django$_{analyses}$

Release 0.0.3

Feb 16, 2023

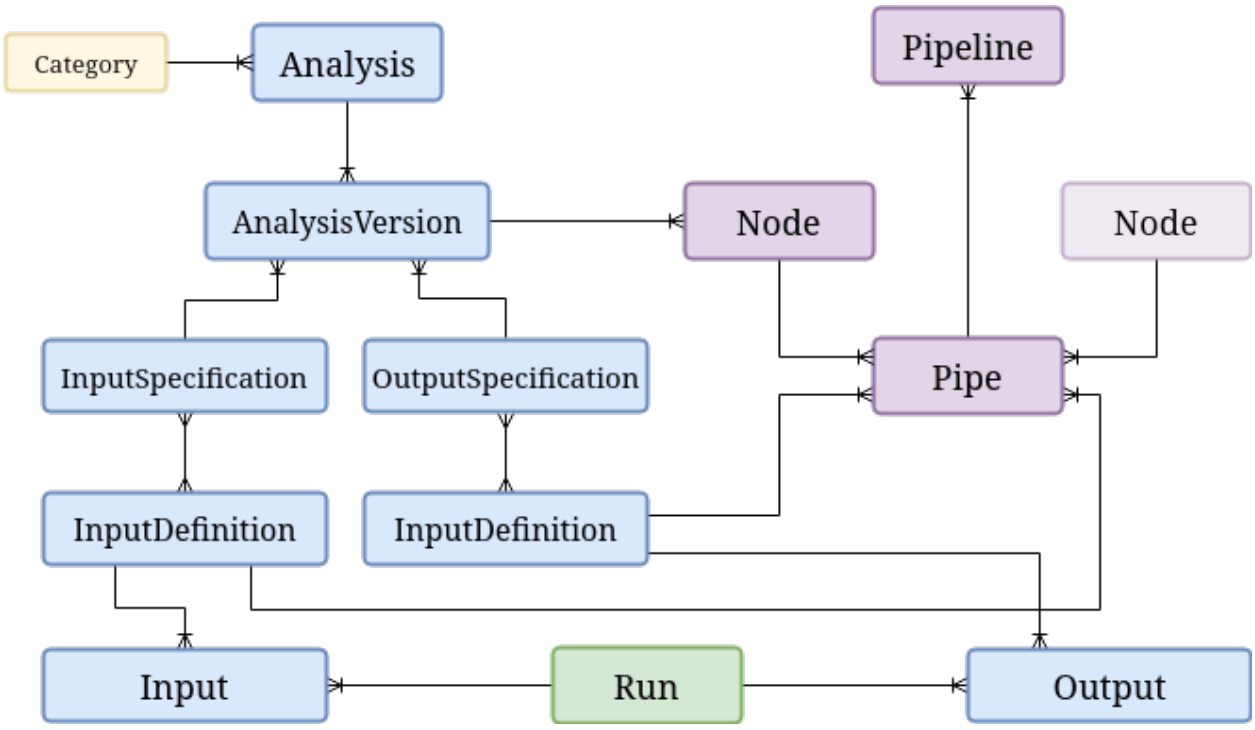
Contents:

CHAPTER 1

Overview

django_analyses provides a database-supported pipeline engine meant to facilitate research management.

A general schema for pipeline management is laid out as follows:



1.1 Analyses

Each *Analysis* may be associated with a number of *AnalysisVersion* instances, and each of those must be provided with an interface, i.e. a Python class exposing some `run()` method and returning a dictionary of results.

For more information, see the *Simplified Analysis Integration Example*.

1.1.1 Input and Output Specifications

`InputSpecification` and `OutputSpecification` simply aggregate a number of `InputDefinition` and `OutputDefinition` sub-classes (respectively) associated with some analysis.

1.1.2 Input and Output Definitions

Currently, there are seven different types of built-in input definitions:

- `BooleanInputDefinition`
- `DirectoryInputDefinition`
- `FileInputDefinition`
- `FloatInputDefinition`
- `IntegerInputDefinition`
- `ListInputDefinition`
- `StringInputDefinition`

and two different kinds of supported output definitions:

- `FileOutputDefinition`
- `FloatOutputDefinition`

Each one of these `InputDefinition` and `OutputDefinition` sub-classes provides unique validation rules (default, minimal/maximal value or length, choices, etc.), and you can easily create more definitions to suit your own needs.

1.2 Pipelines

`Pipeline` instances are used to reference a particular collection of `Node` and `Pipe` instances.

- A `Node` is defined by specifying a distinct combination of an `AnalysisVersion` instance and a configuration for it.
- A `Pipe` connects between a one node's output definition and another's input definition.

For more information, see *Pipeline Generation*.

1.3 Runs

`Run` instances are used to keep a record of every time an analysis version is run with a distinct set of inputs, and associate that event with the resulting outputs.

Whenever a node is executed, the value assigned to each of the `InputDefinition` model's sub-classes detailed in that interface's `InputSpecification` is committed to the database as the corresponding `Input` model's sub-class instance.

If we ever execute a run with identical parameters, the `RunManager` will simply return the existing run.

CHAPTER 2

Installation

1. Install from PyPi:

```
pip install django_analyses
```

2. Add *"django_analyses"* to your project's `INSTALLED_APPS` setting:

Listing 1: settings.py

```
INSTALLED_APPS = [
    ...,
    "django_analyses",
]
```

3. Include the *analyses* URLconf in your prject *urls.py*:

Listing 2: urls.py

```
urlpatterns = [
    ...,
    path("api/", include("django_analyses.urls", namespace="analyses")),
]
```

4. Run:

```
python manage.py migrate
```

5. Start the development server and visit http://127.0.0.1:8000/admin/.
6. Visit http://127.0.0.1:8000/api/analyses/.

CHAPTER 3

User Guide

3.1 Analysis Integration

3.1.1 Simplified Analysis Integration Example

In this example we will add an `ExponentCalculator` class and integrate it into *django_analyses*.

Interface Creation

By default, interfaces are expected to be classes and expose a `run()` method which returns a dictionary of output keys and values. Therefore, an `ExponentCalculator` class might look something like this:

Listing 1: exponent_calculator.py

```
class ExponentCalculator:
    def __init__(self, base: float, exponent: float):
        self.base = base
        self.exponent = exponent

    def run(self) -> dict:
        return {"result": self.base ** self.exponent}
```

Analysis Creation

The `ExponentCalculator` class is one possible implementation of exponentiation. Let's define this procedure as a new available analysis:

```
>>> from django_analyses.models import Analysis
>>> definition = {
>>>     "title": "Exponentiation",
>>>     "description": "Calculate <base> in the power of <exponent>.",
```

(continues on next page)

(continued from previous page)

```
>>> }
>>> analysis = Analysis.objects.create(**definition)
```

Analysis Version Creation

Now, we can create an *AnalysisVersion* instance to represent the `ExponentCalculator` class we've created:

```
>>> from django_analyses.models import AnalysisVersion
>>> definition = {
>>>     "title": "built-in",
>>>     "description": "Calculate the exponent of a number using Python's built-in␣
↪power operator.",
>>> }
>>> analysis_version = AnalysisVersion.objects.create(**definition)
```

Input Specification

The `ExponentCalculator` class expects two `float` type input values which are assigned in its initialization: `base` and `exponent`.

InputSpecification instances are created with an association to a specific *Analysis* (this prevents name clashes between input or output definitions for different analyses) and may be used for a number of its *AnalysisVersion* instances.

```
>>> from django_analyses.models import FloatInputDefinition, InputSpecification
>>> definition = {
>>>     "base": {
>>>         "type": FloatInputDefinition,
>>>         "required": True,
>>>         "description": "Floating point number to be raised by <exponent>.",
>>>     },
>>>     "exponent": {
>>>         "type": FloatInputDefinition,
>>>         "required": True,
>>>         "description": "Floating point number to raise <base> by.",
>>>     },
>>> }
>>> analysis = Analysis.objects.get(title="Exponentiation")
>>> input_specification, created = InputSpecification.objects.from_dict(analysis,␣
↪definition)
>>> input_specification
<InputSpecification:
[Exponentiation]
    base                                    Float
    exponent                                Float
>
>>> created
True
```

Output Specification

The *OutputSpecification* may be created very similarly:

```
>>> from django_analyses.models import FloatOutputDefinition, OutputSpecification
>>> definition = {
>>>     "result": {
>>>         "type": FloatOutputDefinition,
>>>         "description": "Product of <base> multiplied <exponent> times.",
>>>     }
>>> }
>>> analysis = Analysis.objects.get(title="Exponentiation")
>>> output_specification, created = OutputSpecification.objects.from_dict(analysis,␣
↪definition)
>>> output_specification
<OutputSpecification
[Exponentiation]
    result                                  Float
>
>>> created
True
```

Interface Integration

At this stage our new analysis is ready to be "plugged-in". Interfaces are queried from the `ANALYSIS_INTERFACES` dictionary in the project's *settings.py*. Analyses are expected to be registered as `ANALYSIS_INTERFACES["analysis_title"]["analysis_version_title"]`, so in our case:

Listing 2: settings.py

```
from exponent_calculator import ExponentCalculator

...

ANALYSIS_INTERFACES = {"Exponentiation": {"built-in": ExponentCalculator}}
```

3.1.2 Realistic Analysis Integration Example

In this example, we will integrate a basic version of Nipype's interface for FSL's SUSAN noise-reduction algorithm for MRI images. This example is adapted from django_mri.

Input and Output Specification

`InputSpecificationManager` and *`OutputSpecificationManager`* provide a `from_dict()` method which can generate specifications based on a `dict` with the following structure:

```
SPECIFICATION = {
    "definition_key_1": {
        "type": InputDefinitionSubclass,
        "attribute": "value"
    },
    "definition_key_2": {
        "type": InputDefinitionSubclass,
        "attribute": "value2"
    },
}
```

If we try to recreate SUSAN's specifications from the Nipype's docs according to this structure, it might look something like:

Listing 3: susan_specifications.py

```
from django_analyses.models.input.definitions import (
    BooleanInputDefinition,
    FileInputDefinition,
    FloatInputDefinition,
    IntegerInputDefinition,
    StringInputDefinition,
)

from django_analyses.models.output.definitions import FileOutputDefinition


SUSAN_INPUT_SPECIFICATION = {
    "brightness_threshold": {
        "type": FloatInputDefinition,
        "required": True,
        "description": "Should be greater than noise level and less than contrast of␣
↪edges to be preserved.",
    },
    "fwhm": {
        "type": FloatInputDefinition,
        "required": True,
        "description": "FWHM of smoothing, in millimeters.",
    },
    "in_file": {
        "type": FileInputDefinition,
        "required": True,
        "description": "Filename of input time-series.",
    },
    "dimension": {
        "type": IntegerInputDefinition,
        "required": False,
        "default": 3,
        "min_value": 2,
        "max_value": 3,
    },
    "out_file": {
        "type": StringInputDefinition,
        "required": False,
        "description": "Desired output file path.",
        "is_output_path": True,
        "default": "smooth.nii.gz",
    },
    "output_type": {
        "type": StringInputDefinition,
        "required": False,
        "description": "Output file format.",
        "choices": ["NIFTI", "NIFTI_PAIR", "NIFTI_GZ", "NIFTI_PAIR_GZ"],
        "default": "NIFTI_GZ",
    },
    "use_median": {
        "type": BooleanInputDefinition,
        "required": False,
        "default": True,
```

(continues on next page)

(continued from previous page)

```
        "description": "Whether to use a local median filter in the cases where␣
↪single-point noise is detected.",
    },
    "args": {
        "type": StringInputDefinition,
        "required": False,
        "description": "Additional parameters to pass to the command.",
    },
}


SUSAN_OUTPUT_SPECIFICATION = {
    "smoothed_file": {
        "type": FileOutputDefinition,
        "description": "Smoothed output file.",
    }
}
```

Analysis Definition

Similarly to the input and output specifications, the *AnalysisManager* class exposes a *from_list()* method which we could use to easily add analyses to the database.

First we'll create the complete definition in another file:

Listing 4: analysis_definitions.py

```
from susan_specifications import SUSAN_INPUT_SPECIFICATION, SUSAN_OUTPUT_SPECIFICATION

analysis_definitions = [
    {
        "title": "SUSAN",
        "description": "Reduces noise in 2/3D images by averaging voxels with similar␣
↪intensity.",
        "versions": [
            {
                "title": "1.0.0",
                "description": "FSL 6.0 version of the SUSAN algorithm.",
                "input": SUSAN_INPUT_SPECIFICATION,
                "output": SUSAN_OUTPUT_SPECIFICATION,
                "nested_results_attribute": "outputs.get_traitsfree",
            }
        ],
    }
]
```

Note: The `nested_results_attribute` field allows us to integrate smoothly with Nipype's `BaseTraitedSpec` class by extracting the results dictionary from the returned object.

For more information see *Integration Customization*.

All that's left to do is:

```
>>> from analysis_definitions import analysis_definitions
>>> from django_analyses.models import Analysis
>>> results = Analysis.objects.from_list(analysis_definitions)
>>> results
{'SUSAN': {'model': <Analysis: SUSAN>, 'created': True, 'versions': {'1.0.0': {'model
↪': <AnalysisVersion: SUSAN v1.0.0>, 'created': True}}}}
```

The *from_list()* method returns a dictionary with references to the specified *Analysis* and *AnalysisVersion* instances and whether they have been created or not.

SUSAN is now an available analysis in our database. Only one thing missing...

Interface Integration

In order to *django_analyses* to be able to locate the interface used to run this analysis version, we must add to our project's ANALYSIS_INTERFACES setting:

Listing 5: settings.py

```
from nipype.interfaces.fsl import SUSAN

...

ANALYSIS_INTERFACES = {"SUSAN": {"1.0.0": SUSAN}}
```

All done!

3.1.3 Integration Customization

The *AnalysisVersion* model offers three special fields that may be used to customize an instance's integration with its interface:

- *run_method_key*: str determining the name of the interface's method that will be called when executing. By default this will be set to *"run"*.
- *fixed_run_method_kwargs*: dict of fixed keyword arguments to pass to the interface's run() method when called. By default this will be set to {}.
- *nested_results_attribute*: str specifying a nested attribute or method to be called on a returned object to retrieve a dict of the results. By default this will be set to None.

3.1.4 Analysis Execution

One-off Execution

To execute the *ExponentCalculator* interface we created in the *Simplified Analysis Integration Example*, we could run:

```
>>> analysis = Analysis.objects.get(title="Exponentiation")
>>> analysis_version = analysis.version_set.get(title="built-in")
>>> kwargs = {"base": 2, "exponent": 10}
>>> results = analysis_version.run(**kwargs)
>>> results
{'result': 1024.0}
```

This will run the associated interface, however, it will not create any record of this run in the database.

Node Execution

To execute a particular analysis version and record the run (as well as any inputs and outputs) in the database, we must run it as a node. Again, we will use a node previously created in the *Simplified Pipeline Generation Example*:

```
>>> configuration = {"exponent": 2}
>>> node = Node.objects.get(
...     analysis_version=analysis_version,
...     configuration=configuration,
... )
>>> inputs = {"base": 4.5}
>>> results = node.run(inputs=inputs)
```

This time, our *results* are a `Run` instance which is associated with all the recorded inputs and outputs:

```
>>> results
<Run: #1 Exponentiation vbuilt-in run from 2020-01-01 00:00:00.000000>
>>> results.input_set
<InheritanceQuerySet [<FloatInput: 'base' = 4.5>, <FloatInput: 'exponent' = 2.0>]>
>>> results.output_set
<InheritanceQuerySet [<FloatOutput: 'result' = 20.25>]>
>>> results.get_output("result")
20.25
```

Node Iteration

To run a node multiple types, simply provide the `run()` method's *inputs* parameter a list or tuple of input dictionaries, e.g.:

```
>>> inputs = [{"base": 1}, {"base": 2}, {"base": 3}]
>>> results = node.run(inputs=inputs)
>>> results
[<Run: #2 Exponentiation vbuilt-in run from 2020-01-01 00:00:01.000000>,
 <Run: #3 Exponentiation vbuilt-in run from 2020-01-01 00:00:02.000000>,
 <Run: #4 Exponentiation vbuilt-in run from 2020-01-01 00:00:03.000000>]
```

3.2 Pipeline Generation

`Pipeline` instances represent a distinct association pattern between the outputs and inputs of pre-configured analyses.

3.2.1 Simplified Pipeline Generation Example

As a simple example for a pipeline generation flow, we will reuse the `ExponentCalculator` from the *Simplified Analysis Integration Example* to create a pipeline which which computes 3^{x^2} (where x is the provided input).

Nodes

A `Node` instance provides a reference to a particular configuration of some analysis version.

First we need to create a `square` node:

```
>>> from django_analyses.models import AnalysisVersion, Node

# Querying the database for our desired analysis version
>>> exponent_calculation = AnalysisVersion.objects.get(
>>>     analysis__title="Exponentiation", title="built-in"
>>> )

# Creating a 'square' node
>>> configuration = {"exponent": 2}
>>> square = Node.objects.create(
>>>     analysis_version=exponent_calculation, configuration=configuration
>>> )
```

Now, we could use our `square` node to calculate 2^2:

```
>>> run_1 = square.run(inputs={"base": 2})
>>> run_1
<Run: #1 Exponentiation vbuilt-in run from 2020-01-01 00:00:00.000000>
>>> run_1.get_output(key='result')
4
```

Each run will be recorded in the database and returned whenver we call `ExponentCalculator` with the same parameters.

```
>>> run_2 = square.run(inputs={"base": 2})
>>> run_1 == run_2
# True
```

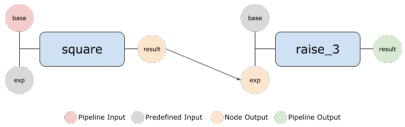
We also need a `raise_3` node for our pipeline:

```
>>> raise_3 = Node.objects.create(analysis_version=exponent_calculation,␣
↪configuration={"base": 3})
```

Pipes

A *`Pipe`* instance is used to stream data across runs by associating one given node's output with another's input.

In our case, the required pipe is represented by the arrow connecting `square`'s result and `raise_3`'s exponent.



First we create the *`Pipeline`* instance:

```
>>> from django_analyses.models import Pipeline
>>> pipeline = Pipeline.objects.create(
>>>     title="Simple Pipeline", description="A simple pipeline."
>>> )
```

And then we can lay down the pipe:

```
>>> from django_analyses.models import Pipe

# Querying the required InputDefinition instances
```

(continues on next page)

(continued from previous page)

```
>>> square_output = exponent_calculation.output_definitions.get(key="result")
>>> raise_3_input = exponent_calculation.input_definitions.get(key="exponent")

# Create the pipe
>>> pipe = Pipe.objects.create(
>>>     pipeline=pipeline,
>>>     source=square,
>>>     base_source_port=square_output,
>>>     destination=raise_3,
>>>     base_destination_port=raise_3_input,
>>> )
```

3.2.2 Realistic Pipeline Generation Example

As a realistic pipeline generation usage example we will create a basic brain MRI preprocessing pipeline using Nipype's interface for FSL.

The pipeline will receive a NIfTI instance from the database as input and then run:

1. Brain extraction (BET)
2. Reorientation to the MNI152 standard brain template space (fslreorient2std)
3. Cropping (robustfov)
4. Linear registration (FLIRT)
5. Nonlinear registration (FNIRT)

django_mri already provides the required analyses, so in order to create a new pipeline we can simply use a `dict` defining the pipes that make up the pipeline.

