idmtools

Institute for Disease Modeling

Aug 12, 2020

CONTENTS

IDM Modeling Tools is a collection of Python scripts and utilities created to streamline user interactions with disease models. This framework provides the user with tools necessary to complete projects, starting from the creation of input files (if required), to calibration of the model to data, to commissioning and running simulations, through the analysis of results. Modelers can use idmtools to run models locally or send suites of simulations to an HPC or other computing source. This framework is free, open-source, and model agnostic: it can be used to interact with a variety of models, such as custom models written in R or Python, or IDM's own EMOD.

CHAPTER

ONE

IDMTOOLS WORKFLOW

idmtools includes a variety of options for each step of the modeling process. Because of this, the tool suite was developed in a modular fashion, so that users can select the utilities they wish to use. In order to simplify the desired workflow, facilitate the modeling process, and make the model (and its results) reusable and sharable, idmtools allows the user to create *assets*. Assets can be added at any level of the process, from running a specific task, through creating a simulation, to creating a *experiment*. This allows the user to create inputs based on their specific needs: they can be transient, or sharable across multiple simulations.

Exact workflows for using idmtools is user-dependent, and can include any of the tasks listed below.

1.1 Installation

You can install IDM Modeling Tools in two different ways. If you intend to use idmtools as IDM builds it, follow the instructions in *Basic installation*. However, if you intend to modify the idmtools source code to add new functionality, follow the instructions in *Developer installation*. Whichever installation method you choose, the prerequisites are the same.

1.1.1 Prerequisites

idmtools uses Docker to run idmtools within a container to keep the idmtools environment securely isolated. You must also have Python 3.6, 3.7, or 3.8 64-bit and Python virtual environments installed to isolate your idmtools installation in a separate Python environment. If you do not already have these installed, see the links below for instructions.

- Windows 10 Pro or Enterprise
- Python 3.6, 3.7, or 3.8 64-bit (https://www.python.org/downloads/release)
- Python virtual environments

  Python virtual environments enable you to isolate your Python environments from one another and give you the option to run multiple versions of Python on the same computer. When using a virtual environment, you can indicate the version of Python you want to use and the packages you want to install, which will remain separate from other Python environments. You may use `virtualenv`, which requires a separate installation, but `venv` is recommended and included with Python 3.3+.

- Docker (https://docs.docker.com/)

  Docker is optional for the basic installation of idmtools; it is needed only for running simulations or analysis locally. It is required for the developer installation.

Basic installation

Follow the steps below if you will use idmtools to run and analyze simulations, but will not make source code changes.

1. Open a command prompt and create a virtual environment in any directory you choose. The command below names the environment "idmtools", but you may use any desired name:

```
python -m venv idmtools
```

2. Activate the virtual environment:

   - On Windows, enter the following:

```
idmtools\Scripts\activate
```

   - On Linux, enter the following:

```
source idmtools/bin/activate
```

3. Install idmtools packages:

```
pip install idmtools[idm] --index-url=https://packages.idmod.org/api/pypi/pypi-
↪production/simple
```

   If you are on Python 3.6, also run:

```
pip install dataclasses
```

Note: When reinstalling idmtools you should use the `--no-cache-dir` and `--force-reinstall` options, such as: `pip install idmtools[idm] --index-url=https://packages.idmod.org/api/pypi/pipi-production/simple --no-cache-dir --force-reinstall`. Otherwise, you may see the error, **idmtools not found**, when attempting to open and run one of the example Python scripts.

---

4. Verify installation by pulling up idmtools help:

```
idmtools --help
```

5. When you are finished, deactivate the virtual environment by entering the following at a command prompt:

```
deactivate
```

Developer installation

Follow the steps below if you will make changes to the idmtools source code to add new functionality.

Install idmtools

1. Install a Git client such as Git Bash or the Git GUI.
2. Open a command prompt and clone the idmtools GitHub repository to a local directory using the following command:

```
git clone https://github.com/InstituteforDiseaseModeling/idmtools.git
```

   To work from the latest approved code, work from the "master" branch. To work from the latest code under active development, work from the "dev" branch.

3. Open a command prompt and create a virtual environment in any directory you choose. The command below names the environment "idmtools", but you may use any desired name:

```
python -m venv idmtools
```

4. Activate the virtual environment:
   - On Windows, enter the following:

```
idmtools\Scripts\activate
```

   - On Linux, enter the following:

```
source idmtools/bin/activate
```

5. In the base directory of the cloned GitHub repository, run the setup script.
   - On Windows, enter the following:

```
pip install py-make
pymake setup-dev
```

   - On Linux, enter the following:

```
make setup-dev
```

6. To verify that idmtools is installed, enter the following command:

```
idmtools --help
```

   You should see a list of available cookie cutter projects and command-line options.

7. For source completion and indexing, set the package paths in your IDE. In PyCharm, select the following directories then right-click and select **Mark Directory as > Source Root**.
   - idmtools/idmtools_core
   - idmtools/idmtools_cli
   - idmtools/idmtools_platform_local
   - idmtools/idmtools_platform_comps
   - idmtools/idmtools_model_emod
   - idmtools/idmtools_models
   - idmtools/idmtools_test

See *CLI reference* for more information on the command-line interface available for interacting with idmtools.

Start the Docker client

1. Create a Docker network named idmtools_network in the idmtools_local_runner directory using the following commands:

```
cd idmtools_platform_local
docker network create idmtools_network
```

Note: The drive where you create the network most be shared with Docker. Open Docker and then under **Settings > Shared Drives**, verify that the drive is shared.

2. Start the local Docker runner using the following commands, depending on your operating system.
   - On Windows, enter the following. Include the first line only if the data/redis-data directory is not already present:

```
mkdir data\redis-data
docker-compose down -v
docker-compose build
docker-compose up -d
```

   - On Linux, enter the following:

```
sudo docker-compose down -v
sudo docker-compose build
sudo ./start.sh
```

3. Open a browser and navigate to http://localhost:5000/data/.

Note: If your password has changed since running Docker, you will need to update your credentials. Open Docker Desktop > Settings > Resources > File sharing and reset your credentials.

Run tests

If you want to run tests on the code, do the following. You can add new tests to the GitHub repository and they will be run using the same commands.

Note: To access and use COMPS you must receive approval and credentials from IDM. Send your request to support@idmod.org.

1. Login to COMPS by navigating to the idmtools root directory and entering the following at a command prompt:

```
python dev_scripts\create_auth_token_args.py --comps_url https://comps2.idmod.org␣
↪--username yourcomps_user --password yourcomps_password
```

2. If you are running the local platform with the nightly idmtools build, enter the following to log in to Docker:

```
docker login idm-docker-staging.packages.idmod.org
```

3. Navigate to the directory containing the code you want to test, such as the root directory or a subdirectory like idmtools_platform_comps, enter the following command:

```
pymake test-all
```

1.2 Configuration

The configuration of idmtools is set in the idmtools.ini file. This file is normally located in the project directory but idmtools will search up through the directory hierachy. An idmtools.ini file must be included when using idmtools.

Below is an example configuration file:

```
[COMMON]
# Number of threads idmtools will use for analysis and other multi-threaded activities
max_threads = 16

# How many simulations per threads during simulation creation
sims_per_thread = 20

# Maximum number of LOCAL simulation ran simultaneously
max_local_sims = 6

# Maxium number of workers processing in parallel
max_workers = 16

# Maxium batch size to retrieve simulations
batch_size = 10

[COMPS]
type = COMPS
endpoint = https://comps.idmod.org
environment = Belegost
priority = Lowest
simulation_root = $COMPS_PATH(USER)\output
node_group = emod_abcd
num_retries = 0
num_cores = 1
max_workers = 16
batch_size = 10
exclusive = False

[COMPS2]
type = COMPS
endpoint = https://comps2.idmod.org
environment = Bayesian
priority = Lowest
simulation_root = $COMPS_PATH(USER)\output
node_group = emod_abcd
num_retries = 0
num_cores = 1
max_workers = 16
batch_size = 10
exclusive = False

[Logging]
# Options are is descending order. The lower the item in the list, the more verbose␣
↪the logging will be
# CRITICAL, ERROR, WARNING, INFO, DEBUG
```

(continues on next page)

(continued from previous page)

```
level = DEBUG
console = off
log_filename = idmtools.log

# This is a test we used to validate loading local from section block
[Custom_Local]
type = Local

[SLURM]
type = COMPS
endpoint = https://comps2.idmod.org
environment = SlurmStage
priority = Highest
simulation_root = $COMPS_PATH(USER)\output
num_retries = 0
num_cores = 1
exclusive = False
max_workers = 16
batch_size = 10
```

1.2.1 Global parameters

The idmtool.ini file includes some global parameters that drive features within idmtools. These primarily control features around workers and threads and are defined within the [COMMON] section of idmtool.ini. Most likely, you will not need to change these.

The following includes an example of the [COMMON] section of idmtools.ini with the default settings:

```
[COMMON]
max_threads = 16
sims_per_thread = 20
max_local_sims = 6
max_workers = 16
batch_size = 10
```

- max_threads - Maximumn number of threads for analysis and other multi-threaded activities.
- sims_per_thread - How many simulations per threads during simulation creation.
- max_local_sims - Maximum simulations to run locally.
- max_workers - Maxium number of workers processing in parallel.
- batch_size - Maxium batch size to retrieve simulations.

1.2.2 Logging

idmtools includes built-in logging, which is configured in the [LOGGING] section of the idmtools.ini file, and includes the following parameters: *level*, *console*, and *log_filename*. Default settings are shown in the following example:

```
[LOGGING]
level = INFO
console = off
log_filename = idmtools.log
```

Logging verbosity is controlled by configuring the parameter, *level*, with one of the below listed options. They are in descending order, where the lower the item in the list, the more verbose logging is included.

CRITICAL
ERROR
WARNING
INFO
DEBUG

Console logging is enabled by configuring the parameter, *console*, to *on*. The *log_filename* parameter can be configured to something other than the default filename, *idmtools.log*.

1.2.3 idmtools.ini wizard

You can use the `config` command to create a configuration block in your project's `idmtools.ini` file.

```
$ idmtools config --help
INI File Used: /home/docs/checkouts/readthedocs.org/user_builds/institute-for-disease-
↪modeling-idmtools/checkouts/v1.4.0/docs/idmtools.ini
Usage: idmtools config [OPTIONS] COMMAND [ARGS]...

  Contains commands related to the creation of idmtools.ini

  With the config command, you can :  - Generate an idmtools.ini file in the
  current directory  - Add a configuration block

Options:
  --config_path FILE  Path to the idmtools.ini file
  --help              Show this message and exit.

Commands:
  block  Command to create/replace a block in the selected idmtools.ini...
```

If you do not specify a config path, the command will use the `idmtools.ini` file in the current directory. To edit a different file, use the `--config_path` argument to specify its path, such as: `idmtools config --config_path C:\my_project\idmtools.ini`.

Use the `block` command to start the wizard that will guide you through the creation of a configuration block in the selected `idmtools.ini`, for example: `idmtools config block`.

1.3 Platforms

idmtools currently supports running on the following platforms:

COMPS: COmputational Modeling Platform Service (COMPS) is a high performance computing cluster used by employees and collaborators at IDM. To support running simulations and analysis on COMPS, idmtools includes the following modules: *idmtools_platform_comps*.

Note: To access and use COMPS you must receive approval and credentials from IDM. Send your request to support@idmod.org.

Local: You can also run simulations and analysis locally on your computer, rather than on a remote high-performance computer (HPC). For more information about these modules, see *idmtools_platform_local*.

You can use the **idmtools.ini** file to configure platform specific settings, as the following examples shows for COMPS:

```
[COMPS]
type = COMPS
endpoint = https://comps.idmod.org
environment = Belegost
priority = Lowest
simulation_root = $COMPS_PATH(USER)\output
node_group = emod_abcd
num_retires = 0
num_cores = 1
max_workers = 16
batch_size = 10
exclusive = False
```

Within your code you use the *Platform* class to specify which platform idmtools will use. For example, the following excerpt sets **platform** to use COMPS and overrides **priority** and **node_group** settings.:

```
platform = Platform('COMPS',priority='AboveNormal',node_group='emod_a')
```

You use the *Platform* class whether you're building or running an experiment, or running analysis on output from simulations.

For additional information about configuring idmtools.ini, see *Configuration*.

1.3.1 Local platform

To run simulations and experiments on the local platform you must have met the installation prerequisites. For more information, see *Installation*. In addition, the Docker client must be running. For more information, see *Start the Docker client* section in *Developer installation*.

Verify local platform is running

Type the following at a command prompt to verify that local platform is running:

```
idmtools local status
```

You should see the status of `running` for each of the following docker containers:

- idmtools_redis
- idmtools_postgres
- idmtools_workers

If not then you may need to run:

```
idmtools local start
```

Run examples

To run the included examples on local platform you must configure the *Platform* to `Local`, such as:

```
platform = Platform('Local')
```

And, you must include the following block in the `idmtools.ini` file:

```
[Local]
type = Local
```

Note: You should be able to use most of the included examples, see *Examples*, on local platform except for those that use *IWorkflowItem* or *Suite* Python classes.

View simulations and experiments

You can the dashboard or the CLI for idmtools to view and monitor the status of your simulations and experiments.

The **dashboard** runs on a localhost server on port 5000 (http://localhost:5000). It is recommended that you use Google Chrome to open the dashboard.

The **CLI** command to see the status of simulations is:

```
idmtools simulation --platform Local status
```

And, for experiments:

```
idmtools experiment --platform Local status
```

1.3.2 Create platform plugin

You can add a new platform to idmtools by creating a new platform plugin, as described in the following sections:

Adding fields to the config CLI

If you are developing a new platform plugin, you will need to add some metadata to the Platform class' fields. All fields with a `help` key in the metadata will be picked up by the `idmtools config block` command line and allow a user to set a value. `help` should contain the help text that will be displayed. A `choices` key can optionally be present to restrict the available choices.

For example, for the given platform:

```
@dataclass(repr=False)
class MyNewPlatform(IPlatform, CacheEnabled):
    field1: int = field(default=1, metadata={"help": "This is the first field."})
    internal_field: imt = field(default=2)
    field2: str = field(default="a", metadata={"help": "This is the second field.",
↪"choices": ["a", "b", "c"]})
```

The CLI wizard will pick up `field1` and `field2` and ask the user to provide values. The type of the field will be enforced and for `field2`, the user will have to select among the `choices`.

Modify fields metadata at runtime

Now, what happens if we want to change the help text, choices, or default value of a field based on a previously set field? For example, let's consider an example platform where the user needs to specify an endpoint. This endpoint needs to be used to retrieve a list of environments and we want the user to choose select one of them.

```
@dataclass(repr=False)
class MyNewPlatform(IPlatform, CacheEnabled):
    endpoint: str = field(default="https://myapi.com", metadata={"help": "Enter the␣
↪URL of the endpoint."})
    environment: str = field(metadata={"help": "Select an environment."})
```

The list of environments is dependent on the endpoint value. To achieve this, we need to provide a `callback` function to the metadata. This function will receive all the previously set user parameters, and will have the opportunity to modify the current field's `choices`, `default`, and `help` parameters.

Let's create a function querying the endpoint to get the list of environments and setting them as choices. Selecting the first one as default.

```
def environment_list(previous_settings:Dict, current_field:Field) -> Dict:
    """
    Allows the CLI to provide a list of available environments.
    Uses the previous_settings to get the endpoint to query for environments.
    Args:
        previous_settings: Previous settings set by the user in the CLI.
        current_field: Current field specs.

    Returns:
        Updates to the choices and default.
    """
    # Retrieve the endpoint set by the user
    # The key of the previous_settings is the name of the field we want the value of
    endpoint = previous_settings["endpoint"]

    # Query the API for environments
    client.connect(endpoint)
    environments = client.get_all_environments()

    # If the current field doesnt have a set default already, set one by using the␣
↪first environment
    # If the field in the platform class has a default, consider it first
    if current_field.default not in environments:
        default_env = environment_choices[0]
    else:
        default_env = current_field.default

    # Return a dictionary that will be applied to the current field
    # Setting the new choices and default at runtime
    return {"choices": environment_choices, "default": default_env}
```

We can then use this function on the field, and the user will be prompted with the correct list of available environments.

```
@dataclass(repr=False)
class MyNewPlatform(IPlatform, CacheEnabled):
    endpoint: str = field(default="https://myapi.com", metadata={"help": "Enter the␣
↪URL of the endpoint"})
    environment: str = field(metadata={"help": "Select an environment ", "callback":␣
↪environment_list})
```

Fields validation

By default the CLI will provide validation on type. For example an `int` field, will only accept an integer value. To fine tune this validation, we can leverage the `validation` key of the metadata.

For example, if you want to create a field that has an integer value between 1 and 10, you can pass a validation function as shown:

```
def validate_number(value):
    if 1 <= value <= 10:
        return True, ''
    return False, "The value needs to be bewtween 1 and 10."

@dataclass(repr=False)
class MyNewPlatform(IPlatform, CacheEnabled):
    custom_validation: int = field(default=1, metadata={"help": "Enter a number␣
↪between 1 and 10.", "validation":validate_number})
```

The validation function will receive the user input as `value` and is expected to return a `bool` representing the result of the validation (`True` if the value is correct, `False` if not) and a `string` to give an error message to the user.

We can leverage the Python partials and make the validation function more generic to use in multiple fields:

```
from functools import partial

def validate_range(value, min, max):
    if min <= value <= max:
        return True, ''
    return False, f"The value needs to be between {min} and {max}."

@dataclass(repr=False)
class MyNewPlatform(IPlatform, CacheEnabled):
    custom_validation: int = field(default=1, metadata={"help": "Enter a number␣
↪between 1 and 10.", "validation":partial(validate_range, min=1, max=10)})
    custom_validation2: int = field(default=100, metadata={"help": "Enter a number␣
↪between 100 and 500.", "validation":partial(validate_range, min=100, max=500)})
```

1.4 Create and run simulations

To create simulations with idmtools, create a Python file that imports the relevant packages, uses the classes and functions to meet your specific needs, and then run the script using `python script_name.py`.

For example, if you would like to create many simulations "on-the-fly" (such as parameter sweeps) then you should use the *`SimulationBuilder`* and *`TemplatedSimulations`* classes. On the other hand, if you would like to create multiple simulations beforehand then you can use the *`Simulation`* class.

See the following examples for each of these scenarios:

1.4.1 SimulationBuilder example

```
"""
        This file demonstrates how to use ExperimentBuilder in PythonExperiment's builder.
        We are then adding the builder to PythonExperiment.

        Parameters for sweeping:
            |__ a = [0,1,2,3,4]

        Expect 5 sims with config parameters, note: "b" is not a sweep parameter, but it is depending on a's value:
            sim1: {a:0, b:2}
            sim2: {a:1, b:3}
            sim3: {a:2, b:4}
            sim4: {a:3, b:5}
            sim5: {a:4, b:6}
"""

import os
import sys
from functools import partial

from idmtools.builders import SimulationBuilder
from idmtools.core.platform_factory import platform
from idmtools.entities.experiment import Experiment
from idmtools.entities.templated_simulation import TemplatedSimulations
from idmtools_models.python.json_python_task import JSONConfiguredPythonTask
from idmtools_test import COMMON_INPUT_PATH


# define a custom sweep callback that sets b to a + 2
def param_update_ab(simulation, param, value):
    # Set B within
    if param == "a":
        simulation.task.set_parameter("b", value + 2)

    return simulation.task.set_parameter(param, value)


if __name__ == "__main__":
    # define what platform we want to use. Here we use a context manager but if you prefer you can
    # use objects such as Platform('COMPS2') instead
    with platform('COMPS2'):
        # define our base task
        base_task = JSONConfiguredPythonTask(script_path=os.path.join(COMMON_INPUT_PATH, "python", "model1.py"),
                                             parameters=dict(c='c-value'))

        # define our input csv sweep
        builder = SimulationBuilder()
        # Sweep parameter "a" and make "b" depends on "a"
        setAB = partial(param_update_ab, param="a")
        builder.add_sweep_definition(setAB, range(0, 5))

        # now define we want to create a series of simulations using the base task and the sweep
```

(continues on next page)

(continued from previous page)

```
        ts = TemplatedSimulations.from_task(base_task, tags=dict(c='c-value'))
        ts.add_builder(builder)

        # define our experiment with its metadata
        experiment = Experiment.from_template(ts,
                                              name=os.path.split(sys.argv[0])[1],
                                              tags={"string_tag": "test", "number_tag
↪": 123}
                                              )

        # run experiment
        experiment.run()
        # wait until done with longer interval
        # in most real scenarios, you probably do not want to wait as this will wait␣
↪until all simulations
        # associated with an experiment are done. We do it in our examples to show␣
↪feature and to enable
        # testing of the scripts
        experiment.wait(refresh_interval=10)
        # use system status as the exit code
        sys.exit(0 if experiment.succeeded else -1)
```

1.4.2 Simulation example

```
"""
        This file demonstrates how to use StandAloneSimulationsBuilder in␣
↪PythonExperiment's builder.

        we create 5 simulations and for each simulation, we set parameter 'a' = [0,4]␣
↪and 'b' = a + 10:
        then add each updated simulation to builder
        then we are adding the builder to PythonExperiment
"""
import copy
import os
import sys

from idmtools.assets import AssetCollection
from idmtools.core.platform_factory import Platform
from idmtools.entities.experiment import Experiment
from idmtools.entities.simulation import Simulation
from idmtools_models.python.json_python_task import JSONConfiguredPythonTask
from idmtools_test import COMMON_INPUT_PATH

if __name__ == "__main__":

    # define our platform
    platform = Platform('COMPS2')

    # create experiment  object and define some extra assets
    assets_path = os.path.join(COMMON_INPUT_PATH, "python", "Assets")
    e = Experiment(name=os.path.split(sys.argv[0])[1],
                   tags={"string_tag": "test", "number_tag": 123},
                   assets=AssetCollection.from_directory(assets_path))
```

(continues on next page)

(continued from previous page)

```
    # define paths to model and extra assets folder container more common assets
    model_path = os.path.join(COMMON_INPUT_PATH, "python", "model.py")

    # define our base task including the common assets. We could also add these␣
↪assets to the experiment above
    base_task = JSONConfiguredPythonTask(script_path=model_path, envelope='parameters
↪')

    base_simulation = Simulation.from_task(base_task)

    # now build our simulations
    for i in range(5):
        # first copy the simulation
        sim = copy.deepcopy(base_simulation)
        # configure it
        sim.task.set_parameter("a", i)
        sim.task.set_parameter("b", i + 10)
        # and add it to the simulations
        e.simulations.append(sim)

    # run the experiment
    e.run(platform=platform)
    # wait on it
    # in most real scenarios, you probably do not want to wait as this will wait␣
↪until all simulations
    # associated with an experiment are done. We do it in our examples to show␣
↪feature and to enable
    # testing of the scripts
    e.wait()
    # use system status as the exit code
    sys.exit(0 if e.succeeded else -1)
```

Many additional examples can be found in the /examples folder of the GitHub repository.

Create simulation tags

During the creation of simulations you can add tags, key:value pairs, included as metadata. The tags can be used for filtering on and searching for simulations. idmtools includes multiple ways for adding tags to simulations:

- (Preferred) Builder callbacks with *SimulationBuilder* or *Simulation*
- Base task with *TemplatedSimulations*
- Specific simulation from *TemplatedSimulations*

(Preferred) Builder callbacks via add_sweep_definition

You can add tags to simulations by using builder callbacks while building experiments with **SimulationBuilder** or **Simulation** classes and the **add_sweep_definition** method. This way supports adding tags to a large set of simulations and gives you full control over the simulation/task object. In addition, built-in tag management support is used when implementing the return values in a dictionary for the tags. For more information see the example in *SimulationBuilder*.

Base task with TemplatedSimulations

You can add tags to all simulations via base task used with the **TemplatedSimulations** class while building simulations. For more information see the example in *TemplatedSimulations*.

Specific simulation from TemplatedSimulations

If you need to add a tag to a specific simulation after building simulations from task with **TemplatedSimulations**, then you must convert the simulations to a list. For more information see the example in *TemplatedSimulations*.

Create EMOD simulations

To create simulations using EMOD you must use the **emodpy** package included with idmtools. Included with **emodpy** is the `emodpy.emod_task.EMODTask` class, inheriting from the *ITask* abstract class, and used for the running and configuration of EMOD simulations and experiments.

For more information about the architecture of job (simulation/experiment) creation and how EMOD leverages idmtools plugin architecture, see *Architecture and packages reference*.

The following Python excerpt shows an example of using **EMODTask** class and **from_default** method to create a task object using default config, campaign, and demographic values from **EMODSir** class and to use the Eradication.exe from local directory:

```
task = EMODTask.from_default(default=EMODSir(), eradication_path=os.path.join(BIN_
↪PATH, "Eradication"))
```

Another option, instead of using **from_default**, is to use the **from_files** method:

```
task = EMODTask.from_files(config_path=os.path.join(INPUT_PATH, "config.json"),
                           campaign_path=os.path.join(INPUT_PATH, "campaign.json"),
                           demographics_paths=os.path.join(INPUT_PATH, "demographics.
↪json"),
                           eradication_path=eradication_path)
```

For complete examples of the above see the following Python scripts:

- (from_default) emodpy.examples.create_sims_from_default_run_analyzer
- (from_files) emodpy.examples.create_sims_eradication_from_github_url

1.5 Parameter sweeps and model iteration

In modeling, parameter sweeps are an important method for fine-tuning parameter values, exploring parameter space, and calibrating simulations to data. A parameter sweep is an iterative process in which simulations are run repeatedly using different values of the parameter(s) of choice. This process enables the modeler to determine a parameter's "best" value (or range of values), or even where in parameter space the model produces desirable (or non-desirable) behaviors.

When fitting models to data, it is likely that there will be numerous parameters that do not have a pre-determined value. Some parameters will have a range of values that are biologically plausible, or have been determined from previous experiments; however, selecting a particular numerical value to use in the model may not be feasible or

realistic. Therefore, the best practice involves using a parameter sweep to narrow down the range of possible values or to provide a range of outcomes for those possible values.

idmtools provides an automated approach to parameter sweeps. With few lines of code, it is possible to test the model over any range of parameter values, with any combination of parameters.

Contents

With a stochastic model (such as EMOD), it is especially important to utilize parameter sweeps, not only for calibration to data or parameter selection, but to fully explore the stochasticity in output. Single model runs may appear to provide good fits to data, but variation will arise and multiple runs are necessary to determine the appropriate range of parameter values necessary to achieve desired outcomes. Multiple iterations of a single set of parameter values should be run to determine trends in simulation output: a single simulation output could provide results that are due to random chance.

1.5.1 How to do parameter sweeps

With idmtools, you can do parameter sweeps with builders or without builders using a base task to set your simulation parameters.

The typical 'output' of idmtools is a config.json file for each created simulation, which contains the parameter values assigned: both the constant values and those being swept.

Using builders

In this release, to support parameter sweeps for models, we have the following builders to assist you:

1. `SimulationBuilder` - you set your sweep parameters in your scripts and it generates a config.json file with your sweeps for your experiment/simulations to use
2. `CSVExperimentBuilder` - you can use a CSV file to do your parameter sweeps
3. `YamlSimulationBuilder` - you can use a Yaml file to do your parameter sweeps
4. `ArmSimulationBuilder` for cross and pair parameters, which allows you to cross parameters, like you cross your arms.

There are two types of sweeping, cross and pair. Cross means you have for example, 3 x 3 = 9 set of parameters, and pair means 3 + 3 = 3 pairs of parameters, for example, a, b, c and d, e, f.

For cross sweeping, let's say again you have parameters a, b, c and d, e, f that you want to cross so you would have the following possible matches: - a & d - a & e - a & f - b & d - b & e - b & f - c & d - c & e - c & f

For Python models, we also support them using a JSONConfiguredPythonTask. In the future we will support additional configured tasks for Python and R models.

Creating sweeps without builders

You can also create sweeps without using builders. Like this example:

```
"""
        This file demonstrates how to create param sweeps without builders.

        we create base task including our common assets, e.g. our python model to run
        we create 5 simulations and for each simulation, we set parameter 'a' = [0,4]␣
↪and 'b' = a + 10 using this task
        then we are adding this to task to our Experiment to run our simulations
"""
import copy
import os
import sys

from idmtools.assets import AssetCollection
from idmtools.core.platform_factory import Platform
from idmtools.entities.experiment import Experiment
from idmtools.entities.simulation import Simulation
from idmtools_models.python.json_python_task import JSONConfiguredPythonTask
from idmtools_test import COMMON_INPUT_PATH

if __name__ == "__main__":

    # define our platform
    platform = Platform('COMPS2')

    # create experiment  object and define some extra assets
    assets_path = os.path.join(COMMON_INPUT_PATH, "python", "Assets")
    e = Experiment(name=os.path.split(sys.argv[0])[1],
                   tags={"string_tag": "test", "number_tag": 123},
                   assets=AssetCollection.from_directory(assets_path))

    # define paths to model and extra assets folder container more common assets
    model_path = os.path.join(COMMON_INPUT_PATH, "python", "model.py")

    # define our base task including the common assets. We could also add these␣
↪assets to the experiment above
    base_task = JSONConfiguredPythonTask(script_path=model_path, envelope='parameters
↪')
    base_simulation = Simulation.from_task(base_task)

    # now build our simulations
    for i in range(5):
        # first copy the simulation
        sim = copy.deepcopy(base_simulation)
        # configure it
        sim.task.set_parameter("a", i)
        sim.task.set_parameter("b", i + 10)
        # and add it to the simulations
        e.simulations.append(sim)

    # run the experiment
    e.run(platform=platform)
    # wait on it
    # in most real scenarios, you probably do not want to wait as this will wait␣
↪until all simulations
```

(continues on next page)

(continued from previous page)

```
    # associated with an experiment are done. We do it in our examples to show␣
↪feature and to enable
    # testing of the scripts
    e.wait()
    # use system status as the exit code
    sys.exit(0 if e.succeeded else -1)
```

Running parameter sweeps in specific models

The following pages provide information about running parameter sweeps in particular models, and include example scripts.

Running parameter sweeps with R models

Running parameter sweeps with Python models

(include information about sweeps in python)

Examples

For Python modelers, we have multiple examples of how to do your parameter sweeps for Python models.

python_model.python_sim

`python_sim`

First, import some necessary system and idmtools packages.

- TemplatedSimulations: A utility that builds simulations from a template
- SimulationBuilder: An interface to different types of sweeps. It is used in conjunction with TemplatedSimulations.
- Platform: To specify the platform you want to run your experiment on
- JSONConfiguredPythonTask: We want to run an task executing a Python script. We will run a task in each simulation using this object. This particular task has a json config that is generated as well. There are other python task we either different or no configuration formats.

```
import os
import sys
from functools import partial
from typing import Any, Dict

from idmtools.builders import SimulationBuilder
from idmtools.core.platform_factory import Platform
from idmtools.entities.experiment import Experiment
from idmtools.entities.simulation import Simulation
from idmtools.entities.templated_simulation import TemplatedSimulations
from idmtools_models.python.json_python_task import JSONConfiguredPythonTask
```

We have python model defined in "model.py" which has 3 parameters: a, b, c and supports a json config from a file named "config".json. We want to sweep the parameters a for the values 0-2 and b for the values 1-3 and keep c as value 0.

To accomplish this, we are going to proceed in a few high-level steps. See https://bit.ly/37DHUf0 for workflow.

1. Define our base task. This task is the common configuration across all our tasks. For us, that means some basic run info like script path as well as our parameter/value we don't plan on sweeping, c.
2. Then we will define our TemplateSimulations object that will use our task to build a series of simulations.
3. Then we will define a SimulationBuilder and define our sweeps. This will involve also writing some callback functions that update the each task's config with the sweep values.
4. Then we will add our simulation builder to our TemplateSimulation object.
5. We will then build our Experiment object using the TemplateSimulations as our simulations list.
6. Lastly we will run the experiment on the platform.

First, let's define our base task. Normally, you want to do set any assets/configurations you want across the all the different Simulations we are going to build for our experiment. Here we set c to 0 since we do not want to sweep it.

```
task = JSONConfiguredPythonTask(script_path=os.path.join("inputs", "python_model_with_
↪deps", "Assets", "model.py"),
                                parameters=(dict(c=0)))
```

Now let's use this task to create a TemplatedSimulation builder. This will build new simulations from sweep builders we will define later. We can also use it to manipulate the base_task or the base_simulation.

```
ts = TemplatedSimulations(base_task=task)
```

We can define common metadata like tags across all the simulations using the base_simulation object.

```
ts.base_simulation.tags['tag1'] = 1
```

Since we have our templated simulation object now, let's define our sweeps.

To do that we need to use a builder:

```
builder = SimulationBuilder()
```

When adding sweep definitions, you need to generally provide two items.

See https://bit.ly/314j7xS for a diagram of how the Simulations are built using TemplateSimulations and SimulationBuilders.

1. A callback function that will be called for every value in the sweep. This function will receive a Simulation object and a value. You then define how to use those within the simulation. Generally, you want to pass those to your task's configuration interface. In this example, we are using JSONConfiguredPythonTask which has a set_parameter function that takes a Simulation, a parameter name, and a value. To pass to this function, we will user either a class wrapper or function partials.
2. A list/generator of values

Since our models uses a json config let's define an utility function that will update a single parameter at a time on the model and add that param/value pair as a tag on our simulation.

```
def param_update(simulation: Simulation, param: str, value: Any) -> Dict[str, Any]:
    """
    This function is called during sweeping allowing us to pass the generated sweep␣
↪values to our Task Configuration
```

(continues on next page)

(continued from previous page)

```
    We always receive a Simulation object. We know that simulations all have tasks␣
↪and that for our particular set
    of simulations they will all include JSONConfiguredPythonTask. We configure the␣
↪model with calls to set_parameter
    to update the config. In addition, we are can return a dictionary of tags to add␣
↪to the simulations so we return
    the output of the 'set_parameter' call since it returns the param/value pair we␣
↪set

    Args:
        simulation: Simulation we are configuring
        param: Param string passed to use
        value: Value to set param to

    Returns:

    """
    return simulation.task.set_parameter(param, value)
```

Let's sweep the parameter 'a' for the values 0-2. Since our utility function requires a Simulation, param, and value, the sweep framework calls our function with a Simulation and value. Let's use the partial function to define that we want the param value to always be "a" so we can perform our sweep.

```
setA = partial(param_update, param="a")
```

Now add the sweep to our builder:

```
builder.add_sweep_definition(setA, range(3))
```

```
# Example Python Experiment with JSON Configuration
# In this example, we will demonstrate how to run a python experiment with JSON␣
↪Configuration

# First, import some necessary system and idmtools packages.
# - TemplatedSimulations: A utility that builds simulations from a template
# - SimulationBuilder: An interface to different types of sweeps. It is used in␣
↪conjunction with TemplatedSimulations
# - Platform: To specify the platform you want to run your experiment on
# - JSONConfiguredPythonTask: We want to run an task executing a Python script. We␣
↪will run a task in each simulation
# using this object. This particular task has a json config that is generated as well.
↪ There are other python task
# we either different or no configuration formats.
import os
import sys
from functools import partial
from typing import Any, Dict

from idmtools.builders import SimulationBuilder
from idmtools.core.platform_factory import Platform
from idmtools.entities.experiment import Experiment
from idmtools.entities.simulation import Simulation
from idmtools.entities.templated_simulation import TemplatedSimulations
from idmtools_models.python.json_python_task import JSONConfiguredPythonTask

```

(continues on next page)

(continued from previous page)

```
# We have python model defined in "model.py" which has 3 parameters: a, b, c and supports
# a json config from a file named "config".json. We want to sweep the parameters a for the values 0-2 and b for the
# values 1-3 and keep c as value 0.
# To accomplish this, we are going to proceed in a few high-level steps. See https://bit.ly/37DHUf0 for workflow
# 1. Define our base task. This task is the common configuration across all our tasks. For us, that means some basic
#    run info like script path as well as our parameter/value we don't plan on sweeping, c
# 2. Then we will define our TemplateSimulations object that will use our task to build a series of simulations
# 3. Then we will define a SimulationBuilder and define our sweeps. This will invlove also writing some callback
# functions that update the each task's config with the swep values
# 4. Then we will add our simulation builder to our TemplateSimulation object.
# 5. We will then build our Experiment object using the TemplateSimulations as our simulations list.
# 6. Lastly we will run the experiment on the platform

# first let's define our base task. Normally, you want to do set any assets/configurations you want across the
# all the different Simulations we are going to build for our experiment. Here we set c to 0 since we do not want to
# sweep it
task = JSONConfiguredPythonTask(script_path=os.path.join("inputs", "python_model_with_deps", "Assets", "model.py"),
                                parameters=(dict(c=0)))

# now let's use this task to create a TemplatedSimulation builder. This will build new simulations from sweep builders
# we will define later. We can also use it to manipulate the base_task or the base_simulation
ts = TemplatedSimulations(base_task=task)
# We can define common metadata like tags across all the simulations using the base_simulation object
ts.base_simulation.tags['tag1'] = 1

# Since we have our templated simulation object now, let's define our sweeps
# To do that we need to use a builder
builder = SimulationBuilder()

# When adding sweep definitions, you need to generally provide two items
# See https://bit.ly/314j7xS for a diagram of how the Simulations are built using TemplateSimulations +
# SimulationBuilders
# 1. A callback function that will be called for every value in the sweep. This function will receive a Simulation
#    object and a value. You then define how to use those within the simulation. Generally, you want to pass those
#    to your task's configuration interface. In this example, we are using JSONConfiguredPythonTask which has a
#    set_parameter function that takes a Simulation, a parameter name, and a value. To pass to this function, we will
#    user either a class wrapper or function partials
# 2. A list/generator of values
```

(continues on next page)

(continued from previous page)

```

# Since our models uses a json config let's define an utility function that will update a single parameter at a
# time on the model and add that param/value pair as a tag on our simulation.


def param_update(simulation: Simulation, param: str, value: Any) -> Dict[str, Any]:
    """
    This function is called during sweeping allowing us to pass the generated sweep values to our Task Configuration

    We always receive a Simulation object. We know that simulations all have tasks and that for our particular set
    of simulations they will all include JSONConfiguredPythonTask. We configure the model with calls to set_parameter
    to update the config. In addition, we are can return a dictionary of tags to add to the simulations so we return
    the output of the 'set_parameter' call since it returns the param/value pair we set

    Args:
        simulation: Simulation we are configuring
        param: Param string passed to use
        value: Value to set param to

    Returns:

    """
    return simulation.task.set_parameter(param, value)


# Let's sweep the parameter 'a' for the values 0-2. Since our utility function requires a Simulation, param, and value
# but the sweep framework all calls our function with Simulation, value, let's use the partial function to define
# that we want the param value to always be "a" so we can perform our sweep
setA = partial(param_update, param="a")
# now add the sweep to our builder
builder.add_sweep_definition(setA, range(3))


# An alternative to using partial is define a class that store the param and is callable later. let's use that technique
# to perform a sweep one the values 1-3 on parameter b

# First define our class. The trick here is we overload __call__ so that after we create the class, and calls to the
# instance will be relayed to the task in a fashion identical to the param_update function above. It is generally not
# best practice to define a class like this in the body of our main script so it is advised to place this in a library
# or at the very least the top of your file.
class setParam:
    def __init__(self, param):
        self.param = param

    def __call__(self, simulation, value):
```

(continues on next page)

(continued from previous page)

```
        return simulation.task.set_parameter(self.param, value)


# Now add our sweep on a list
builder.add_sweep_definition(setParam("b"), [1, 2, 3])
ts.add_builder(builder)

# Now we can create our Experiment using our template builder
experiment = Experiment.from_template(ts, name=os.path.split(sys.argv[0])[1])
# Add our own custom tag to simulation
experiment.tags["tag1"] = 1
# And maybe some custom Experiment Level Assets
experiment.assets.add_directory(assets_directory=os.path.join("inputs", "python_model_
↪with_deps", "Assets"))

# In order to run the experiment, we need to create a `Platform`
# The `Platform` defines where we want to run our simulation.
# You can easily switch platforms by changing the Platform to for example 'Local'
with Platform('COMPS2'):

    # The last step is to call run() on the ExperimentManager to run the simulations.
    experiment.run(True)
    # use system status as the exit code
    sys.exit(0 if experiment.succeeded else -1)
```

python_model.python_SEIR_sim

python_SEIR_sim

Example Python Experiment with JSON Configuration

In this example, we will demonstrate how to run a python experiment with JSON Configuration.

First, import some necessary system and idmtools packages:

```
import os
import sys
import json
from functools import partial
from typing import Any, Dict

from idmtools.analysis.analyze_manager import AnalyzeManager
from idmtools.builders import SimulationBuilder
from idmtools.core import ItemType
from idmtools.core.platform_factory import Platform
from idmtools.entities.experiment import Experiment
from idmtools.entities.simulation import Simulation
from idmtools.entities.templated_simulation import TemplatedSimulations
from idmtools_models.python.json_python_task import JSONConfiguredPythonTask
from inputs.ye_seir_model.custom_csv_analyzer import NodeCSVAnalyzer,␣
↪InfectiousnessCSVAnalyzer
```

Define some constant string used in this example:

```
class ConfigParameters():
    Infectious_Period_Constant = "Infectious_Period_Constant"
```

(continues on next page)

(continued from previous page)

```
    Base_Infectivity_Constant = "Base_Infectivity_Constant"
    Base_Infectivity_Distribution = "Base_Infectivity_Distribution"
    GAUSSIAN_DISTRIBUTION = "GAUSSIAN_DISTRIBUTION"
    Base_Infectivity_Gaussian_Mean = "Base_Infectivity_Gaussian_Mean"
    Base_Infectivity_Gaussian_Std_Dev = "Base_Infectivity_Gaussian_Std_Dev"
```

Script need to be in a main block, other wise AnalyzerManager will have issue with multi threads in Windows OS.

```
if __name__ == '__main__':
```

We have python model defined in "SEIR_model.py" which takes several arguments like "–duration" and "–outbreak_coverage", and supports a json config from a file named "nd_template.json". We want to sweep some arguments passed in to "SEIR_model.py" and some parameters in "nd_template.json".

To accomplish this, we are going to proceed in a few high-level steps. See https://bit.ly/37DHUf0 for workflow

1. Define our base task. This task is the common configuration across all our tasks. For us, that means some basic run info like script path as well as our arguments/value and parameter/value we don't plan on sweeping, "–duration", and most of the parameters inside "nd_template.json".
2. Then we will define our TemplateSimulations object that will use our task to build a series of simulations
3. Then we will define a SimulationBuilder and define our sweeps. This will involve also writing some callback functions that update the each task's config or option with the sweep values
4. Then we will add our simulation builder to our TemplateSimulation object.
5. We will then build our Experiment object using the TemplateSimulations as our simulations list.
6. We will run the experiment on the platform
7. Once and experiment is succeeded, we run two CSV analyzer to analyze results from the python model.

1. First, let's define our base task. Normally, you want to do set any assets/configurations you want across the all the different Simulations we are going to build for our experiment. Here we load config file from a template json file and rename the config_file_name (default value is config.json).

```
parameters = json.load(open(os.path.join("inputs", "ye_seir_model", "Assets",
↪"templates", 'seir_configuration_template.json'), 'r'))
parameters[ConfigParameters.Base_Infectivity_Distribution] = ConfigParameters.
↪GAUSSIAN_DISTRIBUTION
task = JSONConfiguredPythonTask(script_path=os.path.join("inputs", "ye_seir_model",
↪"Assets", "SEIR_model.py"),
                                parameters=parameters,
                                config_file_name="seir_configuration_template.json")
```

We define arguments/value for simulation duration that we don't want to sweep as an option for the task.

```
task.command.add_option("--duration", 40)
```

2. Now, let's use this task to create a TemplatedSimulation builder. This will build new simulations from sweep builders we will define later. We can also use it to manipulate the base_task or the base_simulation .. code-block:: python

    ts = TemplatedSimulations(base_task=task)

We can define common metadata like tags across all the simulations using the base_simulation object

```
ts.base_simulation.tags['simulation_name_tag'] = "SEIR_Model"
```

3. Since we have our templated simulation object now, let's define our sweeps.

To do that we need to use a builder:

```
builder = SimulationBuilder()
```

When adding sweep definitions, you need to generally provide two items.

