# Orange3 Bioinformatics Documentation

Biolab

Oct 05, 2018

# Contents

# CHAPTER 1

# Widgets

## 1.1 Databases Update

Updates local systems biology databases, like gene ontologies, annotations, gene names, protein interaction networks, and similar.

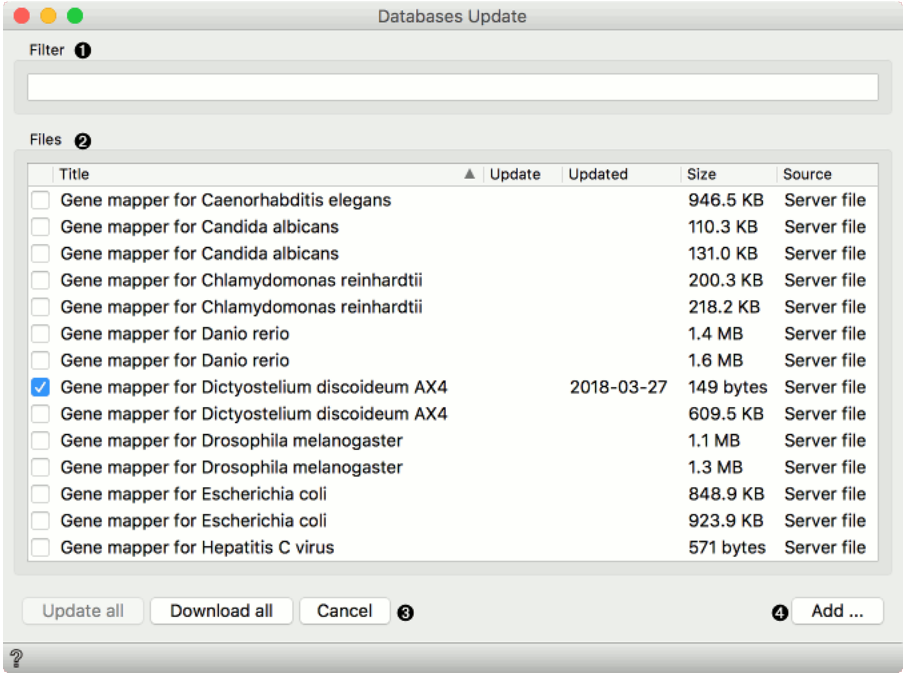
**Inputs**

- None

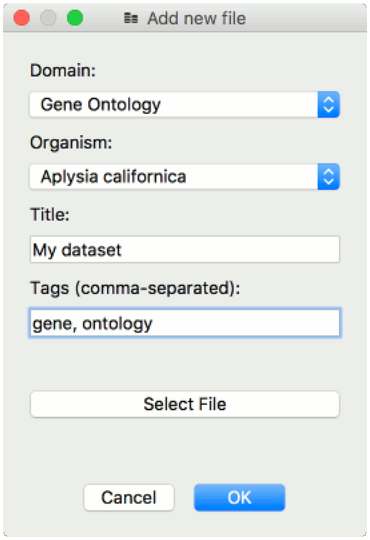
**Outputs**

- None

With the bioinformatics add-on you can access several databases directly from Orange. The widget can also be used to update and manage locally stored databases. To get a more detailed information on the particular database hover on its name.




1. Find the desired database.
2. A list of available databases described with data source, update availability, date of your last update and file size. A large *Update* button will be displayed next to the database that needs to be updated.
3. *Update All* will update and *Download All* will download all of the available databases from the serverfiles. *Cancel* will abort the action.
4. Add a data set from the local machine.




To add a new file to the database, select the domain and the organism of the data. Give the data set a name and, optionally, tag it with appropriate tags. Finally, use the *Select File* button to load the local file. Press OK to complete the process. The data will be stored in a cached folder **locally**. To see the full path to the data, hover on the data set name.

## 1.2 GEO Data Sets

Provides access to data sets from gene expression omnibus GEO DataSets.

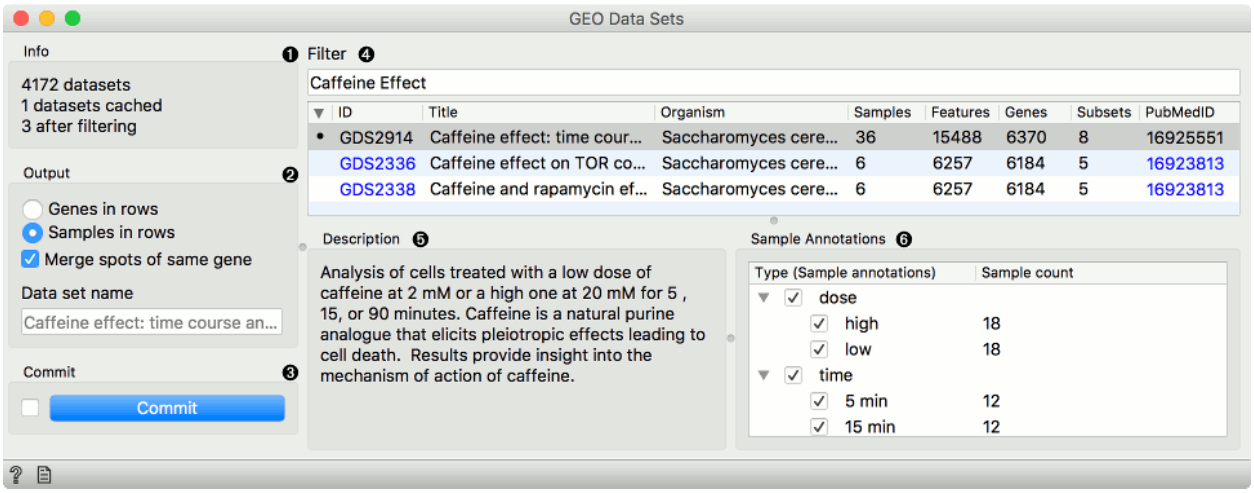
**Inputs**

- None

**Outputs**

- Expression data: Data set selected in the widget with genes or samples in rows.

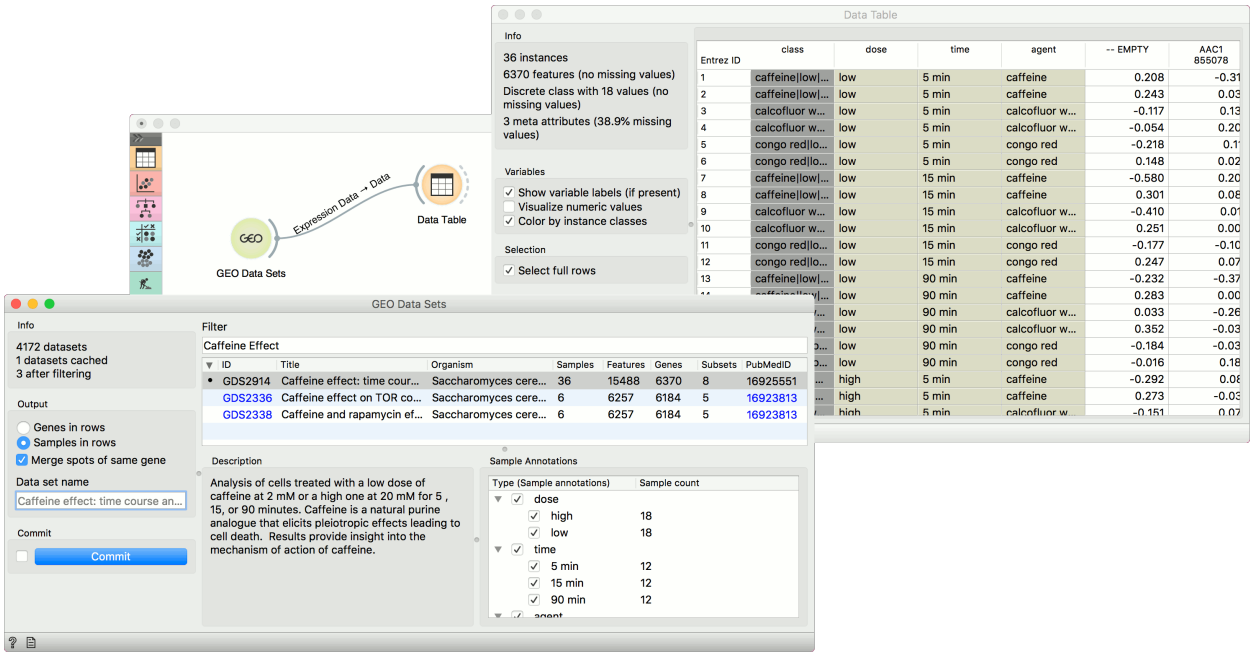
GEO DataSets is a database of gene expression curated profiles maintained by NCBI and included in the Gene Expression Omnibus. This Orange widget provides access to all its data sets and outputs a data set selected for further processing. For convenience, each dowloaded data set is stored locally.




1. Information on the GEO data set collection. Cached data sets are the ones currently stored on the computer.
2. Output features. If *Samples in rows* is selected, genes (or spots) will be used as attributes. Alternatively samples will be used as attributes. *Merge spots of same gene* averages measures of the same gene. Finally, in the *Data set name* you can rename the output data. GEO title will be used as a default name.
3. If *Auto commit is on*, then the selected data set will be automatically communicated to other widgets. Alternatively, click *Commit*.
4. *Filter* allows you to search for the data set. Below you see a list of GEO data sets with an ID number (link to the NCBI Data Set Browser), title, organism used in the experiment, number of samples, features, genes, subsets and a reference number for the PubMed journal (link to the article abstract).
5. Short description of the experiment from which the data set is sourced.
6. Select which *Sample Annotations* will be used in the output.

### 1.2.1 Example

**GEO Data Sets** is similar to the **File** widget, since it is used to load the data. In the example below we selected *Caffeine effect: time course and dose response* dataset from the GEO data base. Do not forget to press *Commit* to output the data. We can inspect the data in *Data Table*.




## 1.3 dictyExpress

Gives access to dictyExpress databases.

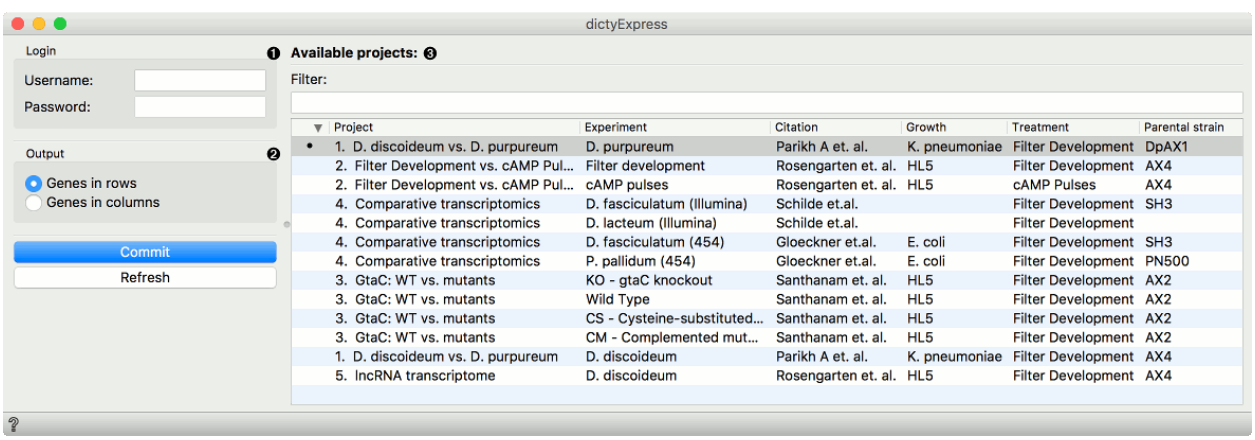
### Inputs

- None

### Outputs

- Data: Selected experiment (time-course gene expression data).

**dictyExpress** widget gives a direct access to the dictyExpress database. It allows you to download the data from selected experiments in *Dictyostelium* by Baylor College of Medicine. The widget requires internet connection to work.

