plot? can be 'log2FoldChange' or 'P-adj'
df	data frame with log2FoldChange & padj values to plot from 2 contrasts
label_x	string, label for x-axis
label_y	string, label for y-axis
lim.x	x-axis limits
lim.y	y-axis limits
color.palette	character vector of colors to use for significance categories 'Both - same LFC sign', 'Both - opposite LFC sign', 'None', label_x, label_y
lab.genes	genes to label (default=NULL)
name.col	gene name column to merge the 2 results, also used for labeling points
lines	3-element character vector to plot gridlines in the order (x=0, y=0, x=y), with 'yes' or 'no' values. E.g. ('yes', 'yes', 'no') will plot dotted lines for x = 0 & y = 0, but not the x = y diagonal.
alpha	float, marker opacity (default=1).
size	float, marker size (default=4).
show.grid	string, can be 'yes' (default) or 'no'.
source	name of source to return event_data from

### Value

plotly handle

### Examples

```

# make mock results df
n_genes <- 100
res1 <- data.frame(
  baseMean = runif(n_genes, 10, 1000),
  log2FoldChange = rnorm(n_genes, 0, 2),
  lfcSE = runif(n_genes, 0.1, 0.5),
  stat = rnorm(n_genes, 0, 3),
  pvalue = runif(n_genes, 0, 1),
  padj = runif(n_genes, 0, 1),
  symbol = paste0("GENE", 1:n_genes),
  row.names = paste0("gene", 1:n_genes)
)

res2 <- data.frame(

```

```

        baseMean = runif(n_genes, 10, 1000),
        log2FoldChange = rnorm(n_genes, 0, 2),
        lfcSE = runif(n_genes, 0.1, 0.5),
        stat = rnorm(n_genes, 0, 3),
        pvalue = runif(n_genes, 0, 1),
        padj = runif(n_genes, 0, 1),
        symbol = paste0("GENE", 1:n_genes),
        row.names = paste0("gene", 1:n_genes)
    )

# add geneid column
res1 <- cbind(geneid=row.names(res1), res1)
res2 <- cbind(geneid=row.names(res2), res2)

# make merged df from the two comparisons
cols.sub <- c('log2FoldChange', 'padj', 'geneid')
df_full <- dplyr::inner_join(
  dplyr::select(as.data.frame(res1), all_of(cols.sub)),
  dplyr::select(as.data.frame(res2), all_of(cols.sub)),
  by = 'geneid',
  suffix = c('.x', '.y')
)

# calculate x & y limits for log2FoldChange
xlim <- range(df_full[['log2FoldChange.x' ]])
ylim <- range(df_full[['log2FoldChange.y' ]])

# get color palette
color.palette <- RColorBrewer::brewer.pal(n=5, name='Set2')

# add significance column
sig.x <- df_full$padj.x < 0.1 & !is.na(df_full$padj.x)
sig.y <- df_full$padj.y < 0.1 & !is.na(df_full$padj.y)
up.x <- df_full$log2FoldChange.x >= 0
up.y <- df_full$log2FoldChange.y >= 0
significance <- rep('None', nrow(df_full))
significance[ sig.x & sig.y & ((up.x & up.y) | (!up.x & !up.y)) ] <- 'Both - same LFC sign'
significance[ sig.x & sig.y & ((up.x & !up.y) | (!up.x & up.y)) ] <- 'Both - opposite LFC sign'
significance[ sig.x & !sig.y ] <- 'A vs B'
significance[ !sig.x & sig.y ] <- 'B vs A'
df_full$significance <- significance

# generate scatter plot
p <- plotScatter.label_ly(compare = 'log2FoldChange',
  df = df_full,
  label_x = 'A vs B',
  label_y = 'B vs A',
  lim.x = xlim,
  lim.y = ylim,
  color.palette = color.palette)

```

---

radarmod	<i>Radar plot module</i>
----------	--------------------------

---

**Description**

UI & module to generate radar plots.