First, this pipeline assumes the MNI152 standard brain template exists in the database:

Listing 6: basic_fsl_preprocessing.py

```
from django_mri.models.nifti import NIfTI


try:
    MNI = NIfTI.objects.get(path__contains="MNI152_T1_2mm_brain")
except NIfTI.DoesNotExist:
    raise NIfTI.DoesNotExist("Could not find MNI152_T1_2mm_brain in the database.")
```

Then, in order to keep things readable, let's define each node's configuration and create a definition of that node:

```
BET_CONFIGURATION = {"robust": True}
REORIENT_CONFIGURATION = {}
ROBUSTFOV_CONFIGURATION = {}
FLIRT_CONFIGURATION = {"reference": MNI.id, "interp": "spline"}
FNIRT_CONFIGURATION = {"ref_file": MNI.id}

BET_NODE = {"analysis_version": "BET", "configuration": BET_CONFIGURATION}
REORIENT_NODE = {
    "analysis_version": "fslreorient2std",
    "configuration": REORIENT_CONFIGURATION,
}
ROBUST_FOV_NODE = {
```

(continues on next page)

(continued from previous page)

```
    "analysis_version": "robustfov",
    "configuration": ROBUSTFOV_CONFIGURATION,
}
FLIRT_NODE = {
    "analysis_version": "FLIRT",
    "configuration": {"reference": MNI.id, "interp": "spline"},
}
FNIRT_NODE = {
    "analysis_version": "FNIRT",
    "configuration": {"ref_file": MNI.id},
}
```

Now we can specify the pipes:

```
BET_TO_REORIENT = {
    "source": BET_NODE,
    "source_port": "out_file",
    "destination": REORIENT_NODE,
    "destination_port": "in_file",
}
REORIENT_TO_FOV = {
    "source": REORIENT_NODE,
    "source_port": "out_file",
    "destination": ROBUST_FOV_NODE,
    "destination_port": "in_file",
}
FOV_TO_FLIRT = {
    "source": ROBUST_FOV_NODE,
    "source_port": "out_roi",
    "destination": FLIRT_NODE,
    "destination_port": "in_file",
}
FLIRT_TO_FNIRT = {
    "source": FLIRT_NODE,
    "source_port": "out_file",
    "destination": FNIRT_NODE,
    "destination_port": "in_file",
}
```

And finally, everything is ready for the pipeline:

```
BASIC_FSL_PREPROCESSING = {
    "title": "Basic FSL Preprocessing",
    "description": "Basic MRI preprocessing pipeline using FSL.",
    "pipes": [BET_TO_REORIENT, REORIENT_TO_FOV, FOV_TO_FLIRT, FLIRT_TO_FNIRT],
}
```

Add the pipeline to the database:

```
>>> from basic_fsl_preprocessing import BASIC_FSL_PREPROCESSING
>>> from django_analyses.models import Pipeline
>>> from django_analyses.pipeline_runner import PipelineRunner
>>> from django_mri.models import Scan

>>> pipeline = Pipeline.objects.from_dict(BASIC_FSL_PREPROCESSING)
>>> scan = Scan.objects.filter(description__icontains="MPRAGE").first()
>>> pipeline_input = {"in_file": scan.nifti}
```

(continues on next page)

(continued from previous page)

```
>>> pipeline_runner = PipelineRunner(pipeline)
>>> results = pipeline_runner.run(inputs=pipeline_input)
>>> results
{<Node:
Node #51
FLIRT v6.0.3:b862cdd5
Configuration: [{'interp': 'spline', 'reference': 585}]
>: <Run: #117 FLIRT v6.0.3:b862cdd5 run from 2020-06-01 12:34:51.103207>,
<Node:
Node #49
BET v6.0.3:b862cdd5
Configuration: [{'robust': True}]
>: <Run: #114 BET v6.0.3:b862cdd5 run from 2020-06-01 12:34:39.941216>,
<Node:
Node #44
FNIRT v6.0.3:b862cdd5
Configuration: [{'ref_file': 585}]
>: <Run: #118 FNIRT v6.0.3:b862cdd5 run from 2020-06-01 12:35:28.673283>,
<Node:
Node #43
robustfov v6.0.3:b862cdd5
Configuration: [{}]
>: <Run: #116 robustfov v6.0.3:b862cdd5 run from 2020-06-01 12:34:48.512874>,
<Node:
Node #42
fslreorient2std v6.0.3:b862cdd5
Configuration: [{}]
>: <Run: #115 fslreorient2std v6.0.3:b862cdd5 run from 2020-06-01 12:34:46.985302>}
```

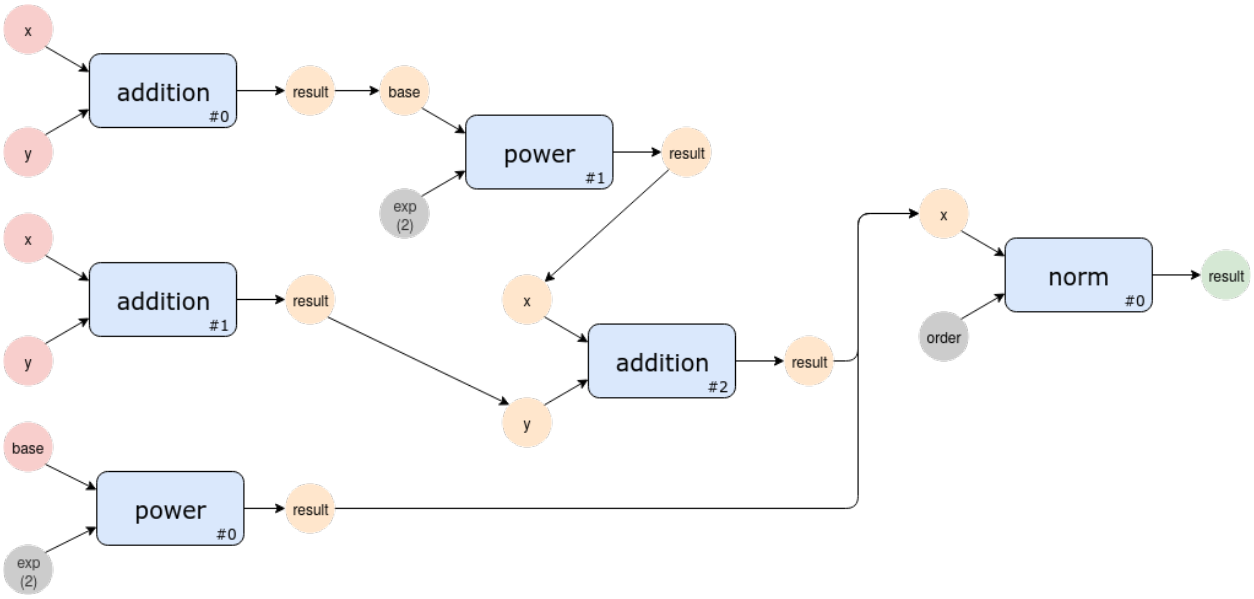
To get our output file we could:

```
>>> from django_analyses.models import Run
>>> from django_mri.models import NIfTI
>>> run = Run.objects.get(id=118)
>>> for output in run.output_set.all():
>>>     print(output.key, output.value)
fieldcoeff_file NIfTI object (653)
log_file /media/dir/analysis/118/log.txt
modulatedref_file NIfTI object (652)
warped_file NIfTI object (649)
field_file NIfTI object (650)
jacobian_file NIfTI object (651)
>>> path = NIfTI.objects.get(id=649).path
>>> path
/media/dir/analysis/118/warped.nii.gz
```

3.2.3 Multiple Identical Nodes

In cases where a particular node needs to be executed multiple times in the same pipeline, the *Pipe* model's *source_run_index* and *destination_run_index* attributes should be used.

As an example for a pipeline running multiple identical nodes, we will create the following pipeline:



We will assume the existence of the three nodes:

- *addition*: Adds two number to return "result".
- *power*: Raises *base* in the power of *exp*. In this case we have a "square" node, where *exp* is preconfigured as 2.
- *norm*: Calculated the norm of the provided vector (*x*) using NumPy's `linalg.norm()` method.

Each node has a designated run index in it's bottom right corner, distinguishing it from other executions of this node.

To create the pipeline, we will first create a pipeline specification dictionary:

Listing 7: multiple_identical_nodes.py

```
ADDITION_NODE = {
    "analysis_version": "addition",
    "configuration": {},
}
SQUARE_NODE = {
    "analysis_version": "power",
    "configuration": {"exp": 2},
}
NORM_NODE = {
    "analysis_version": "norm",
    "configuration": {},
}
PIPELINE = {
    "title": "Multiple Identical Nodes",
    "description": "Simple pipeline with identical nodes.",
    "pipes": [
        # Addition #0 [result] --> Power #1 [base]
        {
            "source": ADDITION_NODE,
            "source_run_index": 0,
            "source_port": "result",
            "destination": SQUARE_NODE,
            "destination_run_index": 1,
            "destination_port": "base",
        },
```

(continues on next page)

(continued from previous page)

```
        # Power #1 [result] --> Addition #2 [x]
        {
            "source": SQUARE_NODE,
            "source_run_index": 1,
            "source_port": "result",
            "destination": ADDITION_NODE,
            "destination_run_index": 2,
            "destination_port": "x",
        },
        # Addition #1 [result] --> Addition #2 [y]
        {
            "source": ADDITION_NODE,
            "source_run_index": 1,
            "source_port": "result",
            "destination": ADDITION_NODE,
            "destination_run_index": 2,
            "destination_port": "y",
        },
        # Power #0 [result] --> Norm #0 [x[0]]
        {
            "source": SQUARE_NODE,
            "source_run_index": 0,
            "source_port": "result",
            "destination": NORM_NODE,
            "destination_run_index": 0,
            "destination_port": "x",
            "index": 0,
        },
        # Addition #2 [result] --> Norm #0 [x[1]]
        {
            "source": ADDITION_NODE,
            "source_run_index": 2,
            "source_port": "result",
            "destination": NORM_NODE,
            "destination_run_index": 0,
            "destination_port": "x",
            "index": 1,
        },
    ],
}
```

Note that we also used the `Pipe` model's `index` attribute to pass two outputs as a single list (vector) to the *norm* nodes *x* parameter.

To run the pipeline, we need to specify both *x* and *y* for the first two *addition* executions, as well as the base to the first *power* execution.

```
>>> from django_analyses.models import AnalysisVersion
>>> from django_analyses.models import Node
>>> from django_analyses.models import Pipeline
>>> from django_analyses.pipeline_runner import PipelineRunner

# Import the pipeline specification dictionary created above
>>> from multiple_identical_nodes import PIPELINE
# Create the pipeline
>>> pipeline = Pipeline.objects.from_dict(PIPELINE)
```

(continues on next page)

(continued from previous page)

```
# Fetch the nodes for the inputs dictionary
>>> addition = AnalysisVersion.objects.get(analysis__title="addition")
>>> power = AnalysisVersion.objects.get(analysis__title="power")
>>> addition_inputs = [{"x": 1, "y": 1}, {"x": 1, "y": 1}]
>>> power_inputs = [{"base": 1}]
>>> inputs = {addition: addition_inputs, power: power_inputs}
>>> runner = PipelineRunner(pipeline=pipeline)
>>> results = runner.run(inputs=inputs)
power v1.0 (#0)
Inputs:
{'base': 1}
...
Outputs:
{'result': 1.0}
————————————————————

addition v1.0 (#0)
Inputs:
{'x': 1, 'y': 1}
...
Outputs:
{'result': 2.0}
————————————————————
————————————————————
power v1.0 (#1)
Inputs:
{'base': 2.0}
...
Outputs:
{'result': 4.0}
————————————————————
————————————————————
addition v1.0 (#1)
Inputs:
{'x': 1, 'y': 1}
...
Outputs:
{'result': 2.0}
————————————————————
————————————————————
addition v1.0 (#2)
Inputs:
{'y': 2.0, 'x': 4.0}
...
Outputs:
{'result': 6.0}
————————————————————
————————————————————
norm vNumPy:1.18 (#0)
Inputs:
{'x': [1.0, 6.0]}
...
Outputs:
{'norm': 6.08276253029822}
————————————————————
done!
>>> norm = AnalysisVersion.objects.get(analysis__title="norm")
```

(continues on next page)

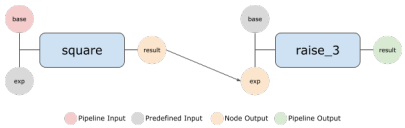
(continued from previous page)

```
>>> results[norm][0].get_output("norm")
6.082762530298219
```

3.2.4 Pipeline Execution

Pipelines are executed using a *PipelineRunner* instance, which wraps-up all the required logic.

We will use the *"Simple Pipeline"* created in the *Simplified Pipeline Generation Example* to calculate 3^{2^2}.



"Simple Pipeline" Illustration. In the following example, we will pass 2 as the "base" input for the "square" node.

```
>>> from django_analyses.pipeline_runner import PipelineRunner
>>> from django_analyses.models import Node
>>> pipeline = Pipeline.objects.get(title="Simple Pipeline")
>>> pipeline_runner = PipelineRunner(pipeline)
>>> square_node = Node.objects.get(analysis_version__analysis__title="Exponentiation")
>>> runs = pipeline_runner.run(inputs={square_node: [{"base": 2}]})
```

The returned `runs` variable is a `dict` instance containing the pipeline's nodes as keys lists of runs as values. Examining `runs` will show that 2^2 returned `[Run #1]` (which was created at the beginning of the pipeline generation tutorial), whereas 3^4 was a novel calculation and therefore a new run has been was created.

Finally, to view our output:

```
>>> from django_analyses.models.analysis_version import AnalysisVersion
>>> raise_3 = AnalysisVersion.objects.get(analysis__title="Exponentiation").first()
>>> runs[raise_3].get_output("result")
81
```

Specifying User Inputs

User inputs are expected to be provided as a dictionary with nodes as keys and lists of dictionary input configurations as values. This format enables a user to specify inputs for multiple entry points, as well as multiple runs of a particular entry node, in case it is required. However, in many cases, pipelines either only have a single entry point or single executions of a number of entry points.

Single Entry Point Input Configuration

For this reason, in case there is only a single entry point, the required input configuration dictionary may be assigned directly, e.g. in the base *Pipeline Execution* example above, we could have also used:

```
>>> runs = pipeline_runner.run(inputs={"base": 2})
```

Because *square_node* is the only entry point in the pipeline, the input configuration dictionary will automatically be assigned to that node.

3.3 `QuerySet` Processing

3.3.1 Minimal Exmaple

This section details the recommended procedure for creating an interface to easily run some node in batch over a default or provided queryset of some data-representing model.

The *`QuerySetRunner`* base class provides a reusable abstraction for the general process of executing some `Node` instance with inputs generated from a queryset.

For example, let us assume a `Scan` model storing data scans in the database, and a `"Scan Preprocessing"` analysis of version `"1.0"` we would like to routinely run with the configuration: `{"harder": True, "better": 100, "stronger": "faster"}`. In addition, we know the analysis receives the `Scan` model's `path` field's value as its `"input_file"`. The resulting subclass will look like:

```
from django_analyses.runner import QuerySetRunner
from myapp.models.scan import Scan

class ScanPreprocessingRunner(QuerySetRunner):
    DATA_MODEL = Scan
    ANALYSIS = "Scan Preprocessing"
    ANALYSIS_VERSION = "1.0"
    ANALYSIS_CONFIGURATION = {
        "harder": True,
        "better": 100,
        "stronger": "faster",
    }
    INPUT_KEY = "input_file"

    def get_instance_representation(self, instance: Scan) -> str:
        return str(instance.path)
```

And that's it!

Note: The *`get_instance_representation()`* method will, if not overriden, return the instance as it is.

Using model instances as inputs is a fairly advanced usage scenario and outside the scope of this tutorial, therefore, the minimal example includes this modification.

To run the specified node over all `Scan` instance in the database:

```
>>> runner = ScanPreprocessingRunner()
>>> runner.run()
Scan Preprocessing v1.0: Batch Execution

 Default execution queryset generation:
Querying Scan instances...
1000 instances found.

 Checking execution status for the input queryset:
Filtering existing runs...
20 existing runs found.
980 instances pending execution.

 Generating input specifications:
980 input specifications prepared.
```

(continues on next page)

(continued from previous page)

```
Successfully started Scan Preprocessing v1.0 execution over 980 Scan instances
```

QuerySetRunner took care of querying all instances of the `Scan` model, checking for pending runs, generating the required input specifications, and running them in the background.

To run over a particular queryset, simply pass the queryset to the *run()* method.

3.3.2 Default `QuerySet` Filtering

To apply custom filtering to the data model's queryset, override the *filter_queryset()* method. For example, if we would like to process only scans with `"anatomical"` in their description:

```
import logging
from django.db.models import QuerySet
from django_analyses.runner import QuerySetRunner
from myapp.models.scan import Scan

class ScanPreprocessingRunner(QuerySetRunner):
    DATA_MODEL = Scan
    ANALYSIS = "Scan Preprocessing"
    ANALYSIS_VERSION = "1.0"
    ANALYSIS_CONFIGURATION = {
        "harder": True,
        "better": 100,
        "stronger": "faster",
    }
    INPUT_KEY = "input_file"
    FILTER__QUERYSET_START = "Filtering anatomical scans..."

    def get_instance_representation(self, instance: Scan) -> str:
        return str(instance.path)

    def filter_queryset(self,
        queryset: QuerySet, log_level: int = logging.INFO
    ) -> QuerySet:
        queryset = super().filter_queryset(queryset, log_level=log_level)
        self.log_filter_start(log_level)
        queryset = queryset.filter(description__icontains="anatomical")
        self.log_filter_end(n_candidates=queryset.count(), log_level=log_level)
        return queryset
```

This time, when we run `ScanPreprocessingRunner`, we get the result:

```
>>> runner = ScanPreprocessingRunner()
>>> runner.run()
Scan Preprocessing v1.0: Batch Execution

 Default execution queryset generation:
Querying Scan instances...
1000 instances found.
Filtering anatomical scans...
500 execution candidates found.

 Checking execution status for the input queryset:
```

(continues on next page)

(continued from previous page)

```
Filtering existing runs...
20 existing runs found.
480 instances pending execution.

 Generating input specifications:
480 input specifications prepared.

Successfully started Scan Preprocessing v1.0 execution over 480 Scan instances
```

Note:

- Filtering is applied to provided querysets as well, not just the default.
- `super().filter_queryset(queryset)` is called to apply any preceding filtering.
- The log message is replaced by overriding the *`FILTER_QUERYSET_START`* class attribute (which is used automatically by *`filter_queryset()`* to log the filtering of the input queryset.

The *`QuerySetRunner`* class provides a wide range of utility attributes and functions that enable the automation of highly customized queryset processing. For more information, simply follow the *`QuerySetRunner`* hyperlink to the class's reference.

CHAPTER 4

Reference

4.1 Module contents

A reusable Django app to manage analyses and pipelines.

4.2 Subpackages

4.2.1 Filters

Module contents

Filters for the app's *models*.

References

- Django REST Framework filtering documentation.
- django-filter's documentation for Integration with DRF.

Subpackages

Input Filters

Module contents

Filters for the `Input` and `InputDefinition` subclasses.