See https://bit.ly/314j7xS for a diagram of how the Simulations are built using TemplateSimulations +

SimulationBuilders

3.1. A callback function that will be called for every value in the sweep. This function will receive a Simulation object and a value. You then define how to use those within the simulation. Generally, you want to pass those to your task's configuration interface. In this example, we are using JSONConfiguredPythonTask which has a set_parameter function that takes a Simulation, a parameter name, and a value. To pass to this function, we will user either a class wrapper or function partials

3.2. A list/generator of values

Since our models uses a json config let's define an utility function that will update a single parameter at a time on the model and add that param/value pair as a tag on our simulation.

```
def param_update(simulation: Simulation, param: str, value: Any) -> Dict[str, Any]:
    """
    This function is called during sweeping allowing us to pass the generated sweep␣
↪values to our Task Configuration

    We always receive a Simulation object. We know that simulations all have tasks␣
↪and that for our particular set
    of simulations they will all include JSONConfiguredPythonTask. We configure the␣
↪model with calls to set_parameter
    to update the config. In addition, we are can return a dictionary of tags to add␣
↪to the simulations so we return
    the output of the 'set_parameter' call since it returns the param/value pair we␣
↪set

    Args:
        simulation: Simulation we are configuring
        param: Param string passed to use
        value: Value to set param to

    Returns:

    """
    return simulation.task.set_parameter(param, value)
```

Let's sweep the parameter 'Base_Infectivity_Gaussian_Mean' for the values 0.5 and 2. Since our utility function requires a Simulation, param, and value but the sweep framework all calls our function with Simulation, value, let's use the partial function to define that we want the param value to always be "Base_Infectivity_Gaussian_Mean" so we can perform our sweep set_base_infectivity_gaussian_mean = partial(param_update, param=ConfigParameters.Base_Infectivity_Gaussian_Mean) now add the sweep to our builder builder.add_sweep_definition(set_base_infectivity_gaussian_mean, [0.5, 2]).

An alternative to using partial is define a class that store the param and is callable later. let's use that technique to perform a sweep one the values 1 and 2 on parameter Base_Infectivity_Gaussian_Std_Dev.

First define our class. The trick here is we overload __call__ so that after we create the class, and calls to the instance will be relayed to the task in a fashion identical to the param_update function above. It is generally not best practice

to define a class like this in the body of our main script so it is advised to place this in a library or at the very least the top of your file.

```
class setParam:
    def __init__(self, param):
        self.param = param

    def __call__(self, simulation, value):
        return simulation.task.set_parameter(self.param, value)
```

Now add our sweep on a list: .. code-block:: python

    builder.add_sweep_definition(setParam(ConfigParameters.Base_Infectivity_Gaussian_Std_Dev), [0.3, 1])

Using the same methodologies, we can sweep on option/arguments that pass to our Python model. You can uncomment the following code to enable it.

3.3 First method:

```
# def option_update(simulation: Simulation, option: str, value: Any) -> Dict[str,␣
↪Any]:
#     simulation.task.command.add_option(option, value)
#     return {option: value}
# set_outbreak_coverage = partial(option_update, option="--outbreak_coverage")
# builder.add_sweep_definition(set_outbreak_coverage, [0.01, 0.1])
#
# # 3.4 second method:
# class setOption:
#     def __init__(self, option):
#         self.option = option
#
#     def __call__(self, simulation, value):
#         simulation.task.command.add_option(self.option, value)
#         return {self.option: value}
# builder.add_sweep_definition(setOption("--population"), [1000, 4000])
```

4. Add our builder to the template simulations

```
ts.add_builder(builder)
```

5. Now we can create our Experiment using our template builder

```
experiment = Experiment(name=os.path.split(sys.argv[0])[1], simulations=ts)
```

Add our own custom tag to simulation

```
experiment.tags['experiment_name_tag'] = "SEIR_Model"
```

And maybe some custom Experiment Level Assets

```
experiment.assets.add_directory(assets_directory=os.path.join("inputs", "ye_seir_model
↪", "Assets"))
```

6. In order to run the experiment, we need to create a *Platform*.

The *Platform* defines where we want to run our simulation.

You can easily switch platforms by changing the Platform to for example 'Local'

```
platform = Platform('COMPS2')
```

The last step is to call run() on the ExperimentManager to run the simulations.

```
platform.run_items(experiment)
platform.wait_till_done(experiment)
```

Check experiment status, only move to Analyzer step if experiment succeeded.

```
if not experiment.succeeded:
    print(f"Experiment {experiment.uid} failed.\n")
    sys.exit(-1)
```

7. Now let's look at the experiment results. Here are two outputs we want to analyze.

```
filenames = ['output/individual.csv']
filenames_2 = ['output/node.csv']
```

Initialize two analyser classes with the path of the output csv file

```
analyzers = [InfectiousnessCSVAnalyzer(filenames=filenames),␣
↪NodeCSVAnalyzer(filenames=filenames_2)]
```

Specify the id Type, in this case an Experiment on COMPS

```
manager = AnalyzeManager(configuration={}, partial_analyze_ok=True, platform=platform,
                         ids=[(experiment.uid, ItemType.EXPERIMENT)],
                         analyzers=analyzers)
```

Now analyze:

```
manager.analyze()
sys.exit(0)
```

Note: To access and use COMPS you must receive approval and credentials from IDM. Send your request to support@idmod.org.

python_model.python_model_allee

`python_model_allee`

In this example, we will demonstrate how to run a python experiment.

First, import some necessary system and idmtools packages. - ExperimentBuilder: To create sweeps - ExperimentManager: To manage our experiment - Platform: To specify the platform you want to run your experiment on - PythonExperiment: We want to run an experiment executing a Python script

```
import os
import sys
from functools import partial

from idmtools.assets import AssetCollection
from idmtools.builders import SimulationBuilder
from idmtools.core.platform_factory import Platform
```

In order to run the experiment, we need to create a *Platform* and an *ExperimentManager*.

The *Platform* defines where we want to run our simulation.

You can easily switch platforms by changing the Platform to for example 'Local' with Platform('Local'):

```
from idmtools.entities.experiment import Experiment
from idmtools.entities.templated_simulation import TemplatedSimulations
from idmtools_models.python.json_python_task import JSONConfiguredPythonTask
from idmtools_platform_comps.utils.python_requirements_ac.requirements_to_asset_
↪collection import RequirementsToAssetCollection

platform = Platform('COMPS2')

pl = RequirementsToAssetCollection(platform,
                                   requirements_path=os.path.join("inputs", "allee_
↪python_model", "requirements.txt"))

ac_id = pl.run()
pandas_assets = AssetCollection.from_id(ac_id, platform=platform)

base_task = JSONConfiguredPythonTask(
    # specify the path to the script. This is most likely a scientific model
    script_path=os.path.join("inputs", "allee_python_model", "run_emod_sweep.py"),
    envelope='parameters',
    parameters=dict(
        fname="runNsim100.json",
        customGrid=1,
        nsims=100
    ),
    common_assets=pandas_assets
)
```

Update and set simulation configuration parameters.

```
def param_update(simulation, param, value):
    return simulation.task.set_parameter(param, 'sweepR04_a_' + str(value) + '.json')

setA = partial(param_update, param="infile")
```

Define our template:

```
ts = TemplatedSimulations(base_task=base_task)
```

Now that the experiment is created, we can add sweeps to it and set additional params

```
builder = SimulationBuilder()
builder.add_sweep_definition(setA, range(7850, 7855))
```

Add sweep builder to template:

```
ts.add_builder(builder)
```

Create experiment:

```
e = Experiment.from_template(
    ts,
    name=os.path.split(sys.argv[0])[1],
```

(continues on next page)

(continued from previous page)

```
    assets=AssetCollection.from_directory(os.path.join("inputs", "allee_python_model
↪"))
)

platform.run_items(e)
```

Use system status as the exit code:

```
sys.exit(0 if e.succeeded else -1)
```

Running parameter sweeps with EMOD

When running parameter sweeps with EMOD, you use the `EMODTask` class for setting the sweep parameters and passing them to the `SimulationBuilder` class using the `add_sweep_definition` method.

In addition to the parameters for sweeping, you must also set the **Run_Number** parameter. This determines the seed for the random number generator. This is particularly important with EMOD in order to explore the stochastic nature of the model. Otherwise, if **Run_Number** is not changed then each simulation will result in the same output.

The following python code excerpt shows an example:

```
# Create TemplatedSimulations with task
ts = TemplatedSimulations(base_task=task)

# Create SimulationBuilder
builder = SimulationBuilder()

# Add sweep parameter to builder
builder.add_sweep_definition(EMODTask.set_parameter_partial("Run_Number"), range(num_
↪seeds))

# Add another sweep parameter to builder
builder.add_sweep_definition(EMODTask.set_parameter_partial("Base_Infectivity"), [0.6,
↪ 1.0, 1.5, 2.0])

# Add builder to templated simulations
ts.add_builder(builder)
```

You can run a parameter sweep using the above code excerpt by running the included example, `create_sims_eradication_from_github_url`.

1.6 Output data

The output produced by running simulations using idmtools depends on the configuration of the model itself. idmtools is itself agnostic to the output format when running simulations. However, the analysis framework expects simulation output in CSV, JSON, XLSX, or TXT to be automatically loaded to a Python object. All other formats are loaded as a raw binary stream. For more information, see *Introduction to analyzers*.

If you are running simulations on COMPS, the configuration of the `idmtools.ini` file will determine where output files can be found. For more information, see *idmtools.ini wizard*

Note: To access and use COMPS you must receive approval and credentials from IDM. Send your request to support@idmod.org.

If you are running simulations or experiments locally, they are saved to your local computer at C:\Users\yourname\.local_data\workers for Windows and ~/.local_data/workers for Linux.

Additionally, when running locally using Docker, output can be found in your browser in the output directory appended after the experiment or simulation ID. For example, the output from an experiment with an ID of S07OASET could be found at http://localhost:5000/data/S07OASET. The output from an individual simulation (ID FCPRIV7H) within that experiment could be found at http://localhost:5000/data/S07OASET/FCPRIV7H.

The python_csv_output.py example below demonstrates how to produce output in CSV format for a simple parameter sweep.

```
# Example Python Experiment
# In this example, we will demonstrate how to run a python experiment.

# First, import some necessary system and idmtools packages.
# - TemplatedSimulations: To create simulation from a template
# - ExperimentManager: To manage our experiment
# - platform: To specify the platform you want to run your experiment on as a context␣
↪object
# - JSONConfiguredPythonTask: We want to run an experiment executing a Python script␣
↪that uses a JSON configuration file
import os
import sys

from idmtools.assets import AssetCollection
from idmtools.builders import SimulationBuilder
from idmtools.core.platform_factory import platform
from idmtools.entities.experiment import Experiment
from idmtools.entities.templated_simulation import TemplatedSimulations
from idmtools_models.python.json_python_task import JSONConfiguredPythonTask

# In order to run the experiment, we need to create a `Platform` and an␣
↪`ExperimentManager`.
# The `Platform` defines where we want to run our simulation.
# You can easily switch platforms by changing the Platform to for example 'Local'
with platform('COMPS2'):
    # define our base task as a python model with json config
    base_task = JSONConfiguredPythonTask(
        script_path=os.path.join("inputs", "csv_inputs", "Assets", "model.py"),
        # set the default parameters to 0
        parameters=(dict(c=0)),
        # add some experiment level assets
        common_assets=AssetCollection.from_directory(os.path.join("inputs", "csv_
↪inputs", "Assets"))
    )

    # create a templating object using the base task
    ts = TemplatedSimulations(base_task=base_task)
    # Define the parameters we are going to want to sweep
    builder = SimulationBuilder()
    # define two partial callbacks so we can use the built in sweep callback function␣
↪on the model
    # Since we want to sweep per parameter, and we want need to define a partial for␣
↪each parameter
```

(continues on next page)

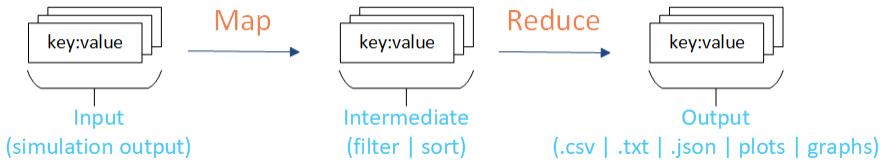
(continued from previous page)

```
    # The JSON model provides utility function for this puprose
    builder.add_sweep_definition(JSONConfiguredPythonTask.set_parameter_partial("a"),␣
↪range(3))
    builder.add_sweep_definition(JSONConfiguredPythonTask.set_parameter_partial("b"),␣
↪[1, 2, 3])
    # add the builder to our template
    ts.add_builder(builder)

    # now build experiment
    e = Experiment.from_template(
        ts,
        name=os.path.split(sys.argv[0])[1],
        tags=dict(tag1=1))

    # now we can run the experiment
    e.run()
    # and wait
    e.wait()
    # use system status as the exit code
    sys.exit(0 if e.succeeded else -1)
```

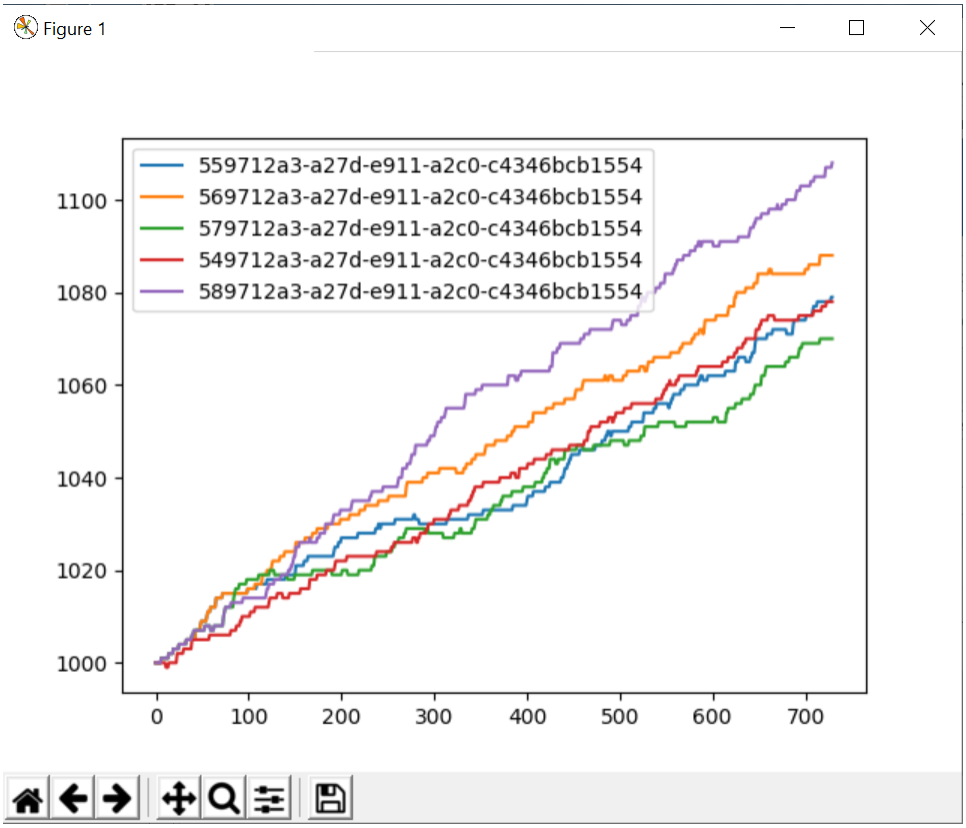
1.7 Introduction to analyzers

The analyzers and examples in idmtools provide support for the MapReduce framework, where you can process large data sets in parallel, typically on a *high-performance computing (HPC)* cluster. The MapReduce framework includes two primary phases, Map and Reduce. Map takes input data, as key:value pairs, and creates an intermediate set of key:value pairs. Reduce takes the intermediate set of key:value pairs and transforms the data (typically reducing it) as output containing a final set of key:value pairs.

An example of this process with idmtools is to use the simulation output data as the input data (key:value pairs), filter and sort a subset of the data to focus on, and then combine and reduce the data to create the final output data.



The analyzers included with idmtools help facilitate this process. For example, if you would like to focus on specific data points from all simulations in one or more experiments then you can do this using analyzers with idmtools and plot the final output.



The analysis framework expects simulation output in CSV, JSON, XLSX, or TXT to be automatically loaded to a Python object. All other formats are loaded as a raw binary stream. The format indicated by the filename of the simulation output determines the data format loaded to the analyzers.

Output format	Object loaded to analyzer
JSON	A dictionary
CSV	A pandas DataFrame
XLSX	A pandas DataFrame
TXT	An rstring
All other files	A bytes object

Example analyzers are inlucded with idmtools to help you get started. For more information, see *Example analyzers*.

You can also create custom analyzers to meet your individual analysis needs. For more information, see *Create an analyzer*.

Integration with Server-Side Modeling Tools (SSMT) increases the performance of running analyzers. You may find this useful when running multiple analyzers across multiple experiments.

1.7.1 Example analyzers

You can use the following example analyzers as templates to get started using idmtools:

idmtools.analysis.add_analyzer.AddAnalyzer
idmtools.analysis.csv_analyzer.CSVAnalyzer
idmtools.analysis.download_analyzer.DownloadAnalyzer
idmtools.analysis.tags_analyzer.TagsAnalyzer

Each example analyzer is configured to run with existing simulation data and already configured options, such as using the COMPS platform and existing experiments. This allows you to easily run these example analyzers for demonstrating some of the tasks you may want to accomplish when analyzing simulation output data. You can then use and modify these examples for your specific needs.

Note: To access and use COMPS you must receive approval and credentials from IDM. Send your request to support@idmod.org.

For a description of each of these analyzers please see the following:

- *AddAnalyzer*: Gets metadata from simulations, maps to key:value pairs, and returns a .txt output file.
- *CSVAnalyzer*: Analyzes .csv output files from simulations and returns a .csv output file.
- *DownloadAnalyzer*: Downloads simulation output files for analysis on local computer resources.
- *Multiple CSV Example*: Analyzes multiple .csv output files from simulations and returns a .csv output file.
- *TagsAnalyzer*: Analyzes tags from simulations and returns a .csv output file.

Each of the included example analyzers inherit from the built-in analyzers and the *IAnalyzer* abstract class:

For more information about the built-in analyzers, see *Create an analyzer*. There are also additional examples, such as forcing analyzers to use a specific working directory and how to perform partial analysis on only succeeded or failed simulations:

Force working directory

You can force analyzers to use a specific working directory other than the default, which is the directory from which the analyzer is run. For example, if you install idmtools to the **\idmtools** directory and then run one of the example analyzers from their default directory, **\examples\analyzers**, then the default working directory would be **\idmtools\examples\analyzers**.

To force a working directory, you use the **force_manager_working_directory** parameter from the *AnalyzeManager* class. The following python code, using the *DownloadAnalyzer* as an example , illustrates different ways on how to use and configure the **force_manager_working_directory** parameter and how it works and interacts with the **working_dir** parameter:

```
from idmtools.analysis.analyze_manager import AnalyzeManager
from idmtools.analysis.download_analyzer import DownloadAnalyzer
from idmtools.core import ItemType
from idmtools.core.platform_factory import Platform

```

(continues on next page)

(continued from previous page)

```
if __name__ == '__main__':
    platform = Platform('COMPS2')
    filenames = ['StdOut.txt']
    experiment_id = '11052582-83da-e911-a2be-f0921c167861'  # comps2 staging exp id

# force_manager_working_directory = False (default value):
# Analyzers will use their own specified working_dir if available. If not, the␣
↪AnalyzeManager
# specified working_dir will be used (default: '.').
#
# force_manager_working_directory = True
# Analyzers will use the AnalyzeManager specified working_dir (default: '.')

# Examples

# This will use the default working_dir for both analyzers (the current run directory,
↪ '.')
analyzers = [DownloadAnalyzer(filenames=filenames, output_path='DL1'),
             DownloadAnalyzer(filenames=filenames, output_path='DL2')]
manager = AnalyzeManager(platform=platform, ids=[(experiment_id, ItemType.
↪EXPERIMENT)],
                         analyzers=analyzers)
manager.analyze()

# This will use the manager-specified working_dir for both analyzers
analyzers = [DownloadAnalyzer(filenames=filenames, output_path='DL1'),
             DownloadAnalyzer(filenames=filenames, output_path='DL2')]
manager = AnalyzeManager(platform=platform, ids=[(experiment_id, ItemType.
↪EXPERIMENT)],
                         analyzers=analyzers, working_dir='use_this_working_dir_for_
↪both_analyzers')
manager.analyze()

# This will use the analyzer-specified working_dir for DL1 and the manager-specified␣
↪dir for DL2
analyzers = [DownloadAnalyzer(filenames=filenames, output_path='Dl1', working_dir=
↪'DL1_working_dir'),
             DownloadAnalyzer(filenames=filenames, output_path='DL2')]
manager = AnalyzeManager(platform=platform, ids=[(experiment_id, ItemType.
↪EXPERIMENT)],
                         analyzers=analyzers, working_dir='use_this_working_dir_if_
↪not_set_by_analyzer')
manager.analyze()

# This will use the manager-specified dir for both DL1 and DL2, even though DL1 tried␣
↪to set its own
analyzers = [DownloadAnalyzer(filenames=filenames, output_path='DL1', working_dir=
↪'DL1_working_dir'),
             DownloadAnalyzer(filenames=filenames, output_path='DL2')]
manager = AnalyzeManager(platform=platform, ids=[(experiment_id, ItemType.
↪EXPERIMENT)],
                         analyzers=analyzers, working_dir='use_this_working_dir_if_
↪not_set_by_analyzer',
                         force_manager_working_directory=True)
manager.analyze()
```

Partial analysis

You can use analyzers for a partial analysis of simulations. This allows you to only analyze succeeded simulations, while one or more simulations within an experiment may have failed. In addition, you can analyze both succeeded and failed simulations.

Analysis on only succeeded simulations

For partial analysis only on the succeeded simulations, where one or more simulations may have failed, you set to **True** the **partial_analyze_ok** parameter from the *AnalyzeManager* class, as seen in the following python code excerpt:

```
analyzers = [CSVAnalyzer(filenames=filenames)]
manager = AnalyzeManager(platform=self.platform, partial_analyze_ok=True,
                         ids=[(experiment_id, ItemType.EXPERIMENT)],
                         analyzers=analyzers)
manager.analyze()
```

Analysis on both succeeded and failed simulations

For analysis on both succeeded and failed simulations, you set to **True** the **analyze_failed_items** parameter from the *AnalyzeManager* class, as seen in the following python code excerpt:

```
analyzers = [CSVAnalyzer(filenames=filenames)]
manager = AnalyzeManager(platform=self.platform, analyze_failed_items=True,
                         ids=[(experiment_id, ItemType.EXPERIMENT)],
                         analyzers=analyzers)
manager.analyze()
```

1.7.2 Create an analyzer

You can use built-in analyzers included with idmtools to help with creating a new analyzer. The following list some of these analyzers, all inheriting from the the *IAnalyzer* abstract class:

For more information about these built-in analyzers, see:

- *AddAnalyzer*
- *CSVAnalyzer*
- *DownloadAnalyzer*
- *TagsAnalyzer*

To create an analyzer methods from the *IAnalyzer* abstract class are used:

All analyzers must also call the *AnalyzeManager* class for analysis management:

The following python code and comments, from the *CSVAnalyzer* class, is an example of how to create an analyzer for analysis of .csv output files from simulations:

```
class CSVAnalyzer(IAnalyzer):
# Arg option for analyzer init are uid, working_dir, parse (True to leverage the␣
↪:class:`OutputParser`;
# False to get the raw data in the :meth:`select_simulation_data`), and filenames
# In this case, we want parse=True, and the filename(s) to analyze
def __init__(self, filenames, parse=True):
    super().__init__(parse=parse, filenames=filenames)
    # Raise exception early if files are not csv files
    if not all(['csv' in os.path.splitext(f)[1].lower() for f in self.filenames]):
        raise Exception('Please ensure all filenames provided to CSVAnalyzer have a␣
↪csv extension.')

def initialize(self):
    if not os.path.exists(os.path.join(self.working_dir, "output_csv")):
        os.mkdir(os.path.join(self.working_dir, "output_csv"))

# Map is called to get for each simulation a data object (all the metadata of the␣
↪simulations) and simulation object
def map(self, data, simulation):
    # If there are 1 to many csv files, concatenate csv data columns into one␣
↪dataframe
    concatenated_df = pd.concat(list(data.values()), axis=0, ignore_index=True,␣
↪sort=True)
    return concatenated_df

# In reduce, we are printing the simulation and result data filtered in map
def reduce(self, all_data):

    results = pd.concat(list(all_data.values()), axis=0,  # Combine a list of all the␣
↪sims csv data column values
                        keys=[str(k.uid) for k in all_data.keys()],  # Add a␣
↪hierarchical index with the keys option
                        names=['SimId'])  # Label the index keys you create with the␣
↪names option
    results.index = results.index.droplevel(1)  # Remove default index

    # Make a directory labeled the exp id to write the csv results to
    # NOTE: If running twice with different filename, the output files will collide
    results.to_csv(os.path.join("output_csv", self.__class__.__name__ + '.csv'))
```

You can quickly see this analyzer in use by running the included `example_analysis_CSVAnalyzer` example class.

1.7.3 Convert analyzers from DTK-Tools to idmtools

Although the use of analyzers in DTK-Tools and idmtools is very similar, being aware of some of the differences may be helpful with the conversion process. For example some of the class and method names are different, as seen in the following diagram:

For additional information about the *IAnalyzer* class and methods, see *IAnalyzer*.

In addition, you can also see an example of a .csv analyzer created in DTK-Tools and how it was converted to idmtools. Other than the class name and some method names changing the core code is almost the same. The primary differences can be seen in the class import statements and the execution of the analysis within the *if __name__ == '__main__':* block of code.

DTK-Tools example analyzer

The following DTK-Tools example performs analysis on simulation output data in .csv files and returns the result data in a .csv file:

```
import os
import pandas as pd
from simtools.Analysis.BaseAnalyzers import BaseAnalyzer
from simtools.Analysis.AnalyzeManager import AnalyzeManager
from simtools.SetupParser import SetupParser


class CSVAnalyzer(BaseAnalyzer):

    def __init__(self, filenames, parse=True):
        super().__init__(parse=parse, filenames=filenames)
        if not all(['csv' in os.path.splitext(f)[1].lower() for f in self.filenames]):
            raise Exception('Please ensure all filenames provided to CSVAnalyzer have␣
↪a csv extension.')

    def initialize(self):
        if not os.path.exists(os.path.join(self.working_dir, "output_csv")):
            os.mkdir(os.path.join(self.working_dir, "output_csv"))

    def select_simulation_data(self, data, simulation):
        concatenated_df = pd.concat(list(data.values()), axis=0, ignore_index=True,␣
↪sort=True)
        return concatenated_df

    def finalize(self, all_data: dict) -> dict:

        results = pd.concat(list(all_data.values()), axis=0,
                            keys=[k.id for k in all_data.keys()],
                            names=['SimId'])
        results.index = results.index.droplevel(1)

        results.to_csv(os.path.join("output_csv", self.__class__.__name__ + '.csv'))


if __name__ == "__main__":

    SetupParser.init(selected_block='HPC', setup_file="simtools.ini")
    filenames = ['output/c.csv']
    analyzers = [CSVAnalyzer(filenames=filenames)]
    manager = AnalyzeManager('9311af40-1337-ea11-a2be-f0921c167861',␣
↪analyzers=analyzers)
    manager.analyze()
```

DTK-Tools converted to idmtools

The following converted from DTK-Tools to idmtools example performs analysis on simulation output data in .csv files and returns the result data in a .csv file:

```
import os
import pandas as pd
from idmtools.entities import IAnalyzer
from idmtools.analysis.analyze_manager import AnalyzeManager
from idmtools.core import ItemType
from idmtools.core.platform_factory import Platform


class CSVAnalyzer(IAnalyzer):

    def __init__(self, filenames, parse=True):
        super().__init__(parse=parse, filenames=filenames)
        if not all(['csv' in os.path.splitext(f)[1].lower() for f in self.filenames]):
            raise Exception('Please ensure all filenames provided to CSVAnalyzer have␣
↪a csv extension.')

    def initialize(self):
        if not os.path.exists(os.path.join(self.working_dir, "output_csv")):
            os.mkdir(os.path.join(self.working_dir, "output_csv"))

    def map(self, data, simulation):
        concatenated_df = pd.concat(list(data.values()), axis=0, ignore_index=True,␣
↪sort=True)
        return concatenated_df

    def reduce(self, all_data):

        results = pd.concat(list(all_data.values()), axis=0,
                            keys=[k.id for k in all_data.keys()],
                            names=['SimId'])
        results.index = results.index.droplevel(1)

        results.to_csv(os.path.join("output_csv", self.__class__.__name__ + '.csv'))


if __name__ == '__main__':

    platform = Platform('COMPS')
    filenames = ['output/c.csv']
    analyzers = [CSVAnalyzer(filenames=filenames)]
    experiment_id = '9311af40-1337-ea11-a2be-f0921c167861'
    manager = AnalyzeManager(configuration={}, partial_analyze_ok=True,␣
↪platform=platform,
                             ids=[(experiment_id, ItemType.EXPERIMENT)],
                             analyzers=analyzers)
    manager.analyze()
```

You can quickly see this analyzer in use by running the included `example_analysis_CSVAnalyzer` example class.

1.7.4 Using analyzers with SSMT

If you have access to COMPS, you can use idmtools to run analyzers on Server-Side Modeling Tools (SSMT). SSMT is integrated with COMPS, allowing you to leverage the HPC compute power for running both the analyzers and any pre or post processing scripts that you may have previously ran locally.

The `idmtools.analysis.platform_analysis.PlatformAnalysis` class is used for sending the needed information (such as analyzers, files, and experiment ids) as a SSMT work item to be run with SSMT and COMPS.

The following example, run_ssmt_analysis.py, shows how to use `idmtools.analysis.platform_analysis.PlatformAnalysis` for running analysis on SSMT:

```
from examples.ssmt.simple_analysis.analyzers.AdultVectorsAnalyzer import␣
↪AdultVectorsAnalyzer
from examples.ssmt.simple_analysis.analyzers.PopulationAnalyzer import␣
↪PopulationAnalyzer
from idmtools.core.platform_factory import Platform
from idmtools.analysis.platform_anaylsis import PlatformAnalysis

if __name__ == "__main__":
    platform = Platform('COMPS2')
    analysis = PlatformAnalysis(platform=platform,
                                experiment_ids=["8bb8ae8f-793c-ea11-a2be-
↪f0921c167861"],
                                analyzers=[PopulationAnalyzer,␣
↪AdultVectorsAnalyzer],
                                analyzers_args=[{'title': 'idm'}, {'name':
↪'global good'}],
                                analysis_name="SSMT Analysis Simple 1")

    analysis.analyze(check_status=True)
    wi = analysis.get_work_item()
    print(wi)
```

In this example two analyzers are run on an existing experiment with the output results saved to an output directory. After you run the example you can see the results by using the returned SSMTWorkItem id and searching for it under **Work Items** in COMPS.

Note: To access and use COMPS you must receive approval and credentials from IDM. Send your request to support@idmod.org.

1.8 Plot data

You can use idmtools to plot the output results of the analysis of simulations and experiments. You must include a plotting library within your script. For example, with Python a common plotting library is matplotlib (https://matplotlib.org/).

The following shows how to add matplotlib to a reduce method for plotting the output results of a population analyzer:

```
def reduce(self, all_data: dict) -> Any:
    output_dir = os.path.join(self.working_dir, "output")

    with open(os.path.join(output_dir, "population.json"), "w") as fp:
```

(continues on next page)

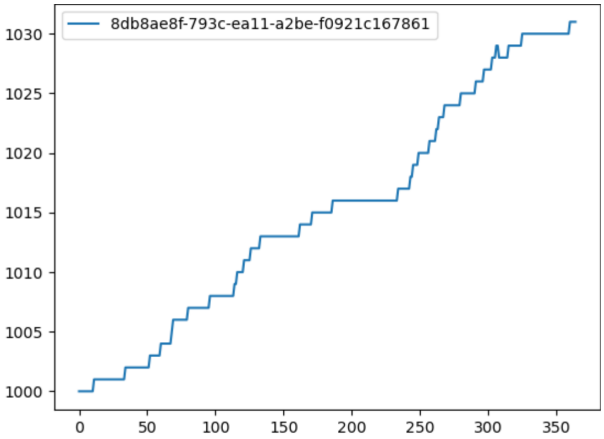
(continued from previous page)

```
        json.dump({str(s.uid): v for s, v in all_data.items()}, fp)

    import matplotlib.pyplot as plt

    fig = plt.figure()
    ax = fig.add_subplot()

    for pop in list(all_data.values()):
        ax.plot(pop)
    ax.legend([str(s.uid) for s in all_data.keys()])
    fig.savefig(os.path.join(output_dir, "population.png"))
```

The reduce method uses the output from the map method, which is **InsetChart.json**, as the input for plotting the results of the **Statistical Population** channel:

```
filenames = ['output/InsetChart.json']

def map(self, data: Any, item: IItem) -> Any:
    return data[self.filenames[0]]["Channels"]["Statistical Population"]["Data"]
```

The final results are plotted and saved to the file, population.png:



1.9 Architecture and packages reference

idmtools is built in Python and includes an architecture designed for ease of use, flexibility, and extensibility. You can quickly get up and running and see the capabilities of idmtools by using one of the many included example Python scripts demonstrating the functionality of the packages.

idmtools is built in a modular fashion, as seen in the diagrams below. idmtools design includes multiple packages and APIs, providing both the flexibility to only include the necessary packages for your modeling needs and the extensibility by using the APIs for any needed customization.

1.9.1 Packages overview

1.9.2 Packages and APIs

The following diagrams help illustrate the primary packages and associated APIs available for modeling and development with idmtools:

Core and job orchestration

Local platform

COMPS platform

Note: To access and use COMPS you must receive approval and credentials from IDM. Send your request to support@idmod.org.

Models

API class specifications

EMOD

EMOD support with idmtools is provided with the **emodpy** package, which leverages idmtools plugin architecture:

idmtools

idmtools package

Subpackages

idmtools.analysis package

Submodules

idmtools.analysis.add_analyzer module

class idmtools.analysis.add_analyzer.**AddAnalyzer**(*filenames=None*, *output_path='output'*)

Bases: *idmtools.entities.ianalyzer.IAnalyzer*

A simple base class to add analyzers.

Examples

```
# Example AddAnalyzer for EMOD Experiment
# In this example, we will demonstrate how to create an AddAnalyzer to analyze an␣
↪experiment's output file

# First, import some necessary system and idmtools packages.
from idmtools.analysis.analyze_manager import AnalyzeManager
from idmtools.analysis.add_analyzer import AddAnalyzer
from idmtools.core import ItemType
from idmtools.core.platform_factory import Platform

if __name__ == '__main__':

    # Set the platform where you want to run your analysis
    # In this case we are running in COMPS, but this can be changed to run 'Local'
    platform = Platform('COMPS2')

    # Arg option for analyzer init are uid, working_dir, data in the method map␣
↪(aka select_simulation_data),
    # and filenames
    # In this case, we want to provide a filename to analyze
    filenames = ['StdOut.txt']
    # Initialize the analyser class with the name of file to save to and start␣
↪the analysis
    analyzers = [AddAnalyzer(filenames=filenames)]

    # Set the experiment you want to analyze
    experiment_id = 'f227704e-0c34-ea11-a2be-f0921c167861'  # comps2 staging exp␣
↪id

    # Specify the id Type, in this case an Experiment
    manager = AnalyzeManager(platform=platform, ids=[(experiment_id, ItemType.
↪EXPERIMENT)], analyzers=analyzers)
    manager.analyze()
```

filter(*item*)
Decide whether the analyzer should process a simulation.

Parameters **item** – An IItem to be considered for processing with this analyzer.

Returns A Boolean indicating whether simulation should be analyzed by this analyzer.

initialize()
Call once after the analyzer has been added to the AnalyzeManager.

Add everything depending on the working directory or unique ID here instead of in __init__.

map(*data*, *item*)
In parallel for each simulation, consume raw data from filenames and emit selected data.

Parameters

- **data** – A dictionary associating filename with content for simulation data.
- **item** – IItem object that the passed data is associated with.

Returns Selected data for the given item.

reduce(*data*)
Combine the *map()* data for a set of items into an aggregate result.

Parameters **all_data** – A dictionary with entries for the item ID and selected data.

idmtools.analysis.analyze_manager module

idmtools.analysis.analyze_manager.**pool_worker_initializer**(*func*, *analyzers*, *cache*, *platform: IPlatform*) → NoReturn
Initialize the pool worker, which allows the process pool to associate the analyzers, cache, and path mapping to the function executed to retrieve data. Using an initializer improves performance.

Parameters

- **func** – The function that the pool will call.
- **analyzers** – The list of all analyzers to run.
- **cache** – The cache object.
- **platform** – The platform to communicate with to retrieve files from.

Returns None

class idmtools.analysis.analyze_manager.**AnalyzeManager**(*platform: IPlatform = None*, *configuration: dict = None*, *ids: List[Tuple[Union[str, uuid.UUID], idmtools.core.enums.ItemType]] = None*, *analyzers: List[idmtools.entities.ianalyzer.IAnalyzer] = None*, *working_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/idmtools-for-disease-modeling-idmtools/checkouts/v1.4.0/docs'*, *partial_analyze_ok: bool = False*, *max_items: Optional[int] = None*, *verbose: bool = True*, *force_manager_working_directory: bool = False*, *exclude_ids: List[Union[str, uuid.UUID]] = None*, *analyze_failed_items: bool = False*)

Bases: *idmtools.core.cache_enabled.CacheEnabled*

ANALYZE_TIMEOUT = 28800

WAIT_TIME = 1.15

EXCEPTION_KEY = '__EXCEPTION__'

exception TimeOutException
Bases: Exception

exception ItemsNotReady
Bases: Exception

add_item(*item:* idmtools.core.interfaces.ientity.IEntity) → NoReturn
Add an additional item for analysis.

Parameters **item** – The new item to add for analysis.

Returns None

add_analyzer(*analyzer:* idmtools.entities.ianalyzer.IAnalyzer) → NoReturn
Add another analyzer to use on the items to be analyzed.

Parameters **analyzer** – An analyzer object (*IAnalyzer*).

Returns:

analyze() → bool
Process the provided items with the provided analyzers. This is the main driver method of *AnalyzeManager*.

Returns True on success; False on failure/exception.

idmtools.analysis.csv_analyzer module

class idmtools.analysis.csv_analyzer.**CSVAnalyzer**(*filenames*, *parse=True*)
Bases: *idmtools.entities.ianalyzer.IAnalyzer*

Provides an analyzer for CSV output

Examples

Simple Example This example covers the basic usage of the CSVAnalyzer

```
# Example CSVAnalyzer for any experiment
# In this example, we will demonstrate how to use a CSVAnalyzer to analyze␣
↪csv files for experiments

# First, import some necessary system and idmtools packages.
from logging import getLogger

from idmtools.analysis.analyze_manager import AnalyzeManager
from idmtools.analysis.csv_analyzer import CSVAnalyzer
from idmtools.core import ItemType
from idmtools.core.platform_factory import Platform


if __name__ == '__main__':

    # Set the platform where you want to run your analysis
    # In this case we are running in COMPS since the Work Item we are␣
↪analyzing was run on COMPS
    logger = getLogger()
    with Platform('COMPS2') as platform:

        # Arg option for analyzer init are uid, working_dir, data in the␣
↪method map (aka select_simulation_data),
        # and filenames
        # In this case, we want to provide a filename to analyze
        filenames = ['output/c.csv']
        # Initialize the analyser class with the path of the output csv file
        analyzers = [CSVAnalyzer(filenames=filenames)]

        # Set the experiment id you want to analyze
        experiment_id = '9311af40-1337-ea11-a2be-f0921c167861'  # staging exp␣
↪id simple sim and csv example

        # Specify the id Type, in this case an Experiment on COMPS
        manager = AnalyzeManager(configuration={}, partial_analyze_ok=True,␣
↪ids=[(experiment_id, ItemType.EXPERIMENT)],
                                 analyzers=analyzers)
        manager.analyze()
```

Multiple CSVs This example covers analyzing multiple CSVs

```
# Example CSVAnalyzer for any experiment with multiple csv outputs
# In this example, we will demonstrate how to use a CSVAnalyzer to analyze␣
↪csv files for experiments

```

(continues on next page)

(continued from previous page)

```
# First, import some necessary system and idmtools packages.
from idmtools.analysis.analyze_manager import AnalyzeManager
from idmtools.analysis.csv_analyzer import CSVAnalyzer
from idmtools.core import ItemType
from idmtools.core.platform_factory import Platform


if __name__ == '__main__':

    # Set the platform where you want to run your analysis
    # In this case we are running in COMPS since the Work Item we are␣
↪analyzing was run on COMPS
    platform = Platform('COMPS2')

    # Arg option for analyzer init are uid, working_dir, data in the method␣
↪map (aka select_simulation_data),
    # and filenames
    # In this case, we have multiple csv files to analyze
    filenames = ['output/a.csv', 'output/b.csv']
    # Initialize the analyser class with the path of the output csv file
    analyzers = [CSVAnalyzer(filenames=filenames)]

    # Set the experiment id you want to analyze
    experiment_id = '1bddce22-0c37-ea11-a2be-f0921c167861'  # staging exp id␣
↪with multiple csv file outputs

    # Specify the id Type, in this case an Experiment on COMPS
    manager = AnalyzeManager(configuration={}, partial_analyze_ok=True,␣
↪platform=platform,
                             ids=[(experiment_id, ItemType.EXPERIMENT)],
                             analyzers=analyzers)
    manager.analyze()
```

initialize()
Call once after the analyzer has been added to the `AnalyzeManager`.

Add everything depending on the working directory or unique ID here instead of in __init__.

map(*data*, *simulation*)
In parallel for each simulation, consume raw data from filenames and emit selected data.

Parameters

- **data** – A dictionary associating filename with content for simulation data.
- **item** – `IItem` object that the passed data is associated with.

Returns Selected data for the given item.

reduce(*all_data*)
Combine the *map()* data for a set of items into an aggregate result.

Parameters **all_data** – A dictionary with entries for the item ID and selected data.

idmtools.analysis.download_analyzer module

class idmtools.analysis.download_analyzer.**DownloadAnalyzer**(*filenames=None*, *output_path=None*, ***kwargs*)

Bases: *idmtools.entities.ianalyzer.IAnalyzer*

A simple base class that will download the files specified in filenames without further treatment.

