
Resolve SDK for Python

Release 20.0.1.dev12+gc2f0da0

Genialis, Inc

Dec 01, 2023

CONTENTS

1	Install	3
2	Usage example	5
3	Documentation	7
3.1	Getting started	7
3.2	Tutorials	10
3.3	Topical documentation	20
3.4	SDK Reference	28
3.5	Contributing	69
	Python Module Index	73
	Index	75

Resolwe SDK for Python supports interaction with [Genialis Server](#). Genialis Server is based on [Resolwe](#) workflow engine and its plugin [Resolwe Bioinformatics](#). You can use it to upload and inspect biomedical data sets, contribute annotations and run analysis.

INSTALL

Install from PyPI:

```
pip install resdk
```

If you would like to contribute to the SDK code base, follow the *installation steps for developers*.

USAGE EXAMPLE

We will download a sample containing raw sequencing reads that were aligned to a genome:

```
import resdk

# Create a Resolwe object to interact with the server
res = resdk.Resolwe(url='https://app.genialis.com')

# Enable verbose logging to standard output
resdk.start_logging()

# Get sample meta-data from the server
sample = res.sample.get('resdk-example')

# Download files associated with the sample
sample.download()
```

Multiple files (fastq, fastQC report, bam, bai...) have downloaded to the working directory. Check them out. To learn more about the Resolwe SDK continue with *Tutorials*.

If you have problems connecting to our server, please contact us at info@genialis.com.

3.1 Getting started

This tutorial is for bioinformaticians. It will help you install the ReSDK and explain some basic commands. We will connect to an instance of [Genialis server](#), do some basic queries, and align raw reads to a genome.

3.1.1 Installation

Installing is easy, just make sure you have [Python](#) and [pip](#) installed on your computer. Run this command in the terminal (CMD on Windows):

```
pip install resdk
```

Note: If you are using Apple silicon you should use Python version 3.10 or higher.

3.1.2 Registration

The examples presented here require access to a public [Genialis Server](#) that is configured for the examples in this tutorial. Some parts of the documentation will work for registered users only. Please [request a Demo](#) on Genialis Server before you continue, and remember your username and password.

3.1.3 Connect to Genialis Server

Start the Python interpreter by typing `python` into the command line. You'll recognize the interpreter by `>>>`. Now we can connect to the Genialis Server:

```
import resdk

# Create a Resolwe object to interact with the server and login
res = resdk.Resolwe(url='https://app.genialis.com')
res.login()

# Enable verbose logging to standard output
resdk.start_logging()
```

Note: If you omit the `login()` line you will be logged as anonymous user. Note that anonymous users do not have access to the full set of features.

The `login()` call will perform interactive login in a web browser. If you wish to log in as a different user, open the link in an incognito window.

Note: When connecting to the server through an interactive session, we suggest you use the `resdk.start_logging()` command. This allows you to see important messages (e.g. warnings and errors) when executing commands.

Note: To avoid copy-pasting of the commands, you can [download all the code](#) used in this section.

3.1.4 Query data

Before we start querying data on the server we should become familiar with what a data object is. Everything that is uploaded or created (via processes) on a server is a data object. The data object contains a complete record of the processing that has occurred. It stores the inputs (files, arguments, parameters...), the process (the algorithm) and the outputs (files, images, numbers...). Let's count all data objects on the server that we can access:

```
res.data.count()
```

This is all of the data on the server you have permissions for. As a new user you can only see a small subset of all data objects. We can see the data objects are referenced by *id*, *slug*, and *name*.

Note: *id* is the auto-generated **unique identifier** of an object. IDs are integers.

slug is the **unique name** of an object. The slug is automatically created from the name but can also be edited by the user. Only lowercase letters, numbers and dashes are allowed (will not accept white space or uppercase letters).

name is an arbitrary, **non unique name** of an object.

Let's say we now want to find some genome indices. We don't always know the *id*, *slug*, or *name* by heart, but we can use [filters](#) to find them. We will first count all genome index data objects:

```
res.data.filter(type='data:index').count()
```

This is quite a lot of objects! We can filter even further:

```
res.data.filter(type="data:index:star", name__contains="Homo sapiens")
```

Note: For a complete list of filtering options use a "wrong" filtering argument and you will receive an informative message with all options listed. For example:

```
res.data.filter(foo="bar")
```

For future work we want to get the genome with a specific slug. We will [get](#) it and store a reference to it for later:

```
# Get data object by slug
genome_index = res.data.get('resdk-example-genome-index')
```

We have now seen how to use filters to find and get what we want. Let's query and get a paired-end FASTQ data object:

```
# All paired-end fastq objects
res.data.filter(type='data:reads:fastq:paired')

# Get specific object by slug
reads = res.data.get('resdk-example-reads')
```

We now have `genome` and `reads` data objects. We can learn about an object by calling certain object attributes. We can find out who created the object:

```
reads.contributor
```

and inspect the list of files it contains:

```
reads.files()
```

These and many other data object attributes/methods are described [here](#).

3.1.5 Run alignment

A common analysis in bioinformatics is to align sequencing reads to a reference genome. This is done by running a certain *process*. A process uses an algorithm or a sequence of algorithms to turn given inputs into outputs. Here we will only test the STAR alignment process, but many more processes are available (see the [Process catalog](#)). This process automatically creates a BAM alignment file and BAI index, along with some other files.

Let's run STAR on our reads, using our genome:

```
bam = res.run(
    slug='alignment-star',
    input={
        'reads': reads.id,
        'genome': genome_index.id,
    },
)
```

This might seem like a complicated statement, but note that we only run a process with specific slug and required inputs. The processing may take some time. Note that we have stored the reference to the alignment object in a `bam` variable. We can check the `status` of the process to determine if the processing has finished:

```
bam.status
```

Status OK indicates that processing has finished successfully. If the status is not OK yet, run the `bam.update()` and `bam.status` commands again in few minutes. We can inspect our newly created data object:

```
# Get the latest info about the object from the server
bam.update()
bam.status
```

As with any other data object, it has its own *id*, *slug*, and *name*. We can explore the process inputs and outputs:

```
# Process inputs
bam.input

# Process outputs
bam.output
```

Download the outputs to your local disk:

```
bam.download()
```

We have come to the end of Getting started. You now know some basic ReSDK concepts and commands. Yet, we have only scratched the surface. By continuing with the Tutorials, you will become familiar with more advanced features, and will soon be able to perform powerful analyses on your data.

3.2 Tutorials

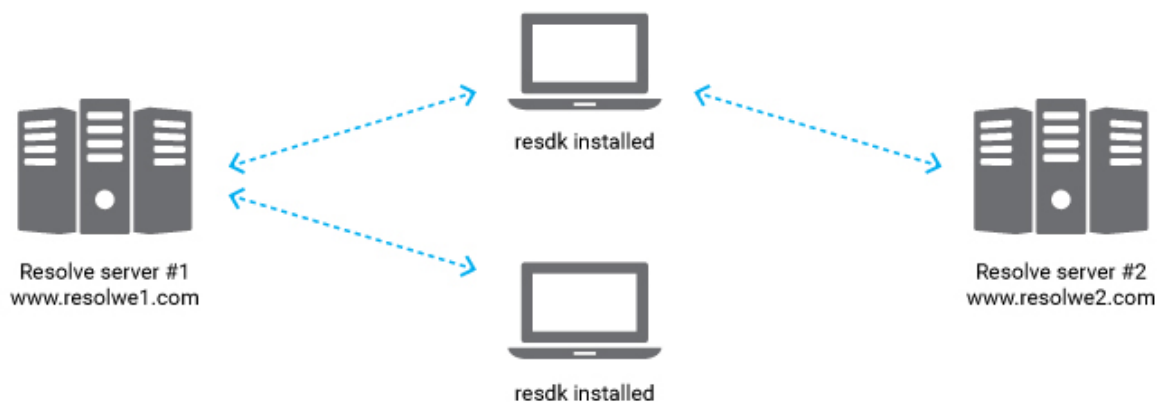
3.2.1 Genialis Server basics

This chapter provides a general overview and explains the basic concepts. We highly recommend reading it even though it is a bit theoretic.

Genialis Server and ReSDK

Genialis Server is a web application that can handle large quantities of biological data, perform complex data analysis, organize results, and automatically document your work in a reproducible fashion. It is based on *Resolwe* and *Resolwe Bioinformatics*. *Resolwe* is an open source dataflow package for the *Django framework* while *Resolwe Bioinformatics* is an extension of *Resolwe* that provides bioinformatics pipelines.

Resolwe SDK for Python allows you to access Genialis Server through Python. It supports accessing, modifying, uploading, downloading and organizing the data.



Genialis Server runs on computers with strong computational capabilities. On the contrary, *resdk* is a Python package on a local computer that interacts with Genialis Server through a RESTful API. The power of *resdk* is its lightweight character. It is installed with one simple command, but supports manipulation of large data sets and heavy computation on a remote server.

Data and Process

The two most fundamental resources in Genialis Server are *Data* and *Process*.

Process stores an algorithm that transforms inputs into outputs. It is a blueprint for one step in the analysis.

Data is an instance of a Process. It is a complete record of the performed processing. It remembers the inputs (files, arguments, parameters...), the algorithm used and the outputs (files, images, numbers...). In addition, Data objects store some useful meta data, making it easy to reproduce the dataflow and access information.

Example use case: you have a file `reads.fastq` with NGS read sequences and want to map them to the genome `genome.fasta` with aligner STAR. Reads are one Data object and genome is another one. Alignment is done by creating a third Data. At the creation, one always needs to define the Process (STAR) and inputs (first and second Data). When the Data object is created, the server automatically runs the given process with provided inputs and computes all inputs, outputs, and meta data.

Samples and Collections

Eventually, you will have many Data objects and want to organize them. Genialis server includes different structures to help you group Data objects: *Sample* and *Collection*.

Sample represents a biological entity. It includes user annotations and Data objects associated with this biological entity. In practice, all Data objects in the Sample are derived from an initial single Data object. Typically, a Sample would contain the following Data: raw reads, preprocessed reads, alignment (bam file), and expressions. A Data object can belong to only one Sample. Two distinct Samples cannot contain the same Data object.

Collection is a group of Samples. In addition to Samples and their Data, Collections may contain Data objects that store other analysis results. Example of this are differential expressions - they are done as combination of many Samples and cannot belong to only one Sample. Each Sample and Data object can only be in one Collection.

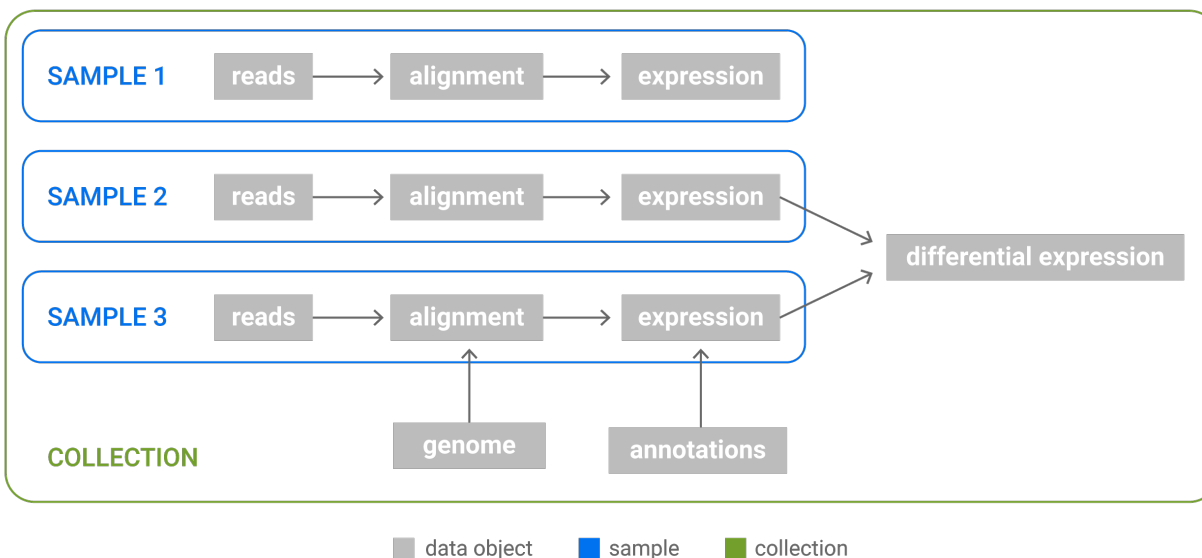


Fig. 1: Relations between Data, Samples and Collection. Samples are groups of Data objects originating from the same biological sample: all Data objects in a Sample are derived from a single NGS reads file. Collections are arbitrary groups of Samples and Data objects that store analysis results.

When a new Data object that represents a biological sample (*i.e.* fastq files, bam files) is uploaded, the unannotated Sample is automatically created. It is the duty of the researcher to properly annotate the Sample. When a Data object

that belongs to an existing Sample is used as an input to trigger a new analysis, the output of this process is automatically attached to an existing Sample.

3.2.2 Query, inspect and download data

Login

By now, you should have an account on the [Genialis Server](#). If not, you can [request a Demo](#). Let's connect to the server by creating a *Resolwe* object:

```
import resdk

# Create a Resolwe object to interact with the server and login
res = resdk.Resolwe(url='https://app.genialis.com')
res.login()

# Enable verbose logging to standard output
resdk.start_logging()
```

If you omit the `login()` you will be logged as anonymous user. Note that this will strongly limit the things you can do.

Note: To avoid copy-pasting of the commands, you can [download all the code](#) used in this section.

Query resources

As you have read in the *Genialis Server basics* section, there are various resources: *Data*, *Sample*, *Collection*, *Process*... each of which has a corresponding entry-point on *Resolwe* object (in our case, this is the `res` variable). For example, to count all *Data* or *Sample* objects:

```
res.data.count()
res.sample.count()
```

Note:

id is the autogenerated unique identifier of an object. IDs are integers.

slug is the **unique name** of an object. The slug is automatically created from the name but can also be edited by the user, although we do not recommend that. Only lowercase letters, numbers and dashes are allowed (will not accept white space or uppercase letters).

name is an arbitrary, **non unique name** of an object.

In practice one typically wants to narrow down the amount of results. This can be done with the `filter(**fields)` method. It returns a list of objects under the conditions defined by `**fields`. For example:

```
# Get all Collection objects with "RNA-Seq" in their name
res.collection.filter(name__contains='RNA-Seq')
```

(continues on next page)

(continued from previous page)

```
# Get all Processes with category "Align"
res.process.filter(category='Align')
```

Note: For a complete list of processes, their categories and definitions, please visit [resolwe-bio docs](#)

But the real power of the `filter()` method is in combining multiple parameters:

```
# Filter by using several fields:
from datetime import datetime

res.data.filter(
    status='OK',
    created__gt=datetime(2018, 10, 1),
    created__lt=datetime(2025, 11, 1),
    ordering='-modified',
    limit=3,
)
```

This will return data objects with OK status, created in October 2018, order them by descending modified date and return first 3 objects. Quite powerful isn't it?