Submodules

django_analyses.filters.input.input module

Definition of an *InputFilter* for the *Input* model.

class django_analyses.filters.input.input.**InputFilter**(*data=None*, *queryset=None*, ***, *request=None*, *prefix=None*)

Bases: django_filters.rest_framework.filterset.FilterSet

Provides useful filtering options for the *Input* model.

class Meta

Bases: object

fields = ('run', 'key')

model

alias of *django_analyses.models.input.input.Input*

base_filters = {'input_type': <django_filters.filters.ChoiceFilter object>, 'key': <

declared_filters = {'input_type': <django_filters.filters.ChoiceFilter object>, 'key'

filter_input_type(*queryset*, *name*, *value*)

filter_key(*queryset*, *name*, *value*)

django_analyses.filters.input.input_definition module

Definition of an *InputDefinitionFilter* for the InputDefinition model.

class django_analyses.filters.input.input_definition.**InputDefinitionFilter**(*data=None*, *queryset=None*, ***, *request=None*, *prefix=None*)

Bases: django_filters.rest_framework.filterset.FilterSet

Provides useful filtering options for the *InputDefinition* model.

class Meta

Bases: object

fields = ('key', 'required', 'is_configuration', 'input_specification')

model

alias of *django_analyses.models.input.definitions.input_definition.InputDefinition*

base_filters = {'input_specification': <django_filters.filters.AllValuesFilter object

declared_filters = {'input_specification': <django_filters.filters.AllValuesFilter ob

Output Filters

Module contents

Filters for the *Output* and *OutputDefinition* subclasses.

Submodules

django_analyses.filters.output.output module

Definition of an *OutputFilter* for the *Output* model.

class django_analyses.filters.output.output.**OutputFilter**(*data=None*, *queryset=None*, ***, *request=None*, *prefix=None*)

Bases: django_filters.rest_framework.filterset.FilterSet

Provides useful filtering options for the *Output* model.

class Meta

Bases: object

fields = ('run', 'key')

model

alias of *django_analyses.models.output.output.Output*

base_filters = {'key': <django_filters.filters.CharFilter object>, 'output_type': <d

declared_filters = {'key': <django_filters.filters.CharFilter object>, 'output_type':

filter_key(*queryset*, *name*, *value*)

filter_output_type(*queryset*, *name*, *value*)

django_analyses.filters.output.output_definition module

Definition of an *OutputDefinitionFilter* for the OutputDefinition model.

class django_analyses.filters.output.output_definition.**OutputDefinitionFilter**(*data=None*, *queryset=None*, ***, *request=None*, *prefix=None*)

Bases: django_filters.rest_framework.filterset.FilterSet

Provides useful filtering options for the *OutputDefinition* model.

class Meta

Bases: object

fields = ('key', 'output_specification')

model
alias of *django_analyses.models.output.definitions.output_definition.OutputDefinition*

base_filters = {'key': <django_filters.filters.CharFilter object>, 'output_specificat

declared_filters = {'output_specification': <django_filters.filters.AllValuesFilter ob

Pipeline Filters

Module contents

Filters for the *Node*, *Pipe*, and *Pipeline* models.

Submodules

django_analyses.filters.pipeline.node module

Definition of a *NodeFilter* for the *Node* model.

class django_analyses.filters.pipeline.node.**NodeFilter**(*data=None*, *queryset=None*, ***, *request=None*, *prefix=None*)

Bases: django_filters.rest_framework.filterset.FilterSet

Provides useful filtering options for the *Node* model.

class Meta
Bases: object

fields = ('analysis_version',)

model
alias of *django_analyses.models.pipeline.node.Node*

base_filters = {'analysis_version': <django_filters.filters.ModelChoiceFilter object>

declared_filters = {}

django_analyses.filters.pipeline.pipe module

Definition of a *PipeFilter* for the *Pipe* model.

class django_analyses.filters.pipeline.pipe.**PipeFilter**(*data=None*, *queryset=None*, ***, *request=None*, *prefix=None*)

Bases: django_filters.rest_framework.filterset.FilterSet

Provides useful filtering options for the *Pipe* model.

class Meta
Bases: object

fields = ('pipeline', 'source', 'base_source_port', 'destination', 'base_destinatio

model
alias of *django_analyses.models.pipeline.pipe.Pipe*

base_filters = {'base_destination_port': <django_filters.filters.ModelChoiceFilter ob

declared_filters = {}

django_analyses.filters.pipeline.pipeline module

Definition of a *PipelineFilter* for the *Pipeline* model.

class django_analyses.filters.pipeline.pipeline.**PipelineFilter**(*data=None*, *queryset=None*, ***, *request=None*, *prefix=None*)

Bases: django_filters.rest_framework.filterset.FilterSet

Provides useful filtering options for the *Pipeline* model.

class Meta

Bases: object

fields = ('title', 'description', 'created', 'modified')

model

alias of *django_analyses.models.pipeline.pipeline.Pipeline*

base_filters = {'created': <django_filters.filters.IsoDateTimeFilter object>, 'descri

declared_filters = {}

Submodules

django_analyses.filters.analysis module

Definition of an *AnalysisFilter* for the *Analysis* model.

class django_analyses.filters.analysis.**AnalysisFilter**(*data=None*, *queryset=None*, ***, *request=None*, *prefix=None*)

Bases: django_filters.rest_framework.filterset.FilterSet

Provides useful filtering options for the *Analysis* model.

class Meta

Bases: object

fields = ('id', 'category')

model

alias of *django_analyses.models.analysis.Analysis*

base_filters = {'category': <django_filters.filters.ModelChoiceFilter object>, 'creat

declared_filters = {'created': <django_filters.filters.DateTimeFromToRangeFilter obje

filter_has_runs(*queryset*, *name*, *value*)

django_analyses.filters.category module

Definition of a `CategoryFilter` for the `Category` model.

class django_analyses.filters.category.**CategoryFilter**(*data=None*, *queryset=None*, ***, *request=None*, *prefix=None*)

Bases: `django_filters.rest_framework.filterset.FilterSet`

Provides useful filtering options for the `Category` model.

class Meta

Bases: `object`

fields = ('id', 'title', 'description', 'parent', 'is_root')

model

alias of `django_analyses.models.category.Category`

base_filters = {'description': <django_filters.filters.LookupChoiceFilter object>, 'i

declared_filters = {'description': <django_filters.filters.LookupChoiceFilter object>

filter_is_root(*queryset*, *name: str*, *value: bool*)

4.2.2 Models

Module contents

Creates `Model` and `Manager` subclasses to represent and manage all the different parts of an analysis pipeline.

For an illustration of the app's models, see the *Overview*.

Subpackages

Input

Module contents

Definition of all the models required to create an input specification for some `AnalysisVersion` and keep track of the inputs associated with some `Run` instance.

Subpackages

django_analyses.models.input.definitions package

Submodules

django_analyses.models.input.definitions.boolean_input_definition module

class django_analyses.models.input.definitions.boolean_input_definition.**BooleanInputDefinit**

Bases: *django_analyses.models.input.definitions.input_definition.InputDefinition*

default
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

get_type() → django_analyses.models.input.definitions.input_definitions.InputDefinitions

input_class
alias of *django_analyses.models.input.types.boolean_input.BooleanInput*

input_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

inputdefinition_ptr
Accessor to the related object on the forward side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Restaurant.place` is a `ForwardOneToOneDescriptor` instance.

inputdefinition_ptr_id

is_output_switch
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

django_analyses.models.input.definitions.directory_input_definition module

class django_analyses.models.input.definitions.directory_input_definition.**DirectoryInputDef**

Bases: *django_analyses.models.input.definitions.input_definition.InputDefinition*

default
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

get_type() → django_analyses.models.input.definitions.input_definitions.InputDefinitions

input_class
alias of *django_analyses.models.input.types.directory_input.DirectoryInput*

input_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

inputdefinition_ptr
Accessor to the related object on the forward side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Restaurant.place` is a `ForwardOneToOneDescriptor` instance.

inputdefinition_ptr_id

is_output_directory
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

django_analyses.models.input.definitions.file_input_definition module

class django_analyses.models.input.definitions.file_input_definition.**FileInputDefinition**(*id ke de sc ti re qu is va db ru co te fie in pu de i- ni ti de fa*

Bases: *django_analyses.models.input.definitions.input_definition.InputDefinition*

default
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

get_type() → django_analyses.models.input.definitions.input_definitions.InputDefinitions

input_class
alias of *django_analyses.models.input.types.file_input.FileInput*

input_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

inputdefinition_ptr
Accessor to the related object on the forward side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Restaurant.place` is a `ForwardOneToOneDescriptor` instance.

inputdefinition_ptr_id

django_analyses.models.input.definitions.float_input_definition module

class django_analyses.models.input.definitions.float_input_definition.**FloatInputDefinition**

Bases: *django_analyses.models.input.definitions.number_input_definition.NumberInputDefinition*

default
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is

executed.

get_type() → django_analyses.models.input.definitions.input_definitions.InputDefinitions

input_class
alias of *django_analyses.models.input.types.float_input.FloatInput*

input_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

max_value
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

min_value
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

numberinputdefinition_ptr
Accessor to the related object on the forward side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Restaurant.place` is a `ForwardOneToOneDescriptor` instance.

numberinputdefinition_ptr_id

django_analyses.models.input.definitions.input_definition module

Definition of the *InputDefinition* class.

class django_analyses.models.input.definitions.input_definition.**InputDefinition**(**args*, ***kwargs*)

Bases: `django.db.models.base.Model`

Represents a single input definition in the database. Instances are used to as the building blocks for *InputSpecification* instances.

booleaninputdefinition
Accessor to the related object on the reverse side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Place.restaurant` is a `ReverseOneToOneDescriptor` instance.

check_input_class_definition() → None
Checks the validity of the assigned *input_class*.

Raises ValidationError – Invalid *input_class* definition

content_type
If this input's value references the value of a field in the database, this references the field's model.

content_type_id

db_value_preprocessing
If values passed as inputs matching this input definition should be extracted from some object, this field specifies the name of the attribute which will be called using *get_db_value()*.

default = None
Child models may allow setting a default value using the appropriate Field subclass.

description
A description of this input definition.

directoryinputdefinition
Accessor to the related object on the reverse side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

Place.restaurant is a ReverseOneToOneDescriptor instance.

extract_nested_value(*obj: Any*, *location: str*) → Any
Extract some nested attribute within an object.

Parameters

- **obj** (*Any*) – The object containing the nested value
- **location** (*str*) – Address of nested attribute within object

Returns Nested attribute value

Return type Any

field_name
If this input's value references the value of a field in the database, this references the field's name within the **:att:'content_type'** model.

fileinputdefinition
Accessor to the related object on the reverse side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

Place.restaurant is a ReverseOneToOneDescriptor instance.

get_db_value(*value: Any*) → Any
Returns the appropriate DB value for inputs in which *db_value_preprocessing* is defined.

Parameters **value** (*Any*) – The object containing the nested value

Returns Nested attribute value

Return type Any

Raises `ValueError` – Value extraction failure

get_or_create_input_instance(***kwargs*) → django_analyses.models.input.input.Input
Creates an instance of the appropriate *django_analyses.models.input.input.Input* subclass.

Returns Created instance

Return type *Input*

input_class = None
Each definition should override this class attribute in order to allow for Input instances creation.

is_configuration
Whether this input definition is a configuration of the analysis parameters or, e.g., a definition of the input or output of it.

key
Input key used when passing inputs to run some analysis.

listinputdefinition
Accessor to the related object on the reverse side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Place.restaurant` is a `ReverseOneToOneDescriptor` instance.

numberinputdefinition
Accessor to the related object on the reverse side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Place.restaurant` is a `ReverseOneToOneDescriptor` instance.

objects = <django_analyses.models.managers.input_definition.InputDefinitionManager obj

pipe_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

required
Whether this input is required for the execution of the analysis.

run_method_input
Whether the created inputs instances should be passed to interface's class at initialization (False) or upon calling the run method (True).

save(**args*, ***kwargs*)
Overrides the model's `save()` method to provide custom functionality.

specification_set
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`Pizza.toppings` and `Topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

stringinputdefinition
Accessor to the related object on the reverse side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Place.restaurant` is a `ReverseOneToOneDescriptor` instance.

validate() → None
Validates input definition instances before calling *save()*. This method should be overridden by subclasses that require some kind of custom validation.

value_attribute
If the actual input to the analysis class is meant to be some attribute of given input, the attribute name may be set here.

django_analyses.models.input.definitions.input_definitions module

class django_analyses.models.input.definitions.input_definitions.**InputDefinitions**
Bases: `enum.Enum`

An enumeration.

BLN = 'Boolean'

DIR = 'Directory'

FIL = 'File'

FLT = 'Float'

INT = 'Integer'

LST = 'List'

STR = 'String'

django_analyses.models.input.definitions.integer_input_definition module

class django_analyses.models.input.definitions.integer_input_definition.**IntegerInputDefinit**

Bases: *django_analyses.models.input.definitions.number_input_definition.NumberInputDefinition*

default
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

get_type() → django_analyses.models.input.definitions.input_definitions.InputDefinitions

input_class
alias of *django_analyses.models.input.types.integer_input.IntegerInput*

input_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by

`create_forward_many_to_many_manager()` defined below.

max_value
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

min_value
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

numberinputdefinition_ptr
Accessor to the related object on the forward side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Restaurant.place` is a `ForwardOneToOneDescriptor` instance.

numberinputdefinition_ptr_id

django_analyses.models.input.definitions.list_input_definition module

class django_analyses.models.input.definitions.list_input_definition.**ListInputDefinition**(*id*
ke
de
sc
tio
re
qu
*is_*
va
db
ru
co
te
fie
in
pu
de
i-
ni
tio
el
e-
m
m
m
de
fa
as

Bases: *django_analyses.models.input.definitions.input_definition.InputDefinition*

as_tuple

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

default
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

element_type
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

expected_type_definition

get_element_type_display(*, *field=<django.db.models.fields.CharField: element_type>*)

get_type() → django_analyses.models.input.definitions.input_definitions.InputDefinitions

input_class
alias of *django_analyses.models.input.types.list_input.ListInput*

input_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

inputdefinition_ptr
Accessor to the related object on the forward side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Restaurant.place` is a `ForwardOneToOneDescriptor` instance.

inputdefinition_ptr_id

max_length
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

min_length
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

raise_max_length_error() → None

raise_min_length_error() → None

raise_not_list_error() → None

raise_wrong_type_error() → None

validate()
Validates input definition instances before calling `save()`. This method should be overridden by subclasses that require some kind of custom validation.

validate_default_value() → None

validate_default_value_max_length() → bool

validate_default_value_min_length() → bool

validate_elements_type_for_default() → bool

django_analyses.models.input.definitions.messages module

django_analyses.models.input.definitions.number_input_definition module

class django_analyses.models.input.definitions.number_input_definition.**NumberInputDefiniti**

Bases: *django_analyses.models.input.definitions.input_definition.InputDefinition*

floatinputdefinition

Accessor to the related object on the reverse side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Place.restaurant` is a `ReverseOneToOneDescriptor` instance.

inputdefinition_ptr

Accessor to the related object on the forward side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Restaurant.place` is a `ForwardOneToOneDescriptor` instance.

inputdefinition_ptr_id

integerinputdefinition
Accessor to the related object on the reverse side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Place.restaurant` is a `ReverseOneToOneDescriptor` instance.

raise_default_over_max_error()

raise_default_under_min_error()

validate() → None
Validates input definition instances before calling `save()`. This method should be overridden by subclasses that require some kind of custom validation.

validate_default()

django_analyses.models.input.definitions.string_input_definition module

Definition of the *StringInputDefinition* class.

class django_analyses.models.input.definitions.string_input_definition.**StringInputDefiniti**

Bases: *django_analyses.models.input.definitions.input_definition.InputDefinition*

Represents a single string input definition in the database.

choices
Possible choices for the string input value.

default
Default value.

dynamic_default
Dynamic default value expressed as some formatting template which requires run-dependent information.

get_type() → django_analyses.models.input.definitions.input_definitions.InputDefinitions

input_class
alias of *django_analyses.models.input.types.string_input.StringInput*

input_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

inputdefinition_ptr
Accessor to the related object on the forward side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Restaurant.place` is a `ForwardOneToOneDescriptor` instance.

inputdefinition_ptr_id

is_output_path
Whether this string determines the output path of the analysis.

is_output_switch
Whether this string determines if some output will be generated or not.

max_length
Maximal string length.

min_length
Minimal string length.

validate() → None
Overrides *django_analyses.models.input.definitions.input_definition.InputDefinition.validate()* to validate choices (in cases where *choices* is defined).

Raises `ValidationError` – Invalid choice

Module contents

InputDefinition subclasses are used to create an *InputSpecification* that may be associated with some *AnalysisVersion*. Each type of input definition is used describe some kind of input that could be passed to associated *AnalysisVersion*.

django_analyses.models.input.types package

Submodules

django_analyses.models.input.types.boolean_input module

class django_analyses.models.input.types.boolean_input.**BooleanInput**(*id*, *run*, *input_ptr*, *value*, *definition*)

Bases: *django_analyses.models.input.input.Input*

definition
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

definition_id

get_type() → django_analyses.models.input.types.input_types.InputTypes

input_ptr
Accessor to the related object on the forward side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Restaurant.place` is a `ForwardOneToOneDescriptor` instance.

input_ptr_id

value
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

django_analyses.models.input.types.directory_input module

class django_analyses.models.input.types.directory_input.**DirectoryInput**(*id*, *run*, *input_ptr*, *value*, *definition*)

Bases: *django_analyses.models.input.input.Input*

default_output_directory

definition
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

definition_id

fix_output_path() → str

get_type() → django_analyses.models.input.types.input_types.InputTypes

input_ptr
Accessor to the related object on the forward side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Restaurant.place` is a `ForwardOneToOneDescriptor` instance.

input_ptr_id

pre_save() → None
If this input class's *value* is a ForeignKey field, fix it in cases it is provided as int (the instance's primary key).

Note: This method may be overridden by subclasses to implement custom functionality before saving. However, be sure to include:

```
super().pre_save()
```

within your custom function.

---

required_path

value
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

django_analyses.models.input.types.file_input module

class django_analyses.models.input.types.file_input.**FileInput**(*id*, *run*, *input_ptr*, *value*, *definition*)

Bases: *django_analyses.models.input.input.Input*

definition
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

definition_id

get_type() → django_analyses.models.input.types.input_types.InputTypes

input_ptr
Accessor to the related object on the forward side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

Restaurant.place is a ForwardOneToOneDescriptor instance.

input_ptr_id

value
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

django_analyses.models.input.types.float_input module

class django_analyses.models.input.types.float_input.**FloatInput**(*id*, *run*, *input_ptr*, *numberinput_ptr*, *value*, *definition*)

Bases: *django_analyses.models.input.types.number_input.NumberInput*

definition

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

definition_id

get_type() → django_analyses.models.input.types.input_types.InputTypes

numberinput_ptr

Accessor to the related object on the forward side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Restaurant.place` is a `ForwardOneToOneDescriptor` instance.

numberinput_ptr_id

value

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

django_analyses.models.input.types.input_types module

class django_analyses.models.input.types.input_types.**InputTypes**

Bases: *django_analyses.utils.choice_enum.ChoiceEnum*

An enumeration.

BLN = 'Boolean'

DIR = 'Directory'

FIL = 'File'

FLT = 'Float'

INT = 'Integer'

LST = 'List'

STR = 'String'

django_analyses.models.input.types.integer_input module

class django_analyses.models.input.types.integer_input.**IntegerInput**(*id*, *run*, *input_ptr*, *numberinput_ptr*, *value*, *definition*)

Bases: *django_analyses.models.input.types.number_input.NumberInput*

definition

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

definition_id

get_type() → django_analyses.models.input.types.input_types.InputTypes

numberinput_ptr

Accessor to the related object on the forward side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Restaurant.place` is a `ForwardOneToOneDescriptor` instance.

numberinput_ptr_id

value

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

django_analyses.models.input.types.list_input module

class django_analyses.models.input.types.list_input.**ListInput**(*id*, *run*, *input_ptr*, *value*, *definition*)

Bases: *django_analyses.models.input.input.Input*

definition

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

definition_id

expected_type

expected_type_definition

get_argument_value()
Returns the input's *value* after applying any manipulations specified by the associated *definition*. This method is used to bridge database-compatible values with non-database-compatible values required by interfaces.