Can be used by creating a child class:

```
class InsetDownloader(DownloadAnalyzer):
    filenames = ['output/InsetChart.json']
```

Or by directly calling it:

```
analyzer = DownloadAnalyzer(filenames=['output/InsetChart.json'])
```

Examples

```
# Example DownloadAnalyzer for EMOD Experiment
# In this example, we will demonstrate how to create an DownloadAnalyzer to␣
↪download simulation output files locally

# First, import some necessary system and idmtools packages.
from idmtools.analysis.analyze_manager import AnalyzeManager
from idmtools.analysis.download_analyzer import DownloadAnalyzer
from idmtools.core import ItemType
from idmtools.core.platform_factory import Platform

if __name__ == '__main__':

    # Set the platform where you want to run your analysis
    # In this case we are running in COMPS, but this can be changed to run 'Local'
    platform = Platform('COMPS2')

    # Arg option for analyzer init are uid, working_dir, data in the method map␣
↪(aka select_simulation_data),
    # and filenames
    # In this case, we want to provide a filename to analyze
    filenames = ['StdOut.txt']
    # Initialize the analyser class with the path of the output files to download
    analyzers = [DownloadAnalyzer(filenames=filenames, output_path='download')]

    # Set the experiment you want to analyze
    experiment_id = '11052582-83da-e911-a2be-f0921c167861'  # comps2 staging exp␣
↪id

    # Specify the id Type, in this case an Experiment
    manager = AnalyzeManager(configuration={}, platform=platform,␣
↪ids=[(experiment_id, ItemType.EXPERIMENT)],
                             analyzers=analyzers)
    manager.analyze()
```

reduce(*all_data: dict*)

Combine the *map()* data for a set of items into an aggregate result.

Parameters **all_data** – A dictionary with entries for the item ID and selected data.

initialize()
Call once after the analyzer has been added to the AnalyzeManager.

Add everything depending on the working directory or unique ID here instead of in __init__.

get_sim_folder(*item*)
Concatenate the specified top-level output folder with the simulation ID.

Parameters **item** – A simulation output parsing thread.

Returns The name of the folder to download this simulation's output to.

map(*data: Dict[str, Any]*, *item: Union[*idmtools.entities.iworkflow_item.IWorkflowItem, idmtools.entities.simulation.Simulation*]*)
Write the downloaded data to the path

Parameters

- **data** –
- **item** –

Returns:

idmtools.analysis.map_worker_entry module

idmtools.analysis.map_worker_entry.**map_item**(*item:* idmtools.core.interfaces.iitem.IItem) → NoReturn
Initialize some worker-global values; a worker process entry point for analyzer item-mapping.

Parameters **item** – The item (often simulation) to process.

Returns None

idmtools.analysis.platform_analysis_bootstrap module

This script is executed as entrypoint in the docker SSMT worker. Its role is to collect the experiment ids and analyzers and run the analysis.

idmtools.analysis.platform_anaylsis module

class idmtools.analysis.platform_anaylsis.**PlatformAnalysis**(*platform*, *experiment_ids*, *analyzers*, *analyzers_args=None*, *analysis_name='WorkItem Test'*, *tags=None*, *additional_files=None*, *asset_collection_id=None*, *asset_files=<idmtools.assets.file_list.FileList object>*, *wait_till_done: bool = True*)
Bases: object

analyze(*check_status=True*)

validate_args()

get_work_item()

idmtools.analysis.tags_analyzer module

class idmtools.analysis.tags_analyzer.**TagsAnalyzer**(*uid=None*, *working_dir=None*, *parse=True*)

Bases: *idmtools.entities.ianalyzer.IAnalyzer*

Provides an analyzer for CSV output

Examples

```
# Example TagsAnalyzer for any experiment
# In this example, we will demonstrate how to use a TagsAnalyzer to put your sim␣
↪tags in a csv file

# First, import some necessary system and idmtools packages.
from idmtools.analysis.analyze_manager import AnalyzeManager
from idmtools.analysis.tags_analyzer import TagsAnalyzer
from idmtools.core import ItemType
from idmtools.core.platform_factory import Platform

if __name__ == '__main__':

    # Set the platform where you want to run your analysis
    # In this case we are running in COMPS since the Work Item we are analyzing␣
↪was run on COMPS
    platform = Platform('COMPS2')

    # Arg option for analyzer init are uid, working_dir, data in the method map␣
↪(aka select_simulation_data),
    # and filenames
    # Initialize the analyser class which just requires an experiment id
    analyzers = [TagsAnalyzer()]

    # Set the experiment id you want to analyze
    experiment_id = '36d8bfdc-83f6-e911-a2be-f0921c167861'  # staging exp id␣
↪JSuresh's Magude exp

    # Specify the id Type, in this case an Experiment on COMPS
    manager = AnalyzeManager(configuration={}, partial_analyze_ok=True,␣
↪platform=platform,
                             ids=[(experiment_id, ItemType.EXPERIMENT)],
                             analyzers=analyzers)
    manager.analyze()
```

initialize()

Call once after the analyzer has been added to the `AnalyzeManager`.

Add everything depending on the working directory or unique ID here instead of in __init__.

map(*data*, *simulation*)

In parallel for each simulation, consume raw data from filenames and emit selected data.

Parameters

- **data** – A dictionary associating filename with content for simulation data.
- **item** – IItem object that the passed data is associated with.

Returns Selected data for the given item.

reduce(*all_data*)

Combine the *map()* data for a set of items into an aggregate result.

Parameters **all_data** – A dictionary with entries for the item ID and selected data.

Module contents

idmtools.assets package

Submodules

idmtools.assets.asset module

class idmtools.assets.asset.**Asset**(*absolute_path: Optional[str] = None*, *relative_path: Optional[str] = <property object>*, *filename: Optional[str] = None*, *content: dataclasses.InitVar = <property object>*, *_length: Optional[int] = None*, *persisted: bool = False*, *handler: Callable = <class 'str'>*, *download_generator_hook: Callable = None*, *checksum: Optional[str] = <property object>*)

Bases: object

A class representing an asset. An asset can either be related to a physical asset present on the computer or directly specified by a filename and content.

Parameters

- **absolute_path** – The absolute path of the asset. Optional if **filename** and **content** are given.
- **relative_path** – The relative path (compared to the simulation root folder).
- **filename** – Name of the file. Optional if **absolute_path** is given.
- **content** – The content of the file. Optional if **absolute_path** is given.
- **checksum** – Optional. Useful in systems that allow single upload based on checksums and retrieving from those systems

  Note: we add this to allow systems who provide asset caching by MD5 opportunity to avoid re-uploading assets

absolute_path: Optional[str] = None

filename: Optional[str] = None

persisted: bool = False

handler

alias of builtins.str

download_generator_hook: Callable = None

property checksum

Returns None.

property extension

property relative_path

property bytes

property length

property content

Returns The content of the file, either from the content attribute or by opening the absolute path.

download_generator() → Generator[bytearray, None, None]
A Download Generator that returns chunks of bytes from the file

Returns Generator of bytearray

download_stream() → _io.BytesIO
Get a bytes IO stream of the asset

Returns BytesIO of the Asset

download_to_path(*dest: str*, *force: bool = False*)
Download an asset to path. This requires loadings the object through the platofrm

Parameters **path** – Path to write to. If it is a directory, the asset filename will be added to it

Returns None

idmtools.assets.asset_collection module

class idmtools.assets.asset_collection.**AssetCollection**(*assets: Union[List[TAsset], AssetCollection] = None*, *tags=None*)
Bases: *idmtools.core.interfaces.ientity.IEntity*

A class that represents a collection of assets.

Parameters **assets** – An optional list of assets to create the collection with.

item_type: *idmtools.core.enums.ItemType* = 5

assets: List[*idmtools.assets.asset.Asset*] = None

classmethod from_id(*item_id: Union[str, uuid.UUID]*, *platform: IPlatform = None*, *as_copy: bool = False*, ***kwargs*) → AssetCollection
Loads a AssetCollection from id

Parameters

- **item_id** – Asset Collection ID
- **platform** – Platform Ojbect
- **as_copy** – Should you load the object as a copy. When True, the contents of AC are copied, but not the id. Useful when editing ACs
- ****kwargs** –

Returns AssetCollection

classmethod from_directory(*assets_directory: str, recursive: bool = True, flatten: bool = False, filters: Optional[List[Union[Callable[[TAsset], bool], Callable]]] = None, filters_mode: idmtools.core.enums.FilterMode = <FilterMode.OR: 1>, relative_path: str = None*) → TAssetCollection

Fill up an `AssetCollection` from the specified directory. See `assets_from_directory()` for arguments.

Returns A created `AssetCollection` object.

static assets_from_directory(*assets_directory: str, recursive: bool = True, flatten: bool = False, filters: Optional[List[Union[Callable[[TAsset], bool], Callable]]] = None, filters_mode: idmtools.core.enums.FilterMode = <FilterMode.OR: 1>, forced_relative_path: str = None*) → List[*idmtools.assets.asset.Asset*]

Create assets for files in a given directory.

Parameters

- **assets_directory** – The root directory of the assets.
- **recursive** – True to recursively traverse the subdirectory.
- **flatten** – Put all the files in root regardless of whether they were in a subdirectory or not.
- **filters** – A list of filters to apply to the assets. The filters are functions taking an `Asset` as argument and returning true or false. True adds the asset to the collection; False filters it out. See `asset_filters()`.
- **filters_mode** – When given multiple filters, either OR or AND the results.
- **forced_relative_path** – Prefix a relative path to the path created from the root directory.

Examples

For **relative_path**, given the following folder structure root/a/1,txt root/b.txt and relative_path="test". Will return assets with relative path: test/a/1,txt and test/b.txt

Given the previous example, if flatten is also set to True, the following relative_path will be set: /1.txt and /b.txt

Returns A list of assets.

copy() → *idmtools.assets.asset_collection.AssetCollection*

Copy our Asset Collection, removing ID and tags

Returns New AssetCollection containing Assets from other AssetCollection

add_directory(*assets_directory: str, recursive: bool = True, flatten: bool = False, filters: Optional[List[Union[Callable[[TAsset], bool], Callable]]] = None, filters_mode: idmtools.core.enums.FilterMode = <FilterMode.OR: 1>, relative_path: str = None*)

Retrieve assets from the specified directory and add them to the collection. See `assets_from_directory()` for arguments.

is_editable(*error=False*) → bool

Checks whether Item is editable

Parameters **error** – Throw error is not

Returns True if editable, False otherwise.

add_asset(*asset: Union[*idmtools.assets.asset.Asset*, str]*, *fail_on_duplicate: bool = True*, ***kwargs*)
Add an asset to the collection.

Parameters

- **asset** – A string or an *Asset* object to add. If a string, the string will be used as the absolute_path and any kwargs will be passed to the Asset constructor
- **fail_on_duplicate** – Raise a **DuplicateAssetError** if an asset is duplicated. If not, simply replace it.
- ****kwargs** – Arguments to pass to Asset constructor when asset is a string

add_assets(*assets: Union[List[TAsset], AssetCollection]*, *fail_on_duplicate: bool = True*)
Add assets to a collection

Parameters

- **assets** – An list of assets as either list or a collection
- **fail_on_duplicate** – Raise a **DuplicateAssetError** if an asset is duplicated. If not, simply replace it.

Returns:

add_or_replace_asset(*asset:* idmtools.assets.asset.Asset)
Add or replaces an asset in a collection

Parameters **asset** – Asset to add or replace

Returns None.

get_one(***kwargs*)
Get an asset out of the collection based on the filers passed.

Examples:

```
>>> a = AssetCollection()
>>> a.get_one(filename="filename.txt")
```

Parameters ****kwargs** – keyword argument representing the filters.

Returns None or Asset if found.

delete(***kwargs*) → NoReturn
Delete an asset based on keywords attributes

Parameters ****kwargs** – Filter for the asset to delete.

remove(***kwargs*) → NoReturn
Remove an asset from the AssetCollection based on keywords attributes

Parameters ****kwargs** – Filter for the asset to remove.

pop(***kwargs*) → *idmtools.assets.asset.Asset*
Get and delete an asset based on keywords.

Parameters ****kwargs** – Filter for the asset to pop.

extend(*assets: List[*idmtools.assets.asset.Asset*]*, *fail_on_duplicate: bool = True*) → NoReturn
Extend the collection with new assets :param assets: Which assets to add :param fail_on_duplicate: Fail if duplicated asset is included.

clear()

set_all_persisted()

property count

property uid

has_asset(*absolute_path: str = None*, *filename: str = None*) → bool
Search for asset by absolute_path or by filename

Parameters

- **absolute_path** – Absolute path of source file
- **filename** – Destination filename

Returns True if asset exists, False otherwise

find_index_of_asset(*absolute_path: str = None*, *filename: str = None*) → Optional[int]
Finds the index of asset by path or filename

Parameters

- **absolute_path** – Path to search
- **filename** – Filename to search

Returns Index number if found. None if not found.

pre_creation() → None
Called before the actual creation of the entity.

post_creation() → None
Called after the actual creation of the entity.

set_tags(*tags: Dict[str, Any]*)

add_tags(*tags: Dict[str, Any]*)

idmtools.assets.content_handlers module

idmtools.assets.content_handlers.**json_handler**(*content*)

idmtools.assets.errors module

exception idmtools.assets.errors.**DuplicatedAssetError**(*asset: TAsset*)
Bases: Exception

idmtools.assets.file_list module

class idmtools.assets.file_list.**FileList**(*root=None*, *files_in_root=None*, *recursive=False*, *ignore_missing=False*, *relative_path=None*, *max_depth=3*)
Bases: object

Special utility class to help handling user files

add_asset_file(*af*)
method used to add asset file :param af: asset file to add

Returns: None

add_file(*path*, *relative_path=''*)
method used to add a file :param path: file oath :param relative_path: file relative path

Returns: None

add_path(*path*, *files_in_dir=None*, *relative_path=None*, *recursive=False*)
Add a path to the file list. :param path: The path to add (needs to be a dictionary) :param files_in_dir: If we want to only retrieve certain files in this path :param relative_path: relative_path: The relative path prefixed to each added files :param recursive: Do we want to browse recursively

Returns: None

Module contents

idmtools.builders package

Submodules

idmtools.builders.arm_simulation_builder module

class idmtools.builders.arm_simulation_builder.**ArmType**(*value*)
Bases: `enum.Enum`

An enumeration.

cross = 0

pair = 1

class idmtools.builders.arm_simulation_builder.**SweepArm**(*type=<ArmType.cross: 0>*, *funcs=[]*)
Bases: `object`

Class that represents a parameter arm.

add_sweep_definition(*func: Callable*, *values: Iterable[Any]*)

get_max_values_count()

adjust_values_length()

class idmtools.builders.arm_simulation_builder.**ArmSimulationBuilder**
Bases: *idmtools.builders.simulation_builder.SimulationBuilder*

Class that represents an experiment builder.

This particular sweep builder build sweeps in "ARMS". This is particular useful in situations where you want to sweep parameters that have branches of parameters. For Example, let's say we have a model with the following parameters: * population * susceptible * recovered * enable_births * birth_rate

Enable births controls an optional feature that is controlled by the birth_rate parameter. If we want to sweep a set of parameters on population, susceptible with enabled_births set to off but also want to sweep the birth_rate we could do that like so

```
###################################
# This example provides how you can check if your sweeps are working as expected
###################################
from functools import partial
from idmtools.builders import ArmSimulationBuilder, SweepArm, ArmType
from idmtools.entities.command_task import CommandTask
from idmtools.entities.templated_simulation import TemplatedSimulations
from tabulate import tabulate


def update_parameter(simulation, parameter, value):
    simulation.task.config[parameter] = value


base_task = CommandTask('example')
base_task.config = dict(enable_births=False)
builder = ArmSimulationBuilder()
# Define our first set of sweeps
arm = SweepArm(type=ArmType.cross)
arm.add_sweep_definition(partial(update_parameter, parameter='population'), [500,␣
↪1000])
arm.add_sweep_definition(partial(update_parameter, parameter='susceptible'), [0.5,
↪ 0.9])
builder.add_arm(arm)
# Now add the sweeps with the birth_rate as well
arm.add_sweep_definition(partial(update_parameter, parameter='enable_births'),␣
↪[True])
arm.add_sweep_definition(partial(update_parameter, parameter='birth_rate'), [0.01,
↪ 0.1])
builder.add_arm(arm)

sims = TemplatedSimulations(base_task=base_task)
sims.add_builder(builder)

print(tabulate([s.task.config for s in list(sims)], headers="keys"))
```

This would result in the output

Table 1: Arm Example Values

enable_births	population	susceptible	birth_rate
False	500	0.5	
False	500	0.9	
False	1000	0.5	
False	1000	0.9	
True	500	0.5	0.01
True	500	0.5	0.1
True	500	0.9	0.01
True	500	0.9	0.1
True	1000	0.5	0.01
True	1000	0.5	0.1
True	1000	0.9	0.01
True	1000	0.9	0.1

Examples

```
"""
        This file demonstrates how to use ArmExperimentBuilder in PythonExperiment
↪'s builder.
        We are then adding the builder to PythonExperiment.

        |__sweep arm1
            |_ a = 1
            |_ b = [2,3]
            |_ c = [4,5]
        |__ sweep arm2
            |_ a = [6,7]
            |_ b = 2
        Expect sims with parameters:
            sim1: {a:1, b:2, c:4}
            sim2: {a:1, b:2, c:5}
            sim3: {a:1, b:3, c:4}
            sim4: {a:1, b:3, c:5}
            sim5: {a:6, b:2}
            sim6: {a:7, b:2}
        Note:
            arm1 and arm2 are adding to total simulations
"""
import os
import sys
from functools import partial

from idmtools.builders import SweepArm, ArmType, ArmSimulationBuilder
from idmtools.core.platform_factory import platform
from idmtools.entities.experiment import Experiment
from idmtools.entities.templated_simulation import TemplatedSimulations
from idmtools_models.python.json_python_task import JSONConfiguredPythonTask
from idmtools_test import COMMON_INPUT_PATH

# define specific callbacks for a, b, and c
setA = partial(JSONConfiguredPythonTask.set_parameter_sweep_callback, param="a")
setB = partial(JSONConfiguredPythonTask.set_parameter_sweep_callback, param="b")
setC = partial(JSONConfiguredPythonTask.set_parameter_sweep_callback, param="c")


if __name__ == "__main__":
    with platform('COMPS2'):
        base_task = JSONConfiguredPythonTask(script_path=os.path.join(COMMON_
↪INPUT_PATH, "python", "model1.py"))
        # define that we are going to create multiple simulations from this task
        ts = TemplatedSimulations(base_task=base_task)

        # define our first sweep Sweep Arm
        arm1 = SweepArm(type=ArmType.cross)
        builder = ArmSimulationBuilder()
        arm1.add_sweep_definition(setA, 1)
        arm1.add_sweep_definition(setB, [2, 3])
        arm1.add_sweep_definition(setC, [4, 5])
        builder.add_arm(arm1)

        # adding more simulations with sweeping
```

(continues on next page)

(continued from previous page)

```
        arm2 = SweepArm(type=ArmType.cross)
        arm2.add_sweep_definition(setA, [6, 7])
        arm2.add_sweep_definition(setB, [2])
        builder.add_arm(arm2)

        # add our builders to our template
        ts.add_builder(builder)

        # create experiment from the template
        experiment = Experiment.from_template(ts, name=os.path.split(sys.
↪argv[0])[1],
                                              tags={"string_tag": "test", "number_
↪tag": 123, "KeyOnly": None})
        # run the experiment
        experiment.run()
        # in most real scenarios, you probably do not want to wait as this will␣
↪wait until all simulations
        # associated with an experiment are done. We do it in our examples to␣
↪show feature and to enable
        # testing of the scripts
        experiment.wait()
        # use system status as the exit code
        sys.exit(0 if experiment.succeeded else -1)
```

add_arm(*arm*)

idmtools.builders.csv_simulation_builder module

class idmtools.builders.csv_simulation_builder.**CsvExperimentBuilder**

Bases: *idmtools.builders.simulation_builder.SimulationBuilder*

Class that represents an experiment builder.

Examples

```
"""
        This file demonstrates how to use CsvExperimentBuilder in PythonExperiment
↪'s builder.
        then adding the builder to PythonExperiment.

        We first load a csv file from local dir which contains parameters/values␣
↪to sweep
        then sweep parameters based in csv file with CsvExperimentBuilder
        the csv file basically already lists all possible combinations of␣
↪parameters you wan to sweep

        Paramaters names(header) and values in csv file
            a,b,c,d
            1,2,3,
            1,3,1,
            2,2,3,4
            2,2,2,5
            2,,3,6
        Expect sims with parameters:
```

(continues on next page)

(continued from previous page)

```
            sim1: {a:1, b:2, c:3}
            sim2: {a:1, b:3, c:1}
            sim3: {a:2, b:2, c:3, d:4}
            sim4: {a:2, b:2, c:2, d:5}
            sim5: {a:2, c:3, d:6}  <-- no 'b'

        This builder can be used to test or simple scenarios.
        for example, you may only want to test list of parameter combinations,␣
↪and do not care about anything else,
        you can list them in csv file so you do not have to go through␣
↪traditional sweep method(i.e ExperimentBuilder's)

"""

import os
import sys
from functools import partial

import numpy as np

from idmtools.builders import CsvExperimentBuilder
from idmtools.core.platform_factory import platform
from idmtools.entities.experiment import Experiment
from idmtools.entities.templated_simulation import TemplatedSimulations
from idmtools_models.python.json_python_task import JSONConfiguredPythonTask
from idmtools_test import COMMON_INPUT_PATH

# define function partials to be used during sweeps
setA = partial(JSONConfiguredPythonTask.set_parameter_sweep_callback, param="a")
setB = partial(JSONConfiguredPythonTask.set_parameter_sweep_callback, param="b")
setC = partial(JSONConfiguredPythonTask.set_parameter_sweep_callback, param="c")
setD = partial(JSONConfiguredPythonTask.set_parameter_sweep_callback, param="d")

if __name__ == "__main__":
    # define what platform we want to use. Here we use a context manager but if␣
↪you prefer you can
    # use objects such as Platform('COMPS2') instead
    with platform('COMPS2'):
        # define our base task
        base_task = JSONConfiguredPythonTask(script_path=os.path.join(COMMON_
↪INPUT_PATH, "python", "model1.py"),
                                             parameters=dict(c='c-value'))
        # define our input csv sweep
        base_path = os.path.abspath(os.path.join(COMMON_INPUT_PATH, "builder"))
        file_path = os.path.join(base_path, 'sweeps.csv')
        builder = CsvExperimentBuilder()
        func_map = {'a': setA, 'b': setB, 'c': setC, 'd': setD}
        type_map = {'a': np.int, 'b': np.int, 'c': np.int, 'd': np.int}
        builder.add_sweeps_from_file(file_path, func_map, type_map)

        # now define we want to create a series of simulations using the base␣
↪task and the sweep
        ts = TemplatedSimulations.from_task(base_task)
        # optionally we could update the base simulation metdata here
        # ts.base_simulations.tags['example'] 'yes'
        ts.add_builder(builder)
```

(continues on next page)

(continued from previous page)

```
        # define our experiment with its metadata
        experiment = Experiment.from_template(ts,
                                              name=os.path.split(sys.argv[0])[1],
                                              tags={"string_tag": "test", "number_
↪tag": 123}
                                              )

        # run the experiment and wait. By default run does not wait
        # in most real scenarios, you probably do not want to wait as this will␣
↪wait until all simulations
        # associated with an experiment are done. We do it in our examples to␣
↪show feature and to enable
        # testing of the scripts
        experiment.run(wait_until_done=True)
        # use system status as the exit code
        sys.exit(0 if experiment.succeeded else -1)
```

add_sweeps_from_file (*file_path*, *func_map=None*, *type_map=None*, *sep=','*)

idmtools.builders.simulation_builder module

class idmtools.builders.simulation_builder.**SimulationBuilder**

Bases: object

Class that represents an experiment builder.

Examples

```
import os
import sys

from idmtools.assets import AssetCollection
from idmtools.builders import SimulationBuilder
from idmtools.core.platform_factory import platform
from idmtools.entities.experiment import Experiment
from idmtools.entities.templated_simulation import TemplatedSimulations
from idmtools_models.python.json_python_task import JSONConfiguredPythonTask
from idmtools_test import COMMON_INPUT_PATH

with platform('COMPS2'):
    base_task = JSONConfiguredPythonTask(
        script_path=os.path.join(COMMON_INPUT_PATH, "compsplatform", "working_
↪model.py"),
        # add common assets from existing collection
        common_assets=AssetCollection.from_id('bd80dd0c-1b31-ea11-a2be-
↪f0921c167861', as_copy=True)
    )

    ts = TemplatedSimulations(base_task=base_task)
    # sweep parameter
    builder = SimulationBuilder()
    builder.add_sweep_definition(JSONConfiguredPythonTask.set_parameter_partial(
↪"min_x"), range(-2, 0))
    builder.add_sweep_definition(JSONConfiguredPythonTask.set_parameter_partial(
↪"max_x"), range(1, 3))
```

(continues on next page)

(continued from previous page)

```
    ts.add_builder(builder)

    e = Experiment.from_template(ts, name=os.path.split(sys.argv[0])[1])
    e.run(wait_until_done=True)
    # use system status as the exit code
    sys.exit(0 if e.succeeded else -1)
```

Add tags with builder callbacks:

```
def update_sim(sim, parameter, value):
    sim.task.set_parameter(parameter, value)
    # set sim tasks,
    return {'custom': 123, parameter:value)

builder = SimulationBuilder()
set_run_number = partial(update_sim, param="Run_Number")
builder.add_sweep_definition(set_run_number, range(0, 2))
# create experiment from builder
exp = Experiment.from_builder(builder, task, name=expname)
```

SIMULATION_ATTR = 'simulation'

add_sweep_definition (*function: Union[Callable[[*idmtools.entities.simulation.Simulation*, Any], Dict[str, Any]], functools.partial]*, *values: Union[List[Any], Iterable]*)

Add a parameter sweep definition. A sweep definition is composed of a function and a list of values to call the function with.

Parameters

- **function** – The sweep function, which must include a **simulation** parameter (or whatever is specified in SIMULATION_ATTR). The function also must include EXACTLY ONE free parameter, which the values will be passed to. The function can also be a partial–any Callable type will work.
- **values** – The list of values to call the function with.

Examples

Examples of valid function:

```
def myFunction(simulation, parameter):
    pass
```

How to deal with functions requiring more than one parameter? Consider the following function:

```
python
def myFunction(simulation, a, b):
    pass
```

Partial solution:

```
python
from functools import partial
func = partial(myFunction, a=3)
eb.add_sweep_definition(func, [1,2,3])
```

Callable class solution:

```
class setP:
    def __init__(self, a):
        self.a = a

    def __call__(self, simulation, b):
        return param_update(simulation, self.a, b)

eb.add_sweep_definition(setP(3), [1,2,3])
```

idmtools.builders.yaml_simulation_builder module

class idmtools.builders.yaml_simulation_builder.**DefaultParamFuncDict**(*default*)
Bases: dict

class idmtools.builders.yaml_simulation_builder.**YamlSimulationBuilder**
Bases: *idmtools.builders.arm_simulation_builder.ArmSimulationBuilder*

Class that represents an experiment builder.

Examples

```
"""
        This file demonstrates how to use YamlExperimentBuilder in␣
↪PythonExperiment's builder.
        then adding the builder to PythonExperiment.

        We first load a yaml file from local dir which contains parameters/values␣
↪to sweep
        then sweep parameters based in yaml file with YamlExperimentBuilder
        Behind the scenes, we are using arm sweep, each group is treated with␣
↪SweepArm and then add to builder

        Parameters in yaml file
            group1:
                - a: 1
                - b: 2
                - c: [3, 4]
                - d: [5, 6]
            group2:
                - c: [3, 4]
                - d: [5, 6, 7]

        Expect sims with parameters:
            sim1: {a:1, b:2, c:3, d:5}
            sim2: {a:1, b:2, c:3, d:6}
            sim3: {a:1, b:2, c:4, d:5}
            sim4: {a:1, b:2, c:4, d:6}
            sim5: {c:3, d:5}
            sim6: {c:3, d:6}
            sim7: {c:3, d:7}
            sim8: {c:4, d:5}
            sim9: {c:4, d:6}
            sim10: {c:4, d:7}

        This builder is very similar with ArmExperimentBuilder. but in more␣
↪direct way. you just need list all cared
```

(continues on next page)

(continued from previous page)

```
        parameter combinations in yaml file, and let builder do the job

"""
import os
import sys
from functools import partial

from idmtools.builders import YamlSimulationBuilder
from idmtools.core.platform_factory import platform
from idmtools.entities.experiment import Experiment
from idmtools.entities.templated_simulation import TemplatedSimulations
from idmtools_models.python.json_python_task import JSONConfiguredPythonTask
from idmtools_test import COMMON_INPUT_PATH

# define function partials to be used during sweeps
setA = partial(JSONConfiguredPythonTask.set_parameter_sweep_callback, param="a")
setB = partial(JSONConfiguredPythonTask.set_parameter_sweep_callback, param="b")
setC = partial(JSONConfiguredPythonTask.set_parameter_sweep_callback, param="c")
setD = partial(JSONConfiguredPythonTask.set_parameter_sweep_callback, param="d")

if __name__ == "__main__":
    # define what platform we want to use. Here we use a context manager but if␣
↪you prefer you can
    # use objects such as Platform('COMPS2') instead
    with platform('COMPS2'):
        # define our base task
        base_task = JSONConfiguredPythonTask(script_path=os.path.join(COMMON_
↪INPUT_PATH, "python", "model1.py"),
                                             parameters=dict(c='c-value'))
        # define our input csv sweep
        base_path = os.path.abspath(os.path.join(COMMON_INPUT_PATH, "builder"))
        file_path = os.path.join(base_path, 'sweeps.yaml')
        builder = YamlSimulationBuilder()
        # define a list of functions to map the specific yaml values
        func_map = {'a': setA, 'b': setB, 'c': setC, 'd': setD}
        builder.add_sweeps_from_file(file_path, func_map)
        # optionally, if you can also pass a function that is used for all␣
↪parameters
        # The default behaviour of the builder is to assume the default function␣
↪will be a partial
        # and attempts to call it with one var(param) before building sweep
        # builder.add_sweeps_from_file(file_path, JSONConfiguredPythonTask.set_
↪parameter_partial)

        # now define we want to create a series of simulations using the base␣
↪task and the sweep
        ts = TemplatedSimulations.from_task(base_task)
        # optionally we could update the base simulation metdata here
        # ts.base_simulations.tags['example'] 'yes'
        ts.add_builder(builder)

        # define our experiment from our template and add some metadata to the␣
↪experiment
        experiment = Experiment.from_template(ts,
                                              name=os.path.split(sys.argv[0])[1],
                                              tags={"string_tag": "test", "number_
↪tag": 123}
```

(continues on next page)

(continued from previous page)

```
                                                    )

        # run the experiment and wait. By default run does not wait
        # in most real scenarios, you probably do not want to wait as this will␣
↪wait until all simulations
        # associated with an experiment are done. We do it in our examples to␣
↪show feature and to enable
        # testing of the scripts
        experiment.run(wait_until_done=True)
        # use system status as the exit code
        sys.exit(0 if experiment.succeeded else -1)
```

add_sweeps_from_file(*file_path*, *func_map: Union[Dict[str, Callable], Callable[[Any], Dict]] = None*, *sweep_type=<ArmType.cross: 0>*)

Add sweeps from a file

Parameters

- **file_path** – Path to file
- **func_map** – Optional function map
- **sweep_type** – Type of sweep

Returns:

Module contents

idmtools.config package

Submodules

idmtools.config.idm_config_parser module

idmtools.config.idm_config_parser.**initialization**(*error=False*, *force=False*)

class idmtools.config.idm_config_parser.**IdmConfigParser**(*dir_path: str = '.'*, *file_name: str = 'idmtools.ini'*)

Bases: object

Class that parses an INI configuration file.

classmethod retrieve_dict_config_block(*field_type*, *section*) → Dict[str, Any]

classmethod retrieve_settings()

classmethod get_section(**args*, ***kwargs*)

classmethod get_option(**args*, ***kwargs*)

classmethod ensure_init(*dir_path: str = '.'*, *file_name: str = 'idmtools.ini'*, *error: bool = False*, *force=False*) → None

Verify that the INI file loaded and a configparser instance is available.

Parameters

- **dir_path** – The directory to search for the INI configuration file.
- **file_name** – The configuration file name to search for.

Returns None

Raises **ValueError** – If the config file is found but cannot be parsed

classmethod get_config_path(**args*, ***kwargs*)

classmethod display_config_path(**args*, ***kwargs*)

classmethod view_config_file(**args*, ***kwargs*)

classmethod display_config_block_details(*block*)
Display the values of a config block

Parameters **block** – Block to print

Returns None

classmethod has_section(**args*, ***kwargs*)

classmethod has_option()

classmethod found_ini() → bool
Did we find the config?

Returns True if did, False Otherwise

classmethod clear_instance() → None
Uninitialize and clean the *IdmConfigParser* instance.

Returns None

Module contents

idmtools.core package

Subpackages

idmtools.core.interfaces package

Submodules

idmtools.core.interfaces.entity_container module

class idmtools.core.interfaces.entity_container.**EntityContainer**(*children: List[IEntity] = None*)

Bases: list

set_status(*status*)

set_status_for_item(*item_id*, *status*)

idmtools.core.interfaces.iassets_enabled module

class idmtools.core.interfaces.iassets_enabled.**IAssetsEnabled**(*assets: idmtools.assets.asset_collection.AssetCollecti = <factory>*)

Bases: object

Base class for objects containing an asset collection.

assets: AssetCollection

abstract gather_assets() → NoReturn
Function called at runtime to gather all assets in the collection.

add_assets(*assets: List[TAsset] = None*, *fail_on_duplicate: bool = True*) → NoReturn
Add more assets to *AssetCollection*.

add_asset(*asset: Optional[TAsset] = None*, *fail_on_duplicate: bool = True*) → NoReturn

idmtools.core.interfaces.ientity module

class idmtools.core.interfaces.ientity.**IEntity**(*_uid: uuid.UUID = None*, *platform_id: uuid.UUID = None*, *_platform: IPlatform = None*, *parent_id: uuid.UUID = None*, *_parent: IEntity = None*, *status: idmtools.core.enums.EntityStatus = None*, *tags: Dict[str, Any] = <factory>*, *item_type: idmtools.core.enums.ItemType = None*, *_platform_object: Any = None*)

Bases: *idmtools.core.interfaces.iitem.IItem*

Interface for all entities in the system.

platform_id: uuid.UUID = None
Platform ID

parent_id: uuid.UUID = None
Item's Parent ID

status: *idmtools.core.enums.EntityStatus* = None
Item's Status

tags: Dict[str, Any]
Item's tags

item_type: *idmtools.core.enums.ItemType* = None
Item Type

update_tags(*tags: dict = None*) → NoReturn
Shortcut to update the tags with the given dictionary :param tags: New tags

post_creation() → None
Called after the actual creation of the entity.

classmethod from_id(*item_id: Union[str, uuid.UUID]*, *platform: IPlatform = None*, ***kwargs*) → IEntity

property parent

property platform

get_platform_object(*force=False*, ***kwargs*)

property done

property succeeded

idmtools.core.interfaces.iitem module

class idmtools.core.interfaces.iitem.**IItem**(*_uid: uuid.UUID = None*)
Bases: object

property uid

property id

property metadata

property pickle_ignore_fields

property metadata_fields

display()

pre_creation() → None
Called before the actual creation of the entity.

post_creation() → None
Called after the actual creation of the entity.

post_setstate()
Function called after restoring the state if additional initialization is required

pre_getstate()
Function called before picking and return default values for "pickle-ignore" fields

idmtools.core.interfaces.inamed_entity module

class idmtools.core.interfaces.inamed_entity.**INamedEntity**(*_uid: uuid.UUID = None*, *platform_id: uuid.UUID = None*, *_platform: 'IPlatform' = None*, *parent_id: uuid.UUID = None*, *_parent: 'IEntity' = None*, *status: idmtools.core.enums.EntityStatus = None*, *tags: Dict[str, Any] = <factory>*, *item_type: idmtools.core.enums.ItemType = None*, *_platform_object: Any = None*, *name: str = None*)
Bases: *idmtools.core.interfaces.ientity.IEntity*

name: str = None

Module contents

Submodules

idmtools.core.cache_enabled module

class idmtools.core.cache_enabled.**CacheEnabled**
Bases: object

Allows a class to leverage Diskcache and expose a cache property.

initialize_cache(*shards: Optional[int] = None*, *eviction_policy=None*)
Initialize cache

Parameters

- **shards** (*Optional[int], optional*) – How many shards. It is best to set this when multi-procressing Defaults to None.
- **eviction_policy** (*[type], optional*) – See Diskcache docs. Defaults to None.

cleanup_cache()

property cache

idmtools.core.context module

idmtools.core.context.**set_current_platform**(*platform: IPlatform*)

idmtools.core.context.**remove_current_platform**()

idmtools.core.context.**get_current_platform**()

idmtools.core.docker_task module

class idmtools.core.docker_task.**DockerTask**(*command: Union[str, idmtools.entities.command_line.CommandLine] = None, platform_requirements: Set[idmtools.entities.platform_requirements.PlatformRequirements] = <factory>, _ITask__pre_creation_hooks: List[Callable[[Union[ForwardRef('Simulation'), ForwardRef('IWorkflowItem')]], NoReturn]] = <factory>, _ITask__post_creation_hooks: List[Callable[[Union[ForwardRef('Simulation'), ForwardRef('IWorkflowItem')]], NoReturn]] = <factory>, common_assets: idmtools.assets.asset_collection.AssetCollection = <factory>, transient_assets: idmtools.assets.asset_collection.AssetCollection = <factory>, _task_log: logging.Logger = <factory>, image_name: str = None, build: bool = False, build_path: Union[str, NoneType] = None, Dockerfile: Union[str, NoneType] = None, pull_before_build: bool = True, use_nvidia_run: bool = False, _DockerTask__image_built: bool = False*)

Bases: *idmtools.entities.itask.ITask*

image_name: str = None

build: bool = False

build_path: Optional[str] = None

Dockerfile: Optional[str] = None

pull_before_build: bool = True

use_nvidia_run: bool = False

gather_common_assets() → *idmtools.assets.asset_collection.AssetCollection*

Gather common(experiment-level) assets from task

Returns AssetCollection containing all the common assets

gather_transient_assets() → *idmtools.assets.asset_collection.AssetCollection*

Gather transient(simulation-level) assets from task

Returns AssetCollection

build_image(*spinner=None*, ***extra_build_args*)

reload_from_simulation(*simulation: Simulation*)

Optional hook that is called when loading simulations from a platform

class idmtools.core.docker_task.**DockerTaskSpecification**

Bases: *idmtools.registry.task_specification.TaskSpecification*

get(*configuration: dict*) → *idmtools.core.docker_task.DockerTask*

Get instance of DockerTask with configuration provided

Parameters **configuration** – configuration for DockerTask

Returns DockerTask with configuration

get_description() → str
Get description of plugin

Returns Plugin description

get_type() → Type[*idmtools.core.docker_task.DockerTask*]
Get type of task provided by plugin

Returns DockerTask

idmtools.core.enums module

class idmtools.core.enums.**EntityStatus**(*value*)
Bases: enum.Enum

An enumeration.

CREATED = 'created'

RUNNING = 'running'

SUCCEEDED = 'succeeded'

FAILED = 'failed'

class idmtools.core.enums.**FilterMode**(*value*)
Bases: enum.Enum

Allows user to specify AND/OR for the filtering system.

AND = 0

OR = 1

class idmtools.core.enums.**ItemType**(*value*)
Bases: enum.Enum

An enumeration.

SUITE = 1

EXPERIMENT = 2

SIMULATION = 3

WORKFLOW_ITEM = 4

ASSETCOLLECTION = 5

idmtools.core.exceptions module

exception idmtools.core.exceptions.**ExperimentNotFound**(*experiment_id: uuid.UUID, platform: TPlatform = None*)
Bases: Exception

exception idmtools.core.exceptions.**UnknownItemException**(*err: str*)
Bases: Exception

exception idmtools.core.exceptions.**NoPlatformException**
Bases: Exception

Cannot find a platform matching the one requested by user

exception idmtools.core.exceptions.**TopLevelItem**
Bases: Exception

Thrown when a parent of a top-level item is requested by the platform

exception idmtools.core.exceptions.**UnsupportedPlatformType**
Bases: Exception

Occurs when an item is not supported by a platform but is requested

exception idmtools.core.exceptions.**NoTaskFound**
Bases: Exception

idmtools.core.experiment_factory module

class idmtools.core.experiment_factory.**ExperimentFactory**
Bases: object

DEFAULT_KEY = 'idmtools.entities.experiment.Experiment'

create(*key*, *fallback=None*, ***kwargs*) → *idmtools.entities.experiment.Experiment*

idmtools.core.logging module

class idmtools.core.logging.**IDMQueueListener**(*queue*, **handlers*, *respect_handler_level=False*)
Bases: logging.handlers.QueueListener

dequeue(*block*)
Dequeue a record and return it, optionally blocking.

The base implementation uses get. You may want to override this method if you want to use timeouts or work with custom queue implementations.

class idmtools.core.logging.**IDMQueueHandler**(*queue*)
Bases: logging.handlers.QueueHandler

prepare(*record*)
Prepares a record for queuing. The object returned by this method is enqueued.

The base implementation formats the record to merge the message and arguments, and removes unpickleable items from the record in-place.

You might want to override this method if you want to convert the record to a dict or JSON string, or send a modified copy of the record while leaving the original intact.

idmtools.core.logging.**setup_logging**(*level: Union[int, str] = 30*, *log_filename: str = 'idmtools.log'*, *console: Union[str, bool] = False*) → logging.handlers.QueueListener
Set up logging.

Parameters

- **level** – Log level. Default to warning. This should be either a string that matches a log level from logging or an int that represent that level.
- **log_filename** – Name of file to log messages to.
- **console** – When set to True or the strings “1”, “y”, “yes”, or “on”, console logging will be enabled.

Returns Returns the `QueueListener` created that writes the log messages. In advanced scenarios with multi-processing, you may need to manually stop the logger.

See also:

For logging levels, see https://coloredlogs.readthedocs.io/en/latest/api.html#id26

idmtools.core.logging.**setup_handlers**(*level*, *log_filename*, *console: bool = False*)

idmtools.core.logging.**exclude_logging_classes**(*items_to_exclude=None*)

idmtools.core.logging.**register_stop_logger_signal_handler**(*listener*) → NoReturn
Register a signal watcher that will stop our logging gracefully in the case of queue based logging.

Parameters **listener** – The log listener object.