Note: For a complete list of filtering options use a “wrong” filtering argument and you will receive an informative message with all options listed. For example:

```
res.data.filter(foo="bar")
```

The `get(**fields)` method searches by the same parameters as `filter` and returns a single object (`filter` returns a list). If only one parameter is given, it will be interpreted as a unique identifier `id` or `slug`, depending on if it is a number or string:

```
# Get object by slug
res.sample.get('resdk-example')
```

Inspect resources

We have learned how to query the resources with `get` and `filter`. Now we will look at how to access the information in these resources. All of the resources share some common attributes like `name`, `id`, `slug`, `created`, `modified`, `contributor` and `permissions`. You can access them like any other Python class attributes:

```
# Get a data object:
data = res.data.get('resdk-example-reads')

# Object creator:
data.contributor

# Date and time of object creation:
data.created

# Name
data.name
```

(continues on next page)

(continued from previous page)

```
# List of permissions
data.permissions
```

Aside from these attributes, each resource class has some specific attributes and methods. For example, some of the most used ones for Data resource:

```
data = res.data.get('resdk-example-reads')
data.status
data.process
data.started
data.finished
data.size
```

You can check list of methods defined for each of the resources in the *reference section*. Note that some attributes and methods are defined in the *BaseResource* and *BaseCollection* classes. *BaseResource* is the parent of all resource classes in resdk. *BaseCollection* is the parent of all collection-like classes: *Sample* and *Collection*

Quite commonly, one wants to inspect list of Data objects in Collection or to know the Sample of a given Data... For such purposes, there are some handy shortcuts:

- *data.sample*
- *data.collection*
- *sample.data*
- *sample.collection*
- *collection.data*
- *collection.samples*

Download data

Resource classes Data, Sample and Collection have the methods `files()` and `download()`.

The `files()` method returns a list of all files on the resource but does not download anything.

```
# Get data by slug
data = res.data.get('resdk-example-reads')

# Print a list of files
data.files()

# Filter the list of files by file name
data.files(file_name='reads.fastq.gz')

# Filter the list of files by field name
data.files(field_name='output.fastq')
```

The method `download()` downloads the resource files. The optional parameters `file_name` and `field_name` have the same effect as in the `files` method. There is an additional parameter, `download_dir`, that allows you to specify the download directory:

```
# Get sample by slug
sample = res.sample.get('resdk-example')

# Download the FASTQ reads file into current directory
sample.download(
    file_name='reads.fastq.gz',
    download_dir='./',
)
```

3.2.3 Create, modify and organize data

To begin, we need some sample data to work with. You may use your own reads (.fastq) files, or download an example set we have provided:

```
import resdk

res = resdk.Resolwe(url='https://app.genialis.com')
res.login()

# Get example reads
example = res.data.get('resdk-example-reads')
# Download them to current working directory
example.download(
    field_name='fastq',
    download_dir='./',
)
```

Note: To avoid copy-pasting of the commands, you can download all the code used in this section.

Organize resources

Before all else, one needs to prepare space for work. In our case, this means creating a “container” where the produced data will reside. So let’s create a collection and than put some data inside!

```
# create a new collection object in your running instance of Resolwe (res)
test_collection = res.collection.create(name='Test collection')
```

Upload files

We will upload fastq single end reads with the `upload-fastq-single` process.

```
# Upload FASTQ reads
reads = res.run(
    slug='upload-fastq-single',
    input={
        'src': './reads.fastq.gz',
    },
)
```

(continues on next page)

(continued from previous page)

```
collection=test_collection,  
)
```

What just happened? First, we chose a process to run, using its slug `upload-fastq-single`. Each process requires some inputs—in this case there is only one input with name `src`, which is the location of reads on our computer. Uploading a fastq file creates a new `Data` on the server containing uploaded reads. Additionally, we ensured that the new `Data` is put inside `test_collection`.

The upload process also created a `Sample` object for the reads data to be associated with. You can access it by:

```
reads.sample
```

Note: You can also upload your files by providing url. Just replace path to your local files with the url. This comes handy when your files are large and/or are stored on a remote server and you don't want to download them to your computer just to upload them to Resolwe server again...

Modify data

Both `Data` with reads and `Sample` are owned by you and you have permissions to modify them. For example:

```
# Change name  
reads.name = 'My first data'  
reads.save()
```

Note the `save()` part! Without this, the change is only applied locally (on your computer). But calling `save()` also takes care that all changes are applied on the server.

Note: Some fields cannot (and should not) be changed. For example, you cannot modify `created` or `contributor` fields. You will get an error if you try.

Annotate Samples

The next thing to do after uploading some data is to annotate samples this data belongs to. This can be done by assigning a value to a predefined field on a given sample. See the example below.

Each sample should be assigned a species. This is done by attaching the `general.species` field on a sample and assigning it a value, e.g. `Homo sapiens`.

```
reads.sample.set_annotation("general.species", "Homo sapiens")
```

Annotation Fields

You might be wondering why the example above requires `general.species` string instead of e.g. just `species`. The answer to this are `AnnotationFields`. These are predefined *objects* that are available to annotate samples. These objects primarily have a name, but also other properties. Let's examine some of those:

```
# Get the field by it's group and name:
field = res.annotation_field.get(group__name="general", name="species")
# Same thing, but in shorter syntax
field = res.annotation_field.from_path("general.species")
# Examine some of the field attributes
field.name
field.group
field.description
```

Note: Each field is uniquely defined by the combination of name and group.

If you wish to examine what fields are available, use a query

```
res.annotation_field.all()
# You can also filter the results
res.annotation_field.filter(group__name="general")
```

You may be wondering whether you can create your own fields / groups. The answer is no. Time has proven that keeping things organized requires the usage of a selected set of predefined fields. If you absolutely feel that you need an additional annotation field, let us know or use resources such as *Metadata*.

Annotation Values

As mentioned before, fields are only one part of the annotation. The other part are annotation values, stored as a standalone resource `AnnotationValues`. They connect the field with the actual value.

```
# Get an AnnotationValue
ann_value = reads.sample.get_annotation("general.species")
# The actual value
ann_value.value
# The corresponding field
ann_value.field
# The corresponding sample
ann_value.sample
```

As a shortcut, you can get all the `AnnotationValues` for a given sample by:

```
reads.sample.annotations
```

Helper methods

Sometimes it is convenient to represent the annotations with the dictionary, where keys are field names and values are annotation values. You can get all the annotation for a given sample in this format by calling:

```
reads.sample.get_annotations()
```

Multiple annotations stored in the dictionary can be assigned to sample by:

```
annotations = {
    "general.species": "Homo sapiens", "general.description": "Description"
}
reads.sample.set_annotations(annotations)
```

Annotation is deleted from the sample by setting its value to `None` when calling `set_annotation` or `set_annotations` helper methods. To avoid confirmation prompt, you can set `force=True`.

```
reads.sample.set_annotation("general.description", None, force=True)
```

Run analyses

Various bioinformatic processes are available to properly analyze sequencing data. Many of these pipelines are available via Resolwe SDK, and are listed in the [Process catalog](#) of the [Resolwe Bioinformatics documentation](#).

After uploading reads file, the next step is to align reads to a genome. We will use STAR aligner, which is wrapped in a process with slug `alignment-star`. Inputs and outputs of this process are described in [STAR process catalog](#). We will define input files and the process will run its algorithm that transforms inputs into outputs.

```
# Get genome
genome_index = res.data.get('resdk-example-genome-index')

alignment = res.run(
    slug='alignment-star',
    input={
        'genome': genome_index,
        'reads': reads,
    },
)
```

Lets take a closer look to the code above. We defined the alignment process, by its slug `'alignment-star'`. For inputs we defined data objects `reads` and `genome`. `reads` object was created with `'upload-fastq-single'` process, while `genome` data object was already on the server and we just used its slug to identify it. The `alignment-star` processor will automatically take the right files from data objects, specified in inputs and create output files: `bam` alignment file, `bai` index and some more...

You probably noticed that we get the result almost instantly, while the typical assembling process runs for hours. This is because processing runs asynchronously, so the returned data object does not have an OK status or outputs when returned.

```
# Get the latest meta data from the server
alignment.update()

# See the process progress
alignment.process_progress
```

(continues on next page)

(continued from previous page)

```
# Print the status of data
alignment.status
```

Status OK indicates that processing has finished successfully, but you will also find other statuses. They are given with two-letter abbreviations. To understand their meanings, check the [status reference](#). When processing is done, all outputs are written to disk and you can inspect them:

```
# See process output
alignment.output
```

Until now, we used `run()` method twice: to upload reads (yes, uploading files is just a matter of using an upload process) and to run alignment. You can check the full signature of the `run()` method.

Run workflows

Typical data analysis is often a sequence of processes. Raw data or initial input is analysed by running a process on it that outputs some data. This data is fed as input into another process that produces another set of outputs. This output is then again fed into another process and so on. Sometimes, this sequence is so commonly used that one wants to simplify its execution. This can be done by using so called “workflow”. Workflows are special processes that run a stack of processes. On the outside, they look exactly the same as a normal process and have a process slug, inputs... For example, we can run workflow “General RNA-seq pipeline” on our reads:

```
# Run a workflow
res.run(
    slug='workflow-bbduk-star-featurecounts-qc',
    input={
        'reads': reads,
        'genome': res.data.get('resdk-example-genome-index'),
        'annotation': res.data.get('resdk-example-annotation'),
        'rrna_reference': res.data.get('resdk-example-rrna-index'),
        'globin_reference': res.data.get('resdk-example-globin-index'),
    }
)
```

Solving problems

Sometimes the data object will not have an “OK” status. In such case, it is helpful to be able to check what went wrong (and where). The `stdout()` method on data objects can help—it returns the standard output of the data object (as string). The output is long but exceedingly useful for debugging. Also, you can inspect the info, warning and error logs.

```
# Update the data object to get the most recent info
alignment.update()

# Print process' standard output
print(alignment.stdout())

# Access process' execution information
alignment.process_info
```

(continues on next page)

(continued from previous page)

```
# Access process' execution warnings
alignment.process_warning

# Access process' execution errors
alignment.process_error
```

3.3 Topical documentation

Here you can browse through topical documentation about various parts of ReSDK.

3.3.1 Knowledge base

Genialis Knowledge base (KB) is a collection of “features” (genes, transcripts, ...) and “mappings” between these features. It comes very handy when performing various tasks with genomic features e.g.:

- find all aliases of gene BRCA2
- finding all genes of type `protein_coding`
- find all transcripts of gene FHTT
- converting `gene_id` to `gene_symbol`
- ...

Feature

Feature object represents a genomic feature: a gene, a transcript, etc. You can query Feature objects by feature endpoint, similarly like `Data`, `Sample` or any other ReSDK resource:

```
feature = res.feature.get(feature_id="BRCA2")
```

To examine all attributes of a Feature, see the *SDK Reference*. Here we will list a few most commonly used ones:

```
# Get the feature:
feature = res.feature.get(feature_id="BRCA2")

# Database where this feature is defined, e.g. ENSEMBL, UCSC, NCBI, ...
feature.source

# Unique identifier of a feature
feature.feature_id

# Feature species
feature.species

# Feature type, e.g. gene, transcript, exon, ...
feature.type

# Feature name
feature.name
```

(continues on next page)

(continued from previous page)

```
# List of feature aliases
feature.aliases
```

The real power is in the filter capabilities. Here are some examples:

```
# Count number of Human "protein-coding" transcripts in ENSEMBL database
res.feature.filter(
    species="Homo sapiens",
    type="transcript",
    subtype="protein-coding",
    source="ENSEMBL",
).count()

# Convert all gene IDs in a list `gene_ids` to gene symbols::
gene_ids = ["ENSG00000139618", "ENSG00000189283"]
genes = res.feature.filter(
    feature_id__in=gene_ids,
    type="gene",
    species="Homo sapiens",
)
mapping = {g.feature_id: g.name for g in genes}
gene_symbols = [mapping[gene_id] for gene_id in gene_ids]
```

Warning: It might look tempting to simplify the last example with:

```
gene_symbols = [g.name for g in genes]
```

Don't do this. The order of entries in the `genes` can be arbitrary and therefore cause that the resulting list `gene_symbols` is not ordered in the same way as `gene_ids`.

Mapping

Mapping is a *connection* between two features. Features can be related in various ways. The type of mapping is indicated by `relation_type` attribute. It is one of the following options:

- `crossdb`: Two features from different sources (databases) that describe same feature. Example: connection for gene BRCA2 between database "UniProtKB" and "UCSC".
- `ortholog`: Two features from different species that describe orthologous gene.
- `transcript`: Connection between gene and it's transcripts.
- `exon`: Connection between gene / transcript and it's exons.

Again, we will only list an example and then let your imagination fly:

```
# Find UniProtKB ID for gene with given ENSEMBL ID:
mapping = res.mapping.filter(
    source_id="ENSG00000189283",
    source_db="ENSEMBL",
    target_db="UniProtKB",
    source_species="Homo sapiens",
```

(continues on next page)

(continued from previous page)

```

    target_species="Homo sapiens",
)
uniprot_id = mapping[0].target_id

```

3.3.2 ReSDK Tables

ReSDK tables are helper classes for aggregating collection data in tabular format. Currently, we have four flavours:

- *RNATables*
- *MethylationTables*
- *MATables*
- *VariantTables*

RNATables

Imagine you are modelling gene expression data from a given collection. Ideally, you would want all expression values organized in a table where rows represents samples and columns represent genes. Class *RNATables* gives you just that (and more).

We will present the functionality of *RNATables* through an example. We will:

- Create an instance of *RNATables* and examine it's attributes
- Fetch raw expressions and select *TIS signature genes* with sufficient coverage
- Normalize expression values (log-transform) and visualize samples in a simple PCA plot

First, connect to a Resolwe server, pick a collection and create an instance of *RNATables*:

```

import resdk
from resdk.tables import RNATables
res = resdk.Resolwe(url='https://app.genialis.com/')
res.login()
collection = res.collection.get("sum149-fresh-for-rename")
sum149 = RNATables(collection)

```

Object *sum149* is an instance of *RNATables* and has many attributes. For a complete list see the *SDK Reference*, here we list the most commonly used ones:

```

# Expressions raw counts
sum149.rc

# Expressions normalized counts
sum149.exp
# See normalization method
sum149.exp.attrs["exp_type"]

# Sample metadata
sum149.meta

# Sample QC metrics
sum149.qc

```

(continues on next page)

(continued from previous page)

```
# Dictionary that maps gene ID's into gene symbols
sum149.readable_columns
# This is handy to rename column names (gene ID's) to gene symbols
sum149.rc.rename(columns=sum149.readable_columns)
```

Note: Expressions and metadata are cached in memory as well as on disk. At each time they are re-requested a check is made that local and server side of data is synced. If so, cached data is used. Otherwise, new data will be pulled from server.