**Usage**

```
radarUI(id, panel, type = "")
radarServer(id, obj, config, type = "")
```

**Arguments**

id	Module id
panel	string, can be 'sidebar' or 'main'
type	string, if 'comp' then show the comparison view
obj	reactiveValues object containing GeneTonic object
config	reactive list with config settings

**Value**

UI returns tagList with plot UI server invisibly returns NULL (used for side effects)

**Examples**

```
library(shiny)

# get enrichResult object
data(eres_dex, package='carnation')

# convert to GeneTonic object
gt <- GeneTonic::shake_enrichResult(eres_dex)

obj <- reactive({
  list(l_gs = gt$l_gs,
       anno_df = gt$anno_df,
       label = 'comp1')
})

config <- reactiveVal(get_config())

# run simple shiny app with plot
if(interactive()){
  shinyApp(
    ui = fluidPage(
      sidebarPanel(radarUI('p', 'sidebar')),
```

```
        mainPanel(radarUI('p', 'main'))
      ),
      server = function(input, output, session){
        radarServer('p', obj, config)
      }
    )
  }
```

---

read\_access\_yaml

*Read access yaml with user groups and data areas*

---

### Description

This function reads the access yaml file and returns user groups and data areas as a list of data frames.

### Usage

```
read_access_yaml()
```

### Value

return carnation access settings from yaml file

### Examples

```
# save access details to file
home <- Sys.getenv('HOME')

# create carnation data area if it doesn't exist
carnation_home <- file.path(home, 'carnation/data')
if(!dir.exists(carnation_home)) dir.create(carnation_home)

create_access_yaml(user = 'admin',
                  user_group = 'admin',
                  data_area = carnation_home)

al <- read_access_yaml()
```

---

res_cell	<i>A DESeqResults object testing the difference between two cell lines of smooth muscle cells</i>
----------	---

---

**Description**

A DESeqResults object testing the difference between two cell lines of smooth muscle cells

**Format**

A DESeqResults object, generated in the DESeq2 framework

**Details**

This DESeqResults object on the data from the airway package has been created comparing two smooth muscle cell lines, accounting for the effect of dexamethasone treatment.

Details on how this object has been created are included in the create\_carnation\_data.R script, included in the scripts folder of the Carnation package.

**References**

Himes BE, Jiang X, Wagner P, Hu R, Wang Q, Klanderma B, Whitaker RM, Duan Q, Lasky-Su J, Nikolos C, Jester W, Johnson M, Panettieri R Jr, Tantisira KG, Weiss ST, Lu Q. "RNA-Seq Transcriptome Profiling Identifies CRISPLD2 as a Glucocorticoid Responsive Gene that Modulates Cytokine Function in Airway Smooth Muscle Cells." PLoS One. 2014 Jun 13;9(6):e99625. PMID: 24926665. GEO: GSE52778

---

res_dex	<i>A DESeqResults object testing the effect of dexamethasone on smooth muscle cells</i>
---------	---

---

**Description**

A DESeqResults object testing the effect of dexamethasone on smooth muscle cells

**Format**

A DESeqResults object, generated in the DESeq2 framework

**Details**

This DESeqResults object on the data from the airway package has been created comparing dexamethasone treated vs untreated samples, accounting for the different cell lines included.

Details on how this object has been created are included in the create\_carnation\_data.R script, included in the scripts folder of the Carnation package.

## References

Himes BE, Jiang X, Wagner P, Hu R, Wang Q, Klanderman B, Whitaker RM, Duan Q, Lasky-Su J, Nikolos C, Jester W, Johnson M, Panettieri R Jr, Tantisira KG, Weiss ST, Lu Q. “RNA-Seq Transcriptome Profiling Identifies CRISPLD2 as a Glucocorticoid Responsive Gene that Modulates Cytokine Function in Airway Smooth Muscle Cells.” PLoS One. 2014 Jun 13;9(6):e99625. PMID: 24926665. GEO: GSE52778

---

run\_carnation

*Carnation*

---

## Description

Interactive shiny dashboard for exploring RNA-Seq analysis.