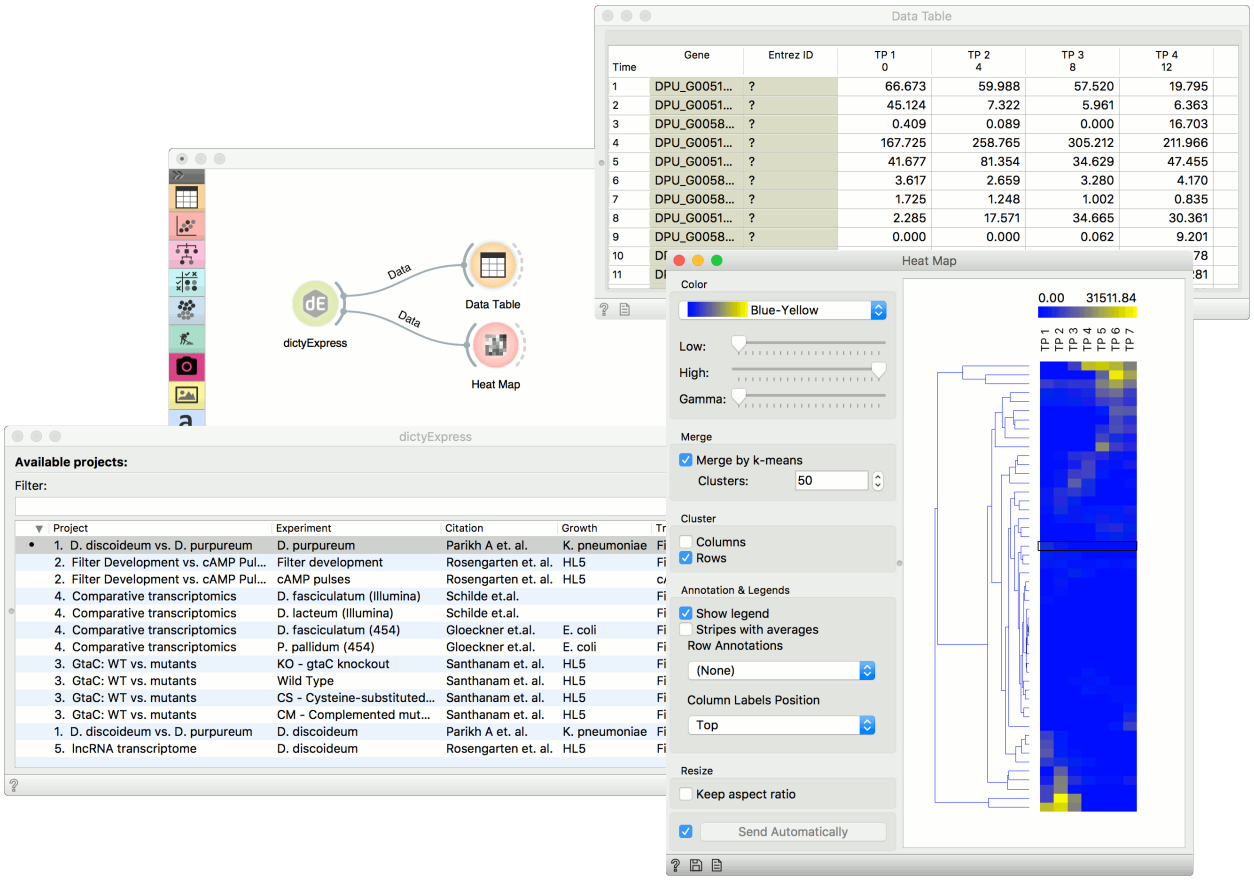



1. Log into the database to access personal files.
2. Define the output. Genes from experiments can be either in rows or in columns. Press *Commit* to output the data and *Refresh* to update the list of experiments.

3. List of available experiments. Use *Filter* to find a particular experiment.

### 1.3.1 Example

**dictyExpress** widget can be used to retrieve data from a database, just like **GEO Data Sets** and similar to the **File** widget. We have retrieved the *D. discoideum vs. D. purpureum* data and sent it to the output by pressing *Commit*. We have observed the data in a **Data Table** and in a **Heat Map**, where we used *Merge by k-means* and clustering by rows to find similar genes.




## 1.4 Gene Name Matcher

Match input gene ID's with corresponding Entrez ID's.

**Inputs**

- Data: Data set.

**Outputs**

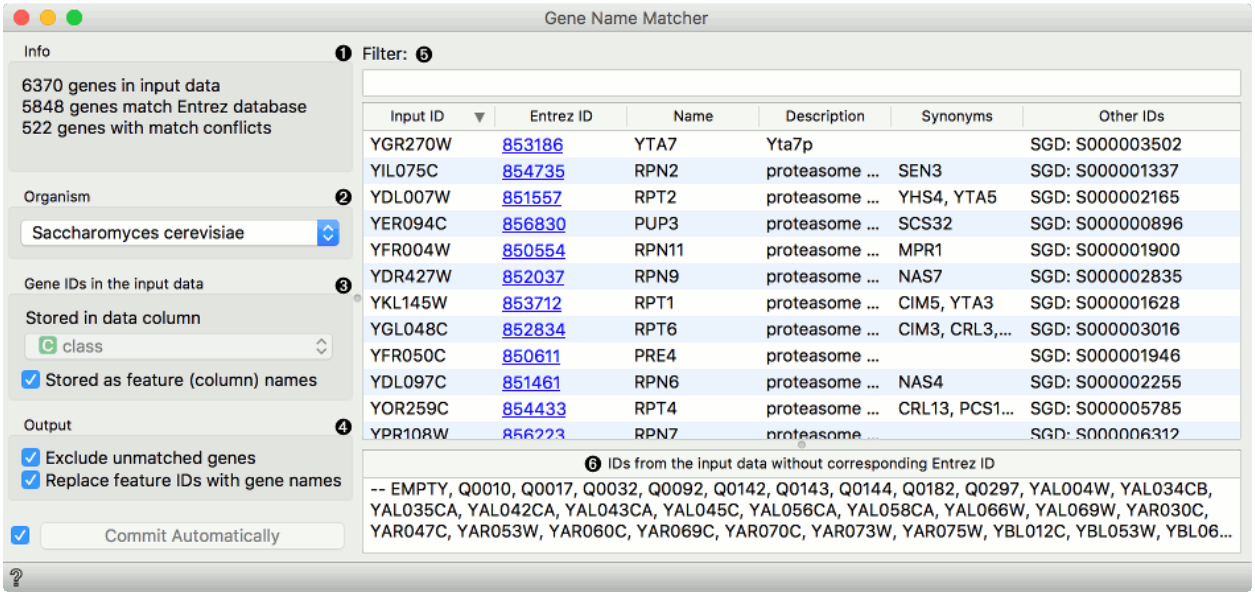
- Data: Instances with meta data that the user has manually selected in the widget.
- Genes: All genes from the input with included gene info summary and matcher result.

To work with widgets in the bioinformatics add-on data sets must be properly annotated. We need to specify:

- Location of genes in a table (rows, columns)

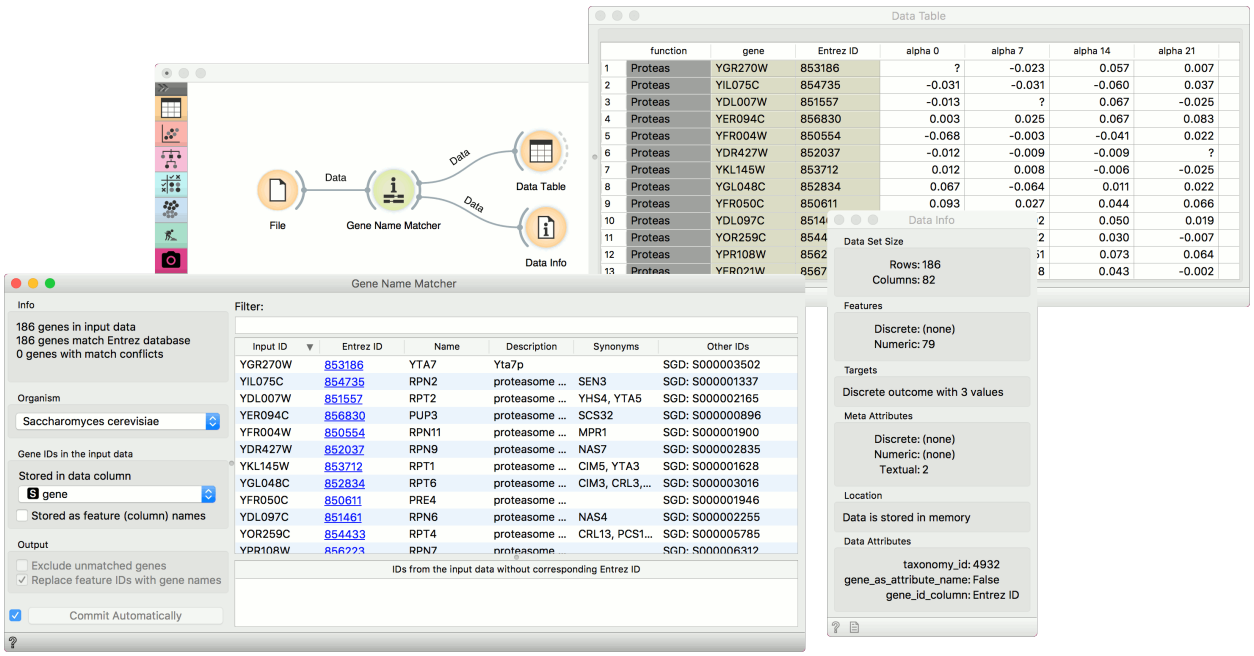
- ID from the NCBI Gene database (Entrez ID)
- Organism (Taxonomy ID)

Gene Name Matcher is a useful widget that presents information on the genes from the NCBI Gene database and outputs annotated data table. You can also select a subset and feed it to other widgets. By clicking on the gene Entrez ID in the list, you will be taken to the NCBI site with the information on the gene.




### 1.4.1 Example

First we load *brown-selected.tab* (from *Browse documentation data sets*) with the **File** widget and feed our data to the Gene Name Matcher. Orange recognized the organism correctly, but we have to tell it where our gene labels are. To do this, we tick off *Stored as feature (column) name* and select *gene* attribute from the list. Then we can observe gene info provided from the NCBI Gene database. In the **Data Table** we can see the Entrez ID column included as a meta attribute. The data is also properly annotated (see *Data Attributes* section in **Data Info** widget).




## 1.5 Differential Expression

Plots differential gene expression for selected experiments.

