Stalker Documentation

Release 0.1.1.a2

Erkan Ozgur Yilmaz

January 13, 2011

CONTENTS

CHAPTER

ONE

ABOUT

Stalker is a Production Digital Asset Management (ProdAM) System designed specially for Animation and VFX Studios and licensed under BSD License.

Features:

- Platform independent
- Flexible design
- Designed for animation and vfx studios
- Default installation handles nearly all the asset management needs of an animation and vfx studio
- Customisable with configuration scripts
- Customisable object model (Stalker Object Model - SOM)
- Integrated messaging system
- Integrated production planing and tracking tools
- PySide user interfaces
- Can be used with any kind of database backend that SQLAlchmey supports
- Can be connected to all the major 3d animation packages like Maya, Houdini, Nuke, XSI, Vue, Blender etc. and any application that has a Python API
- Can work as a stand-alone application

Stalker is build over these other OpenSource projects:

- Python
- SQLAlchemy
- Jinja2
- Beaker
- PySide

1.1 Source

The latest development version is available in Google Project page of Stalker or can be directly cloned with the following command if you already have mercurial installed:

```
hg clone https://stalker.googlecode.com/hg/ stalker
```

1.2 Table of Contents

1.2.1 Installation

How to Install Stalker

This document will let you install and run Stalker.

Install Python

Stalker is completely written with Python, so it requires Python. It currently works with Python version 2.5 to 2.7. So you first need to have Python installed in your system. On Linux and OSX there is a system wide Python already installed. For Windows, you need to download the Python installer suitable for your Windows operating system (32 or 64 bit) from Python.org

Install Stalker

The easiest way to install the latest version of Stalker along with all its dependencies is to use the *setuptools*. If your system doesn't have setuptools (particularly Windows) you need to install *setuptools* by using *ez_setup* bootstrap script.

Installing *setuptools* with *ez_setup*:

These steps are generally needed just for Windows. Linux and OSX users can skip this part.

1. download ez_setup.py
2. run the following command in the command prompt/shell/terminal:

   ```
   python ez_setup
   ```

   It will install or build the *setuptools* if there are no suitable installer for your operating system.

After installing the *setuptools* you can run the following command:

```
easy_install -U stalker
```

Now you have installed Stalker along with all its dependencies.

Checking the installation of Stalker

If everything went ok you should be able to import and check the version of Stalker by using the Python prompt like this:

```
>>> import stalker
>>> stalker.__version__
0.1.1
```

For developers

Developers can clone the latest development version of Stalker from Google Code. Use the following command to clone:

```
hg clone https://stalker.googlecode.com/hg/ stalker
```

Developers also need to install these Ptyhon packages:

1. Nose
2. Coverage
3. Mocker
4. Sphinx
5. Pygments

The following command will install them all:

```
easy_install nose coverage mocker sphinx pygments
```

Installing a Database

Stalker uses a database to store all the values in to. The only database backend that doesn't require any extra installation is SQLite3. You can setup Stalker to run with an SQLite3 database. But it is much suitable to have a dedicated database server in your studio.

See the SQLAlchemy documentation for supported databases.

1.2.2 Tutorial

Introduction

Using Stalker is all about interacting with a database by using the Stalker Object Model. Stalker uses the powerfull SQLAlchemy ORM.

This tutorial section let you familiarise with the Stalker Python API and Stalker Object Model (SOM). If you used SQLAlchemy before you will feel at home and if you aren't you will see that it is fun dealing with databases with SOM.

Part I - Basics

Lets say that we just installed Stalker (as you are right now) and want to use Stalker in our first project.

The first thing we are going to learn about is the to how to connect to the database (default in-memory database nothing else is setup) so we can enter information about our studio.

We are going to use the helper script to connect to the default database. Use the following command to connect to the database:

```
from stalker import db
db.setup()
```

This will create an in-memory SQLite3 database, which is useless other than testing purposes. To be able to get more out of Stalker we should give a proper database information. The basic setup is to use a file based SQLite3 database:

```
db.setup("sqlite:///M:\\studio.db") # assumed Windows
```

This command will do the following:

1. setup the database connection, by creating an engine
2. create the SQLite3 database file if doesn't exist
3. create a session instance
4. do the mapping

Lets continue by creating a user for yourself in the database. The first thing we need to do is to import the `User` class in to the current namespace:

```
from stalker.core.models.user import User
```

then create the `User` object:

```
myUser = User(first_name="Erkan Ozgur",
              last_name="Yilmaz",
              login_name="eoyilmaz",
              email="eoyilmaz@gmail.com",
              password="secret",
              description="This is me")
```

Then lets add a new `Department` object to define your department:

```
from satlker.core.models.department import Department
tds_department = Department(name="TDs",
                            description="This is the TDs department")
```

Now add your user to the department:

```
tds_department.members.append(myUser)
```

We have created succesfully a `User` and a `Department` and we assigned the user as one of the member of the *TDs Department*.

For now, because we didn't tell Stalker to commit the changes, no data is saved to the database. But it doesn't keep us using Stalker, as if these information are in the database already. Lets show this by querying all the departments, then getting the first one and gettings its first members name:

```
print db.query(Department).first().members[0].first_name
```

this should print out "Erkan Ozgur". So even though we didn't commit the data to the database, Stalker lets us use the `db.query` to get objects out of the database.

So lets send all these data to database:

```
db.session.add(myUser)
db.session.commit()
```

If you noticed, we have just added `user1` to the database and Stalker will add all the connected object (like the department) to the database too.