## Usage

```
run_carnation(
  credentials = NULL,
  passphrase = NULL,
  enable_admin = TRUE,
  config_path = NULL,
  ...
)
```

## Arguments

credentials	path to encrypted sqlite db with user credentials.
passphrase	passphrase for credentials db.
enable_admin	if TRUE, admin view is shown. Note, this is only available if credentials have sqlite backend.
config_path	optional path to a local config yaml override.
...	parameters passed to shinyApp() call

## Value

shinyApp object

## Examples

```
if(interactive()){
  shiny::runApp(
    run_carnation()
  )
}
```

---

savemod

*Save object module UI*


---

### Description

Module UI & server to save carnation object.

### Usage

```
saveUI(id)
```

```
saveServer(id, original, current, coldata, pattern, username, config)
```

### Arguments

id	Module id
original	original carnation object
current	current carnation object
coldata	reactiveValues object containing object metadata
pattern	regex pattern for finding carnation data
username	user name
config	reactive list with config settings

### Value

UI returns actionButton Server returns reactive with trigger to refresh the app

### Examples

```
library(shiny)
library(DESeq2)

# default username
username <- reactive({ NULL })

# internal carnation config
config <- reactiveVal(get_config())

# regex to find carnation files
pattern <- reactive({ config()$server$pattern })

# get example object
obj <- make_example_carnation_object()

# make reactive with obj & path
original <- reactiveValues( obj = obj, path = "/path/to/carnation/obj.rds" )
```

```

# extract metadata
coldata <- reactive({ lapply(obj$dds, colData) })

# edit metadata
coldata_edit <- lapply(coldata, function(x){
  x$type <- 'new'; x
})

# add to object
edit_obj <- obj
for(name in names(edit_obj$dds)){
  colData(edit_obj$dds[[ name ]]) <- coldata_edit[[ name ]]
}

# run simple shiny app with plot
shinyApp(
  ui = fluidPage(
    saveUI('p')
  ),
  server = function(input, output, session){
    save_event <- saveServer('save_object',
                             original=original,
                             current=reactive({ edit_obj }),
                             coldata=coldata,
                             pattern=pattern(),
                             username=username,
                             config)
  }
)

```

---

save\_access\_yaml

*Save access yaml to file*


---

### Description

This function saves access details (user groups and data areas) to the designated access yaml file.

### Usage

```
save_access_yaml(lst)
```

### Arguments

lst                    list of data frames with user\_groups and data\_areas

### Value

save access settings to yaml file

**Examples**

```

# save access details to file
home <- Sys.getenv('HOME')

# create carnation data area if it doesn't exist
carnation_home <- file.path(home, 'carnation/data')
if(!dir.exists(carnation_home)) dir.create(carnation_home)

create_access_yaml(user = 'admin',
                  user_group = 'admin',
                  data_area = carnation_home)

# read access yaml
lst <- read_access_yaml()

# add new user
lst$user_group$admin <- c(lst$user_group$admin, 'user1')

# save to access settings
save_access_yaml(lst)

```

---

scattermod

*Scatterplot module*


---

**Description**

Module UI + server for generating scatter plots.

**Usage**

```
scatterPlotUI(id, panel)
```

```
scatterPlotServer(id, obj, plot_args, gene_scratchpad, reset_genes, config)
```

**Arguments**

id	Module id
panel	string, can be 'sidebar' or 'main' passed to UI
obj	reactiveValues object containing carnation object passed to server
plot_args	reactive containing 'fdr.thres' (p adj threshold), 'fc.thres' (log2FC)
gene_scratchpad	reactive containing gene scratchpad genes
reset_genes	reactive to reset gene scratchpad selection
config	reactive list with config settings passed to server

**Value**

UI returns tagList with scatter plot UI. Server invisibly returns NULL (used for side effects).