Returns Input value as expected by the interface

Return type Any

get_html_repr(*index: int*) → str

get_type() → django_analyses.models.input.types.input_types.InputTypes

input_ptr
Accessor to the related object on the forward side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Restaurant.place` is a `ForwardOneToOneDescriptor` instance.

input_ptr_id

query_related_instance() → django.db.models.query.QuerySet
If this input's definition points to a related model's field, returns the related instance (i.e. the instance in which the field's value is this input's value).

Returns related instance

Return type Any

raise_incorrect_type_error() → None

raise_max_length_error() → None

raise_min_length_error() → None

raise_not_list_error() → None

valid_elements

valid_max_length

valid_min_length

validate() → None
Run any custom validations before saving the model.

Note: This method may be overridden by subclasses to implement custom functionality before saving. However, be sure to include:

```
super().pre_save()
```

within your custom function.

---

classmethod validate_element_types(*value: list*, *expected_type: type*) → bool

validate_max_length() → bool

validate_min_length() → bool

value
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

django_analyses.models.input.types.number_input module

class django_analyses.models.input.types.number_input.**NumberInput**(*id*, *run*, *input_ptr*)

Bases: *django_analyses.models.input.input.Input*

floatinput
Accessor to the related object on the reverse side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Place.restaurant` is a `ReverseOneToOneDescriptor` instance.

input_ptr
Accessor to the related object on the forward side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Restaurant.place` is a `ForwardOneToOneDescriptor` instance.

input_ptr_id

integerinput
Accessor to the related object on the reverse side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Place.restaurant` is a `ReverseOneToOneDescriptor` instance.

raise_max_value_error() → None

raise_min_value_error() → None

valid_max_value

valid_min_value

validate() → None
Run any custom validations before saving the model.

Note: This method may be overridden by subclasses to implement custom functionality before saving. However, be sure to include:

```
super().pre_save()
```

within your custom function.

validate_max_value() → bool

validate_min_value() → bool

django_analyses.models.input.types.string_input module

class django_analyses.models.input.types.string_input.**StringInput**(*id*, *run*, *input_ptr*, *value*, *definition*)

Bases: *django_analyses.models.input.input.Input*

default_output_directory

default_value_formatting_dict

definition

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

definition_id

fix_output_path() → str

get_default_value_formatting_dict() → dict

get_type() → django_analyses.models.input.types.input_types.InputTypes

input_ptr

Accessor to the related object on the forward side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Restaurant.place` is a `ForwardOneToOneDescriptor` instance.

input_ptr_id

pre_save() → None

If this input class's *value* is a `ForeignKey` field, fix it in cases it is provided as `int` (the instance's primary key).

Note: This method may be overridden by subclasses to implement custom functionality before saving. However, be sure to include:

```
super().pre_save()
```

within your custom function.

raise_invalid_choice_error() → None

raise_max_length_error() → None

raise_min_length_error() → None

required_path

valid_choice

valid_max_length

valid_min_length

validate() → None
Run any custom validations before saving the model.

Note: This method may be overridden by subclasses to implement custom functionality before saving. However, be sure to include:

```
super().pre_save()
```

within your custom function.

validate_from_choices() → bool

validate_max_length() → bool

validate_min_length() → bool

value
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

Module contents

Submodules

django_analyses.models.input.input module

Definition of the base *Input* model.

class django_analyses.models.input.input.**Input**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

This model serves as a base class for different types of inputs.

argument_value
Returns the value of the input as expected by the interface.

Returns Input value as expected by the interface

Return type Any

definition = None

get_argument_value() → Any
Returns the input's *value* after applying any manipulations specified by the associated *definition*. This method is used to bridge database-compatible values with non-database-compatible values required by interfaces.

Returns Input value as expected by the interface

Return type Any

key
Returns the *key* of the associated *definition*.

Returns Input definition key

Return type str

objects = <model_utils.managers.InheritanceManager object>

pre_save() → None
If this input class's *value* is a ForeignKey field, fix it in cases it is provided as int (the instance's primary key).

Note: This method may be overridden by subclasses to implement custom functionality before saving. However, be sure to include:

```
super().pre_save()
```

within your custom function.

---

query_related_instance() → Any
If this input's definition points to a related model's field, returns the related instance (i.e. the instance in which the field's value is this input's value).

Returns related instance

Return type Any

raise_required_error()
Raises ValidationError to indicate this input is required.

Raises ValidationError – Required input not provided

run
The *Run* instance in which this input was included.

save(**args*, ***kwargs*)
Overrides the model's save() method to provide custom functionality.

validate() → None
Run any custom validations before saving the model.

Note: This method may be overridden by subclasses to implement custom functionality before saving. However, be sure to include:

```
super().pre_save()
```

within your custom function.

---

value = None
The *value* field is meant to be overriden by subclasses according to the type of data provided as input.

value_is_foreign_key
Checks whether the *value* attribute is a ForeignKey field.

Returns *value* is foreign key or not

Return type bool

django_analyses.models.input.input_specification module

Definition of the *InputSpecification* class.

class django_analyses.models.input.input_specification.**InputSpecification**(*id*, *created*, *modified*, *analysis*)

Bases: django_extensions.db.models.TimeStampedModel

analysis
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

analysis_version_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

base_input_definitions
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

Pizza.toppings and Topping.pizzas are ManyToManyDescriptor instances.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

configuration_keys

default_configuration

get_configuration_keys() → set

get_default_input_configurations() → dict

input_definitions

objects = <django_analyses.models.managers.input_specification.InputSpecificationManag

validate_keys(***kwargs*) → None

validate_kwargs(***kwargs*) → None

validate_required(***kwargs*) → None

django_analyses.models.input.utils module

class django_analyses.models.input.utils.**ListElementTypes**
Bases: *django_analyses.utils.choice_enum.ChoiceEnum*

An enumeration.

BLN = 'Boolean'

FIL = 'File'

FLT = 'Float'

INT = 'Integer'

STR = 'String'

Managers

Module contents

`Manager` subclasses for some of the app's models.

References

- Django's documentation on managers.

Submodules

django_analyses.models.managers.analysis module

Definition of a custom `Manager` for the *`Analysis`* class.

class django_analyses.models.managers.analysis.**AnalysisManager**
Bases: `django.db.models.manager.Manager`

Custom `Manager` for the *`Analysis`* class.

from_dict(*definition: dict*) → Tuple[django.db.models.base.Model, bool, bool]
Gets or creates an *`Analysis`* instance based on a dictionary definition.

Parameters **definition** (*dict*) – Analysis definition

Returns analysis, created, versions_created

Return type Tuple[models.Model, bool, bool]

See also:

- *Realistic Analysis Integration Example*

from_list (*definitions: list*) → Dict[str, Dict[KT, VT]]
Gets or creates `Analysis` instances using a list of analysis dictionary definitions.

Parameters **definition** (`list`) – Analysis definitions

Returns A dictionary with analysis titles as keys and analysis version dictionaries as values.

Return type Dict[str, Dict]

See also:

- *Realistic Analysis Integration Example*

django_analyses.models.managers.analysis_version module

Definition of a custom `Manager` for the `AnalysisVersion` class.

class django_analyses.models.managers.analysis_version.**AnalysisVersionManager**
Bases: `django.db.models.manager.Manager`

Custom `Manager` for the `AnalysisVersion` class.

from_dict (*analysis*, *definition: dict*)

from_list (*analysis*, *definitions: list*) → dict

get_by_string_id (*string_id: str*)
Gets an `AnalysisVersion` instance using a *string_id*.

There are two valid ways to specify the *string_id*:

- <*analysis title*>.<*analysis version title*> - Specific analysis version.
- <*analysis title*> - Latest analysis version (by descending title order).

Parameters **string_id** (`str`) – A string identifying some analysis version

Returns The matching analsis version

Return type `AnalysisVersion`

Raises `self.model.DoesNotExist` – Matching analysis version does not exist

get_kwargs_from_definition (*analysis*, *definition: dict*) → dict
Converts a dictionary definition of an anlysis version to valid keyword arguments that can be used to create a new instance.

Parameters

- **analysis** (`Analysis`) – The analysis to which the created instance will belong
- **definition** (`dict`) – Analysis version dictionary definition

Returns Keyword arguments

Return type dict

See also:

- *from_dict()*

django_analyses.models.managers.input_definition module

Definition of the *InputDefinitionManager* class.

class django_analyses.models.managers.input_definition.**InputDefinitionManager**
Bases: model_utils.managers.InheritanceManager

Custom manager for the *InputDefinition* model.

from_dict(*key: str*, *definition: dict*)
Creates an input definition (an instance of any subclass of the *InputDefinition* model) using the provided data. Expects *definition* to include a *type* key with the subclass (model) itself.

Parameters

- **key** (*str*) – Input definition key
- **definition** (*dict*) – Any other fields to populate

Returns Created input definition

Return type *InputDefinition*

from_specification_dict(*specification: dict*) → list
Creates multiple input definitions using some specification dictionary.

Parameters **specification** (*dict*) – A dictionary specification of input definitions

Returns Created input definitions

Return type list

django_analyses.models.managers.input_specification module

Definition of the *InputSpecificationManager* class.

class django_analyses.models.managers.input_specification.**InputSpecificationManager**
Bases: django.db.models.manager.Manager

Custom manager for the *InputSpecification* model.

filter_by_definitions(*analysis*, *definitions: list*) → django.db.models.query.QuerySet
Returns a queryset of input specifications that match the provided arguments.

Parameters

- **analysis** (Analysis) – The analysis with which the input specification is associated
- **definitions** (*list*) – Input definitions contained in the queried specification

Returns Matching input specifications

Return type QuerySet

from_dict(*analysis*, *specification: dict*) → tuple
Creates a new input specification from a dictionary of input definitions.

Parameters

- **analysis** (Analysis) – The analysis with which the input specification will be associated
- **specification** (*dict*) – Input specification dictionary

Returns

- *Tuple[~django_analyses.models.input.input_specification.InputSpecification,*
- *bool]* – Input specification, created

django_analyses.models.managers.output_definition module

class django_analyses.models.managers.output_definition.**OutputDefinitionManager**
Bases: model_utils.managers.InheritanceManager

from_specification_dict(*specification: dict*) → list

django_analyses.models.managers.output_specification module

class django_analyses.models.managers.output_specification.**OutputSpecificationManager**
Bases: django.db.models.manager.Manager

filter_by_definitions(*analysis*, *definitions: list*) → django.db.models.query.QuerySet

from_dict(*analysis*, *specification: dict*) → tuple

django_analyses.models.managers.run module

Definition of the *RunManager* class.

class django_analyses.models.managers.run.**RunManager**
Bases: django.db.models.manager.Manager

Manager for the *Run* model. Handles the creation and retrieval of runs when *Node* are executed.

create_and_execute(*analysis_version: django_analyses.models.analysis_version.AnalysisVersion*, *configuration: dict*, *user: django.contrib.auth.models.User = None*)
Execute *analysis_version* with the provided configuration (keyword arguments) and return the created run.

Parameters

- **analysis_version** (AnalysisVersion) – AnalysisVersion to execute
- **configuration** (*dict*) – Full input configuration (excluding default values)
- **user** (*User, optional*) – User who executed the run, by default None

Returns Resulting run instance

Return type *Run*

filter_by_configuration(*analysis_version: django_analyses.models.analysis_version.AnalysisVersion*, *configuration: Union[Dict[str, Any], Iterable[Dict[str, Any]]]*, *strict: bool = False*, *ignore_non_config: bool = False*) → django.db.models.query.QuerySet
Returns a queryset of *analysis_version* runs matching the provided *configuration*.

Parameters

- **analysis_version** (AnalysisVersion) – Analysis version runs to query

- **configuration** (*Union[Dict[str, Any], Iterable[Dict[str, Any]]]*) – Configuration options to filter by
- **strict** (*bool, optional*) – Whether to exclude runs with non-default value configurations not included in the provided *configuration*, by default False
- **ignore_non_config** (*bool, optional*) – Whether to exclude keys that match definitions for which the *is_configuration* attribute is set to False, by default False

Returns Matching runs

Return type models.QuerySet

get_existing(*analysis_version: django_analyses.models.analysis_version.AnalysisVersion*, *configuration: dict*)

Returns an existing run of the provided *analysis_version* with the specified *configuration*.

Parameters

- **analysis_version** (AnalysisVersion) – The desired AnalysisVersion instance for which a run is queried
- **configuration** (*dict*) – Full input configuration (excluding default values)

Returns Existing run with the specified *analysis_version* and *configuration*

Return type *Run*

Raises ObjectDoesNotExist – No matching run exists

get_or_execute(*analysis_version: django_analyses.models.analysis_version.AnalysisVersion*, *configuration: dict*, *user: django.contrib.auth.models.User = None*, *return_created: bool = False*)

Get or execute a run of *analysis_version* with the provided keyword arguments.

Parameters

- **analysis_version** (AnalysisVersion) – AnalysisVersion to retrieve or execute
- **configuration** (*dict*) – Full input configuration (excluding default values)
- **user** (*User, optional*) – User who executed the run, by default None, by default None
- **return_created** (*bool*) – Whether to also return a boolean indicating if the run already existed in the database or created, defaults to False

Returns New or existings run instance

Return type *Run*

Output

Module contents

Definition of all the models required to create an output specification for some *AnalysisVersion* and keep track of the outputs associated with some *Run* instance.

Subpackages

django_analyses.models.output.definitions package

Submodules

django_analyses.models.output.definitions.file_output_definition module

class django_analyses.models.output.definitions.file_output_definition.**FileOutputDefinitio**

Bases: *django_analyses.models.output.definitions.output_definition.OutputDefinition*

get_type() → str

output_class
alias of *django_analyses.models.output.types.file_output.FileOutput*

output_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

outputdefinition_ptr
Accessor to the related object on the forward side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Restaurant.place` is a `ForwardOneToOneDescriptor` instance.

outputdefinition_ptr_id

validate_existence
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

django_analyses.models.output.definitions.float_output_definition module

class django_analyses.models.output.definitions.float_output_definition.**FloatOutputDefiniti**

Bases: *django_analyses.models.output.definitions.output_definition.OutputDefinition*

get_type() → str

output_class
alias of *django_analyses.models.output.types.float_output.FloatOutput*

output_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

outputdefinition_ptr
Accessor to the related object on the forward side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Restaurant.place` is a `ForwardOneToOneDescriptor` instance.

outputdefinition_ptr_id

django_analyses.models.output.definitions.output_definition module

class django_analyses.models.output.definitions.output_definition.**OutputDefinition**(*id*, *key*, *description*)

Bases: `django.db.models.base.Model`

check_output_class_definition() → None

create_output_instance(**kwargs*) → django_analyses.models.output.output.Output

description
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

fileoutputdefinition
Accessor to the related object on the reverse side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Place.restaurant` is a `ReverseOneToOneDescriptor` instance.

floatoutputdefinition
Accessor to the related object on the reverse side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Place.restaurant` is a `ReverseOneToOneDescriptor` instance.

key
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

listoutputdefinition
Accessor to the related object on the reverse side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Place.restaurant` is a `ReverseOneToOneDescriptor` instance.

objects = <django_analyses.models.managers.output_definition.OutputDefinitionManager ob

output_class = None

pipe_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

pre_output_instance_create(*kwargs: dict*) → None

specification_set
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`Pizza.toppings` and `Topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

django_analyses.models.output.definitions.output_definitions module

class django_analyses.models.output.definitions.output_definitions.**OutputDefinitions**

Bases: `enum.Enum`

An enumeration.

FIL = 'File'

FLT = 'Float'

LST = 'List'

Module contents

django_analyses.models.output.types package

Submodules

django_analyses.models.output.types.file_output module

class django_analyses.models.output.types.file_output.**FileOutput**(*id*, *run*, *output_ptr*, *value*, *definition*)

Bases: *django_analyses.models.output.output.Output*

definition

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

definition_id

get_type() → str

output_ptr

Accessor to the related object on the forward side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Restaurant.place` is a `ForwardOneToOneDescriptor` instance.

output_ptr_id

pre_save() → None

raise_missing_output_error() → None

validate() → None

value
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

django_analyses.models.output.types.float_output module

class django_analyses.models.output.types.float_output.**FloatOutput**(*id*, *run*, *output_ptr*, *value*, *definition*)

Bases: *django_analyses.models.output.output.Output*

definition
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

definition_id

get_type() → str

output_ptr
Accessor to the related object on the forward side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Restaurant.place` is a `ForwardOneToOneDescriptor` instance.

output_ptr_id

value
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

django_analyses.models.output.types.output_types module

class django_analyses.models.output.types.output_types.**OutputTypes**
Bases: *django_analyses.utils.choice_enum.ChoiceEnum*

An enumeration.

FIL = 'File'

FLT = 'Float'

LST = 'List'

Module contents

Submodules

django_analyses.models.output.output module

class django_analyses.models.output.output.**Output**(*id*, *run*)
Bases: django.db.models.base.Model

definition = None

fileoutput
Accessor to the related object on the reverse side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

Place.restaurant is a ReverseOneToOneDescriptor instance.

floatoutput
Accessor to the related object on the reverse side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

Place.restaurant is a ReverseOneToOneDescriptor instance.

get_json_value() → Any

json_value

key

listoutput
Accessor to the related object on the reverse side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

Place.restaurant is a ReverseOneToOneDescriptor instance.

objects = <model_utils.managers.InheritanceManager object>

pre_save() → None

run
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

save(**args*, ***kwargs*)
Save the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

validate() → None

value = None

value_is_foreign_key

django_analyses.models.output.output_specification module

class django_analyses.models.output.output_specification.**OutputSpecification**(*id*, *created*, *modified*, *analysis*)

Bases: `django_extensions.db.models.TimeStampedModel`

analysis
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

analysis_version_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

base_output_definitions
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`Pizza.toppings` and `Topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

objects = <django_analyses.models.managers.output_specification.OutputSpecificationMana

output_definitions

Pipeline

Module contents

Definition of the *Node*, *Pipe*, and *Pipeline* models.

Submodules

django_analyses.models.pipeline.node module

Definition of the *Node* class.

class django_analyses.models.pipeline.node.**Node**(**args*, ***kwargs*)

Bases: `django_extensions.db.models.TimeStampedModel`

A `Model` representing a single pipeline node in the database. A node is simply a reference to some distinct configuration of a particular analysis version. Nodes are the building blocks of a *Pipeline*, and *Pipe* instances are used to attach them to one another.

analysis_version

The analysis version this node holds a configuration for.

configuration

The configuration of the analysis version ran when executing this node.

get_configuration() → dict

Undo any changes made to the node's configuration for serialization before passing them on to the interface.

Returns Node's analysis version configuration

Return type dict

get_full_configuration(*inputs: dict = None*) → dict

Returns a "full" input configuration, combining any provided inputs with this node's *configuration* and any default values configured in this analysis version's input specification.

Parameters **inputs** (*dict*) – User provided inputs for the execution of the node

Returns Full configuration to pass to the interface

Return type dict

get_required_nodes(*pipeline=None*, *run_index: int = None*) → django.db.models.query.QuerySet

Returns a queryset of *Node* instances that are a previous step in some *Pipeline* instance (i.e. there is a pipe in which the retuned nodes are the source and this node is the destination).

Parameters

- **pipeline** (*Pipeline*) – A pipeline instance to filter the pipe set with, optional
- **run_index** (*int*) – If this node is executed more than once in a given pipeline, filter by the index of the node's run, optional

Returns Required nodes

Return type models.QuerySet

get_requiring_nodes(*pipeline=None*, *run_index: int = None*) → django.db.models.query.QuerySet

Returns a queryset of *Node* instances that are the next step in some *Pipeline* instance (i.e. there is a pipe in which the retuned nodes are the destination and this node is the source).