Part II - Getting Hot

Lets say that we have this new project comming and you want to start using Stalker with it, lets create a `Project` object for it:

```
from stalker.core.models.project import Project
new_project = Project(name="Fancy Commercial")
```

Lets enter more information about this new project:

```
from datetime import datetime
from stalker.core.models.imageFormat import ImageFormat

new_project.description = """The commercial is about this fancy product. The
                             client want us to have a shinny look with their
                             product bla bla bla..."""
new_project.image_format = ImageFormat(name="HD 1080", width=1920, height=1080)
new_project.fps = 25
new_project.due = datetime(2011,2,15)
new_project.lead = myUser
```

You can group projects by their types, by using a `ProjectType` object:

```
from stalker.core.models.types import ProjectType

commercial_project_type = ProjectType(name="Commercial")
new_project.type = commercial_project_type
```

To save all the data to the database:

```
db.session.add(new_commercial)
db.session.commit()
```

Again we've just added the `new_project` object to the database but Stalker is smart enough to add all the connected objects to it.

Project objects contains `Sequence`s, so lets create one:

```
from stalker.core.models.sequence import Sequence
seq1 = Sequence(name="Sequence 1", code="SEQ1")

# add it to the project
new_project.sequences.append(seq1)
```

And Sequences contains `Shot`s:

```
from stalker.core.models.shot import Shot

sh001 = Shot(name="Shot 1", code="SH001")
sh002 = Shot(name="Shot 2", code="SH002")
sh003 = Shot(name="Shot 3", code="SH003")

# assign them to the sequence
seq1.shots.extend([sh001, sh002, sh003])
```

Part III - Pipeline

Infact we skipped a lot of stuff here to take little steps every time, for example Stalker doesn't know much about the pipeline of those shots.

To simply specify the *Pipeline* we should create a couple of `PipelineStep`s and assign them to an `AssetType` to group the same kind of shots and make it easy next time to create that kind a shot:

```
from stalker.core.models.pipelineStep import PipelineStep
from stalker.core.models.types import AssetType

# create the pipeline steps of a particular Shot asset
previz    = PipelineStep(name="Previz"     , code="PRE")
matchmove = PipelineStep(name="Match Move" , code="MM")
anim      = PipelineStep(name="Animation"  , code="ANIM")
light     = PipelineStep(name="Ligting"    , code="LIGHT")
comp      = PipelineStep(name="Compositing", code="COMP")

simple_shot_pipeline_steps = [previz, match, anim, light, comp]

# assign them as the steps of "Shot" assets
shot_asset_type = AssetType(name="Shot", steps=shot_pipeline_steps)

# and set our shot objects asset_type to stho_asset_type
#
# instead of writing down shot1.type = shot_asset_type
# we are going to do something more interesting
for shot in seq1.shots:
    shot.type = shot_asset_type
```

So by doing that we inform Stalker about the steps of one kind of asset (*Shot* in our case).

Part IV - Task & Resource Management

Now we have a couple of Shots with couple of steps inside it but we didn't created any `Task` to let somebody to finish the job.

Lets assign all this stuff to our self (for now):

```
from datetime import timedelta
from stalker.core.models.task import Task

task1 = Task(name="Previz",
             resources=[myUser],
             bid=timedelta(days=1),
             pipeline_step=previz)

task2 = Task(name="Match Move",
             resources=[myUser],
             bid=timedelta(days=2),
             pipeline_step=matchmove)

task3 = Task(name="Animation",
             resources=[myUser],
             bid=timedelta(days=2),
             pipeline_step=anim)

task4 = Task(name="Lighting",
             resources=[myUser],
             bid=timedelta(days=2),
             pipeline_step=light)

task5 = Task(name="Compositing",
             resources=[myUser],
             bid=timedelta(days=2),
             pipeline_step=comp)
```

Now we are created all the tasks, but they are not connected to our Shots yet. Lets connect them to the `shot001`:

```
sh001.tasks = [task1, task2, task3, task4, task5]
```

And one of the good sides of the tasks are, dependencies can be defined between them, so Stalker nows which job should be done before the others:

```
# animation needs match moving and previz to be finished
task3.depends = [task1, task2]

# compositing can not start before anything rendered or animated
task5.depends = [task3, task4]
```

Now Stalker nows the hierarchy of the tasks. Next versions of Stalker will have a `Scheduler` included to solve the task timings and create data for things like Gantt Charts.

Lets commit the changes again:

```
session.commit()
```

This time we didn't add anything to the session, cause we have added the `new_project` in a previous commit, and because all the objects are attached to the project object in some way, Stalker can track this changes and add the missing related objects to the database.

Part V - Asset Management

Now we have created a lot of things but other then storing all the data in the database, we didn't do much. Stalker still doesn't have information about a lot of things. For example, it doesn't know how to handle your asset versions (`Version`) namely it doesn't know how to store your data that you are going to create while completing this tasks.