**Examples**

```
library(shiny)

# Create reactive values to simulate app state
oobj <- make_example_carnation_object()

obj <- reactiveValues(
  dds = oobj$dds,
  rld = oobj$rld,
  res = oobj$res,
  all_dds = oobj$all_dds,
  all_rld = oobj$all_rld,
  dds_mapping = oobj$dds_mapping
)

plot_args <- reactive({
  list(
    fdr.thres=0.1,
    fc.thres=0
  )
})

gene_scratchpad <- reactive({ c('gene1', 'gene2') })
reset_genes <- reactiveVal()

config <- reactiveVal(get_config())

shinyApp(
  ui = fluidPage(
    sidebarPanel(scatterPlotUI('p', 'sidebar')),
    mainPanel(scatterPlotUI('p', 'sidebar'))
  ),
  server = function(input, output, session){
    scatter_data <- scatterPlotServer('p', obj, plot_args,
                                     gene_scratchpad, reset_genes, config)
  }
)
```

---

 settingsmod

*Settings module*


---

**Description**

Module UI & server for user access details interface.

Server code for settings module

**Usage**

```
settingsUI(id, panel, username)
```

```
settingsServer(id, details, depth, end_offset, assay_fun, config)
```

**Arguments**

id	Module id
panel	context for generating ui elements ('sidebar' or 'main')
username	user name
details	reactive list with user name & app location details
depth	project name depth
end_offset	project name end offset
assay_fun	function to parse assay names from file path
config	reactive list with config settings

**Value**

UI returns tagList with module UI Server returns reactive with list containing user access details

**Examples**

```
library(shiny)

# default username
username <- reactive({ NULL })

# internal carnation config
config <- reactiveVal(get_config())

# regex to find carnation files
pattern <- reactive({ config()$server$pattern })

# access permissions
assay.list <- reactiveValues(l=read_access_yaml())

if(interactive()){
  shinyApp(
    ui = fluidPage(
      sidebarPanel(uiOutput('settings_sidebar')),
      mainPanel(uiOutput('settings_main'))
    ),
    server = function(input, output, session){
      output$settings_main <- renderUI({
        settingsUI('settings', panel='main', username=username)
      })

      output$settings_sidebar <- renderUI({
        settingsUI('settings', panel='sidebar', username=username)
      })
    }
  )
}
```

```

    })
    settings <- settingsServer('p',
                              details=reactive({
                                list(username=username,
                                      where=NULL)
                                }),
                              depth=2,
                              end_offset=0,
                              assay_fun=function(x)
                                sub(paste0(pattern(), '\\.rds$'), '',
                                    basename(x),
                                    ignore.case=TRUE),
                              config
                                )
  }
}

```

---

set\_config

*Set config*


---

### Description

This function updates a limited subset of the package config YAML. Only stable user-facing settings are writable; style settings and other internal options are intentionally left untouched.

### Usage

```

set_config(
  config_path = get_config_path(),
  de_analysis = NULL,
  fdr_threshold = NULL,
  log2fc_threshold = NULL,
  max_upload_size = NULL,
  cores = NULL,
  pattern = NULL
)

```

### Arguments

config_path	character path to the config YAML file to update. Defaults to the local config returned by <code>get_config_path()</code> . If the file does not exist yet, it is initialized from the bundled package config.
de_analysis	optional list with DE analysis config updates. Currently only <code>de_analysis\$column_names</code> is supported, and the provided aliases are merged into the existing column-name mappings.

**fdr\_threshold** optional numeric FDR threshold between 0 and 1.  
**log2fc\_threshold** optional numeric log2 fold-change threshold greater than or equal to 0.  
**max\_upload\_size** optional positive numeric upload limit in MB.  
**cores** optional positive integer number of cores to use.  
**pattern** optional character suffix pattern used to match dataset filenames before the trailing `.rds`. Use `""` to match all RDS files.