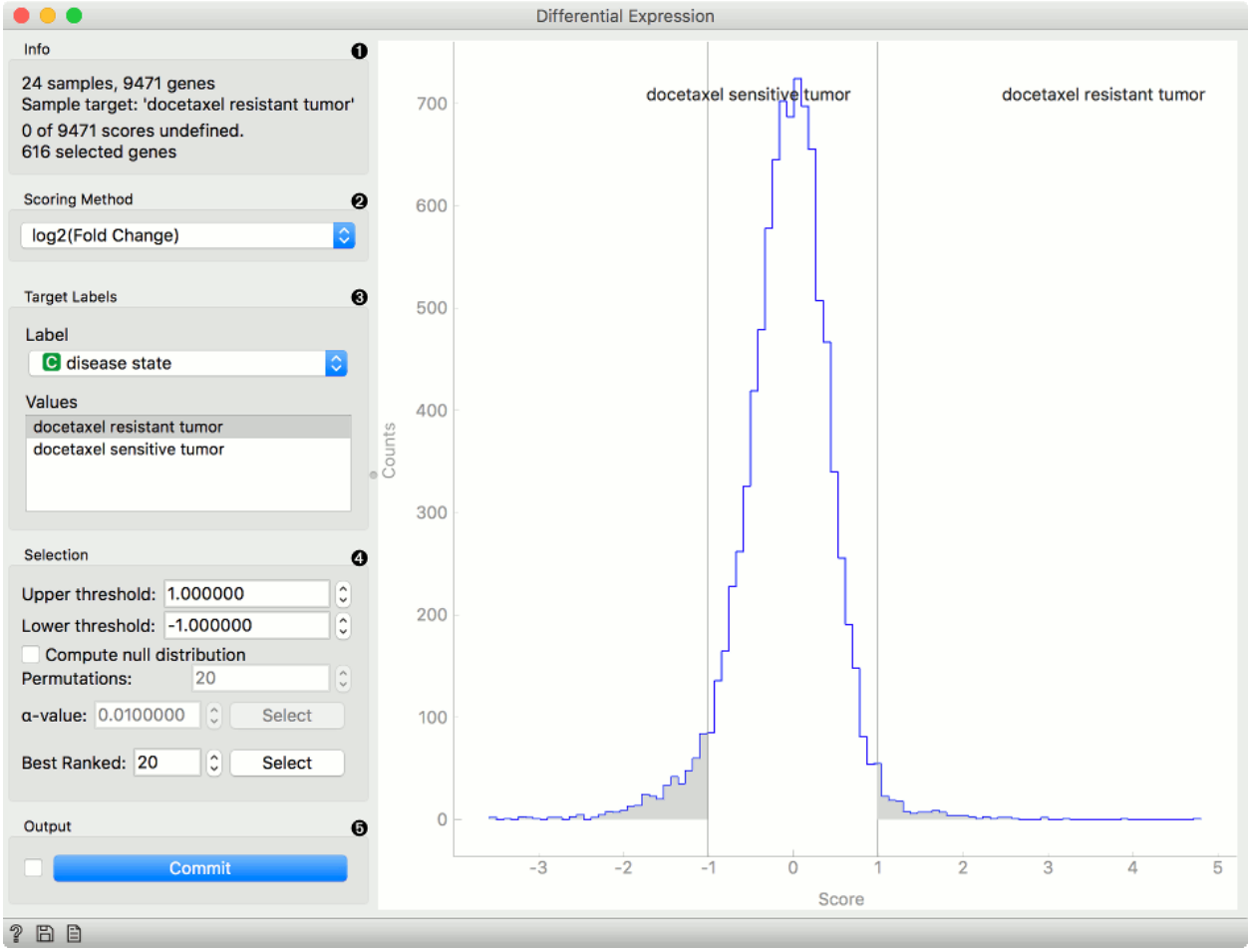
### Inputs

- Data: Data set.

### Outputs

- Data Subset: Differentially expressed genes.
- Remaining Data Subset: Genes that were not differentially expressed.
- Selected Genes: Genes from the select data with scores appended.

This widget plots a differential gene expression graph for a sample target. It takes gene expression data as an input (from **dictyExpress**, **GEO Data Sets**, etc.) and outputs a selected data subset (normally the most interesting genes).



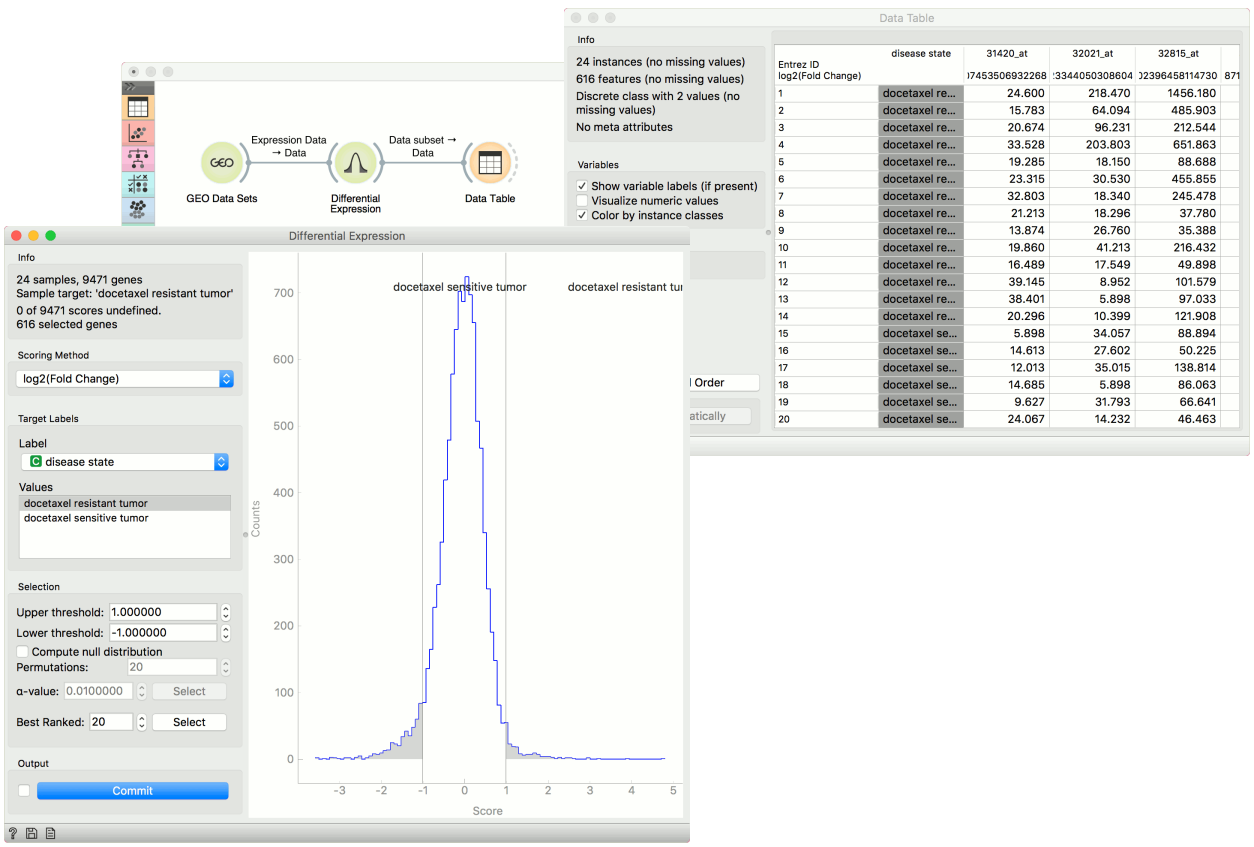

1. Information of the data input and output. The first line shows the number of samples and genes in the data set. The second line displays the selected sample target (read around which the graph is plotted). The third line shows the number of undefined genes (missing data) and the fourth the number of genes in the output.
2. Select the plotting method in *Scoring method*:
   - Fold change: final to initial value ratio
   - **log2 (fold change)**: binary logarithmic transformation of fold change values
   - T-test: parametric test of null hypothesis
   - **T-test (P-value)**: parametric test of null hypothesis with P-value as criterium
   - ANOVA: variance distribution
   - **ANOVA (P-value)**: variance distribution with P-value as criterium
   - Signal to NoiseRatio: biological signal to noise ratio
   - Mann-Whitney: non-parametric test of null hypothesis with P-value as criterium
   - Hypergeometric test: for binary expression data.
3. Select *Target Labels*. Labels depend on the attributes in the input. In *Values* you can change the sample target (default value is the first value on the list, alphabetically or numerically).
4. *Selection* box controls the output data.

- By setting the *Lower threshold* and *Upper threshold* values you are outputting the data outside this interval (the most interesting expression levels). You can also manually place the threshold lines by dragging left or right in the plot.
- If you click *Compute null distribution* box, the widget will calculate null distribution and display it in the plot. *Permutations* field allows you to set the precision of null distribution (the more permutations the more precise the distribution), while alpha-value will be the allowed probability of false positives. Press *Select* to output this data.
- The final option is to set the number of best ranked genes and output them with *Select*.

5. When *Auto commit is on* is ticked, the widget will automatically apply the changes. Alternatively press *Commit*. If the *Add gene scores to output* is ticked, the widget will append an additional column with gene scores to the data.

### 1.5.1 Example

From the GEO Data Sets widget, we selected *Breast cancer and docetaxel treatment* (GDS360) with 14 treatment resistant and 10 treatment sensitive tumors. Then we used the **Differential Expression** widget to select the most interesting genes. We left the upper and lower threshold at default (1 and -1) and output the data. Then we observed the selected data subset in a **Data Table**. The table shows selected genes with an additional gene score label.




## 1.6 GO Browser

Provides access to Gene Ontology database.

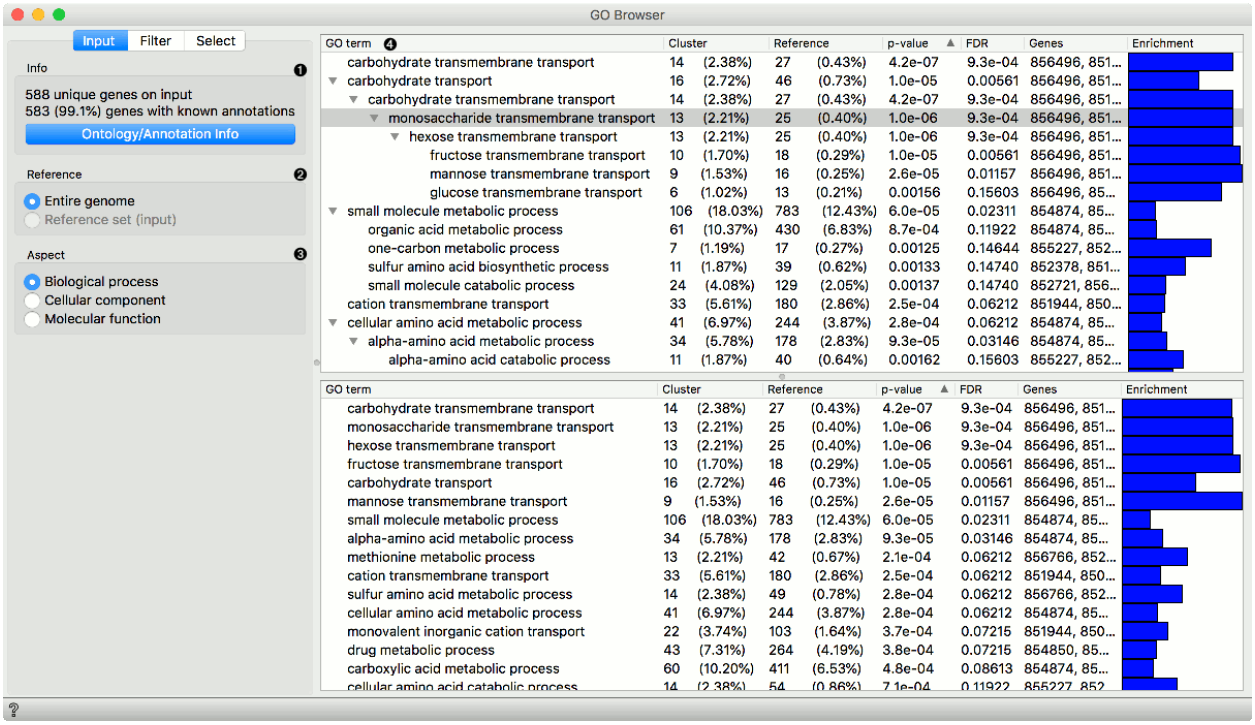
### Inputs

- Cluster Data: Data on clustered genes.
- Reference Data: Data with genes for the reference set (optional).

### Outputs

- Data on Selected Genes: Data on genes from the selected GO node.
- Enrichment Report: Data on GO enrichment analysis.