In our example we will only work with a set of TIS signature genes:

```
TIS_GENES = ["CD3D", "ID01", "CIITA", "CD3E", "CCL5", "GZMK", "CD2", "HLA-DRA", "CXCL13",
↪ "IL2RG", "NKG7", "HLA-E", "CXCR6", "LAG3", "TAGAP", "CXCL10", "STAT1", "GZMB"]
```

We will identify low expressed genes and only keep the ones with average raw expression above 20:

```
tis_rc = sum149.rc.rename(columns=sum149.readable_columns)[TIS_GENES]
mean = tis_rc.mean(axis=0)
high_expressed_genes = mean.loc[mean > 20].index
```

Now, lets select TPM normalized expressions and keep only highly expressed tis genes. We also transform to $\log_2(\text{TPM} + 1)$:

```
import numpy as np
tis_tpm = sum149.exp.rename(columns=sum149.readable_columns)[high_expressed_genes]
tis_tpm_log = np.log(tis_tpm + 1)
```

Finally, we perform PCA and visualize the results:

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2, whiten=True)
Y = pca.fit_transform(tis_tpm_log)

import matplotlib.pyplot as plt
for ((x, y), sample_name) in zip(Y, tis_tpm.index):
    plt.plot(x, y, 'bo')
    plt.text(x, y, sample_name)
plt.xlabel(f"PC1 ({{pca.explained_variance_ratio_[0]}})")
plt.ylabel(f"PC2 ({{pca.explained_variance_ratio_[1]}})")
plt.show()
```

MethylationTables

Similar as RNATables provide access to raw counts and normalized expression values of RNA data, MethylationTables allow for fast access of beta and m-values of methylation data:

```
meth = resdk.tables.MethylationTables(<collection-with-methylation-data>)

# Methylation beta-values
meth.beta

# Methylation m-values
meth.mval
```

MATables

Similar as RNATables provide access to raw counts and normalized expression values of RNA data, MATables allow for fast access of expression values per probe of microarray:

```
ma = resdk.tables.MATables(<collection-with-microarray-data>)

# Microarray expressions values (columns are probe ID's)
ma.exp
```

VariantTables

Similar as RNATables provide access to raw counts and normalized expression values of RNA data, VariantTables allow for fast access of variant data present in Data of type data:mutationstable:

```
vt = resdk.tables.VariantTables(<collection-with-variant-data>)
vt.variants
```

The output of the above would look something like this:

sample_id	chr1_123_C>T	chr1_126_T>C
101	2	NaN
102	0	2

In rows, there are sample ID's. In columns there are variants where each variant is given as: <chromosome>_<position>_<nucleotide-change>. Values in table can be:

- 0 (wild-type / no mutation)
- 1 (heterozygous mutation),
- 2 (homozygous mutation)
- NaN (QC filters are failing - mutation status is unreliable)

Inspecting depth

The reason for NaN values may be that the read depth on certain position is too low for GATK to reliably call a variant. In such case, it is worth inspecting the depth or depth per base:

```
# Similar as above but one gets depth on particular variant / sample
vt.depth
# One can also get depth for specific base
vt.depth_a
vt.depth_c
vt.depth_t
vt.depth_g
```

Filtering mutations

Process `mutations-table` on Genialis Platform accepts either `mutations` or `geneset` input which specifies the genes of interest. It restricts the scope of mutation search to just a few given genes.

However, it can happen that not all the samples have the same `mutations` or `geneset` input. In such cases, it makes little sense to merge the information about mutations from multiple samples. By default, `VariantTables` checks that all `Data` is computed with same `mutations / geneset` input. If this is not true, it will raise an error.

But if you provide additional argument `geneset` it will limit the mutations to only those in the given `geneset`. An example:

```
# Sample 101 has mutations input "FHIT, BRCA2"
# Sample 102 has mutations input "BRCA2"

# This would cause error, since the mutations inputs are not the same
vt = resdk.tables.VariantTables(<collection>)
vt.variants

# This would limit the variants to just the ones in BRCA2 gene.
vt = resdk.tables.VariantTables(<collection>, geneset=["BRCA2"])
vt.variants
```

3.3.3 Genesets

Geneset is a special kind of `Data` resource. In addition to all of the functionality of `Data`, it also has `genes` attribute and support for set-like operations (intersection, union, etc...).

In the most common case, genesets exist somewhere on Resolwe server and user just fetches them:

```
# Get one geneset by slug
gs = res.geneset.get("my-slug")

# Get all human genesets in a given collection:
genesets = res.geneset.filter(collection=<my-collection>, species="Homo sapiens"):
```

What one gets is an object (or list of them) of type `Geneset`. This object has all the attributes of `Data` plus some additional ones:

```
# Set of genes in the geneset:
gs.genes
# Source of genes, e.g. ENSEMBL, UCSC, NCBI...
gs.source
# Species of the genes in the geneset
gs.species
```

A common thing to do with Geneset objects is to perform set-like operations on them to create new Geneset. This is easily done with exactly the same syntax as for Python set objects:

```
gs1 = res.geneset.get("slug-1")
gs2 = res.geneset.get("slug-2")

# Union
gs3 = gs1 | gs2
# Intersection
gs3 = gs1 & gs2
# Difference
gs3 = gs1 - gs2
```

Note: Performing these operations is only possible on genesets that have equal values of species and source attribute. Otherwise newly created sets would not make sense and would be inconsistent.

So far, geneset `gs3` only exists locally. One can easily save it to Resolwe server:

```
gs3.save()
# As with Data, it is a good practice to include it in a collection:
gs3.collection = <my_collection>
gs.save()
```

Alternative way of creating genesets is to use `Resolwe.geneset.create` method. In such case, you need to enter the genes, species and source information manually:

```
res.geneset.create(genes=["MYC", "FHT"], source="UCSC", species="Homo sapiens")
```

3.3.4 Metadata

Samples are normally annotated with the use of `AnnotationFields` and `AnnotationValues`. However in some cases the available `AnnotationFields` do not suffice and it comes handy to upload sample annotations in a table where each row holds information about some sample in collection. In general, there can be multiple rows referring to the same sample in the collection (for example one sample received two or more distinct treatments). In such cases one can upload this tables with the process `Metadata` table. However, quite often there is exactly one-on-one mapping between rows in such table and samples in collection. In such case, please use the “unique” flavour of the above process, `Metadata` table (one-to-one).

Metadata in ReSDK is just a special kind of `Data` resource that simplifies retrieval of the above mentioned tables. In addition to all of the functionality of `Data`, it also has two additional attributes: `df` and `unique`:

```
# The "df" attribute is pandas.DataFrame of the output named "table"
# The index of df are sample ID's
m.df
```

(continues on next page)

(continued from previous page)

```
# Attribute "unique" is signalling if this is metadata is unique or not
m.unique
```

Note: Behind the scenes, `df` is not an attribute but rather a property. So it has getter and setter methods (`get_df` and `set_df`). This comes handy if the default parsing logic does not suffice. In such cases you can provide your own parser and keyword arguments for it. Example:

```
import pandas
m.get_df(parser=pandas.read_csv, sep="\t", skiprows=[1, 2, 3])
```

In the most common case, Metadata objects exist somewhere on Resolwe server and user just fetches them:

```
# Get one metadata by slug
m = res.metadata.get("my-slug")

# Filter metadata by some conditions, e.g. get all metadata
# from a given collection:
ms = res.metadata.filter(collection=<my-collection>):
```

Sometimes, these objects need to be updated, and one can easily do that. However, `df` and `unique` are upload protected - they can be set during object creation but cannot be set afterwards:

```
m.unique = False # Will fail on already existing object
m.df = <new-df> # Will fail on already existing object
```

Sometimes one wishes to create a new Metadata. This can be achieved in the same manner as for other ReSDK resources:

```
m = res.metadata.create(df=<my-df>, collection=<my-collection>)

# Creating metadata without specifying df / collection will fail
m = res.metadata.create() # Fail
m = res.metadata.create(collection=<my-collection>) # Fail
m = res.metadata.create(df=<my-df>) # Fail
```

Alternatively, one can also build this object gradually from scratch and then call `save()`:

```
m = Metadata(resolwe=<resolwe>)
m.collection = <my-collection>
my_df = m.set_index(<my-df>)
m.df = my_df
m.save()
```

where `m.set_index(<my-df>)` is a helper function that finds `Sample` name/slug/ID column or index name, maps it to `Sample ID` and sets it as index. This function is recommended to use because the validation step is trying to match `m.df` index with `m.collection` sample ID's.

Deleting Metadata works the same as for any other resource. Be careful, this cannot be undone and you need to have sufficient permissions:

```
m.delete()
```

3.4 SDK Reference

3.4.1 Resolwe

class `resdk.Resolwe`(*username=None, password=None, url=None*)

Connect to a Resolwe server.

Parameters

- **username** (*str*) – user’s email
- **password** (*str*) – user’s password
- **url** (*str*) – Resolwe server instance

data_usage(***query_params*)

Get per-user data usage information.

Display number of samples, data objects and sum of data object sizes for currently logged-in user. For admin users, display data for **all** users.

get_or_run(*slug=None, input={}*)

Return existing object if found, otherwise create new one.

Parameters

- **slug** (*str*) – Process slug (human readable unique identifier)
- **input** (*dict*) – Input values

get_query_by_resource(*resource*)

Get ResolweQuery for a given resource.

login(*username=None, password=None*)

Interactive login.

If only username is given prompt the user for password via shell. If username is not given, prompt for interactive login.

run(*slug=None, input={}, descriptor=None, descriptor_schema=None, collection=None, data_name="", process_resources=None*)

Run process and return the corresponding Data object.

1. Upload files referenced in inputs
2. Create Data object with given inputs
3. Command is run that processes inputs into outputs
4. Return Data object

The processing runs asynchronously, so the returned Data object does not have an OK status or outputs when returned. Use `data.update()` to refresh the Data resource object.

Parameters

- **slug** (*str*) – Process slug (human readable unique identifier)
- **input** (*dict*) – Input values
- **descriptor** (*dict*) – Descriptor values
- **descriptor_schema** (*str*) – A valid descriptor schema slug

- **collection** (*int/resource*) – Collection resource or it's id into which data object should be included
- **data_name** (*str*) – Default name of data object
- **process_resources** (*dict*) – Process resources

Returns

data object that was just created

Return type

Data object

version_check()

Check that the server is compatible with the client.

3.4.2 Resolwe Query

class `resdk.ResolweQuery`(*resolwe, resource, slug_field='slug'*)

Query resource endpoints.

A Resolwe instance (for example “res”) has several endpoints:

- `res.data`
- `res.collection`
- `res.sample`
- `res.process`
- ...

Each such endpoint is an instance of the `ResolweQuery` class. `ResolweQuery` supports queries on corresponding objects, for example:

```
res.data.get(42) # return Data object with ID 42.
res.sample.filter(contributor=1) # return all samples made by contributor 1
```

This object is lazy loaded which means that actual request is made only when needed. This enables composing multiple filters, for example:

```
res.data.filter(contributor=1).filter(name='My object')
```

is the same as:

```
res.data.filter(contributor=1, name='My object')
```

This is especially useful, because all endpoints at Resolwe instance are such queries and can be filtered further before transferring any data.

To get a list of all supported query parameters, use one that does not exist and you will get a helpful error message with a list of allowed ones.