**Value**

Updated config list, invisibly.

**Examples**

```

cfg_out <- tempfile(fileext = ".yaml")

set_config(
  config_path = cfg_out,
  de_analysis = list(
    column_names = list(
      padj = "qvalue",
      log2FoldChange = c("logFC", "avg_log2FC")
    )
  ),
  fdr_threshold = 0.05,
  log2fc_threshold = 1,
  max_upload_size = 50,
  cores = 2,
  pattern = "carnation"
)

```

---

`summarize_res_list`      *Combine everything in the results list into a single table*

---

**Description**

Combine everything in the results list into a single table

**Usage**

```

summarize_res_list(
  res.list,
  dds.list,
  dds_mapping,
  alpha,
  lfc.thresh,
  labels = NULL
)

```

**Arguments**

<code>res.list</code>	Named list of lists, where each sublist contains the following names: <code>c('res', 'dds', 'label')</code> . "res" is a DESeqResults object, "dds" is either the indexing label for the dds.list object or the DESeq object, and "label" is a nicer-looking label to use. NOTE: backwards compatibility with older versions of lcdb-wf depends on no dds.list object being passed.
<code>dds.list</code>	List of DESeqDataSet objects whose names are expected to match 'dds' slots in the 'res.list' object
<code>dds_mapping</code>	List mapping names of dds.list to res.list elements
<code>alpha</code>	false-discovery rate threshold
<code>lfc.thresh</code>	log2FoldChange threshold
<code>labels</code>	list of descriptions for res.list elements

**Value**

Dataframe

**Examples**

```
n_genes <- 100

# make mock dds list
dds_list <- list(main=DESeq2::makeExampleDESeqDataSet(n=n_genes))

# make mock results df
res1 <- data.frame(
  baseMean = runif(n_genes, 10, 1000),
  log2FoldChange = rnorm(n_genes, 0, 2),
  lfcSE = runif(n_genes, 0.1, 0.5),
  stat = rnorm(n_genes, 0, 3),
  pvalue = runif(n_genes, 0, 1),
  padj = runif(n_genes, 0, 1),
  symbol = paste0("GENE", 1:n_genes),
  row.names = paste0("gene", 1:n_genes)
)

res2 <- data.frame(
  baseMean = runif(n_genes, 10, 1000),
  log2FoldChange = rnorm(n_genes, 0, 2),
  lfcSE = runif(n_genes, 0.1, 0.5),
  stat = rnorm(n_genes, 0, 3),
  pvalue = runif(n_genes, 0, 1),
  padj = runif(n_genes, 0, 1),
  symbol = paste0("GENE", 1:n_genes),
  row.names = paste0("gene", 1:n_genes)
)

# make list of results
res_list <- list(
```

```

        comp1=res1,
        comp2=res2
    )

# make dds mapping
dds_mapping <- list(comp1='main', comp2='main')

# get summary
df <- summarize_res_list(res_list, dds_list, dds_mapping, alpha=0.1, lfc.thresh=0)

```

---

sumovmod

*Summary overview plot module*


---

## Description

UI & module to generate summary overview plots.

## Usage

```

sumovPlotUI(id, panel, type = "")

sumovPlotServer(id, obj, config, type = "")

```

## Arguments

id	Module id
panel	string, can be 'sidebar' or 'main'
type	string, if 'comp' then show the comparison view
obj	reactiveValues object containing GeneTonic object
config	reactive list with config settings

## Value

UI returns tagList with plot UI server invisibly returns NULL (used for side effects)

## Examples

```

library(shiny)

# get enrichResult object
data(eres_dex, package='carnation')

# convert to GeneTonic object
gt <- GeneTonic::shake_enrichResult(eres_dex)

obj <- reactive({
  list(l_gs = gt$l_gs,