A fileserver in Stalkers' term is called a `Repository`. Repositories stores the information about the fileservers in your system. You can have several file servers let say one for Commercials and other one for big Movie projects. You can define repositories and assign projects to those repositories. Lets create one repository for our commercial project:

```
from stalker.core.models.repository import Repository
repo1 = Repository(
    name="Commercial Repository",
    description="""This is where the commercial projects are going to be
                   stored"""
)
```

A Repository object could show the root path of the repository according to your operating system. Lets enter the paths for all the major operating systems:

```
repo1.windows_path = "M:\\PROJECTS"
repo1.linux_path   = "/mnt/M"
repo1.osx_path     = "/Volumes/M"
```

And if you ask a repository object for the path of the repository it will always give the correct answer accroding to your operating system:

```
print repo1.path
# outputs:
# if you are running the command on a computer with Windows it will output:
#
# M:\PROJECTS
#
# and for Linux:
# /mnt/M
```

```
#
# for OSX:
# /Volumes/M
#
```

Assigning this repository to our project or vice versa is not enough, Stalker still doesn't know about the project `Structure`, or in other words it doesn't have information about the folder structure about your project. To explain the project structure we can use the `Structure` object:

```
from stalker.core.models.structure import Structure

structure1 = Structure(
    name="Commercial Projects Structure",
    description="""This is a project structure, which can be used for simple
        commercial projects"""
)

# lets create the folder structure as a Jinja2 template
project_template = """
   {{ project.code }}
   {{ project.code }}/Assets
   {{ project.code }}/Sequences"

   {% if project.sequences %}
       {% for sequence in project.sequences %}
           {% set seq_path = project.code + '/Sequences/' + sequence.code %}
           {{ seq_path }}
           {{ seq_path }}/Edit
           {{ seq_path }}/Edit/AnimaticStoryboard
           {{ seq_path }}/Edit/Export
           {{ seq_path }}/Storyboard
           {{ seq_path }}/Shots

           {% if sequence.shots %}
               {% for shot in sequence.shots %}
                   {% set shot_path = seq_path + '/SHOTS/' + shot.code %}
                   {{ shot_path }}
               {% endfor %}
           {% endif %}

       {% endfor %}

   {% endif %}

   {{ project.code }}/References
"""

structure1.project_template = project_template
```

Now we have entered a couple of Jinja2 directives as a string. This template will be used when creating the project structure by calling `create()`. It is safe to call the `create()` over and over or whenever you've added new data that will add some extra folders to the project structure.

The above template will produce the following folders for our project:

```
M:/PROJECTS/FANCY_COMMERCIAL
M:/PROJECTS/FANCY_COMMERCIAL/Assets
M:/PROJECTS/FANCY_COMMERCIAL/References
M:/PROJECTS/FANCY_COMMERCIAL/Sequences
```

```
M:/PROJECTS/FANCY_COMMERCIAL/Sequences/SEQ1
M:/PROJECTS/FANCY_COMMERCIAL/Sequences/SEQ1/Edit
M:/PROJECTS/FANCY_COMMERCIAL/Sequences/SEQ1/Edit/AnimaticStoryboard
M:/PROJECTS/FANCY_COMMERCIAL/Sequences/SEQ1/Edit/Export
M:/PROJECTS/FANCY_COMMERCIAL/Sequences/SEQ1/Storyboard
M:/PROJECTS/FANCY_COMMERCIAL/Sequences/SEQ1/Shots
M:/PROJECTS/FANCY_COMMERCIAL/Sequences/SEQ1/Shots/SH001
M:/PROJECTS/FANCY_COMMERCIAL/Sequences/SEQ1/Shots/SH002
M:/PROJECTS/FANCY_COMMERCIAL/Sequences/SEQ1/Shots/SH003
```

Imagine what else can be done here with that kind of template system. And you can use a lot of variables inside this templates.

We are still not done with defining the templates. Even though Stalker now knows what is the project structure like, it is not aware of the placements of indivudual asset `Version` files. An asset `Version` is an object holding information about every single iteration of one asset and has a connection to files in the repository. So before creating a new version for any kind of asset, we need to tell Stalker where to place the related files. This can be done by using a `TypeTemplate` object.

A *TypeTemplate* object has information about the path, the filename, and the Type of the asset to apply this template to:

1.2.3 Design

The design of Stalker is mentioned in the following sections.

Mission

The project is about creating an Open Source Production Asset Management (ProdAM) System called Stalker which is designed for Vfx/Animation Studios. Stalker will be consisting of a framework and the interface those has been build over that framework. Stalker will have a very flexible object model design that lets the pipeline TDs to customize it in a wide variety of perspectives. The out of the package installation will meet the basic needs of majority of studios without too much effort.

Introduction

An Asset Management Systems' duty is to hold the data which are created by the users of the system in an orginized manner, and let them quickly reach and find their files. A Production Asset Management Systems' duty is, in addition to the asset management systems', also handle the production steps and collaboration of the users. If more information about this subject is needed, there are great books about Digital Asset Management (DAM) Systems.

The usage of an asset management system in an animation/vfx studio is an obligatory duty for the sake of the studio itself. Even the benefits of the system becomes silly to be mentioned when compared to the lack of even a simple system to organize stuff.

Every studio outside establishes and developes their own asset management system. Stalker will try to be the framework that these proprietry asset management systems will be build over. Thus reducing the work repeated on every big projects start.

Concepts

There are a couple of design concepts those needs to be clarified before any further explanation of Stalker.

Stalker Object Model (SOM)

Stalker has a very robust object model, which is called **Stalker Object Model** or **SOM**. The idea behind SOM is to create a simple class hierarchy which is both usable right out of the box and also expandable by the studios' pipeline TDs. SOM is actually a little bit more complex than the basic possible model, it is designed in this way just to be able to create a simple pipeline to be able to build the system on it.

Lets look at how a simple studio works and try to create our asset management concepts around it.

An animation/vfx studios duty is to complete a **Project**. A project, generally is about to create a **Sequence** of **Shots** which are a series of images those at the end converts to a movie. So a sequence in general contains Shots. Shots are a special type of **Assets** which are related to a range of time. So basically to complete a project the studio should complete the sequences thus the shots.