Parameters

- **pipeline** (*Pipeline*) – A pipeline instance to filter the pipe set with, optional
- **run_index** (*int*) – If this node is executed more than once in a given pipeline, filter by the index of the node's run, optional

Returns Requiring nodes

Return type models.QuerySet

get_run_set() → django.db.models.query.QuerySet

Returns all the existing *Run* instances that match this node's *configuration* value.

Returns Existing node runs

Return type models.QuerySet

is_entry_node(*pipeline*) → bool

Determines whether this node is an entry point of the specified pipeline by checking if the first run of this node has any dependencies (i.e. has any pipes leading to it).

Parameters **pipeline** (*Pipeline*) – The pipeline to check

Returns Whether this node is an entry point of the given pipeline or not

Return type bool

objects = <django.db.models.manager.Manager object>

pipe_destination_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

pipe_source_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

required_nodes
Returns the queryset of required nodes returned by calling *get_required_nodes()*, or *None* if it is empty.

Returns Required nodes

Return type models.QuerySet

See also:

- *get_required_nodes()*

requiring_nodes
Returns the queryset of requiring nodes returned by calling *get_requiring_nodes()*, or *None* if it is empty.

Returns Requiring nodes

Return type models.QuerySet

See also:

- *get_requiring_nodes()*

run(*inputs: Union[Dict[str, Any], List[Dict[str, Any]], Tuple[Dict[str, Any]]], user: django.contrib.auth.models.User = None, return_created: bool = False*) → Union[django_analyses.models.run.Run, List[django_analyses.models.run.Run], Tuple[django_analyses.models.run.Run, bool], List[Tuple[django_analyses.models.run.Run, bool]]]
Run this node (the interface associated with this node's *analysis_version*) with the given inputs.

Parameters

- **inputs** (*dict*) – Any user-provided inputs to pass to the interface
- **user** (*User, optional*) – The user creating this run, by default None
- **return_created** (*bool*) – Whether to also return a boolean indicating if the run already existed in the database or created, defaults to False

Returns The created or retreived run instance/s

Return type Union[*Run*, List[*Run*], Tuple[*Run*, bool], List[Tuple[*Run*, bool]]]

save(**args*, ***kwargs*)
Overrides the model's `save()` method to provide custom validation.

Hint: For more information, see Django's documentation on overriding model methods.

validate() → None
Validates that the assigned configuration matches the chosen analysis version's input specification.

django_analyses.models.pipeline.pipe module

Definition of the *Pipe* class.

class django_analyses.models.pipeline.pipe.**Pipe**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

A Model representing a directed association between one *Node*'s output and another *Node*'s input within the context of a particular *Pipeline*.

base_destination_port
An input definition of the *destination* node that will be provided some input by the *source* node.

Note: This field holds the reference to the base *InputDefinition* instance.

See also:

- *destination_port*

base_source_port
An output definition of the *source* node that will provide some input of the *destination* node.

Note: This field holds the reference to the base *OutputDefinition* instance.

See also:

- *source_port*

destination
The *destination* *Node*, i.e. a node that will be provided an input from the *source* node.

destination_port
Returns the *InputDefinition* subclass of the assigned *base_destination_port*.

Returns The *destination* input definition

Return type *InputDefinition*

destination_run_index
If the destination node has multiple executions within the pipline, this attribute determines the index of the execution that will be used.

index
The *index* field is used to listify arguments in transit between nodes. An integer indicates expected input's index in the destination ListInput, and *None* indicates the index doesn't matter.

objects = <django_analyses.models.managers.pipe.PipeManager object>

pipeline
The *Pipeline* instance to which this pipe belongs.

source
The *source* *Node*, i.e. a node that will provide some input for the destination node.

source_port
Returns the *OutputDefinition* subclass of the assigned *base_source_port*.

Returns The *source* output definition

Return type *OutputDefinition*

source_run_index
If the source node has multiple executions within the pipline, this attribute determines the index of the execution that will be used.

django_analyses.models.pipeline.pipeline module

Definition of the *Pipeline* class.

class django_analyses.models.pipeline.pipeline.**Pipeline**(**args*, ***kwargs*)
Bases: django_extensions.db.models.TitleDescriptionModel, django_extensions.db.models.TimeStampedModel

A pipeline essentially represents a set of *Pipe* instances constituting a distinct analysis procedure.

class Meat
Bases: object

ordering = ('title',)

count_node_runs(*node: django_analyses.models.pipeline.node.Node*) → int
Returns the number of times a particular node is meant to run during the execution of this pipeline.

Parameters **node** (Node) – Node to count

Returns Number of separate runs of *node* within this pipeline

Return type int

entry_nodes
Returns the "entry" node/s of this pipeline.

Returns Entry nodes

Return type list

See also:

- *get_entry_nodes()*

get_entry_nodes() → list
Returns the "entry" node/s of this pipeline, i.e. nodes that are a *source* of some *Pipe* but not a *destination* in any.

Returns List of *Node* instances

Return type list

get_node_set() → django.db.models.query.QuerySet
Returns all *Node* instances used in this pipeline.

Returns Pipeline nodes

Return type QuerySet

node_set
Returns all *Node* instances used in this pipeline.

Returns Pipeline nodes

Return type QuerySet

See also:

- *get_node_set()*

objects = <django_analyses.models.managers.pipeline.PipelineManager object>

pipe_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

Submodules

django_analyses.models.analysis module

Definition of the *Analysis* class.

class django_analyses.models.analysis.**Analysis**(**args*, ***kwargs*)
Bases: `django_extensions.db.models.TitleDescriptionModel`, `django_extensions.db.models.TimeStampedModel`

A `Model` representing a single analysis in the database.

category
A *Category* instance associated with this analysis.

inputspecification_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

objects = <django_analyses.models.managers.analysis.AnalysisManager object>

outputspecification_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

version_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

django_analyses.models.analysis_version module

Definition of the *AnalysisVersion* class.

class django_analyses.models.analysis_version.**AnalysisVersion**(**args*, ***kwargs*)
Bases: `django_extensions.db.models.TitleDescriptionModel`, `django_extensions.db.models.TimeStampedModel`

A `Model` representing a single analysis version in the database.

Each *AnalysisVersion* instance should be assigned an interface through the project's `ANALYSIS_INTERFACES` setting (for more information see *Interface Integration* and *Integration Customization*).

analysis
The *Analysis* instance to which this analysis version belongs.

extract_results(*results: Any*) → dict
Extracts a results dictionary from an arbitrary results object in case the *nested_results_attribute* is not *None*.

Parameters **results** (*Any*) – Arbitrary results object

Returns Results dictionary

Return type dict

fixed_run_method_kwargs
Any "fixed" keyword arguments that should always be passed to the interface's *run* method at execution.

get_interface() → object
Queries the project's settings to locate the instance's interface. For more information see *Interface Integration*.

Returns Interface class used to run this version of the analysis

Return type `object`

Raises `NotImplementedError` – No interface could be found for this analysis

get_interface_initialization_kwargs(***kwargs*) → dict
Returns the parameters required at the interface's class initialization.

Returns Initialization parameters as a keyword arguments dict

Return type dict

get_run_method_kwargs(***kwargs*) → dict
Returns the parameters required when calling the interface's *run()* method.

Returns *run()* method parameters as a keyword arguments dict

Return type dict

input_definitions
Returns the associated instance's *InputDefinition* subclasses as defined in its *input_specification*.

Returns *InputDefinition* subclasses

Return type QuerySet

input_specification
The *InputSpecification* instance specifying the *InputDefinition* subclasses associated with this analysis version.

interface
Returns the associated interface for this instance.

Returns Analysis interface class

Return type type

max_parallel
Maximal number of parallel executions that may be run using Celery. This attribute is used in `execute_node()` to chunk an iterable of node inputs in case it is longer than this value. For more information see Celery's Chunks documentation.

nested_results_attribute
Analysis interfaces are expected to return a dictionary of the results. In case this analysis version's interface returns some object which conatins the desired dictionary, this field allows specifying the attribute or method that may be used to retrieve it.

Example

Nipype's interfaces generally return some kind of `InterfaceResults` object with an `outputs` attribute that may be used to create a dictionary of the results by calling the `get_traitsfree()` method. In order to integrate smoothly with Nipype's interfaces, we could simply specify *nested_results_attribute="outputs.get_traitsfree"* when creating the appropriate analysis versions.

nested_results_parts
Splits the *nested_results_attribute* at "." indices in case of a deeply nested attribute.

Returns Listed parts of nested result dictionary location

Return type list

node_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

objects = <django_analyses.models.managers.analysis_version.AnalysisVersionManager obj

output_definitions
Returns the associated instance's *OutputDefinition* subclasses as defined in its *output_specification*.

Returns *OutputDefinition* subclasses

Return type QuerySet

output_specification
The *OutputSpecification* instance specifying the *OutputDefinition* subclasses associated with this analysis version.

run(***kwargs*) → dict
Runs the interface safely by validating the input according to the instance's *input_specification* and applying any special integration customizations (for more information see *Integration Customization*).

Returns Results dictionary

Return type dict

run_interface(***kwargs*) → dict
Call the interface class's *run()* method with the given keyword arguments.

Returns Dictionary of results

Return type dict

run_method_key
Custom *run* method name for the interface. Each analysis version is expected to have some class associated with it and used as an interface for running the analysis. This field determines the name of the method that will be called (default value is *"run"*).

run_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

update_input_with_defaults(*configuration: dict*) → dict
Updates a configuration specified as keyword arguments with the instance's *input_specification* defaults.

Parameters **configuration** (*dict*) – Input configuration (excluding default values)

Returns Configuration updated with default values

Return type dict

django_analyses.models.category module

Definition of the *Category* class.

class django_analyses.models.category.**Category**(**args*, ***kwargs*)
Bases: django_extensions.db.models.TitleDescriptionModel, django_extensions.db.models.TimeStampedModel

A `Model` representing a category of analyses in the databse.

analysis_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

objects = <django.db.models.manager.Manager object>

parent
If this is a nested category, this field holds the association with the parent.

subcategories
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

django_analyses.models.run module

Definition of the *Run* model.

class django_analyses.models.run.**Run**(**args*, ***kwargs*)
Bases: `django_extensions.db.models.TimeStampedModel`

`Model` representing a single analysis version's run in the database.

analysis_version
The *AnalysisVersion* that was run.

base_input_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

base_output_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

check_null_configuration() → bool
Checks whether this run's configuration is equivalent to the input specification's default settings.

Returns Whether this run has only default configuration settings

Return type bool

duration
Returns the time delta between this instance's *end_time* and *start_time*. If *end_time* isn't set yet returns time since *start_time*.

Returns Run duration

Return type datetime.timedelta

end_time
Run end time.

fix_input_value(*inpt*) → Any
Reverts changes that may have been made to a given input in order to return the "raw" input configuration of this run (i.e. the input as it was passed by the user).

Parameters **inpt** (*Input*) – An input of this run

Returns The "raw" input value

Return type Any

get_input(*key: str*) → Any
Returns a particular output created in this run according to its definition's `key` field.

Parameters **key** (*str*) – The desired *Input*'s associated definition keyword

Returns Input value

Return type Any

get_input_configuration(*include_non_configuration: bool = True*, *include_defaults: bool = True*) → dict
Returns the input configuration of this run.

Parameters

- **include_non_configuration** (*bool*) – Whether to include inputs for which the associated definition's `is_configuration` attribute is set to False, default is True
- **include_defaults** (*bool*) – Whether to include default input configuration values, default is True

Returns Input configuration

Return type dict

get_input_set() → model_utils.managers.InheritanceQuerySet
Returns the *Input* subclasses' instances created for this execution of the *analysis_version*.

Returns This run's inputs

Return type `InheritanceQuerySet`

get_output(*key: str*) → Any
Returns a particular output created in this run according to its definition's `key` field.

Parameters **key** (*str*) – The desired *Output*'s associated definition keyword

Returns Output value

Return type Any

get_output_configuration() → dict
Returns the output configuration of this run.

Returns Output configuration

Return type dict

get_output_parser() → object

get_output_set() → model_utils.managers.InheritanceQuerySet
Returns the *Output* subclasses' instances created on this execution of the *analysis_version*.

Returns This run's outputs

Return type `InheritanceQuerySet`

get_raw_input_configuration() → dict
Returns the "raw" configuration of this run. As some inputs may have been transformed, this method is used to reverse these changes in case this run's parameters are compared in the future to new input parameters.

Returns Raw input configuration as provided by the user

Return type dict

get_results_json() → dict
Returns a JSON serializable dictionary of the results.

Returns JSON serializable output dictionary

Return type dict

get_status_display(*, *field=<django.db.models.fields.CharField: status>*)

get_visualizer(*provider: str = None*) → callable

input_configuration
Returns a dictionary with the full input configuration of this run.

Returns Full input configuration

Return type dict

See also:

- *get_input_configuration()*

input_defaults
Returns the default configuration parameters according to this instance's *InputSpecification*'s *default_configuration* property.

Returns This analysis version's default input configuration

Return type dict

input_set
Returns the *Input* subclasses' instances created for this run.

Returns This run's inputs

Return type QuerySet

See also:

- *get_input_set()*

objects = <django_analyses.models.managers.run.RunManager object>

output_configuration
Returns a dictionary of the output configuration of this run.

Returns Output configuration

Return type dict

See also:

- *get_output_configuration()*

output_parser

output_set
Returns the *Output* subclasses' instances created by this run.

Returns This run's outputs

Return type QuerySet

See also:

- *get_output_set()*

parse_output()

path
Retruns the default Path for any artifacts created by this run.

Returns Run artifacts directory under MEDIA_ROOT.

Return type pathlib.Path

raw_input_configuration
Returns a dictionary of this run's raw input configuration.

Returns This run's raw input configuration

Return type dict

See also:

- *get_raw_input_configuration()*

start_time
Run start time.

status
The status of this run.

task_result
The `TaskResult` instance associated with this run.

task_result_id

traceback
Traceback saved in case of run failure.

user
The user who created this run.

visualize() → None

4.2.3 Runner

Module contents

Submodules

django_analyses.runner.queryset_runner module

Definition of the *QuerySetRunner* class.

class django_analyses.runner.queryset_runner.**QuerySetRunner**
Bases: `object`

Base class for batch queryset processing.

Example

For a general usage example, see the *QuerySet Processing* section in the documentation.

ANALYSIS_CONFIGURATION = None
Common input specification dictionary as it is expected to be defined in the required *Node*'s *configuration* field. If none is provided, defaults to _NO_CONFIGURATION.

ANALYSIS_TITLE = ''
Required *Analysis* instance title.

ANALYSIS_VERSION_TITLE = ''
Required `AnalysisVersion` instance title.

BASE_QUERY = None
Query to use when retrieving the base queryset.

See also:

- *get_base_queryset()*

BASE_QUERY_END = '{n_instances} instances found.'

BASE_QUERY_START = 'Querying {model_name} instances...'

BATCH_RUN_START = '\x1b[4m\x1b[95m\x1b[1m{analysis_version}\x1b[0m\x1b[4m\x1b[95m: Ba

DATA_MODEL = None
QuerySet model.

DEFAULT_QUERYSET_QUERY = '\n\x1b[94m Default execution queryset generation:\x1b[0m'

EXECUTION_STARTED = '\n\x1b[92mSuccessfully started {analysis_version} execution over

FILTER_QUERYSET_END = '{n_candidates} execution candidates found.'

FILTER_QUERYSET_START = 'Filtering queryset...'

INPUT_GENERATION = '\n \x1b[94mGenerating input specifications:\x1b[0m'

INPUT_GENERATION_FINISHED = '{n_inputs} input specifications prepared.'

INPUT_GENERATION_PROGRESSBAR_KWARGS = {'desc': 'Preparing inputs', 'unit': 'instance
A dictionary used for tqdm progressbar customization during input generation.

INPUT_KEY = ''
The associated `AnalysisVersion` instance's *`InputDefinition`* which will be used to query pending runs and execute them.

INPUT_QUERYSET_VALIDATION = '\n\x1b[94m Input queryset validation:\x1b[0m'

INPUT_QUERY_END = '{n_existing} runs found.'

INPUT_QUERY_START = 'Querying existing runs...'

NONE_PENDING = '\x1b[92mCongratulations! No pending {model_name} instances were detecte

NONE_PENDING_IN_QUERYSET = '\x1b[92mAll {n_instances} provided {model_name} instances

NO_CANDIDATES = '\x1b[93mNo execution candidates detected in {model_name} queryset!\x1

PENDING_FOUND = '{n_existing} existing runs found.\n\x1b[1m{n_pending}\x1b[0m instance

PENDING_QUERY_START = '\n \x1b[94mChecking execution status for the {queryset_descript

PREPROCESSING_FAILURE = '\x1b[93mFailed to preprocess {model_name} #{instance_id}!\x1b

PREPROCESSING_FAILURE_REPORT = '\x1b[93m\x1b[1m{n_invalid} of {n_total} {model_name} i

STATUS_QUERY_PROGRESSBAR_KWARGS = {'desc': None, 'unit': 'instance'}
A dictionary used for tqdm progressbar customization during input generation.

analysis
Returns the required analysis.

Returns Analysis to be executed

Return type *Analysis*

See also:

query_analysis()

analysis_version
Returns the required analysis version.

Returns Analysis version to be executed

Return type *AnalysisVersion*

See also:

query_analysis_version()

configuration
Returns the configuration dictionary for the execution node.

Returns Node configuration

Return type dict

See also:

create_configuration()

create_configuration() → dict
Returns the configuration dictionary for the execution node.

Returns Node configuration

Return type dict

create_input_specification(*instance: django.db.models.base.Model*) → dict
Returns an input specification dictionary with the given data *instance* as input.

Parameters **instance** (*Model*) – Data instance to be processed

Returns Input specification dictionary

Return type dict

See also:

create_inputs()

create_inputs(*queryset: django.db.models.query.QuerySet*, *progressbar: bool = True*, *max_total: int = None*) → List[Dict[str, List[str]]]
Returns a list of dictionary input specifications.

Parameters

- **instances** (*QuerySet*) – Batch of instances to run the analysis over
- **progressbar** (*bool, optional*) – Whether to display a progressbar, by default True

Returns Input specifications

Return type List[Dict[str, List[str]]]

See also:

create_input_specification()

evaluate_queryset(*queryset: django.db.models.query.QuerySet*, *apply_filter: bool = True*, *log_level: int = 20*) → django.db.models.query.QuerySet
Evaluates a provided queryset by applying any required filters or generating the default queryset (if `None`).

Parameters

- **queryset** (*QuerySet*) – Provided queryset
- **apply_filter** (*bool*) – Whether to pass the queryset through *filter_queryset()* or not
- **log_level** (*int, optional*) – Logging level to use, by default 20 (INFO)

Returns Evaluated execution queryset

Return type QuerySet

See also:

filter_queryset()

filter_queryset(*queryset: django.db.models.query.QuerySet*, *log_level: int = 20*) → django.db.models.query.QuerySet
Applies any custom filtering to the a given data model's queryset.

Parameters

- **queryset** (*QuerySet*) – A collection of the data model's instances
- **log_level** (*int, optional*) – Logging level to use, by default 20 (INFO)

Returns Fitered queryset

Return type QuerySet

get_base_queryset (*log_level: int = 20*) → django.db.models.query.QuerySet
Returns the base queryset of the data model's instances.

Parameters **log_level** (*int, optional*) – Logging level to use, by default 20 (INFO)

Returns All data model instances

Return type QuerySet

get_instance_representation (*instance: django.db.models.base.Model*) → Any
Returns the representation of a single instance from the queryset as an *Input* instance's *value*.