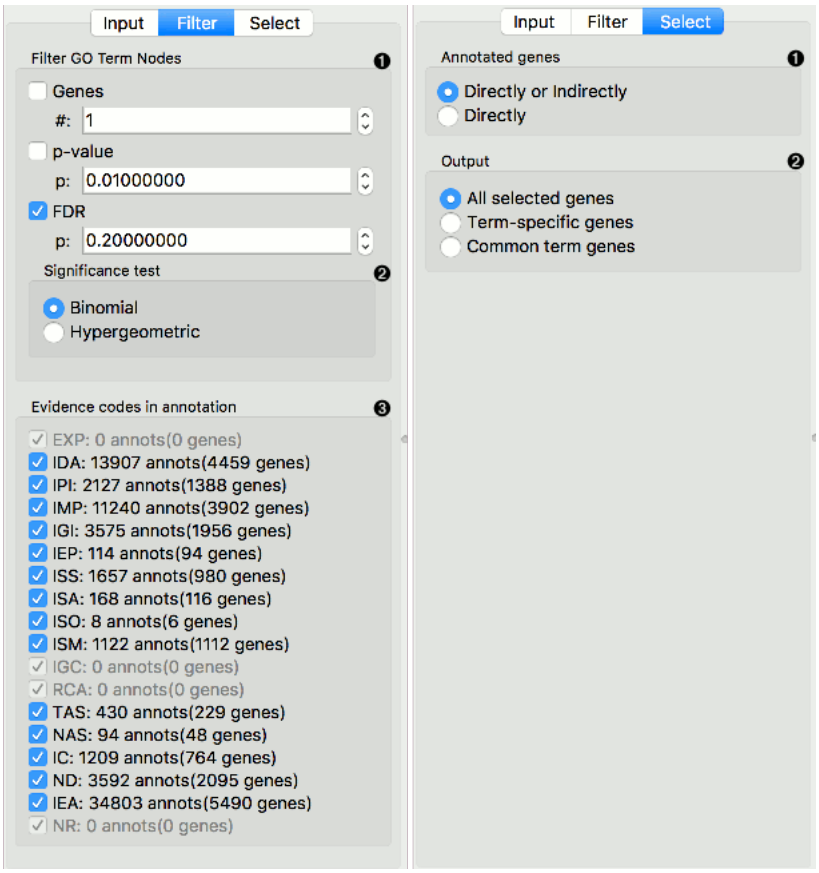
**GO Browser** widget provides access to Gene Ontology database. Gene Ontology (GO) classifies genes and gene products to terms organized in a graph structure called an ontology. The widget takes any data on genes as an input (it is best to input statistically significant genes, for example from the output of the **Differential Expression** widget) and shows a ranked list of GO terms with p-values. This is a great tool for finding biological processes that are over- or under-represented in a particular gene set. The user can filter input data by selecting terms in a list.




### INPUT tab

1. Information on the input data set. *Ontology/Annotation Info* reports the current status of the GO database.
2. Select the reference. You can either have the *entire genome* as reference or a *reference set* from the input.
3. Select the ontology where you want to calculate the enrichment. There are three *Aspect* options:
   - Biological process
   - Cellular component)
   - Molecular function
4. A ranked tree (upper pane) and list (lower pane) of GO terms for the selected aspect:
   - **GO term**
   - **Cluster**: number of genes from the input that are also annotated to a particular GO term (and its proportion in all the genes from that term).

- **Reference**: number of genes that are annotated to a particular GO term (and its proportion in the entire genome).
- **P-value**: probability of seeing as many or more genes at random. The closer the p-value is to zero, the more significant a particular GO term is. Value is written in e notation).
- **FDR**: false discovery rate - a multiple testing correction that means a proportion of false discoveries among all discoveries up to that FDR value.
- **Genes**: genes in a biological process.
- Enrichment level




**FILTER tab**

1. *Filter GO Term Nodes* by:
   - **Genes** is a minimal number of genes mapped to a term
   - **P-value** is a max term p-value
   - **FDR**: is a max term false discovery rate
2. *Significance test* specifies distribution to use for null hypothesis:
   - Binomial: use a binomial distribution
   - Hypergeometric: use a hypergeometric distribution
3. Evidence codes in annotation show how the annotation to a particular term is supported.

**SELECT tab**

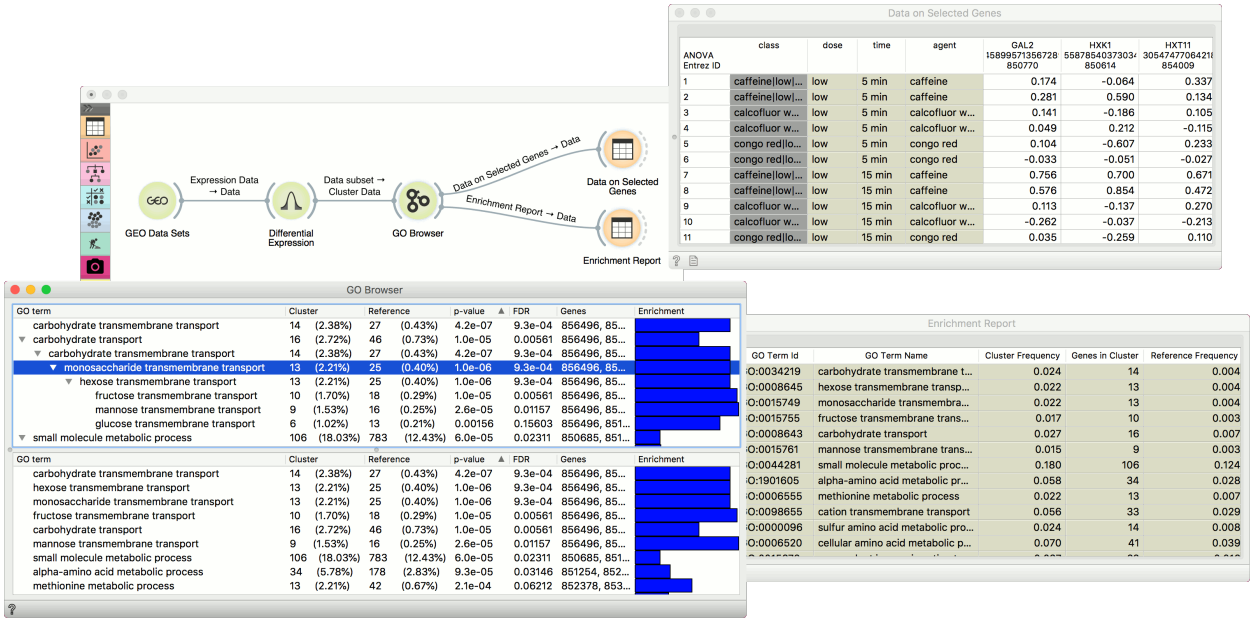
1. *Annotated genes* outputs genes that are:

- **Directly or Indirectly** annotated (direct and inherited annotations)
- **Directly** annotated (inherited annotations won't be in the output)

2. *Output*:
   - **All selected genes**: outputs genes annotated to all selected GO terms
   - **Term-specific genes**: outputs genes that appear in only one of selected GO terms
   - **Common term genes**: outputs genes common to all selected GO terms
   - **Add GO Term as class**: adds GO terms as class attribute

### 1.6.1 Example

In the example below we have used **GEO Data Sets** widget, in which we have selected *Caffeine effects: time course and dose response* data set, and connected it to a **Differential Expression**. Differential analysis allows us to select genes with the highest statistical relevance (we used ANOVA scoring and agent label) and feed them to **GO Browser**. This widget lists four biological processes for our selected genes. Say we are interested in finding out more about *monosaccharide transmembrane transport* as this term has a high enrichment rate. To learn more about which genes are annotated to this GO term, select it in the view and observe the results in a **Data Table**, where we see all the genes participating in this process listed. The other output of **GO Browser** widget is enrichment report, which we observe in the second **Data Table**.




## 1.7 KEGG Pathways

Diagrams of molecular interactions, reactions, and relations.

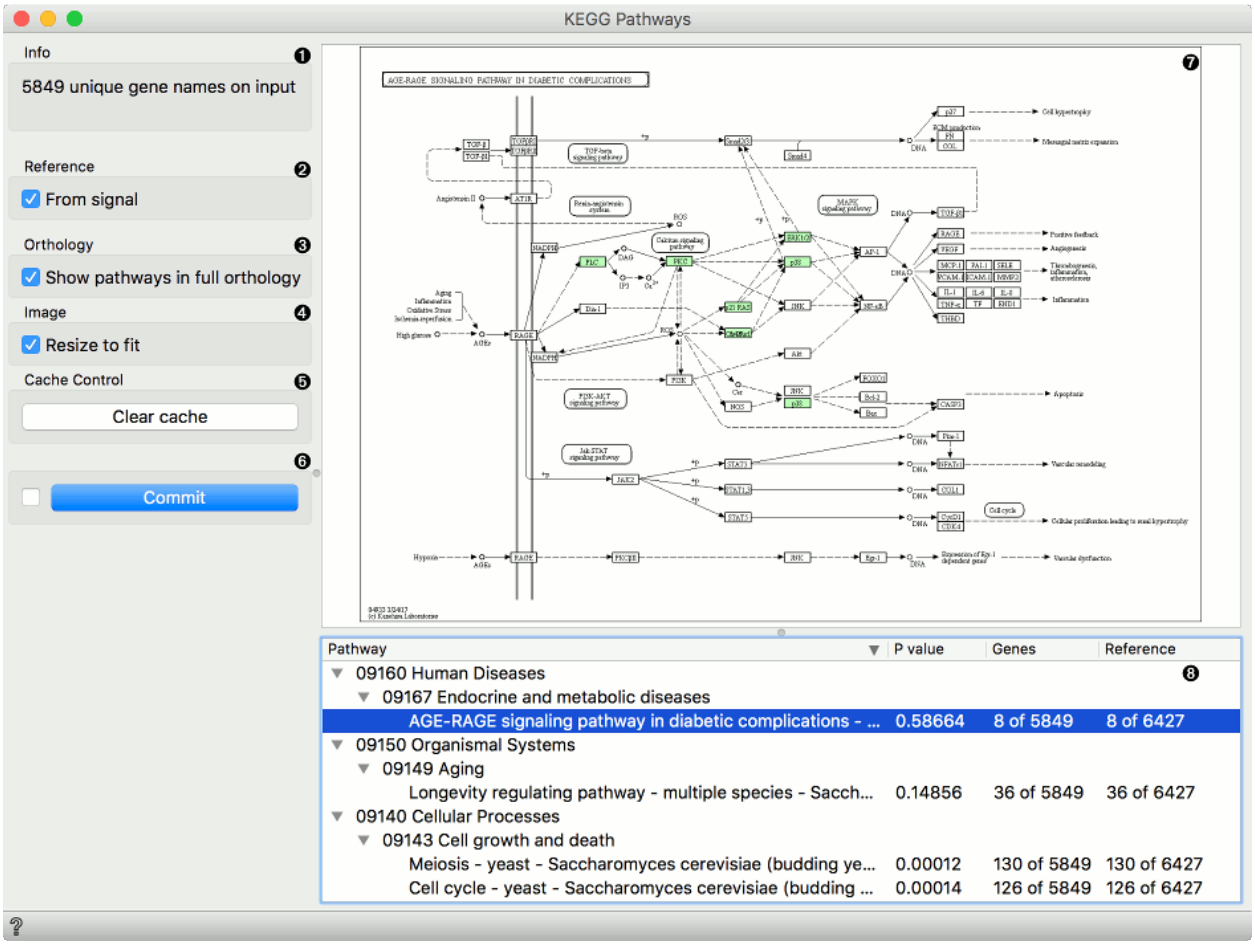
**Inputs**

- Data: Data set.
- Reference: Referential data set.

**Outputs**

- Selected Data: Data subset.
- Unselected Data: Remaining data.