```
res.data.filter(foo="bar")
```

all()

Return copy of the current queryset.

This is handy function to get newly created query without any filters.

clear_cache()

Clear cache.

count()

Return number of objects in current query.

create(model_data)**

Return new instance of current resource.

delete(force=False)

Delete objects in current query.

Parameters

force (*bool*) – Do not trigger confirmation prompt. WARNING: Be sure that you really know what you are doing as deleted objects are not recoverable.

filter(filters)**

Return clone of current query with added given filters.

get(*args, **kwargs)

Get object that matches given parameters.

If only one non-keyworded argument is given, it is considered as id if it is number and as slug otherwise.

Parameters

uid (*int for ID or string for slug*) – unique identifier - ID or slug

Return type

object of type `self.resource`

Raises

- **ValueError** – if non-keyworded and keyworded arguments are combined or if more than one non-keyworded argument is given
- **LookupError** – if none or more than one objects are returned

iterate(chunk_size=100, show_progress=False)

Iterate through query.

This can come handy when one wishes to iterate through hundreds or thousands of objects and would otherwise get “504 Gateway-timeout”.

The method cannot be used together with the following filters: limit, offset and ordering, and will raise a `ValueError`.

search(text)

Full text search.

3.4.3 Resources

Resource classes

class `resdk.resources.base.BaseResource(resolwe, **model_data)`

Abstract resource.

One and only one of the identifiers (slug, id or model_data) should be given.

Parameters

- **resolve** (*Resolve object*) – Resolve instance
- **model_data** – Resource model data

delete(*force=False*)

Delete the resource object from the server.

Parameters

force (*bool*) – Do not trigger confirmation prompt. WARNING: Be sure that you really know what you are doing as deleted objects are not recoverable.

classmethod fetch_object(*resolve, id=None, slug=None*)

Return resource instance that is uniquely defined by identifier.

fields()

Resource fields.

id

unique identifier of an object

save()

Save resource to the server.

update()

Update resource fields from the server.

class `resdk.resources.base.BaseResolveResource`(*resolve, **model_data*)

Base class for Resolve resources.

One and only one of the identifiers (slug, id or model_data) should be given.

Parameters

- **resolve** (*Resolve object*) – Resolve instance
- **model_data** – Resource model data

property contributor

Contributor.

property created

Creation time.

current_user_permissions

current user permissions

delete(*force=False*)

Delete the resource object from the server.

Parameters

force (*bool*) – Do not trigger confirmation prompt. WARNING: Be sure that you really know what you are doing as deleted objects are not recoverable.

classmethod fetch_object(*resolve, id=None, slug=None*)

Return resource instance that is uniquely defined by identifier.

fields()

Resource fields.

id

unique identifier of an object

property modified

Modification time.

name

name of resource

property permissions

Permissions.

save()

Save resource to the server.

slug

human-readable unique identifier

update()

Clear permissions cache and update the object.

version

resource version

class `resdk.resources.Data(resolwe, **model_data)`

Resolwe Data resource.

Parameters

- **resolwe** (*Resolwe object*) – Resolwe instance
- **model_data** – Resource model data

checksum

checksum field calculated on inputs

property children

Get children of this Data object.

property collection

Get collection.

property contributor

Contributor.

property created

Creation time.

current_user_permissions

current user permissions

delete(*force=False*)

Delete the resource object from the server.

Parameters

force (*bool*) – Do not trigger confirmation prompt. **WARNING:** Be sure that you really know what you are doing as deleted objects are not recoverable.

descriptor

annotation data, with the form defined in `descriptor_schema`

descriptor_dirty

indicate whether *descriptor* doesn't match *descriptor_schema* (is dirty)

property descriptor_schema

Get descriptor schema.

download(*file_name=None, field_name=None, download_dir=None*)

Download Data object's files and directories.

Download files and directories from the Resolwe server to the download directory (defaults to the current working directory).

Parameters

- **file_name** (*string*) – name of file or directory
- **field_name** (*string*) – file or directory field name
- **download_dir** (*string*) – download path

Return type

None

Data objects can contain multiple files and directories. All are downloaded by default, but may be filtered by name or output field:

- `re.data.get(42).download(file_name='alignment7.bam')`
- `re.data.get(42).download(field_name='bam')`

duplicate()

Duplicate (make copy of) data object.

Returns

Duplicated data object

duplicated

duplicated

classmethod fetch_object(*resolwe, id=None, slug=None*)

Return resource instance that is uniquely defined by identifier.

fields()

Resource fields.

files(*file_name=None, field_name=None*)

Get list of downloadable file fields.

Filter files by file name or output field.

Parameters

- **file_name** (*string*) – name of file
- **field_name** (*string*) – output field name

Return type

List of tuples (data_id, file_name, field_name, process_type)

property finished

Get finish time.

id

unique identifier of an object

input

actual input values

property modified

Modification time.

name

name of resource

output

actual output values

property parents

Get parents of this Data object.

property permissions

Permissions.

property process

Get process.

process_cores

process cores

process_error

error log message (list of strings)

process_info

info log message (list of strings)

process_memory

process memory

process_progress

process progress in percentage

process_rc

Process algorithm return code

process_resources

process_resources

process_warning

warning log message (list of strings)

property sample

Get sample.

save()

Save resource to the server.

scheduled

scheduled

size

size

slug

human-readable unique identifier

property started

Get start time.

status

process status - Possible values: UP (Uploading - for upload processes), RE (Resolving - computing input data objects) WT (Waiting - waiting for process since the queue is full) PP (Preparing - preparing the environment for processing) PR (Processing) OK (Done) ER (Error) DR (Dirty - Data is dirty)

stdout()

Return process standard output (stdout.txt file content).

Fetch stdout.txt file from the corresponding Data object and return the file content as string. The string can be long and ugly.

Return type

string

tags

data object's tags

update()

Clear cache and update resource fields from the server.

version

resource version

class `resdk.resources.collection.BaseCollection`(*resolwe*, ***model_data*)

Abstract collection resource.

One and only one of the identifiers (slug, id or model_data) should be given.

Parameters

- **resolwe** (*Resolwe object*) – Resolwe instance
- **model_data** – Resource model data

property contributor

Contributor.

property created

Creation time.

current_user_permissions

current user permissions

property data

Return list of attached Data objects.

data_types()

Return a list of data types (process_type).

Return type

List

delete(*force=False*)

Delete the resource object from the server.

Parameters

force (*bool*) – Do not trigger confirmation prompt. WARNING: Be sure that you really know what you are doing as deleted objects are not recoverable.

description

description

descriptor

descriptor

descriptor_dirty

descriptor_dirty

property descriptor_schema

Descriptor schema.

download(*file_name=None, field_name=None, download_dir=None*)

Download output files of associated Data objects.

Download files from the Resolwe server to the download directory (defaults to the current working directory).

Parameters

- **file_name** (*string*) – name of file
- **field_name** (*string*) – field name
- **download_dir** (*string*) – download path

Return type

None

Collections can contain multiple Data objects and Data objects can contain multiple files. All files are downloaded by default, but may be filtered by file name or Data object type:

- `re.collection.get(42).download(file_name='alignment7.bam')`
- `re.collection.get(42).download(data_type='bam')`

duplicated

duplicated

classmethod fetch_object(*resolwe, id=None, slug=None*)

Return resource instance that is uniquely defined by identifier.

fields()

Resource fields.

files(*file_name=None, field_name=None*)

Return list of files in resource.

id

unique identifier of an object

property modified

Modification time.

name

name of resource

property permissions

Permissions.

save()

Save resource to the server.

settings

settings

slug

human-readable unique identifier

tags

tags

update()

Clear cache and update resource fields from the server.

version

resource version

class `resdk.resources.Collection(resolwe, **model_data)`

Resolwe Collection resource.

Parameters

- **resolwe** (*Resolwe object*) – Resolwe instance
- **model_data** – Resource model data

assign_to_billing_account(*billing_account_name*)

Assign given collection to a billing account.

property contributor

Contributor.

create_background_relation(*category, background, cases*)

Create background relation.

Parameters

- **category** (*str*) – Category of relation
- **background** (*Sample*) – Background sample
- **cases** (*Sample*) – Case samples (signals)

create_compare_relation(*category, samples, labels=[]*)

Create compare relation.

Parameters

- **category** (*str*) – Category of relation (i.e. case-control, ...)
- **samples** (*list*) – List of samples to include in relation.
- **labels** (*list*) – List of labels assigned to corresponding samples. If given it should be of same length as samples.

create_group_relation(*category, samples, labels=[]*)

Create group relation.

Parameters

- **category** (*str*) – Category of relation (i.e. replicates, clones, ...)
- **samples** (*list*) – List of samples to include in relation.
- **labels** (*list*) – List of labels assigned to corresponding samples. If given it should be of same length as samples.

create_series_relation(*category, samples, positions=[], labels=[]*)

Create series relation.

Parameters

- **category** (*str*) – Category of relation (i.e. case-control, ...)
- **samples** (*list*) – List of samples to include in relation.
- **positions** (*list*) – List of positions assigned to corresponding sample (i.e. 10, 20, 30). If given it should be of same length as samples. Note that this elements should be machine-sortable by default.
- **labels** (*list*) – List of labels assigned to corresponding samples. If given it should be of same length as samples.

property created

Creation time.

current_user_permissions

current user permissions

property data

Return list of data objects on collection.

data_types()

Return a list of data types (process_type).

Return type

List

delete(*force=False*)

Delete the resource object from the server.

Parameters

- **force** (*bool*) – Do not trigger confirmation prompt. **WARNING:** Be sure that you really know what you are doing as deleted objects are not recoverable.

description

description

descriptor

descriptor

descriptor_dirty

descriptor_dirty

property descriptor_schema

Descriptor schema.

download(*file_name=None, field_name=None, download_dir=None*)

Download output files of associated Data objects.

Download files from the Resolwe server to the download directory (defaults to the current working directory).

Parameters

- **file_name** (*string*) – name of file
- **field_name** (*string*) – field name
- **download_dir** (*string*) – download path

Return type

None

Collections can contain multiple Data objects and Data objects can contain multiple files. All files are downloaded by default, but may be filtered by file name or Data object type:

- `re.collection.get(42).download(file_name='alignment7.bam')`
- `re.collection.get(42).download(data_type='bam')`

duplicate()

Duplicate (make copy of) collection object.

Returns

Duplicated collection

duplicated

duplicatied

classmethod fetch_object(*resolwe, id=None, slug=None*)

Return resource instance that is uniquely defined by identifier.

fields()

Resource fields.

files(*file_name=None, field_name=None*)

Return list of files in resource.

id

unique identifier of an object

property modified

Modification time.

name

name of resource

property permissions

Permissions.

property relations

Return list of data objects on collection.

property samples

Return list of samples on collection.

save()

Save resource to the server.

settings

settings

slug

human-readable unique identifier

tags

tags

update()

Clear cache and update resource fields from the server.

version

resource version

class `resdk.resources.Sample(resolwe, **model_data)`

Resolwe Sample resource.

Parameters

- **resolwe** (*Resolwe object*) – Resolwe instance
- **model_data** – Resource model data

property annotations

Get the annotations for the given sample.

property background

Get background sample of the current one.

property collection

Get collection.

property contributor

Contributor.

property created

Creation time.

current_user_permissions

current user permissions

property data

Get data.

data_types()

Return a list of data types (*process_type*).

Return type

List

delete(force=False)

Delete the resource object from the server.

Parameters

force (*bool*) – Do not trigger confirmation prompt. **WARNING:** Be sure that you really know what you are doing as deleted objects are not recoverable.

description

description

descriptor

descriptor

descriptor_dirty

descriptor_dirty

property descriptor_schema

Descriptor schema.

download(*file_name=None, field_name=None, download_dir=None*)

Download output files of associated Data objects.

Download files from the Resolwe server to the download directory (defaults to the current working directory).

Parameters

- **file_name** (*string*) – name of file
- **field_name** (*string*) – field name
- **download_dir** (*string*) – download path

Return type

None

Collections can contain multiple Data objects and Data objects can contain multiple files. All files are downloaded by default, but may be filtered by file name or Data object type:

- `re.collection.get(42).download(file_name='alignment7.bam')`
- `re.collection.get(42).download(data_type='bam')`

duplicate()

Duplicate (make copy of) sample object.

Returns

Duplicated sample

duplicated

duplicated

classmethod fetch_object(*resolwe, id=None, slug=None*)

Return resource instance that is uniquely defined by identifier.

fields()

Resource fields.

files(*file_name=None, field_name=None*)

Return list of files in resource.

get_annotation(*full_path: str*) → *AnnotationValue*

Get the AnnotationValue from full path.

Raises

LookupError – when field at the specified path does not exist.

get_annotations() → *Dict[str, Any]*

Get all annotations for the given sample in a dictionary.

get_bam()

Return bam object on the sample.

get_cuffquant()

Get cuffquant.

get_expression()

Get expression.

get_macs()

Return list of bed objects on the sample.

get_primary_bam(*fallback_to_bam=False*)

Return primary bam object on the sample.

If the primary bam object is not present and *fallback_to_bam* is set to True, a bam object will be returned.

get_reads(*filters*)**

Return the latest fastq object in sample.

If there are multiple fastq objects in sample (trimmed, filtered, subsampled...), return the latest one. If any other of the fastq objects is required, one can provide additional *filter* arguments and limits search to one result.

id

unique identifier of an object

property is_background

Return True if given sample is background to any other and False otherwise.

property modified

Modification time.

name

name of resource

property permissions

Permissions.

property relations

Get Relation objects for this sample.

save()

Save resource to the server.

set_annotation(*full_path: str, value, force=False*) → Optional[*AnnotationValue*]

Create/update annotation value.

If value is None the annotation is deleted and None is returned. If force is set to True no explicit confirmation is required to delete the annotation.

set_annotations(*annotations: Dict[str, Any]*)

Bulk set annotations on the sample.

settings

settings

slug

human-readable unique identifier

tags

tags

update()

Clear cache and update resource fields from the server.

version

resource version

class `resdk.resources.Relation(resolve, **model_data)`

Resolve Relation resource.

Parameters

- **resolve** (*Resolve object*) – Resolve instance
- **model_data** – Resource model data

add_sample(*sample, label=None, position=None*)

Add `sample` object to relation.

category

category of the relation

property collection

Return collection object to which relation belongs.

property contributor

Contributor.

property created

Creation time.

current_user_permissions

current user permissions

delete(*force=False*)

Delete the resource object from the server.

Parameters

- **force** (*bool*) – Do not trigger confirmation prompt. **WARNING:** Be sure that you really know what you are doing as deleted objects are not recoverable.

descriptor

annotation data, with the form defined in `descriptor_schema`

descriptor_dirty

indicate whether *descriptor* doesn't match *descriptor_schema* (is dirty)

property descriptor_schema

Get descriptor schema.

classmethod `fetch_object(resolve, id=None, slug=None)`

Return resource instance that is uniquely defined by identifier.

fields()

Resource fields.

id

unique identifier of an object

property modified

Modification time.

name

name of resource

partitions

list of RelationPartition objects in the Relation

property permissions

Permissions.

remove_samples(*samples)

Remove sample objects from relation.

property samples

Return list of sample objects in the relation.

save()

Check that collection is saved and save instance.

slug

human-readable unique identifier

type

type of the relation

unit(where applicable, e.g. for serieses)

unit (where applicable, e.g. for serieses)

update()

Clear cache and update resource fields from the server.

version

resource version

class `resdk.resources.Process(resolwe, **model_data)`

Resolwe Process resource.

Parameters

- **resolwe** (*Resolwe object*) – Resolwe instance
- **model_data** – Resource model data

category

used to group processes in a GUI. Examples: `upload:`, `analyses:variants:`, ...

property contributor

Contributor.

property created

Creation time.

current_user_permissions

current user permissions

data_name

the default name of data object using this process. When data object is created you can assign a name to it. But if you don't, the name of data object is determined from this field. The field is a expression which can take values of other fields.

delete(*force=False*)

Delete the resource object from the server.

Parameters

force (*bool*) – Do not trigger confirmation prompt. **WARNING:** Be sure that you really know what you are doing as deleted objects are not recoverable.

description

process description

entity_always_create

entity_always_create

entity_descriptor_schema

entity_descriptor_schema

entity_input

entity_input

entity_type

entity_type

classmethod **fetch_object**(*resolwe, id=None, slug=None*)

Return resource instance that is uniquely defined by identifier.

fields()

Resource fields.

id

unique identifier of an object

input_schema

specifications of inputs

is_active

Boolean stating wether process is active

property modified

Modification time.

name

name of resource

output_schema

specification of outputs

property permissions

Permissions.

persistence

Measure of how important is to keep the process outputs when optimizing disk usage. Options: RAW/CACHED/TEMP. For processes, used on frontend use TEMP - the results of this processes can be quickly re-calculated any time. For upload processes use RAW - this data should never be deleted, since it

cannot be re-calculated. For analysis use CACHED - the results can still be calculated from imported data but it can take time.

print_inputs()

Pretty print input_schema.

priority

process priority - not used yet

requirements

required Docker image, amount of memory / CPU ...

run

the heart of process - here the algorithm is defined.

save()

Save resource to the server.

scheduling_class

Scheduling class

slug

human-readable unique identifier

type

the type of process "type:sub_type:sub_sub_type:..."

update()

Clear permissions cache and update the object.

version

resource version

class resdk.resources.DescriptorSchema(resolwe, **model_data)

Resolwe DescriptorSchema resource.

Parameters

- **resolwe** (*Resolwe object*) – Resolwe instance
- **model_data** – Resource model data

property contributor

Contributor.

property created

Creation time.

current_user_permissions

current user permissions

delete(*force=False*)

Delete the resource object from the server.

Parameters

force (*bool*) – Do not trigger confirmation prompt. **WARNING:** Be sure that you really know what you are doing as deleted objects are not recoverable.

description

description

classmethod `fetch_object(resolve, id=None, slug=None)`

Return resource instance that is uniquely defined by identifier.

fields()

Resource fields.

id

unique identifier of an object

property `modified`

Modification time.

name

name of resource

property `permissions`

Permissions.

save()

Save resource to the server.

schema

schema

slug

human-readable unique identifier

update()

Clear permissions cache and update the object.

version

resource version

class `resdk.resources.AnnotationValue(resolve: Resolve, **model_data)`

Resolve AnnotationValue resource.

delete(*force=False*)

Delete the resource object from the server.

Parameters

force (*bool*) – Do not trigger confirmation prompt. **WARNING:** Be sure that you really know what you are doing as deleted objects are not recoverable.

classmethod `fetch_object(resolve, id=None, slug=None)`

Return resource instance that is uniquely defined by identifier.

property `field: AnnotationField`

Get annotation field.

fields()

Resource fields.

id

unique identifier of an object

property `sample`

Get sample.

sample_id: `Optional[int]`

sample

save()

Save resource to the server.

update()

Update resource fields from the server.

class `resdk.resources.AnnotationGroup`(*resolve: Resolve, **model_data*)

Resolve AnnotationGroup resource.

delete(*force=False*)

Delete the resource object from the server.

Parameters

force (*bool*) – Do not trigger confirmation prompt. WARNING: Be sure that you really know what you are doing as deleted objects are not recoverable.

classmethod `fetch_object`(*resolve, id=None, slug=None*)

Return resource instance that is uniquely defined by identifier.

fields()

Resource fields.

id

unique identifier of an object

save()

Save resource to the server.

update()

Update resource fields from the server.

class `resdk.resources.AnnotationField`(*resolve: Resolve, **model_data*)

Resolve AnnotationField resource.

delete(*force=False*)

Delete the resource object from the server.

Parameters

force (*bool*) – Do not trigger confirmation prompt. WARNING: Be sure that you really know what you are doing as deleted objects are not recoverable.

classmethod `fetch_object`(*resolve, id=None, slug=None*)

Return resource instance that is uniquely defined by identifier.

fields()

Resource fields.

property `group: AnnotationGroup`

Get annotation group.

id

unique identifier of an object

save()

Save resource to the server.

update()

Update resource fields from the server.

class `resdk.resources.User`(*resolwe=None, **model_data*)

Resolwe User resource.

One and only one of the identifiers (slug, id or model_data) should be given.

Parameters

- **resolwe** (*Resolwe object*) – Resolwe instance
- **model_data** – Resource model data

delete(*force=False*)

Delete the resource object from the server.

Parameters

force (*bool*) – Do not trigger confirmation prompt. **WARNING:** Be sure that you really know what you are doing as deleted objects are not recoverable.

classmethod `fetch_object`(*resolwe, id=None, slug=None*)

Return resource instance that is uniquely defined by identifier.

fields()

Resource fields.

first_name

user's first name

get_name()

Return user's name.

id

unique identifier of an object

save()

Save resource to the server.

update()

Update resource fields from the server.

class `resdk.resources.Group`(*resolwe=None, **model_data*)

Resolwe Group resource.

One and only one of the identifiers (slug, id or model_data) should be given.

Parameters

- **resolwe** (*Resolwe object*) – Resolwe instance
- **model_data** – Resource model data

add_users(**users*)

Add users to group.

delete(*force=False*)

Delete the resource object from the server.

Parameters

force (*bool*) – Do not trigger confirmation prompt. **WARNING:** Be sure that you really know what you are doing as deleted objects are not recoverable.

classmethod `fetch_object`(*resolwe, id=None, slug=None*)

Return resource instance that is uniquely defined by identifier.

fields()

Resource fields.

id

unique identifier of an object

name

group's name

remove_users(**users*)

Remove users from group.

save()

Save resource to the server.

update()

Clear cache and update resource fields from the server.

property users

Return list of users in group.

class `resdk.resources.Geneset`(*resolwe, genes=None, source=None, species=None, **model_data*)

Resolwe Geneset resource.

Parameters

- **resolwe** (*Resolwe object*) – Resolwe instance
- **model_data** – Resource model data

checksum

checksum field calculated on inputs

property children

Get children of this Data object.

property collection

Get collection.

property contributor

Contributor.

property created

Creation time.

current_user_permissions

current user permissions

delete(*force=False*)

Delete the resource object from the server.

Parameters

force (*bool*) – Do not trigger confirmation prompt. **WARNING:** Be sure that you really know what you are doing as deleted objects are not recoverable.

descriptor

annotation data, with the form defined in `descriptor_schema`

descriptor_dirty

indicate whether *descriptor* doesn't match *descriptor_schema* (is dirty)

property descriptor_schema

Get descriptor schema.

download(*file_name=None, field_name=None, download_dir=None*)

Download Data object's files and directories.

Download files and directories from the Resolwe server to the download directory (defaults to the current working directory).

Parameters

- **file_name** (*string*) – name of file or directory
- **field_name** (*string*) – file or directory field name
- **download_dir** (*string*) – download path

Return type

None

Data objects can contain multiple files and directories. All are downloaded by default, but may be filtered by name or output field:

- `re.data.get(42).download(file_name='alignment7.bam')`
- `re.data.get(42).download(field_name='bam')`

duplicate()

Duplicate (make copy of) data object.

Returns

Duplicated data object

duplicated

duplicated

classmethod fetch_object(*resolwe, id=None, slug=None*)

Return resource instance that is uniquely defined by identifier.

fields()

Resource fields.

files(*file_name=None, field_name=None*)

Get list of downloadable file fields.

Filter files by file name or output field.

Parameters

- **file_name** (*string*) – name of file
- **field_name** (*string*) – output field name

Return type

List of tuples (data_id, file_name, field_name, process_type)

property finished

Get finish time.

property genes

Get genes.

id

unique identifier of an object

input

actual input values

property modified

Modification time.

name

name of resource

output

actual output values

property parents

Get parents of this Data object.

property permissions

Permissions.

property process

Get process.

process_cores

process cores

process_error

error log message (list of strings)

process_info

info log message (list of strings)

process_memory

process memory

process_progress

process progress in percentage

process_rc

Process algorithm return code

process_resources

process_resources

process_warning

warning log message (list of strings)

property sample

Get sample.

save()

Save Geneset to the server.

If Geneset is already on the server update with save() from base class. Otherwise, create a new Geneset by running process with slug “create-geneset”.

scheduled

scheduled

set_operator(*operator, other*)

Perform set operations on Geneset object by creating a new Genseset.

Parameters

- **operator** – string -> set operation function name
- **other** – Geneset object

Returns

new Geneset object

size

size

slug

human-readable unique identifier

property source

Get source.

property species

Get species.

property started

Get start time.

status

process status - Possible values: UP (Uploading - for upload processes), RE (Resolving - computing input data objects) WT (Waiting - waiting for process since the queue is full) PP (Preparing - preparing the environment for processing) PR (Processing) OK (Done) ER (Error) DR (Dirty - Data is dirty)

stdout()

Return process standard output (stdout.txt file content).

Fetch stdout.txt file from the corresponding Data object and return the file content as string. The string can be long and ugly.

Return type

string

tags

data object's tags

update()

Clear cache and update resource fields from the server.

version

resource version

class `resdk.resources.Metadata`(*resolwe, **model_data*)

Metadata resource.

Parameters

- **resolwe** (*Resolwe object*) – Resolwe instance
- **model_data** – Resource model data

checksum

checksum field calculated on inputs

property children

Get children of this Data object.

property collection

Get collection.

property contributor

Contributor.

property created

Creation time.

current_user_permissions

current user permissions

delete(*force=False*)

Delete the resource object from the server.

Parameters

force (*bool*) – Do not trigger confirmation prompt. WARNING: Be sure that you really know what you are doing as deleted objects are not recoverable.

descriptor

annotation data, with the form defined in `descriptor_schema`

descriptor_dirty

indicate whether *descriptor* doesn't match *descriptor_schema* (is dirty)

property descriptor_schema

Get descriptor schema.

property df

Get table as `pd.DataFrame`.

property df_bytes

Get file contents of table output in bytes form.

download(*file_name=None, field_name=None, download_dir=None*)

Download Data object's files and directories.

Download files and directories from the Resolwe server to the download directory (defaults to the current working directory).

Parameters

- **file_name** (*string*) – name of file or directory
- **field_name** (*string*) – file or directory field name
- **download_dir** (*string*) – download path

Return type

None

Data objects can contain multiple files and directories. All are downloaded by default, but may be filtered by name or output field:

- `re.data.get(42).download(file_name='alignment7.bam')`

- `re.data.get(42).download(field_name='bam')`

duplicate()

Duplicate (make copy of) data object.

Returns

Duplicated data object

duplicated

duplicated

classmethod fetch_object (*resolwe*, *id=None*, *slug=None*)

Return resource instance that is uniquely defined by identifier.

fields()

Resource fields.

files (*file_name=None*, *field_name=None*)

Get list of downloadable file fields.

Filter files by file name or output field.

Parameters

- **file_name** (*string*) – name of file
- **field_name** (*string*) – output field name

Return type

List of tuples (data_id, file_name, field_name, process_type)

property finished

Get finish time.

get_df (*parser=None*, ***kwargs*)

Get table as pd.DataFrame.

id

unique identifier of an object

input

actual input values

property modified

Modification time.

name

name of resource

output

actual output values

property parents

Get parents of this Data object.

property permissions

Permissions.

property process

Get process.

process_cores

process cores

process_error

error log message (list of strings)

process_info

info log message (list of strings)

process_memory

process memory

process_progress

process progress in percentage

process_rc

Process algorithm return code

process_resources

process_resources

process_warning

warning log message (list of strings)

property sample

Get sample.

save()

Save Metadata to the server.

If Metadata is already uploaded: update. Otherwise, create new one.

scheduled

scheduled

set_df(*value*)

Set df.

set_index(*df*)

Set index of df to Sample ID.

If there is a column with Sample ID just set that as index. If there is Sample name or Sample slug column, map sample name / slug to sample ID's and set ID's as an index. If no suitable column in there, raise an error. Works also if any of the above options is already an index with appropriate name.

size

size

slug

human-readable unique identifier

property started

Get start time.

status

process status - Possible values: UP (Uploading - for upload processes), RE (Resolving - computing input data objects) WT (Waiting - waiting for process since the queue is full) PP (Preparing - preparing the environment for processing) PR (Processing) OK (Done) ER (Error) DR (Dirty - Data is dirty)

stdout()

Return process standard output (stdout.txt file content).

Fetch stdout.txt file from the corresponding Data object and return the file content as string. The string can be long and ugly.

Return type

string

tags

data object's tags

property unique

Get unique attribute.

This attribute tells if Metadata has one-to-one or one-to-many relation to collection samples.

update()

Clear cache and update resource fields from the server.

validate_df(df)

Validate df property.

Validates that df:

- is an instance of pandas.DataFrame
- index contains sample IDs that match some samples:
 - If not matches, raise warning
 - If there are samples in df but not in collection, raise warning
 - If there are samples in collection but not in df, raise warning

version

resource version

class `resdk.resources.kb.Feature(resolwe, **model_data)`

Knowledge base Feature resource.

aliases

Aliases

delete(force=False)

Delete the resource object from the server.

Parameters

force (bool) – Do not trigger confirmation prompt. WARNING: Be sure that you really know what you are doing as deleted objects are not recoverable.

description

Description

feature_id

Feature ID

classmethod `fetch_object(resolwe, id=None, slug=None)`

Return resource instance that is uniquely defined by identifier.

fields()

Resource fields.

full_name

Full name

id

unique identifier of an object

name

Name

save()

Save resource to the server.

source

Source

species

Species

sub_type

Feature subtype (tRNA, protein coding, rRNA, ...)

type

Feature type (gene, transcript, exon, ...)

update()

Update resource fields from the server.

class `resdk.resources.kb.Mapping(resolwe, **model_data)`

Knowledge base Mapping resource.

delete(force=False)

Delete the resource object from the server.

Parameters

force (*bool*) – Do not trigger confirmation prompt. WARNING: Be sure that you really know what you are doing as deleted objects are not recoverable.

classmethod `fetch_object(resolwe, id=None, slug=None)`

Return resource instance that is uniquely defined by identifier.

fields()

Resource fields.

id

unique identifier of an object

save()

Save resource to the server.

source_db

Source database

source_id

Source feature ID

source_species

Source feature species

target_db

Target database

target_id

Target feature ID

target_species

Target feature species

update()

Update resource fields from the server.

Permissions

Resources like `resdk.resources.Data`, `resdk.resources.Collection`, `resdk.resources.Sample`, and `resdk.resources.Process` include a `permissions` attribute to manage permissions. The `permissions` attribute is an instance of `resdk.resources.permissions.PermissionsManager`.

class `resdk.resources.permissions.PermissionsManager`(*all_permissions, api_root, resolwe*)

Helper class to manage permissions of the BaseResource.

clear_cache()

Clear cache.

copy_from(*source*)

Copy permissions from some other object to self.

property editors

Get users with edit permission.

fetch()

Fetch permissions from server.

property owners

Get users with owner permission.

set_group(*group, perm*)

Set perm permission to group.

When assigning permissions, only the highest permission needs to be given. Permission hierarchy is:

- none (no permissions)
- view
- edit
- share
- owner

Some examples:

```
collection = res.collection.get(...)
# Add share, edit and view permission to BioLab:
collection.permissions.set_group('biolab', 'share')
# Remove share and edit permission from BioLab:
collection.permissions.set_group('biolab', 'view')
```

(continues on next page)

(continued from previous page)

```
# Remove all permissions from BioLab:
collection.permissions.set_group('biolab', 'none')
```

set_public(*perm*)

Set *perm* permission for public.

Public can only get two sorts of permissions:

- none (no permissions)
- view

Some examples:

```
collection = res.collection.get(...)
# Add view permission to public:
collection.permissions.set_public('view')
# Remove view permission from public:
collection.permissions.set_public('none')
```

set_user(*user, perm*)

Set *perm* permission to *user*.

When assigning permissions, only the highest permission needs to be given. Permission hierarchy is:

- none (no permissions)
- view
- edit
- share
- owner

Some examples:

```
collection = res.collection.get(...)
# Add share, edit and view permission to John:
collection.permissions.set_user('john', 'share')
# Remove share and edit permission from John:
collection.permissions.set_user('john', 'view')
# Remove all permissions from John:
collection.permissions.set_user('john', 'none')
```

property viewers

Get users with view permission.

Utility functions

Resource utility functions.

`resdk.resources.utils.fill_spaces(word, desired_length)`

Fill spaces at the end until word reaches desired length.

`resdk.resources.utils.flatten_field(field, schema, path)`

Reduce dicts of dicts to dot separated keys.

Parameters

- **field** (*dict*) – Field instance (e.g. input)
- **schema** (*dict*) – Schema instance (e.g. input_schema)
- **path** (*string*) – Field path

Returns

flattened instance

Return type

dictionary

`resdk.resources.utils.get_collection_id(collection)`

Return id attribute of the object if it is collection, otherwise return given value.

`resdk.resources.utils.get_data_id(data)`

Return id attribute of the object if it is data, otherwise return given value.

`resdk.resources.utils.get_descriptor_schema_id(dschema)`

Get descriptor schema id.

Return id attribute of the object if it is descriptor schema, otherwise return given value.

`resdk.resources.utils.get_process_id(process)`

Return id attribute of the object if it is process, otherwise return given value.

`resdk.resources.utils.get_relation_id(relation)`

Return id attribute of the object if it is relation, otherwise return given value.

`resdk.resources.utils.get_sample_id(sample)`

Return id attribute of the object if it is sample, otherwise return given value.

`resdk.resources.utils.get_user_id(user)`

Return id attribute of the object if it is relation, otherwise return given value.

`resdk.resources.utils.is_collection(collection)`

Return True if passed object is Collection and False otherwise.

`resdk.resources.utils.is_data(data)`

Return True if passed object is Data and False otherwise.

`resdk.resources.utils.is_descriptor_schema(data)`

Return True if passed object is DescriptorSchema and False otherwise.

`resdk.resources.utils.is_group(group)`

Return True if passed object is Group and False otherwise.

`resdk.resources.utils.is_process(process)`

Return True if passed object is Process and False otherwise.

`resdk.resources.utils.is_relation(relation)`

Return True if passed object is Relation and False otherwise.

`resdk.resources.utils.is_sample(sample)`

Return True if passed object is Sample and False otherwise.

`resdk.resources.utils.is_user(user)`

Return True if passed object is User and False otherwise.

`resdk.resources.utils.iterate_fields(fields, schema)`

Recursively iterate over all DictField sub-fields.

Parameters

- **fields** (*dict*) – Field instance (e.g. input)
- **schema** (*dict*) – Schema instance (e.g. input_schema)

`resdk.resources.utils.iterate_schema(fields, schema, path=None)`

Recursively iterate over all schema sub-fields.

Parameters

- **fields** (*dict*) – Field instance (e.g. input)
- **schema** (*dict*) – Schema instance (e.g. input_schema)

Path schema

Field path

Path schema

string

`resdk.resources.utils.parse_resolwe_datetime(dtime)`

Convert string representation of time to local datetime.datetime object.

3.4.4 ReSDK Tables

Helper classes for aggregating collection data in tabular format.

Table classes

`class resdk.tables.microarray.MATables(collection: Collection, cache_dir: Optional[str] = None, progress_callable: Optional[Callable] = None)`

A helper class to fetch collection’s microarray, qc and meta data.

This class enables fetching given collection’s data and returning it as tables which have samples in rows and microarray / qc / metadata in columns.

A simple example:

```
# Get Collection object
collection = res.collection.get("collection-slug")

# Fetch collection microarray and metadata
tables = MATables(collection)
meta = tables.meta
exp = tables.exp
```

```
__init__(collection: Collection, cache_dir: Optional[str] = None, progress_callable: Optional[Callable] = None)
```

Initialize class.

Parameters

- **collection** – collection to use
- **cache_dir** – cache directory location, if not specified system specific cache directory is used
- **progress_callable** – custom callable that can be used to report progress. By default, progress is written to stderr with tqdm

```
static clear_cache() → None
```

Remove ReSDK cache files from the default cache directory.