Parameters **instance** (*Model*) – Instance to be represented as input value

Returns Input value

Return type Any

get_or_create_node () → django_analyses.models.pipeline.node.Node
Get or create the required execution node according to the specified analysis configuration.

Returns Execution node

Return type *Node*

has_run (*instance: django.db.models.base.Model*) → bool
Check whether the provided *instance* has an existing run in the databaes or not.

Parameters **instance** (*Model*) – Data instance to check

Returns Whether the data instance has an existing run or not

Return type bool

input_definition
Returns the data instance's matching input definition.

Returns Data instance input definition

Return type *InputDefinition*

See also:

query_input_definition()

input_set
Returns a queryset of existing *Input* instances for the executed node.

Returns Existing inputs

Return type QuerySet

See also:

query_input_set()

log_base_query_end (*n_instances: int*, *log_level: int = 20*) → None
Logs the result of querying the base queryset.

Parameters **log_level** (*int, optional*) – Logging level to use, by default 20 (INFO)

log_base_query_start(*log_level: int = 20*) → None
Logs the generation of the base queryset.

Parameters **log_level** (*int, optional*) – Logging level to use, by default 20 (INFO)

log_execution_start(*n_instances: int*, *log_level: int = 20*) → None
Log the start of a batch execution over some queryset.

Parameters

- **n_instances** (*int*) – Number of instances in the queryset
- **log_level** (*int, optional*) – Logging level to use, by default 20 (INFO)

See also:

run()

log_filter_end(*n_candidates: int*, *log_level: int = 20*) → None
Logs the result of queryset filtering prior to execution.

Parameters

- **n_candidates** (*int*) – Number of execution candidates in queryset after filtering
- **log_level** (*int, optional*) – Logging level to use, by default 20 (INFO)

log_filter_start(*log_level: int = 20*) → None
Logs the beginning of queryset filtering prior to execution.

Parameters **log_level** (*int, optional*) – Logging level to use, by default 20 (INFO)

log_none_pending(*queryset: django.db.models.query.QuerySet*, *log_level: int = 20*) → None
Log an empty queryset of pending instances.

Parameters

- **queryset** (*QuerySet*) – Provided or generated execution queryset
- **log_level** (*int, optional*) – Logging level to use, by default 20 (INFO)

See also:

query_progress()

log_pending(*existing: django.db.models.query.QuerySet*, *pending: django.db.models.query.QuerySet*, *log_level: int = 20*) → None
Log the number of pending vs. existing instances.

Parameters

- **existing** (*QuerySet*) – Instances with existing runs
- **pending** (*QuerySet*) – Instances pending execution
- **log_level** (*int, optional*) – Logging level to use, by default 20 (INFO)

log_progress_query_end(*queryset: django.db.models.query.QuerySet*, *existing: django.db.models.query.QuerySet*, *pending: django.db.models.query.QuerySet*, *log_level: int = 20*) → None
Logs the execution progress query's result.

Parameters

- **queryset** (*QuerySet*) – Full execution queryset
- **existing** (*QuerySet*) – Instances with existing reults
- **pending** (*QuerySet*) – Instances pending execution

- **log_level** (*int, optional*) – Logging level to use, by default 20 (INFO)

See also:

query_progress()

log_progress_query_start (*log_level: int = 20*) → None
Logs the beginning of queryset filtering prior to execution.

Parameters **log_level** (*int, optional*) – Logging level to use, by default 20 (INFO)

See also:

query_progress()

log_run_start (*log_level: int = 20*) → None
Logs the beginning of the *run()* method's execution.

Parameters **log_level** (*int, optional*) – Logging level to use, by default 20 (INFO)

See also:

run()

node
Returns the required execution node.

Returns Node to be executed

Return type *Node*

See also:

get_or_create_node()

query_analysis () → django_analyses.models.analysis.Analysis
Returns the analysis to be executed.

Returns Executed analysis

Return type *Analysis*

query_analysis_version () → django_analyses.models.analysis_version.AnalysisVersion
Returns the analysis version to be executed.

Returns Executed analysis version

Return type *AnalysisVersion*

query_input_definition () → django_analyses.models.input.definitions.input_definition.InputDefinition
Returns the input definition which corresponds to the queryset instances.

Returns Instance input definition

Return type *InputDefinition*

query_input_set (*log_level: int = 10*) → django.db.models.query.QuerySet
Returns a queryset of existing *Input* instances of the execution node.

Parameters **log_level** (*int, optional*) – Logging level to use, by default 20 (INFO)

Returns Existing inputs

Return type QuerySet

query_progress(*queryset: django.db.models.query.QuerySet = None*, *apply_filter: bool = True*, *log_level: int = 20*, *progressbar: bool = True*) → Tuple[django.db.models.query.QuerySet, django.db.models.query.QuerySet]

Splits *queryset* to instances with and without existing runs. If no *queryset* is provided, generates the default execution queryset.

Parameters

- **queryset** (*QuerySet, optional*) – Queryset to split by run status, by default None
- **apply_filter** (*bool*) – Whether to pass the queryset through *filter_queryset()* or not
- **log_level** (*int, optional*) – Logging level to use, by default 20 (INFO)
- **progressbar** (*bool, optional*) – Whether to display a progressbar, by default True

Returns Existing, Pending

Return type Tuple[QuerySet, QuerySet]

run(*queryset: django.db.models.query.QuerySet = None*, *max_total: int = None*, *prep_progressbar: bool = True*, *log_level: int = 20*, *dry: bool = False*)

Execute this class's *node* in batch over all data instances in *queryset*. If none provided, queries a default execution queryset.

Parameters

- **queryset** (*QuerySet, optional*) – Queryset to run, by default None
- **max_total** (*int, optional*) – Maximal total number of runs, by default None
- **prep_progressbar** (*bool, optional*) – Whether to display a progressbar for input generation, by default True
- **log_level** (*int, optional*) – Logging level to use, by default 20 (INFO)
- **dry** (*bool, optional*) – Whether this is a dry run (no execution) or not, by default False

4.2.4 Serializers

Subpackages

django_analyses.serializers.input package

Subpackages

django_analyses.serializers.input.definitions package

Submodules

django_analyses.serializers.input.definitions.boolean_input_definition module

class django_analyses.serializers.input.definitions.boolean_input_definition.**BooleanInputDe**

Bases: rest_framework.serializers.ModelSerializer

class Meta

Bases: object

fields = '__all__'

model

alias of *django_analyses.models.input.definitions.boolean_input_definition.BooleanInputDefinition*

django_analyses.serializers.input.definitions.directory_input_definition module

class django_analyses.serializers.input.definitions.directory_input_definition.**DirectoryInp**

Bases: rest_framework.serializers.ModelSerializer

class Meta

Bases: object

fields = '__all__'

model

alias of *django_analyses.models.input.definitions.directory_input_definition.DirectoryInputDefinition*

django_analyses.serializers.input.definitions.file_input_definition module

class django_analyses.serializers.input.definitions.file_input_definition.**FileInputDefinit**

Bases: rest_framework.serializers.ModelSerializer

class Meta

Bases: object

fields = '__all__'

model

alias of *django_analyses.models.input.definitions.file_input_definition.FileInputDefinition*

django_analyses.serializers.input.definitions.float_input_definition module

class django_analyses.serializers.input.definitions.float_input_definition.**FloatInputDefini**

Bases: rest_framework.serializers.ModelSerializer

class Meta

Bases: object

fields = '__all__'

model

alias of *django_analyses.models.input.definitions.float_input_definition.FloatInputDefinition*

django_analyses.serializers.input.definitions.input_definition module

class django_analyses.serializers.input.definitions.input_definition.**InputDefinitionSeriali**

Bases: *django_analyses.serializers.utils.polymorphic.PolymorphicSerializer*

class Meta

Bases: object

fields = '__all__'

model

alias of *django_analyses.models.input.definitions.input_definition.InputDefinition*

get_serializer(*input_type: str*) → rest_framework.serializers.Serializer

Return a dict to map class names to their respective serializer classes. To be implemented by all PolymorphicSerializer subclasses.

django_analyses.serializers.input.definitions.input_definition.**get_extra_input_definition_s**

django_analyses.serializers.input.definitions.integer_input_definition module

class django_analyses.serializers.input.definitions.integer_input_definition.**IntegerInputDe**

Bases: rest_framework.serializers.ModelSerializer

class Meta

Bases: object

fields = '__all__'

model

alias of *django_analyses.models.input.definitions.integer_input_definition.IntegerInputDefinition*

django_analyses.serializers.input.definitions.list_input_definition module

class django_analyses.serializers.input.definitions.list_input_definition.**ListInputDefinit**

Bases: rest_framework.serializers.ModelSerializer

class Meta

Bases: object

fields = '__all__'

model

alias of *django_analyses.models.input.definitions.list_input_definition.ListInputDefinition*

django_analyses.serializers.input.definitions.string_input_definition module

class django_analyses.serializers.input.definitions.string_input_definition.**StringInputDef**

Bases: rest_framework.serializers.ModelSerializer

class Meta

Bases: object

fields = '__all__'

model

alias of *django_analyses.models.input.definitions.string_input_definition.StringInputDefinition*

Module contents

django_analyses.serializers.input.types package

Submodules

django_analyses.serializers.input.types.boolean_input module

class django_analyses.serializers.input.types.boolean_input.**BooleanInputSerializer**(*instance=N*
data=<cla
'rest_frame
***kwargs*)

Bases: rest_framework.serializers.ModelSerializer

class Meta

Bases: object

fields = ('id', 'key', 'value', 'run', 'definition')

model

alias of *django_analyses.models.input.types.boolean_input.BooleanInput*

django_analyses.serializers.input.types.directory_input module

class django_analyses.serializers.input.types.directory_input.**DirectoryInputSerializer**(*insta*
data=
*'rest_*
***kw*

Bases: rest_framework.serializers.ModelSerializer

class Meta

Bases: object

fields = ('id', 'key', 'value', 'run', 'definition')

model

alias of *django_analyses.models.input.types.directory_input.DirectoryInput*

django_analyses.serializers.input.types.file_input module

class django_analyses.serializers.input.types.file_input.**FileInputSerializer**(*instance=None*,
data=<class
'rest_framework.fiel
***kwargs*)

Bases: rest_framework.serializers.ModelSerializer

class Meta

Bases: object

fields = ('id', 'key', 'value', 'run', 'definition')

model

alias of *django_analyses.models.input.types.file_input.FileInput*

django_analyses.serializers.input.types.float_input module

class django_analyses.serializers.input.types.float_input.**FloatInputSerializer**(*instance=None*,
data=<class
'rest_framework.
***kwargs*)

Bases: rest_framework.serializers.ModelSerializer

class Meta

Bases: object

fields = ('id', 'key', 'value', 'run', 'definition')

model

alias of *django_analyses.models.input.types.float_input.FloatInput*

django_analyses.serializers.input.types.integer_input module

class django_analyses.serializers.input.types.integer_input.**IntegerInputSerializer**(*instance=N*
data=<cla
'rest_frame
***kwargs*)

Bases: rest_framework.serializers.ModelSerializer

class Meta
Bases: object

fields = ('id', 'key', 'value', 'run', 'definition')

model
alias of *django_analyses.models.input.types.integer_input.IntegerInput*

django_analyses.serializers.input.types.list_input module

class django_analyses.serializers.input.types.list_input.**ListInputSerializer**(*instance=None*, *data=<class 'rest_framework.fiel*, ***kwargs*)

Bases: rest_framework.serializers.ModelSerializer

class Meta
Bases: object

fields = ('id', 'key', 'value', 'run', 'definition')

model
alias of *django_analyses.models.input.types.list_input.ListInput*

django_analyses.serializers.input.types.string_input module

class django_analyses.serializers.input.types.string_input.**StringInputSerializer**(*instance=Non*, *data=<class 'rest_framewo*, ***kwargs*)

Bases: rest_framework.serializers.ModelSerializer

class Meta
Bases: object

fields = ('id', 'key', 'value', 'run', 'definition')

model
alias of *django_analyses.models.input.types.string_input.StringInput*

Module contents

Submodules

django_analyses.serializers.input.input module

class django_analyses.serializers.input.input.**InputSerializer**(*instance=None*, *data=<class 'rest_framework.fields.empty'>*, ***kwargs*)

Bases: *django_analyses.serializers.utils.polymorphic.PolymorphicSerializer*

class Meta
Bases: object

fields = '__all__'

model
alias of *django_analyses.models.input.input.Input*

get_serializer(*input_type: str*) → rest_framework.serializers.Serializer
Return a dict to map class names to their respective serializer classes. To be implemented by all PolymorphicSerializer subclasses.

django_analyses.serializers.input.input.**get_extra_input_serializers**() → dict

django_analyses.serializers.input.input_specification module

class django_analyses.serializers.input.input_specification.**InputSpecificationSerializer**(*in*
da
're
***

Bases: rest_framework.serializers.HyperlinkedModelSerializer

class Meta
Bases: object

fields = ('id', 'analysis', 'created', 'modified', 'input_definitions_count')

model
alias of *django_analyses.models.input.input_specification.InputSpecification*

Module contents

django_analyses.serializers.output package

Subpackages

django_analyses.serializers.output.definitions package

Submodules

django_analyses.serializers.output.definitions.file_output_definition module

class django_analyses.serializers.output.definitions.file_output_definition.**FileOutputDefi**

Bases: rest_framework.serializers.ModelSerializer

class Meta
Bases: object

fields = '__all__'

model
alias of *django_analyses.models.output.definitions.file_output_definition.FileOutputDefinition*

django_analyses.serializers.output.definitions.output_definition module

class django_analyses.serializers.output.definitions.output_definition.**OutputDefinitionSeri**

Bases: *django_analyses.serializers.utils.polymorphic.PolymorphicSerializer*

class Meta

Bases: object

fields = '__all__'

model

alias of *django_analyses.models.output.definitions.output_definition.OutputDefinition*

get_serializer(*output_type: str*) → rest_framework.serializers.Serializer

Return a dict to map class names to their respective serializer classes. To be implemented by all PolymorphicSerializer subclasses.

django_analyses.serializers.output.definitions.output_definition.**get_extra_output_definitio**

Module contents

django_analyses.serializers.output.types package

Submodules

django_analyses.serializers.output.types.file_output module

class django_analyses.serializers.output.types.file_output.**FileOutputSerializer**(*instance=None*, *data=<class 'rest_framework.**kwargs*)

Bases: rest_framework.serializers.ModelSerializer

class Meta

Bases: object

fields = ('id', 'key', 'value', 'run', 'definition')

model

alias of *django_analyses.models.output.types.file_output.FileOutput*

Module contents

Submodules

django_analyses.serializers.output.output module

class django_analyses.serializers.output.output.**OutputSerializer**(*instance=None*, *data=<class 'rest_framework.fields.empty'>*, ***kwargs*)

Bases: *django_analyses.serializers.utils.polymorphic.PolymorphicSerializer*

class Meta

Bases: object

fields = '__all__'

model

alias of *django_analyses.models.output.output.Output*

get_serializer(*output_type: str*) → rest_framework.serializers.Serializer

Return a dict to map class names to their respective serializer classes. To be implemented by all PolymorphicSerializer subclasses.

django_analyses.serializers.output.output.**get_extra_output_serializers**() → dict

django_analyses.serializers.output.output_specification module

class django_analyses.serializers.output.output_specification.**OutputSpecificationSerializer**

Bases: rest_framework.serializers.HyperlinkedModelSerializer

class Meta

Bases: object

fields = ('id', 'analysis', 'created', 'modified', 'output_definitions_count')

model

alias of *django_analyses.models.output.output_specification.OutputSpecification*

Module contents

django_analyses.serializers.pipeline package

Submodules

django_analyses.serializers.pipeline.node module

class django_analyses.serializers.pipeline.node.**NodeSerializer**(*instance=None*, *data=<class 'rest_framework.fields.empty'>*, ***kwargs*)

Bases: rest_framework.serializers.HyperlinkedModelSerializer

class Meta

Bases: object

fields = ('id', 'analysis_version', 'configuration', 'created', 'modified', 'url')

model
alias of *django_analyses.models.pipeline.node.Node*

django_analyses.serializers.pipeline.pipe module

class django_analyses.serializers.pipeline.pipe.**PipeSerializer**(*instance=None*, *data=<class 'rest_framework.fields.empty'>*, ***kwargs*)

Bases: rest_framework.serializers.HyperlinkedModelSerializer

class Meta
Bases: object

fields = ('id', 'pipeline', 'source', 'source_port', 'destination', 'destination_po

model
alias of *django_analyses.models.pipeline.pipe.Pipe*

django_analyses.serializers.pipeline.pipeline module

class django_analyses.serializers.pipeline.pipeline.**PipelineSerializer**(*instance=None*, *data=<class 'rest_framework.fields.empty*, ***kwargs*)

Bases: rest_framework.serializers.HyperlinkedModelSerializer

class Meta
Bases: object

fields = ('id', 'title', 'description', 'node_set', 'pipe_set', 'created', 'modifie

model
alias of *django_analyses.models.pipeline.pipeline.Pipeline*

Module contents

django_analyses.serializers.utils package

Submodules

django_analyses.serializers.utils.polymorphic module

Polymorphic serializer class, copied from: https://stackoverflow.com/questions/48911345/drf-how-to-serialize-models-inheritance-read-write

class django_analyses.serializers.utils.polymorphic.**PolymorphicSerializer**(*instance=None*, *data=<class 'rest_framework.fields.e*, ***kwargs*)

Bases: rest_framework.serializers.Serializer

Serializer to handle multiple subclasses of another class

- For serialized dict representations, a 'type' key with the class name as the value is expected: ex. {'type': 'Decimal', ... }
- This type information is used in tandem with get_serializer_map(...) to manage serializers for multiple subclasses

create(*validated_data*)

get_serializer()
Return a dict to map class names to their respective serializer classes. To be implemented by all PolymorphicSerializer subclasses.

get_type(*instance*) → str

to_internal_value(*data*)
Dict of native values <- Dict of primitive datatypes.

to_representation(*instance*)
Object instance -> Dict of primitive datatypes.

update(*instance*, *validated_data*)

validate_type_key_exists(*data: dict*) → None

Module contents

Submodules

django_analyses.serializers.analysis module

class django_analyses.serializers.analysis.**AnalysisSerializer**(*instance=None*, *data=<class 'rest_framework.fields.empty'>*, ***kwargs*)

Bases: rest_framework.serializers.HyperlinkedModelSerializer

class Meta
Bases: object

fields = ('id', 'title', 'description', 'category', 'created', 'modified', 'url')

model
alias of *django_analyses.models.analysis.Analysis*

django_analyses.serializers.analysis_version module

class django_analyses.serializers.analysis_version.**AnalysisVersionSerializer**(*instance=None*, *data=<class 'rest_framework.fiel*
***kwargs*)

Bases: rest_framework.serializers.HyperlinkedModelSerializer

class Meta
Bases: object

fields = ('id', 'analysis', 'title', 'description', 'input_specification', 'output_

model
alias of *django_analyses.models.analysis_version.AnalysisVersion*

django_analyses.serializers.category module

class django_analyses.serializers.category.**CategorySerializer**(*instance=None*, *data=<class 'rest_framework.fields.empty'>*, ***kwargs*)

Bases: rest_framework.serializers.HyperlinkedModelSerializer

class Meta

Bases: object

fields = ('id', 'title', 'description', 'created', 'modified', 'parent', 'subcatego

model

alias of *django_analyses.models.category.Category*

django_analyses.serializers.run module

class django_analyses.serializers.run.**MiniUserSerializer**(*instance=None*, *data=<class 'rest_framework.fields.empty'>*, ***kwargs*)

Bases: rest_auth.serializers.UserDetailsSerializer

Minified serializer class for the User model.

class Meta

Bases: rest_auth.serializers.Meta

fields = ('id', 'username', 'first_name', 'last_name', 'full_name', 'email')

get_full_name(*instance: django.contrib.auth.models.User*) → str

class django_analyses.serializers.run.**RunSerializer**(*instance=None*, *data=<class 'rest_framework.fields.empty'>*, ***kwargs*)

Bases: rest_framework.serializers.HyperlinkedModelSerializer

class Meta

Bases: object

fields = ('id', 'user', 'analysis_version', 'created', 'modified', 'start_time', 'e

model

alias of *django_analyses.models.run.Run*

duration(*instance: django_analyses.models.run.Run*)

Module contents

4.2.5 Utils

Submodules

django_analyses.utils.choice_enum module

Definition of the *ChoiceEnum* class.

class django_analyses.utils.choice_enum.**ChoiceEnum**
Bases: enum.Enum

A Python enum with a method to provide choices in the format that Django expects them.

choices = <bound method ChoiceEnum.choices of <enum 'ChoiceEnum'>>

django_analyses.utils.input_manager module

Definition of the *InputManager* class.

class django_analyses.utils.input_manager.**InputManager**(*run*, *configuration: dict*)
Bases: object

Creates *Input* subclass instances according to the provided *Run*'s associated *InputSpecification*.

all_input_instances

convert_raw_configuration_to_input_instances() → List[django_analyses.models.input.input.Input]
Coverts a user-provided input configuration dictionary to *Input* subclass instances.