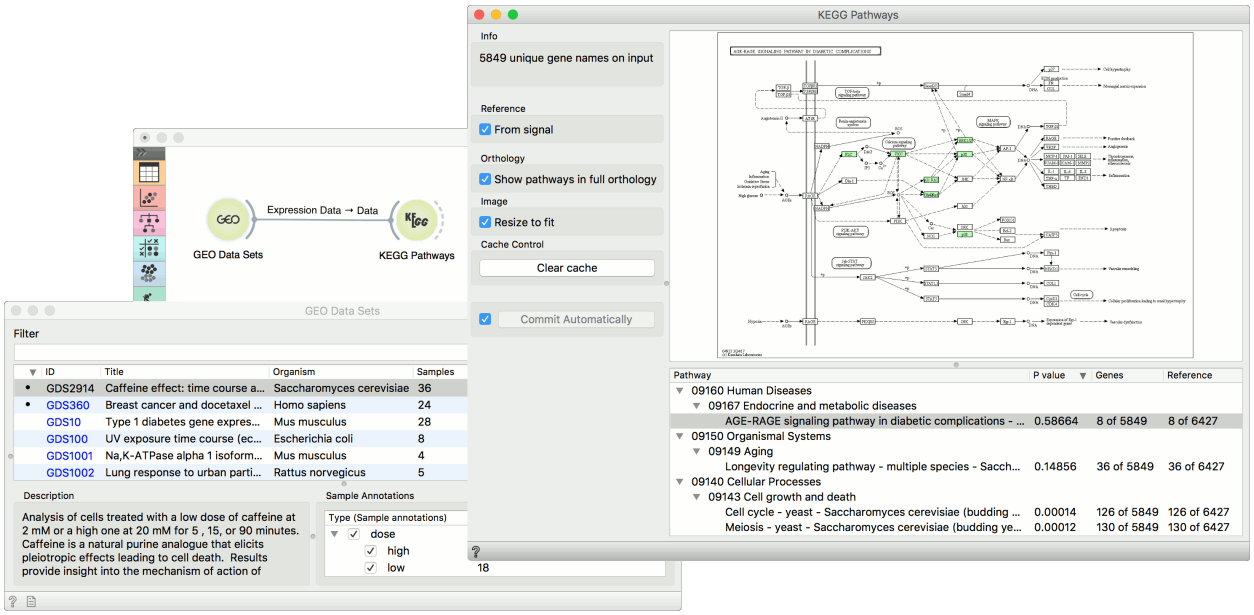
**KEGG Pathways** widget displays diagrams of molecular interactions, reactions and relations from the KEGG Pathways Database. It takes data on gene expression as an input, matches the genes to the biological processes and displays a list of corresponding pathways. To explore the pathway, the user can click on any process from the list or arrange them by P-value to get the most relevant processes at the top.




1. Information on the input genes.
2. If you have a separate reference set in the input, tick *From signal* to use these data as reference.
3. To have pathways listed and displayed by vertical descent, tick *Show pathways in full orthology*.
4. To fit the image to screen, tick *Resize to fit*. Untick the box if you wish to explore the pathways.
5. To clear all locally cached KEGG data, press *Clear cache*.
6. When *Auto commit is on*, the widget will automatically apply the changes. Alternatively press *Commit*.
7. A list of pathways either as processes or in full orthology. Click on the process to display the pathway. You can sort the data by P-value to get the most relevant results at the top.

### 1.7.1 Example

This simple example shows how to visualize interactions with **KEGG Pathways**. We have loaded the *Caffeine effect: time courses and dose response* (GDS2914) data with the **GEO Data Sets** widget. Then we have observed the pathways in **KEGG Pathways**. We have used reference from signal and selected *AGE-RAGE signaling pathway in diabetic complications*.




## 1.8 Gene Set Enrichment

Enrich gene sets.

**Inputs**

- Data: Data set.
- Custom Gene Sets: Genes to compare.
- Reference Genes: Genes used as reference.

**Outputs**

- Matched Genes: Gene that match.

TODO Description.

`widgets/images/gene_set_enrichment/Gene-Set-Enrichment-stamped.png`

1. Info
2. Custom Gene Set Term Column
3. Reference

4. Gene Sets
5. If *Commit Automatically* is ticked, results will be automatically sent to the output. Alternatively, press *Commit*.
6. Filtering.

### 1.8.1 Example

TODO Example

## 1.9 Cluster Analysis

Display differentially expressed genes that characterize the cluster.

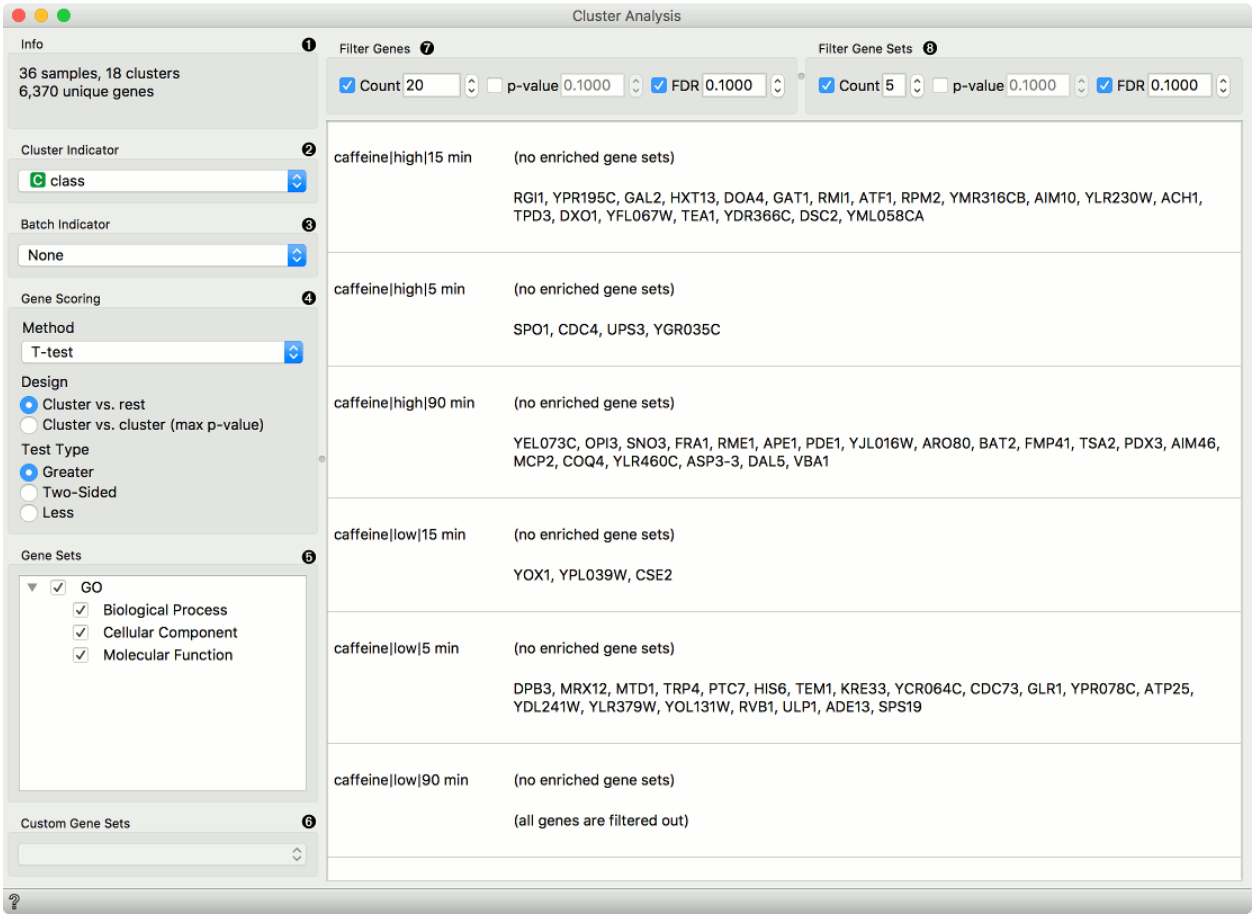
**Inputs**

- Data: Data set.
- Custom Gene Sets: Genes to compare.

**Outputs**

- Selected Data: Data selected in the widget.

**Cluster Analysis** widget displays differentially expressed genes that characterize the cluster, and corresponding gene terms that describe differentially expressed genes.



1. Info
2. Cluster Indicator
3. Batch Indicator
4. Gene Scoring
5. Gene Sets
6. Custom Gene Sets
7. Filter Genes
8. Filter Gene Sets

### 1.9.1 Example

TODO: Example

CHAPTER 2

# Scripting Reference

## 2.1 Organism Taxonomy (`taxonomy`)

This module provides access to the NCBI's organism taxonomy information and organism name unification across different modules.

`orangecontrib.bioinformatics.ncbi.taxonomy.`**`name`**(*tax_id*)

Return the scientific name for organism with provided taxonomy id.

**Parameters** **`tax_id`** (*`str`*) – Taxonomy if (NCBI taxonomy database)

`orangecontrib.bioinformatics.ncbi.taxonomy.`**`other_names`**(*tax_id*)

Return a list of (name, name_type) tuples excluding the scientific name.

Use *`name()`* to retrieve the scientific name.

**Parameters** **`tax_id`** (*`str`*) – Taxonomy if (NCBI taxonomy database)

`orangecontrib.bioinformatics.ncbi.taxonomy.`**`search`**(*string*, *onlySpecies=True*, *exact=False*)

Search the NCBI taxonomy database for an organism.

**Parameters**

- **`string`** (*`str`*) – Search string.
- **`onlySpecies`** (*`bool`*) – Return only taxids of species (and subspecies).
- **`exact`** (*`bool`*) – Return only taxids of organism that exactly match the string.

`orangecontrib.bioinformatics.ncbi.taxonomy.`**`lineage`**(*tax_id*)

Return a list of taxids ordered from the topmost node (root) to taxid.

**Parameters** **`tax_id`** (*`str`*) – Taxonomy if (NCBI taxonomy database)

`orangecontrib.bioinformatics.ncbi.taxonomy.`**`common_taxids`**()

Return taxonomy IDs for most common organisms.

**Return type** `list` of common organisms

orangecontrib.bioinformatics.ncbi.taxonomy.**common_taxid_to_name**(*tax_id*)
Return a name for a common organism taxonomy id.

**Parameters** **tax_id** (*str*) – Taxonomy if (NCBI taxonomy database)

**Return type** str

orangecontrib.bioinformatics.ncbi.taxonomy.**taxname_to_taxid**(*organism_name*)
Return taxonomy ID for a taxonomy name.

**Parameters** **organism_name** (*str*) – Official organism name, e.g., 'Homo sapiens'

**Return type** str

### 2.1.1 Examples

The following script takes the list of taxonomy IDs and prints out their name:

```
import orangecontrib.bioinformatics.ncbi.taxonomy

for taxid in orangecontrib.bioinformatics.ncbi.taxonomy.common_taxids():
    print("%-6s %s" % (taxid, orangecontrib.bioinformatics.ncbi.taxonomy.name(taxid)))
```

The output of the script is:

```
3702   Arabidopsis thaliana
9913   Bos taurus
6239   Caenorhabditis elegans
5476   Candida albicans
3055   Chlamydomonas reinhardtii
7955   Danio rerio
352472 Dictyostelium discoideum AX4
7227   Drosophila melanogaster
562    Escherichia coli
11103  Hepatitis C virus
9606   Homo sapiens
10090  Mus musculus
2104   Mycoplasma pneumoniae
4530   Oryza sativa
5833   Plasmodium falciparum
4754   Pneumocystis carinii
10116  Rattus norvegicus
4932   Saccharomyces cerevisiae
4896   Schizosaccharomyces pombe
31033  Takifugu rubripes
8355   Xenopus laevis
4577   Zea mays
```

## 2.2 Gene name matching and ncbi info (gene)

### 2.2.1 Example

```
from orangecontrib.bioinformatics.ncbi.gene import GeneMatcher, GENE_INFO_TAGS

# specify input
```

(continues on next page)

(continued from previous page)

```
organism = 9606
genes_symbols_to_match = ['HB1', 'BCKDHB', 'TWIST1']

# initialize gene matcher object
gene_matcher = GeneMatcher(organism)
gene_matcher.genes = genes_symbols_to_match

# run matching process
gene_matcher.run_matcher()

# inspect results
for gene in gene_matcher.genes:
    print("\ninput name: " + gene.input_name,
          "\nid from ncbi: ", gene.ncbi_id,
          "\nmatch type: ", gene.type_of_match
          )
    if gene.ncbi_id is None and gene.possible_hits:
        print('possible_hits: ', [hit.ncbi_id for hit in gene.possible_hits])
```

Output:

```
input name: HB1
id from ncbi:  None
match type:  None
possible_hits:  [3887, 6331, 8184]

input name: BCKDHB
id from ncbi:  594
match type:  Symbol match

input name: TWIST1
id from ncbi:  7291
match type:  Symbol match
```

Two out of three genes had a unique match with corresponding ncbi gene id. Symbol 'HB1' is used in multiple genes so we store them for further analysis.