```

```

        anno_df = gt$anno_df,
        label = 'comp1')
  })

  config <- reactiveVal(get_config())

  # run simple shiny app with plot
  if(interactive()){
    shinyApp(
      ui = fluidPage(
        sidebarPanel(sumovPlotUI('p', 'sidebar')),
        mainPanel(sumovPlotUI('p', 'main'))
      ),
      server = function(input, output, session){
        sumovPlotServer('p', obj, config)
      }
    )
  }

```

---

top.genes

*Get top DE genes by log2FoldChange or adjusted p-value*


---

### Description

Get top DE genes by log2FoldChange or adjusted p-value

### Usage

```
top.genes(res, fdr.thres = 0.01, fc.thres = 0, n = 10, by = "log2FoldChange")
```

### Arguments

res	data.frame with DE analysis results
fdr.thres	FDR threshold
fc.thres	log2FoldChange threshold
n	number of genes to return
by	metric to determine top genes ('log2FoldChange' or 'padj')

### Value

vector of gene symbols

### Examples

```

# get DE results
data(res_dex, package='carnation')

g <- top.genes(res_dex)

```

---

upsetmod

*Upset plot module*

---

## Description

Module UI & server to generate upset plots.

## Usage

```
upsetPlotUI(id, panel)
```

```
upsetPlotServer(id, obj, plot_args, gene_scratchpad, reset_genes, config)
```

## Arguments

id	Module id
panel	string, can be 'sidebar' or 'main'
obj	reactiveValues object containing carnation object
plot_args	reactive containing 'fdr.thres' (padj threshold) & 'fc.thres' (log2FC)
gene_scratchpad	reactiveValues object containing genes selected in scratchpad
reset_genes	reactive to reset gene scratchpad selection
config	reactive list with config settings

## Value

UI returns tagList with upset plot UI. Server returns reactive with list containing upset table, intersections & selected genes.

## Examples

```
library(shiny)

oobj <- make_example_carnation_object()

obj <- reactiveValues(
  dds = oobj$dds,
  rld = oobj$rld,
  res = oobj$res,
  all_dds = oobj$all_dds,
  all_rld = oobj$all_rld,
  dds_mapping = oobj$dds_mapping
)

plot_args <- reactive({
  list(
    fdr.thres=0.1,
```

```

    fc.thres=0
  )
})

gene_scratchpad <- reactive({ c('gene1', 'gene2') })

reset_genes <- reactiveVal()

config <- reactiveVal(get_config())

shinyApp(
  ui = fluidPage(
    sidebarPanel(upsetPlotUI('p', 'sidebar')),
    mainPanel(upsetPlotUI('p', 'sidebar'))
  ),
  server = function(input, output, session){
    upset_data <- upsetPlotServer('p', obj, plot_args,
                                  gene_scratchpad,
                                  reset_genes, config)
  }
)

```

---

validate\_carnation\_object

*Validate a carnation object*

---

### Description

This function takes various input data types (DE results, counts, enrichment, pattern analysis) and validates them according to carnation's requirements, returning a normalized intermediate object. Expensive derived-object creation steps such as variance-stabilized counts and GeneTonic conversion are handled separately by `materialize_carnation_object()`.

### Usage

```

validate_carnation_object(
  res_list,
  dds_list,
  rld_list = NULL,
  labels = NULL,
  enrich_list = NULL,
  degpatterns = NULL,
  metadata = NULL,
  dds_mapping = NULL,
  config = NULL
)