```
property exp: DataFrame
```

Return expressions values table as a pandas DataFrame object.

```
property meta: DataFrame
```

Return samples metadata table as a pandas DataFrame object.

Returns

table of metadata

```
property qc: DataFrame
```

Return samples QC table as a pandas DataFrame object.

Returns

table of QC values

```
property readable_index: Dict[int, str]
```

Get mapping from index values to readable names.

```
class resdk.tables.ml_ready.MLTables(collection, name)
```

Machine-learning ready tables.

```
__init__(collection, name)
```

Initialize class.

Parameters

collection – Collection to use

```
property exp
```

Get ML ready expressions as pandas.DataFrame.

These expressions are normalized and batch effect corrected - thus ready to be taken into ML procedures.

```
class resdk.tables.rna.RNATables(collection: Collection, cache_dir: Optional[str] = None,
                                progress_callable: Optional[Callable] = None, expression_source:
                                Optional[str] = None, expression_process_slug: Optional[str] = None)
```

A helper class to fetch collection's expression and meta data.

This class enables fetching given collection's data and returning it as tables which have samples in rows and expressions/metadata in columns.

When calling `RNATables.exp`, `RNATables.rc` and `RNATables.meta` for the first time the corresponding data gets downloaded from the server. This data then gets cached in memory and on disc and is used in consequent calls. If the data on the server changes the updated version gets re-downloaded.

A simple example:

```
# Get Collection object
collection = res.collection.get("collection-slug")

# Fetch collection expressions and metadata
tables = RNATables(collection)
exp = tables.exp
rc = tables.rc
meta = tables.meta
```

```
__init__(collection: Collection, cache_dir: Optional[str] = None, progress_callable: Optional[Callable] = None, expression_source: Optional[str] = None, expression_process_slug: Optional[str] = None)
```

Initialize class.

Parameters

- **collection** – collection to use
- **cache_dir** – cache directory location, if not specified system specific cache directory is used
- **progress_callable** – custom callable that can be used to report progress. By default, progress is written to stderr with tqdm
- **expression_source** – Only consider samples in the collection with specified source
- **expression_process_slug** – Only consider samples in the collection with specified process slug

property build: `str`

Get build.

check_heterogeneous_collections()

Ensure consistency among expressions.

static clear_cache() → `None`

Remove ReSDK cache files from the default cache directory.

property exp: `DataFrame`

Return expressions table as a pandas DataFrame object.

Which type of expressions (TPM, CPM, FPKM, ...) get returned depends on how the data was processed. The expression type can be checked in the returned table attribute `attrs['exp_type']`:

```
exp = tables.exp
print(exp.attrs['exp_type'])
```

Returns

table of expressions

property meta: `DataFrame`

Return samples metadata table as a pandas DataFrame object.

Returns

table of metadata

property qc: `DataFrame`

Return samples QC table as a pandas DataFrame object.

Returns

table of QC values

property rc: DataFrame

Return expression counts table as a pandas DataFrame object.

Returns

table of counts

property readable_columns: Dict[str, str]

Map of source gene ids to symbols.

This also gets fetched only once and then cached in memory and on disc. `RNATables.exp` or `RNATables.rc` must be called before this as the mapping is specific to just this data. Its intended use is to rename table column labels from gene ids to symbols.

Example of use:

```
exp = exp.rename(columns=tables.readable_columns)
```

Returns

dict with gene ids as keys and gene symbols as values

property readable_index: Dict[int, str]

Get mapping from index values to readable names.

```
class resdk.tables.methylation.MethylationTables(collection: Collection, cache_dir: Optional[str] =
None, progress_callable: Optional[Callable] =
None)
```

A helper class to fetch collection's methylation and meta data.

This class enables fetching given collection's data and returning it as tables which have samples in rows and methylation/metadata in columns.

A simple example:

```
# Get Collection object
collection = res.collection.get("collection-slug")

# Fetch collection methylation and metadata
tables = MethylationTables(collection)
meta = tables.meta
beta = tables.beta
m_values = tables.mval
```

```
__init__(collection: Collection, cache_dir: Optional[str] = None, progress_callable: Optional[Callable] =
None)
```

Initialize class.

Parameters

- **collection** – collection to use
- **cache_dir** – cache directory location, if not specified system specific cache directory is used
- **progress_callable** – custom callable that can be used to report progress. By default, progress is written to stderr with tqdm

property beta: DataFrame

Return beta values table as a pandas DataFrame object.

static clear_cache() → None

Remove ReSDK cache files from the default cache directory.

property meta: DataFrame

Return samples metadata table as a pandas DataFrame object.

Returns

table of metadata

property mval: DataFrame

Return m-values as a pandas DataFrame object.

property qc: DataFrame

Return samples QC table as a pandas DataFrame object.

Returns

table of QC values

property readable_index: Dict[int, str]

Get mapping from index values to readable names.

```
class resdk.tables.variant.VariantTables(collection: Collection, geneset: Optional[List[str]] = None,
                                         filtering: bool = True, cache_dir: Optional[str] = None,
                                         progress_callable: Optional[Callable] = None)
```

A helper class to fetch collection's variant and meta data.

This class enables fetching given collection's data and returning it as tables which have samples in rows and variants in columns.

A simple example:

```
# Get Collection object
collection = res.collection.get("collection-slug")

tables = VariantTables(collection)
# Get variant data
tables.variants
# Get depth per variant or coverage for specific base
tables.depth
tables.depth_a
tables.depth_c
tables.depth_g
tables.depth_t
```

```
__init__(collection: Collection, geneset: Optional[List[str]] = None, filtering: bool = True, cache_dir:
         Optional[str] = None, progress_callable: Optional[Callable] = None)
```

Initialize class.

Parameters

- **collection** – Collection to use.
- **geneset** – Only consider mutations from this gene-set. Can be a list of gene symbols or a valid geneset Data object id / slug.
- **filtering** – Only show variants that pass QC filters.

- **cache_dir** – Cache directory location, if not specified system specific cache directory is used.
- **progress_callable** – Custom callable that can be used to report progress. By default, progress is written to stderr with tqdm.

static clear_cache() → None

Remove ReSDK cache files from the default cache directory.

property depth: DataFrame

Get depth table.

property depth_a: DataFrame

Get depth table for adenine.

property depth_c: DataFrame

Get depth table for cytosine.

property depth_g: DataFrame

Get depth table for guanine.

property depth_t: DataFrame

Get depth table for thymine.

property filter: DataFrame

Get filter table.

Values can be:

- PASS - Variant has passed filters:
- DP : Insufficient read depth (< 10.0)
- QD: insufficient quality normalized by depth (< 2.0)
- **FS: insufficient phred-scaled p-value using Fisher's exact test to detect strand bias (> 30.0)**
- SnpCluster: Variant is part of a cluster

For example, if a variant has read depth 8, GATK will mark it as DP.

property geneset

Get geneset.

property meta: DataFrame

Return samples metadata table as a pandas DataFrame object.

Returns

table of metadata

property qc: DataFrame

Return samples QC table as a pandas DataFrame object.

Returns

table of QC values

property readable_index: Dict[int, str]

Get mapping from index values to readable names.

property variants: DataFrame

Get variants table.

There are 4 possible values:

- 0 - wild-type, no variant
- 1 - heterozygous mutation
- 2 - homozygous mutation
- NaN - QC filters are failing - mutation status is unreliable

3.4.5 Exceptions

Custom ReSDK exceptions.

class resdk.exceptions.ValidationError

An error while validating data.

3.4.6 Logging

Module contents:

1. Parent logger for all modules in resdk library
2. Handler STDOUT_HANDLER is “turned off” by default
3. Handler configuration functions
4. Override sys.excepthook to log all uncaught exceptions

Parent logger

Loggers in resdk are named by their module name. This is achieved by:

```
logger = logging.getLogger(__name__)
```

This makes it easy to locate the source of a log message.

Logging handlers

The handler STDOUT_HANDLER is created but not automatically added to ROOT_LOGGER, which means they do not do anything. The handlers are activated when users call logger configuration functions like `start_logging()`.

Handler configuration functions

As a good logging practice, the library does not register handlers by default. The reason is that if the library is included in some application, developers of that application will probably want to register loggers by themselves. Therefore, if a user wishes to register the pre-defined handlers she can run:

```
import resdk
resdk.start_logging()
```


`resdk_logger.start_logging(logging_level=logging.INFO)`

Start logging resdk with the default configuration.

Parameters

logging_level (*int*) – logging threshold level - integer in [0-50]

Return type

None

Logging levels:

- `logging.DEBUG(10)`
- `logging.INFO(20)`
- `logging.WARNING(30)`
- `logging.ERROR(40)`
- `logging.CRITICAL(50)`

`resdk_logger.log_to_stdout(level=None)`

Configure logging to stdout.

Parameters

- **is_on** (*bool*) – If True, log to standard output
- **level** (*int*) – logging threshold level - integer in [0-50]

Return type

None

Log uncaught exceptions

All python exceptions are handled by function, stored in `sys.excepthook`. By rewriting the default implementation, we can modify it for our purposes - to log all uncaught exceptions.

Note#1: Modified behaviour (logging of all uncaught exceptions) applies only when running in non-interactive mode.

Note#2: Any exception can be caught/uncaught and it can happen in interactive/non-interactive mode. This makes 4 different scenarios. The `sys.excepthook` modification takes care of uncaught exceptions in non-interactive mode. In interactive mode, user is notified directly if exception is raised. If exception is caught and not re-raised, it should be logged somehow, since it can provide valuable information for developer when debugging. Therefore, we should use the following convention for logging in resdk: “Exceptions are explicitly logged only when they are caught and not re-raised.”

3.5 Contributing

3.5.1 Installing prerequisites

Make sure you have [Python 3.7+](#) installed on your system. If you don't have it yet, follow [these instructions](#).

3.5.2 Preparing environment

Fork the main Resolwe SDK for Python git repository.

If you don't have Git installed on your system, follow [these instructions](#).

Clone your fork (replace <username> with your GitHub account name) and change directory:

```
git clone https://github.com/<username>/resolwe-bio-py.git
cd resolwe-bio-py
```

Prepare Resolwe SDK for Python for development:

```
pip install -e .[docs,package,test]
```

Note: We recommend using [venv](#) to create an isolated Python environment.

3.5.3 Running tests

Run unit tests:

```
py.test
```

3.5.4 Coverage report

To see the tests' code coverage, use:

```
py.test --cov=resdk
```

To generate an HTML file showing the tests' code coverage, use:

```
py.test --cov=resdk --cov-report=html
```

3.5.5 Building documentation

```
python setup.py build_sphinx
```

3.5.6 Preparing release

Checkout the latest code and create a release branch:

```
git checkout master
git pull
git checkout -b release-<new-version>
```

Replace the *Unreleased* heading in docs/CHANGELOG.rst with the new version, followed by release's date (e.g. *13.2.0 - 2018-10-23*).

Commit changes to git:

```
git commit -a -m "Prepare release <new-version>"
```

Push changes to your fork and open a pull request:

```
git push --set-upstream <resdk-fork-name> release-<new-version>
```

Wait for the tests to pass and the pull request to be approved. Merge the code to master:

```
git checkout master
git merge --ff-only release-<new-version>
git push <resdk-upstream-name> master <new-version>
```

Tag the new release from the latest commit:

```
git checkout master
git tag -sm "Version <new-version>" <new-version>
```

Push the tag to the main ReSDK's git repository:

```
git push <resdk-upstream-name> master <new-version>
```

The tagged code will be released to PyPI automatically.

PYTHON MODULE INDEX

r

- `resdk.exceptions`, 68
- `resdk.query`, 29
- `resdk.resdk_logger`, 68
- `resdk.resolwe`, 28
- `resdk.resources`, 30
- `resdk.resources.kb`, 57
- `resdk.resources.utils`, 61
- `resdk.tables`, 62

Symbols

`__init__()` (*resdk.tables.methylation.MethylationTables* method), 65
`__init__()` (*resdk.tables.microarray.MATables* method), 62
`__init__()` (*resdk.tables.ml_ready.MLTables* method), 63
`__init__()` (*resdk.tables.rna.RNATables* method), 64
`__init__()` (*resdk.tables.variant.VariantTables* method), 66

A

`add_sample()` (*resdk.resources.Relation* method), 43
`add_users()` (*resdk.resources.Group* method), 49
aliases (*resdk.resources.kb.Feature* attribute), 57
`all()` (*resdk.ResolveQuery* method), 29
AnnotationField (class in *resdk.resources*), 48
AnnotationGroup (class in *resdk.resources*), 48
annotations (*resdk.resources.Sample* property), 40
AnnotationValue (class in *resdk.resources*), 47
`assign_to_billing_account()`
 (*resdk.resources.Collection* method), 37

B

background (*resdk.resources.Sample* property), 40
BaseCollection (class in *resdk.resources.collection*), 35
BaseResolveResource (class in *resdk.resources.base*), 31
BaseResource (class in *resdk.resources.base*), 30
beta (*resdk.tables.methylation.MethylationTables* property), 65
build (*resdk.tables.rna.RNATables* property), 64

C

category (*resdk.resources.Process* attribute), 44
category (*resdk.resources.Relation* attribute), 43
`check_heterogeneous_collections()`
 (*resdk.tables.rna.RNATables* method), 64
checksum (*resdk.resources.Data* attribute), 32
checksum (*resdk.resources.Geneset* attribute), 50
checksum (*resdk.resources.Metadata* attribute), 53

children (*resdk.resources.Data* property), 32
children (*resdk.resources.Geneset* property), 50
children (*resdk.resources.Metadata* property), 54
`clear_cache()` (*resdk.ResolveQuery* method), 29
`clear_cache()` (*resdk.resources.permissions.PermissionsManager* method), 59
`clear_cache()` (*resdk.tables.methylation.MethylationTables* static method), 66
`clear_cache()` (*resdk.tables.microarray.MATables* static method), 63
`clear_cache()` (*resdk.tables.rna.RNATables* static method), 64
`clear_cache()` (*resdk.tables.variant.VariantTables* static method), 67
Collection (class in *resdk.resources*), 37
collection (*resdk.resources.Data* property), 32
collection (*resdk.resources.Geneset* property), 50
collection (*resdk.resources.Metadata* property), 54
collection (*resdk.resources.Relation* property), 43
collection (*resdk.resources.Sample* property), 40
contributor (*resdk.resources.base.BaseResolveResource* property), 31
contributor (*resdk.resources.Collection* property), 37
contributor (*resdk.resources.collection.BaseCollection* property), 35
contributor (*resdk.resources.Data* property), 32
contributor (*resdk.resources.DescriptorSchema* property), 46
contributor (*resdk.resources.Geneset* property), 50
contributor (*resdk.resources.Metadata* property), 54
contributor (*resdk.resources.Process* property), 44
contributor (*resdk.resources.Relation* property), 43
contributor (*resdk.resources.Sample* property), 40
`copy_from()` (*resdk.resources.permissions.PermissionsManager* method), 59
`count()` (*resdk.ResolveQuery* method), 30
`create()` (*resdk.ResolveQuery* method), 30
`create_background_relation()`
 (*resdk.resources.Collection* method), 37
`create_compare_relation()`
 (*resdk.resources.Collection* method), 37
`create_group_relation()`

(resdk.resources.Collection method), 37
 create_series_relation()
 (resdk.resources.Collection method), 38
 created *(resdk.resources.base.BaseResolweResource property)*, 31
 created *(resdk.resources.Collection property)*, 38
 created *(resdk.resources.collection.BaseCollection property)*, 35
 created *(resdk.resources.Data property)*, 32
 created *(resdk.resources.DescriptorSchema property)*, 46
 created *(resdk.resources.Geneset property)*, 50
 created *(resdk.resources.Metadata property)*, 54
 created *(resdk.resources.Process property)*, 44
 created *(resdk.resources.Relation property)*, 43
 created *(resdk.resources.Sample property)*, 40
 current_user_permissions
 (resdk.resources.base.BaseResolweResource attribute), 31
 current_user_permissions
 (resdk.resources.Collection attribute), 38
 current_user_permissions
 (resdk.resources.collection.BaseCollection attribute), 35
 current_user_permissions *(resdk.resources.Data attribute)*, 32
 current_user_permissions
 (resdk.resources.DescriptorSchema attribute), 46
 current_user_permissions *(resdk.resources.Geneset attribute)*, 50
 current_user_permissions
 (resdk.resources.Metadata attribute), 54
 current_user_permissions *(resdk.resources.Process attribute)*, 44
 current_user_permissions *(resdk.resources.Relation attribute)*, 43
 current_user_permissions *(resdk.resources.Sample attribute)*, 40

D

Data *(class in resdk.resources)*, 32
 data *(resdk.resources.Collection property)*, 38
 data *(resdk.resources.collection.BaseCollection property)*, 35
 data *(resdk.resources.Sample property)*, 40
 data_name *(resdk.resources.Process attribute)*, 44
 data_types() *(resdk.resources.Collection method)*, 38
 data_types() *(resdk.resources.collection.BaseCollection method)*, 35
 data_types() *(resdk.resources.Sample method)*, 40
 data_usage() *(resdk.Resolwe method)*, 28
 delete() *(resdk.ResolweQuery method)*, 30
 delete() *(resdk.resources.AnnotationField method)*, 48

delete() *(resdk.resources.AnnotationGroup method)*, 48
 delete() *(resdk.resources.AnnotationValue method)*, 47
 delete() *(resdk.resources.base.BaseResolweResource method)*, 31
 delete() *(resdk.resources.base.BaseResource method)*, 31
 delete() *(resdk.resources.Collection method)*, 38
 delete() *(resdk.resources.collection.BaseCollection method)*, 35
 delete() *(resdk.resources.Data method)*, 32
 delete() *(resdk.resources.DescriptorSchema method)*, 46
 delete() *(resdk.resources.Geneset method)*, 50
 delete() *(resdk.resources.Group method)*, 49
 delete() *(resdk.resources.kb.Feature method)*, 57
 delete() *(resdk.resources.kb.Mapping method)*, 58
 delete() *(resdk.resources.Metadata method)*, 54
 delete() *(resdk.resources.Process method)*, 45
 delete() *(resdk.resources.Relation method)*, 43
 delete() *(resdk.resources.Sample method)*, 40
 delete() *(resdk.resources.User method)*, 49
 depth *(resdk.tables.variant.VariantTables property)*, 67
 depth_a *(resdk.tables.variant.VariantTables property)*, 67
 depth_c *(resdk.tables.variant.VariantTables property)*, 67
 depth_g *(resdk.tables.variant.VariantTables property)*, 67
 depth_t *(resdk.tables.variant.VariantTables property)*, 67
 description *(resdk.resources.Collection attribute)*, 38
 description *(resdk.resources.collection.BaseCollection attribute)*, 36
 description *(resdk.resources.DescriptorSchema attribute)*, 46
 description *(resdk.resources.kb.Feature attribute)*, 57
 description *(resdk.resources.Process attribute)*, 45
 description *(resdk.resources.Sample attribute)*, 40
 descriptor *(resdk.resources.Collection attribute)*, 38
 descriptor *(resdk.resources.collection.BaseCollection attribute)*, 36
 descriptor *(resdk.resources.Data attribute)*, 32
 descriptor *(resdk.resources.Geneset attribute)*, 50
 descriptor *(resdk.resources.Metadata attribute)*, 54
 descriptor *(resdk.resources.Relation attribute)*, 43
 descriptor *(resdk.resources.Sample attribute)*, 40
 descriptor_dirty *(resdk.resources.Collection attribute)*, 38
 descriptor_dirty *(resdk.resources.collection.BaseCollection attribute)*, 36
 descriptor_dirty *(resdk.resources.Data attribute)*, 32
 descriptor_dirty *(resdk.resources.Geneset attribute)*, 50

- descriptor_dirty (*resdk.resources.Metadata* attribute), 54
 descriptor_dirty (*resdk.resources.Relation* attribute), 43
 descriptor_dirty (*resdk.resources.Sample* attribute), 41
 descriptor_schema (*resdk.resources.Collection* property), 38
 descriptor_schema (*resdk.resources.collection.BaseCollection* class method), 48
 descriptor_schema (*resdk.resources.Data* property), 32
 descriptor_schema (*resdk.resources.Geneset* property), 51
 descriptor_schema (*resdk.resources.Metadata* property), 54
 descriptor_schema (*resdk.resources.Relation* property), 43
 descriptor_schema (*resdk.resources.Sample* property), 41
 DescriptorSchema (class in *resdk.resources*), 46
 df (*resdk.resources.Metadata* property), 54
 df_bytes (*resdk.resources.Metadata* property), 54
 download() (*resdk.resources.Collection* method), 38
 download() (*resdk.resources.collection.BaseCollection* method), 36
 download() (*resdk.resources.Data* method), 33
 download() (*resdk.resources.Geneset* method), 51
 download() (*resdk.resources.Metadata* method), 54
 download() (*resdk.resources.Sample* method), 41
 duplicate() (*resdk.resources.Collection* method), 39
 duplicate() (*resdk.resources.Data* method), 33
 duplicate() (*resdk.resources.Geneset* method), 51
 duplicate() (*resdk.resources.Metadata* method), 55
 duplicate() (*resdk.resources.Sample* method), 41
 duplicated (*resdk.resources.Collection* attribute), 39
 duplicated (*resdk.resources.collection.BaseCollection* attribute), 36
 duplicated (*resdk.resources.Data* attribute), 33
 duplicated (*resdk.resources.Geneset* attribute), 51
 duplicated (*resdk.resources.Metadata* attribute), 55
 duplicated (*resdk.resources.Sample* attribute), 41
- ## E
- editors (*resdk.resources.permissions.PermissionsManager* property), 59
 entity_always_create (*resdk.resources.Process* attribute), 45
 entity_descriptor_schema (*resdk.resources.Process* attribute), 45
 entity_input (*resdk.resources.Process* attribute), 45
 entity_type (*resdk.resources.Process* attribute), 45
 exp (*resdk.tables.microarray.MATables* property), 63
 exp (*resdk.tables.ml_ready.MLTables* property), 63
 exp (*resdk.tables.rna.RNATables* property), 64
- ## F
- Feature (class in *resdk.resources.kb*), 57
 feature_id (*resdk.resources.kb.Feature* attribute), 57
 fetch() (*resdk.resources.permissions.PermissionsManager* method), 59
 fetch_object() (*resdk.resources.AnnotationField* class method), 48
 fetch_object() (*resdk.resources.AnnotationGroup* class method), 48
 fetch_object() (*resdk.resources.AnnotationValue* class method), 47
 fetch_object() (*resdk.resources.base.BaseResolweResource* class method), 31
 fetch_object() (*resdk.resources.base.BaseResource* class method), 31
 fetch_object() (*resdk.resources.Collection* class method), 39
 fetch_object() (*resdk.resources.collection.BaseCollection* class method), 36
 fetch_object() (*resdk.resources.Data* class method), 33
 fetch_object() (*resdk.resources.DescriptorSchema* class method), 46
 fetch_object() (*resdk.resources.Geneset* class method), 51
 fetch_object() (*resdk.resources.Group* class method), 49
 fetch_object() (*resdk.resources.kb.Feature* class method), 57
 fetch_object() (*resdk.resources.kb.Mapping* class method), 58
 fetch_object() (*resdk.resources.Metadata* class method), 55
 fetch_object() (*resdk.resources.Process* class method), 45
 fetch_object() (*resdk.resources.Relation* class method), 43
 fetch_object() (*resdk.resources.Sample* class method), 41
 fetch_object() (*resdk.resources.User* class method), 49
 field (*resdk.resources.AnnotationValue* property), 47
 fields() (*resdk.resources.AnnotationField* method), 48
 fields() (*resdk.resources.AnnotationGroup* method), 48
 fields() (*resdk.resources.AnnotationValue* method), 47
 fields() (*resdk.resources.base.BaseResolweResource* method), 31
 fields() (*resdk.resources.base.BaseResource* method), 31
 fields() (*resdk.resources.Collection* method), 39

fields() (*resdk.resources.collection.BaseCollection method*), 36

fields() (*resdk.resources.Data method*), 33

fields() (*resdk.resources.DescriptorSchema method*), 47

fields() (*resdk.resources.Geneset method*), 51

fields() (*resdk.resources.Group method*), 50

fields() (*resdk.resources.kb.Feature method*), 57

fields() (*resdk.resources.kb.Mapping method*), 58

fields() (*resdk.resources.Metadata method*), 55

fields() (*resdk.resources.Process method*), 45

fields() (*resdk.resources.Relation method*), 43

fields() (*resdk.resources.Sample method*), 41

fields() (*resdk.resources.User method*), 49

files() (*resdk.resources.Collection method*), 39

files() (*resdk.resources.collection.BaseCollection method*), 36

files() (*resdk.resources.Data method*), 33

files() (*resdk.resources.Geneset method*), 51

files() (*resdk.resources.Metadata method*), 55

files() (*resdk.resources.Sample method*), 41

fill_spaces() (*in module resdk.resources.utils*), 61

filter (*resdk.tables.variant.VariantTables property*), 67

filter() (*resdk.ResolveQuery method*), 30

finished (*resdk.resources.Data property*), 33

finished (*resdk.resources.Geneset property*), 51

finished (*resdk.resources.Metadata property*), 55

first_name (*resdk.resources.User attribute*), 49

flatten_field() (*in module resdk.resources.utils*), 61

full_name (*resdk.resources.kb.Feature attribute*), 57

G

genes (*resdk.resources.Geneset property*), 51

Geneset (*class in resdk.resources*), 50

geneset (*resdk.tables.variant.VariantTables property*), 67

get() (*resdk.ResolveQuery method*), 30

get_annotation() (*resdk.resources.Sample method*), 41

get_annotations() (*resdk.resources.Sample method*), 41

get_bam() (*resdk.resources.Sample method*), 41

get_collection_id() (*in module resdk.resources.utils*), 61

get_cuffquant() (*resdk.resources.Sample method*), 41

get_data_id() (*in module resdk.resources.utils*), 61

get_descriptor_schema_id() (*in module resdk.resources.utils*), 61

get_df() (*resdk.resources.Metadata method*), 55

get_expression() (*resdk.resources.Sample method*), 42

get_macsc() (*resdk.resources.Sample method*), 42

get_name() (*resdk.resources.User method*), 49

get_or_run() (*resdk.ResolveQuery method*), 28

get_primary_bam() (*resdk.resources.Sample method*), 42

get_process_id() (*in module resdk.resources.utils*), 61

get_query_by_resource() (*resdk.ResolveQuery method*), 28

get_reads() (*resdk.resources.Sample method*), 42

get_relation_id() (*in module resdk.resources.utils*), 61

get_sample_id() (*in module resdk.resources.utils*), 61

get_user_id() (*in module resdk.resources.utils*), 61

Group (*class in resdk.resources*), 49

group (*resdk.resources.AnnotationField property*), 48

I

id (*resdk.resources.AnnotationField attribute*), 48

id (*resdk.resources.AnnotationGroup attribute*), 48