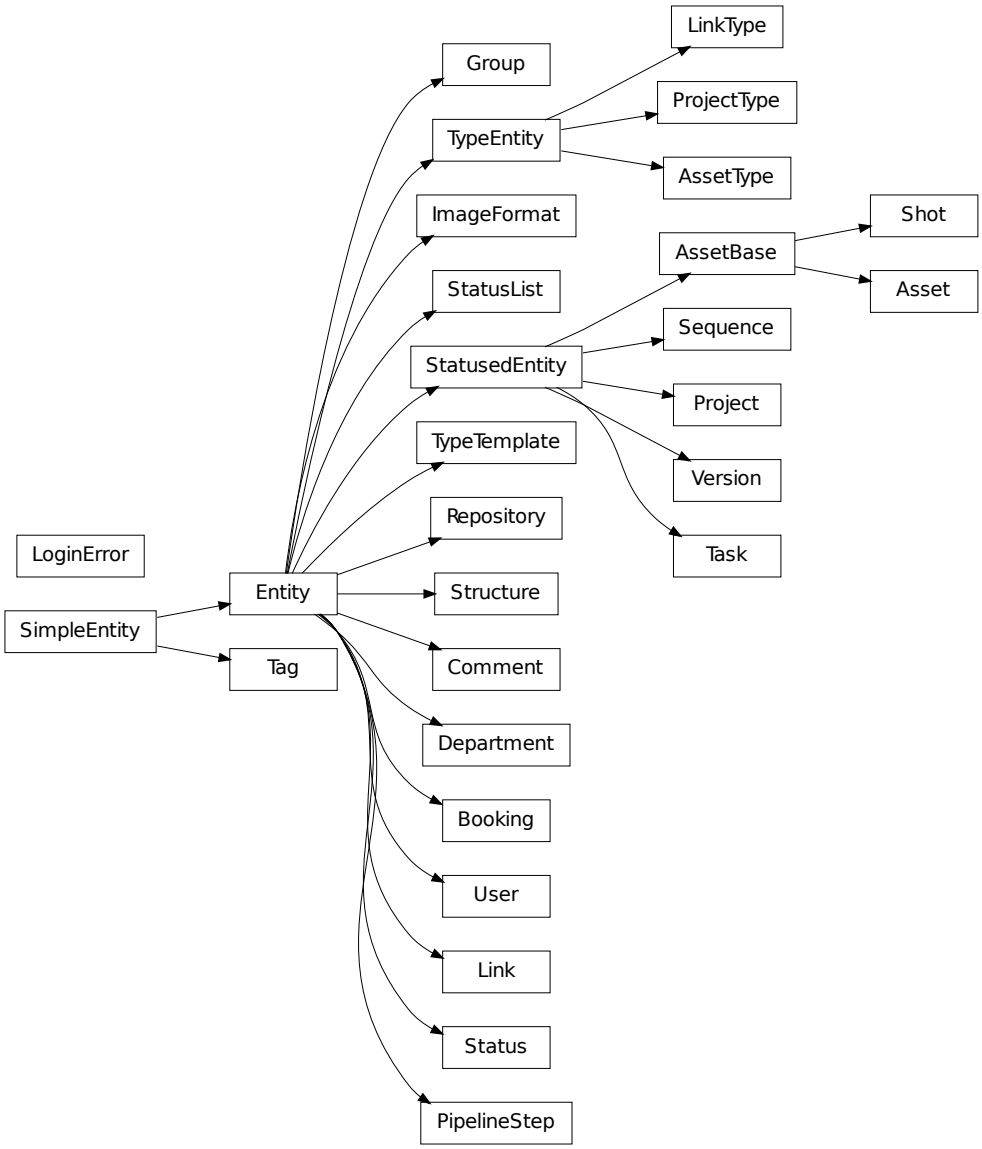
We have considered Shots as a special form of assets, so assets have **AssetTypes**, it is *Shot* for a Shot asset, and lets say, it is *Character* for a character asset, or *Vehicle* for a vehicle asset (pretty straight).

AssetType also defines the **PipelineSteps** of that special asset type. For example a Shot can have steps like *Animation*, *FX*, *Layout*, *Lighting*, *Compositing* etc. and a character can have *Design*, *Model*, *Rig*, *Shading* steps. All these steps defines differentiable **Tasks** those need to be done sequently or in parallel to complete that shot or asset. Again, an asset or shot has an asset type, which defines the steps thus tasks those needs to be done.

A Task relates to a work, a work is a quantity of time spend or going to be spend for that specific task. At the end of the work generally a **User** creates **Versions** for a task. Versions are list of files showing the different incarnations or the progress of a subject in the fileserver or in Stalkers term the **Repository**. Also while creating those files to complete the tasks a user should be booked. **Bookings** are special type of objects holds information about how much time has been spent for a given task.

All the names those shown in bold fonts are a class in SOM. and there are a series of other classes to accomodate the needs of a simple studio.

The inheritance diagram of the classes in the SOM is shown below:



Stalker is a configurable and expandible system. Both of these feature allows the system to have a flexible structure.

There are two levels of expansion, the first level is the simplest one, by just adding different statuses, different asset-Types or these kind of things in which Stalker's current design is ready to.

The second level of expansion is achieved by expanding the SOM. Expanding the some includes creating new classes and database tables, and updating the old ones which are already comming with Stalker. These expansion schemes are further explained in How To Expand Stalker.