One can also display gene information from NCBI database.

```
gene_of_interest = gene_matcher.genes[0].possible_hits[0]
gene_of_interest.load_ncbi_info()

for tag in GENE_INFO_TAGS:
    print(tag + ':', getattr(gene_of_interest, tag))
```

Output:

```
tax_id: 9606
gene_id: 3887
symbol: KRT81
synonyms: |HB1|Hb-1|KRTHB1|MLN137|ghHkb1|hHAKB2-1|
db_refs: MIM:602153|HGNC:HGNC:6458|Ensembl:ENSG00000205426|Vega:OTTHUMG00000167574
description: keratin 81
locus_tag: -
chromosome: 12
map_location: 12q13.13
type_of_gene: protein-coding
```

(continues on next page)

(continued from previous page)

```
symbol_from_nomenclature_authority: KRT81
full_name_from_nomenclature_authority: keratin 81
nomenclature_status: O
other_designations: keratin, type II cuticular Hb1|K81|MLN 137|ghHb1|hair keratin K2.
↪9|hard keratin, type II, 1|keratin 81, type II|keratin, hair, basic, 1|metastatic␣
↪lymph node 137 gene protein|type II hair keratin Hb1|type-II keratin Kb21
modification_date: 20171105
```

### 2.2.2 Class References

## 2.3 NCBI's Gene Expression Omnibus interface (geo)

This module provides an interface to NCBI's Gene Expression Omnibus repository. It supports GEO DataSets query and retrieval.

In the following example *GDS.get_data* construct a data set with genes in rows and samples in columns. Notice that the annotation about each sample is retained in `.attributes`.

```
>>> from orangecontrib.bioinformatics.geo.dataset import GDS
>>> gds = GDS("GDS1676")
>>> data = gds.get_data()
>>> len(data)
719
>>> data[200]
[0.503, 0.690, 0.607, -2.250, 0.000, ...] {CD40}
>>> data.domain.attributes[0]
ContinuousVariable(name='GSM63816', number_of_decimals=3)
>>> data.domain.attributes[0].attributes
{'infection': 'acute', 'time': '1 d', 'dose': '20 U/ml IL-2'}
```

### 2.3.1 Class References

**class** orangecontrib.bioinformatics.geo.dataset.**GDSInfo**

**__init__**()

Retrieve infomation about GEO DataSets.

The class accesses the Orange server file that either resides on the local computer or is automatically retrieved from Orange server. Calls to this class do not access any NCBI's servers.

Constructor returning the object with GEO DataSets information. The constructor will download GEO DataSets information file (gds_info.pickled) from Orange server, it will first check the local copy.

An instance behaves like a dictionary: the keys are GEO DataSets IDs, and the dictionary values for is a dictionary providing various information about the particular data set.

#### Example

```
>>> info = GDSInfo()
>>> list(info.keys())[:5]
```

(continues on next page)

(continued from previous page)

```
['GDS10', 'GDS100', 'GDS1001', 'GDS1002', 'GDS1003']
>>> info['GDS10']['title']
'Type 1 diabetes gene expression profiling'
>>> info['GDS10']['platform_organism']
'Mus musculus'
```

**class** orangecontrib.bioinformatics.geo.dataset.**GDS**

**get_data**(*report_genes=True*, *merge_function=<function spots_mean>*, *sample_type=None*, *transpose=False*)

**Parameters**

- **report_genes** (*bool*) – Microarray spots reported in the GEO data set can either be merged according to their gene ids (if True) or can be left as spots.
- **sample_type** – the type of annotation, or (if `transpose` is True) the type of class labels to be included in the data set. The entire annotation of samples will be included either in the class value or in the `.attributes` field of each data set attribute.
- **transpose** – The output table can have spots/genes in rows and samples in columns (False, default) or samples in rows and spots/genes in columns (True).
- **merge_function** –

**Returns** the GEO DataSet as an `Orange.data.Table`.

**parse_file**(*remove_unknown=None*)

Parse GDS data file. Create self.info and self.gds_data

### 2.3.2 More Examples

The following script prints out information about a specific data set. It does not download the data set, just uses the (local) GEO data sets information file (`dataset_info.py`).

```
""" Documentation script """
import textwrap


from orangecontrib.bioinformatics.geo.dataset import GDSInfo

gds_info = GDSInfo()
gds = gds_info["GDS10"]

print("ID:")
print(gds["dataset_id"])
print("Features: ")
print(gds["feature_count"])
print("Genes:")
print(gds["gene_count"])
print("Organism:")
print(gds["platform_organism"])
print("PubMed ID:")
print(gds["pubmed_id"])
print("Sample types:")

for sample_type in set([sinfo["type"] for sinfo in gds["subsets"]]):
```

(continues on next page)

(continued from previous page)

```
    ss = [sinfo["description"] for sinfo in gds["subsets"] if sinfo["type"] == sample_
↪type]
    print("  %s (%s)" % (sample_type, ", ".join(ss)))

print("")
print("Description:")
print("\n".join(textwrap.wrap(gds["description"], 70)))
```

The output of this script is:

```
ID:
GDS10
Features:
39114
Genes:
29942
Organism:
Mus musculus
PubMed ID:
11827943
Sample types:
  tissue (spleen, thymus)
  disease state (diabetic, diabetic-resistant, nondiabetic)
  strain (NOD, Idd3, Idd5, Idd3+Idd5, Idd9, B10.H2g7, B10.H2g7 Idd3)

Description:
Examination of spleen and thymus of type 1 diabetes nonobese diabetic
(NOD) mouse, four NOD-derived diabetes-resistant congenic strains and
two nondiabetic control strains.
```

Samples in GEO data sets belong to sample subsets, which in turn belong to specific types. The above GDS10 has three sample types, of which the subsets for the tissue type are spleen and thymus. For supervised data mining it would be useful to find out which data sets provide enough samples for each label. It is (semantically) convenient to perform classification within sample subsets of the same type. The following script goes through all data sets and finds those with enough samples within each of the subsets for a specific type. The function `valid` determines which subset types (if any) satisfy our criteria (`dataset_samples.py`).

```
""" Documentation script

Check all data files from GEO, find those which include at least N
samples in all sample subsets of at least one sample type. Useful
when, for instance, filtering out the data sets that could be used for
supervised machine learning.
"""

from orangecontrib.bioinformatics.geo.dataset import GDSInfo, GDS


def valid(info, n=40):
    """Return a set of subset types containing more than n samples in every subset"""
    invalid = set()
    subsets = set([sinfo["type"] for sinfo in info["subsets"]])
    for sampleinfo in info["subsets"]:
        if len(sampleinfo["sample_id"]) < n:
            invalid.add(sampleinfo["type"])
    return subsets.difference(invalid)
```

(continues on next page)

(continued from previous page)

```
def report(stypes, info):
    """Pretty-print GDS and valid susbset types"""
    for id, sts in stypes:
        print(id)
        for st in sts:
            gds = info[id]
            print("  %s:" % st +
                  ", ".join(["%s/%d" % (sinfo["description"], len(sinfo["sample_id"]))
                             for sinfo in gds["subsets"] if sinfo["type"] == st]))



gdsinfo = GDSInfo()
valid_subset_types = [(id, valid(info)) for id, info in sorted(gdsinfo.items()) if␣
↪valid(info)]
report(valid_subset_types, gdsinfo)

print('datasets = ' + str(len(valid_subset_types)))
print('type subsets = ' + str(sum(len(b) for _, b in valid_subset_types)))
```

The requested number of samples, `n=40`, seems to be a quite a stringent criteria met - at the time of writing this - by 40 data sets with 48 sample subsets. The output starts with:

```
GDS1292
  tissue:raphe magnus/40, somatomotor cortex/43
GDS1293
  tissue:raphe magnus/40, somatomotor cortex/41
GDS1412
  protocol:no treatment/47, hormone replacement therapy/42
GDS1490
  other:non-neural/50, neural/100
GDS1611
  genotype/variation:wild type/48, upf1 null mutant/48
GDS2373
  gender:male/82, female/48
GDS2808
  protocol:training set/44, validation set/50
```

Let us now pick data set GDS2960 and see if we can predict the disease state. We will use logistic regression, and within 10-fold cross validation measure AUC, the area under ROC. AUC is the probability of correctly distinguishing the two classes, (e.g., the disease and control). From (`predict_disease_state.py`):

```
""" Documentation script """
from Orange.classification import LogisticRegressionLearner
from Orange.evaluation.testing import CrossValidation
from Orange.evaluation.scoring import AUC

from orangecontrib.bioinformatics.geo.dataset import GDS

gds = GDS("GDS2960")
data = gds.get_data(sample_type="disease state", transpose=True, report_genes=True)
print("Samples: %d, Genes: %d" % (len(data), len(data.domain.attributes)))

learners = [LogisticRegressionLearner()]
```

(continues on next page)

(continued from previous page)

```
results = CrossValidation(data, learners, k=10)

print("AUC = %.3f" % AUC(results)[0])
```

The output of this script is:

```
Samples: 101, Genes: 4069
AUC = 0.996
```

The AUC for this data set is very high, indicating that using these gene expression data it is almost trivial to separate the two classes.

## 2.4 Gene Ontology (go)

Provides access to Gene Ontology and its gene annotations.

**class** orangecontrib.bioinformatics.go.**Ontology**

*Ontology* is the class representing a gene ontology.

**Parameters**

- **filename** (*str*) – A filename of an .obo formated file.
- **progress_callback** – Optional *float* -> *None* function.

### Example

```
>>> # Load the current ontology (downloading it if necessary)
>>> ontology = Ontology()
```

```
>>> term_ids = list(ontology)
>>> term = ontology[term_ids[0]]
```

**__contains__**(*termid*)

Return *True* if a term with *termid* is present in the ontology.

**__getitem__**(*termid*)

Return a *Term* object with *termid*.

**Parameters** **term** (*str*) – An id of a 'Term' in the ontology.

**Return type** *Term*

**__iter__**()

Iterate over all term ids in ontology.

**__len__**()

Return number of terms in ontology.

**defined_slims_subsets**()

Return a list of defined subsets in the ontology.

**Return type** `list` of `str`

**extract_sub_graph**(*terms*)

Return all sub terms of *terms*.

**Parameters** **terms** (*list*) – A list of term IDs.

**extract_super_graph**(*terms*)
Return all super terms of *terms* up to the most general one.

**Parameters** **terms** (*list*) – A list of term IDs.

**named_slims_subset**(*subset*)
Return all term IDs in a named *subset*.

**Parameters** **subset** (*str*) – A string naming a subset in the ontology.

**Return type** `list` of `str`

**See also:**

*defined_slims_subsets()*

**set_slims_subset**(*subset*)
Set the *slims_subset* term subset to *subset*.

**Parameters** **subset** (*set*) – A subset of GO term IDs.

*subset* may also be a string, in which case the call is equivalent to `ont.set_slims_subsets(ont.named_slims_subset(subset))`

**slims_for_term**(*term*)
Return a list of slim term IDs for *term*.

This is a list of *most specific* slim terms to which *term* belongs.

**Parameters** **term** (*str*) – Term ID.

**class** orangecontrib.bioinformatics.go.**Term**

**id**
The term id.

**namespace**
The namespace of the term.

**def_**
The term definition (Note the use of trailing underscore to avoid conflict with a python keyword).

**is_a**
List of term ids this term is a subterm of (parent terms).

**related**
List of (rel_type, term_id) tuples with rel_type specifying the relationship type with term_id.

**class** orangecontrib.bioinformatics.go.**Annotations**
*Annotations* object holds the annotations.

**Parameters**

- **organism** (*str*) – an organism specifier (e.g. `'9606'`). Annotations for that organism will be loaded.
- **ontology** (*Ontology*) – *Ontology* object for annotations

**gene_annotations = None**
A dictionary mapping a gene (gene_id) to a set of all annotations of that gene.

**term_anotations = None**
A dictionary mapping a GO term id to a set of annotations that are directly annotated to that term

**annotations = None**
A list of all `AnnotationRecords` instances.

**add_annotation**(*a*)
Add a single `AnotationRecord` instance to this object.

**get_genes_with_known_annotation**(*genes*)
Return only genes with known annotation

**Parameters** **genes** – List of genes

**get_annotations_by_go_id**(*go_id*)
Return a set of all annotations (instances of `AnnotationRecord`) for GO term *id* and all it's subterms.