Returns None

idmtools.core.platform_factory module

idmtools.core.platform_factory.**platform**(**args*, ***kwds*)

class idmtools.core.platform_factory.**Platform**(*block*, ***kwargs*)
Bases: `object`

idmtools.core.system_information module

idmtools.core.system_information.**get_data_directory**() → str

idmtools.core.system_information.**get_filtered_environment_vars**(*exclude=None*)

class idmtools.core.system_information.**SystemInformation**(*data_directory: Union[str, NoneType] = '/home/docs/.local_data', user: Union[str, NoneType] = 'docs', python_version: str = '3.7.3', python_build: str = ('default', 'Jan 24 2020 02:22:02'), python_packages: List[str] = <factory>, environment_variables: Dict[str, str] = <factory>, os_name: str = 'Linux', hostname: str = 'build-393625-project-5702-institute-for-disease-modeling-idmtool', system_version: str = '#53-Ubuntu SMP Wed Sep 18 13:35:53 UTC 2019', system_architecture: str = 'x86_64', system_processor: str = 'x86_64', system_architecture_details: str = ('64bit', ''), default_docket_socket_path: str = '/var/run/docker.sock', cwd: str = '/home/docs/checkouts/readthedocs.org/user_build/for-disease-modeling-idmtools/checkouts/v1.4.0/docs', user_group_str: str = '1000:1000', version: str = '1.4.0.0'*)

Bases: object

data_directory: Optional[str] = '/home/docs/.local_data'

user: Optional[str] = 'docs'

python_version: str = '3.7.3'

python_build: str = ('default', 'Jan 24 2020 02:22:02')

python_implementation = 'CPython'

python_packages: List[str]

environment_variables: Dict[str, str]

os_name: str = 'Linux'

hostname: str = 'build-393625-project-5702-institute-for-disease-modeling-idmtool'

system_version: str = '#53-Ubuntu SMP Wed Sep 18 13:35:53 UTC 2019'

```
system_architecture:  str = 'x86_64'
system_processor:  str = 'x86_64'
system_architecture_details:  str = ('64bit', '')
default_docket_socket_path:  str = '/var/run/docker.sock'
cwd:  str = '/home/docs/checkouts/readthedocs.org/user_builds/institute-for-disease-mo
user_group_str:  str = '1000:1000'
version:  str = '1.4.0.0'
```

class idmtools.core.system_information.**LinuxSystemInformation**(*data_directory: Union[str, NoneType] = '/home/docs/.local_data', user: Union[str, NoneType] = 'docs', python_version: str = '3.7.3', python_build: str = ('default', 'Jan 24 2020 02:22:02'), python_packages: List[str] = <factory>, environment_variables: Dict[str, str] = <factory>, os_name: str = 'Linux', hostname: str = 'build-393625-project-5702-institute-for-disease-modeling-idmtool', system_version: str = '#53-Ubuntu SMP Wed Sep 18 13:35:53 UTC 2019', system_architecture: str = 'x86_64', system_processor: str = 'x86_64', system_architecture_details: str = ('64bit', ''), default_docket_socket_path: str = '/var/run/docker.sock', cwd: str = '/home/docs/checkouts/readthedocs.org/use for-disease-modeling-idmtools/checkouts/v1.4.0/docs', user_group_str: str = <factory>, version: str = '1.4.0.0'*)

Bases: *idmtools.core.system_information.SystemInformation*

class idmtools.core.system_information.**WindowsSystemInformation**(*data_directory: Optional[str] = '/home/docs/.local_data', user: Optional[str] = 'docs', python_version: str = '3.7.3', python_build: str = ('default', 'Jan 24 2020 02:22:02'), python_packages: List[str] = <factory>, environment_variables: Dict[str, str] = <factory>, os_name: str = 'Linux', hostname: str = 'build-393625-project-5702-institute-for-disease-modeling-idmtool', system_version: str = '#53-Ubuntu SMP Wed Sep 18 13:35:53 UTC 2019', system_architecture: str = 'x86_64', system_processor: str = 'x86_64', system_architecture_details: str = ('64bit', ''), default_docket_socket_path: str = '/var/run/docker.sock', cwd: str = '/home/docs/checkouts/readthedocs.org/for-disease-modeling-idmtools/checkouts/v1.4.0/docs', user_group_str: str = '1000:1000', version: str = '1.4.0.0'*)

Bases: *idmtools.core.system_information.SystemInformation*

default_docket_socket_path: str = '//var/run/docker.sock'

idmtools.core.system_information.**get_system_information**() → *idmtools.core.system_information.SystemInformation*

Fetch the system-appropriate information inspection object.

Returns *SystemInformation* with platform-specific implementation.

idmtools.core.task_factory module

class idmtools.core.task_factory.**DynamicTaskSpecification**(*task_type: Type[*idmtools.entities.itask.ITask*]*, *description: str = ''*)

Bases: *idmtools.registry.task_specification.TaskSpecification*

This class allows users to quickly define a spec for special tasks

get(*configuration: dict*) → *idmtools.entities.itask.ITask*

Return a new model using the passed in configuration.

Parameters **configuration** – The INI configuration file to use.

Returns The new model.

get_description() → str

Get a brief description of the plugin and its functionality.

Returns The plugin description.

get_type() → Type[*idmtools.entities.itask.ITask*]

class idmtools.core.task_factory.**TaskFactory**

Bases: object

DEFAULT_KEY = 'idmtools.entities.command_task.CommandTask'

register(*spec:* idmtools.registry.task_specification.TaskSpecification) → NoReturn

Register a TaskSpecification dynamically

Parameters **spec** – Specification to register

Returns:

register_task(*task: Type[*idmtools.entities.itask.ITask*]*) → NoReturn

Dynamically register a class using the DynamicTaskSpecification

Parameters **task** – Task to register

Returns:

create(*key*, *fallback=None*, ***kwargs*) → *idmtools.entities.itask.ITask*

Module contents

idmtools.entities package

Subpackages

idmtools.entities.iplatform_ops package

Submodules

idmtools.entities.iplatform_ops.iplatform_asset_collection_operations module

class idmtools.entities.iplatform_ops.iplatform_asset_collection_operations.**IPlatformAsset**

Bases: *idmtools.core.cache_enabled.CacheEnabled*, abc.ABC

platform: 'IPlatform'

platform_type: Type

pre_create(*asset_collection:* idmtools.assets.asset_collection.AssetCollection, ***kwargs*) → NoReturn

Run the platform/AssetCollection post creation events

Parameters

- **asset_collection** – AssetCollection to run post-creation events
- ****kwargs** – Optional arguments mainly for extensibility

Returns NoReturn

post_create(*asset_collection:* idmtools.assets.asset_collection.AssetCollection, ***kwargs*) → NoReturn

Run the platform/AssetCollection post creation events

Parameters

- **asset_collection** – AssetCollection to run post-creation events
- ****kwargs** – Optional arguments mainly for extensibility

Returns NoReturn

create(*asset_collection:* idmtools.assets.asset_collection.AssetCollection, *do_pre: bool = True*, *do_post: bool = True*, ***kwargs*) → Any

Creates an AssetCollection from an IDMTools AssetCollection object. Also performs pre-creation and post-creation locally and on platform

Parameters

- **asset_collection** – AssetCollection to create
- **do_pre** – Perform Pre creation events for item
- **do_post** – Perform Post creation events for item
- ****kwargs** – Optional arguments mainly for extensibility

Returns Created platform item and the UUID of said item

abstract platform_create(*asset_collection:* idmtools.assets.asset_collection.AssetCollection, ***kwargs*) → Any
Creates an workflow_item from an IDMTools AssetCollection object

Parameters

- **asset_collection** – AssetCollection to create
- ****kwargs** – Optional arguments mainly for extensibility

Returns Created platform item and the UUID of said item

batch_create(*asset_collections:* *List[*idmtools.assets.asset_collection.AssetCollection*]*, *display_progress:* *bool* = *True*, ***kwargs*) → List[*idmtools.assets.asset_collection.AssetCollection*]
Provides a method to batch create asset collections items

Parameters

- **asset_collections** – List of asset collection items to create
- **display_progress** – Show progress bar
- ****kwargs** –

Returns List of tuples containing the create object and id of item that was created

abstract get(*asset_collection_id: uuid.UUID*, ***kwargs*) → Any
Returns the platform representation of an AssetCollection

Parameters

- **asset_collection_id** – Item id of AssetCollection
- ****kwargs** –

Returns Platform Representation of an AssetCollection

to_entity(*asset_collection: Any*, ***kwargs*) → *idmtools.assets.asset_collection.AssetCollection*
Converts the platform representation of AssetCollection to idmtools representation

Parameters **asset_collection** – Platform AssetCollection object

Returns IDMTools suite object

idmtools.entities.iplatform_ops.iplatform_experiment_operations module

class idmtools.entities.iplatform_ops.iplatform_experiment_operations.**IPlatformExperimentOp**

Bases: abc.ABC

platform: 'IPlatform'

platform_type: Type

abstract get(*experiment_id: uuid.UUID*, ***kwargs*) → Any
Returns the platform representation of an Experiment

Parameters

- **experiment_id** – Item id of Experiments
- ****kwargs** –

Returns Platform Representation of an experiment

pre_create(*experiment:* idmtools.entities.experiment.Experiment, ***kwargs*) → NoReturn
Run the platform/experiment post creation events

Parameters

- **experiment** – Experiment to run post-creation events
- ****kwargs** – Optional arguments mainly for extensibility

Returns NoReturn

post_create(*experiment:* idmtools.entities.experiment.Experiment, ***kwargs*) → NoReturn
Run the platform/experiment post creation events

Parameters

- **experiment** – Experiment to run post-creation events
- ****kwargs** – Optional arguments mainly for extensibility

Returns NoReturn

create(*experiment:* idmtools.entities.experiment.Experiment, *do_pre: bool = True*, *do_post: bool = True*, ***kwargs*) → *idmtools.entities.experiment.Experiment*
Creates an experiment from an IDMTools simulation object. Also performs local/platform pre and post creation events

Parameters

- **experiment** – Experiment to create
- **do_pre** – Perform Pre creation events for item
- **do_post** – Perform Post creation events for item
- ****kwargs** – Optional arguments mainly for extensibility

Returns Created platform item and the UUID of said item

abstract platform_create(*experiment:* idmtools.entities.experiment.Experiment, ***kwargs*) → Any
Creates an experiment from an IDMTools experiment object

Parameters

- **experiment** – Experiment to create
- ****kwargs** – Optional arguments mainly for extensibility

Returns Created platform item and the UUID of said item

batch_create(*experiments: List[*idmtools.entities.experiment.Experiment*]*, *display_progress: bool = True*, ***kwargs*) → List[Tuple[*idmtools.entities.experiment.Experiment*]]
Provides a method to batch create experiments

Parameters

- **experiments** – List of experiments to create
- **display_progress** – Show progress bar
- ****kwargs** –

Returns List of tuples containing the create object and id of item that was created

abstract get_children(*experiment: Any*, ***kwargs*) → List[Any]
Returns the children of an experiment object

Parameters

- **experiment** – Experiment object
- ****kwargs** – Optional arguments mainly for extensibility

Returns Children of experiment object

abstract get_parent(*experiment: Any*, ***kwargs*) → Any
Returns the parent of item. If the platform doesn't support parents, you should throw a TopLevelItem error

Parameters

- **experiment** –
- ****kwargs** –

Returns:

Raise: TopLevelItem

to_entity(*experiment: Any*, ***kwargs*) → *idmtools.entities.experiment.Experiment*
Converts the platform representation of experiment to idmtools representation

Parameters **experiment** – Platform experiment object

Returns IDMTools experiment object

pre_run_item(*experiment:* idmtools.entities.experiment.Experiment, ***kwargs*)

Trigger right before commissioning experiment on platform. This ensures that the item is created. It also ensures that the children(simulations) have also been created

Parameters **experiment** – Experiment to commission

Returns:

post_run_item(*experiment:* idmtools.entities.experiment.Experiment, ***kwargs*)
Trigger right after commissioning experiment on platform.

Parameters **experiment** – Experiment just commissioned

Returns:

run_item(*experiment:* idmtools.entities.experiment.Experiment, ***kwargs*)
Called during commissioning of an item. This should create the remote resource

Parameters **experiment** –

Returns:

abstract platform_run_item(*experiment:* idmtools.entities.experiment.Experiment, ***kwargs*)
Called during commissioning of an item. This should perform what is needed to commission job on platform

Parameters **experiment** –

Returns:

abstract send_assets(*experiment: Any*, ***kwargs*)
Transfer Experiment assets to the platform. :param experiment: Experiment to send assets for

Returns:

abstract refresh_status(*experiment:* idmtools.entities.experiment.Experiment, ***kwargs*)
Refresh status for experiment object. This should update the object directly. For experiments it is best if all simulation states are updated as well

Parameters **experiment** – Experiment to get status for

Returns None

get_assets(*experiment:* idmtools.entities.experiment.Experiment, *files: List[str]*, ***kwargs*) → Dict[str, Dict[str, bytearray]]
Get files from experiment

Parameters

- **experiment** – Experiment to get files from
- **files** – List files
- ****kwargs** –

Returns Dict with each sim id and the files contents matching specified list

list_assets(*experiment:* idmtools.entities.experiment.Experiment, *children: bool = False*, ***kwargs*) → List[*idmtools.assets.asset.Asset*]
List available assets for a experiment

Parameters

- **experiment** – Experiment to list files for
- **children** – Should we load assets from children as well?

Returns List of Assets

platform_list_asset(*experiment:* idmtools.entities.experiment.Experiment, ***kwargs*) → List[*idmtools.assets.asset.Asset*]

idmtools.entities.iplatform_ops.iplatform_simulation_operations module

class idmtools.entities.iplatform_ops.iplatform_simulation_operations.**IPlatformSimulationOp**

Bases: *idmtools.core.cache_enabled.CacheEnabled*, abc.ABC

platform: 'IPlatform'

platform_type: Type

abstract get(*simulation_id: uuid.UUID*, ***kwargs*) → Any
Returns the platform representation of an Simulation

Parameters

- **simulation_id** – Item id of Simulations
- ****kwargs** –

Returns Platform Representation of an simulation

pre_create(*simulation:* idmtools.entities.simulation.Simulation, ***kwargs*) → NoReturn
Run the platform/simulation post creation events

Parameters

- **simulation** – simulation to run post-creation events
- ****kwargs** – Optional arguments mainly for extensibility

Returns NoReturn

post_create(*simulation:* idmtools.entities.simulation.Simulation, ***kwargs*) → NoReturn
Run the platform/simulation post creation events

Parameters

- **simulation** – simulation to run post-creation events
- ****kwargs** – Optional arguments mainly for extensibility

Returns NoReturn

create(*simulation:* idmtools.entities.simulation.Simulation, *do_pre: bool = True*, *do_post: bool = True*, ***kwargs*) → Any
Creates an simulation from an IDMTools simulation object. Also performs pre-creation and post-creation locally and on platform

Parameters

- **simulation** – Simulation to create
- **do_pre** – Perform Pre creation events for item
- **do_post** – Perform Post creation events for item
- ****kwargs** – Optional arguments mainly for extensibility

Returns Created platform item and the UUID of said item

abstract platform_create(*simulation:* idmtools.entities.simulation.Simulation, ***kwargs*) → Any
Creates an simulation on Platform from an IDMTools Simulation Object

Parameters

- **simulation** – Simulation to create
- ****kwargs** – Optional arguments mainly for extensibility

Returns Created platform item and the UUID of said item

batch_create(*sims: List[*idmtools.entities.simulation.Simulation*]*, *display_progress: bool = True*, ***kwargs*) → List[*idmtools.entities.simulation.Simulation*]
Provides a method to batch create simulations

Parameters

- **sims** – List of simulations to create
- **display_progress** – Show progress bar
- ****kwargs** –

Returns List of tuples containing the create object and id of item that was created

abstract get_parent(*simulation: Any*, ***kwargs*) → Any
Returns the parent of item. If the platform doesn't support parents, you should throw a TopLevelItem error

Parameters

- **simulation** –
- ****kwargs** –

Returns:

Raise: TopLevelItem

to_entity(*simulation: Any*, *load_task: bool = False*, *parent: Optional[idmtools.entities.experiment.Experiment] = None*, ***kwargs*) → *idmtools.entities.simulation.Simulation*

Converts the platform representation of simulation to idmtools representation

Parameters

- **simulation** – Platform simulation object
- **load_task** – Load Task Object as well. Can take much longer and have more data on platform
- **parent** – Optional parent object

Returns IDMTools simulation object

pre_run_item(*simulation:* idmtools.entities.simulation.Simulation, ***kwargs*)

Trigger right before commissioning experiment on platform. This ensures that the item is created. It also ensures that the children(simulations) have also been created

Parameters **simulation** – Experiment to commission

Returns:

post_run_item(*simulation:* idmtools.entities.simulation.Simulation, ***kwargs*)

Trigger right after commissioning experiment on platform.

Parameters **simulation** – Experiment just commissioned

Returns:

run_item(*simulation:* idmtools.entities.simulation.Simulation, ***kwargs*)

Called during commissioning of an item. This should create the remote resource

Parameters **simulation** –

Returns:

abstract platform_run_item(*simulation:* idmtools.entities.simulation.Simulation, ***kwargs*)

Called during commissioning of an item. This should create the remote resource but not upload assets

Parameters **simulation** – Simulation to run

Returns:

abstract send_assets(*simulation: Any*, ***kwargs*)

abstract refresh_status(*simulation:* idmtools.entities.simulation.Simulation, ***kwargs*)

Refresh status for simulation object

Parameters **simulation** – Experiment to get status for

Returns None

abstract get_assets(*simulation:* idmtools.entities.simulation.Simulation, *files: List[str]*, ***kwargs*) → Dict[str, bytearray]
Get files from simulation

Parameters

- **simulation** – Simulation to fetch files from
- **files** – Files to get
- ****kwargs** –

Returns Dictionary containing filename and content

abstract list_assets(*simulation:* idmtools.entities.simulation.Simulation, ***kwargs*) → List[*idmtools.assets.asset.Asset*]
List available assets for a simulation

Parameters **simulation** – Simulation of assets Assets

Returns List of filenames

idmtools.entities.iplatform_ops.iplatform_suite_operations module

class idmtools.entities.iplatform_ops.iplatform_suite_operations.**IPlatformSuiteOperations**(

Bases: abc.ABC

platform: 'IPlatform'

platform_type: Type

abstract get(*suite_id: uuid.UUID*, ***kwargs*) → Any
Returns the platform representation of an Suite

Parameters

- **suite_id** – Item id of Suites
- ****kwargs** –

Returns Platform Representation of an suite

batch_create(*suites: List[*idmtools.entities.suite.Suite*]*, *display_progress: bool = True*, ***kwargs*) → List[Tuple[Any, uuid.UUID]]
Provides a method to batch create suites

Parameters

- **display_progress** – Display progress bar
- **suites** – List of suites to create
- ****kwargs** –

Returns List of tuples containing the create object and id of item that was created

pre_create(*suite:* idmtools.entities.suite.Suite, ***kwargs*) → NoReturn
Run the platform/suite post creation events

Parameters

- **suite** – Experiment to run post-creation events
- ****kwargs** – Optional arguments mainly for extensibility

Returns NoReturn

post_create(*suite:* idmtools.entities.suite.Suite, ***kwargs*) → NoReturn
Run the platform/suite post creation events

Parameters

- **suite** – Experiment to run post-creation events
- ****kwargs** – Optional arguments mainly for extensibility

Returns NoReturn

create(*suite:* idmtools.entities.suite.Suite, *do_pre: bool = True*, *do_post: bool = True*, ***kwargs*) → Tuple[Any, uuid.UUID]
Creates an simulation from an IDMTools suite object. Also performs pre-creation and post-creation locally and on platform

Parameters

- **suite** – Suite to create
- **do_pre** – Perform Pre creation events for item
- **do_post** – Perform Post creation events for item
- ****kwargs** – Optional arguments mainly for extensibility

Returns Created platform item and the UUID of said item

abstract platform_create(*suite:* idmtools.entities.suite.Suite, ***kwargs*) → Tuple[Any, uuid.UUID]
Creates an suite from an IDMTools suite object

Parameters

- **suite** – Suite to create
- ****kwargs** – Optional arguments mainly for extensibility

Returns Created platform item and the UUID of said item

pre_run_item(*suite:* idmtools.entities.suite.Suite, ***kwargs*)

Trigger right before commissioning experiment on platform. This ensures that the item is created. It also ensures that the children(simulations) have also been created

Parameters **suite** – Experiment to commission

Returns:

post_run_item(*suite:* idmtools.entities.suite.Suite, ***kwargs*)
Trigger right after commissioning suite on platform.

Parameters **suite** – Experiment just commissioned

Returns:

run_item(*suite:* idmtools.entities.suite.Suite, ***kwargs*)
Called during commissioning of an item. This should create the remote resource

Parameters **workflow_item** –

Returns:

platform_run_item(*suite:* idmtools.entities.suite.Suite, ***kwargs*)
Called during commissioning of an item. This should perform what is needed to commission job on platform

Parameters **suite** –

Returns:

abstract get_parent(*suite: Any*, ***kwargs*) → Any
Returns the parent of item. If the platform doesn't support parents, you should throw a TopLevelItem error

Parameters

- **suite** –
- ****kwargs** –

Returns:

Raise: TopLevelItem

abstract get_children(*suite: Any*, ***kwargs*) → List[Any]
Returns the children of an suite object

Parameters

- **suite** – Suite object
- ****kwargs** – Optional arguments mainly for extensibility

Returns Children of suite object

to_entity(*suite: Any*, ***kwargs*) → *idmtools.entities.suite.Suite*
Converts the platform representation of suite to idmtools representation

Parameters **suite** – Platform suite object

Returns IDMTools suite object

abstract refresh_status(*experiment:* idmtools.entities.suite.Suite, ***kwargs*)
Refresh status of suite :param experiment:

Returns:

get_assets(*suite:* idmtools.entities.suite.Suite, *files: List[str]*, ***kwargs*) → Dict[str, Dict[str, Dict[str, bytearray]]]
Fetch assets for suite :param suite: suite to get assets for :param files: Files to load :param **kwargs:

Returns Nested dictionaries in the structure experiment_id { simulation_id { files = content } } }

idmtools.entities.iplatform_ops.iplatform_workflowitem_operations module

class idmtools.entities.iplatform_ops.iplatform_workflowitem_operations.**IPlatformWorkflowIt**

Bases: *idmtools.core.cache_enabled.CacheEnabled*, abc.ABC

platform: 'IPlatform'

platform_type: Type

abstract get(*workflow_item_id: uuid.UUID*, ***kwargs*) → Any
Returns the platform representation of an WorkflowItem

Parameters

- **workflow_item_id** – Item id of WorkflowItems
- ****kwargs** –

Returns Platform Representation of an workflow_item

batch_create(*workflow_items: List[*idmtools.entities.iworkflow_item.IWorkflowItem*]*, *display_progress: bool = True*, ***kwargs*) → List[Any]
Provides a method to batch create workflow items

Parameters

- **workflow_items** – List of worfklow items to create
- **display_progress** – Whether to display progress bar
- ****kwargs** –

Returns List of tuples containing the create object and id of item that was created

pre_create(*workflow_item:* idmtools.entities.iworkflow_item.IWorkflowItem, ***kwargs*) → NoReturn
Run the platform/workflow item post creation events

Parameters

- **workflow_item** – IWorkflowItem to run post-creation events
- ****kwargs** – Optional arguments mainly for extensibility

Returns NoReturn

post_create(*workflow_item:* idmtools.entities.iworkflow_item.IWorkflowItem, ***kwargs*) → NoReturn
Run the platform/workflow item post creation events

Parameters

- **workflow_item** – IWorkflowItem to run post-creation events
- ****kwargs** – Optional arguments mainly for extensibility

Returns NoReturn

create(*workflow_item:* idmtools.entities.iworkflow_item.IWorkflowItem, *do_pre: bool = True*, *do_post: bool = True*, ***kwargs*) → Any
Creates an workflow item from an IDMTools IWorkflowItem object. Also performs pre-creation and post-creation locally and on platform

Parameters

- **workflow_item** – Suite to create
- **do_pre** – Perform Pre creation events for item
- **do_post** – Perform Post creation events for item
- ****kwargs** – Optional arguments mainly for extensibility

Returns Created platform item and the UUID of said item

abstract platform_create(*workflow_item:* idmtools.entities.iworkflow_item.IWorkflowItem, ***kwargs*) → Tuple[Any, uuid.UUID]
Creates an workflow_item from an IDMTools workflow_item object

Parameters

- **workflow_item** – WorkflowItem to create
- ****kwargs** – Optional arguments mainly for extensibility

Returns Created platform item and the UUID of said item

pre_run_item(*workflow_item:* idmtools.entities.iworkflow_item.IWorkflowItem, ***kwargs*)

Trigger right before commissioning experiment on platform. This ensures that the item is created. It also ensures that the children(simulations) have also been created

Parameters **workflow_item** – Experiment to commission

Returns:

post_run_item(*workflow_item:* idmtools.entities.iworkflow_item.IWorkflowItem, ***kwargs*)

Trigger right after commissioning workflow item on platform.

Parameters **workflow_item** – Experiment just commissioned

Returns:

run_item(*workflow_item:* idmtools.entities.iworkflow_item.IWorkflowItem, ***kwargs*)

Called during commissioning of an item. This should create the remote resource

Parameters **workflow_item** –

Returns:

abstract platform_run_item(*workflow_item:* idmtools.entities.iworkflow_item.IWorkflowItem, ***kwargs*)

Called during commissioning of an item. This should perform what is needed to commission job on platform

Parameters **workflow_item** –

Returns:

abstract get_parent(*workflow_item: Any*, ***kwargs*) → Any

Returns the parent of item. If the platform doesn't support parents, you should throw a TopLevelItem error

Parameters

- **workflow_item** –
- ****kwargs** –

Returns:

Raise: TopLevelItem

abstract get_children(*workflow_item: Any*, ***kwargs*) → List[Any]

Returns the children of an workflow_item object

Parameters

- **workflow_item** – WorkflowItem object
- ****kwargs** – Optional arguments mainly for extensibility

Returns Children of workflow_item object

to_entity(*workflow_item: Any*, ***kwargs*) → *idmtools.entities.iworkflow_item.IWorkflowItem*

Converts the platform representation of workflow_item to idmtools representation

Parameters **workflow_item** – Platform workflow_item object

Returns IDMTools workflow_item object

abstract refresh_status(*workflow_item:* idmtools.entities.iworkflow_item.IWorkflowItem, ***kwargs*)
Refresh status for workflow item :param workflow_item: Item to refresh status for

Returns None

abstract send_assets(*workflow_item: Any*, ***kwargs*)
Send assets for workflow item to platform

Parameters **workflow_item** – Item to send assets for

Returns:

abstract get_assets(*workflow_item:* idmtools.entities.iworkflow_item.IWorkflowItem, *files: List[str]*, ***kwargs*) → Dict[str, bytearray]
Load assets for workflow item :param workflow_item: Item :param files: List of files to load :param **kwargs:

Returns Dictionary with filename as key and value as binary content

abstract list_assets(*workflow_item:* idmtools.entities.iworkflow_item.IWorkflowItem, ***kwargs*) → List[*idmtools.assets.asset.Asset*]
List available assets for a workflow item

Parameters **workflow_item** – workflow item to list files for

Returns List of filenames

idmtools.entities.iplatform_ops.utils module

idmtools.entities.iplatform_ops.utils.**batch_items**(*items: Union[Iterable, Generator]*, *batch_size=16*)
Batch items

Parameters

- **items** –
- **batch_size** –

Returns:

idmtools.entities.iplatform_ops.utils.**item_batch_worker_thread**(*create_func: Callable*, *items: List*) → List
Default batch worker thread function. It just calls create on each item

Parameters

- **create_func** – Create function for item
- **items** – Items to create

Returns List of items created

idmtools.entities.iplatform_ops.utils.**batch_create_items**(*items: Union[Iterable, Generator]*, *batch_worker_thread_func: Callable[[List], List] = None*, *create_func: Callable[[...], Any] = None*, *display_progress: bool = True*, *progress_description: str = 'Commissioning items'*, ***kwargs*)

Batch create items. You must specify either batch_worker_thread_func or create_func

Parameters

- **items** – Items to create
- **batch_worker_thread_func** – Optional Function to execute. Should take a list and return a list
- **create_func** – Optional Create function
- **display_progress** – Enable progress bar
- **progress_description** – Description to show in progress bar
- ****kwargs** –

Returns:

idmtools.entities.iplatform_ops.utils.**show_progress_of_batch**(*futures: List[concurrent.futures._base.Future]*, *progress_description: str*, *total: int*) → List

Show progress bar for batch

Parameters

- **futures** – List of futures that are still running/queued
- **progress_description** – Progress description
- **total** – Total items being loaded(since we are loading in batches)

Returns:

Module contents

Submodules

idmtools.entities.command_line module

class idmtools.entities.command_line.**CommandLine**(*executable=None*, **args*, ***kwargs*)

Bases: object

A class to construct command line strings from executable, options, and params

property executable

add_argument(*arg*)

add_option(*option*, *value*)

property options

property arguments

property cmd

idmtools.entities.command_task module

class idmtools.entities.command_task.**CommandTask**(*command: Union[str, idmtools.entities.command_line.CommandLine] = None, platform_requirements: Set[idmtools.entities.platform_requirements.PlatformRequirer = <factory>, _ITask__pre_creation_hooks: List[Callable[[Union[ForwardRef('Simulation'), ForwardRef('IWorkflowItem')]], NoReturn]] = <factory>, _ITask__post_creation_hooks: List[Callable[[Union[ForwardRef('Simulation'), ForwardRef('IWorkflowItem')]], NoReturn]] = <factory>, common_assets: idmtools.assets.asset_collection.AssetCollection = <factory>, transient_assets: idmtools.assets.asset_collection.AssetCollection = <factory>, _task_log: logging.Logger = <factory>, gather_common_asset_hooks: List[Callable[[idmtools.entities.itask.ITask], idmtools.assets.asset_collection.AssetCollection]] = <factory>, gather_transient_asset_hooks: List[Callable[[idmtools.entities.itask.ITask], idmtools.assets.asset_collection.AssetCollection]] = <factory>*)

Bases: *idmtools.entities.itask.ITask*

gather_common_asset_hooks: List[Callable[[ITask], AssetCollection]]
Hooks to gather common assets

gather_transient_asset_hooks: List[Callable[[ITask], AssetCollection]]
Defines an extensible simple task that implements functionality through optional supplied use hooks

gather_common_assets() → *idmtools.assets.asset_collection.AssetCollection*
Gather common(experiment-level) assets for task

Returns AssetCollection containing common assets

gather_transient_assets() → *idmtools.assets.asset_collection.AssetCollection*
Gather transient(experiment-level) assets for task

Returns AssetCollection containing transient assets

reload_from_simulation(*simulation: Simulation*)
Optional hook that is called when loading simulations from a platform

class idmtools.entities.command_task.**CommandTaskSpecification**
Bases: *idmtools.registry.task_specification.TaskSpecification*

get(*configuration: dict*) → *idmtools.entities.command_task.CommandTask*
Get instance of CommandTask with configuration

Parameters **configuration** – configuration for CommandTask

Returns CommandTask with configuration

get_description() → str
Get description of plugin

Returns Plugin description

get_example_urls() → List[str]
Get example urls related to CommandTask

Returns List of urls that have examples related to CommandTask

get_type() → Type[*idmtools.entities.command_task.CommandTask*]
Get task type provided by plugin

Returns CommandTask

idmtools.entities.experiment module

class idmtools.entities.experiment.**Experiment**(*_uid: uuid.UUID = None, platform_id: uuid.UUID = None, _platform: IPlatform = None, parent_id: uuid.UUID = None, _parent: IEntity = None, status: idmtools.core.enums.EntityStatus = None, tags: Dict[str, Any] = <factory>, _platform_object: Any = None, name: str = None, assets: idmtools.assets.asset_collection.AssetCollection = <factory>, suite_id: <module 'uuid' from '/home/docs/.pyenv/versions/3.7.3/lib/python3.7/uuid.py'> = None, task_type: str = 'idmtools.entities.command_task.CommandTask', platform_requirements: Set[idmtools.entities.platform_requirements.PlatformRequirement = <factory>, simulations: dataclasses.InitVar = <property object>, _Experiment__simulations: Union[idmtools.core.interfaces.entity_container.EntityContainer, Generator[Simulation, None, None], idmtools.entities.templated_simulation.TemplatedSimulations, Iterator[Simulation]] = <factory>, gather_common_assets_from_task: bool = None*)
Bases: *idmtools.core.interfaces.iassets_enabled.IAssetsEnabled*, *idmtools.core.interfaces.inamed_entity.INamedEntity*

Class that represents a generic experiment. This class needs to be implemented for each model type with specifics.

Parameters

- **name** – The experiment name.
- **assets** – The asset collection for assets global to this experiment.

suite_id: <module 'uuid' from '/home/docs/.pyenv/versions/3.7.3/lib/python3.7/uuid.py
Suite ID

item_type: *idmtools.core.enums.ItemType* = 2
Item Item(always an experiment)

task_type: str = 'idmtools.entities.command_task.CommandTask'
Task Type(defaults to command)

platform_requirements: Set[PlatformRequirements]
List of Requirements for the task that a platform must meet to be able to run

frozen: bool = False
Is the Experiment Frozen

gather_common_assets_from_task: bool = None
Determines if we should gather assets from a the first task. Only use when not using TemplatedSimulations

property suite

display()

pre_creation() → None
Experiment pre_creation callback

Returns:

property done
Return if an experiment has finished executing

Returns True if all simulations have ran, False otherwise

property succeeded
Return if an experiment has succeeded. An experiment is succeeded when all simulations have succeeded

Returns True if all simulations have succeeded, False otherwise

property simulations
Simulation in this experiment

property simulation_count
Return the total simulations Returns:

refresh_simulations() → NoReturn
Refresh the simulations from the platform

Returns:

refresh_simulations_status()

pre_getstate()
Return default values for `pickle_ignore_fields()`. Call before pickling.

gather_assets() → NoReturn
Function called at runtime to gather all assets in the collection.

classmethod from_task(*task*, *name: str = None*, *tags: Dict[str, Any] = None*, *assets: idmtools.assets.asset_collection.AssetCollection = None*, *gather_common_assets_from_task: bool = True*) → *idmtools.entities.experiment.Experiment*
Creates an Experiment with one Simulation from a task

Parameters

- **task** – Task to use
- **assets** – Asset collection to use for common tasks. Defaults to gather assets from task
- **name** – Name of experiment
- **tags** –
- **gather_common_assets_from_task** – Whether we should attempt to gather assets from the Task object for the experiment. With large amounts of tasks, this can be expensive as we loop through all

Returns:

classmethod from_builder(*builders: Union[*idmtools.builders.simulation_builder.SimulationBuilder*, List[*idmtools.builders.simulation_builder.SimulationBuilder*]]*, *base_task:* idmtools.entities.itask.ITask, *name: str = None*, *assets:* idmtools.assets.asset_collection.AssetCollection *= None*, *tags: Dict[str, Any] = None*) → *idmtools.entities.experiment.Experiment*

Creates an experiment from a SimulationBuilder object(or list of builders

Parameters

- **builders** – List of builder to create experiment from
- **base_task** – Base task to use as template
- **name** – Experiment name
- **assets** – Experiment level assets
- **tags** – Experiment tags

Returns Experiment object from the builders

classmethod from_template(*template:* idmtools.entities.templated_simulation.TemplatedSimulations, *name: str = None*, *assets:* idmtools.assets.asset_collection.AssetCollection *= None*, *tags: Dict[str, Any] = None*) → *idmtools.entities.experiment.Experiment*

Creates an Experiment from a TemplatedSimulation object

Parameters

- **template** – TemplatedSimulation object
- **name** – Experiment name
- **assets** – Experiment level assets
- **tags** – Tags

Returns Experiment object from the TemplatedSimulation object

list_static_assets(*children: bool = False*, *platform: IPlatform = None*, ***kwargs*) → List[*idmtools.assets.asset.Asset*]

List assets that have been uploaded to a server already

Parameters

- **children** – When set to true, simulation assets will be loaded as well
- **platform** – Optional platform to load assets list from
- ****kwargs** –

Returns List of assets

run(*wait_until_done: bool = False*, *platform: IPlatform = None*, ***run_opts*) → NoReturn
Runs an experiment on a platform

Parameters

- **wait_until_done** – Whether we should wait on experiment to finish running as well. Defaults to False
- **platform** – Platform object to use. If not specified, we first check object for platform object then the current context
- ****run_opts** – Options to pass to the platform

Returns None

wait(*timeout: int = None*, *refresh_interval=None*, *platform: IPlatform = None*)
Wait on an experiment to finish running

Parameters

- **timeout** – Timeout to wait
- **refresh_interval** – How often to refresh object
- **platform** – Platform. If not specified, we try to determine this from context

Returns:

to_dict()

classmethod from_id(*item_id: Union[str, uuid.UUID]*, *platform: IPlatform = None*, ***kwargs*)
→ Experiment
Helper function to provide better intellisense to end users

Parameters

- **item_id** – Item id to load
- **platform** – Optional platform. Fallbacks to context
- ****kwargs** – Optional arguments to be passed on to the platform

Returns:

print(*verbose: bool = False*)
Print summary of experiment :param verbose: Verbose printing

Returns:

class idmtools.entities.experiment.**ExperimentSpecification**
Bases: *idmtools.registry.experiment_specification.ExperimentPluginSpecification*

get_description() → str
Get a brief description of the plugin and its functionality.

Returns The plugin description.

get(*configuration: dict*) → *idmtools.entities.experiment.Experiment*
Experiment is going

get_type() → Type[*idmtools.entities.experiment.Experiment*]

idmtools.entities.generic_workitem module

class idmtools.entities.generic_workitem.**GenericWorkItem**(*_uid: uuid.UUID = None, platform_id: uuid.UUID = None, _platform: IPlatform = None, parent_id: uuid.UUID = None, _parent: IEntity = None, status: idmtools.core.enums.EntityStatus = None, tags: Dict[str, Any] = <factory>, _platform_object: Any = None, name: str = None, assets: idmtools.assets.asset_collection.AssetCollection = <factory>, item_name: str = 'Idm WorkItem Test', asset_collection_id: uuid.UUID = None, asset_files: idmtools.assets.file_list.FileList = None, user_files: idmtools.assets.file_list.FileList = None, related_experiments: list = None, related_simulations: list = None, related_suites: list = None, related_work_items: list = None, related_asset_collections: list = None, work_item_type: str = None*)

Bases: *idmtools.entities.iworkflow_item.IWorkflowItem*

Idm GenericWorkItem

tags

idmtools.entities.ianalyzer module

class idmtools.entities.ianalyzer.**IAnalyzer**(*uid=None, working_dir: Optional[str] = None, parse: bool = True, filenames: Optional[List[str]] = None*)

Bases: object

An abstract base class carrying the lowest level analyzer interfaces called by ExperimentManager.

initialize() → NoReturn

Call once after the analyzer has been added to the AnalyzeManager.

Add everything depending on the working directory or unique ID here instead of in __init__.

per_group(*items: List[*idmtools.core.interfaces.iitem.IItem*]*) → NoReturn
Call once before running the apply on the items.

Parameters **items** – Objects with attributes of type `ItemId`. IDs of one or more higher-level hierarchical objects can be obtained from these IDs in order to perform tasks with them.

Returns None

filter(*item: Union[*idmtools.entities.iworkflow_item.IWorkflowItem, idmtools.entities.simulation.Simulation*]*) → bool
Decide whether the analyzer should process a simulation.

Parameters **item** – An `IItem` to be considered for processing with this analyzer.

Returns A Boolean indicating whether simulation should be analyzed by this analyzer.

abstract map(*data: Dict[str, Any]*, *item: Union[*idmtools.entities.iworkflow_item.IWorkflowItem, idmtools.entities.simulation.Simulation*]*) → Any
In parallel for each simulation, consume raw data from filenames and emit selected data.

Parameters

- **data** – A dictionary associating filename with content for simulation data.
- **item** – `IItem` object that the passed data is associated with.

Returns Selected data for the given item.

abstract reduce(*all_data: Dict[Union[*idmtools.entities.iworkflow_item.IWorkflowItem, idmtools.entities.simulation.Simulation*], Any]*) → Any
Combine the *map()* data for a set of items into an aggregate result.

Parameters **all_data** – A dictionary with entries for the item ID and selected data.

destroy() → NoReturn
Call after the analysis is done.

class idmtools.entities.ianalyzer.**BaseAnalyzer**(*uid=None*, *working_dir: Optional[str] = None*, *parse: bool = True*, *filenames: Optional[List[str]] = None*)
Bases: *idmtools.entities.ianalyzer.IAnalyzer*

idmtools.entities.iplatform module

class idmtools.entities.iplatform.**IPlatform**(**args*, ***kwargs*)
Bases: *idmtools.core.interfaces.iitem.IItem*, *idmtools.core.cache_enabled.CacheEnabled*

Interface defining a platform. A platform needs to implement basic operation such as:

- Creating experiment
- Creating simulation
- Commissioning
- File handling

platform_type_map: Dict[Type, *idmtools.core.enums.ItemType*] = None

supported_types: Set[ItemType]

static get_caller()
Trace the stack and find the caller.

Returns The direct caller.

validate_inputs_types() → NoReturn
Validate user inputs and case attributes with the correct data types.

Returns None

get_item(*item_id: Union[str, uuid.UUID]*, *item_type:* idmtools.core.enums.ItemType = *None*, *force: bool = False*, *raw: bool = False*, ***kwargs*) → Any
Retrieve an object from the platform. This function is cached; force allows you to force the refresh of the cache. If no **object_type** is passed, the function will try all the types (experiment, suite, simulation).

Parameters

- **item_id** – The ID of the object to retrieve.
- **item_type** – The type of the object to be retrieved.
- **force** – If True, force the object fetching from the platform.
- **raw** – Return either an idmtools object or a platform object.

Returns The object found on the platform or None.

Raises **ValueError** – If the item type is not supported

get_children(*item_id: uuid.UUID*, *item_type:* idmtools.core.enums.ItemType, *force: bool = False*, *raw: bool = False*, *item: Any = None*, ***kwargs*) → Any
Retrieve the children of a given object.

Parameters

- **item_id** – The ID of the object for which we want the children.
- **force** – If True, force the object fetching from the platform.
- **raw** – Return either an idmtools object or a platform object.
- **item_type** – Pass the type of the object for quicker retrieval.
- **item** – optional platform or idm item to use instead of loading

Returns The children of the object or None.

get_children_by_object(*parent:* idmtools.core.interfaces.ientity.IEntity) → List[*idmtools.core.interfaces.ientity.IEntity*]
Returns a list of children for an entity

Parameters **parent** – Parent object

Returns List of children

get_parent_by_object(*child:* idmtools.core.interfaces.ientity.IEntity) → *idmtools.core.interfaces.ientity.IEntity*
Parent of object

Parameters **child** – Child object to find parent for

Returns Returns parent object

get_parent(*item_id: uuid.UUID*, *item_type:* idmtools.core.enums.ItemType = *None*, *force: bool = False*, *raw: bool = False*, ***kwargs*)
Retrieve the parent of a given object.

Parameters

- **item_id** – The ID of the object for which we want the parent.
- **force** – If True, force the object fetching from the platform.

- **raw** – Return either an idmtools object or a platform object.
- **item_type** – Pass the type of the object for quicker retrieval.

Returns The parent of the object or None.

get_cache_key(*force*, *item_id*, *item_type*, *kwargs*, *raw*, *prefix='p'*)

create_items(*items: Union[List[*idmtools.core.interfaces.ientity.IEntity*]*, idmtools.core.interfaces.ientity.IEntity*]*) → List[*idmtools.core.interfaces.ientity.IEntity*]

Create items (simulations, experiments, or suites) on the platform. The function will batch the items based on type and call the self._create_batch for creation :param items: The list of items to create.

Returns List of item IDs created.

run_items(*items: Union[*idmtools.core.interfaces.ientity.IEntity*, List[*idmtools.core.interfaces.ientity.IEntity*]]*, ***kwargs*)

Run items on the platform. :param items:

Returns:

flatten_item(*item:* idmtools.core.interfaces.ientity.IEntity) → List[*idmtools.core.interfaces.ientity.IEntity*]

Flatten an item: resolve the children until getting to the leaves. For example, for an experiment, will return all the simulations. For a suite, will return all the simulations contained in the suites experiments.

Parameters **item** – Which item to flatten

Returns List of leaves

refresh_status(*item:* idmtools.core.interfaces.ientity.IEntity) → NoReturn

Populate the platform item and specified item with its status.

Parameters **item** – The item to check status for.

get_files(*item:* idmtools.core.interfaces.ientity.IEntity, *files: Union[Set[str], List[str]]*, *output: str = None*) → Union[Dict[str, Dict[str, bytearray]], Dict[str, bytearray]]

Get files for a platform entity

Parameters

- **item** – Item to fetch files for
- **files** – List of file names to get
- **output** – save files to

Returns

For simulations, this returns a dictionary with filename as key and values being binary data from file or a dict.