```

**Arguments**

res_list	Named list of DE results. Each element should be either: <ul style="list-style-type: none"> <li>• A data frame with DE results containing gene, symbol, pvalue, padj, log2FoldChange, and baseMean columns (or tool-specific alternatives)</li> <li>• A list with slots: res (data frame), dds (name reference to dds_list element), label (comparison label)</li> </ul>
dds_list	Named list of count data. Each element should be either: <ul style="list-style-type: none"> <li>• A DESeqDataSet object</li> <li>• A data frame or matrix of raw counts (first column=gene IDs, remaining=samples)</li> </ul>
rld_list	Optional named list of variance-stabilized count objects. If NULL, these can be generated later via materialize_carnation_object().
labels	Optional named list of comparison labels. If NULL and res_list contains nested structure with label slots, labels will be extracted.
enrich_list	Optional named list of functional enrichment results. Should be structured as: enrich_list[[func_id]][[effect]][[pathway]]. Each enrichment result must be a data frame in clusterProfiler format: <ul style="list-style-type: none"> <li>• Over-representation: ID, Description, GeneRatio, BgRatio, pvalue, p.adjust, qvalue, geneID, Count</li> <li>• GSEA: ID, Description, core_enrichment, setSize, pvalue, p.adjust, qvalue, NES</li> </ul>
degpatterns	Optional named list of pattern analysis results. Each element should be either a data frame or a list with \$normalized slot containing a data frame with columns: genes, value, and either cluster or columns starting with "cutoff".
metadata	Optional data frame with sample metadata. Required if dds_list contains count matrices instead of DESeqDataSet objects. First column should be sample names matching column names in count matrices.
dds_mapping	Optional named list mapping res_list elements to dds_list objects. Required if res_list is a list of data frames.
config	Optional config list. If NULL, will use get_config(), including any supported local config overrides.

**Details**

This function performs comprehensive validation of all input data:

- DE results: Checks for required columns (with support for DESeq2, edgeR, limma), ensures gene and symbol columns exist
- Counts: Validates structure, checks sample name matching with metadata
- Enrichment: Validates clusterProfiler format (OR or GSEA)
- Pattern analysis: Checks for required columns (genes, value, cluster)

If validation fails, the function will stop with an informative error message.

**Value**

A validated list with canonical slots `res_list`, `dds_list`, optional `rld_list`, `labels`, `dds_mapping`, `enrich_list`, `degpatterns`, and `metadata` when supplied.

A list containing normalized inputs with elements `res_list`, `dds_list`, optional `rld_list`, `labels`, `dds_mapping`, and optional `enrich_list`, `degpatterns`, and `metadata`.

**Examples**

```
# Minimal example with DE results and counts
library(DESeq2)

# Create example data
dds <- makeExampleDESeqDataSet()
dds <- DESeq(dds)
res <- results(dds, contrast = c("condition", "A", "B"))
rld <- varianceStabilizingTransformation(dds, blind = TRUE)

# Validate object inputs
obj <- validate_carnation_object(
  res_list = list(
    comp1 = list(
      res = as.data.frame(res),
      dds = "main",
      label = "A vs B"
    )
  ),
  dds_list = list(main = dds),
  rld_list = list(main = rld)
)

materialized <- materialize_carnation_object(obj, cores = 1)
final_obj <- make_final_object(materialized)

# Save for use with carnation
saveRDS(final_obj, "my_analysis.rds")

# Alternative: start from count matrix and metadata
counts <- as.data.frame(counts(dds))
counts$gene <- rownames(counts)
counts <- counts[, c(ncol(counts), 1:(ncol(counts)-1))]
metadata <- as.data.frame(colData(dds))
metadata$sample <- rownames(metadata)
metadata <- metadata[, c(ncol(metadata), 1:(ncol(metadata)-1))]

obj <- validate_carnation_object(
  res_list = list(comp1 = as.data.frame(res)),
  dds_list = list(main = counts),
  metadata = metadata,
  dds_mapping = list(comp1 = "main")
)
```