Features and Requirements

Features:

1. Developed purely in Python (2.6 and over) using TDD (Test Driven Development) practices
2. SQLAlchemy for the database back-end and ORM
3. PyQt/PySide and web based user interfaces. All the interfaces designed in MVC structure.
4. Jinja2 as the template engine
5. Users are able to select their preferred database like PostgreSQL, MySQL, Oracle, SQLite etc. (whatever SQLAlchemy supports)
6. It is possible to use both one or different databases for studio specific and project specific data. It is mostly beneficial when the setup uses SQLite. The project specific data could be kept in project folder as an SQLite db file and the studio specific data can be another SQLite db file or another database connection to PostgreSQL, MySQL, Oracle etc. databases. In an SQLite setup, the database can be backed up with the project folder itself.
7. Configuration files lets the user to configure all the aspects of the asset/project management.
8. Uses Jinja2 as the templating system for the file and folder naming convention will be used like:

   {repository.path}/{project.name}/assets/{asset.name}/{pipelineStep.name}/ {asset.variation.name}/{asset.name}_{asset.type.name}_v{asset.version}.{ asset.fileFormat.extension}
9. file and folders and file sequences can be uploaded to the server as assets, and the server decides where to place the folder or file by using the templating system.
10. The event system gives full control for every CRUD (create/insert, read, update, delete) by giving step like before insert, after insert callbacks.
11. The messaging system allows the users collaborate more efficiently.

Usage Examples

Let's dance with Stalker a little bit.

When you first setup Stalker you will have nothing but an empty database. So lets create some data and store them in the database.

First import some modules:

First of all import and setup the default database (an in-memory SQLite database)

```
>>> from stalker import db # the database module
>>> db.setup()
```

By calling the `setup()` we have created all the mappings for SOM and also we have created the `session` object which is stored under `stalker.db.meta.session` (this is used to have a Singleton SQLAlchemy metadata).

Lets import the SOM which is stalker.core.models

```
>>> from stalker.core.models.user import User
```

Stalker comes with an *admin* user already defined in to it. To create other things in the database we need to have the admin user by querying it.

```
>>> dbSession = db.meta.session
>>> admin = dbSession.query(User).filter_by(name="admin").first()
```

Lets create another user

```
>>> newUser = User(name="eoyilmaz",
                   login_name="eoyilmaz",
                   first_name="Erkan Ozgur",
                   last_name="Yilmaz",
                   password="secret",
                   email="eoyilmaz@gmail.com")
```

Save the data to the database

```
>>> session.add(newUser)
>>> session.commmit()
```

Create a query for users:

```
>>> query = session.query(user.User)
```

Get all the users:

```
>>> users = query.all()
```

or select a couple of users by filters:

```
>>> users = query.filter_by(name="Ozgur")
```

or select the first user matching query criteria:

```
>>> user_ozgur = query.filter_by(name="Ozgur").first()
```

* UPDATE BELOW *

Now add them to the project:

```
>>> newProject.users.append(users)
```

Save the new project to the database:

```
>>> mapper.session.save(newProject)
>>> mapper.session.flush()
```

Let's ask the tasks of one user:

```
>>> ozgur = query.filter_by(name="ozgur")
>>> tasks = ozgur.tasks
```

Get the on going tasks of this user:

```
>>> onGoingTasks = [task for task in ozgur.tasks if not task.isComplete]
```

Get the on going tasks of this user by using the database:

```
>>> taskQuery = mapper.sessison.query(user.User).filter_by(name="ozgur").join(task.Task).filter_by(st
>>> onGoingTasks = taskQuery.all()
```

Get the "rig" tasks of ozgur:

```
>>> rigTasks =  taskQuery.join(pipelineStep.pipelineStep).filter_by(name="Rig").all()
```

As you see all the functionalities of SQLAlchemy is fully supported. At the end all the models are plain old python objects (POPO) and the persistancy part is handled with SQLAlchemy.

How To Customize Stalker

This part explains the customization of Stalker.

How To Extend SOM

This part explains how to extend Stalker Object Model or SOM.

1.2.4 How To Contribute

Stalker started as an Open Source project with the expectation of contributions. The soul of the open source is to share the knowledge and contribute.

These are the areas that you can contribute to:

- Documentation
- Testing the code
- Writing the code
- Creating user interface elements (graphics, icons etc.)

Development Style

Stalker is developed strictly by following TDD practices. So every participant should follow TDD methodology. Skipping this steps is highly prohibited. Every added code to the trunk should have a corresponding test and the tests should be written before implementing a single line of code.

DRY is also another methodology that a participant should follow. So nothing should be repeated. If something needs to be repeated, then it is a good sign that this part needs to be in a special module, class or function.

Testing

As stated above all the code written should have a corresponding test.

Adding new features should start with design sketches. These sketches could be plain text files or mind maps or anything that can express the thing in you mind. While writing down these sketches, it should be kept in mind that these files also could be used to generate the documentation of the system. So writing down the sketches as rest files inside the docs is something very meaningful.

The design should be followed by the tests. And the test should be followed by the implementation, and the implementation should be followed by tests again, until you are confident about your code and it is rock solid. Then the refactoring phase can start, and because you have enough tests that will keep your code doing a certain thing, you can freely change your code, because you know that you code will do the same thing if it passes all the tests.

The first tests written should always fail by having:

```
self.fail("the test is not implemented yet")
```

failures. This is something good to have. This will inform us that the test is not written yet. So lets start adding the code that will pass the tests.

The test framework of Stalker is unitTest and nose to help testing.