:param `str` go_id: GO term id

**get_genes_by_go_term**(*go_id*, *evidence_codes=None*)
Return a list of genes annotated by specified *evidence_codes* to GO term 'id' and all it's subterms."

**Parameters**

- **go_id** (*str*) – GO term id
- **evidence_codes** (*list-of-strings*) – List of evidence codes to consider when matching annotations to terms.

**get_enriched_terms**(*genes*, *reference=None*, *evidence_codes=None*, *slims_only=False*, *aspect=None*, *prob=<orangecontrib.bioinformatics.utils.statistics.Binomial object>*, *use_fdr=True*, *progress_callback=None*)
Return a dictionary of enriched terms, with tuples of (list_of_genes, p_value, reference_count) for items and term ids as keys. P-Values are FDR adjusted if use_fdr is True (default).

**Parameters**

- **genes** – List of genes
- **reference** – List of genes (if None all genes included in the annotations will be used).
- **evidence_codes** – List of evidence codes to consider.
- **slims_only** – If *True* return only slim terms.
- **aspect** – Which aspects to use. Use all by default; one of Process (biological process), Function (molecular function) or Component (cellular component)
- **prob** –
- **use_fdr** –
- **progress_callback** –

**get_annotated_terms**(*genes*, *direct_annotation_only=False*, *evidence_codes=None*, *progress_callback=None*)
Return all terms that are annotated by genes with evidence_codes.

**add**(*line*)
Add one annotation

**append**(*line*)
Add one annotation

**extend**(*lines*)
Add multiple annotations

**class** orangecontrib.bioinformatics.go.**AnnotationRecord**

An annotation record mapping a gene to a term.

See ftp://ftp.ncbi.nlm.nih.gov/gene/DATA/README for description if individual fields under <gene2go> section.

**classmethod from_string**(*string*)

Create an instance from a line in a annotations file format from serverfiles.

### 2.4.1 Example

Load the ontology and print out some terms:

```
from orangecontrib.bioinformatics import go
ontology = go.Ontology()
term = ontology["GO:0097194"] # execution phase of apoptosis

# print a term
print(term)

# access fields by name
print(term.id, term.name)
# note the use of underscore due to a conflict with a python def keyword
print(term.def_)
```

Searching the annotation (part of `code/go/gene_annotations.py`)

```
from orangecontrib.bioinformatics import go

ontology = go.Ontology()

# Load annotations for yeast.
annotations = go.Annotations("4932", ontology=ontology)

# keys are symbol names, values are Entrez IDs
genes = {'RRB1': '855161', 'OST4': '851366', 'VID27': '855509'}
res = annotations.get_enriched_terms(genes.values())


print(annotations.gene_annotations['855161'])
for a in annotations.gene_annotations['855161']:
    print(ontology[a.go_id].name + " with evidence code " + a.evidence)


# Get all genes annotated to the same terms as RRB1
ids = set([a.go_id for a in annotations.gene_annotations['855161']])
for term_id in ids:
    ants = annotations.get_annotations_by_go_id(term_id)
    genes = set([a.gene_id for a in ants])
    print(", ".join(genes) + " annotated to " + term_id + " " + ontology[term_id].
↪name)
```

Term enrichment (part of `code/go/enrichment.py`)

```
from orangecontrib.bioinformatics import go
```

(continues on next page)

(continued from previous page)

```
ontology = go.Ontology()
annotations = go.Annotations("4932", ontology=ontology)

# keys are symbol names, values are Entrez IDs
genes_ids = {'Yta7p': '853186', 'RPN2': '854735', 'RPT2': '851557'}
res = annotations.get_enriched_terms(genes_ids.values())

print(res)
print("Enriched terms:")
for go_id, (genes, p_value, ref) in res.items():
    if p_value < 0.05:
        print(ontology[go_id].name + " with p-value: %.4f " % p_value + ", ".
↪join(genes))

# And again for slims
annotations.ontology.set_slims_subset('goslim_yeast')

res = annotations.get_enriched_terms(genes_ids.values(), slims_only=True)
print("\n\nEnriched slim terms:")
for go_id, (genes, p_value, _) in res.items():
    if p_value < 0.2:
        print(ontology[go_id].name + " with p-value: %.4f " % p_value + ", ".
↪join(genes))
```

## 2.5 Gene sets (geneset)

This module can load either gene sets distributed with Orange or custom gene sets in the GMT file format.

### 2.5.1 Loading gene sets

`orangecontrib.bioinformatics.geneset.`**`list_all`**(***kwargs*)

Returns available gene sets from the server files repository.

**Parameters** **kwargs** –

- *organism* (`str`) – Taxonomy id (NCBI taxonomy database)

**Return type** `list` of (hierarchy, organism)

#### Example

**The available gene set collection can be listed with:**

```
>>> list_all(organism='10090')
```

`orangecontrib.bioinformatics.geneset.`**`load_gene_sets`**(*hierarchy*, *tax_id*)

Initialize gene sets from a given hierarchy.

**Parameters** **hierarchy** (*tuple*) – gene set hierarchy.

**Return type** *GeneSets*

#### Example

**Gene sets provided with Orange are organized hierarchically:**

```
>>> list_of_genesets= list_all(organism='10090')
    [(('KEGG', 'Pathways'), '10090'),
     (('KEGG', 'pathways'), '10090'),
     (('GO', 'biological_process'), '10090'),
     (('GO', 'molecular_function'), '10090'),
     (('GO', 'cellular_component'), '10090')]
>>> load_gene_sets(list_of_genesets[0])
```

### 2.5.2 Supporting functionality

**class** orangecontrib.bioinformatics.geneset.**GeneSets**(*sets=None*)

Bases: set

A collection of gene sets: contains *GeneSet* objects.

**common_hierarchy**()

Return a common hierarchy.

**common_org**()

Return a common organism.

**static from_gmt_file_format**(*file_path*)

Load GeneSets object from GMT file.

**Parameters** **file_path** – path to a file on local disk

**Return type** *GeneSets*

**genes**()

**Returns** All genes from GeneSets

**hierarchies**()

Return all hierarchies.

**split_by_hierarchy**()

Split gene sets by hierarchies. Return a list of *GeneSets* objects.

**to_gmt_file_format**(*file_path*)

The GMT file format is a tab delimited file format that describes gene sets.

In the GMT format, each row represents a gene set. Columns: gs_id gmt_description Gene Gene Gene ... gmt_description: 'gs_id','hierarchy','organism','name','genes','description','link'

**Parameters** **file_path** – Path to where file will be created

**update**(*sets*)

Update a set with the union of itself and others.

**class** orangecontrib.bioinformatics.geneset.**GeneSet**(*gs_id=None, hierarchy=None, organism=None, name=None, genes=None, description=None, link=None*)

**gmt_description**()

Represent GeneSet as line in GMT file format

**Returns** Comma-separated GeneSet attributes.

**set_enrichment** (*reference*, *query*)

**Parameters**

- **reference** –
- **query** –

### 2.5.3 Helper functions to work with serverfiles

orangecontrib.bioinformatics.geneset.**filename** (*hierarchy*, *organism*)

Obtain a filename for given hierarchy and organism.

**Parameters**

- **hierarchy** – GeneSet hierarchy, example: ('GO', 'biological_process')
- **organism** – Taxonomy ID

**Returns** Filename for given hierarchy and organism

#### Example

```
>>> filename(('CustomSet', 'subsets'), '6500')
'CustomSet-subsets-6500.gmt'
```

orangecontrib.bioinformatics.geneset.**filename_parse** (*fn*)

Returns a hierarchy and the organism from the gene set filename format.

**Parameters** **fn** – GeneSets file name (.gmt)

**Returns** A hierarchy and taxonomy id for given filename

#### Example

```
>>> filename_parse('Custom-set-6500.gmt')
(('Custom', 'set'), '6500')
```

## 2.6 D. discoideum Mutant Phenotypes (dicty.phenotypes)

This modules provides an interface to Dictyostelium mutant phenotypes data from the dictyBase. The mutants are presented as DictyMutant objects with their respective name, strain descriptor, associated genes and associated phenotypes.

```
>>> from orangecontrib.bio.dicty.phenotypes import *
>>> # Create a set of all mutant objects
>>> dicty_mutants = mutants()
>>> # List a set of all genes referenced by a single mutant
>>> print(mutant_genes(dicty_mutants[0]))
  ['acbA']
>>> # List a set of all phenotypes referenced by a single mutant
```

(continues on next page)

(continued from previous page)

```
>>> print(mutant_phenotypes(dicty_mutants[0]))
  ['decreased lipid binding']
```

### 2.6.1 Classes and Functions

Mutant Phenotypes

`orangecontrib.bioinformatics.dicty.phenotypes.`**`mutants`**`()`
Return all `DictyMutant` objects.

`orangecontrib.bioinformatics.dicty.phenotypes.`**`genes`**`()`
Return a set of all genes referenced in the Dictybase.

`orangecontrib.bioinformatics.dicty.phenotypes.`**`phenotypes`**`()`
Return a set of all phenotypes referenced in Dictybase.

`orangecontrib.bioinformatics.dicty.phenotypes.`**`mutant_genes`**`(`*mutant*`)`
Return a set of all genes referenced by a *mutant* in Dictybase.

`orangecontrib.bioinformatics.dicty.phenotypes.`**`mutant_phenotypes`**`(`*mutant*`)`
Return a set of all phenotypes referenced by a *mutant* in Dictybase.

`orangecontrib.bioinformatics.dicty.phenotypes.`**`gene_mutants`**`()`
Return a dictionary { gene: set(mutant_objects for mutant), ... }.

`orangecontrib.bioinformatics.dicty.phenotypes.`**`phenotype_mutants`**`()`
Return a dictionary { phenotype: set(mutant_objects for mutant), ... }.

## 2.7 KEGG - Kyoto Encyclopedia of Genes and Genomes (`kegg`)

### 2.7.1 KEGG - Kyoto Encyclopedia of Genes and Genomes

`kegg` is a python module for accessing KEGG (Kyoto Encyclopedia of Genes and Genomes) using its web services.

**Note:** This module requires slumber and requests packages.

```
>>> # Create a KEGG Genes database interface
>>> genome = KEGGGenome()
>>> # List all available entry ids
>>> keys = list(genome.keys())
>>> print(keys[0])
T01001
>>> # Retrieve the entry for the key.
>>> entry = genome[keys[0]]
>>> print(entry.entry_key)
T01001
>>> print(entry.definition)
Homo sapiens (human)
>>> print(entry)
ENTRY       T01001            Complete  Genome
NAME        hsa, HUMAN, 9606
DEFINITION  Homo sapiens (human)
...
```

The *Organism* class can be a convenient starting point for organism specific databases.

```
>>> organism = Organism("Homo sapiens")  # searches for the organism by name
>>> print(organism.org_code)  # prints the KEGG organism code
hsa
>>> genes = organism.genes  # get the genes database for the organism
>>> gene_ids = list(genes.keys()) # KEGG gene identifiers
>>> entry = genes["hsa:672"]
>>> print(entry.definition)
(RefSeq) BRCA1, DNA repair associated
>>> # print the entry in DBGET database format.
>>> print(entry)
ENTRY       672               CDS       T01001
NAME        BRCA1, BRCAI, BRCC1, BROVCA1, FANCS, IRIS, PNCA4, PPP1R53, PSCP, RNF53
DEFINITION  ...
```

**class** orangecontrib.bioinformatics.kegg.**Organism**(*org*)

A convenience class for retrieving information regarding an organism in the KEGG Genes database.

**Parameters** **org** (*str*) – KEGG organism code (e.g. "hsa", "sce"). Can also be a descriptive name (e.g. 'yeast', "homo sapiens") in which case the organism code will be searched for by using KEGG *find* api.

**See also:**

***organism_name_search()*** Search KEGG for an organism code

**org**

KEGG organism code.

**genes**

An *Genes* database instance for this organism.

**gene_aliases**()

Return a list of sets of equal genes (synonyms) in KEGG for this organism.

---

**Note:** This only includes 'ncbi-geneid' and 'ncbi-proteinid' records from the KEGG Genes DBLINKS entries.

---

**pathways**(*with_ids=None*)

Return a list of all pathways for this organism.