Returns Created inputs

Return type List[*Input*]

create_input_instances() → dict
Creates all the required *Input* subclass instances for the provided run, including the creation of the destination directories of generated files.

Returns Full input configuration

Return type dict

create_required_paths() → None
Creates all the directories required for generated files.

get_all_input_instances() → List[django_analyses.models.input.input.Input]
Returns all *Input* subclass instances for the provided run.

Returns Input instances

Return type List[*Input*]

get_full_configuration() → dict
Returns the complete input configuration dictionary to be passed to the appropriate analysis interface.

Returns Full input configuration

Return type dict

get_missing_dynamic_default_definitions() → List[django_analyses.models.input.definitions.string_input_defi
Returns a list of missing string inputs with a *dynamic_default* value.

Returns List of missing dynamic_default instances

Return type List[*StringInputDefinition*]

get_missing_input_definitions() → Set[django_analyses.models.input.definitions.input_definition.InputDefinition]
Returns the *InputDefinition* subclass instances missing in the configuration for the given run.

Returns Input definitions missing in the provided configuration

Return type Set[*InputDefinition*]

get_missing_output_directory_definition() → List[django_analyses.models.input.definitions.directory_input_d
Returns missing output directory definition. There should only be one at the most, however, it is returned as a list so it can easily be appended to other missing inputs.

Returns List of missing output directory configurations that should be generated

Return type List[*StringInputDefinition*]

get_missing_output_path_definitions() → List[django_analyses.models.input.definitions.string_input_definition.
Returns missing output path definitions.

Returns List of missing output path configurations that should be generated

Return type List[*StringInputDefinition*]

get_or_create_input_instance_from_raw(*key: str*, *value: Any*) → Tuple[django_analyses.models.input.input.Input, bool]
Get or create the appropriate `Input` instance from some user-provided dictionary item configuration.

Parameters

- **key** (*str*) – Input definition key
- **value** (*Any*) – Input value

Returns Matching Input instance and whether is has been created or not

Return type Tuple[*Input*, bool]

Raises `InputDefinition.DoesNotExist` – Invalid input definition key within the input dictionary

get_or_create_missing_inputs() → List[django_analyses.models.input.input.Input]
Returns a list of `Input` subclass instances required to complete the provided run's input configuration.

Returns Missing input instances

Return type List[*Input*]

get_required_paths() → List[django_analyses.models.input.input.Input]
Returns all input instances that require some parent directory to exist.

Returns Input instances

Return type List[*Input*]

input_definition_is_a_missing_output_directory(*input_definition: django_analyses.models.input.definitions.input_definition.I* → bool
Checks whether the provided input definition represents an output directory, and whether it is missing in the provided configuration.

Parameters **input_definition** (`InputDefinition`) – Input definition in question

Returns Whether the provided input definition represents an output directory and is missing in the complete input configuration

Return type bool

input_definition_is_a_missing_output_path(*input_definition: django_analyses.models.input.definitions.input_definition.InputDe* → bool
Checks whether the provided input definition represents an output path, and whether it is missing in the provided configuration.

Parameters **input_definition** (`InputDefinition`) – Input definition in question

Returns Whether the provided input definition represents an output path and is missing in the complete input configuration

Return type bool

missing_input_definitions

missing_inputs

raw_input_instances

required_paths

django_analyses.utils.output_manager module

class django_analyses.utils.output_manager.**OutputManager**(*run*, *results: dict*)
Bases: object

create_output_instance(*key: str*, *value*) → django_analyses.models.output.output.Output

create_output_instances() → list

Module contents

Utilities for the *django_analyses* app.

4.2.6 Views

Submodules

django_analyses.views.analysis module

class django_analyses.views.analysis.**AnalysisViewSet**(***kwargs*)
Bases: *django_analyses.views.defaults.DefaultsMixin*, rest_framework.viewsets.ModelViewSet

filter_class
alias of *django_analyses.filters.analysis.AnalysisFilter*

pagination_class
alias of *django_analyses.views.pagination.StandardResultsSetPagination*

queryset

serializer_class
alias of *django_analyses.serializers.analysis.AnalysisSerializer*

django_analyses.views.analysis_version module

class django_analyses.views.analysis_version.**AnalysisVersionViewSet**(***kwargs*)
Bases: *django_analyses.views.defaults.DefaultsMixin*, rest_framework.viewsets.ModelViewSet

filter_class
alias of django_analyses.filters.analysis_version.AnalysisVersionFilter

pagination_class
alias of *django_analyses.views.pagination.StandardResultsSetPagination*

queryset

serializer_class
alias of *django_analyses.serializers.analysis_version.AnalysisVersionSerializer*

django_analyses.views.category module

class django_analyses.views.category.**CategoryViewSet**(***kwargs*)
Bases: *django_analyses.views.defaults.DefaultsMixin*, rest_framework.viewsets.ModelViewSet

filter_class
alias of *django_analyses.filters.category.CategoryFilter*

pagination_class
alias of *django_analyses.views.pagination.StandardResultsSetPagination*

queryset

serializer_class
alias of *django_analyses.serializers.category.CategorySerializer*

django_analyses.views.defaults module

Default ViewSet mixin.

class django_analyses.views.defaults.**DefaultsMixin**
Bases: object

Default settings for view authentication, permissions, filtering and pagination.

authentication_classes = (<class 'rest_framework.authentication.BasicAuthentication'>,

filter_backends = (<class 'django_filters.rest_framework.backends.DjangoFilterBackend'

permission_classes = (<class 'rest_framework.permissions.IsAuthenticated'>,)

django_analyses.views.input module

class django_analyses.views.input.**InputViewSet**(***kwargs*)
Bases: *django_analyses.views.defaults.DefaultsMixin*, rest_framework.viewsets.ModelViewSet

download(*request: rest_framework.request.Request*, *pk: int = None*) → rest_framework.response.Response

filter_class
alias of *django_analyses.filters.input.input.InputFilter*

get_queryset()
Get the list of items for this view. This must be an iterable, and may be a queryset. Defaults to using *self.queryset*.

This method should always be used rather than accessing *self.queryset* directly, as *self.queryset* gets evaluated only once, and those results are cached for all subsequent requests.

You may want to override this if you need to provide different querysets depending on the incoming request.

(Eg. return a list of items that is specific to the user)

html_repr(*request: rest_framework.request.Request*, *input_id: int = None*, *index: int = None*) → rest_framework.response.Response

pagination_class
alias of *django_analyses.views.pagination.StandardResultsSetPagination*

serializer_class
alias of *django_analyses.serializers.input.input.InputSerializer*

django_analyses.views.input_definition module

class django_analyses.views.input_definition.**InputDefinitionViewSet**(***kwargs*)
Bases: *django_analyses.views.defaults.DefaultsMixin*, rest_framework.viewsets.ModelViewSet

filter_class
alias of *django_analyses.filters.input.input_definition.InputDefinitionFilter*

get_queryset()
Get the list of items for this view. This must be an iterable, and may be a queryset. Defaults to using *self.queryset*.

This method should always be used rather than accessing *self.queryset* directly, as *self.queryset* gets evaluated only once, and those results are cached for all subsequent requests.

You may want to override this if you need to provide different querysets depending on the incoming request.

(Eg. return a list of items that is specific to the user)

pagination_class
alias of *django_analyses.views.pagination.StandardResultsSetPagination*

serializer_class
alias of *django_analyses.serializers.input.definitions.input_definition.InputDefinitionSerializer*

django_analyses.views.input_specification module

class django_analyses.views.input_specification.**InputSpecificationViewSet**(***kwargs*)
Bases: *django_analyses.views.defaults.DefaultsMixin*, rest_framework.viewsets.ModelViewSet

filter_class
alias of django_analyses.filters.input.input_specification.InputSpecificationFilter

pagination_class
alias of *django_analyses.views.pagination.StandardResultsSetPagination*

queryset

serializer_class
alias of *django_analyses.serializers.input.input_specification.InputSpecificationSerializer*

django_analyses.views.node module

class django_analyses.views.node.**NodeViewSet**(***kwargs*)
Bases: *django_analyses.views.defaults.DefaultsMixin*, rest_framework.viewsets.ModelViewSet

filter_class
alias of *django_analyses.filters.pipeline.node.NodeFilter*

pagination_class
alias of *django_analyses.views.pagination.StandardResultsSetPagination*

queryset

serializer_class
alias of *django_analyses.serializers.pipeline.node.NodeSerializer*

django_analyses.views.output module

class django_analyses.views.output.**OutputViewSet**(***kwargs*)
Bases: *django_analyses.views.defaults.DefaultsMixin*, rest_framework.viewsets.ModelViewSet

download(*request: rest_framework.request.Request*, *pk: int = None*) → rest_framework.response.Response

filter_class
alias of *django_analyses.filters.output.output.OutputFilter*

get_queryset()
Get the list of items for this view. This must be an iterable, and may be a queryset. Defaults to using *self.queryset*.

This method should always be used rather than accessing *self.queryset* directly, as *self.queryset* gets evaluated only once, and those results are cached for all subsequent requests.

You may want to override this if you need to provide different querysets depending on the incoming request.

(Eg. return a list of items that is specific to the user)

html_repr(*request: rest_framework.request.Request*, *output_id: int = None*, *index: int = None*) → rest_framework.response.Response

pagination_class
alias of *django_analyses.views.pagination.StandardResultsSetPagination*

serializer_class
alias of *django_analyses.serializers.output.output.OutputSerializer*

django_analyses.views.output_definition module

class django_analyses.views.output_definition.**OutputDefinitionViewSet**(***kwargs*)
Bases: *django_analyses.views.defaults.DefaultsMixin*, rest_framework.viewsets.ModelViewSet

filter_class
alias of *django_analyses.filters.output.output_definition.OutputDefinitionFilter*

get_queryset()
Get the list of items for this view. This must be an iterable, and may be a queryset. Defaults to using *self.queryset*.

This method should always be used rather than accessing *self.queryset* directly, as *self.queryset* gets evaluated only once, and those results are cached for all subsequent requests.

You may want to override this if you need to provide different querysets depending on the incoming request.

(Eg. return a list of items that is specific to the user)

pagination_class
alias of *django_analyses.views.pagination.StandardResultsSetPagination*

serializer_class
alias of *django_analyses.serializers.output.definitions.output_definition.OutputDefinitionSerializer*

django_analyses.views.output_specification module

class django_analyses.views.output_specification.**OutputSpecificationViewSet**(***kwargs*)
Bases: *django_analyses.views.defaults.DefaultsMixin*, rest_framework.viewsets.ModelViewSet

filter_class
alias of django_analyses.filters.output.output_specification.OutputSpecificationFilter

pagination_class
alias of *django_analyses.views.pagination.StandardResultsSetPagination*

queryset

serializer_class
alias of *django_analyses.serializers.output.output_specification.OutputSpecificationSerializer*

django_analyses.views.pagination module

class django_analyses.views.pagination.**StandardResultsSetPagination**
Bases: rest_framework.pagination.PageNumberPagination

Default pagination parameters. This didn't work as part of the DefaultsMixin and therefore has to be defined separately in the

'pagination_class' configuration.

page_size = 100

page_size_query_param = 'page_size'

django_analyses.views.pipe module

class django_analyses.views.pipe.**PipeViewSet**(***kwargs*)
Bases: *django_analyses.views.defaults.DefaultsMixin*, rest_framework.viewsets.ModelViewSet

filter_class
alias of *django_analyses.filters.pipeline.pipe.PipeFilter*

pagination_class
alias of *django_analyses.views.pagination.StandardResultsSetPagination*

queryset

serializer_class
alias of *django_analyses.serializers.pipeline.pipe.PipeSerializer*

django_analyses.views.pipeline module

class django_analyses.views.pipeline.**PipelineViewSet**(***kwargs*)
Bases: *django_analyses.views.defaults.DefaultsMixin*, rest_framework.viewsets.ModelViewSet

filter_class
alias of *django_analyses.filters.pipeline.pipeline.PipelineFilter*

pagination_class
alias of *django_analyses.views.pagination.StandardResultsSetPagination*

queryset

serializer_class
alias of *django_analyses.serializers.pipeline.pipeline.PipelineSerializer*

django_analyses.views.run module

class django_analyses.views.run.**RunViewSet**(***kwargs*)
Bases: *django_analyses.views.defaults.DefaultsMixin*, rest_framework.viewsets.ModelViewSet

filter_class
alias of django_analyses.filters.run.RunFilter

ordering_fields = ('analysis_version__analysis__title', 'analysis_version__title', 'st

pagination_class
alias of *django_analyses.views.pagination.StandardResultsSetPagination*

queryset

serializer_class
alias of *django_analyses.serializers.run.RunSerializer*

to_zip(*request: rest_framework.request.Request*, *pk: int*) → django.http.response.FileResponse

Module contents

4.3 Submodules

4.4 django_analyses.admin module

Registers various `admin` models to generate the app's admin site interface.

References

- The Django admin site

class django_analyses.admin.**AnalysisAdmin**(*model*, *admin_site*)
Bases: `django.contrib.admin.options.ModelAdmin`

fields = ('title', 'description', 'category', 'created', 'modified')

inlines = [<class 'django_analyses.admin.AnalysisVersionInline'>]

list_display = ('title', 'description', 'run_count')

media

readonly_fields = ('run_count', 'created', 'modified')

run_count(*instance: django_analyses.models.analysis.Analysis*) → int

class django_analyses.admin.**AnalysisVersionAdmin**(*model*, *admin_site*)
Bases: `django.contrib.admin.options.ModelAdmin`

analysis_link(*instance: django_analyses.models.input.input_specification.InputSpecification*) → str

fieldsets = ((None, {'fields': ('analysis', 'title', 'description', 'input_specificat

id_link(*instance: django_analyses.models.analysis_version.AnalysisVersion*) → str

inlines = (<class 'django_analyses.admin.NodeInline'>,)

input_specification_link(*instance: django_analyses.models.analysis_version.AnalysisVersion*) → str

list_display = ('id_link', 'analysis_link', 'title', 'description', 'input_specificati

media

name(*instance*) → str

output_specification_link(*instance: django_analyses.models.analysis_version.AnalysisVersion*) → str

readonly_fields = ('analysis', 'id_link', 'input_specification_link', 'output_specifica

run_count(*instance: django_analyses.models.analysis_version.AnalysisVersion*) → int

class django_analyses.admin.**AnalysisVersionInline**(*parent_model*, *admin_site*)
Bases: `django.contrib.admin.options.TabularInline`

class Media
Bases: `object`

css = {'all': ('django_analyses/css/hide_admin_original.css',)}

can_delete = False

extra = 0

fields = ('id_link', 'title', 'description', 'input_specification_link', 'output_speci

has_add_permission(*request*, *obj*)
Return True if the given request has permission to add an object. Can be overridden by the user in subclasses.

id_link(*instance: django_analyses.models.analysis_version.AnalysisVersion*) → str

input_specification_link(*instance: django_analyses.models.analysis_version.AnalysisVersion*) → str

media

model
alias of *django_analyses.models.analysis_version.AnalysisVersion*

output_specification_link(*instance: django_analyses.models.analysis_version.AnalysisVersion*) → str

readonly_fields = ('id_link', 'title', 'description', 'input_specification_link', 'outp

run_count(*instance: django_analyses.models.analysis_version.AnalysisVersion*) → int

class django_analyses.admin.**AnalysisVersionInputSpecInline**(*parent_model*, *admin_site*)
Bases: django.contrib.admin.options.TabularInline

class Media
Bases: object

css = {'all': ('django_analyses/css/hide_admin_original.css',)}

can_delete = False

extra = 0

fields = ('version_link', 'description', 'output_specification_link')

has_add_permission(*request*, *obj*)
Return True if the given request has permission to add an object. Can be overridden by the user in subclasses.

media

model
alias of *django_analyses.models.analysis_version.AnalysisVersion*

output_specification_link(*instance: django_analyses.models.analysis_version.AnalysisVersion*) → str

readonly_fields = ('version_link', 'description', 'output_specification_link')

version_link(*instance: django_analyses.models.analysis_version.AnalysisVersion*) → str

class django_analyses.admin.**AnalysisVersionOutputSpecInline**(*parent_model*, *admin_site*)
Bases: django.contrib.admin.options.TabularInline

class Media
Bases: object

css = {'all': ('django_analyses/css/hide_admin_original.css',)}

can_delete = False

extra = 0

fields = ('version_link', 'description', 'input_specification_link')

has_add_permission(*request*, *obj*)
Return True if the given request has permission to add an object. Can be overridden by the user in subclasses.

input_specification_link(*instance: django_analyses.models.analysis_version.AnalysisVersion*) → str

media

model
alias of *django_analyses.models.analysis_version.AnalysisVersion*

readonly_fields = ('version_link', 'description', 'input_specification_link')

version_link(*instance: django_analyses.models.analysis_version.AnalysisVersion*) → str

class django_analyses.admin.**InputAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

class Media
Bases: object

js = ('//cdnjs.cloudflare.com/ajax/libs/jquery/3.3.1/jquery.min.js',)

analysis_version(*instance*) → str

analysis_version_link(*instance: django_analyses.models.input.input.Input*) → str

change_view(**args*, ***kwargs*)

check_fileness(*instance: django_analyses.models.input.input.Input*) → bool

definition_link(*instance: django_analyses.models.input.input.Input*) → str

fields = ('analysis_version_link', 'definition_link', 'run_link', 'input_type', '_valu

get_queryset(*request*)
Return a QuerySet of all model instances that can be edited by the admin site. This is used by changelist_view.

input_type(*instance: django_analyses.models.input.input.Input*) → str

list_display = ('analysis_version_link', 'run_link', 'definition_link', 'input_type',

list_display_links = None

list_filter = ('run__analysis_version',)

media

readonly_fields = ('analysis_version_link', 'definition_link', 'run_link', 'input_type

run_link(*instance: django_analyses.models.input.input.Input*) → str

search_fields = ('run__id',)

class django_analyses.admin.**InputDefinitionAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

class Media
Bases: object

css = {'all': ('django_analyses/css/hide_admin_original.css',)}

choices(*instance*) → list

fieldsets = ((None, {'fields': ['key', 'description', 'required', 'default', 'min_val

get_queryset(*request*)
Return a QuerySet of all model instances that can be edited by the admin site. This is used by changelist_view.