For experiments, this returns a dictionary with key as sim id and then the values as a dict of the simulations described above

get_files_by_id(*item_id: uuid.UUID*, *item_type:* idmtools.core.enums.ItemType, *files: Union[Set[str], List[str]]*, *output: str = None*) → Union[Dict[str, Dict[str, bytearray]], Dict[str, bytearray]]

Get files by item id (UUID) :param item_id: COMPS Item, say, Simulation Id or WorkItem Id :param item_type: Item Type :param files: List of files to retrieve :param output: save files to

Returns: dict with key/value: file_name/file_content

are_requirements_met(*requirements: Set[*idmtools.entities.platform_requirements.PlatformRequirements*]*) → bool

Does the platform support the list of requirements

Parameters **requirements** – Requirements

Returns True if all the requirements are supported

is_task_supported(*task:* idmtools.entities.itask.ITask) → bool
Is a task supported on this platform. This depends on the task properly setting its requirements. See *idmtools.entities.itask.ITask.platform_requirements* and *idmtools.entities.platform_requirements.PlatformRequirements*

Parameters **task** – Task to check support of

Returns True if the task is supported, False otherwise.

wait_till_done(*item: Union[*idmtools.entities.experiment.Experiment, idmtools.entities.iworkflow_item.IWorkflowItem, idmtools.entities.suite.Suite*]*, *timeout: int = 86400*, *refresh_interval: int = 5*, *progress: bool = True*)
Wait for the experiment to be done.

Parameters

- **item** – Experiment/Workitem to wait on
- **refresh_interval** – How long to wait between polling.
- **timeout** – How long to wait before failing.
- **progress** – Should we display progress

See also:

idmtools.entities.iplatform.IPlatform.wait_till_done_progress()
idmtools.entities.iplatform.IPlatform.__wait_until_done_progress_callback()
idmtools.entities.iplatform.IPlatform.__wait_till_callback()

wait_till_done_progress(*item: Union[*idmtools.entities.experiment.Experiment, idmtools.entities.iworkflow_item.IWorkflowItem, idmtools.entities.suite.Suite*]*, *timeout: int = 86400*, *refresh_interval: int = 5*)
Wait on an item to complete with progress bar

Parameters

- **item** – Item to monitor
- **timeout** – Timeout on waiting
- **refresh_interval** – How often to refresh

Returns None

See also:

idmtools.entities.iplatform.IPlatform.__wait_until_done_progress_callback()
idmtools.entities.iplatform.IPlatform.wait_till_done() idmtools.entities.iplatform.IPlatform.__wait_till_callback()

get_related_items(*item:* idmtools.entities.iworkflow_item.IWorkflowItem, *relation_type:* idmtools.entities.relation_type.RelationType) → Dict[str, Dict[str, str]]
Retrieve all related objects :param item: SSMTWorkItem :param relation_type: Depends or Create

Returns: dict with key the object type

idmtools.entities.itask module

class idmtools.entities.itask.**ITask**(*command: Union[str, idmtools.entities.command_line.CommandLine] = None, platform_requirements: Set[idmtools.entities.platform_requirements.PlatformRequirements] = <factory>, _ITask__pre_creation_hooks: List[Callable[[Union[ForwardRef('Simulation'), ForwardRef('IWorkflowItem')]], NoReturn]] = <factory>, _ITask__post_creation_hooks: List[Callable[[Union[ForwardRef('Simulation'), ForwardRef('IWorkflowItem')]], NoReturn]] = <factory>, common_assets: idmtools.assets.asset_collection.AssetCollection = <factory>, transient_assets: idmtools.assets.asset_collection.AssetCollection = <factory>, _task_log: logging.Logger = <factory>*)

Bases: object

command: Union[str, *idmtools.entities.command_line.CommandLine*] = None
The Command to run

platform_requirements: Set[PlatformRequirements]

common_assets: AssetCollection
Common(Experiment-level) assets

transient_assets: AssetCollection
Transient(Simulation-level) assets

property metadata_fields
Collect all metadata fields

Returns: set of metadata filed names

add_pre_creation_hook(*hook: Callable[[Union[Simulation, IWorkflowItem]], NoReturn]*)
Called before a simulation is created on a platform. Each hook receives either a Simulation or WorkflowTask

Parameters **hook** – Function to call on event

Returns None

add_post_creation_hook(*hook: Callable[[Union[Simulation, IWorkflowItem]], NoReturn]*)
Called after a simulation has been created on a platform. Each hook receives either a Simulation or WorkflowTask

Parameters **hook** – Function to call on event

Returns:

add_platform_requirement(*requirement: Union[*idmtools.entities.platform_requirements.PlatformRequirements, *str]*) → NoReturn
Adds a platform requirements to a task :param requirement: Requirement to add task

Returns None

pre_creation(*parent: Union[Simulation, IWorkflowItem]*)
Optional Hook called at the time of creation of task. Can be used to setup simulation and experiment level hooks :param parent:

Returns:

post_creation(*parent: Union[Simulation, IWorkflowItem]*)
Optional Hook called at the after creation task. Can be used to setup simulation and experiment level hooks :param parent:

Returns:

abstract gather_common_assets() → *idmtools.assets.asset_collection.AssetCollection*
Function called at runtime to gather all assets in the collection.

abstract gather_transient_assets() → *idmtools.assets.asset_collection.AssetCollection*
Function called at runtime to gather all assets in the collection

gather_all_assets() → *idmtools.assets.asset_collection.AssetCollection*
Collect all common and transient assets

Returns: new AssetCollection

copy_simulation(*base_simulation: Simulation*) → Simulation

Called when copying a simulation for batching. Override you your task has specific concerns when copying simulations.

reload_from_simulation(*simulation: Simulation*)
Optional hook that is called when loading simulations from a platform

to_simulation()
Convert task to simulation

Returns: new simulation

pre_getstate()
Return default values for `pickle_ignore_fields()`. Call before pickling.

Returns: dict

post_setstate()

property pickle_ignore_fields

to_dict() → Dict
Select metadata fields and make a new dict

Returns: dict

idmtools.entities.iworkflow_item module

class idmtools.entities.iworkflow_item.**IWorkflowItem**(*_uid: uuid.UUID = None, platform_id: uuid.UUID = None, _platform: IPlatform = None, parent_id: uuid.UUID = None, _parent: IEntity = None, status: idmtools.core.enums.EntityStatus = None, tags: Dict[str, Any] = <factory>, _platform_object: Any = None, name: str = None, assets: idmtools.assets.asset_collection.AssetCollection = <factory>, item_name: str = 'Idm WorkItem Test', asset_collection_id: uuid.UUID = None, asset_files: idmtools.assets.file_list.FileList = None, user_files: idmtools.assets.file_list.FileList = None, related_experiments: list = None, related_simulations: list = None, related_suites: list = None, related_work_items: list = None, related_asset_collections: list = None, work_item_type: str = None*)

Bases: *idmtools.core.interfaces.iassets_enabled.IAssetsEnabled*, *idmtools.core.interfaces.inamed_entity.INamedEntity*, abc.ABC

Interface of idmtools work item

item_name: str = 'Idm WorkItem Test'

tags: Dict[str, Any]

asset_collection_id: uuid.UUID = None

asset_files: *idmtools.assets.file_list.FileList* = None

user_files: *idmtools.assets.file_list.FileList* = None

related_experiments: list = None

related_simulations: list = None

related_suites: list = None

related_work_items: list = None

related_asset_collections: list = None

work_item_type: str = None

item_type: ItemType = 4

gather_assets() → NoReturn

Function called at runtime to gather all assets in the collection.

add_file(*af*)
Methods used to add new file :param af: file to add

Returns: None

clear_user_files()
Clear all existing user files

Returns: None

pre_creation() → None
Called before the actual creation of the entity.

run(*wait_on_done: bool = False*, *wait_on_done_progress: bool = True*, *platform: IPlatform = None*)
Run the item on specified platform

Parameters

- **wait_on_done** – Should we wait on item to finish running? Default is false
- **wait_on_done_progress** – When waiting, should we try to show progress
- **platform** – optional platform object

Returns:

wait(*wait_on_done_progress: bool = True*, *platform: IPlatform = None*)
Wait on item to finish

Parameters

- **wait_on_done_progress** – Should we show progress as we wait?
- **platform** – Optional platform object

Returns:

to_dict() → Dict

idmtools.entities.platform_requirements module

class idmtools.entities.platform_requirements.**PlatformRequirements**(*value*)
Bases: enum.Enum

Defines possible requirements a task could need from a platform

SHELL = 'shell'

NativeBinary = 'NativeBinary'

LINUX = 'Linux'

WINDOWS = 'windows'

GPU = 'gpu'

PYTHON = 'python'

DOCKER = 'docker'

SINGULARITY = 'singularity'

idmtools.entities.relation_type module

class idmtools.entities.relation_type.**RelationType**(*value*)
Bases: enum.Enum

An enumeration representing the type of relationship for related entities

DependsOn = 0

Created = 1

idmtools.entities.simulation module

class idmtools.entities.simulation.**Simulation**(*_uid: uuid.UUID = None*, *platform_id: uuid.UUID = None*, *_platform: IPlatform = None*, *parent_id: uuid.UUID = None*, *_parent: IEntity = None*, *status: idmtools.core.enums.EntityStatus = None*, *tags: Dict[str, Any] = <factory>*, *item_type: idmtools.core.enums.ItemType = <ItemType.SIMULATION: 3>*, *_platform_object: Any = None*, *name: str = None*, *assets: idmtools.assets.asset_collection.AssetCollection = <factory>*, *task: ITask = None*, *pre_creation_hooks: List[Callable[[], NoReturn]] = <factory>*, *_Simulation__assets_gathered: bool = False*)
Bases: *idmtools.core.interfaces.iassets_enabled.IAssetsEnabled*, *idmtools.core.interfaces.inamed_entity.INamedEntity*

Class that represents a generic simulation. This class needs to be implemented for each model type with specifics.

task: ITask = None
Task representing the configuration of the command to be executed

item_type: *idmtools.core.enums.ItemType* = 3
Item Type. Should not be changed from Simulation

pre_creation_hooks: List[Callable[], NoReturn]]
List of hooks that we can modify to add additional behaviour before creation of simulations

property experiment

pre_creation()
Called before the actual creation of the entity.

post_creation() → None
Called after the actual creation of the entity.

pre_getstate()
Return default values for pickle_ignore_fields(). Call before pickling.

gather_assets()
Gather all the assets for the simulation.

classmethod from_task(*task: ITask*, *tags: Dict[str, Any] = None*, *asset_collection: idmtools.assets.asset_collection.AssetCollection = None*)
Create a simulation from a task

Parameters

- **task** – Task to create from
- **tags** – Tags to create on the simulation
- **asset_collection** – Simulation Assets

Returns:

list_static_assets(*platform: IPlatform = None*, ***kwargs*) → List[*idmtools.assets.asset.Asset*]
List assets that have been uploaded to a server already

Parameters

- **children** – When set to true, simulation assets will be loaded as well
- **platform** – Optional platform to load assets list from
- ****kwargs** –

Returns List of assets

to_dict() → Dict
Do a lightweight conversation to json :returns: Dict representing json of object

idmtools.entities.suite module

class idmtools.entities.suite.**Suite**(*_uid: uuid.UUID = None*, *platform_id: uuid.UUID = None*, *_platform: IPlatform = None*, *parent_id: uuid.UUID = None*, *_parent: IEntity = None*, *status: idmtools.core.enums.EntityStatus = None*, *tags: Dict[str, Any] = <factory>*, *_platform_object: Any = None*, *name: str = None*, *experiments: idmtools.core.interfaces.entity_container.EntityContainer = <factory>*, *description: str = None*)
Bases: *idmtools.core.interfaces.inamed_entity.INamedEntity*, abc.ABC

Class that represents a generic suite (a collection of experiments).

Parameters **experiments** – The child items of this suite.

experiments: EntityContainer

item_type: *idmtools.core.enums.ItemType* = 1

description: str = None

add_experiment(*experiment: Experiment*) → NoReturn
Add an experiment to the suite :param experiment: the experiment to be linked to suite

display()

pre_creation()
Called before the actual creation of the entity.

post_creation()
Called after the actual creation of the entity.

property done
Return if an suite has finished executing

Returns True if all experiments have ran, False otherwise

property succeeded
Return if an suite has succeeded. An suite is succeeded when all experiments have succeeded

Returns True if all experiments have succeeded, False otherwise

run(*wait_until_done: bool = False*, *platform: IPlatform = None*, ***run_opts*) → NoReturn
Runs an experiment on a platform

Parameters

- **wait_until_done** – Whether we should wait on experiment to finish running as well. Defaults to False
- **platform** – Platform object to use. If not specified, we first check object for platform object then the
- **context** (*current*) –
- ****run_opts** – Options to pass to the platform

Returns None

wait(*timeout: int = None*, *refresh_interval=None*, *platform: IPlatform = None*)
Wait on an experiment to finish running

Parameters

- **timeout** – Timeout to wait
- **refresh_interval** – How often to refresh object
- **platform** – Platform. If not specified, we try to determine this from context

Returns:

to_dict() → Dict

idmtools.entities.task_proxy module

class idmtools.entities.task_proxy.**TaskProxy**(*command: Union[str, idmtools.entities.command_line.CommandLine] = None*, *is_docker: bool = False*, *is_gpu: bool = False*)

Bases: object

This class is used to reduce the memory footprint of tasks after a simulation has been provisioned

command: Union[str, *idmtools.entities.command_line.CommandLine*] = None

is_docker: bool = False

is_gpu: bool = False

static from_task(*task: ITask*)

idmtools.entities.templated_simulation module

idmtools.entities.templated_simulation.**simulation_generator**(*builders*, *new_sim_func*, *additional_sims=None*, *batch_size=10*)

class idmtools.entities.templated_simulation.**TemplatedSimulations**(*builders: Set[idmtools.builders.simulation_bu = <factory>*, *base_simulation: idmtools.entities.simulation.Simulation = None*, *base_task: idmtools.entities.itask.ITask = None*, *parent: Experiment = None*, *tags: dataclasses.InitVar = <property object>*, *_TemplatedSimulations__extra_simulations: List[idmtools.entities.simulation.Sim = <factory>*)

Bases: object

Class for building templated simulations and commonly used with SimulationBuilder class.

Examples

Add tags to all simulations via base task:

```
ts = TemplatedSimulations(base_task=task)
ts.tags = {'a': 'test', 'b': 9}
ts.add_builder(builder)
```

Add tags to a specific simulation:

```
experiment =  Experiment.from_builder(builder, task, name=expname)
experiment.simulations = list(experiment.simulations)
experiment.simulations[2].tags['test']=123
```

builders: Set[SimulationBuilder]

base_simulation: *idmtools.entities.simulation.Simulation* = None

base_task: *idmtools.entities.itask.ITask* = None

parent: Experiment = None

property builder
For backward-compatibility purposes.

Returns The last TExperimentBuilder.

add_builder(*builder:* idmtools.builders.simulation_builder.SimulationBuilder) → None
Add builder to builder collection.

Parameters **builder** – A builder to be added.

Returns None

property pickle_ignore_fields

display()

simulations() → Generator[idmtools.entities.simulation.Simulation, None, None]

add_simulation(*simulation:* idmtools.entities.simulation.Simulation)
Add a simulation that was built outside template engine to template generator. This is useful we you can build most simulations through a template but need a some that cannot. This is especially true for large simulation sets

Parameters **simulation** – Simulation to add

Returns:

new_simulation()
Return a new simulation object. The simulation will be copied from the base simulation of the experiment.

Returns The created simulation.

property tags

classmethod from_task(*task:* idmtools.entities.itask.ITask, *tags: Dict[str, Any] = None*) → *idmtools.entities.templated_simulation.TemplatedSimulations*

Module contents

idmtools.registry package

Submodules

idmtools.registry.experiment_specification module

class idmtools.registry.experiment_specification.**ExperimentPluginSpecification**
Bases: *idmtools.registry.plugin_specification.PluginSpecification*, abc.ABC

classmethod get_name(*strip_all: bool = True*) → str
Get name of plugin. By default we remove the PlatformSpecification portion. :param strip_all: When true, ExperimentPluginSpecification and ExperimentPluginSpec is stripped from name. :param When false only Specification and Spec is Stripped:

Returns:

get(*configuration: dict*) → Experiment
Return a new model using the passed in configuration.

Parameters **configuration** – The INI configuration file to use.

Returns The new model.

get_type() → Type[Experiment]

class idmtools.registry.experiment_specification.**ExperimentPlugins**(*strip_all: bool = True*)

Bases: object

get_plugins() → Set[*idmtools.registry.experiment_specification.ExperimentPluginSpecification*]

get_plugin_map() → Dict[str, *idmtools.registry.experiment_specification.ExperimentPluginSpecification*]

idmtools.registry.master_plugin_registry module

class idmtools.registry.master_plugin_registry.**MasterPluginRegistry**

Bases: object

get_plugin_map() → Dict[str, *idmtools.registry.plugin_specification.PluginSpecification*]

get_plugins() → Set[*idmtools.registry.plugin_specification.PluginSpecification*]

idmtools.registry.platform_specification module

class idmtools.registry.platform_specification.**PlatformSpecification**

Bases: *idmtools.registry.plugin_specification.PluginSpecification*, abc.ABC

classmethod get_name(*strip_all: bool = True*) → str

Get name of plugin. By default we remove the PlatformSpecification portion. :param strip_all: When true, PlatformSpecification is stripped from name. When false only Specification is Stripped

Returns:

example_configuration()

Example configuration for the platform. This is useful in help or error messages.

Returns:

get(*configuration: dict*) → IPlatform

Return a new platform using the passed in configuration.

Parameters **configuration** – The INI configuration file to use.

Returns The new platform.

get_type() → Type[IPlatform]

class idmtools.registry.platform_specification.**PlatformPlugins**(*strip_all: bool = True*)

Bases: object

get_plugins() → Set[*idmtools.registry.platform_specification.PlatformSpecification*]

get_plugin_map() → Dict[str, *idmtools.registry.platform_specification.PlatformSpecification*]

idmtools.registry.plugin_specification module

class idmtools.registry.plugin_specification.**ProjectTemplate**(*name: str, url: Union[str, List[str]], description: str = None, info: str = None*)

Bases: object

name: str

url: Union[str, List[str]]

description: str = None

info: str = None

static read_templates_from_json_stream(*s*) → List[*idmtools.registry.plugin_specification.ProjectTemplate*]

Read Project Template from stream

Parameters **s** – Stream where json data resides

Returns:

class idmtools.registry.plugin_specification.**PluginSpecification**

Bases: object

Base class for all plugins.

classmethod get_name(*strip_all: bool = True*) → str

Get the name of the plugin. Although it can be overridden, the best practice is to use the class name as the plugin name.

Returns The name of the plugin as a string.

get_description() → str

Get a brief description of the plugin and its functionality.

Returns The plugin description.

get_project_templates() → List[*idmtools.registry.plugin_specification.ProjectTemplate*]

Returns a list of project templates related to the a plugin Returns:

get_example_urls() → List[str]

Returns a list of URLs that a series of Examples for plugin can be downloaded from

Returns List of urls

get_help_urls() → Dict[str, str]

Returns a dictionary of topics and links to help

Returns:

static get_version_url(*version: str, extra: str = None, repo_base_url: str = 'https://github.com/InstituteforDiseaseModeling/idmtools/tree/', nightly_branch: str = 'dev'*)

Build a url using version

Here we assume the tag will exist for that specific version :param version: Version to look up. If it contains nightly, we default to nightly_branch :param extra: Extra parts of url pass base :param repo_base_url: Optional url :param nightly_branch: default to dev

Returns URL for item

idmtools.registry.task_specification module

class idmtools.registry.task_specification.**TaskSpecification**
Bases: *idmtools.registry.plugin_specification.PluginSpecification*, abc.ABC

classmethod get_name(*strip_all: bool = True*) → str
Get name of plugin. By default we remove the PlatformSpecification portion. :param strip_all: When true, TaskSpecification and TaskSpec is stripped from name. When false only :param Specification and Spec is Stripped:

Returns:

get(*configuration: dict*) → *idmtools.entities.itask.ITask*
Return a new model using the passed in configuration.

Parameters **configuration** – The INI configuration file to use.

Returns The new model.

get_type() → Type[*idmtools.entities.itask.ITask*]

class idmtools.registry.task_specification.**TaskPlugins**(*strip_all: bool = True*)
Bases: object

get_plugins() → Set[*idmtools.registry.task_specification.TaskSpecification*]

get_plugin_map() → Dict[str, *idmtools.registry.task_specification.TaskSpecification*]

idmtools.registry.utils module

idmtools.registry.utils.**is_a_plugin_of_type**(*value*, *plugin_specification: Type[*idmtools.registry.plugin_specification.PluginSpecification*]*) → bool
Determine if a value of a plugin specification is of type *PluginSpecification*.

Parameters

- **value** – The value to inspect.
- **plugin_specification** – Plugin specification to check against.

Returns A Boolean indicating True if the plugin is of a subclass of *PluginSpecification*, else False.

idmtools.registry.utils.**load_plugin_map**(*entrypoint: str*, *spec_type: Type[*idmtools.registry.plugin_specification.PluginSpecification*]*, *strip_all: bool = True*) → Dict[str, Type[*idmtools.registry.plugin_specification.PluginSpecification*]]
Load plugins from entry point with the indicated type of specification into a map.

> **Warning:** This could cause name collisions if plugins of the same name are installed.

Parameters

- **entrypoint** – The name of the entry point.
- **spec_type** – The type of plugin specification.
- **strip_all** – Pass through for get_name from Plugins. Changes names in plugin registries

Returns Returns a dictionary of name and *PluginSpecification*.

Return type (Dict[str, Type[*PluginSpecification*]])

idmtools.registry.utils.**plugins_loader**(*entry_points_name: str*, *plugin_specification: Type[*idmtools.registry.plugin_specification.PluginSpecification*]*) → Set[*idmtools.registry.plugin_specification.PluginSpecification*]

Loads all the plugins of type `PluginSpecification` from entry point name. idmtools also supports loading plugins through a list of strings representing the paths to modules containing plugins.

Parameters

- **entry_points_name** – Entry point name for plugins.
- **plugin_specification** – Plugin specification to load.

Returns All the plugins of the type indicated.

Return type (Set[*PluginSpecification*])

idmtools.registry.utils.**discover_plugins_from**(*library: Any*, *plugin_specification: Type[*idmtools.registry.plugin_specification.PluginSpecification*]*) → List[Type[*idmtools.registry.plugin_specification.PluginSpecific*

Search a library object for plugins of type `PluginSpecification`.

Currently it detects module and classes. In the future support for strings will be added.

Parameters

- **library** – Library object to discover plugins from.
- **plugin_specification** – Specification to search for.

Returns List of plugins.

Return type List[Type[*PluginSpecification*]]

Module contents

idmtools.services package

Submodules

idmtools.services.ipersistance_service module

class idmtools.services.ipersistance_service.**IPersistenceService**

Bases: object

cache_directory = '/home/docs/checkouts/readthedocs.org/user_builds/institute-for-dise

cache_name = None

classmethod retrieve(*uid*)

classmethod save(*obj*)

classmethod delete(*uid*)

classmethod clear()

classmethod list()

classmethod length()

idmtools.services.platforms module

class idmtools.services.platforms.**PlatformPersistService**
Bases: *idmtools.services.ipersistance_service.IPersistenceService*

cache_name = 'platforms'

Module contents

idmtools.utils package

Subpackages

idmtools.utils.display package

Submodules

idmtools.utils.display.displays module

class idmtools.utils.display.displays.**IDisplaySetting**(*header: str = None*, *field: str = None*)
Bases: object

Base class for a display setting. The child class needs to implement the *display()* method.

Includes:

- header: Optional header for the display.
- field: If specified, the *get_object()* will call getattr for this field on the object.

get_object(*obj: Any*) → Any

abstract display(*obj: Any*) → str
Display the object. Note that the attribute (identified by self.field) should be handled with *get_object()*.

Parameters **obj** – The object to consider for display.

Returns A string representing what to show.

class idmtools.utils.display.displays.**StringDisplaySetting**(*header: str = None*, *field: str = None*)
Bases: *idmtools.utils.display.displays.IDisplaySetting*

Class that displays the object as string.

display(*obj*)
Display the object. Note that the attribute (identified by self.field) should be handled with get_object().

Parameters **obj** – The object to consider for display.

Returns A string representing what to show.

class idmtools.utils.display.displays.**DictDisplaySetting**(*header: str = None*, *field: str = None*, *max_items: int = 10*, *flat: bool = False*)

Bases: *idmtools.utils.display.displays.IDisplaySetting*

Class that displays a dictionary.

display(*obj: Any*) → str

Display the object. Note that the attribute (identified by self.field) should be handled with `get_object()`.

Parameters **obj** – The object to consider for display.

Returns A string representing what to show.

class idmtools.utils.display.displays.**TableDisplay**(*columns*, *max_rows=5*, *field=None*)

Bases: *idmtools.utils.display.displays.IDisplaySetting*

Class that displays the object as a table.

display(*obj*)

Display the object. Note that the attribute (identified by self.field) should be handled with `get_object()`.

Parameters **obj** – The object to consider for display.

Returns A string representing what to show.

idmtools.utils.display.settings module

Module contents

idmtools.utils.display.**display**(*obj*, *settings*)

idmtools.utils.filters package

Submodules

idmtools.utils.filters.asset_filters module

This module contains all the default filters for the assets.

A filter function needs to take only one argument: an asset. It returns True/False indicating whether to add or filter out the asset.

You can notice functions taking more than only an asset. To use those functions, use must create a partial before adding it to a filters list. For example:

```
python
fname = partial(file_name_is, filenames=["a.txt", "b.txt"])
AssetCollection.from_directory(... filters=[fname], ...)
```

idmtools.utils.filters.asset_filters.**default_asset_file_filter**(*asset: TAsset*) → bool

Default filter to leave out Python caching. This filter is used in the creation of *AssetCollection*, regardless of user filters.

idmtools.utils.filters.asset_filters.**file_name_is**(*asset: TAsset*, *filenames: List[str]*) → bool

Restrict filtering to assets with the indicated filenames.

Parameters

- **asset** – The asset to filter.
- **filenames** – List of filenames to filter on.

idmtools.utils.filters.asset_filters.**file_extension_is**(*asset: TAsset*, *extensions: List[str]*) → bool

Restrict filtering to assets with the indicated filetypes.

Parameters

- **asset** – The asset to filter.
- **extensions** – List of extensions to filter on.

idmtools.utils.filters.asset_filters.**asset_in_directory**(*asset: TAsset*, *directories: List[str]*) → bool

Restrict filtering to assets within a given directory. This filter is not strict and simply checks if the directory portion is present in the assets absolute path.

Parameters

- **asset** – The asset to filter.
- **directories** – List of directory portions to include.

Module contents

Submodules

idmtools.utils.collections module

idmtools.utils.collections.**cut_iterable_to**(*obj: Iterable*, *to: int*) → Tuple[Union[List, Mapping], int]

Cut an iterable to a certain length.

Parameters

- **obj** – The iterable to cut.
- **to** – The number of elements to return.

Returns A list or dictionary (depending on the type of object) of elements and the remaining elements in the original list or dictionary.

class idmtools.utils.collections.**ParentIterator**(**args*, ***kwds*)

Bases: collections.abc.Iterator, typing.Generic

append(*item*)

class idmtools.utils.collections.**ResetGenerator**(**args*, ***kwds*)

Bases: collections.abc.Iterator, typing.Generic

Iterator that counts upward forever.

next_gen()

idmtools.utils.collections.**duplicate_list_of_generators**(*lst: List[Generator]*)
Copy a list of iterators using tee :param lst: List of generators

Returns Tuple with duplicate of iterators

idmtools.utils.command_line module

idmtools.utils.command_line.**suppress_output**(*stdout=True*, *stderr=True*)
Suppress any print/logging from a block of code.

Parameters

- **stdout** – If True, hide output from stdout; if False, show it.
- **stderr** – If True, hide output from stderr; if False, show it.

idmtools.utils.decorators module

class idmtools.utils.decorators.**abstractstatic**(*function*)
Bases: staticmethod

A decorator for defining a method both as static and abstract.

idmtools.utils.decorators.**optional_decorator**(*decorator: Callable, condition: Union[bool, Callable[], bool]]*)

class idmtools.utils.decorators.**SingletonDecorator**(*klass*)
Bases: object

Wraps a class in a singleton decorator.

Example

In the below example, we would print out *99* since *z* is referring to the same object as *x*:

```
class Thing:
    y = 14
Thing = SingletonDecorator(Thing)
x = Thing()
x.y = 99
z = Thing()
print(z.y)
```

class idmtools.utils.decorators.**LoadOnCallSingletonDecorator**(*klass*)
Bases: object

Additional class decorator that creates a singleton instance only when a method or attribute is accessed. This is useful for expensive tasks like loading plugin factories that should only be executed when finally needed and not on declaration.

Examples

```
import time
class ExpensiveFactory:
    def __init__():
        time.sleep(1000)
        self.items = ['a', 'b', 'c']
    def get_items():
        return self.items

ExpensiveFactory = LoadOnCallSingletonDecorator(ExpensiveFactory)
ExpensiveFactory.get_items()
```

ensure_created()

idmtools.utils.decorators.**cache_for**(*ttl=datetime.timedelta(seconds=60)*)

idmtools.utils.decorators.**optional_yaspin_load**(**yargs*, ***ykwargs*) → Callable

Adds a CLI spinner to a function if:

- yaspin package is present.
- NO_SPINNER environment variable is not defined.

Parameters

- ***yargs** – Arguments to pass to yaspin constructor.
- ****ykwargs** – Keyword arguments to pass to yaspin constructor.

Examples

```
@optional_yaspin_load(text="Loading test", color="yellow")
def test():
    time.sleep(100)
```

Returns A callable wrapper function.

class idmtools.utils.decorators.**ParallelizeDecorator**(*queue=None*, *pool_type: Optional[Type[concurrent.futures._base.Executor]] = <class 'concurrent.futures.thread.ThreadPoolExecutor'>*)

Bases: object

ParallelizeDecorator allows you to easily parallelize a group of code. A simple of example would be

Examples

```
op_queue = ParallelizeDecorator()

class Ops:
    op_queue.parallelize
    def heavy_op():
        time.sleep(10)

    def do_lots_of_heavy():
```

(continues on next page)

(continued from previous page)

```
        futures = [self.heavy_op() for i in range(100)]
        results = op_queue.get_results(futures)
```

parallelize(*func*)

join()

get_results(*futures*, *ordered=False*)

idmtools.utils.dropbox_location module

idmtools.utils.dropbox_location.**get_current_user**()

idmtools.utils.dropbox_location.**get_dropbox_location**()

idmtools.utils.entities module

idmtools.utils.entities.**get_dataclass_common_fields**(*src*, *dest*, *exclude_none: bool = True*) → Dict

Extracts fields from a dataclass source object who are also defined on destination object. Useful for situations like nested configurations of data class options

Parameters

- **src** – Source dataclass object
- **dest** – Dest dataclass object
- **exclude_none** – When true, values of None will be excluded

Returns:

idmtools.utils.entities.**as_dict**(*src*, *exclude: List[str] = None*, *exclude_private_fields: bool = True*)

Converts a dataclass to a dict while also obeys rules for exclusion :param src: :param exclude: List of fields to exclude :param exclude_private_fields: Should fields that star

Returns:

idmtools.utils.entities.**validate_user_inputs_against_dataclass**(*field_type*, *field_value*)

idmtools.utils.entities.**get_default_tags**() → Dict[str, str]

Get common default tags. Currently this is the version of idmtools Returns:

idmtools.utils.file module

idmtools.utils.file.**scan_directory**(*basedir: str*, *recursive: bool = True*) → Iterable[posix.DirEntry]

Scan a directory recursively or not.

Parameters

- **basedir** – The root directory to start from.
- **recursive** – True to search the subfolders recursively; False to stay in the root directory.

Returns An iterator yielding all the files found.

idmtools.utils.file.**file_contents_to_generator**(*filename*, *chunk_size=128*) → Generator[bytearray, None, None]

Create a generator from file contents in chunks(useful for streaming binary data and piping) :param filename: :param chunk_size:

Returns:

idmtools.utils.file_parser module

class idmtools.utils.file_parser.**FileParser**

Bases: object

classmethod parse(*filename*, *content=None*)

classmethod load_json_file(*filename*, *content*)

classmethod load_raw_file(*filename*, *content*)

classmethod load_csv_file(*filename*, *content*)

classmethod load_xlsx_file(*filename*, *content*)

classmethod load_txt_file(*filename*, *content*)

classmethod load_bin_file(*filename*, *content*)

idmtools.utils.filter_simulations module

class idmtools.utils.filter_simulations.**FilterItem**

Bases: object

static filter_item(*platform: idmtools.entities.iplatform.IPlatform*, *item: idmtools.core.interfaces.ientity.IEntity*, *skip_sims=[]*, *max_simulations: int = None*, ***kwargs*)

Filter simulations from Experiment or Suite, by default it filter status with Succeeded. If user wants to filter by other status, it also can be done, for example:

```
filter_item(platform, exp, status=EntityStatus.FAILED
```

If user wants to filter by tags, it also can be done, for example:

```
filter_item(platform, exp, tags={'Run_Number': '2'})
```

Parameters

- **platform** –
- **item** –
- **skip_sims** – list of sim ids
- **max_simulations** –
- **kwargs** – extra filters

Returns: list of simulation ids

classmethod filter_item_by_id(*platform: idmtools.entities.iplatform.IPlatform*, *item_id: uuid.UUID*, *item_type: idmtools.core.enums.ItemType = <ItemType.EXPERIMENT: 2>*, *skip_sims=[]*, *max_simulations: int = None*, ***kwargs*)

Filter simulations from Experiment or Suite :param platform: COMPSPlatform :param item_id: Experiment/Suite id :param item_type: Experiment or Suite :param skip_sims: list of sim ids :param max_simulations: #sims to be returned :param kwargs: extra filters

Returns: list of simulation ids

idmtools.utils.gitrepo module

class idmtools.utils.gitrepo.**GitRepo**(*repo_owner: str = None*, *repo_name: str = None*)

Bases: object

repo_owner: str = None

repo_name: str = None

property path

property branch

property verbose

property repo_home_url

Construct repo home url Returns: repo home url

property repo_example_url

Construct repo example url Returns: repo example url

property api_example_url

Construct api url of the examples for download Returns: api url

parse_url(*url: str*, *branch: str = None*, *update: bool = True*)

Parse url for owner, repo, branch and example path :param url: example url :param branch: user branch to replace the branch in url :param update: True/False - update repo or not

Returns: None

list_public_repos(*repo_owner: str = None*, *page: int = 1*, *raw: bool = False*)

Utility method to retrieve all public repos :param repo_owner: the owner of the repo :param page: pagination of results :param raw: bool - return rwo data or simplified list

Returns: repo list

list_repo_releases(*repo_owner: str = None*, *repo_name: str = None*, *raw: bool = False*)

Utility method to retrieve all releases of the repo :param repo_owner: the owner of the repo :param repo_name: the name of repo :param raw: bool - return raw data or simplified list

Returns: the release list of the repo

download(*path: str = ''*, *output_dir: str = './'*, *branch: str = 'master'*) → int

Download files with example url provided :param path: local file path to the repo :param output_dir: user local folder to download files to :param branch: specify branch for files download from

Returns: total file count downloaded

peep(*path: str = ''*, *branch: str = 'master'*)

Download files with example url provided :param path: local file path to the repo :param branch: specify branch for files download from

Returns: None

idmtools.utils.hashing module

Fast hash of Python objects.

class idmtools.utils.hashing.**Hasher**(*hash_name='md5'*)
Bases: pickle._Pickler

A subclass of pickler to do hashing, rather than pickling.

hash(*obj*, *return_digest=True*)

save(*obj*)

memoize(*obj*)
Disable memoization for strings so hashing happens on value and not reference.

save_set(*set_items*)

idmtools.utils.hashing.**hash_obj**(*obj*, *hash_name='md5'*)
Quick calculation of a hash to identify uniquely Python objects.

Parameters **hash_name** – The hashing algorithm to use. 'md5' is faster; 'sha1' is considered safer.

idmtools.utils.hashing.**ignore_fields_in_dataclass_on_pickle**(*item*)

idmtools.utils.hashing.**calculate_md5**(*filename: str*, *chunk_size: int = 8192*) → str
Calculate MD5

Parameters

- **filename** – Filename to caclulate md5 for
- **chunk_size** – Chunk size

Returns:

idmtools.utils.info module

idmtools.utils.info.**get_doc_base_url**()

idmtools.utils.info.**get_pip_packages_10_to_6**()
Load packages for versions 1.0 to 6 of pip.

Returns None

Raises **ImportError** – If the pip version is different.

idmtools.utils.info.**get_pip_packages_6_to_9**()
Get packages for pip versions 6 through 9.

Returns None

Raises **ImportError** – If the pip version is different.

idmtools.utils.info.**get_pip_packages_10_to_current**()
Get packages for pip versions 10 to current.

Returns None

Raises **ImportError** – If the pip version is different.

idmtools.utils.info.**get_packages_from_pip**()
Attempt to load pacakges from pip.

Returns A list of packages installed.

Return type (List[str])

idmtools.utils.info.**get_packages_list**() → List[str]
Return a list of installed packages in the current environment. Currently idmtools depends on pip for this functionality and since it is just used for troubleshooting, errors can be ignored.

Returns A list of packages installed.

Return type (List[str])

idmtools.utils.json module

class idmtools.utils.json.**DefaultEncoder**(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)
Bases: json.encoder.JSONEncoder

A default JSON encoder to naively make Python objects serializable by using their __dict__.

default(*o*)
Implement this method in a subclass such that it returns a serializable object for o, or calls the base implementation (to raise a TypeError).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

class idmtools.utils.json.**IDMJSONEncoder**(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)
Bases: json.encoder.JSONEncoder

default(*o*)
Implement this method in a subclass such that it returns a serializable object for o, or calls the base implementation (to raise a TypeError).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

idmtools.utils.json.**load_json_file**(*path: str*) → Union[Dict[Any, Any], List]
Load a json object from a file

Parameters **path** – Path to file

Returns Contents of file parsed by JSON

idmtools.utils.language module

idmtools.utils.language.**on_off**(*test*) → str
Print on or off depending on boolean state of test

Parameters **test** – Boolean/object to check state

Returns On or off

idmtools.utils.language.**pluralize**(*word*, *plural_suffix='s'*)

idmtools.utils.language.**verbose_timedelta**(*delta*)

idmtools.utils.language.**get_qualified_class_name**(*cls: Type*) → str
Return the full class name for an object

Parameters **cls** – Class object to get name

Returns:

idmtools.utils.language.**get_qualified_class_name_from_obj**(*obj: object*) → str
Return the full class name from object

Parameters **obj** – Object

Example

` a = Platform('COMPS') class_name = get_qualified_class_name(a)
print(class_name) 'idmtools_platform_comps.comps_platform.COMPSPlatform' `

Returns Full module path to class of object

idmtools.utils.local_os module

class idmtools.utils.local_os.**LocalOS**
Bases: object

A Central class for representing values whose proper access methods may differ between platforms.

exception UnknownOS
Bases: Exception

os_mapping = {'darwin': 'mac', 'linux': 'lin', 'windows': 'win'}

username = 'docs'

name = 'lin'

static is_window()

idmtools.utils.time module

idmtools.utils.time.**timestamp**(*time=None*)
Return a timestamp.

Parameters **time** – A time object; if None provided, use now.

Returns A string timestamp in UTC, format YYYYMMDD_HHmmSS.

Module contents

Module contents

idmtools_models

idmtools_models package

Subpackages

idmtools_models.python package

Submodules

idmtools_models.python.json_python_task module

class idmtools_models.python.json_python_task.**JSONConfiguredPythonTask**(*command: Union[str, idmtools.entities.command_line.C = None, platform_requirements: Set[idmtools.entities.platform = <factory>, _ITask__pre_creation_hooks List[Callable[[Union[Forwa ForwardRef('IWorkflowItem')]], NoReturn]] = <factory>, _ITask__post_creation_hook List[Callable[[Union[Forwa ForwardRef('IWorkflowItem')]], NoReturn]] = <factory>, common_assets: idmtools.assets.asset_collection. = <factory>, transient_assets: idmtools.assets.asset_collection. = <factory>, _task_log: logging.Logger = <factory>, script_path: str = None, python_path: str = 'python',*

idmtools_models.python.python_task.PythonTask

configfile_argument: Optional[str] = '--config'

gather_common_assets()
Return the common assets for a JSON Configured Task a derived class Returns:

gather_transient_assets() → *idmtools.assets.asset_collection.AssetCollection*
Get Transient assets. This should general be the config.json

Returns Transient assets

reload_from_simulation(*simulation:* idmtools.entities.simulation.Simulation, ***kwargs*)
Reload the task from a simulation

Parameters

- **simulation** – Simulation to reload from
- ****kwargs** –

Returns None

See Also *idmtools_models.json_configured_task.JSONConfiguredTask.reload_from_simulation()* *idmtools_models.python.python_task.PythonTask.reload_from_simulation()*

pre_creation(*parent:* *Union[*idmtools.entities.simulation.Simulation, idmtools.entities.iworkflow_item.IWorkflowItem*]*)
Pre-creation

Parameters **parent** –

Returns None

See Also *idmtools_models.json_configured_task.JSONConfiguredTask.pre_creation()* *idmtools_models.python.python_task.PythonTask.pre_creation()*

post_creation(*parent:* *Union[*idmtools.entities.simulation.Simulation, idmtools.entities.iworkflow_item.IWorkflowItem*]*)
Post-creation

Parameters **parent** – Parent

Returns:

See Also idmtools_models.json_configured_task.JSONConfiguredTask.post_creation() idmtools_models.python.python_task.PythonTask.post_creation()

class idmtools_models.python.json_python_task.**JSONConfiguredPythonTaskSpecification**
Bases: *idmtools.registry.task_specification.TaskSpecification*

get(*configuration: dict*) → *idmtools_models.python.json_python_task.JSONConfiguredPythonTask*
Get instance of JSONConfiguredPythonTask with configuration

Parameters **configuration** – Configuration for task

Returns JSONConfiguredPythonTask with configuration

get_description() → str
Get description for plugin

Returns Plugin Description

get_type() → Type[*idmtools_models.python.json_python_task.JSONConfiguredPythonTask*]
Get Type for Plugin

Returns JSONConfiguredPythonTask

idmtools_models.python.python_task module

class idmtools_models.python.python_task.**PythonTask**(*command: Union[str, idmtools.entities.command_line.CommandLine] = None, platform_requirements: Set[idmtools.entities.platform_requirements.PlatformReq = <factory>, _ITask__pre_creation_hooks: List[Callable[[Union[ForwardRef('Simulation'), ForwardRef('IWorkflowItem')]], NoReturn]] = <factory>, _ITask__post_creation_hooks: List[Callable[[Union[ForwardRef('Simulation'), ForwardRef('IWorkflowItem')]], NoReturn]] = <factory>, common_assets: idmtools.assets.asset_collection.AssetCollection = <factory>, transient_assets: idmtools.assets.asset_collection.AssetCollection = <factory>, _task_log: logging.Logger = <factory>, script_path: str = None, python_path: str = 'python'*)

Bases: *idmtools.entities.itask.ITask*

script_path: str = None

python_path: str = 'python'

platform_requirements: Set[PlatformRequirements]

property command
Update executable with new python_path Returns: re-build command

retrieve_python_dependencies()
Retrieve the Pypi libraries associated with the given model script. .. rubric:: Notes

This function scan recursively through the whole directory where the model file is contained. This function relies on pipreqs being installed on the system to provide dependencies list.