# Index

## \* internal

- carnation-package, 4
- add.set.column, 5
- add\_metadata, 5
- alluvialmod, 6
- alluvialServer (alluvialmod), 6
- alluvialUI (alluvialmod), 6
- carnation (carnation-package), 4
- carnation-package, 4
- check\_user\_access, 7
- cnetmod, 8
- cnetPlotServer (cnetmod), 8
- cnetPlotUI (cnetmod), 8
- create\_access\_yaml, 9
- degmod, 10
- degpatterns\_dex, 12
- dendrogramServer (dendromod), 13
- dendrogramUI (dendromod), 13
- dendromod, 13
- distillmod, 14
- distillPlotServer (distillmod), 14
- distillPlotUI (distillmod), 14
- dlmod, 15
- downloadButtonServer (dlmod), 15
- downloadButtonUI (dlmod), 15
- dummy\_genetonic, 16
- emapmod, 17
- enrich\_to\_genetonic, 18
- enrichmapServer (emapmod), 17
- enrichmapUI (emapmod), 17
- enrichServer (funenrichmod), 21
- enrichUI (funenrichmod), 21
- eres\_cell, 19
- eres\_dex, 19
- format\_genes, 20
- fromList.with.names, 21
- funenrichmod, 21
- fuzzymod, 23
- fuzzyPlotServer (fuzzymod), 23
- fuzzyPlotUI (fuzzymod), 23
- geneplotmod, 24
- genePlotServer (geneplotmod), 24
- genePlotUI (geneplotmod), 24
- get\_access\_path, 28
- get\_config, 28
- get\_config\_path, 29
- get\_degplot, 29
- get\_gene\_counts, 30
- get\_project\_name\_from\_path, 31
- get\_upset\_table, 32
- get\_y\_init, 33
- getcountplot, 26
- gs\_radar, 33
- heatmapmod, 35
- heatmapServer (heatmapmod), 35
- heatmapUI (heatmapmod), 35
- helpButtonServer (helpmod), 36
- helpButtonUI (helpmod), 36
- helpmod, 36
- helpModal, 37
- horizonmod, 38
- horizonServer (horizonmod), 38
- horizonUI (horizonmod), 38
- in\_admin\_group, 40
- init\_local\_config, 39
- install\_carnation, 40
- is\_site\_admin, 41
- is\_valid\_pattern\_obj, 42
- loadDataServer (loadmod), 42
- loadDataUI (loadmod), 42
- loadmod, 42
- make\_example\_carnation\_object, 44

make\_final\_object, 45  
makeEnrichResult, 43  
maplotmod, 46  
maPlotServer (maplotmod), 46  
maPlotUI (maplotmod), 46  
materialize\_carnation\_object, 47  
metadataServer (metamod), 48  
metadataUI (metamod), 48  
metamod, 48  
my.summary, 49

patternPlotServer (degmod), 10  
patternPlotUI (degmod), 10  
pcamod, 50  
pcaPlotServer (pcamod), 50  
pcaPlotUI (pcamod), 50  
plotMA.label, 52  
plotMA.label\_ly, 53  
plotPCA.ly, 54  
plotPCA.san, 55  
plotScatter.label, 56  
plotScatter.label\_ly, 58

radarmod, 61  
radarServer (radarmod), 61  
radarUI (radarmod), 61  
read\_access\_yaml, 62  
res\_cell, 63  
res\_dex, 63  
run\_carnation, 64  
run\_carnation(), 4

save\_access\_yaml, 66  
savemod, 65  
saveServer (savemod), 65  
saveUI (savemod), 65  
scattermod, 67  
scatterPlotServer (scattermod), 67  
scatterPlotUI (scattermod), 67  
set\_config, 70  
settingsmod, 68  
settingsServer (settingsmod), 68  
settingsUI (settingsmod), 68  
summarize\_res\_list, 71  
sumovmod, 73  
sumovPlotServer (sumovmod), 73  
sumovPlotUI (sumovmod), 73

top.genes, 74

upsetmod, 75  
upsetPlotServer (upsetmod), 75  
upsetPlotUI (upsetmod), 75  
validate\_carnation\_object, 76