These python modules should be installed to test Stalker properly:

- Nose

- Coverage
- Mocker

The Mocker library should be used to isolate the currently tested part of the code.

The coverage of the tests should be kept as close as possible to %100.

There is a helper script in the root of the project, called *doTests*. This is a shell script for linux, which runs all the necessary tests and prints the tests results and the coverage table.

Code Style

For the general coding style every participant should strictly follow PEP 8 rules, and there are some extra rules as listed below:

- Class names should start with an upper-case letter, function and method names should start with lower-case letter
- The class definitions should be preceded by 72 # characters, if you are using Wing IDE it is trivial cause it has these kind of templates:

```
##############################################################################
class StatusBase(object):
    """The StatusBase class
    """
    pass
```

- The method or function definitions should be preceded by 70 - characters, and the line should be commented out, again if you are using Wing IDE it does that automatically:

```
#----------------------------------------------------------------------
def __init__(self, name, abbreviation, thumbnail=None):

    self._name = self._checkName(name)
```

- There should be 3 spaces before and after functions and class methods:

```
##############################################################################
class StatusBase(object):
    """The StatusBase class
    """



    #--------------------------------------------------------------------------
    def __init__(self, name, abbreviation, thumbnail=None):
        self._name = self._checkName(name)



    #--------------------------------------------------------------------------
    def _checkName(self, name):
        """checks the name attribute
        """

        if name == "" or not isinstance(name, (str, unicode) ):
            raise(ValueError("the name shouldn't be empty and it should \
                be a str or unicode"))
```

```
            return name.title()
```

- And also there should be 6 spaces before and after a class body:

```
#-*- coding: utf-8 -*-




##########################################################################
class A(object):
    pass




##########################################################################
class B(object):
    pass




pass
```

- Any lines that may contain a code or comment can not be longer than 80 characters, all the longer lines should be cancelled with "\" character and should continue properly from the line below:

```
#--------------------------------------------------------------------------
def _checkName(self, name):
    """checks the name attribute
    """

    if name == "" or not isinstance(name, (str, unicode) ):
        raise(ValueError("the name shouldn't be empty and it should be a \
        str or unicode"))

    return name.title()
```

- If anything is going to be checked against being None you should do it in this way:

```
if a is None:
    pass
```

- Do not add docstrings to __init__ rather use the classes' own docstring.
- The first line in the docstring should be a brief summary separated from the rest by a blank line.

If you are going to add a new python file (*.py), there is an empty py file with the name empty_code_template_file.py under docs/_static. Before starting anything, duplicate this file and place it under the folder you want. This files has the necessary shebang and the GPL 3 license text.

SCM - Mercurial (HG)

The choice of SCM is Mercurial. Every developer should be familiar with it. It is a good start to go the Selenic Mercurial Site and do the tutorial if you don't feel familiar enough with hg.

1.2.5 Stalker Development Roadmap

This section describes the direction Stalker is going.

Roadmap Based on Versions

Below you can find the roadmap based on the version

0.1.0:

- A complete working set of models in SOM

0.2.0:

- The SQLAlchemy integration to have the database part, tables and mappers

0.3.0:

- Ability to extend SOM

0.5.0:

- Interactions with host programs (MAYA, NUKE, HOUDINI etc.)

1.0.0:

- PyQt/PySide interfaces

1.5.0:

- Web interface

2.0.0:

- New feature that I can't really see right now.

Continued on next page

Table 1.1 – continued from previous page

1.2.6 Summary

stalker	Stalker is a Production Asset Management System (ProdAM) designed
stalker.db	This is the database module of Stalker.
stalker.db.auth	This is the authentication system of Stalker. Uses Beaker for
stalker.db.auth.authenticate	Authenticates the given username and password, r
stalker.db.auth.create_session	creates the session
stalker.db.auth.get_user	returns the user from stored s
stalker.db.auth.login	Persist a user id in the session.
stalker.db.auth.login_required	a decorator that implements login functionality to any
stalker.db.auth.permission_required	a decorator that implements permission checking to any
stalker.db.auth.logout	removes the current session
stalker.db.mapper	this is the default mapper to map the default models to the d
stalker.db.tables	this file contains the tags
stalker.db.setup	
stalker.core.models	
stalker.core.models.asset	
stalker.core.models.asset.Asset	The Asset class is the whole idea behind Sta
stalker.core.models.assetBase	
stalker.core.models.assetBase.AssetBase	This is the base class for Shot and Asset cla
stalker.core.models.booking	
stalker.core.models.booking.Booking	Booking holds the information about when a user done whi
stalker.core.models.comment	
stalker.core.models.comment.Comment	The Comment data model which derives from th
stalker.core.models.department	
stalker.core.models.department.Department	A department holds information about a studios
stalker.core.models.entity	
stalker.core.models.entity.SimpleEntity	The base class of all th
stalker.core.models.entity.Entity	This is the entity class which is derived from the Si
stalker.core.models.entity.StatusedEntity	This is a normal entity class that derives from Ent
stalker.core.models.entity.TypeEntity	TypeEntity is the entry point for
stalker.core.models.error	
stalker.core.models.error.LoginError	Raised when the login information is not correct or
stalker.core.models.group	
stalker.core.models.group.Group	the group class
stalker.core.models.imageFormat	
stalker.core.models.imageFormat.ImageFormat	the image format
stalker.core.models.link	
stalker.core.models.link.Link	Holds data about external links
stalker.core.models.pipelineStep	
stalker.core.models.pipelineStep.PipelineStep	A PipelineStep object represents the general pipelin
stalker.core.models.project	
stalker.core.models.project.Project	the project clas
stalker.core.models.repository	
stalker.core.models.repository.Repository	Repository is a class to hold repository s
stalker.core.models.sequence	
stalker.core.models.sequence.Sequence	the sequence clas
stalker.core.models.shot	
stalker.core.models.shot.Shot	The Shot class to manage Shot
stalker.core.models.status	