**list_pathways**()

List all pathways for this organism.

**get_enriched_pathways**(*genes*, *reference=None*, *prob=<orangecontrib.bioinformatics.utils.statistics.Binomial object>*, *callback=None*)

Return a dictionary with enriched pathways ids as keys and (list_of_genes, p_value, num_of_reference_genes) tuples as items.

**get_pathways_by_genes**(*gene_ids*)

Pathways that include all genes in gene_ids.

orangecontrib.bioinformatics.kegg.**KEGGOrganism**

alias of *orangecontrib.bioinformatics.kegg.Organism*

orangecontrib.bioinformatics.kegg.**organism_name_search**(*name*)

Search for a organism by *name* and return it's KEGG organism code.

orangecontrib.bioinformatics.kegg.**pathways**(*org*)
Return a list of all KEGG pathways for an KEGG organism code *org*.

orangecontrib.bioinformatics.kegg.**from_taxid**(*taxid*)
Return a KEGG organism code for a an NCBI Taxonomy id string *taxid*.

orangecontrib.bioinformatics.kegg.**to_taxid**(*name*)
Return a NCBI Taxonomy id for a given KEGG Organism name

### 2.7.2 DBEntry (entry)

The *entry.DBEntry* represents a DBGET databas entry. The individual KEGG Database interfaces below provide their own specialization for this base class.

**class** orangecontrib.bioinformatics.kegg.entry.**DBEntry**(*text=None*)
Bases: object

A DBGET entry object.

**entry_key**
Primary entry key used for identifying the entry.

**parse**(*text*)
Parse *text* string containing a formated DBGET entry.

**format**(*section_indent=12*)
Return a DBGET formated string representation.

### 2.7.3 KEGG Databases interface (databases)

**class** orangecontrib.bioinformatics.kegg.databases.**DBDataBase**(***kwargs*)
Bases: object

Base class for a DBGET database interface.

**ENTRY_TYPE**
alias of *orangecontrib.bioinformatics.kegg.entry.DBEntry*

**DB = None**
A database name/abbreviation (e.g. 'pathway'). Needs to be set in a subclass or object instance's constructor before calling the base. __init__

**iterkeys**()
Return an iterator over the *keys*.

**iteritems**()
Return an iterator over the *items*.

**itervalues**()
Return an iterator over all *DBDataBase.ENTRY_TYPE* instances.

**keys**()
Return an iterator over all database keys. These are unique KEGG identifiers that can be used to query the database.

**values**()
Return an iterator over all *DBDataBase.ENTRY_TYPE* instances.

**items**()
Return an iterator over all (key, *DBDataBase.ENTRY_TYPE*) tuples.

**get** (*key*, *default=None*)
Return an *DBDataBase.ENTRY_TYPE* instance for the *key*. Raises `KeyError` if not found.

**get_text** (*key*)
Return the database entry for *key* as plain text.

**get_entry** (*key*)
Return the database entry for *key* as an instance of *ENTRY_TYPE*.

**find** (*name*)
Find *name* using kegg *find* api.

**pre_cache** (*keys=None*, *batch_size=10*, *progress_callback=None*)
Retrieve all the entries for *keys* and cache them locally for faster subsequent retrieval. If *keys* is `None` then all entries will be retrieved.

**batch_get** (*keys*)
Batch retrieve all entries for keys. This can be significantly faster then getting each entry separately especially if entries are not yet cached.

**class** orangecontrib.bioinformatics.kegg.databases.**GenomeEntry** (*text*)
Bases: *orangecontrib.bioinformatics.kegg.entry.DBEntry*

Entry for a KEGG Genome database.

**organism_code**
A three or four letter KEGG organism code (e.g. 'hsa', 'sce', . . . )

**taxid**
Organism NCBI taxonomy id.

**annotation**
ANNOTATION

**chromosome**
CHROMOSOME

**comment**
COMMENT

**data_source**
DATA_SOURCE

**definition**
DEFINITION

**disease**
DISEASE

**entry**
ENTRY

**keywords**
KEYWORDS

**name**
NAME

**original_db**
ORIGINAL_DB

**plasmid**
PLASMID

**reference**
REFERENCE

**statistics**
STATISTICS

**taxonomy**
TAXONOMY

**class** orangecontrib.bioinformatics.kegg.databases.**Genome**
Bases: *orangecontrib.bioinformatics.kegg.databases.DBDataBase*

An interface to the A KEGG GENOME database.

**ENTRY_TYPE**
alias of *GenomeEntry*

**org_code_to_entry_key**(*code*)
Map an organism code ('hsa', 'sce', ...) to the corresponding kegg identifier (T + 5 digit number).

**search**(*string*, *relevance=False*)
Search the genome database for string using `bfind`.

**class** orangecontrib.bioinformatics.kegg.databases.**GeneEntry**(*text=None*)
Bases: *orangecontrib.bioinformatics.kegg.entry.DBEntry*

**aaseq**
AASEQ

**brite**
BRITE

**class_**
CLASS

**dblinks**
DBLINKS

**definition**
DEFINITION

**disease**
DISEASE

**drug_target**
DRUG_TARGET

**entry**
ENTRY

**module**
MODULE

**motif**
MOTIF

**name**
NAME

**ntseq**
NTSEQ

**organism**
ORGANISM

**orthology**
ORTHOLOGY

**pathway**
PATHWAY

**position**
POSITION

**structure**
STRUCTURE

**class** orangecontrib.bioinformatics.kegg.databases.**Genes**(*org_code*)
Bases: *orangecontrib.bioinformatics.kegg.databases.DBDataBase*

Interface to the KEGG Genes database.

**Parameters** **org_code** (*str*) – KEGG organism code (e.g. 'hsa').

**ENTRY_TYPE**
alias of *GeneEntry*

**class** orangecontrib.bioinformatics.kegg.databases.**CompoundEntry**(*text=None*)
Bases: *orangecontrib.bioinformatics.kegg.entry.DBEntry*

**atom**
ATOM

**bond**
BOND

**brite**
BRITE

**comment**
COMMENT

**dblinks**
DBLINKS

**entry**
ENTRY

**enzyme**
ENZYME

**exact_mass**
EXACT_MASS

**formula**
FORMULA

**mol_weight**
MOL_WEIGHT

**name**
NAME

**pathway**
PATHWAY

**reaction**
REACTION

**reference**
REFERENCE

**remark**
REMARK

**class** orangecontrib.bioinformatics.kegg.databases.**Compound**
Bases: *orangecontrib.bioinformatics.kegg.databases.DBDataBase*

**ENTRY_TYPE**
alias of *CompoundEntry*

**class** orangecontrib.bioinformatics.kegg.databases.**ReactionEntry**(*text=None*)
Bases: *orangecontrib.bioinformatics.kegg.entry.DBEntry*

**definition**
DEFINITION

**entry**
ENTRY

**enzyme**
ENZYME

**equation**
EQUATION

**name**
NAME

**class** orangecontrib.bioinformatics.kegg.databases.**Reaction**
Bases: *orangecontrib.bioinformatics.kegg.databases.DBDataBase*

**ENTRY_TYPE**
alias of *ReactionEntry*

**class** orangecontrib.bioinformatics.kegg.databases.**EnzymeEntry**(*text=None*)
Bases: *orangecontrib.bioinformatics.kegg.entry.DBEntry*

**all_reac**
ALL_REAC

**class_**
CLASS

**comment**
COMMENT

**dblinks**
DBLINKS

**entry**
ENTRY

**genes**
GENES

**name**
NAME

**orthology**
ORTHOLOGY

**pathway**
PATHWAY

**product**
PRODUCT

**reaction**
REACTION

**reference**
REFERENCE

**substrate**
SUBSTRATE

**sysname**
SYSNAME

**class** orangecontrib.bioinformatics.kegg.databases.**Enzyme**
Bases: *orangecontrib.bioinformatics.kegg.databases.DBDataBase*

**ENTRY_TYPE**
alias of *EnzymeEntry*

**class** orangecontrib.bioinformatics.kegg.databases.**PathwayEntry**(*text=None*)
Bases: *orangecontrib.bioinformatics.kegg.entry.DBEntry*

**class_**
CLASS

**compound**
COMPOUND

**dblinks**
DBLINKS

**description**
DESCRIPTION

**disease**
DISEASE

**drug**
DRUG

**entry**
ENTRY

**enzyme**
ENZYME

**ko_pathway**
KO_PATHWAY

**module**
MODULE

**name**
NAME

**organism**
ORGANISM

**pathway_map**
PATHWAY_MAP

**reference**
REFERENCE

**rel_pathway**
REL_PATHWAY

**class** orangecontrib.bioinformatics.kegg.databases.**Pathway**(*prefix='map'*)
Bases: *orangecontrib.bioinformatics.kegg.databases.DBDataBase*

KEGG Pathway database

**Parameters** **prefix** (*str*) – KEGG Organism code ('hsa', ...) or 'map', 'ko', 'ec' or 'rn'

**ENTRY_TYPE**
alias of *PathwayEntry*

### 2.7.4 KEGG Pathway (pathway)

**class** orangecontrib.bioinformatics.kegg.pathway.**Pathway**(*pathway_id, local_cache=None, connection=None*)
Bases: object

Class representing a KEGG Pathway (parsed from a "kgml" file)

**Parameters** **pathway_id** (*str*) – A KEGG pathway id (e.g. 'path:hsa05130')

**name**
*Pathway name/id (e.g. "path* – hsa05130")

**org**
Pathway organism code (e.g. 'hsa')

**number**
Pathway number as a string (e.g. '05130')

**title**
Pathway title string.

**image**
URL of the pathway image.

**link**
URL to a pathway on the KEGG web site.

**get_image**()
Return an local filesystem path to an image of the pathway. The image will be downloaded if not already cached.

**classmethod list**(*organism*)
List all pathways for KEGG organism code *organism*.

### 2.7.5 Utilities

**class** orangecontrib.bioinformatics.kegg.entry.parser.**DBGETEntryParser**
A DBGET entry parser (inspired by xml.dom.pulldom).

### Example

```
>>> stream = StringIO(
...     "ENTRY foo\n"
...     "NAME  foo's name\n"
...     "  BAR A subsection of 'NAME'\n"
... )
>>> parser = DBGETEntryParser()
>>> for event, title, contents_part in parser.parse(stream):
...    print(parser.EVENTS[event], title, repr(contents_part))
...
ENTRY_START None None
SECTION_START ENTRY 'foo\n'
SECTION_END ENTRY None
SECTION_START NAME "foo's name\n"
SUBSECTION_START BAR "A subsection of 'NAME'\n"
SUBSECTION_END BAR None
SECTION_END NAME None
ENTRY_END None None
```

**ENTRY_END = 1**
Entry end event

**ENTRY_START = 0**
Entry start events

**SECTION_END = 3**
Section end event

**SECTION_START = 2**
Section start event

**SUBSECTION_END = 5**
Subsection end event

**SUBSECTION_START = 4**
Subsection start event

**TEXT = 6**
Text element event

## 2.8 Resolwe module (resolwe)

Resolwe module

orangecontrib.bioinformatics.resolwe.**connect**(*username*, *password*, *url*, *server_type*)
Connect to Resolwe server

**Parameters**

- **username** (*str*) –
- **password** (*str*) –
- **url** (*str*) –
- **server_type** (*str*) – genesis or resolwe

**Returns** Instance of GenAPI or ResolweAPI

**class** orangecontrib.bioinformatics.resolwe.**GenAPI**

Python module that leverages Genesis PyAPI (Python API for accsess to DictyExpress database).

It supports connection to the server and data retrieval functionalities.

**download_etc_data**(*gen_data_id*, ***kwargs*)

Function downloads etc data of a chosen experiment from the server.

**Parameters** **gen_data_id** (*str*) – id of GeneData object

**Return type** data in json like format

**fetch_etc_objects**(***kwargs*)

Function downloads all available GenData etc objects from DictyExpress database.

**Return type** list of GenData objects

# Python Module Index

## O

# Index

## Symbols



## A



## B



## C

## D



## E

## F



## G

## H



## I



## K



## L



## M



## N



## O

## P



## R



## S

## T



## U



## V