id (*resdk.resources.AnnotationValue attribute*), 47

id (*resdk.resources.base.BaseResolveResource attribute*), 31

id (*resdk.resources.base.BaseResource attribute*), 31

id (*resdk.resources.Collection attribute*), 39

id (*resdk.resources.collection.BaseCollection attribute*), 36

id (*resdk.resources.Data attribute*), 33

id (*resdk.resources.DescriptorSchema attribute*), 47

id (*resdk.resources.Geneset attribute*), 52

id (*resdk.resources.Group attribute*), 50

id (*resdk.resources.kb.Feature attribute*), 58

id (*resdk.resources.kb.Mapping attribute*), 58

id (*resdk.resources.Metadata attribute*), 55

id (*resdk.resources.Process attribute*), 45

id (*resdk.resources.Relation attribute*), 43

id (*resdk.resources.Sample attribute*), 42

id (*resdk.resources.User attribute*), 49

input (*resdk.resources.Data attribute*), 33

input (*resdk.resources.Geneset attribute*), 52

input (*resdk.resources.Metadata attribute*), 55

input_schema (*resdk.resources.Process attribute*), 45

is_active (*resdk.resources.Process attribute*), 45

is_background (*resdk.resources.Sample property*), 42

is_collection() (*in module resdk.resources.utils*), 61

is_data() (*in module resdk.resources.utils*), 61

is_descriptor_schema() (*in module resdk.resources.utils*), 61

is_group() (*in module resdk.resources.utils*), 61

is_process() (*in module resdk.resources.utils*), 61

is_relation() (*in module resdk.resources.utils*), 61

is_sample() (*in module resdk.resources.utils*), 62

is_user() (*in module resdk.resources.utils*), 62

iterate() (*resdk.ResolveQuery method*), 30

iterate_fields() (*in module resdk.resources.utils*), 62

iterate_schema() (*in module resdk.resources.utils*), 62

L

log_to_stdout() (*resdk.resdk_logger* method), 69
 login() (*resdk.Resolwe* method), 28

M

Mapping (class in *resdk.resources.kb*), 58
 MATables (class in *resdk.tables.microarray*), 62
 meta (*resdk.tables.methylation.MethylationTables* property), 66
 meta (*resdk.tables.microarray.MATables* property), 63
 meta (*resdk.tables.rna.RNATables* property), 64
 meta (*resdk.tables.variant.VariantTables* property), 67
 Metadata (class in *resdk.resources*), 53
 MethylationTables (class in *resdk.tables.methylation*), 65
 MLTables (class in *resdk.tables.ml_ready*), 63
 modified (*resdk.resources.base.BaseResolweResource* property), 31
 modified (*resdk.resources.Collection* property), 39
 modified (*resdk.resources.collection.BaseCollection* property), 36
 modified (*resdk.resources.Data* property), 34
 modified (*resdk.resources.DescriptorSchema* property), 47
 modified (*resdk.resources.Geneset* property), 52
 modified (*resdk.resources.Metadata* property), 55
 modified (*resdk.resources.Process* property), 45
 modified (*resdk.resources.Relation* property), 43
 modified (*resdk.resources.Sample* property), 42
 module
 resdk.exceptions, 68
 resdk.query, 29
 resdk.resdk_logger, 68
 resdk.resolwe, 28
 resdk.resources, 30
 resdk.resources.kb, 57
 resdk.resources.utils, 61
 resdk.tables, 62
 mval (*resdk.tables.methylation.MethylationTables* property), 66

N

name (*resdk.resources.base.BaseResolweResource* attribute), 32
 name (*resdk.resources.Collection* attribute), 39
 name (*resdk.resources.collection.BaseCollection* attribute), 36
 name (*resdk.resources.Data* attribute), 34
 name (*resdk.resources.DescriptorSchema* attribute), 47
 name (*resdk.resources.Geneset* attribute), 52
 name (*resdk.resources.Group* attribute), 50
 name (*resdk.resources.kb.Feature* attribute), 58
 name (*resdk.resources.Metadata* attribute), 55

name (*resdk.resources.Process* attribute), 45
 name (*resdk.resources.Relation* attribute), 44
 name (*resdk.resources.Sample* attribute), 42

O

output (*resdk.resources.Data* attribute), 34
 output (*resdk.resources.Geneset* attribute), 52
 output (*resdk.resources.Metadata* attribute), 55
 output_schema (*resdk.resources.Process* attribute), 45
 owners (*resdk.resources.permissions.PermissionsManager* property), 59

P

parents (*resdk.resources.Data* property), 34
 parents (*resdk.resources.Geneset* property), 52
 parents (*resdk.resources.Metadata* property), 55
 parse_resolwe_datetime() (in module *resdk.resources.utils*), 62
 partitions (*resdk.resources.Relation* attribute), 44
 permissions (*resdk.resources.base.BaseResolweResource* property), 32
 permissions (*resdk.resources.Collection* property), 39
 permissions (*resdk.resources.collection.BaseCollection* property), 37
 permissions (*resdk.resources.Data* property), 34
 permissions (*resdk.resources.DescriptorSchema* property), 47
 permissions (*resdk.resources.Geneset* property), 52
 permissions (*resdk.resources.Metadata* property), 55
 permissions (*resdk.resources.Process* property), 45
 permissions (*resdk.resources.Relation* property), 44
 permissions (*resdk.resources.Sample* property), 42
 PermissionsManager (class in *resdk.resources.permissions*), 59
 persistence (*resdk.resources.Process* attribute), 45
 print_inputs() (*resdk.resources.Process* method), 46
 priority (*resdk.resources.Process* attribute), 46
 Process (class in *resdk.resources*), 44
 process (*resdk.resources.Data* property), 34
 process (*resdk.resources.Geneset* property), 52
 process (*resdk.resources.Metadata* property), 55
 process_cores (*resdk.resources.Data* attribute), 34
 process_cores (*resdk.resources.Geneset* attribute), 52
 process_cores (*resdk.resources.Metadata* attribute), 55
 process_error (*resdk.resources.Data* attribute), 34
 process_error (*resdk.resources.Geneset* attribute), 52
 process_error (*resdk.resources.Metadata* attribute), 56
 process_info (*resdk.resources.Data* attribute), 34
 process_info (*resdk.resources.Geneset* attribute), 52
 process_info (*resdk.resources.Metadata* attribute), 56
 process_memory (*resdk.resources.Data* attribute), 34
 process_memory (*resdk.resources.Geneset* attribute), 52

process_memory (*resdk.resources.Metadata attribute*), 56
 process_progress (*resdk.resources.Data attribute*), 34
 process_progress (*resdk.resources.Geneset attribute*), 52
 process_progress (*resdk.resources.Metadata attribute*), 56
 process_rc (*resdk.resources.Data attribute*), 34
 process_rc (*resdk.resources.Geneset attribute*), 52
 process_rc (*resdk.resources.Metadata attribute*), 56
 process_resources (*resdk.resources.Data attribute*), 34
 process_resources (*resdk.resources.Geneset attribute*), 52
 process_resources (*resdk.resources.Metadata attribute*), 56
 process_warning (*resdk.resources.Data attribute*), 34
 process_warning (*resdk.resources.Geneset attribute*), 52
 process_warning (*resdk.resources.Metadata attribute*), 56

Q

qc (*resdk.tables.methylation.MethylationTables property*), 66
 qc (*resdk.tables.microarray.MATables property*), 63
 qc (*resdk.tables.rna.RNATables property*), 64
 qc (*resdk.tables.variant.VariantTables property*), 67

R

rc (*resdk.tables.rna.RNATables property*), 65
 readable_columns (*resdk.tables.rna.RNATables property*), 65
 readable_index (*resdk.tables.methylation.MethylationTables property*), 66
 readable_index (*resdk.tables.microarray.MATables property*), 63
 readable_index (*resdk.tables.rna.RNATables property*), 65
 readable_index (*resdk.tables.variant.VariantTables property*), 67
 Relation (*class in resdk.resources*), 43
 relations (*resdk.resources.Collection property*), 39
 relations (*resdk.resources.Sample property*), 42
 remove_samples() (*resdk.resources.Relation method*), 44
 remove_users() (*resdk.resources.Group method*), 50
 requirements (*resdk.resources.Process attribute*), 46
 resdk.exceptions
 module, 68
 resdk.query
 module, 29
 resdk.resdk_logger
 module, 68
 resdk.resolwe
 module, 28
 resdk.resources
 module, 30
 resdk.resources.kb
 module, 57
 resdk.resources.utils
 module, 61
 resdk.tables
 module, 62
 Resolwe (*class in resdk*), 28
 ResolweQuery (*class in resdk*), 29
 RNATables (*class in resdk.tables.rna*), 63
 run (*resdk.resources.Process attribute*), 46
 run() (*resdk.Resolwe method*), 28

S

Sample (*class in resdk.resources*), 40
 sample (*resdk.resources.AnnotationValue property*), 47
 sample (*resdk.resources.Data property*), 34
 sample (*resdk.resources.Geneset property*), 52
 sample (*resdk.resources.Metadata property*), 56
 sample_id (*resdk.resources.AnnotationValue attribute*), 47
 samples (*resdk.resources.Collection property*), 39
 samples (*resdk.resources.Relation property*), 44
 save() (*resdk.resources.AnnotationField method*), 48
 save() (*resdk.resources.AnnotationGroup method*), 48
 save() (*resdk.resources.AnnotationValue method*), 48
 save() (*resdk.resources.base.BaseResolweResource method*), 32
 save() (*resdk.resources.base.BaseResource method*), 31
 save() (*resdk.resources.Collection method*), 39
 save() (*resdk.resources.collection.BaseCollection method*), 37
 save() (*resdk.resources.Data method*), 34
 save() (*resdk.resources.DescriptorSchema method*), 47
 save() (*resdk.resources.Geneset method*), 52
 save() (*resdk.resources.Group method*), 50
 save() (*resdk.resources.kb.Feature method*), 58
 save() (*resdk.resources.kb.Mapping method*), 58
 save() (*resdk.resources.Metadata method*), 56
 save() (*resdk.resources.Process method*), 46
 save() (*resdk.resources.Relation method*), 44
 save() (*resdk.resources.Sample method*), 42
 save() (*resdk.resources.User method*), 49
 scheduled (*resdk.resources.Data attribute*), 34
 scheduled (*resdk.resources.Geneset attribute*), 52
 scheduled (*resdk.resources.Metadata attribute*), 56
 scheduling_class (*resdk.resources.Process attribute*), 46
 schema (*resdk.resources.DescriptorSchema attribute*), 47
 search() (*resdk.ResolweQuery method*), 30

- set_annotation() (*resdk.resources.Sample* method), 42
 set_annotations() (*resdk.resources.Sample* method), 42
 set_df() (*resdk.resources.Metadata* method), 56
 set_group() (*resdk.resources.permissions.PermissionsManager* method), 59
 set_index() (*resdk.resources.Metadata* method), 56
 set_operator() (*resdk.resources.Geneset* method), 53
 set_public() (*resdk.resources.permissions.PermissionsManager* method), 60
 set_user() (*resdk.resources.permissions.PermissionsManager* method), 60
 settings (*resdk.resources.Collection* attribute), 39
 settings (*resdk.resources.collection.BaseCollection* attribute), 37
 settings (*resdk.resources.Sample* attribute), 42
 size (*resdk.resources.Data* attribute), 34
 size (*resdk.resources.Geneset* attribute), 53
 size (*resdk.resources.Metadata* attribute), 56
 slug (*resdk.resources.base.BaseResolweResource* attribute), 32
 slug (*resdk.resources.Collection* attribute), 40
 slug (*resdk.resources.collection.BaseCollection* attribute), 37
 slug (*resdk.resources.Data* attribute), 34
 slug (*resdk.resources.DescriptorSchema* attribute), 47
 slug (*resdk.resources.Geneset* attribute), 53
 slug (*resdk.resources.Metadata* attribute), 56
 slug (*resdk.resources.Process* attribute), 46
 slug (*resdk.resources.Relation* attribute), 44
 slug (*resdk.resources.Sample* attribute), 42
 source (*resdk.resources.Geneset* property), 53
 source (*resdk.resources.kb.Feature* attribute), 58
 source_db (*resdk.resources.kb.Mapping* attribute), 58
 source_id (*resdk.resources.kb.Mapping* attribute), 58
 source_species (*resdk.resources.kb.Mapping* attribute), 58
 species (*resdk.resources.Geneset* property), 53
 species (*resdk.resources.kb.Feature* attribute), 58
 start_logging() (*resdk.resdk_logger* method), 68
 started (*resdk.resources.Data* property), 35
 started (*resdk.resources.Geneset* property), 53
 started (*resdk.resources.Metadata* property), 56
 status (*resdk.resources.Data* attribute), 35
 status (*resdk.resources.Geneset* attribute), 53
 status (*resdk.resources.Metadata* attribute), 56
 stdout() (*resdk.resources.Data* method), 35
 stdout() (*resdk.resources.Geneset* method), 53
 stdout() (*resdk.resources.Metadata* method), 56
 sub_type (*resdk.resources.kb.Feature* attribute), 58
- T**
- tags (*resdk.resources.Collection* attribute), 40
 tags (*resdk.resources.collection.BaseCollection* attribute), 37
 tags (*resdk.resources.Data* attribute), 35
 tags (*resdk.resources.Geneset* attribute), 53
 tags (*resdk.resources.Metadata* attribute), 57
 tags (*resdk.resources.Sample* attribute), 42
 target_db (*resdk.resources.kb.Mapping* attribute), 58
 target_id (*resdk.resources.kb.Mapping* attribute), 59
 target_species (*resdk.resources.kb.Mapping* attribute), 59
 type (*resdk.resources.kb.Feature* attribute), 58
 type (*resdk.resources.Process* attribute), 46
 type (*resdk.resources.Relation* attribute), 44
- U**
- unique (*resdk.resources.Metadata* property), 57
 unit (*resdk.resources.Relation* attribute), 44
 update() (*resdk.resources.AnnotationField* method), 48
 update() (*resdk.resources.AnnotationGroup* method), 48
 update() (*resdk.resources.AnnotationValue* method), 48
 update() (*resdk.resources.base.BaseResolweResource* method), 32
 update() (*resdk.resources.base.BaseResource* method), 31
 update() (*resdk.resources.Collection* method), 40
 update() (*resdk.resources.collection.BaseCollection* method), 37
 update() (*resdk.resources.Data* method), 35
 update() (*resdk.resources.DescriptorSchema* method), 47
 update() (*resdk.resources.Geneset* method), 53
 update() (*resdk.resources.Group* method), 50
 update() (*resdk.resources.kb.Feature* method), 58
 update() (*resdk.resources.kb.Mapping* method), 59
 update() (*resdk.resources.Metadata* method), 57
 update() (*resdk.resources.Process* method), 46
 update() (*resdk.resources.Relation* method), 44
 update() (*resdk.resources.Sample* method), 43
 update() (*resdk.resources.User* method), 49
 User (*class in resdk.resources*), 49
 users (*resdk.resources.Group* property), 50
- V**
- validate_df() (*resdk.resources.Metadata* method), 57
 ValidationError (*class in resdk.exceptions*), 68
 variants (*resdk.tables.variant.VariantTables* property), 67
 VariantTables (*class in resdk.tables.variant*), 66
 version (*resdk.resources.base.BaseResolweResource* attribute), 32
 version (*resdk.resources.Collection* attribute), 40
 version (*resdk.resources.collection.BaseCollection* attribute), 37

`version` (*resdk.resources.Data* attribute), 35
`version` (*resdk.resources.DescriptorSchema* attribute),
47
`version` (*resdk.resources.Geneset* attribute), 53
`version` (*resdk.resources.Metadata* attribute), 57
`version` (*resdk.resources.Process* attribute), 46
`version` (*resdk.resources.Relation* attribute), 44
`version` (*resdk.resources.Sample* attribute), 43
`version_check()` (*resdk.Resolwe* method), 29
`viewers` (*resdk.resources.permissions.PermissionsManager*
property), 60