Continued on next page

Table 1.1 – continued from previous page

stalker.core.models.status.Status	The Status clas
stalker.core.models.status.StatusList	the list version of the
stalker.core.models.structure	
stalker.core.models.structure.Structure	A structure object is the place to hold data about
stalker.core.models.tag	
stalker.core.models.tag.Tag	the tag class
stalker.core.models.task	
stalker.core.models.task.Task	the task class
stalker.core.models.types	
stalker.core.models.types.AssetType	The AssetType class holds the information about
stalker.core.models.types.ProjectType	Helps to create different t
stalker.core.models.types.LinkType	The type of Link is hold
stalker.core.models.types.TypeTemplate	The TypeTemplate model holds templates f
stalker.core.models.user	
stalker.core.models.user.User	The user class is designed to hold data about a Us
stalker.core.models.version	
stalker.core.models.version.Version	The Version class is the connection of Assets to vers

stalker

Stalker is a Production Asset Management System (ProdAM) designed for animation and vfx studios. See docs for more information.

stalker.db

This is the database module of Stalker.

Whenever stalker.db or something under it imported, the `setup()` becomes available to let one setup the database.

Functions

`setup`([database, mappers, engine_settings])

stalker.db.auth

This is the authentication system of Stalker. Uses Beaker for the session management.

This helper module is written to help users to persist their login information in their system. The aim of this function is not security. So one can quickly by-pass this system and get himself/herself logged in or query information from the database without login.

The user information is going to be used in the database to store who created, updated, read or delete the data.

There are two functions to log a user in, first one is `authenticate()`, which accepts username and password and returns a `User` object:

```
from stalker.db import auth
userObj = auth.authenticate("username", "password")
```

The second one is the `login()` which uses a given `User` object and creates a Beaker Session and stores the logged in user id in that session.

The `get_user()` can be used to get the authenticated and logged in `User` object.

The basic usage of the system is as follows:

```
from stalker import db
from stalker.db import auth
from stalker.core.models import user

# directly get the user from the database if there is a user_id
# in the current auth.SESSION
#
# in this way we prevent asking the user for login information all the time
if "user_id" in auth.SESSION:
    userObj = auth.get_user()
else:
    # ask the username and password of the user
    # then authenticate the given user
    username, password = the_interface_for_login()
    userObj = auth.authenticate(username, password)

# login with the given user.User object, this will also create the session
# if there is no one defined
auth.login(userObj)
```

The module also introduces a decorator called `login_required()` to help adding the authentication functionality to any function or method

Functions

`authenticate`([username, password])	Authenticates the given username and password, returns a
`create_session`()	creates the session
`get_user`()	returns the user from stored session
`login`(user_obj)	Persist a user id in the session.
`login_required`(view[, error_message])	a decorator that implements login functionality to any function or
`logout`()	removes the current session
`permission_required`(permission_group[, ...])	a decorator that implements permission checking to any function or

stalker.db.auth.authenticate

stalker.db.auth.**authenticate**(*username=''*, *password=''*)

Authenticates the given username and password, returns a stalker.core.models.user.User object

There needs to be a already setup database for the authentication to hapen.

stalker.db.auth.create_session

stalker.db.auth.**create_session**()
creates the session

stalker.db.auth.get_user

stalker.db.auth.**get_user**()
returns the user from stored session

stalker.db.auth.login

stalker.db.auth.**login**(*user_obj*)
Persist a user id in the session. This way a user doesn't have to reauthenticate on every request

stalker.db.auth.login_required

stalker.db.auth.**login_required**(*view*, *error_message=None*)
a decorator that implements login functionality to any function or method

The view should be a function returning True or False

stalker.db.auth.permission_required

stalker.db.auth.**permission_required**(*permission_group*, *error_message=None*)
a decorator that implements permission checking to any function or method

Checks if the logged in user is in the given permission group and then calls the decorated function

stalker.db.auth.logout

stalker.db.auth.**logout**()
removes the current session

stalker.db.mapper

this is the default mapper to map the default models to the default tables

You can use your also use your own mappers. See the docs.

Functions

backref(name, **kwargs)	Create a back reference with explicit arguments, which are the same arguments one can send to relationship().
mapper(class_, *args, **params[, local_table])	Return a new Mapper object.
relationship(argument, **kwargs[, secondary])	Provide a relationship of a primary Mapper to a secondary Mapper.
setup()	setups the mapping
synonym(name[, map_column, descriptor, ...])	Set up *name* as a synonym to another mapped property.

stalker.db.tables

this file contains the tags table

Classes

Column(*args, **kwargs)	Represents a column in a database table.
DateTime([timezone])	A type for datetime.datetime() objects.
Float(**kwargs[, precision, asdecimal])	A type for float numbers.
ForeignKey(column[, _constraint, use_alter, ...])	Defines a dependency between two columns.
Integer(*args, **kwargs)	A type for int integers.
String([length, convert_unicode, ...])	The base for all string and character types.
Table(*args, **kw)	Represent a table in a database.
UniqueConstraint(*columns, **kw)	A table-level UNIQUE constraint.

stalker.db.setup

stalker.db.**setup**(*database=None*, *mappers=[]*, *engine_settings=None*)

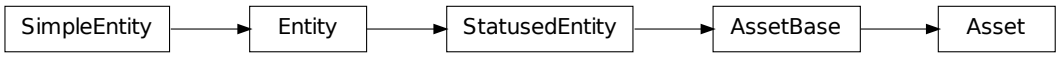
stalker.core.models

stalker.core.models.asset

Classes

Asset(**kwargs[, status_list, status])	The Asset class is the whole idea behind Stalker.

stalker.core.models.asset.Asset



class stalker.core.models.asset.**Asset**(*status_list=[ ]*, *status=0*, ***kwargs*)

Bases: stalker.core.models.assetBase.AssetBase

The Asset class is the whole idea behind Stalker.