Returns List of libraries required by the script

gather_common_assets() → *idmtools.assets.asset_collection.AssetCollection*
Get the common assets. This should be a set of assets that are common to all tasks in an experiment

Returns AssetCollection

gather_transient_assets() → *idmtools.assets.asset_collection.AssetCollection*
Gather transient assets. Generally this is the simulation level assets

Returns:

reload_from_simulation(*simulation:* idmtools.entities.simulation.Simulation, ***kwargs*)
Reloads a python task from a simulation

Parameters **simulation** – Simulation to reload

Returns:

pre_creation(*parent:* *Union[*idmtools.entities.simulation.Simulation, idmtools.entities.iworkflow_item.IWorkflowItem*]*)
Called before creation of parent

Parameters **parent** – Parent

Returns None

Raise: ValueError if script name is not provided

class idmtools_models.python.python_task.**PythonTaskSpecification**
Bases: *idmtools.registry.task_specification.TaskSpecification*

get(*configuration: dict*) → *idmtools_models.python.python_task.PythonTask*
Get instance of Python Task with specified configuration

Parameters **configuration** – Configuration for task

Returns Python task

get_description() → str
Description of the plugin

Returns Description string

get_example_urls() → List[str]
Return List of urls that have examples using PythonTask

Returns List of urls(str) that point to examples

get_type() → Type[*idmtools_models.python.python_task.PythonTask*]
Get Type for Plugin

Returns PythonTask

Module contents

idmtools_models.r package

Submodules

idmtools_models.r.json_r_task module

class idmtools_models.r.json_r_task.**JSONConfiguredRTask**(*command: Union[str, idmtools.entities.command_line.CommandLine] = None, platform_requirements: Set[idmtools.entities.platform_requirements.Platfor = <factory>, _ITask__pre_creation_hooks: List[Callable[[Union[ForwardRef('Simulation'), ForwardRef('IWorkflowItem')]], NoReturn]] = <factory>, _ITask__post_creation_hooks: List[Callable[[Union[ForwardRef('Simulation'), ForwardRef('IWorkflowItem')]], NoReturn]] = <factory>, common_assets: idmtools.assets.asset_collection.AssetCollection = <factory>, transient_assets: idmtools.assets.asset_collection.AssetCollection = <factory>, _task_log: logging.Logger = <factory>, image_name: str = None, build: bool = False, build_path: Union[str, NoneType] = None, Dockerfile: Union[str, NoneType] = None, pull_before_build: bool = True, use_nvidia_run: bool = False, _DockerTask__image_built: bool = False, script_path: str = None, r_path: str = 'Rscript', extra_libraries: list = <factory>, parameters: dict = <factory>, envelope: str = None, config_file_name: str = 'config.json', is_config_common: bool = False, command_line_argument: str = None, command_line_argument_no_filename: bool = True, configfile_argument: Union[str, NoneType] = '--config'*)

Bases: *idmtools_models.json_configured_task.JSONConfiguredTask*, *idmtools_models.r.r_task.RTask*

configfile_argument: Optional[str] = '--config'

gather_common_assets()
Return the common assets for a JSON Configured Task a derived class Returns:

gather_transient_assets() → *idmtools.assets.asset_collection.AssetCollection*
Get Transient assets. This should general be the config.json

Returns Transient assets

reload_from_simulation(*simulation:* idmtools.entities.simulation.Simulation, ***kwargs*)
Reload from Simulation. To do this, the process is

1. First check for a configfile name from arguments, then tags, or the default name
2. Load the json config file
3. Check if we got an envelope argument from parameters or the simulation tags, or on the task object

Parameters

- **simulation** – Simulation object with metadata to load info from
- **config_file_name** – Optional name of config file
- **envelope** – Optional name of envelope

Returns Populates the config with config from object

pre_creation(*parent:* *Union[*idmtools.entities.simulation.Simulation, idmtools.entities.iworkflow_item.IWorkflowItem*]*)
Called before creation of parent

Parameters **parent** – Parent

Returns None

Raise: ValueError if script name is not provided

post_creation(*parent:* *Union[*idmtools.entities.simulation.Simulation, idmtools.entities.iworkflow_item.IWorkflowItem*]*)
Optional Hook called at the after creation task. Can be used to setup simulation and experiment level hooks :param parent:

Returns:

class idmtools_models.r.json_r_task.**JSONConfiguredRTaskSpecification**
Bases: *idmtools.registry.task_specification.TaskSpecification*

get(*configuration: dict*) → *idmtools_models.r.json_r_task.JSONConfiguredRTask*
Get instance of JSONConfiguredRTaskSpecification with configuration provided

Parameters **configuration** – Configuration for object

Returns JSONConfiguredRTaskSpecification with configuration

get_description() → str
Get description of plugin

Returns Description of plugin

get_example_urls() → List[str]
Get Examples for JSONConfiguredRTask

Returns List of Urls that point to examples for JSONConfiguredRTask

get_type() → Type[*idmtools_models.r.json_r_task.JSONConfiguredRTask*]
Get Type for Plugin

Returns JSONConfiguredRTask

idmtools_models.r.r_task module

class idmtools_models.r.r_task.**RTask**(*command: Union[str, idmtools.entities.command_line.CommandLine] = None, platform_requirements: Set[idmtools.entities.platform_requirements.PlatformRequirements] = <factory>, _ITask__pre_creation_hooks: List[Callable[[Union[ForwardRef('Simulation'), ForwardRef('IWorkflowItem')]], NoReturn]] = <factory>, _ITask__post_creation_hooks: List[Callable[[Union[ForwardRef('Simulation'), ForwardRef('IWorkflowItem')]], NoReturn]] = <factory>, common_assets: idmtools.assets.asset_collection.AssetCollection = <factory>, transient_assets: idmtools.assets.asset_collection.AssetCollection = <factory>, _task_log: logging.Logger = <factory>, image_name: str = None, build: bool = False, build_path: Union[str, NoneType] = None, Dockerfile: Union[str, NoneType] = None, pull_before_build: bool = True, use_nvidia_run: bool = False, _DockerTask__image_built: bool = False, script_path: str = None, r_path: str = 'Rscript', extra_libraries: list = <factory>*)
Bases: *idmtools.core.docker_task.DockerTask*

script_path: str = None

r_path: str = 'Rscript'

extra_libraries: list

property command
Update executable with new python_path Returns: re-build command

reload_from_simulation(*simulation:* idmtools.entities.simulation.Simulation, ***kwargs*)
Optional hook that is called when loading simulations from a platform

gather_common_assets() → *idmtools.assets.asset_collection.AssetCollection*
Gather R Assets Returns:

gather_transient_assets() → *idmtools.assets.asset_collection.AssetCollection*
Gather transient assets. Generally this is the simulation level assets

Returns:

pre_creation(*parent: Union[*idmtools.entities.simulation.Simulation, idmtools.entities.iworkflow_item.IWorkflowItem*]*)
Called before creation of parent

Parameters **parent** – Parent

Returns None

Raise: ValueError if script name is not provided

class idmtools_models.r.r_task.**RTaskSpecification**

Bases: *idmtools.registry.task_specification.TaskSpecification*

get(*configuration: dict*) → *idmtools_models.r.r_task.RTask*

Get instance of RTask

Parameters **configuration** – configuration for task

Returns RTask with configuration

get_description() → str

Returns the Description of the plugin

Returns Plugin Description

get_type() → Type[*idmtools_models.r.r_task.RTask*]

Get Type for Plugin

Returns RTask

Module contents

Submodules

idmtools_models.json_configured_task module

class idmtools_models.json_configured_task.**JSONConfiguredTask**(*command: Union[str, idmtools.entities.command_line.CommandLine = None, platform_requirements: Set[idmtools.entities.platform_requirement = <factory>, _ITask__pre_creation_hooks: List[Callable[[Union[Simulation, IWorkflowItem]], NoReturn]] = <factory>, _ITask__post_creation_hooks: List[Callable[[Union[Simulation, IWorkflowItem]], NoReturn]] = <factory>, common_assets: idmtools.assets.asset_collection.AssetCollecti = <factory>, transient_assets: idmtools.assets.asset_collection.AssetCollecti = <factory>, _task_log: logging.Logger = <factory>, parameters: dict = <factory>, envelope: str = None, config_file_name: str = 'config.json', is_config_common: bool = False, command_line_argument: str = None, command_line_argument_no_filename: bool = True*)

Bases: *idmtools.entities.itask.ITask*

Defines an extensible simple task that implements functionality through optional supplied use hooks

parameters: dict

envelope: str = None

config_file_name: str = 'config.json'

is_config_common: bool = False

command_line_argument: str = None

command_line_argument_no_filename: bool = True

gather_common_assets() → *idmtools.assets.asset_collection.AssetCollection*
Gather assets common across an Experiment(Set of Simulations)

Returns Common AssetCollection

gather_transient_assets() → *idmtools.assets.asset_collection.AssetCollection*
Gather assets that are unique to this simulation/worktiem

Returns Simulation/workitem level AssetCollection

set_parameter(*key: Union[str, int, float]*, *value: Union[str, int, float, Dict[Union[str, int, float], Any]]*)
Update a parameter. The type hinting encourages JSON supported types

Parameters

- **key** – Config
- **value** –

Returns:

get_parameter(*key: Union[str, int, float]*) → Union[str, int, float, Dict[Union[str, int, float], Any]]
Returns a parameter value

Parameters **key** – Key of parameter

Returns Value of parameter

Raises **KeyError** –

update_parameters(*values: Dict[Union[str, int, float], Union[str, int, float, Dict[Union[str, int, float], Any]]]*)
Perform bulk update from another dictionary

Parameters **values** –

Returns:

reload_from_simulation(*simulation:* idmtools.entities.simulation.Simulation, *config_file_name: Optional[str] = None*, *envelope: Optional[str] = None*, ***kwargs*)
Reload from Simulation. To do this, the process is

1. First check for a configfile name from arguments, then tags, or the default name
2. Load the json config file
3. Check if we got an envelope argument from parameters or the simulation tags, or on the task object

Parameters

- **simulation** – Simulation object with metadata to load info from
- **config_file_name** – Optional name of config file
- **envelope** – Optional name of envelope

Returns Populates the config with config from object

pre_creation(*parent: Union[Simulation, WorkflowItem]*)
Optional Hook called at the time of creation of task. Can be used to setup simulation and experiment level hooks :param parent:

Returns:

static set_parameter_sweep_callback(*simulation: idmtools.entities.simulation.Simulation*, *param: str*, *value: Any*) → Dict[str, Any]

classmethod set_parameter_partial(*parameter: str*)

class idmtools_models.json_configured_task.**JSONConfiguredTaskSpecification**

Bases: *idmtools.registry.task_specification.TaskSpecification*

get(*configuration: dict*) → *idmtools_models.json_configured_task.JSONConfiguredTask*

Get instance of JSONConfiguredTask with configuration specified

Parameters **configuration** – Configuration for configuration

Returns JSONConfiguredTask with configuration

get_description() → str

Get description for plugin

Returns Description of plugin

get_example_urls() → List[str]

Get list of urls with examples for JSONConfiguredTask

Returns List of urls that point to examples relating to JSONConfiguredTask

get_type() → Type[*idmtools_models.json_configured_task.JSONConfiguredTask*]

Get task type provided by plugin

Returns JSONConfiguredTask

idmtools_models.templated_script_task module

class idmtools_models.templated_script_task.**TemplatedScriptTask**(*command: Union[str, idmtools.entities.command_line.CommandL = None, platform_requirements: Set[idmtools.entities.platform_requirem = <factory>, _ITask__pre_creation_hooks: List[Callable[[Union[Simulation, IWorkflowItem]], NoReturn]] = <factory>, _ITask__post_creation_hooks: List[Callable[[Union[Simulation, IWorkflowItem]], NoReturn]] = <factory>, common_assets: idmtools.assets.asset_collection.AssetColle = <factory>, transient_assets: idmtools.assets.asset_collection.AssetColle = <factory>, _task_log: logging.Logger = <factory>, script_path: str = None, template: str = None, template_file: str = None, template_is_common: bool = True, variables: Dict[str, Any] = <factory>, path_sep: str = '/', extra_command_arguments: str = '', gather_common_asset_hooks: List[Callable[[idmtools.entities.itask.IT idmtools.assets.asset_collection.AssetColle = <factory>, gather_transient_asset_hooks: List[Callable[[idmtools.entities.itask.IT idmtools.assets.asset_collection.AssetColle*

Defines a task to run a script using a template. Best suited to shell scripts

script_path: str = None
Name of script

template: str = None
The template contents

template_file: str = None
The template file. You can only use either template or template_file at once

template_is_common: bool = True
Controls whether a template should be an experiment or a simulation level asset

variables: Dict[str, Any]

path_sep: str = '/'
Platform Path Separator. For Windows execution platforms, use , otherwise use the default of /

extra_command_arguments: str = ''
Extra arguments to add to the command line

gather_common_asset_hooks: List[Callable[[ITask], AssetCollection]]
Hooks to gather common assets

gather_transient_asset_hooks: List[Callable[[ITask], AssetCollection]]
Hooks to gather transient assets

gather_common_assets() → *idmtools.assets.asset_collection.AssetCollection*
Gather common(experiment-level) assets for task

Returns AssetCollection containing common assets

gather_transient_assets() → *idmtools.assets.asset_collection.AssetCollection*
Gather transient(experiment-level) assets for task

Returns AssetCollection containing transient assets

reload_from_simulation(*simulation:* idmtools.entities.simulation.Simulation)
Reload a templated script task. When reloading, you will only have the rendered template available

Parameters **simulation** –

Returns:

pre_creation(*parent:* *Union[*idmtools.entities.simulation.Simulation, idmtools.entities.iworkflow_item.IWorkflowItem*]*)
Before creating simulation, we need to set our command line

Parameters **parent** – Parent object

Returns:

class idmtools_models.templated_script_task.**ScriptWrapperTask**(*command: Union[str, idmtools.entities.command_line.CommandLine = None, platform_requirements: Set[idmtools.entities.platform_requirement = <factory>, _ITask__pre_creation_hooks: List[Callable[[Union[Simulation, IWorkflowItem]], NoReturn]] = <factory>, _ITask__post_creation_hooks: List[Callable[[Union[Simulation, IWorkflowItem]], NoReturn]] = <factory>, common_assets: idmtools.assets.asset_collection.AssetCollectio = <factory>, transient_assets: idmtools.assets.asset_collection.AssetCollectio = <factory>, _task_log: logging.Logger = <factory>, template_script_task: idmtools_models.templated_script_task.Templ = None, task: idmtools.entities.itask.ITask = None*)

Bases: *idmtools.entities.itask.ITask*

Allows you to wrap a script with another script

See also:

idmtools_models.templated_script_task.TemplatedScriptTask

Raises **ValueError if the template Script Task is not defined** –

template_script_task: ***idmtools_models.templated_script_task.TemplatedScriptTask*** **= Non**

task: ***idmtools.entities.itask.ITask*** **= None**

gather_common_assets()
Gather all the common assets Returns:

gather_transient_assets() → *idmtools.assets.asset_collection.AssetCollection*
Gather all the transient assets Returns:

reload_from_simulation(*simulation:* idmtools.entities.simulation.Simulation)
Reload from simulation

Parameters **simulation** – simulation

Returns:

pre_creation (*parent: Union[idmtools.entities.simulation.Simulation, idmtools.entities.iworkflow_item.IWorkflowItem]*)
Before creation, create the true command by adding the wrapper name

Parameters **parent** –

Returns:

post_creation (*parent: Union[idmtools.entities.simulation.Simulation, idmtools.entities.iworkflow_item.IWorkflowItem]*)
Optional Hook called at the after creation task. Can be used to setup simulation and experiment level hooks :param parent:

Returns:

idmtools_models.templated_script_task.**get_script_wrapper_task**(*task: idmtools.entities.itask.ITask, wrapper_script_name: str, template_content: str = None, template_file: str = None, template_is_common: bool = True, variables: Dict[str, Any] = None, path_sep: str = '/'*) → *idmtools_models.templated_script_task.ScriptWrapperTask*

Convenience function that will wrap a task for you with some defaults

Parameters

- **task** – Task to wrap
- **wrapper_script_name** – Wrapper script name
- **template_content** – Template Content
- **template_file** – Template File
- **template_is_common** – Is the template experiment level
- **variables** – Variables
- **path_sep** – Path sep(Window or Linux)

Returns ScriptWrapperTask wrapping the task

See also:

idmtools_models.templated_script_task.get_script_wrapper_windows_task()
idmtools_models.templated_script_task.get_script_wrapper_unix_task()

idmtools_models.templated_script_task.**get_script_wrapper_windows_task**(*task: idmtools.entities.itask.ITask*, *wrapper_script_name: str = 'wrapper.bat'*, *template_content: str = '\nset PYTHONPATH=%cd%\\Assets\\;%PYT*, *template_file: str = None*, *template_is_common: bool = True*, *variables: Dict[str, Any] = None*) → *idmtools_models.templated_script*

Get wrapper script task for windows platforms

The default content wraps a bash script that adds the assets directory to the python path

```
set PYTHONPATH=%cd%/Assets/;%PYTHONPATH%
%*
```

You can adapt this script to modify any pre-scripts you need or call others scripts in succession

Parameters

- **task** – Task to wrap
- **wrapper_script_name** – Wrapper script name(defaults to wrapper.bat)
- **template_content** – Template Content.
- **template_file** – Template File
- **template_is_common** – Is the template experiment level
- **variables** – Variables for template

Returns ScriptWrapperTask

See Also:: *idmtools_models.templated_script_task.get_script_wrapper_task()*
idmtools_models.templated_script_task.get_script_wrapper_unix_task()

idmtools_models.templated_script_task.**get_script_wrapper_unix_task**(*task: idmtools.entities.itask.ITask*, *wrapper_script_name: str = 'wrapper.sh'*, *template_content: str = None*, *template_file: str = None*, *template_is_common: bool = True*, *variables: Dict[str, Any] = None*)

Get wrapper script task for unix platforms

The default content wraps a bash script that adds the assets directory to the python path

```
set PYTHONPATH=$(pwd)/Assets/:$PYTHONPATH
%*
```

You can adapt this script to modify any pre-scripts you need or call others scripts in succession

Parameters

- **task** – Task to wrap
- **wrapper_script_name** – Wrapper script name(defaults to wrapper.sh)
- **template_content** – Template Content
- **template_file** – Template File
- **template_is_common** – Is the template experiment level
- **variables** – Variables for template

Returns ScriptWrapperTask

See Also: *idmtools_models.templated_script_task.get_script_wrapper_task()* *idmtools_models.templated_script_task.get_script_wrapper_windows_task()*

class idmtools_models.templated_script_task.**TemplatedScriptTaskSpecification**

Bases: *idmtools.registry.task_specification.TaskSpecification*

get(*configuration: dict*) → *idmtools_models.templated_script_task.TemplatedScriptTask*

Get instance of TemplatedScriptTask with configuration

Parameters **configuration** – configuration for TemplatedScriptTask

Returns TemplatedScriptTask with configuration

get_description() → str
Get description of plugin

Returns Plugin description

get_example_urls() → List[str]
Get example urls related to TemplatedScriptTask

Returns List of urls that have examples related to CommandTask

get_type() → Type[*idmtools_models.templated_script_task.TemplatedScriptTask*]
Get task type provided by plugin

Returns TemplatedScriptTask

class idmtools_models.templated_script_task.**ScriptWrapperTaskSpecification**
Bases: *idmtools.registry.task_specification.TaskSpecification*

get(*configuration: dict*) → *idmtools_models.templated_script_task.ScriptWrapperTask*
Get instance of ScriptWrapperTask with configuration

Parameters **configuration** – configuration for ScriptWrapperTask

Returns TemplatedScriptTask with configuration

get_description() → str
Get description of plugin

Returns Plugin description

get_example_urls() → List[str]
Get example urls related to ScriptWrapperTask

Returns List of urls that have examples related to CommandTask

get_type() → Type[*idmtools_models.templated_script_task.ScriptWrapperTask*]
Get task type provided by plugin

Returns TemplatedScriptTask

Module contents

idmtools_platform_comps

idmtools_platform_comps package

Subpackages

idmtools_platform_comps.cli package

Submodules

idmtools_platform_comps.cli.cli_functions module

idmtools_platform_comps.cli.cli_functions.**validate_range**(*value: float*, *min: float*, *max: float*) → Tuple[bool, str]
Function used to validate an integer value between min and max :param value: The value set by the user :param min: Minimum value :param max: Maximum value

Returns: tuple with validation result and error message if needed

idmtools_platform_comps.cli.cli_functions.**environment_list**(*previous_settings: Dict*, *current_field: dataclasses.Field*) → Dict

Allows the CLI to provide a list of available environments. Uses the previous_settings to get the endpoint to query for environments :param previous_settings: previous settings set by the user in the CLI. :param current_field: Current field specs

Returns: updates to the choices and default

idmtools_platform_comps.cli.comps module

idmtools_platform_comps.cli.utils module

Module contents

idmtools_platform_comps.comps_operations package

Submodules

idmtools_platform_comps.comps_operations.asset_collection_operations module

class idmtools_platform_comps.comps_operations.asset_collection_operations.**CompsPlatformAss**

Bases: *idmtools.entities.iplatform_ops.iplatform_asset_collection_operations.IPlatformAssetCollectionOperations*

platform: 'COMPSPlatform'

platform_type
alias of COMPS.Data.AssetCollection.AssetCollection

get(*asset_collection_id: uuid.UUID*, *load_children: Optional[List[str]] = None*, *query_criteria: Optional[COMPS.Data.QueryCriteria.QueryCriteria] = None*, ***kwargs*) → COMPS.Data.AssetCollection.AssetCollection
Get an asset collection by id

Parameters

- **asset_collection_id** – Id of asset collection
- **load_children** – Optional list of children to load. Defaults to assets and tags
- **query_criteria** – Optional query_criteria. Ignores children default
- ****kwargs** –

Returns COMPSAssetCollection

platform_create(*asset_collection:* idmtools.assets.asset_collection.AssetCollection, ***kwargs*) → COMPS.Data.AssetCollection.AssetCollection
Create AssetCollection

Parameters

- **asset_collection** – AssetCollection to create
- ****kwargs** –

Returns COMPSAssetCollection

to_entity(*asset_collection: Union[COMPS.Data.AssetCollection.AssetCollection, COMPS.Data.SimulationFile.SimulationFile, List[COMPS.Data.SimulationFile.SimulationFile], COMPS.Data.OutputFileMetadata.OutputFileMetadata]*, ***kwargs*) → *idmtools.assets.asset_collection.AssetCollection*
Convert COMPS Asset Collection or Simulation File to IDM Asset Collection

Parameters

- **asset_collection** – Comps asset/asset collection to convert to idm asset collection
- ****kwargs** –

Returns AssetCollection

idmtools_platform_comps.comps_operations.experiment_operations module

class idmtools_platform_comps.comps_operations.experiment_operations.**CompsPlatformExperime**

Bases: *idmtools.entities.iplatform_ops.iplatform_experiment_operations.IPlatformExperimentOperations*

platform: 'COMPSPlatform'

platform_type
alias of COMPS.Data.Experiment.Experiment

get(*experiment_id: uuid.UUID*, *columns: Optional[List[str]] = None*, *load_children: Optional[List[str]] = None*, *query_criteria: Optional[COMPS.Data.QueryCriteria.QueryCriteria] = None*, ***kwargs*) → COMPS.Data.Experiment.Experiment
Fetch experiments from COMPS

Parameters

- **experiment_id** – Experiment ID
- **columns** – Optional Columns. If not provided, id, name, and suite_id are fetched
- **load_children** – Optional Children. If not provided, tags and configuration are specified
- **query_criteria** – Optional QueryCriteria
- ****kwargs** –

Returns COMPSExperiment with items

pre_create(*experiment:* idmtools.entities.experiment.Experiment, ***kwargs*) → NoReturn
Pre-create for Experiment. At moment, validation related to COMPS is all that is done

Parameters

- **experiment** – Experiment to run pre-create for
- ****kwargs** –

Returns:

platform_create(*experiment:* idmtools.entities.experiment.Experiment, *num_cores: Optional[int] = None*, *executable_path: Optional[str] = None*, *command_arg: Optional[str] = None*, *priority: Optional[str] = None*, *check_command: bool = True*) → COMPS.Data.Experiment.Experiment
Create Experiment on the COMPS Platform

Parameters

- **experiment** – IDMTools Experiment to create
- **num_cores** – Optional num of cores to allocate using MPI
- **executable_path** – Executable path
- **command_arg** – Command Argument
- **priority** – Priority of command
- **check_command** – Run task hooks on item

Returns COMPSExperiment that was created

post_run_item(*experiment:* idmtools.entities.experiment.Experiment, ***kwargs*)
Ran after experiment. Nothing is done on comps other that alerting the user to the item

Parameters

- **experiment** – Experiment to run post run item
- ****kwargs** –

Returns None

get_children(*experiment: COMPS.Data.Experiment.Experiment*, *columns: Optional[List[str]] = None*, *children: Optional[List[str]] = None*, ***kwargs*) → List[COMPS.Data.Simulation.Simulation]
Get children for a COMPSExperiment

Parameters

- **experiment** – Experiment to get children of Comps Experiment
- **columns** – Columns to fetch. If not provided, id, name, experiment_id, and state will be loaded
- **children** – Children to load. If not provided, Tags will be loaded
- ****kwargs** –

Returns Simulations belonging to the Experiment

get_parent(*experiment: COMPS.Data.Experiment.Experiment*, ***kwargs*) → COMPS.Data.Suite.Suite
Get Parent of experiment

Parameters

- **experiment** – Experiment to get parent of
- ****kwargs** –

Returns Suite of the experiment

platform_run_item(*experiment:* idmtools.entities.experiment.Experiment, ***kwargs*)
Run experiment on COMPS. Here we commission the experiment

Parameters

- **experiment** – Experiment to run
- ****kwargs** –

Returns None

send_assets(*experiment:* idmtools.entities.experiment.Experiment, ***kwargs*)
Send assets related to the experiment

Parameters

- **experiment** – Experiment to send assets for
- ****kwargs** –

Returns None

refresh_status(*experiment:* idmtools.entities.experiment.Experiment, ***kwargs*)
Reload status for experiment(load simulations)

Parameters

- **experiment** – Experiment to load status for
- ****kwargs** –

Returns None

to_entity(*experiment: COMPS.Data.Experiment.Experiment*, *parent: Optional[COMPS.Data.Suite.Suite] = None*, *children: bool = True*, ***kwargs*) → *idmtools.entities.experiment.Experiment*
Converts a COMPSExperiment to an idmtools Experiment

Parameters

- **experiment** – COMPS Experiment objet to convert
- **parent** – Optional suite parent
- **children** – Should we load children objects?
- ****kwargs** –

Returns Experiment

get_assets_from_comps_experiment(*experiment: COMPS.Data.Experiment.Experiment*) → Optional[*idmtools.assets.asset_collection.AssetCollection*]

platform_list_asset(*experiment:* idmtools.entities.experiment.Experiment, ***kwargs*) → List[*idmtools.assets.asset.Asset*]

idmtools_platform_comps.comps_operations.simulation_operations module

idmtools_platform_comps.comps_operations.simulation_operations.**comps_batch_worker**(*simulations: List[idmtool interface: idmtools_platfor num_cores: Optional[int] = None, priority: Optional[str] = None*) → List[COMPS

Run batch worker

Parameters

- **simulations** – Batch of simulation to process
- **interface** – SimulationOperation Interface
- **num_cores** – Optional Number of core to allocate for MPI
- **priority** – Optional Priority to set to

Returns List of Comps Simulations

class idmtools_platform_comps.comps_operations.simulation_operations.**CompsPlatformSimulatio**

Bases: *idmtools.entities.iplatform_ops.iplatform_simulation_operations.IPlatformSimulationOperations*

platform: 'COMPSPlatform'

platform_type
alias of COMPS.Data.Simulation.Simulation

get(*simulation_id: uuid.UUID, columns: Optional[List[str]] = None, load_children: Optional[List[str]] = None, query_criteria: Optional[COMPS.Data.QueryCriteria.QueryCriteria] = None, **kwargs*) → COMPS.Data.Simulation.Simulation
Get Simulation from Comps

Parameters

- **simulation_id** – ID
- **columns** – Optional list of columns to load. Defaults to "id", "name", "experiment_id", "state"
- **load_children** – Optional children to load. Defaults to "tags", "configuration"
- **query_criteria** – Optional query_criteria object to use your own custom criteria object
- ****kwargs** –

Returns COMPSSimulation

platform_create(*simulation:* idmtools.entities.simulation.Simulation, *num_cores: int = None*, *priority: str = None*, *enable_platform_task_hooks: bool = True*) → COMPS.Data.Simulation.Simulation

Create Simulation on COMPS

Parameters

- **simulation** – Simulation to create
- **num_cores** – Optional number of MPI Cores to allocate
- **priority** – Priority to load
- **enable_platform_task_hooks** – Should platform task hoooks be ran

Returns COMPS Simulation

to_comps_sim(*simulation:* idmtools.entities.simulation.Simulation, *num_cores: int = None*, *priority: str = None*, *config: COMPS.Data.Configuration.Configuration = None*)

Covert IDMTools object to COMPS Object

Parameters

- **simulation** – Simulation object to convert
- **num_cores** – Optional Num of MPI Cores to allocate
- **priority** – Optional Priority
- **config** – Optional Configuration objet

Returns COMPS Simulation

static get_simulation_config_from_simulation(*simulation:* idmtools.entities.simulation.Simulation, *num_cores: int = None*, *priority: str = None*) → COMPS.Data.Configuration.Configuration

Get the comps configuration for a Simulation Object

Parameters

- **simulation** – Simulation
- **num_cores** – Optional Num of core for MPI
- **priority** – Optional Priority

Returns Configuration

batch_create(*simulations: List[idmtools.entities.simulation.Simulation]*, *num_cores: int = None*, *priority: str = None*) → List[COMPS.Data.Simulation.Simulation]
Perform batch creation of Simulations

Parameters

- **simulations** – Simulation to create
- **num_cores** – Optional MPI Cores to allocate per simulation
- **priority** – Optional Priority

Returns List of COMPSSimulations that were created

get_parent(*simulation: Any*, ***kwargs*) → COMPS.Data.Experiment.Experiment
Get the parent of the simulation

Parameters

- **simulation** – Simulation to load parent for
- ****kwargs** –

Returns COMPSExperiment

platform_run_item(*simulation: idmtools.entities.simulation.Simulation*, ***kwargs*)
Called during commissioning of an item. This should create the remote resource but not upload assets

Parameters **simulation** – Simulation to run

Returns:

send_assets(*simulation: idmtools.entities.simulation.Simulation*, *comps_sim: Optional[COMPS.Data.Simulation.Simulation] = None*, *add_metadata: bool = False*, ***kwargs*)
Send assets to Simulation

Parameters

- **simulation** – Simulation to send asset for
- **comps_sim** – Optional COMPSSimulation object to prevent reloading it
- **add_metadata** – Add idmtools metadata object
- ****kwargs** –

Returns None

refresh_status(*simulation: idmtools.entities.simulation.Simulation*, *additional_columns: Optional[List[str]] = None*, ***kwargs*)
Refresh status of a simulation

Parameters

- **simulation** – Simulation to refresh
- **additional_columns** – Optional additional columns to load from COMPS
- ****kwargs** –

Returns:

to_entity(*simulation: COMPS.Data.Simulation.Simulation*, *load_task: bool = False*, *parent: Optional[idmtools.entities.experiment.Experiment] = None*, *load_parent: bool = False*, *load_metadata: bool = False*, ***kwargs*) → *idmtools.entities.simulation.Simulation*
Convert COMPS simulation object to IDM Tools simulation object

Parameters

- **simulation** – Simulation object
- **load_task** – Should we load tasks. Defaults to No. This can increase the load items on fetchs
- **parent** – Optional parent object to prevent reloads
- **load_parent** – Force load of parent(Beware, This could cause loading loops)
- **metadata** – Should we load metadata by default. If load task is enabled, this is also enabled
- ****kwargs** –

Returns Simulation object

get_asset_collection_from_comps_simulation(*simulation: COMPS.Data.Simulation.Simulation*) → Optional[*idmtools.assets.asset_collection.AssetCollection*]

Get assets from COMPS Simulation

Parameters **simulation** – Simulation to get assets from

Returns:

get_assets(*simulation:* idmtools.entities.simulation.Simulation, *files: List[str]*, ***kwargs*) → Dict[str, bytearray]

Fetch the files associated with a simulation

Parameters

- **simulation** – Simulation
- **files** – List of files to download
- ****kwargs** –

Returns Dictionary of filename -> ByteArray

list_assets(*simulation:* idmtools.entities.simulation.Simulation, *common_assets: bool = False*, ***kwargs*) → List[*idmtools.assets.asset.Asset*]

List assets for a simulation

Parameters

- **simulation** – Simulation to load data for
- **common_assets** – Should we load asset files
- ****kwargs** –

Returns AssetCollection

retrieve_output_files(*simulation:* idmtools.entities.simulation.Simulation)

all_files(*simulation:* idmtools.entities.simulation.Simulation, *common_assets: bool = False*, *outfiles: bool = True*, ***kwargs*) → List[*idmtools.assets.asset.Asset*]

Returns all files for a specific simulation including experiments or non-assets

Parameters

- **simulation** – Simulation all files
- **common_assets** – Include experiment assets
- **outfiles** – Include output files
- ****kwargs** –

Returns AssetCollection

idmtools_platform_comps.comps_operations.suite_operations module

class idmtools_platform_comps.comps_operations.suite_operations.**CompsPlatformSuiteOperatio**

Bases: *idmtools.entities.iplatform_ops.iplatform_suite_operations.IPlatformSuiteOperations*

platform: 'COMPSPlatform'

platform_type
alias of COMPS.Data.Suite.Suite

get(*suite_id: uuid.UUID*, *columns: Optional[List[str]] = None*, *load_children: Optional[List[str]] = None*, *query_criteria: Optional[COMPS.Data.QueryCriteria.QueryCriteria] = None*, ***kwargs*) → COMPS.Data.Suite.Suite
Get COMPS Suite

Parameters

- **suite_id** – Suite id
- **columns** – Optional list of columns. Defaults to id and name
- **load_children** – Optional list of children to load. Defaults to "tags", "configuration"
- **query_criteria** – Optional query criteria
- ****kwargs** –

Returns: COMPSSuite

platform_create(*suite:* idmtools.entities.suite.Suite, ***kwargs*) → Tuple[COMPS.Data.Suite.Suite, uuid.UUID]
Create suite on COMPS

Parameters

- **suite** – Suite to create
- ****kwargs** –

Returns COMPS Suite object and a UUID

get_parent(*suite: COMPS.Data.Suite.Suite*, ***kwargs*) → Any
Get parent of suite. We always return None on COMPS

Parameters

- **suite** –
- ****kwargs** –

Returns None

get_children(*suite: COMPS.Data.Suite.Suite*, ***kwargs*) → List[Union[COMPS.Data.Experiment.Experiment, COMPS.Data.WorkItem.WorkItem]]
Get children for a suite :param suite: Suite to get children for :param **kwargs: Any arguments to pass on to loading functions

Returns List of COMPS Experiments/Workitems that are part of the suite

refresh_status(*suite:* idmtools.entities.suite.Suite, ***kwargs*)
Refresh the status of a suite. On comps, this is done by refreshing all experiments :param suite: Suite to refresh status of :param **kwargs:

Returns:

to_entity(*suite: COMPS.Data.Suite.Suite*, *children: bool = True*, ***kwargs*) → *idmtools.entities.suite.Suite*
Convert a COMPS Suite to an IDM Suite

Parameters

- **suite** – Suite to Convert
- **children** – When true, load simulations, false otherwise
- ****kwargs** –

Returns IDM Suite

idmtools_platform_comps.comps_operations.workflow_item_operations module

class idmtools_platform_comps.comps_operations.workflow_item_operations.**CompsPlatformWorkfl**

Bases: *idmtools.entities.iplatform_ops.iplatform_workflowitem_operations.IPlatformWorkflowItemOperations*

platform: 'COMPSPlatform'

platform_type
alias of COMPS.Data.WorkItem.WorkItem

get(*workflow_item_id: uuid.UUID*, *columns: Optional[List[str]] = None*, *load_children: Optional[List[str]] = None*, *query_criteria: Optional[COMPS.Data.QueryCriteria.QueryCriteria] = None*, ***kwargs*) → COMPS.Data.WorkItem.WorkItem
Get COMPSWorkItem

Parameters

- **workflow_item_id** – Item id
- **columns** – Optional columns to load. Defaults to "id", "name", "state"
- **load_children** – Optional list of COMPS Children objects to load. Defaults to "Tags"
- **query_criteria** – Optional QueryCriteria
- ****kwargs** –

Returns COMPSWorkItem

platform_create(*work_item:* idmtools.entities.iworkflow_item.IWorkflowItem, ***kwargs*) → Tuple[Any]
Creates an workflow_item from an IDMTools work_item object

Parameters

- **work_item** – WorkflowItem to create
- ****kwargs** – Optional arguments mainly for extensibility

Returns Created platform item and the UUID of said item

platform_run_item(*work_item:* idmtools.entities.iworkflow_item.IWorkflowItem, ***kwargs*)
Start to rum COMPS WorkItem created from work_item :param work_item: workflow item

Returns: None

get_parent(*work_item:* idmtools.entities.iworkflow_item.IWorkflowItem, ***kwargs*) → Any
Returns the parent of item. If the platform doesn't support parents, you should throw a TopLevelItem error :param work_item: COMPS WorkItem :param **kwargs: Optional arguments mainly for extensibility

Returns: item parent

Raise: TopLevelItem

get_children(*work_item:* idmtools.entities.iworkflow_item.IWorkflowItem, ***kwargs*) → List[Any]
Returns the children of an workflow_item object

Parameters

- **work_item** – WorkflowItem object
- ****kwargs** – Optional arguments mainly for extensibility

Returns Children of work_item object

refresh_status(*workflow_item:* idmtools.entities.iworkflow_item.IWorkflowItem, ***kwargs*)
Refresh status for workflow item :param work_item: Item to refresh status for

Returns None

send_assets(*workflow_item:* idmtools.entities.iworkflow_item.IWorkflowItem, ***kwargs*)
Add asset as WorkItemFile :param workflow_item: workflow item

Returns: None

list_assets(*workflow_item:* idmtools.entities.iworkflow_item.IWorkflowItem, ***kwargs*) → List[str]
Get list of asset files :param workflow_item: workflow item :param **kwargs: Optional arguments mainly for extensibility

Returns: list of assets associated with WorkItem

get_assets(*workflow_item:* idmtools.entities.iworkflow_item.IWorkflowItem, *files: List[str]*, ***kwargs*) → Dict[str, bytearray]
Retrieve files association with WorkItem :param workflow_item: workflow item :param files: list of file paths :param **kwargs: Optional arguments mainly for extensibility

Returns: dict with key/value: file_path/file_content

to_entity(*work_item: COMPS.Data.WorkItem.WorkItem*, ***kwargs*) → *idmtools.entities.iworkflow_item.IWorkflowItem*
Converts the platform representation of workflow_item to idmtools representation

Parameters

- **work_item** – Platform workflow_item object
- ****kwargs** – Optional arguments mainly for extensibility

Returns IDMTools workflow item

get_related_items(*item:* idmtools.entities.iworkflow_item.IWorkflowItem, *relation_type:* *COMPS.Data.WorkItem.RelationType*) → Dict[str, List[Dict[str, str]]]

Get related WorkItems, Suites, Experiments, Simulations and AssetCollections :param item: workflow item :param relation_type: RelationType

Returns: Dict

Module contents

idmtools_platform_comps.ssmt_operations package

Submodules

idmtools_platform_comps.ssmt_operations.simulation_operations module

class idmtools_platform_comps.ssmt_operations.simulation_operations.**SSMTPlatformSimulationO**

Bases: *idmtools_platform_comps.comps_operations.simulation_operations.CompsPlatformSimulationOperations*

get_assets(*simulation:* idmtools.entities.simulation.Simulation, *files:* *List[str]*, ***kwargs*) → Dict[str, bytearray]

Fetch the files associated with a simulation

Parameters

- **simulation** – Simulation
- **files** – List of files to download
- ****kwargs** –

Returns Dictionary of filename -> ByteArray

platform

idmtools_platform_comps.ssmt_operations.workflow_item_operations module

class idmtools_platform_comps.ssmt_operations.workflow_item_operations.**SSMTPlatformWorkflow**

Bases: *idmtools_platform_comps.comps_operations.workflow_item_operations.CompsPlatformWorkflowItemOperations*

get_assets(*simulation:* idmtools.entities.simulation.Simulation, *files:* *List[str]*, ***kwargs*) → Dict[str, bytearray]

Retrieve files association with WorkItem :param workflow_item: workflow item :param files: list of file paths :param **kwargs: Optional arguments mainly for extensibility

Returns: dict with key/value: file_path/file_content

platform

Module contents

idmtools_platform_comps.ssmt_work_items package

Submodules

idmtools_platform_comps.ssmt_work_items.comps_workitems module

class idmtools_platform_comps.ssmt_work_items.comps_workitems.**SSMTWorkItem**(*_uid: uuid.UUID = None*, *platform_id: uuid.UUID = None*, *_platform: IPlatform = None*, *parent_id: uuid.UUID = None*, *_parent: IEntity = None*, *status: idmtools.core.enums.EntityStatus = None*, *tags: Dict[str, Any] = <factory>*, *_platform_object: Any = None*, *name: str = None*, *assets: idmtools.assets.asset_collection.AssetCollection = <factory>*, *item_name: str = 'Idm*

ICOMPSWorkflowItem

Idm SSMTWorkItem

docker_image: str = None

command: str = None

get_base_work_order()
builder basic work order Returns: work order as a dictionary

get_comps_ssmt_image_name()
build comps ssmt docker image name :param user_image: the image name provided by user

Returns: final validated name

class idmtools_platform_comps.ssmt_work_items.comps_workitems.**InputDataWorkItem**(*_uid: uuid.UUID = None, platform_id: uuid.UUID = None, _platform: IPlatform = None, parent_id: uuid.UUID = None, _parent: IEntity = None, status: idmtools.core.enum = None, tags: Dict[str, Any] = <factory>, _platform_object: Any = None, name: str = None, assets: idmtools.assets.ass = <factory>, item_name: str = 'Idm WorkItem Test',*

Bases: *idmtools_platform_comps.ssmt_work_items.icomps_workflowitem.ICOMPSWorkflowItem*

Idm InputDataWorkItem

work_order

class idmtools_platform_comps.ssmt_work_items.comps_workitems.**VisToolsWorkItem**(*_uid: uuid.UUID = None*, *platform_id: uuid.UUID = None*, *_platform: IPlatform = None*, *parent_id: uuid.UUID = None*, *_parent: IEntity = None*, *status: idmtools.core.enums. = None*, *tags: Dict[str, Any] = <factory>*, *_platform_object: Any = None*, *name: str = None*, *assets: idmtools.assets.asset = <factory>*, *item_name: str = 'Idm WorkItem Test'*,

Bases: *idmtools_platform_comps.ssmt_work_items.icomps_workflowitem.ICOMPSWorkflowItem*

Idm VisToolsWorkItem

work_order

idmtools_platform_comps.ssmt_work_items.icomps_workflowitem module

class idmtools_platform_comps.ssmt_work_items.icomps_workflowitem.**ICOMPSWorkflowItem**(*_uid: uuid.UU = None, platform_id uuid.UU = None, _platform: IPlatform = None, parent_id: uuid.UU = None, _parent: IEntity = None, status: idmtools.co = None, tags: Dict[str Any] = <factory>, _platform_ob Any = None, name: str = None, assets: idmtools.as = <factory>, item_na str = 'Idm*