list_display = ('key', 'description', 'min_value', 'max_value', 'default', 'choices',

list_filter = ('specification_set__analysis__title', 'specification_set__id')

max_value(*instance*)

media

min_value(*instance*)

readonly_fields = ('default', 'choices', 'min_value', 'max_value')

class django_analyses.admin.**InputDefinitionsInline**(*parent_model*, *admin_site*)
Bases: django.contrib.admin.options.TabularInline

can_delete = False

default(*instance: django_analyses.models.input.definitions.input_definition.InputDefinition*) → str

description(*instance: django_analyses.models.input.definitions.input_definition.InputDefinition*) → str

extra = 0

fields = ('id_link', 'key', 'input_type', 'description', 'required', 'is_configuration

has_add_permission(*request*, *obj*)
Return True if the given request has permission to add an object. Can be overridden by the user in subclasses.

id_link(*instance: django_analyses.models.input.definitions.input_definition.InputDefinition*) → str

input_type(*instance: django_analyses.models.input.definitions.input_definition.InputDefinition*) → str

is_configuration(*instance: django_analyses.models.input.definitions.input_definition.InputDefinition*) → str

key(*instance: django_analyses.models.input.definitions.input_definition.InputDefinition*) → str

media

model
alias of django_analyses.models.input.input_specification.InputSpecification_base_input_definitions

readonly_fields = ('id_link', 'key', 'input_type', 'description', 'required', 'is_conf

required(*instance: django_analyses.models.input.definitions.input_definition.InputDefinition*) → bool

verbose_name_plural = 'Input Definitions'

class django_analyses.admin.**InputInline**(*parent_model*, *admin_site*)
Bases: django.contrib.admin.options.TabularInline

can_delete = False

definition_link(*instance: django_analyses.models.input.input.Input*) → str

extra = 0

get_queryset(*request*)
Return a QuerySet of all model instances that can be edited by the admin site. This is used by changelist_view.

has_add_permission(*request*, *obj*)
Return True if the given request has permission to add an object. Can be overridden by the user in subclasses.

id_link(*instance: django_analyses.models.input.input.Input*) → str

input_type(*instance: django_analyses.models.input.input.Input*) → str

media

model
alias of *django_analyses.models.input.input.Input*

readonly_fields = ('id_link', 'definition_link', 'input_type', 'value')

class django_analyses.admin.**InputSpecificationAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

class Media
Bases: object

css = {'all': ('django_analyses/css/hide_admin_original.css',)}

analysis_link(*instance: django_analyses.models.input.input_specification.InputSpecification*) → str

analysis_versions(*instance: django_analyses.models.input.input_specification.InputSpecification*) → str

fields = ('analysis',)

inlines = [<class 'django_analyses.admin.AnalysisVersionInputSpecInline'>, <class 'dja

input_definitions_count(*instance: django_analyses.models.input.input_specification.InputSpecification*) → int

list_display = ('id', 'analysis_link', 'analysis_versions', 'input_definitions_count')

list_filter = ('analysis',)

media

readonly_fields = ('analysis', 'analysis_link', 'analysis_versions', 'input_definition

class django_analyses.admin.**NodeAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

analysis_version_link(*instance: django_analyses.models.pipeline.node.Node*) → str

fields = ('analysis_version_link', 'configuration')

formfield_overrides = {<class 'django.db.models.fields.json.JSONField'>: {'widget':

list_display = ('id', 'analysis_version_link', 'configuration')

list_filter = ('analysis_version__analysis',)

media

readonly_fields = ('analysis_version_link',)

search_fields = ('id', 'analysis_version__analysis__title', 'analysis_version__title',

class django_analyses.admin.**NodeInline**(*parent_model*, *admin_site*)
Bases: django.contrib.admin.options.TabularInline

class Media
Bases: object

```
        css = {'all':  ('django_analyses/css/hide_admin_original.css',)}
```

```
    can_delete = False
```

```
    extra = 0
```

```
    fields = ('id_link', 'configuration')
```

has_add_permission(*request*, *obj*)
Return True if the given request has permission to add an object. Can be overridden by the user in subclasses.

id_link(*instance: django_analyses.models.pipeline.node.Node*) → str

media

model
alias of *django_analyses.models.pipeline.node.Node*

```
    readonly_fields = ('id_link', 'configuration')
```

class django_analyses.admin.**OutputAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

class Media
Bases: object

```
        js = ('//cdnjs.cloudflare.com/ajax/libs/jquery/3.3.1/jquery.min.js',)
```

analysis_version(*instance: django_analyses.models.output.output.Output*) → str

analysis_version_link(*instance: django_analyses.models.output.output.Output*) → str

change_view(**args*, ***kwargs*)

check_fileness(*instance: django_analyses.models.output.output.Output*) → bool

definition_link(*instance: django_analyses.models.output.output.Output*) → str

download(*instance: django_analyses.models.output.output.Output*) → str

```
    fields = ('analysis_version_link', 'definition_link', 'run_link', 'output_type', '_val
```

get_queryset(*request*)
Return a QuerySet of all model instances that can be edited by the admin site. This is used by changelist_view.

```
    list_display = ('analysis_version_link', 'run_link', 'definition_link', 'output_type',
```

```
    list_display_links = None
```

```
    list_filter = ('run__analysis_version',)
```

media

output_type(*instance: django_analyses.models.output.output.Output*) → str

```
    readonly_fields = ('analysis_version_link', 'definition_link', 'run_link', 'output_typ
```

run_link(*instance: django_analyses.models.output.output.Output*) → str

```
    search_fields = ('run__id',)
```

class django_analyses.admin.**OutputDefinitionAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

analysis(*instance*)

```
    list_display = ('key', 'description', 'analysis')
```

list_filter = ('specification_set__analysis__title', 'specification_set__id')

media

class django_analyses.admin.**OutputDefinitionsInline**(*parent_model*, *admin_site*)
Bases: django.contrib.admin.options.TabularInline

can_delete = False

description(*instance: django_analyses.models.output.definitions.output_definition.OutputDefinition*) → str

extra = 0

fields = ('id_link', 'key', 'output_type', 'description')

has_add_permission(*request*, *obj*)
Return True if the given request has permission to add an object. Can be overridden by the user in subclasses.

id_link(*instance: django_analyses.models.output.definitions.output_definition.OutputDefinition*) → str

key(*instance: django_analyses.models.output.definitions.output_definition.OutputDefinition*) → str

media

model
alias of django_analyses.models.output.output_specification.OutputSpecification_base_output_definitions

output_type(*instance: django_analyses.models.output.definitions.output_definition.OutputDefinition*) → str

readonly_fields = ('id_link', 'key', 'output_type', 'description')

verbose_name_plural = 'Output Definitions'

class django_analyses.admin.**OutputInline**(*parent_model*, *admin_site*)
Bases: django.contrib.admin.options.TabularInline

can_delete = False

definition_link(*instance: django_analyses.models.output.output.Output*) → str

download(*instance: django_analyses.models.output.output.Output*) → str

extra = 0

get_queryset(*request*)
Return a QuerySet of all model instances that can be edited by the admin site. This is used by changelist_view.

has_add_permission(*request*, *obj*)
Return True if the given request has permission to add an object. Can be overridden by the user in subclasses.

id_link(*instance: django_analyses.models.output.output.Output*) → str

media

model
alias of *django_analyses.models.output.output.Output*

output_type(*instance: django_analyses.models.output.output.Output*) → str

readonly_fields = ('id_link', 'definition_link', 'output_type', 'value', 'download')

class django_analyses.admin.**OutputSpecificationAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

class Media
Bases: object

css = {'all': ('django_analyses/css/hide_admin_original.css',)}

analysis_link(*instance: django_analyses.models.output.output_specification.OutputSpecification*) → str

analysis_versions(*instance: django_analyses.models.output.output_specification.OutputSpecification*) → str

fields = ('analysis',)

inlines = [<class 'django_analyses.admin.AnalysisVersionOutputSpecInline'>, <class 'dj

list_display = ('id', 'analysis_link', 'analysis_versions', 'output_definitions_count'

list_filter = ('analysis',)

media

output_definitions_count(*instance: django_analyses.models.output.output_specification.OutputSpecification*) → int

readonly_fields = ('analysis', 'analysis_link', 'analysis_versions', 'output_definitio

class django_analyses.admin.**PipeAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

destination_analysis_version(*instance: django_analyses.models.pipeline.pipe.Pipe*) → str

destination_configuration(*instance: django_analyses.models.pipeline.pipe.Pipe*) → str

destination_node(*instance: django_analyses.models.pipeline.pipe.Pipe*) → str

destination_port_key(*instance: django_analyses.models.pipeline.pipe.Pipe*) → str

fieldsets = ((None, {'fields': ['pipeline', 'source_analysis_version', 'source_port_k

formfield_overrides = {<class 'django.db.models.fields.json.JSONField'>: {'widget':

id_link(*instance: django_analyses.models.pipeline.pipe.Pipe*) → str

list_display = ('id_link', 'pipeline_link', 'source_analysis_version', 'source_node',

list_filter = ('pipeline', ('source__analysis_version__analysis', <class 'django_analy

media

pipeline_link(*instance: django_analyses.models.pipeline.pipe.Pipe*) → str

readonly_fields = ('id_link', 'pipeline_link', 'source_analysis_version', 'source_port

search_fields = ('id', 'pipeline__title', 'source__analysis_version__analysis__title',

source_analysis_version(*instance: django_analyses.models.pipeline.pipe.Pipe*) → str

source_configuration(*instance: django_analyses.models.pipeline.pipe.Pipe*) → str

source_node(*instance: django_analyses.models.pipeline.pipe.Pipe*) → str

source_port_key(*instance: django_analyses.models.pipeline.pipe.Pipe*) → str

class django_analyses.admin.**PipeInLine**(*parent_model*, *admin_site*)
Bases: django.contrib.admin.options.TabularInline

class Media
Bases: object

css = {'all': ('django_analyses/css/hide_admin_original.css',)}

can_delete = False

destination_analysis_version(*instance: django_analyses.models.pipeline.pipe.Pipe*) → str

destination_node(*instance: django_analyses.models.pipeline.pipe.Pipe*) → str

destination_port_key(*instance: django_analyses.models.pipeline.pipe.Pipe*) → str

extra = 0

fields = ('id_link', 'source_analysis_version', 'source_node', 'source_port_key', 'des

has_add_permission(*request*, *obj*)
Return True if the given request has permission to add an object. Can be overridden by the user in subclasses.

id_link(*instance: django_analyses.models.input.input.Input*) → str

media

model
alias of *django_analyses.models.pipeline.pipe.Pipe*

readonly_fields = ('id_link', 'source_analysis_version', 'source_node', 'source_port_k

source_analysis_version(*instance: django_analyses.models.pipeline.pipe.Pipe*) → str

source_node(*instance: django_analyses.models.pipeline.pipe.Pipe*) → str

source_port_key(*instance: django_analyses.models.pipeline.pipe.Pipe*) → str

class django_analyses.admin.**PipelineAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

fields = ('title', 'description', 'created', 'modified')

inlines = (<class 'django_analyses.admin.PipeInLine'>,)

list_display = ('id', 'title', 'description', 'node_count', 'pipe_count')

media

node_count(*instance: django_analyses.models.pipeline.pipeline.Pipeline*) → int

pipe_count(*instance: django_analyses.models.pipeline.pipeline.Pipeline*) → int

readonly_fields = ('created', 'modified', 'node_count', 'pipe_count')

search_fields = ('id', 'title', 'description')

class django_analyses.admin.**PrettyJSONWidget**(*attrs=None*)
Bases: django.forms.widgets.Textarea

format_value(*value*)
Return a value as it should appear when rendered in a template.

media

class django_analyses.admin.**RunAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

class Media
Bases: object

```
    css = {'all':  ('django_analyses/css/hide_admin_original.css',)}
```

analysis_version_link(*instance: django_analyses.models.run.Run*) → str

download(*instance: django_analyses.models.run.Run*) → str

duration(*instance: django_analyses.models.run.Run*) → datetime.timedelta

```
fieldsets = ((None, {'fields':  ('analysis_version_link', 'user')}), ('Execution', {'f
```

```
inlines = (<class 'django_analyses.admin.InputInline'>, <class 'django_analyses.admin.
```

```
list_display = ('id', 'analysis_version_link', 'user_link', 'start_time', 'end_time',
```

```
list_filter = ('status', 'start_time', 'analysis_version__analysis', 'user')
```

media

```
readonly_fields = ('analysis_version', 'analysis_version_link', 'user_link', 'start_ti
```

```
search_fields = ('id', 'analysis_version__title', 'analysis_version__analysis__title')
```

user_link(*instance: django_analyses.models.run.Run*) → str

django_analyses.admin.**custom_titled_filter**(*title: str*)
Copied from SO: https://stackoverflow.com/a/21223908/4416932

4.5 django_analyses.apps module

Definition of the *DjangoAnalysesConfig* class.

References

- Django applications

class django_analyses.apps.**DjangoAnalysesConfig**(*app_name*, *app_module*)
Bases: django.apps.config.AppConfig

django_analyses app configuration.

References

- AppConfig attributes

name = 'django_analyses'
Full Python path to the application.

ready()
Loads the app's signals.

References

- ready()

4.6 django_analyses.pipeline_runner module

Definition of the `PipelineRunner` class.

class django_analyses.pipeline_runner.**PipelineRunner**(*pipeline: django_analyses.models.pipeline.pipeline.Pipeline*, *quiet: bool = False*)

Bases: `object`

Manages the execution of pipelines.

generate_user_inputs_string(*user_inputs: dict*) → str
Formats the user provided input dictionary as a readable string.

Parameters **user_inputs** (*dict*) – Standardized user provided inputs

Returns Formatted user provided input dictionary

Return type str

get_destination_kwarg(*pipe: django_analyses.models.pipeline.pipe.Pipe*) → dict
Composes a keyword argument to include in a destination node's configuration.

Parameters **pipe** (Pipe) – A pipe with an existing run matching its source node

Returns Destination node keyword argument

Return type dict

get_incoming_pipes(*node: django_analyses.models.pipeline.node.Node*, *run_index: int*) → django.db.models.query.QuerySet
Returns all pipes in the pipeline that declare the given node instance as their destination.

Parameters

- **node** (Node) – Destination node
- **run_index** (*int*) – The desired run index of the specified node

Returns Pipes with the provided node as destination

Return type QuerySet

get_node_inputs(*node: django_analyses.models.pipeline.node.Node*, *user_inputs: Dict[django_analyses.models.pipeline.node.Node, List[Dict[str, Any]]]*, *run_index: int*) → dict
Returns the node's input configuration, including inputs specified by the user and preceding nodes' outputs.

Parameters

- **node** (Node) – Node for which to compose an input configuration
- **user_inputs** (*Dict[Node, List[Dict[str, Any]]]*) – User provided input configurations

Returns Input configuration

Return type dict

get_node_user_inputs(*user_inputs: Dict[django_analyses.models.pipeline.node.Node, List[Dict[str, Any]]]*, *node: django_analyses.models.pipeline.node.Node*, *run_index: int*) → Dict[str, Any]
Returns the input configuration dictionary provided by the user for the specified node's execution.

Parameters

- **user_inputs** (*Dict[Node, List[Dict[str, Any]]]*) – User provided input configurations
- **node** (*Node*) – The node to look for
- **run_index** (*int*) – The index of the requested node's execution

Returns User provided input configuration

Return type Dict[str, Any]

get_safe_results() → dict
Returns a JSON-serializable dictionary of the pipeline's outputs.

Returns Results dictionary with node IDs as keys a list of result dictionaries for each run of that node

Return type dict

has_required_runs(*node: django_analyses.models.pipeline.node.Node*, *run_index: int*) → bool
Checks whether the provided node is ready to be run by evaluating the execution state of the nodes it requires (nodes that generate output meant to be piped to it).

Parameters

- **node** (*Node*) – Node to evaluate
- **run_index** (*int*) – Filter by the index of the node's run

Returns Whether all required nodes have been executed or not

Return type bool

pending_nodes
Nodes that were not yet executed.

Returns Pending nodes

Return type list

report_node_execution_end(*run*) → None
Reports the end of a *node*'s execution.

Parameters **run** (*Run*) – The created run instance

report_node_execution_failure(*node: django_analyses.models.pipeline.node.Node*, *user_inputs: dict*, *node_inputs: dict*, *exception: str*) → None
Reports a failure in a *node*'s execution.

Parameters

- **node** (*Node*) – The executed node
- **user_inputs** (*dict*) – Standardized user provided inputs
- **node_inputs** (*dict*) – The complete input configuration for this node's execution
- **exception** (*str*) – The raised exception

Raises RuntimeError – [description]

report_node_execution_start(*node: django_analyses.models.pipeline.node.Node*, *run_index: int*, *inputs: dict*, *first: bool = False*) → None
Reports the start of a *node*'s execution.

Parameters

- **node** (Node) – The executed node
- **run_index** (*int*) – The index of the node's run in this pipeline
- **inputs** (*dict*) – The node's input configuration dictionary
- **first** (*bool, optional*) – Whether this is the first execution of this pipeline, by default False

reset_runs_dict() → None
Resets the `runs` dictionary before a new execution.

run(*inputs: dict*) → dict
Runs `pipeline` with the provided *inputs*.

Parameters **inputs** (*dict*) – Input configurations to be passed to the nodes, this may either be provided as a dictionary with nodes as keys and configurations as values or simply an input configuration if there's only one entry node

Returns Resulting run instances

Return type dict

run_entry_nodes(*user_inputs: Dict[django_analyses.models.pipeline.node.Node, List[Dict[str, Any]]]*) → None
Runs the "entry" nodes of the pipeline, i.e. nodes that are not the destination of any other node.

Parameters **user_inputs** (*Dict[Node, List[Dict[str, Any]]]*) – User provided input configurations

run_node(*node: django_analyses.models.pipeline.node.Node, user_inputs: Dict[django_analyses.models.pipeline.node.Node, List[Dict[str, Any]]]*) → None
Runs the provided node and stores the created *Run* instances in the class's `runs` attribute.

Parameters

- **node** (Node) – Node to be executed
- **user_inputs** (*Dict[Node, List[Dict[str, Any]]]*) – User provided input configurations

standardize_user_input(*user_input: Union[List[Dict[str, Any]], Dict[Union[str, django_analyses.models.pipeline.node.Node], Any]]*) → Dict[django_analyses.models.pipeline.node.Node, List[Dict[str, Any]]]
Standardizes user input to conform with the desired format (a dictionary with nodes as keys and list of input dictionaries as values).

Parameters **user_input** (*Union[List[Dict[str, Any]], Dict[Union[str, Node], Any]]*) – User input as either a dictionary of nodes and their input dictionaries, or an input dictionary (or list of input dictionaries) specified for a singular entry node

Returns Standardized user input

Return type Dict[*Node*, List[Dict[str, Any]]]

validate_user_input_keys(*user_input: Dict[Union[str, django_analyses.models.pipeline.node.Node], Any]*) → bool
Validates all keys of a user input dictionary are either nodes or strings. Return *True* if all are nodes, *False* if all are strings, and raises a *ValueError* in any other case.

Parameters **user_input** (*Dict[Union[str, Node], Any]*) – User input dictionary

Returns True of all keys are nodes, False if all keys are strings

Return type bool

4.7 django_analyses.urls module

django_analyses.urls.**path**(*route*, *view*, *kwargs=None*, *name=None*, *, *Pattern=<class 'django.urls.resolvers.RoutePattern'>*)

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

Index

A

B

C

D

E

F

G

H

J

K

L

M

N

O

P

Q

R

S

T

U

V