Interface of idmtools work item

item_name: str = 'Idm WorkItem Test'

work_order: dict

work_item_type: str = None

plugin_key: str = '1.0.0.0_RELEASE'

get_base_work_order()

load_work_order(*wo_file*)

set_work_order(*wo*)
Update wo for the name with value :param wo: user wo

Returns: None

update_work_order(*name*, *value*)
Update wo for the name with value :param name: wo arg name :param value: wo arg value

Returns: None

clear_wo_args()
Clear all existing wo args

Returns: None

Module contents

idmtools_platform_comps.utils package

Subpackages

idmtools_platform_comps.utils.python_requirements_ac package

Submodules

idmtools_platform_comps.utils.python_requirements_ac.create_asset_collection module

idmtools_platform_comps.utils.python_requirements_ac.create_asset_collection.**calculate_md5**

Calculate and md5

idmtools_platform_comps.utils.python_requirements_ac.create_asset_collection.**build_asset_f**
Utility function to build all library files :param prefix: used to identify library files

Returns: file paths as a list

idmtools_platform_comps.utils.python_requirements_ac.create_asset_collection.**get_first_simu**
Retrieve the first simulation from an experiment :param exp_id: use input (experiment id)

Returns: list of files paths

idmtools_platform_comps.utils.python_requirements_ac.create_asset_collection.**main**()

idmtools_platform_comps.utils.python_requirements_ac.install_requirements module

idmtools_platform_comps.utils.python_requirements_ac.install_requirements.**install_packages**
Install our packages to a local directory :param requirements_file: requirements file :param python_paths: system Python path

Returns: None

idmtools_platform_comps.utils.python_requirements_ac.install_requirements.**set_python_dates**

idmtools_platform_comps.utils.python_requirements_ac.install_requirements.**compile_all**(*python*

idmtools_platform_comps.utils.python_requirements_ac.requirements_to_asset_collection module

class idmtools_platform_comps.utils.python_requirements_ac.requirements_to_asset_collectio

Bases: object

platform: *idmtools_platform_comps.comps_platform.COMPSPlatform* **= None**
Platform object

requirements_path: **str = None**
Path to requirements file

pkg_list: **list = None**
list of packages

local_wheels: **list = None**
list of wheel files locally to upload and install

property checksum

Returns The md5 of the requirements.

property requirements

Returns Consolidated requirements.

run(*rerun=False*)

The working logic of this utility:

1. check if asset collection exists for given requirements, return ac id if exists
2. create an Experiment to install the requirements on COMPS
3. create a WorkItem to create a Asset Collection

Returns: return ac id based on the requirements if Experiment and WorkItem Succeeded

save_updated_requirements()
Save consolidated requirements to a file requirements_updated.txt Returns:

retrieve_ac_by_tag(*md5_check=None*)
Retrieve comps asset collection given ac tag :param md5_check: also can use custom md5 string as search tag

Returns: comps asset collection

retrieve_ac_from_wi(*wi*)
Retrieve ac id from file ac_info.txt saved by WI :param wi: SSMTWorkItem (which was used to create ac from library)

Returns: COMPS asset collection

add_wheels_to_assets(*experiment*)

run_experiment_to_install_lib()
Create an Experiment which will run another py script to install requirements Returns: Experiment created

run_wi_to_create_ac(*exp_id*)
Create an WorkItem which will run another py script to create new asset collection :param exp_id: the Experiment id (which installed requirements)

Returns: work item created

static get_latest_version(*pkg_name*, *display_all=False*)
Utility to get the latest version for a given package name :param pkg_name: package name given :param display_all: determine if output all package releases

Returns: the latest version of ven package

consolidate_requirements()

Combine requiremtns and dynamic requirements (a list):

- get the latest version of package if version is not provided
- dynamic requirements will overwrites the requirements file

Returns: the consolidated requirements (as a list)

Module contents

Submodules

idmtools_platform_comps.utils.disk_usage module

class idmtools_platform_comps.utils.disk_usage.**ExperimentInfo**(*id*, *name*, *owner*, *size*, *sims*)

Bases: object

class idmtools_platform_comps.utils.disk_usage.**DiskSpaceUsage**

Bases: object

TOP_COUNT = 15

OWNERS = []

static get_experiment_info(*experiment: COMPS.Data.Experiment.Experiment*, *cache*, *refresh*)

Adds the experiment information for a given experiment to the cache: - raw_size: the size in bytes - size: the formatted size (in KB, MB or GB) - sims: the number of simulations This function is used by the process pool to parallelize the retrieval of experiment info :param experiment: The experiment to analyze :param cache: :param refresh:

Returns:

static exp_str(*info*, *display_owner=True*)

Format an experiment and its information to a string.

static top_count_experiments(*experiments_info*)

Displays the top count of all experiments analyzed

static total_size_per_user(*experiments_info*)

Displays the total disk space occupied per user

static top_count_experiments_per_user(*experiments_info*)

Display the top count biggest experiments per user

static gather_experiment_info(*refresh=False*, *max_workers: int = 6*)

static display(*platform: idmtools_platform_comps.comps_platform.COMPSPlatform*, *users*, *top=15*, *save=False*, *refresh=False*)

static save_to_file(*experiments_info*)

class idmtools_platform_comps.utils.disk_usage.**DiskEncoder**(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)

Bases: json.encoder.JSONEncoder

default(*o*)

Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

idmtools_platform_comps.utils.download_experiment module

idmtools_platform_comps.utils.download_experiment.**get_script_extension**()

idmtools_platform_comps.utils.download_experiment.**download_asset**(*asset*, *path*)

idmtools_platform_comps.utils.download_experiment.**write_script**(*simulation:* idmtools.entities.simulation.Simulation, *path*)

Writes a shell script to execute simulation :param simulation: :param path:

Returns:

idmtools_platform_comps.utils.download_experiment.**write_experiment_script**(*experiment:* idmtools.entities.experiment *path:* *str*)

Write an experiment script :param experiment: :param path:

Returns:

idmtools_platform_comps.utils.download_experiment.**download_experiment**(*experiment:* idmtools.entities.experiment.Expe *destination:* *str*)

Downloads experiment to local directory. Usefule for troubleshooting experiments

Parameters

- **experiment** – Experiment to download
- **destination** – Destionation Directory

Returns:

idmtools_platform_comps.utils.general module

idmtools_platform_comps.utils.general.**fatal_code**(*e: Exception*) → bool

Uses to determine if we should stop retrying based on request status code

Parameters **e** – Exeception to check

Returns True is exception is a request and status code matches 404

idmtools_platform_comps.utils.general.**convert_comps_status**(*comps_status:* *COMPS.Data.Simulation.SimulationState*) → *idmtools.core.enums.EntityStatus*

Convert status from COMPS to IDMTools

Parameters **comps_status** – Status in Comps

Returns EntityStatus

idmtools_platform_comps.utils.general.**convert_comps_workitem_status**(*comps_status: COMPS.Data.WorkItem.WorkIten* → *idmtools.core.enums.EntityStatus*

Convert status from COMPS to IDMTools Created = 0 # WorkItem has been saved to the database CommissionRequested = 5 # WorkItem is ready to be processed by the next available worker of the correct type Commissioned = 10 # WorkItem has been commissioned to a worker of the correct type and is beginning execution Validating = 30 # WorkItem is being validated Running = 40 # WorkItem is currently running Waiting = 50 # WorkItem is waiting for dependent items to complete ResumeRequested = 60 # Dependent items have completed and WorkItem is ready to be processed by the next available worker of the correct type CancelRequested = 80 # WorkItem cancellation was requested Canceled = 90 # WorkItem was successfully canceled Resumed = 100 # WorkItem has been claimed by a worker of the correct type and is resuming Canceling = 120 # WorkItem is in the process of being canceled by the worker Succeeded = 130 # WorkItem completed successfully Failed = 140 # WorkItem failed :param comps_status: Status in Comps

Returns EntityStatus

idmtools_platform_comps.utils.general.**clean_experiment_name**(*experiment_name: str*) → str

Enforce any COMPS-specific demands on experiment names. :param experiment_name: name of the experiment

Returns:the experiment name allowed for use

idmtools_platform_comps.utils.general.**get_file_from_collection**(*platform: idmtools.entities.iplatform.IPlatform*, *collection_id: uuid.UUID*, *file_path: str*) → bytearray

Retrieve a file from an asset collection

Parameters

- **platform** – Platform object to use
- **collection_id** – Asset Collection ID
- **file_path** – Path within collection

Examples:: >>> import uuid >>> get_file_from_collection(platform, uuid.UUID("fc461146-3b2a-441f-bc51-0bff3a9c1ba0"), "StdOut.txt")

Returns Object Byte Array

idmtools_platform_comps.utils.general.**get_file_as_generator**(*file: Union[COMPS.Data.SimulationFile.Simulati COMPS.Data.AssetCollectionFile.AssetColle COMPS.Data.AssetFile.AssetFile, COMPS.Data.WorkItemFile.WorkItemFile, COMPS.Data.OutputFileMetadata.OutputFil chunk_size: int = 128, resume_byte_pos: Optional[int] = None*) → Generator[bytearray, None, None]

Get file as a generator

Parameters

- **file** – File to stream contents through a generator
- **chunk_size** – Size of chunks to load
- **resume_byte_pos** – Optional start of download

Returns:

idmtools_platform_comps.utils.general.**get_asset_for_comps_item**(*platform: idmtools.entities.iplatform.IPlatform*, *item: idmtools.core.interfaces.ientity.IEntity*, *files: List[str]*, *cache=None*) → Dict[str, bytearray]

Retrieve assets from an Entity(Simulation, Experiment, WorkItem)

Parameters

- **platform** – Platform Object to use
- **item** – Item to fetch assets from
- **files** – List of file names to retrieve
- **cache** – Cache object to use

Returns Dictionary in structure of filename -> bytearray

idmtools_platform_comps.utils.lookups module

idmtools_platform_comps.utils.lookups.**get_experiment_by_id**(*exp_id*, *query_criteria: COMPS.Data.QueryCriteria.QueryCriteria = None*) → COMPS.Data.Experiment.Experiment

idmtools_platform_comps.utils.lookups.**get_simulation_by_id**(*sim_id*, *query_criteria: COMPS.Data.QueryCriteria.QueryCriteria = None*) → COMPS.Data.Simulation.Simulation

Fetches simulation by id and optional query criteria. Wrapped in additional Retry Logic. Used by other lookup methods

Parameters

- **sim_id** –
- **query_criteria** – Optional QueryCriteria to search with

Returns Simulation with ID

idmtools_platform_comps.utils.lookups.**get_all_experiments_for_user**(*user: str*) → List[COMPS.Data.Experiment.Exp

Returns all the experiments for a specific user

Parameters **user** – username to locate

Returns Experiments for a user

idmtools_platform_comps.utils.lookups.**get_simulations_from_big_experiments**(*experiment_id*)

Parameters **experiment_id** –

Returns:

idmtools_platform_comps.utils.package_version module

class idmtools_platform_comps.utils.package_version.**LinkHTMLParser**(*, *convert_charrefs=True*)

Bases: html.parser.HTMLParser

previous_tag = None

pkg_version = []

handle_starttag(*tag*, *attrs*)

idmtools_platform_comps.utils.package_version.**get_latest_package_version_from_pypi**(*pkg_name*, *display_all=F*

Utility to get the latest version for a given package name :param pkg_name: package name given :param display_all: determine if output all package releases

Returns: the latest version of ven package

idmtools_platform_comps.utils.package_version.**get_latest_package_version_from_artifactory**(*p*

Utility to get the latest version for a given package name :param pkg_name: package name given :param display_all: determine if output all package releases

Returns: the latest version of ven package

idmtools_platform_comps.utils.package_version.**get_latest_ssmt_image_version_from_artifacto**

Utility to get the latest version for a given package name :param pkg_name: package name given :param display_all: determine if output all package releases

Returns: the latest version of ven package

idmtools_platform_comps.utils.package_version.**get_latest_version_from_site**(*pkg_url*, *display_all=False*)

Utility to get the latest version for a given package name :param pkg_name: package name given :param display_all: determine if output all package releases

Returns: the latest version of ven package

idmtools_platform_comps.utils.python_version module

idmtools_platform_comps.utils.python_version.**platform_task_hooks**(*task*, *platform*)

Update task with new python command: python3 :param task: PythonTask or CommandTask :param platform: the platform user uses

Returns: re-build task

Module contents

Submodules

idmtools_platform_comps.comps_cli module

class idmtools_platform_comps.comps_cli.**CompsCLI**

Bases: idmtools_cli.iplatform_cli.IPlatformCLI

get_experiment_status(**args*, ***kwargs*) → NoReturn

Parameters

- **id** –
- **tags** –

Returns:

get_simulation_status(**args*, ***kwargs*) → NoReturn

Parameters

- **id** –
- **experiment_id** –
- **status** –
- **tags** –

Returns:

get_platform_information() → dict

class idmtools_platform_comps.comps_cli.**COMPSCLISpecification**

Bases: idmtools_cli.iplatform_cli.PlatformCLISpecification

get(*configuration: dict*) → *idmtools_platform_comps.comps_cli.CompsCLI*

Factor that should return a new platform using the passed in configuration :param configuration:

Returns:

get_additional_commands() → NoReturn

get_description() → str

Get a brief description of the plugin and its functionality.

Returns The plugin description.

idmtools_platform_comps.comps_platform module

class idmtools_platform_comps.comps_platform.**COMPSPriority**(*value*)
Bases: enum.Enum

An enumeration.

Lowest = 'Lowest'

BelowNormal = 'BelowNormal'

Normal = 'Normal'

AboveNormal = 'AboveNormal'

Highest = 'Highest'

class idmtools_platform_comps.comps_platform.**COMPSPlatform**(**args*, ***kwargs*)
Bases: *idmtools.entities.iplatform.IPlatform*, *idmtools.core.cache_enabled.CacheEnabled*

Represents the platform allowing to run simulations on COMPS.

MAX_SUBDIRECTORY_LENGTH = 35

endpoint: str = 'https://comps2.idmod.org'

environment: str = 'Bayesian'

priority: str = 'Lowest'

simulation_root: str = '$COMPS_PATH(USER)\\output'

node_group: str = None

num_retries: int = 0

num_cores: int = 1

max_workers: int = 16

batch_size: int = 10

exclusive: bool = False

docker_image: str = None

post_setstate()
Function called after restoring the state if additional initialization is required

idmtools_platform_comps.plugin_info module

class idmtools_platform_comps.plugin_info.**COMPSPlatformSpecification**
Bases: *idmtools.registry.platform_specification.PlatformSpecification*

get_description() → str
Get a brief description of the plugin and its functionality.

Returns The plugin description.

get(***configuration*) → *idmtools_platform_comps.comps_platform.COMPSPlatform*
Return a new platform using the passed in configuration.

Parameters **configuration** – The INI configuration file to use.

Returns The new platform.

example_configuration()
Example configuration for the platform. This is useful in help or error messages.

Returns:

get_type() → Type[*idmtools_platform_comps.comps_platform.COMPSPlatform*]

get_example_urls() → List[str]
Returns a list of URLs that a series of Examples for plugin can be downloaded from

Returns List of urls

class idmtools_platform_comps.plugin_info.**SSMTPlatformSpecification**
Bases: *idmtools.registry.platform_specification.PlatformSpecification*

get_description() → str
Get a brief description of the plugin and its functionality.

Returns The plugin description.

get(***configuration*) → *idmtools_platform_comps.comps_platform.COMPSPlatform*
Return a new platform using the passed in configuration.

Parameters **configuration** – The INI configuration file to use.

Returns The new platform.

example_configuration()
Example configuration for the platform. This is useful in help or error messages.

Returns:

get_type() → Type[*idmtools_platform_comps.ssmt_platform.SSMTPlatform*]

get_example_urls() → List[str]
Returns a list of URLs that a series of Examples for plugin can be downloaded from

Returns List of urls

idmtools_platform_comps.ssmt_platform module

class idmtools_platform_comps.ssmt_platform.**SSMTPlatform**(**args*, ***kwargs*)
Bases: *idmtools_platform_comps.comps_platform.COMPSPlatform*

Represents the platform allowing to run simulations on SSMT.

Module contents

idmtools_platform_local

idmtools_platform_local package

Subpackages

idmtools_platform_local.cli package

Submodules

idmtools_platform_local.cli.experiment module

idmtools_platform_local.cli.experiment.**prettify_experiment**(*experiment: Dict[str, Any]*)

Prettifies a JSON Experiment object for printing on a console. This includes - Making a pretty progress bar - URL-ifying the data paths - sorting the columns

Parameters **experiment** – JSON representation of the Experiment(from API)

Returns:

idmtools_platform_local.cli.experiment.**status**(*id: Optional[str], tags: Optional[List[Tuple[str, str]]]*)

List the status of experiment(s) with the ability to filter by experiment id and tags

Parameters

- **id** (*Optional[str]*) – Optional ID of the experiment you want to filter by
- **tags** (*Optional[List[Tuple[str, str]]]*) – Optional list of tuples in form of tag_name tag_value to user to filter experiments with

idmtools_platform_local.cli.experiment.**extra_commands**()

This ensures our local platform specific commands are loaded

idmtools_platform_local.cli.local module

class idmtools_platform_local.cli.local.**LocalCliContext**(*config=None*)

Bases: object

do: *idmtools_platform_local.infrastructure.docker_io.DockerIO* **= None**

sm: *idmtools_platform_local.infrastructure.service_manager.DockerServiceManager* **= Non**

idmtools_platform_local.cli.local.**cli_command_type**

alias of *idmtools_platform_local.cli.local.LocalCliContext*

idmtools_platform_local.cli.local.**stop_services**(*cli_context: idmtools_platform_local.cli.local.LocalCliContext, delete_data*)

idmtools_platform_local.cli.local.**container_status_text**(*name*, *container*)

idmtools_platform_local.cli.simulation module

idmtools_platform_local.cli.simulation.**prettify_simulation**(*simulation: Dict[str, Any]*)

Prettifies a JSON Simulation object for printing on a console. This includes - Making a pretty progress bar - URL-ifying the data paths

Parameters **simulation** – JSON representation of the Experiment(from API)

Returns:

idmtools_platform_local.cli.simulation.**status**(*id: Optional[str], experiment_id: Optional[str], status: Optional[str], tags: Optional[List[Tuple[str, str]]]*)

List of statuses for simulation(s) with the ability to filter by id, experiment_id, status, and tags

Parameters

- **id** (*Optional[str]*) – Optional Id of simulation
- **experiment_id** (*Optional[str]*) – Optional experiment id
- **status** (*Optional[str]*) – Optional status string to filter by
- **tag** (*Optional[List[Tuple[str, str]]]*) – Optional list of tuples in form of tag_name tag_value to user to filter experiments with

Returns None

idmtools_platform_local.cli.utils module

idmtools_platform_local.cli.utils.**get_service_info**(*service_manger:* idmtools_platform_local.infrastructure.service_manager.Docke *diff: bool*, *logs: bool*) → str

idmtools_platform_local.cli.utils.**colorize_status**(*status:* idmtools_platform_local.status.Status) → str

Colorizes a status for the console :param status: Status to colorize :type status: Status

Returns Unicode colorized string of the status

Return type str

idmtools_platform_local.cli.utils.**parent_status_to_progress**(*status: Dict[*idmtools_platform_local.status.Status*, int]*, *width: int = 12*) → str

Convert a status object into a colorized progress bar for the console

Parameters

- **status** (*Dict[Status, int]*) – Status dictionary. The dictionary should Status values for keys and the values should be the total number of simulations in the specific status. An example would be {Status.done: 30, Status.created: 1}
- **width** (*int*) – The desired width of the progress bar

Returns Progress bar of the status

Return type str

idmtools_platform_local.cli.utils.**urlize_data_path**(*path: str*) → str

URL-ize a data-path so it can be made click-able in the console(if the console supports it) :param path: path to urilze :type path: str

Returns Path as URL.

Return type str

Module contents

idmtools_platform_local.client package

Submodules

idmtools_platform_local.client.base module

class idmtools_platform_local.client.base.**BaseClient**
Bases: object

base_url = 'http://localhost:5000/api'

classmethod get(*path*, ***kwargs*) → requests.models.Response

classmethod post(*path*, ***kwargs*) → requests.models.Response

classmethod put(*path*, ***kwargs*) → requests.models.Response

classmethod delete(*path*, ***kwargs*) → requests.models.Response

idmtools_platform_local.client.experiments_client module

class idmtools_platform_local.client.experiments_client.**ExperimentsClient**
Bases: *idmtools_platform_local.client.base.BaseClient*

path_url = 'experiments'

classmethod get_all(*tags: Optional[List[Tuple[str, str]]] = None*, *page: Optional[int] = None*, *per_page: Optional[int] = None*) → List[Dict[str, Any]]
Get all experiments with options to filter by tags

Parameters

- **per_page** – How many experiments to return per page
- **page** – Which page
- **tags** (*Optional[List[Tuple[str, str]]]*) – List of tags/values to filter experiment by

Returns returns list of experiments

Return type List[Dict[str, Any]]

classmethod get_one(*id: str*, *tags: Optional[List[Tuple[str, str]]] = None*) → Dict[str, Any]
Convenience method to get one experiment

Parameters

- **id** (*str*) – ID of the experiment
- **tags** (*Optional[List[Tuple[str, str]]]*) – List of tags/values to filter experiment by

Returns Dictionary containing the experiment objects

Return type dict

classmethod delete(*id: str*, *delete_data: bool = False*, *ignore_doesnt_exist: bool = True*) → bool
Delete an experiment. Optionally you can delete the experiment data. WARNING: Deleting the data is irreversible

Parameters

- **id** (*str*) – ID of the experiments
- **delete_data** (*bool*) – Delete data directory including simulations
- **ignore_doesnt_exist** – Ignore error if the specific experiment doesn't exist

Returns True if deletion is succeeded

idmtools_platform_local.client.healthcheck_client module

class idmtools_platform_local.client.healthcheck_client.**HealthcheckClient**

Bases: *idmtools_platform_local.client.base.BaseClient*

path_url = 'healthcheck'

classmethod get_all() → List[Dict[str, Any]]

Get all experiments with options to filter by tags

Parameters

- **per_page** – How many experiments to return per page
- **page** – Which page
- **tags** (*Optional[List[Tuple[str, str]]]*) – List of tags/values to filter experiment by

Returns returns list of experiments

Return type List[Dict[str, Any]]

classmethod get_one() → Dict[str, Any]

Convenience method to get one experiment

Parameters

- **id** (*str*) – ID of the experiment
- **tags** (*Optional[List[Tuple[str, str]]]*) – List of tags/values to filter experiment by

Returns Dictionary containing the experiment objects

Return type dict

classmethod delete(**args*, ***kwargs*) → bool

classmethod post(**args*, ***kwargs*) → bool

idmtools_platform_local.client.simulations_client module

class idmtools_platform_local.client.simulations_client.**SimulationsClient**

Bases: *idmtools_platform_local.client.base.BaseClient*

path_url = 'simulations'

classmethod get_all(*experiment_id: Optional[str] = None*, *status: Optional[*idmtools_platform_local.status.Status*] = None*, *tags: Optional[List[Tuple[str, str]]] = None*, *page: Optional[int] = None*, *per_page: Optional[int] = None*) → List[Dict[str, Any]]

Parameters

- **id** (*Optional[str]*) – ID of the simulation
- **experiment_id** (*Optional[str]*) – ID of experiments
- **status** (*Optional[Status]*) – Optional status
- **tags** (*Optional[List[Tuple[str, str]]]*) – List of tags/values to filter experiment by

Returns return list of simulations

Return type List[Dict[str, Any]]

classmethod get_one(*simulation_id: str*, *experiment_id: Optional[str] = None*, *status: Optional[*idmtools_platform_local.status.Status*] = None*, *tags: Optional[List[Tuple[str, str]]] = None*) → Dict[str, Any]

Args: simulation_id (str): ID of the simulation experiment_id (Optional[str]): ID of experiments status (Optional[Status]): Optional status tags (Optional[List[Tuple[str, str]]]): List of tags/values to filter experiment by

Returns the simulation as a dict

Return type Dict[str, Any]

classmethod cancel(*simulation_id: str*) → Dict[str, Any]

Marks a simulation to be canceled. Canceled jobs are only truly canceled when the queue message is processed

Parameters **simulation_id** (*st*) –

Returns:

Module contents

idmtools_platform_local.infrastructure package

Submodules

idmtools_platform_local.infrastructure.base_service_container module

class idmtools_platform_local.infrastructure.base_service_container.**BaseServiceContainer**(*co*
st
=
N
im
ag
st
=
N
cl
do
=
N
co
fig
st
=
N
ne
wo
st
=
N

Bases: abc.ABC

container_name: str = None

image: str = None

client: docker.client.DockerClient = None

config_prefix: str = None

network: str = None

static get_common_config(*container_name: str*, *image: str*, *network: str*, *port_bindings: Optional[Dict] = None*, *volumes: Optional[Dict] = None*, *mem_limit: Optional[str] = None*, *mem_reservation: Optional[str] = None*, *environment: Optional[List[str]] = None*, *extra_labels: Optional[Dict] = None*, ***extras*) → dict

Returns portions of docker container configs that are common between all the different containers used within our platform

Parameters

- **mem_limit** (*Optional[str]*) – Limit memory
- **mem_reservation** (*Optional[str]*) – Reserve memory

Returns:

Notes

Memory strings should match those used by docker. See –memory details at https://docs.docker.com/engine/reference/run/#runtime-constraints-on-resources

get() → Optional[docker.models.containers.Container]

get_or_create(*spinner=None*) → docker.models.containers.Container
Get or Create a container

Parameters **spinner** – Optional spinner to display

Returns Docker container object representing service container

static ensure_container_is_running(*container: docker.models.containers.Container*) → docker.models.containers.Container
Ensures is running :param container:

Returns:

create(*spinner=None*) → docker.models.containers.Container

static wait_on_status(*container*, *sleep_interval: float = 0.2*, *max_time: float = 2*, *statutes_to_wait_for: List[str] = None*)

stop(*remove=False*)

restart()

get_logs()

abstract get_configuration() → Dict

idmtools_platform_local.infrastructure.docker_io module

class idmtools_platform_local.infrastructure.docker_io.**DockerIO**(*host_data_directory: str = '/home/docs/.local_data'*)

Bases: object

host_data_directory: str = '/home/docs/.local_data'

delete_files_below_level(*directory*, *target_level=1*, *current_level=1*)

cleanup(*delete_data: bool = True*, *shallow_delete: bool = False*) → NoReturn
Stops the running services, removes local data, and removes network. You can optionally disable the deleting of local data

Parameters

- **delete_data** (*bool*) – When true, deletes local data
- **shallow_delete** (*bool*) – Deletes the data but not the container folders(redis, workers). Preferred to preserve permissions and resolve docker issues

Returns (NoReturn)

copy_to_container(*container: docker.models.containers.Container*, *destination_path: str*, *file: Optional[Union[str, bytes]] = None*, *content: [<class 'str'>, <class 'bytes'>] = None*, *dest_name: Optional[str] = None*) → bool
Copies a physical file or content in memory to a container. You can also choose a different name for the destination file by using the dest_name option

Parameters

- **container** – Container to copy the file to
- **file** – Path to the file to copy
- **content** – Content to copy
- **destination_path** – Path within the container to copy the file to(should be a directory)
- **dest_name** – Optional parameter for destination filename. By default the source filename is used

Returns (bool) True if the copy succeeds, False otherwise

sync_copy(*futures*)

Sync the copy operations queue in the io_queue. This allows us to take advantage of multi-threaded copying while also making it convenient to have sync points, such as uploading the assets in parallel but pausing just before sync point

Parameters **futures** –

Returns:

copy_multiple_to_container(*container: docker.models.containers.Container*, *files: Dict[str, Dict[str, Any]]*, *join_on_copy: bool = True*)

static create_archive_from_bytes(*content: Union[bytes, _io.BytesIO, BinaryIO]*, *name: str*) → _io.BytesIO

Create a tar archive from bytes. Used to copy to docker

Parameters

- **content** – Content to copy into tar
- **name** – Name for file in archive

Returns (BytesIO) Return bytesIO object

create_directory(*dir: str*) → bool

Create a directory in a container

Parameters

- **dir** – Path to directory to create
- **container** – Container to create directory in. Default to worker container

Returns (ExecResult) Result of the mkdir operation

idmtools_platform_local.infrastructure.postgres module

class idmtools_platform_local.infrastructure.postgres.**PostgresContainer**(*container_name: str = 'idmtools_postgres', image: str = 'postgres:11.4', client: docker.client.DockerClient = None, config_prefix: str = 'postgres_', network: str = None, host_data_directory: str = None, port: int = 5432, mem_limit: str = '128m', mem_reservation: str = '32m', run_as: str = None, password: str = 'idmtools', data_volume_name: str = 'idmtools_local_postgres'*)

Bases: *idmtools_platform_local.infrastructure.base_service_container.*

BaseServiceContainer

host_data_directory: str = None

port: int = 5432

mem_limit: str = '128m'

mem_reservation: str = '32m'

run_as: str = None

image: str = 'postgres:11.4'

container_name: str = 'idmtools_postgres'

password: str = 'idmtools'

data_volume_name: str = 'idmtools_local_postgres'

config_prefix: str = 'postgres_'

get_configuration() → Dict
Returns the docker config for the postgres container

Returns (dict) Dictionary representing the docker config for the postgres container

create(*spinner=None*) → docker.models.containers.Container

create_postgres_volume() → NoReturn
Creates our postgres volume Returns:

idmtools_platform_local.infrastructure.redis module

class idmtools_platform_local.infrastructure.redis.**RedisContainer**(*container_name: str = 'idmtools_redis', image: str = 'redis:5.0.4-alpine', client: docker.client.DockerClient = None, config_prefix: str = 'redis_', network: str = None, host_data_directory: str = None, mem_limit: str = '256m', mem_reservation: str = '64m', run_as: str = None, port: int = 6379, data_volume_name: str = None*)

Bases: *idmtools_platform_local.infrastructure.base_service_container.BaseServiceContainer*

host_data_directory: str = None

mem_limit: str = '256m'

mem_reservation: str = '64m'

run_as: str = None

port: int = 6379

image: str = 'redis:5.0.4-alpine'

data_volume_name: str = None

container_name: str = 'idmtools_redis'

config_prefix: str = 'redis_'

get_configuration() → dict

idmtools_platform_local.infrastructure.service_manager module

class idmtools_platform_local.infrastructure.service_manager.**DockerServiceManager**(*client: docker.client*
host_data_d
str = */home/docs/*
net-
work: str = *'idm-tools'*,
re-
dis_image: str = *'redis:5.0.4-alpine'*,
heart-
beat_timeou
int = *15*,
re-
dis_port: int = *6379*,
run-
time: Union[str, None-Type] = *'runc'*,
re-
dis_mem_lim
str = *'256m'*,
re-
dis_mem_res
str = *'32m'*,
post-
gres_image: str = *'post-gres:11.4'*,
post-
gres_mem_li
str = *'128m'*,
post-
gres_mem_r

client: DockerClient

host_data_directory: str = '/home/docs/.local_data'

network: str = 'idmtools'

redis_image: str = 'redis:5.0.4-alpine'

heartbeat_timeout: int = 15

redis_port: int = 6379

runtime: Optional[str] = 'runc'

redis_mem_limit: str = '256m'

redis_mem_reservation: str = '32m'

postgres_image: str = 'postgres:11.4'

postgres_mem_limit: str = '128m'

postgres_mem_reservation: str = '32m'

postgres_port: Optional[str] = 5432

workers_image: str = None

workers_ui_port: int = 5000

workers_mem_limit: str = None

workers_mem_reservation: str = '64m'

run_as: Optional[str] = None

init_services()

cleanup(*delete_data: bool = False*, *tear_down_brokers: bool = False*) → NoReturn

Stops the containers and removes the network. Optionally the postgres data container can be deleted as well as closing any active Redis connections

Parameters

- **delete_data** – Delete postgres data
- **tear_down_brokers** – True to close redis brokers, false otherwise

Returns NoReturn

static setup_broker(*heartbeat_timeout*)

static restart_brokers(*heartbeat_timeout*)

create_services(*spinner=None*) → NoReturn

Create all the components of our

Our architecture is as depicted in the UML diagram below

Returns (NoReturn)

wait_on_ports_to_open(*ports: List[str]*, *wait_between_tries: Union[int, float] = 0.2*, *max_retries: int = 5*, *sleep_after: Union[int, float] = 0.5*) → bool

Polls list of port attributes(eg postgres_port, redis_port and checks if they are currently open. We use this to verify postgres/redis are ready for our workers

Parameters

- **ports** – List of port attributes
- **wait_between_tries** – Time between port checks
- **max_retries** – Max checks
- **sleep_after** – Sleep after all our found open(Postgres starts accepting connections before actually ready)

Returns True if ports are ready

stop_services(*spinner=None*) → NoReturn
Stops all running IDM Tools services

Returns (NoReturn)

get(*container_name: str*, *create=True*) → docker.models.containers.Container
Get the server with specified name

Parameters

- **container_name** – Name of container
- **create** – Create if it doesn't exists

Returns:

get_container_config(*service:* idmtools_platform_local.infrastructure.base_service_container.BaseServiceContainer, *opts=None*)
Get the container config for the service

Parameters

- **service** – Service to get config for
- **opts** – Opts to Extract. Should be a fields object

Returns:

restart_all(*spinner=None*) → NoReturn
Restart all the services IDM-Tools services

Returns (NoReturn)

static is_port_open(*host: str*, *port: int*) → bool
Check if a port is open

Parameters

- **host** – Host to check
- **port** – Port to check

Returns True if port is open, False otherwise

static stop_service_and_wait(*service*) → bool
Stop server and wait

Parameters **service** – Service to stop

Returns:

get_network() → docker.models.networks.Network
Fetches the IDM Tools network

Returns (Network) Return Docker network object

idmtools_platform_local.infrastructure.workers module

idmtools_platform_local.infrastructure.workers.**get_worker_image_default**()

class idmtools_platform_local.infrastructure.workers.**WorkersContainer**(*container_name: str = 'idmtools_workers', image: str = 'docker-production.packages.idmod.org client: docker.client.DockerClient = None, config_prefix: str = 'workers_', network: str = None, host_data_directory: str = None, postgres_port: int = 5432, redis_port: int = 6379, ui_port: int = 5000, mem_limit: str = '16g', mem_reservation: str = '64m', run_as: str = None, debug_api: bool = True, data_volume_name: str = None*)

Bases: *idmtools_platform_local.infrastructure.base_service_container.*

BaseServiceContainer

host_data_directory: str = None

postgres_port: int = 5432

redis_port: int = 6379

ui_port: int = 5000

mem_limit: str = '16g'

mem_reservation: str = '64m'

run_as: str = None

debug_api: bool = True

image: str = 'docker-production.packages.idmod.org/idmtools/local_workers:1.4.0'

container_name: str = 'idmtools_workers'

data_volume_name: str = None

config_prefix: str = 'workers_'

get_configuration() → Dict

create(*spinner=None*) → docker.models.containers.Container

Module contents

idmtools_platform_local.internals package

Subpackages

idmtools_platform_local.internals.data package

Submodules

idmtools_platform_local.internals.data.job_status module

class idmtools_platform_local.internals.data.job_status.**JobStatus**(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base

Generic status table. At moment we only have one which contains both experiments and simulations We do it this way to allow for more flexible support in future for non-emod-ish workflows(ie a bunch of single jobs instead of an experiment with sub simulations

uuid

parent_uuid

status

data_path

tags

extra_details

created

updated

to_dict(*as_experiment=True*)

Module contents

idmtools_platform_local.internals.tasks package

idmtools_platform_local.internals.ui package

Subpackages

idmtools_platform_local.internals.ui.controllers package

Submodules

idmtools_platform_local.internals.ui.controllers.experiments module

idmtools_platform_local.internals.ui.controllers.experiments.**progress_to_status_str**(*progress*)

idmtools_platform_local.internals.ui.controllers.experiments.**handle_backoff_exc**(*details*)

idmtools_platform_local.internals.ui.controllers.experiments.**experiment_filter**(*id: Optional[str], tags: Optional[List[Tuple[str]]], page: int = 1, per_page: int = 10*) → Tuple[Dict, int]

List the status of experiment(s) with the ability to filter by experiment id and tags

Parameters

- **id** (*Optional[str]*) – Optional ID of the experiment you want to filter by
- **tags** (*Optional[List[Tuple[str, str]]]*) – Optional list of tuples in form of tag_name tag_value to user to filter experiments with
- **page** (*int*) – Which page to load. Defaults to 1
- **per_page** (*int*) – Experiments per page. Defaults to 50

class idmtools_platform_local.internals.ui.controllers.experiments.**Experiments**

Bases: flask_restful.Resource

get(*id=None*)

delete(*id*)

endpoint = 'experiments'

mediatypes()

methods = {'DELETE', 'GET'}

idmtools_platform_local.internals.ui.controllers.healthcheck module

class idmtools_platform_local.internals.ui.controllers.healthcheck.**HealthCheck**
Bases: flask_restful.Resource

get()

endpoint = 'healthcheck'

mediatypes()

methods = {'GET'}

idmtools_platform_local.internals.ui.controllers.simulations module

idmtools_platform_local.internals.ui.controllers.simulations.**sim_status**(*id: Optional[str], experiment_id: Optional[str], status: Optional[str], tags: Optional[List[Tuple[str, str]]], page: int = 1, per_page: int = 20*) → Tuple[Dict, int]

List of statuses for simulation(s) with the ability to filter by id, experiment_id, status, and tags

Parameters

- **id** (*Optional[str]*) – Optional Id of simulation
- **experiment_id** (*Optional[str]*) – Optional experiment id
- **status** (*Optional[str]*) – Optional status string to filter by
- **tags** (*Optional[List[Tuple[str, str]]]*) – Optional list of tuples in form of tag_name tag_value to user to filter experiments with
- **page** (*int*) – Which page to load. Defaults to 1
- **per_page** (*int*) – Simulations per page. Defaults to 50

Returns None

class idmtools_platform_local.internals.ui.controllers.simulations.**Simulations**

Bases: flask_restful.Resource

get(*id=None*)

put(*id*)

endpoint = 'simulations'

mediatypes()

methods = {'GET', 'PUT'}

idmtools_platform_local.internals.ui.controllers.utils module

idmtools_platform_local.internals.ui.controllers.utils.**validate_tags**(*tags*)

Module contents

Submodules

idmtools_platform_local.internals.ui.app module

idmtools_platform_local.internals.ui.app.**autoindex**(*path='.'*)

idmtools_platform_local.internals.ui.config module

idmtools_platform_local.internals.ui.config.**start_db**(*db=None*)

idmtools_platform_local.internals.ui.utils module

class idmtools_platform_local.internals.ui.utils.**DateTimeEncoder**(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)

Bases: flask.json.JSONEncoder

default(*o*)

Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    return JSONEncoder.default(self, o)
```

Module contents

idmtools_platform_local.internals.workers package

Submodules

idmtools_platform_local.internals.workers.brokers module

idmtools_platform_local.internals.workers.database module

idmtools_platform_local.internals.workers.database.**create_db**(*engine*)

idmtools_platform_local.internals.workers.database.**get_session**() → sqlalchemy.orm.session.Session

idmtools_platform_local.internals.workers.database.**get_db**() → sqlalchemy.engine.base.Engine

idmtools_platform_local.internals.workers.database.**reset_db**()

idmtools_platform_local.internals.workers.database.**get_or_create**(*session: sqlalchemy.orm.session.Session*, *model*, *filter_args: List[str]*, ***model_args*)

idmtools_platform_local.internals.workers.run module

idmtools_platform_local.internals.workers.run_broker module

idmtools_platform_local.internals.workers.utils module

idmtools_platform_local.internals.workers.utils.**create_or_update_status**(*uuid*, *data_path=None*, *tags=None*, *status=<Status.created: 'created'>*, *parent_uuid=None*, *extra_details=None*, *session=None*, *autoclose=True*, *autocommit=True*)

idmtools_platform_local.internals.workers.utils.**get_host_data_bind**()

Module contents

Module contents

idmtools_platform_local.platform_operations package

Submodules

idmtools_platform_local.platform_operations.experiment_operations module

class idmtools_platform_local.platform_operations.experiment_operations.**LocalPlatformExperi**

Bases: *idmtools.entities.iplatform_ops.iplatform_experiment_operations.IPlatformExperimentOperations*

platform: 'LocalPlatform'

platform_type
alias of *idmtools_platform_local.platform_operations.uitils.ExperimentDict*

get(*experiment_id: uuid.UUID*, ***kwargs*) → *idmtools_platform_local.platform_operations.uitils.ExperimentDict*
Get the experiment object by id

Parameters

- **experiment_id** – Id
- ****kwargs** –

Returns Experiment Dict object

platform_create(*experiment:* idmtools.entities.experiment.Experiment, ***kwargs*) → Dict
Create an experiment.

Parameters

- **experiment** – Experiment to create
- ****kwargs** –

Returns Created experiment object and UUID

get_children(*experiment: Dict*, ***kwargs*) → List[*idmtools_platform_local.platform_operations.uitils.SimulationDict*]
Get children for an experiment

Parameters

- **experiment** – Experiment to get chidren for
- ****kwargs** –

Returns List of simulation dicts

get_parent(*experiment: Any*, ***kwargs*) → None
Experiment on local platform have no parents so return None

Parameters

- **experiment** –
- ****kwargs** –

Returns:

platform_run_item(*experiment:* idmtools.entities.experiment.Experiment, ***kwargs*)
Run the experiment

Parameters **experiment** – experiment to run

Returns:

send_assets(*experiment:* idmtools.entities.experiment.Experiment, ***kwargs*)
Sends assets for specified experiment

Parameters **experiment** – Experiment to send assets for

Returns:

refresh_status(*experiment:* idmtools.entities.experiment.Experiment, ***kwargs*)
Refresh status of experiment

Parameters **experiment** – Experiment to refresh status for

Returns:

static from_experiment(*experiment:* idmtools.entities.experiment.Experiment) → Dict
Create a experiment dictionary from Experiment object

Parameters **experiment** – Experiment object

Returns Experiment as a local platform dict

to_entity(*experiment: Dict, children: bool = True, **kwargs*) → *idmtools.entities.experiment.Experiment*
Convert an ExperimentDict to an Experiment

Parameters

- **experiment** – Experiment to convert
- ****kwargs** –

Returns object as an IExperiment object

list_assets(*experiment:* idmtools.entities.experiment.Experiment, ***kwargs*) → List[*idmtools.assets.asset.Asset*]
List assets for a sim

Parameters **experiment** – Experiment object

Returns:

idmtools_platform_local.platform_operations.simulation_operations module

class idmtools_platform_local.platform_operations.simulation_operations.**LocalPlatformSimula**

Bases: *idmtools.entities.iplatform_ops.iplatform_simulation_operations.IPlatformSimulationOperations*

platform: 'LocalPlatform'

platform_type
alias of *idmtools_platform_local.platform_operations.uitils.SimulationDict*

get(*simulation_id: uuid.UUID*, ***kwargs*) → Dict
Fetch simulation with specified id :param simulation_id: simulation id :param **kwargs:

Returns SimulationDIct

platform_create(*simulation:* idmtools.entities.simulation.Simulation, ***kwargs*) → Dict
Create a simulation object

Parameters

- **simulation** – Simulation to create
- ****kwargs** –

Returns Simulation dict and created id

batch_create(*sims: List[*idmtools.entities.simulation.Simulation*]*, ***kwargs*) → List[*idmtools_platform_local.platform_operations.uitils.SimulationDict*]
Batch creation of simulations.

This is optimized by bulk uploading assets after creating of all the assets

Parameters

- **sims** – List of sims to create
- ****kwargs** –

Returns List of SimulationDict object and their IDs

get_parent(*simulation:* idmtools_platform_local.platform_operations.uitils.SimulationDict, ***kwargs*) → *idmtools_platform_local.platform_operations.uitils.ExperimentDict*
Get the parent of a simulation, aka its experiment

Parameters

- **simulation** – Simulation to get parent from
- ****kwargs** –

Returns ExperimentDict object

platform_run_item(*simulation:* idmtools.entities.simulation.Simulation, ***kwargs*)
On the local platform, simulations are ran by queue and commissioned through create :param simulation:

Returns:

send_assets(*simulation:* idmtools.entities.simulation.Simulation, *worker: docker.models.containers.Container = None*, ***kwargs*)
Transfer assets to local sim folder for simulation

Parameters

- **simulation** – Simulation object
- **worker** – docker worker containers. Useful in batches

Returns:

refresh_status(*simulation:* idmtools.entities.simulation.Simulation, ***kwargs*)
Refresh status of a sim

Parameters **simulation** –

Returns:

get_assets(*simulation:* idmtools.entities.simulation.Simulation, *files: List[str]*, ***kwargs*) → Dict[str, bytearray]

Get assets for a specific simulation

Parameters

- **simulation** – Simulation object to fetch files for
- **files** – List of files to fetch

Returns Returns a dict containing mapping of filename->bytearry

list_assets(*simulation:* idmtools.entities.simulation.Simulation, ***kwargs*) → List[*idmtools.assets.asset.Asset*]

List assets for a sim

Parameters **simulation** – Simulation object

Returns:

to_entity(*local_sim: Dict*, *load_task: bool = False*, *parent: Optional[*idmtools.entities.experiment.Experiment*] = None*, ***kwargs*) → *idmtools.entities.simulation.Simulation*

Convert a sim dict object to an ISimulation

Parameters

- **local_sim** – simulation to convert
- **load_task** – Load Task Object as well. Can take much longer and have more data on platform
- **parent** – optional experiment object
- ****kwargs** –

Returns ISimulation object

idmtools_platform_local.platform_operations.uitils module

class idmtools_platform_local.platform_operations.uitils.**ExperimentDict**

Bases: dict

class idmtools_platform_local.platform_operations.uitils.**SimulationDict**

Bases: dict

idmtools_platform_local.platform_operations.uitils.**local_status_to_common**(*status*) → EntityStatus

Convert local platform status to idmtools status :param status:

Returns:

idmtools_platform_local.platform_operations.uitils.**download_lp_file**(*filename: str*, *buffer_size: int = 128*) → Generator[bytes, None, None]

Returns a generator to download files on the local platform :param filename: :param buffer_size:

Returns:

Module contents

Submodules

idmtools_platform_local.config module

idmtools_platform_local.config.**get_api_path**()

idmtools_platform_local.local_cli module

class idmtools_platform_local.local_cli.**LocalCLI**

Bases: idmtools_cli.iplatform_cli.IPlatformCLI

get_experiment_status(*id: Optional[str]*, *tags: Optional[List[Tuple[str, str]]]*) → NoReturn

Parameters

- **id** –
- **tags** –

Returns:

get_simulation_status(*id: Optional[str]*, *experiment_id: Optional[str]*, *status: Optional[str]*, *tags: Optional[List[Tuple[str, str]]]*) → NoReturn

Parameters

- **id** –
- **experiment_id** –
- **status** –
- **tags** –

Returns:

get_platform_information(*platform: LocalPlatform*) → dict

class idmtools_platform_local.local_cli.**LocalCLISpecification**

Bases: idmtools_cli.iplatform_cli.PlatformCLISpecification

get(*configuration: dict*) → *idmtools_platform_local.local_cli.LocalCLI*

Factor that should return a new platform using the passed in configuration :param configuration:

Returns:

get_additional_commands() → NoReturn

get_description() → str
Get a brief description of the plugin and its functionality.

Returns The plugin description.

idmtools_platform_local.local_platform module

class idmtools_platform_local.local_platform.**LocalPlatform**(**args*, ***kwargs*)
Bases: *idmtools.entities.iplatform.IPlatform*

Represents the platform allowing to run simulations locally.

host_data_directory: str = '/home/docs/.local_data'

network: str = 'idmtools'

redis_image: str = 'redis:5.0.4-alpine'

redis_port: int = 6379

runtime: Optional[str] = None

redis_mem_limit: str = '128m'

redis_mem_reservation: str = '64m'

postgres_image: str = 'postgres:11.4'

postgres_mem_limit: str = '64m'

postgres_mem_reservation: str = '32m'

postgres_port: Optional[str] = 5432

workers_mem_limit: str = '16g'

workers_mem_reservation: str = '128m'

workers_image: str = None

workers_ui_port: int = 5000

heartbeat_timeout: int = 15

default_timeout: int = 45

launch_created_experiments_in_browser: bool = False

auto_remove_worker_containers: bool = True

cleanup(*delete_data: bool = False*, *shallow_delete: bool = False*, *tear_down_brokers: bool = False*)

post_setstate()
Function called after restoring the state if additional initialization is required

idmtools_platform_local.plugin_info module

class idmtools_platform_local.plugin_info.**LocalPlatformSpecification**
Bases: *idmtools.registry.platform_specification.PlatformSpecification*

get_description() → str
Get a brief description of the plugin and its functionality.

Returns The plugin description.

get(***configuration*) → *idmtools.entities.iplatform.IPlatform*
Build our local platform from the passed in configuration object

We do our import of platform here to avoid any weir :param configuration:

Returns:

example_configuration()
Example configuration for the platform. This is useful in help or error messages.

Returns:

get_type() → Type[LocalPlatform]

idmtools_platform_local.status module

class idmtools_platform_local.status.**Status**(*value*)
Bases: enum.Enum

Our status enum for jobs

created = 'created'

in_progress = 'in_progress'

canceled = 'canceled'

failed = 'failed'

done = 'done'

Module contents

1.10 User Recipes

1.10.1 Asset Collections

Modifying Asset Collection

```
# This recipes demos how to extend/modify and existing AssetCollection
from idmtools.assets import AssetCollection, Asset
from idmtools.core.platform_factory import Platform

with Platform("COMPS2") as platform:
    # first we start by loading our existing asset collection
    existing_ac = AssetCollection.from_id("98d329b5-95d6-ea11-a2c0-f0921c167862")
    # now we want to add one file to it. Since asset collection on the server our
↪immutable, what we can do is the following
```

(continues on next page)

(continued from previous page)

```
    #
    # create a new asset collection object
    ac = AssetCollection(existing_ac)
    # or
    # ac = AssetCollection.from_id("98d329b5-95d6-ea11-a2c0-f0921c167862", as_
↪copy=True)
    # ac = existing_ac.copy()
    # ac = AssetCollection()
    # ac += existing_ac
    # add our items to the new collection
    ac.add_asset(Asset(filename="Example", content="Blah"))

    # then depending on the workflow, we can create directly or use within an␣
↪Experiment/Task/Simulation
    platform.create_items(ac)

    # Experiment
    # e = Experiment.from_task(..., assets=ac)

    # Task
    # task = CommandTask(common_assets = ac)
    # or
    # task.common_assets = ac
```

1.11 CLI reference

1.11.1 Templates

You can use the cookiecutter templates included with idmtools to get started with python projects and idmtools. These templates provide a logical, reasonably standardized, but flexible project structure for doing and sharing data science work. To see the list of included cookiecutter templates type the following at a command prompt.

```
$ idmtools init --help
INI File Used: /home/docs/checkouts/readthedocs.org/user_builds/institute-for-disease-
↪modeling-idmtools/checkouts/v1.4.0/docs/idmtools.ini
Usage: idmtools init [OPTIONS] COMMAND [ARGS]...

  Commands to help start or extend projects through templating.

Options:
  --help  Show this message and exit.

Commands:
  data-science          A logical, reasonably standardized, but flexible...
  docker-science        This project is a tiny template for machine
                        learning...

  reproducible-science  A boilerplate for reproducible and transparent...
```

1.11.2 Simulations

You can use the `simulation` command to get the status of simulations for the local platform. To see the list of options type the following at a command prompt.

```
$ idmtools simulation --platform Local status --help
INI File Used: /home/docs/checkouts/readthedocs.org/user_builds/institute-for-disease-
↪modeling-idmtools/checkouts/v1.4.0/docs/idmtools.ini
Usage: idmtools simulation status [OPTIONS]

  List of statuses for simulation(s) with the ability to filter by id,
  experiment_id, status, and tags

  For Example Get the status of simulations for the platform using the local
  platform defaults, you would run idmtools simulation --platform Local
  status

  Another example would be to use a platform defined in a configuration
  block while also filtering tags where a == 0 idmtools simulation --config-
  block COMPS2 status --tags a 0

  Multiple tags idmtools simulation --config-block COMPS2 status --tags a 0
  --tags a 3

Options:
  --id TEXT             Filter status by simulation ID
  --experiment-id TEXT  Filter status by experiment ID
  --tags TEXT...        Tag to filter by. This should be in the form name
                        value. For example, if you have a tag type=PythonTask
                        you would use --tags type PythonTask. In addition, you
                        can provide multiple tags, ie --tags a 1 --tags b 2.
                        This will perform an AND based query on the tags
                        meaning only jobs contains ALL the tags specified will
                        be displayed

  --help                Show this message and exit.
```

1.11.3 Experiments

You can use the `experiment` command to get the status of and to delete experiments for the local platform. Local platform must be running to use these commands. To see the list of commands and options for `status`, type the following at a command prompt.

```
$ idmtools experiment --platform Local status --help
INI File Used: /home/docs/checkouts/readthedocs.org/user_builds/institute-for-disease-
↪modeling-idmtools/checkouts/v1.4.0/docs/idmtools.ini
Usage: idmtools experiment status [OPTIONS]

  List the status of experiment(s) with the ability to filter by experiment
  id and tags

  Some examples: Get the status of simulations for the platform using the
  local platform defaults, you would run

  idmtools simulation --platform Local status
```

(continues on next page)

(continued from previous page)

```
  Another example would be to use a platform defined in a configuration
  block while also filtering tags where a == 0

  idmtools experiment --config-block COMPS2 status --tags a 0

  Multiple tags:

  idmtools experiment --config-block COMPS2 status --tags a 0 --tags a 3

Options:
  --id TEXT       Filter status by experiment ID
  --tags TEXT...  Tag to filter by. This should be in the form name value. For
                  example, if you have a tag type=PythonTask you would use
                  --tags type PythonTask. In addition, you can provide
                  multiple tags, ie --tags a 1 --tags b 2. This will perform
                  an AND based query on the tags meaning only jobs contains
                  ALL the tags specified will be displayed

  --help          Show this message and exit.
```

To see the list of commands and options for `delete`, type the following at a command prompt.

```
$ idmtools experiment --platform Local delete --help
INI File Used: /home/docs/checkouts/readthedocs.org/user_builds/institute-for-disease-
↪modeling-idmtools/checkouts/v1.4.0/docs/idmtools.ini
Usage: idmtools experiment delete [OPTIONS] EXPERIMENT_ID

  Delete an experiment, and optionally, its data

Options:
  --data / --no-data  Should we delete the data as well?
  --help              Show this message and exit.
```

1.11.4 Platforms

IDM includes commands for managing the local platform. To see the list of commands type the following at a command prompt.

```
$ idmtools local --help
INI File Used: /home/docs/checkouts/readthedocs.org/user_builds/institute-for-disease-
↪modeling-idmtools/checkouts/v1.4.0/docs/idmtools.ini
Usage: idmtools local [OPTIONS] COMMAND [ARGS]...

  Commands related to managing the local platform

Options:
  --run-as TEXT  Change the default user you run docker containers as. Useful
                 is situations where you need to access docker with sudo.
                 Example values are "1000:1000"

  --help         Show this message and exit.

Commands:
  down     Shutdown the local execution platform(and optionally delete data
  info
```

(continues on next page)

(continued from previous page)

```
  restart  Restart the local execution platform
  start    Start the local execution platform
  status   Check the status of the local execution platform
  stop
```

The platform settings are contained in the idmtools.ini file. For more information, see *Configuration*.

1.11.5 Examples

You can use IDM CLI to download the included Python example scripts from GitHub to a local folder using the `gitrepo` command. To see the list of commands and options for `gitrepo`, type the following at a command prompt:

```
$ idmtools gitrepo --help
INI File Used: /home/docs/checkouts/readthedocs.org/user_builds/institute-for-disease-
↪modeling-idmtools/checkouts/v1.4.0/docs/idmtools.ini
Usage: idmtools gitrepo [OPTIONS] COMMAND [ARGS]...

Options:
  --help  Show this message and exit.

Commands:
  download   Download files from GitHub repo to user location Args: url:...
  peep       Display all current files/dirs of the repo folder (not...
  releases   Display all the releases of the repo Args: owner: Repo owner...
  repos      Display all public repos of the owner Args: owner: Repo owner...
  view       Display all idmtools available examples Args: raw: True/False...
```

or view examples by type through

```
$ idmtools examples list
INI File Used: /home/docs/checkouts/readthedocs.org/user_builds/institute-for-disease-
↪modeling-idmtools/checkouts/v1.4.0/docs/idmtools.ini

COMPSPlatform
    - https://github.com/InstituteforDiseaseModeling/idmtools/tree/v1.4.0/examples/
↪ssmt
    - https://github.com/InstituteforDiseaseModeling/idmtools/tree/v1.4.0/examples/
↪workitem
    - https://github.com/InstituteforDiseaseModeling/idmtools/tree/v1.4.0/examples/
↪vistools

SSMTPlatform
    - https://github.com/InstituteforDiseaseModeling/idmtools/tree/v1.4.0/examples/
↪ssmt
    - https://github.com/InstituteforDiseaseModeling/idmtools/tree/v1.4.0/examples/
↪vistools

PythonTask
    - https://github.com/InstituteforDiseaseModeling/idmtools/tree/v1.4.0/examples/
↪load_lib

CommandTask
    - https://github.com/InstituteforDiseaseModeling/corvid-idmtools
```

(continues on next page)

(continued from previous page)

```
JSONConfiguredRTask
    - https://github.com/InstituteforDiseaseModeling/idmtools/tree/v1.4.0/examples/r_
↪model

JSONConfiguredTask
    - https://github.com/InstituteforDiseaseModeling/idmtools/tree/v1.4.0/examples/
↪python_model
    - https://github.com/InstituteforDiseaseModeling/idmtools/tree/v1.4.0/examples/
↪load_lib
```

To see the list of commands and options for downloading examples, type the following at a command prompt:

```
$ idmtools gitrepo download --help
INI File Used: /home/docs/checkouts/readthedocs.org/user_builds/institute-for-disease-
↪modeling-idmtools/checkouts/v1.4.0/docs/idmtools.ini
Usage: idmtools gitrepo download [OPTIONS]

  Download files from GitHub repo to user location
  Args:
      url: GitHub repo files url
      output: Local folder

  Returns: Files download count

Options:
  --type TEXT    Download examples by type(COMPSPlatform, PythonTask, etc)
  --url TEXT     Repo files url
  --output TEXT  Files download destination
  --help         Show this message and exit.
```

or

```
$ idmtools examples download --help
INI File Used: /home/docs/checkouts/readthedocs.org/user_builds/institute-for-disease-
↪modeling-idmtools/checkouts/v1.4.0/docs/idmtools.ini
Usage: idmtools examples download [OPTIONS]

  Download examples from specified location
  Args:
      url: GitHub repo files url
      output: Local folder

  Returns: Files download count

Options:
  --type TEXT    Download examples by type(COMPSPlatform, PythonTask, etc)
  --url TEXT     Repo files url
  --output TEXT  Files download destination
  --help         Show this message and exit.
```

To see a list of IDM examples available for downloading, type `idmtools gitrepo download` at a command prompt.

1.11.6 Troubleshooting

You can use troubleshooting commands to get information abouts plugins (CLI, Platform, and Task) and to get detailed system information. To see the list of troubleshooting commands, type the following at a command prompt:

```
$ idmtools info --help
INI File Used: /home/docs/checkouts/readthedocs.org/user_builds/institute-for-disease-
↪modeling-idmtools/checkouts/v1.4.0/docs/idmtools.ini
Usage: idmtools info [OPTIONS] COMMAND [ARGS]...

  Troubleshooting and debugging information

Options:
  --help  Show this message and exit.

Commands:
  plugins  Commands to get information about installed IDM-Tools plugins
  system   Provide an output with details about your current execution...
```

To see the list of troubleshooting commands and options for the `plugins` command, type the following at a command prompt:

```
$ idmtools info plugins --help
INI File Used: /home/docs/checkouts/readthedocs.org/user_builds/institute-for-disease-
↪modeling-idmtools/checkouts/v1.4.0/docs/idmtools.ini
Usage: idmtools info plugins [OPTIONS] COMMAND [ARGS]...

  Commands to get information about installed IDM-Tools plugins

Options:
  --help  Show this message and exit.

Commands:
  cli       List CLI plugins
  platform  List Platform plugins
  task      List Task plugins
```

To see the list of troubleshooting options for the `system` command, type the following at a command prompt:

```
$ idmtools info system --help
INI File Used: /home/docs/checkouts/readthedocs.org/user_builds/institute-for-disease-
↪modeling-idmtools/checkouts/v1.4.0/docs/idmtools.ini
Usage: idmtools info system [OPTIONS]

  Provide an output with details about your current execution platform and
  IDM-Tools install

Options:
  --copy-to-clipboard / --no-copy-to-clipboard
                                  Copy output to clipboard
  --no-format-for-gh / --format-for-gh
                                  When copying to clipboard, do we want to
                                  formatted for Github

  --issue / --no-issue            Copy data and format for github alias
  --output-filename TEXT          Output filename
  --help                          Show this message and exit.
```

IDM includes a command-line interface (CLI) with options and commands to assist with getting started, managing and monitoring, and troubleshooting simulations and experiments. After you've installed IDM you can view the available options and commands by typing the following at a command prompt

```
$ idmtools --help
INI File Used: /home/docs/checkouts/readthedocs.org/user_builds/institute-for-disease-
↪modeling-idmtools/checkouts/v1.4.0/docs/idmtools.ini
Usage: idmtools [OPTIONS] COMMAND [ARGS]...

  Allows you to perform multiple idmtools commands

Options:
  --debug / --no-debug  When selected, enables console level logging
  --help                Show this message and exit.

Commands:
  config       Contains commands related to the creation of idmtools.ini...
  examples     Display a list of examples organized by plugin type
  experiment   Contains commands related to experiments Some useful
               examples...

  gitrepo
  info         Troubleshooting and debugging information
  init         Commands to help start or extend projects through templating.
  init-export  Export list of project templates
  local        Commands related to managing the local platform
  simulation   Contains commands related to simulations Some useful
               examples...
```

1.12 Glossary

The following terms describe both the features and functionality of the idmtools software, as well as information relevant to using idmtools.

analyzer Functionality that uses the MapReduce framework to process large data sets in parallel, typically on a *high-performance computing (HPC)* cluster. For example, if you would like to focus on specific data points from all simulations in one or more experiments then you can do this using analyzers with idmtools and plot the final output.

asset collection A collection of user created input files, such as demographics, temperature, weather, binaries, and overlay files. These files are stored in COMPS and can be available for use by other users.

assets See *asset collection*.

builder A function and list of values with which to call that function that is used to sweep through parameter values in a simulation.

calibration The process of adjusting the parameters of a simulation to better match the data from a particular time and place.

EMOD An agent-based mechanistic disease transmission model built by IDM that can be used with idmtools. See the EMOD GitHub repo.

entity Each of the interfaces or classes that are well-defined models, types, and validations for idmtools items, such as simulations, analyzers, or tasks.

experiment Logical grouping of simulations. This allows for managing numerous simulations as a single unit or grouping.

high-performance computing (HPC) The use of parallel processing for running advanced applications efficiently, reliably, and quickly.

parameter sweep An iterative process in which simulations are run repeatedly using different values of the parameter(s) of choice. This process enables the modeler to determine what a parameter's "best" value or range of values.

platform The computing resource on which the simulation runs. See *Platforms* for more information on those that are currently supported.

server-side modeling tools (SSMT) Modeling tools used with COMPS that handle computation on the server side, rather than the client side, to speed up analysis.

simulation An individual run of a model. Generally, multiple simulations are run as part of an experiement.

suite Logical grouping of experiments. This allows for managing multiple experiments as a single unit or grouping.

task The individual actions that are processed for each simulation.

1.13 Changelog

1.13.1 0.1.0

Analyzers

- #0060 - Analyzer base class

Bugs

- #0095 - idmtools is not working for python 3.6
- #0096 - pytest (and pytest-runner) should be installed by setup
- #0105 - UnicodeDecodeError when run python example in LocalPlatform mode
- #0114 - It should be possible to set *base_simulation* in the *PythonExperiment* constructor
- #0115 - *PythonSimulation* constructor should abstract the *parameters* dict
- #0124 - Can not run teststest_python_simulation.py from console
- #0125 - relative_path for AssetCollection does not work
- #0126 - Same test in issue #125 does not working for localPlatform
- #0129 - new python model root node changed from "config" to "parameters"
- #0137 - PythonExperiment fails if pass assets
- #0138 - test_sir.py does not set parameter
- #0142 - experiment.batch_simulations seems not to be batching
- #0143 - COMPSPlatform's refresh_experiment_status() get called too much from ExperimentManager's wait_till_done() mathod
- #0150 - missing pandas package
- #0151 - log throw error from IPersistanceService.py's save method
- #0161 - tests/test_python_simulation.py's test_add_dirs_to_assets_comps() return different asset files for windows and linux

- #0171 - Workflow: fix loop detection
- #0203 - Running new builds on Linux fails in Bamboo due to datapostgres-data file folder permissions
- #0206 - test_python_simulation.py failed for all local test in windows

CLI

- #0007 - Command line functions definition
- #0118 - Add the printing of children in the EntityContainer

Configuration

- #0047 - Configuration file read on a per-folder basis
- #0048 - Validation for the configuration file
- #0049 - Configuration file is setting correct parameters in platform

Core

- #0006 - Service catalog
- #0014 - Package organization and pre-requisites
- #0081 - Allows the sweeps to be created in arms
- #0087 - Raise an exception if we have 2 files with the same relative path in the asset collection
- #0091 - Refactor the Experiment/Simulation objects to not persist the simulations
- #0092 - Generalize the simulations/experiments for Experiment/Suite
- #0102 - [Local Runner] Retrieve simulations for experiment
- #0107 - LocalPlatform does not detect duplicate files in AssetCollectionFile for pythonExperiment
- #0140 - Fetch simulations at runtime
- #0148 - Add python tasks
- #0180 - switch prettytable for tabulate

Documentation

- #0004 - Notebooks exploration for examples
- #0085 - Setup Sphinx and GitHub pages for the docs
- #0090 - “Development installation steps” missing some steps

Models

- #0008 - Which models support out of the box?
- #0136 - Create an envelope argument for the PythonSimulation

Platforms

- #0068 - [Local Runner] Simulation status monitoring
- #0069 - [Local Runner] Database
- #0094 - Batch and parallelize simulation creation in the COMPSPlatform

1.13.2 1.0.0

Analyzers

- #0034 - Create the Plotting step
- #0057 - Output files retrieval
- #0196 - Filtering
- #0197 - Select_simulation_data
- #0198 - Finalize
- #0279 - Port dtk-tools analyze system to idmtools
- #0283 - Fix up all platform-based test due to analyzer/platform refactor/genericization
- #0337 - Change AnalyzeManager to support passing ids (Experiment, Simulation, Suite)
- #0338 - Two AnalyzeManager files - one incorrect and needs to be removed
- #0340 - Cleanup DownloadAnalyzer
- #0344 - AnalyzeManager configuration should be option parameter
- #0589 - Rename suggestion: example_analysis_multiple_cases => example_analysis_MultipleCases
- #0592 - analyzers error on platform.get_files for COMPS: argument of type 'NoneType' is not iterable
- #0594 - analyzer error multiprocessing pool StopIteration error in finalize_results
- #0614 - Convenience function to exclude items in analyze manager
- #0619 - Ability to get exp sim object ids in analyzers

Bugs

- #0124 - Can not run teststest_python_simulation.py from console
- #0125 - relative_path for AssetCollection does not work
- #0129 - new python model root node changed from "config" to "parameters"
- #0142 - experiment.batch_simulations seems not to be batching
- #0143 - COMPSPlatform's refresh_experiment_status() get called too much from ExperimentManager's wait_till_done() mathod

- #0150 - missing pandas package
- #0184 - Missing 'data' dir for test_experiment_manager test. (TestPlatform)
- #0223 - UnicodeDecodeError for testcases in test_dtk.py when run with LocalPlatform
- #0236 - LocalRunner: ExperimentsClient get_all method should have parameter 'tags' not 'tag'
- #0265 - load_files for DTKExperiment create nested 'parameters' in config.json
- #0266 - load_files for demographics.json does not work
- #0272 - diskcache objects cause cleanup failure if used in failing processes
- #0294 - Docker containers failed to start if they are created but stopped
- #0299 - Sometime in Windows command line, local docker runner stuck and no way to stop from command line
- #0302 - Local Platform delete is broken
- #0318 - Postgres Connection error on Local Platform
- #0320 - COMPSPlatform Asset handling - currently DuplicatedAssetError content is not same
- #0323 - idmtools is not retro-compatible with pre-idmtools experiments
- #0332 - with large number of simulations, local platform either timeout on dramatiq or stuck on persistamce-Service save method
- #0339 - Analyzer tests fails on AnalyzeManager analyze len(self.potential_items) == 0
- #0341 - AnalyzeManager Runtime error on worker_pool
- #0346 - UnknownItemException for analyzers on COMPSPlatform PythonExperiments
- #0350 - RunTask in local platform should catch exception
- #0351 - AnalyzeManager finalize_results Process cannot access the cache.db because it is being used by another process
- #0352 - Current structure of code leads to circular dependencies or classes as modules
- #0367 - Analyzer does not work with reduce method with no hashable object
- #0375 - AnalyzerManager does not work for case to add experiment to analyzermanager
- #0376 - AnalyzerManager does not work for simulation
- #0378 - experiment/simulation display and print are messed up in latest dev
- #0386 - Local platform cannot create more than 20 simulations in a given experiment
- #0398 - Ensure that redis and postgres ports work as expected
- #0399 - PopulaionAnalyzer does not return all items in reduce mathod in centos platform
- #0424 - ExperimentBuilder's add_sweep_definition is not flexible enough to take more parameters
- #0427 - Access to the experiment object in analyzers
- #0453 - cli: "idmtools local down –delete-data" not really delete any .local_data in user default dir
- #0458 - There is no way to add custom tags to simulations
- #0465 - BuilderExperiment for sweep "string" is wrong
- #0545 - pymake docker-local always fail in centos
- #0553 - BLOCKING: idmtools_model_r does not get built with make setup-dev
- #0560 - docker-compose build does not work for r-model example

- #0562 - workflow_item_operations get workitem querycriteria fails
- #0564 - typing is missing in asset_collection.py which almost break every tests
- #0565 - missing 'copy' in local_platform.py
- #0566 - test_tasks.py fail for case test_command_is_required
- #0567 - 'platform_supports' is missing for test_comps_plugin.py in idmtools_platform_comps/tests
- #0570 - webui for localhost:5000 got 403 error
- #0572 - python 3.7.3 less version will fail for task type changing
- #0585 - print(platform) throws exception for Python 3.6
- #0588 - Running the dev installation in a virtualenv "installs" it globally
- #0598 - CSVAnalyzer pass wrong value to parse in super().__init__ call
- #0602 - Analyzer doesn't work for my Python SEIR model
- #0605 - When running multiple analyzers together, 'data' in one analyzer should not contains data from other analyzer
- #0606 - can not import cached_property
- #0608 - Cannot add custom tag to AssetCollection in idmtools
- #0613 - idmtools webui does not working anymore
- #0616 - AssetCollection pre_creation failed if no tag
- #0617 - AssetCollection's find_index_of_asset is wrong
- #0618 - analyzer-manager should fail if map status return False
- #0641 - Remove unused code in the python_requirements_ac
- #0644 - Platform cannot run workitem directly
- #0646 - platform.get_items(ac) not return tags
- #0667 - analyzer_manager could stuck on _run_and_wait_for_reducing

CLI

- #0009 - Boilerplate command
- #0118 - Add the printing of children in the EntityContainer
- #0187 - Move the CLI package to idmtools/cli
- #0190 - Add a platform attribute to the CLI commands
- #0191 - Create a PlatformFactory
- #0241 - CLI should be distinct package and implement as plugins
- #0251 - Setup for the CLI package should provide a entrypoint for easy use of commands
- #0252 - Add –debug to cli main level

Configuration

- #0248 - Logging needs to support user configuration through the idmtools.ini
- #0392 - Improve IdmConfigParser: make decorator for ensure_ini() method...
- #0597 - Platform should not be case sensitive.

Core

- #0032 - Create NextPointAlgorithm Step
- #0042 - Stabilize the IStep object
- #0043 - Create the generic Workflow object
- #0044 - Implement validation for the Steps of a workflow based on Marshmallow
- #0058 - Filtering system for simulations
- #0081 - Allows the sweeps to be created in arms
- #0091 - Refactor the Experiment/Simulation objects to not persist the simulations
- #0141 - Standard Logging throughout tools
- #0169 - Handle 3.6 requirements automatically
- #0172 - Decide what state to store for tasks
- #0173 - workflows: Decide on state storage scheme
- #0174 - workflows: Reimplement state storage
- #0175 - workflows: Create unit tests of core classes and behaviors
- #0176 - workflows: reorganize files into appropriate repo/directory
- #0180 - switch prettytable for tabulate
- #0200 - Platforms should be plugins
- #0238 - Simulations of Experiment should be made pickle ignored
- #0244 - Inputs values needs to be validated when creating a Platform
- #0257 - CsvExperimentBuilder does not handle csv field with empty space
- #0268 - demographics filenames should be loaded to asset collection
- #0274 - Unify id attribute naming scheme
- #0281 - Improve Platform to display selected Block info when creating a platform
- #0297 - Fix issues with platform factory
- #0308 - idmtools: Module names should be consistent
- #0315 - Basic support of suite in the tools
- #0357 - ExperimentPersistService.save are not consistent
- #0359 - SimulationPersistService is not used in Idmtools
- #0361 - assets in Experiment should be made “pickle-ignore”
- #0362 - base_simulation in Experiment should be made “pickle-ignore”
- #0368 - PersistService should support clear() method

- #0369 - The method create_simulations of Experiment should consider pre-defined max_workers and batch_size in idmtools.ini
- #0370 - Add unit test for deepcopy on simulations
- #0371 - Wrong type for platform_id in IEntity definition
- #0391 - Improve Asset and AssetCollection classes by using @dataclass (field) for clear comparison
- #0394 - Remove the ExperimentPersistService
- #0438 - Support pulling Eradication from URLs and bamboo
- #0518 - Add a task class.
- #0520 - Rename current experiment builders to sweep builders
- #0526 - Create New Generic Experiment Class
- #0527 - Create new Generic Simulation Class
- #0528 - Remove old Experiments/Simulations
- #0529 - Create New Task API
- #0530 - Rename current model api to simulation/experiment API.
- #0538 - Refactor platform interface into subinterfaces
- #0681 - idmtools should have way to query comps with filter

Developer/Test

- #0631 - Ensure setup.py is consistent throughout

Documentation

- #0100 - Installation steps documented for users
- #0312 - idmtools: there is a typo in README
- #0360 - The tools should refer to “EMOD” not “DTK”
- #0474 - Stand alone builder
- #0486 - Overview of the analysis in idmtools
- #0510 - Local platform options
- #0512 - SSMT platform options
- #0578 - Add installation for users
- #0593 - Simple Python SEIR model demo example
- #0632 - Update idmtools_core setup.py to remove model emod from idm install

Feature Request

- #0061 - Built-in DownloadAnalyzer
- #0064 - Support of CSV files
- #0070 - [Local Runner] Output files serving
- #0233 - Add local runner timeout
- #0437 - Prompt users for docker credentials when not available
- #0603 - Implement install custom requirement libs to asset collection with WorkItem

Models

- #0021 - Python model
- #0024 - R Model support
- #0053 - Support of demographics files
- #0212 - Models should be plugins
- #0287 - Add info about support models/docker support to platform
- #0288 - Create DockerExperiment and subclasses
- #0519 - Move experiment building to ExperimentBuilder
- #0521 - Create Generic Dictionary Config Task
- #0522 - Create PythonTask
- #0523 - Create PythonDictionaryTask
- #0524 - Create RTask
- #0525 - Create EModTask
- #0535 - Create DockerTask

Platforms

- #0025 - LOCAL Platform
- #0027 - SSMT Platform
- #0094 - Batch and parallelize simulation creation in the COMPSPlatform
- #0122 - Ability to create an AssetCollection based on a COMPS asset collection id
- #0130 - User configuration and data storage location
- #0186 - The *local_runner* client should move to the *idmtools* package
- #0194 - COMPS Files retrieval system
- #0195 - LOCAL Files retrieval system
- #0221 - Local runner for experiment/simulations have different file hierarchy than COMPS
- #0254 - Local Platform Asset should be implemented via API or Docker socket
- #0264 - idmtools_local_runner's tasks/run.py should have better handle for unhandled exception

- #0276 - Docker services should be started for end-users without needing to use docker-compose
- #0280 - Generalize sim/exp/suite format of ISimulation, IExperiment, IPlatform
- #0286 - Add special GPU queue to Local Platform
- #0305 - Create a website for local platform
- #0306 - AssetCollection's assets_from_directory logic wrong if set flatten and relative path at same time
- #0313 - idmtools: MAX_SUBDIRECTORY_LENGTH = 35 should be made Global in COMPSPlatform definition
- #0314 - Fix local platform to work with latest analyze/platform updates
- #0316 - Integrate website with Local Runner Container
- #0321 - COMPSPlatform _retrieve_experiment errors on experiments with and without suites
- #0329 - Experiment level status
- #0330 - Paging on simulation/experiment APIs for better UI experience
- #0333 - ensure pyComps allows compatible releases
- #0364 - Local platform should use production artfactory for docker images
- #0381 - Support Work Items in COMPS Platform
- #0387 - Local platform webUI only show simulations up to 20
- #0393 - local platform tests keep getting EOFError while logger is in DEBUG and console is on
- #0405 - Support analysis of data from Work Items in Analyze Manager
- #0407 - Support Service Side Analysis through SSMT
- #0447 - Set limitation for docker container's access to memory
- #0532 - Make updates to ExperimentManager/Platform to support tasks
- #0540 - Create initial SSMT Plaform from COMPS Platform
- #0596 - COMPSPlatform.get_files(item,..) not working for Experiment or Suite
- #0635 - Update SSMT base image
- #0639 - Add a way for the python_requirements_ac to use additional wheel file
- #0676 - ssmt missing QueryCriteria support
- #0677 - ssmt: refresh_status returns None

User Experience

- #0457 - Option to analyze failed simulations

1.13.3 1.0.1

Analyzers

- #0778 - Add support for context platforms to analyzer manager

Bugs

- #0637 - pytest: ValueError: I/O operation on closed file, Printed at the end of tests.
- #0663 - SSMT PlatformAnalysis can not put 2 analyzers in same file as main entry
- #0696 - Rename num_retires to num_retries on COMPS Platform
- #0702 - Can not analyze workitem
- #0739 - Logging should load defaults with default config block is missing
- #0741 - MAX_PATH issues with RequirementsToAssetCollection WI create_asset_collection
- #0752 - type hint in analyzer_manager is wrong
- #0758 - Workitem config should be validated on WorkItem for PythonAsset Collection
- #0776 - Fix hook execution order for pre_creation
- #0779 - Additional Sims is not being detected on TemplatedSimulations
- #0788 - Correct requirements on core
- #0791 - Missing asset file with RequirementsToAssetCollection

Core

- #0343 - Genericize experiment_factory to work for other items
- #0611 - Consider excluding idmtools.log and COMPS_log.log on SSMT WI submission
- #0737 - Remove standalone builder in favor of regular python

Developer/Test

- #0083 - Setup python linting for the Pull requests
- #0671 - Python Linting
- #0735 - Tag or remove local tests in idmtools-core tests
- #0736 - Mark set of smoke tests to run in github actions
- #0773 - Move model-emod to new repo
- #0794 - build idmtools_platform_local fail with idmtools_webui error

Documentation

- #0015 - Add cookiecutter projects
- #0423 - Create a clear document on what features are provided by what packages
- #0473 - Create sweep without builder
- #0476 - ARM builder
- #0477 - CSV builder
- #0478 - YAML builder
- #0487 - Creation of an analyzer
- #0488 - Base analyzer - Constructor
- #0489 - Base analyzer - Filter function
- #0490 - Base analyzer - Parsing
- #0491 - Base analyzer - Working directory
- #0492 - Base analyzer - Map function
- #0493 - Base analyzer - Reduce function
- #0494 - Base analyzer - per group function
- #0495 - Base analyzer - Destroy function
- #0496 - Features of AnalyzeManager - Overview
- #0497 - Features of AnalyzeManager - Partial analysis
- #0498 - Features of AnalyzeManager - Max items
- #0499 - Features of AnalyzeManager - Working directory forcing
- #0500 - Features of AnalyzeManager - Adding items
- #0501 - Built-in analyzers - InsetChart analyzer
- #0502 - Built-in analyzers - CSV Analyzer
- #0503 - Built-in analyzers - Tags analyzer
- #0504 - Built-in analyzers - Download analyzer
- #0508 - Logging and Debugging
- #0509 - Global parameters
- #0511 - COMPS platform options
- #0629 - Update docker endpoint on ssmt/local platform to use external endpoint for pull/running
- #0630 - Investigate packaging idmtools as wheel file
- #0714 - Document the Versioning details
- #0717 - Sweep Simulation Builder
- #0720 - Documentation on Analyzing Failed experiments
- #0721 - AddAnalyer should have example in its self documentation
- #0722 - CSVAnalyzer should have example in its self documentation
- #0723 - DownloadAnalyzer should have example in its self documentation

- #0724 - PlatformAnalysis should have explanation of its used documented
- #0727 - SimulationBuilder Sweep builder documentation
- #0734 - idmtools does not install dataclasses on python3.6
- #0751 - Switch to apidoc generated RSTs for modules and remove from source control

Feature Request

- #0059 - Chaining of Analyzers
- #0097 - Ability to batch simulations within simulation
- #0704 - Tthere is no way to load custom wheel using the RequirementsToAssets utility
- #0784 - Remove default node_group value 'emod_abcd' from platform
- #0786 - Improve Suite support

Platforms

- #0277 - Need way to add tags to COMPSPlatform ACs after creation
- #0638 - Change print statement to logger in python_requirements_ac utility
- #0640 - Better error reporting when the python_requirements_ac fails
- #0651 - A user should not need to specify the default SSMT image
- #0688 - Load Custom Library Utility should support install packages from Artifactory
- #0705 - Should have way to regenerate AssetCollection id from RequirementsToAssetCollection
- #0757 - Set PYTHONPATH on Slurm

User Experience

- #0760 - Email for issues and feature requests
- #0781 - Suites should support run on object
- #0787 - idmtools should print experiment id by default in console

1.13.4 1.1.0

Additional Changes

- #0845 - Sprint 1 Retrospective Results

Bugs

- #0430 - test_docker_operations.test_port_taken_has_coherent_error fails in Linux VM with no host machine
- #0650 - analyzer_manager.py _run_and_wait_for_mapping fail frequently in bamboo
- #0706 - Correct the number of simulations being submitted in the progress bar
- #0846 - Checking for platform not installed
- #0872 - python executable is not correct for slurm production

CLI

- #0342 - Add list of task to cli
- #0543 - develop idm cookie cutter templates needs
- #0820 - Add examples url to plugins specifications and then each plugin if they have examples
- #0869 - CLI: idmtools gitrepo view - CommandTask points to /corvid-idmtools

Core

- #0273 - Add kwargs functionality to CacheEnabled
- #0818 - Create Download Examples Core Functionality
- #0828 - Add a master plugin registry

Developer/Test

- #0652 - Packing process should be fully automated
- #0731 - Add basic testing to Github Actions to Pull Requests
- #0785 - Add a miniconda agent to the bamboo testing of idmtools
- #0833 - Add emodpy to idm and full extra installs in core
- #0844 - For make setup-dev, we may want put login to artifactory first

Documentation

- #0729 - Move local platform worker container to Github Actions
- #0814 - High Level Diagram of Packages/Repos for idmtools
- #0858 - Fix doc publish to ghpages
- #0861 - emodpy - add updated api diagram (API class specifications) to architecture doc

Platforms

- #0728 - Restructure local platform docker container build for Github Action
- #0730 - Move SSMT Image build to github actions
- #0826 - SSMT Build as part of GithubActions

User Experience

- #0010 - Configuration file creation command
- #0684 - Create process for Changelog for future releases
- #0819 - Create Download Examples CLI Command
- #0821 - Provide plugin method to get Help URLs for plugin

1.13.5 1.2.0

Bugs

- #0859 - After install idmtools, still can not find model 'idmtools'
- #0873 - Task Plugins all need a get_type
- #0877 - Change RequirementsToAssetCollection to link AssetCollection and retrieve Id more reliability
- #0881 - With CommandTask, experiment must have an asset to run
- #0882 - CommandTask totally ignores common_assets
- #0893 - CommandTask: with transient asset hook, it will ignore user's transient_assets

Developer/Test

- #0885 - Platform to lightly execute tasks locally to enable better testing of Task life cycle

Documentation

- #0482 - Running experiments locally
- #0768 - Update breadcrumbs for docs
- #0860 - Create .puml files for UML doc examples within docs, add new files to existing .puml in diagrams directory, link to files
- #0867 - Examples - document cli download experience for example scripts
- #0870 - CLI - update documentation to reflect latest changes
- #0875 - Enable JSON Documentation Builds on Help for future Help Features
- #0889 - Parameter sweeps with EMOD
- #0896 - Add version to docs build
- #0903 - Add version to documentation

Feature Request

- #0832 - Implement underlying API needed for reload_from_simulation
- #0876 - Add option to optionally rebuild tasks on reload
- #0883 - Add new task type TemplateScriptTask to support Templated Scripts

Platforms

- #0692 - Get Docker Public Repo naming aligned with others

User Experience

- #0713 - Move all user output to customer logger

1.13.6 1.3.0

Bugs

- #0921 - PlatformAnalysis requires login before execution
- #0937 - RequirementsToAssetCollection fail with Max length
- #0946 - Upgrade pycomps to 2.3.7
- #0972 - Template script wrapper task should proxy calls where possible
- #0984 - Make idmtools_metadata.json default to off

Documentation

- #0481 - Overview of the local platform
- #0483 - Monitoring local experiments
- #0910 - Add documentation on plotting analysis output using matplotlib as an example
- #0925 - Platform Local - add documentation (getting started, run example, etc)
- #0965 - Add Analysis Output Format Support Table
- #0969 - Create base documentation for creating a new platform plugin

Feature Request

- #0830 - Support for python 3.8
- #0924 - YamlSimulationBuilder should accept a single function to be mapped to all values

Models

- #0834 - Add a COVASIM example with idmtools

Platforms

- #0852 - Add emodpy to SSMT image

User Experience

- #0682 - Support full query criteria on COMPS items

PYTHON MODULE INDEX

i

INDEX

A

B

C

D

E

F

G

J

L

M

N

O

P

S

T

U

V

W

Y