__init__(*status_list=[ ]*, *status=0*, ***kwargs*)

Methods

__init__(**kwargs[, status_list, status])	

Attributes

code	
created_by	gets and sets the User object who has created this
date_created	gets and sets the datetime.datetime object which shows when
date_updated	gets and sets the datetime.datetime object which shows when
description	the description of the entity
name	the name of the entity
nice_name	this is the nice name of the SimpleEntity. It has the same
status	this is the property that sets and returns the status attribute
status_list	this is the property that sets and returns the status_list attribute
tags	a list of Tag objects which shows the related tags to the
updated_by	gets and sets the User object who has updated this

created_by
gets and sets the User object who has created this AuditEntity

date_created
gets and sets the datetime.datetime object which shows when this object has been created

date_updated
gets and sets the datetime.datetime object which shows when this object has been updated

description
the description of the entity

name
the name of the entity

nice_name
this is the nice name of the SimpleEntity. It has the same value with the name (contextually) but with a different format like, all the whitespaces replaced by underscores ("_"), all the CamelCase form will be expanded by underscore (_) characters and it is always lowercase.

There is also the code attribute which is simple the uppercase form of nice_name if it is not defined differently (i.e set to another value).

status
this is the property that sets and returns the status attribute

status_list
this is the property that sets and returns the status_list attribute

tags
a list of Tag objects which shows the related tags to the entity

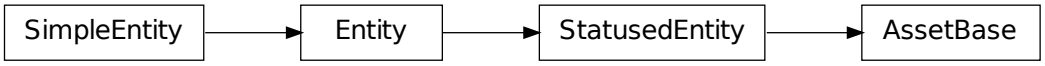
updated_by
gets and sets the User object who has updated this AuditEntity

stalker.core.models.assetBase

Classes

AssetBase(**kwargs[, status_list, status])	This is the base class for Shot and Asset classes.

stalker.core.models.assetBase.AssetBase



class stalker.core.models.assetBase.**AssetBase**(*status_list*=[ ], *status=0*, ***kwargs*)

Bases: stalker.core.models.entity.StatusedEntity

This is the base class for Shot and Asset classes.

__init__(*status_list*=[ ], *status=0*, ***kwargs*)

Methods

__init__(**kwargs[, status_list, status])	

Attributes

code	
created_by	gets and sets the User object who has created this
date_created	gets and sets the datetime.datetime object which shows when
date_updated	gets and sets the datetime.datetime object which shows when
description	the description of the entity
name	the name of the entity
nice_name	this is the nice name of the SimpleEntity. It has the same
status	this is the property that sets and returns the status attribute
status_list	this is the property that sets and returns the status_list attribute
tags	a list of Tag objects which shows the related tags to the
updated_by	gets and sets the User object who has updated this

created_by
gets and sets the User object who has created this AuditEntity

date_created
gets and sets the datetime.datetime object which shows when this object has been created

date_updated
gets and sets the datetime.datetime object which shows when this object has been updated

description
the description of the entity

name
the name of the entity

nice_name
this is the nice name of the SimpleEntity. It has the same value with the name (contextually) but with a different format like, all the whitespaces replaced by underscores ("_"), all the CamelCase form will be expanded by underscore (_) characters and it is always lowercase.

There is also the code attribute which is simple the uppercase form of nice_name if it is not defined differently (i.e set to another value).

status
this is the property that sets and returns the status attribute

status_list
this is the property that sets and returns the status_list attribute

tags
a list of Tag objects which shows the related tags to the entity

updated_by
gets and sets the User object who has updated this AuditEntity

stalker.core.models.booking

Classes

Booking(**kwargs[, tags])	Booking holds the information about when a user done which task and

stalker.core.models.booking.Booking

class stalker.core.models.booking.**Booking**(*tags*=[ ], ***kwargs*)

Bases: stalker.core.models.entity.Entity

Booking holds the information about when a user done which task and spend how many hours on doing that task.

__init__(*tags*=[ ], ***kwargs*)

Methods

__init__(**kwargs[, tags])	

Attributes

code	
created_by	gets and sets the User object who has created this
date_created	gets and sets the datetime.datetime object which shows when
date_updated	gets and sets the datetime.datetime object which shows when
description	the description of the entity
name	the name of the entity
nice_name	this is the nice name of the SimpleEntity. It has the same
tags	a list of Tag objects which shows the related tags to the
updated_by	gets and sets the User object who has updated this

created_by
gets and sets the User object who has created this AuditEntity

date_created
gets and sets the datetime.datetime object which shows when this object has been created

date_updated
gets and sets the datetime.datetime object which shows when this object has been updated

description
the description of the entity

name
the name of the entity

nice_name
this is the `nice name` of the SimpleEntity. It has the same value with the name (contextually) but with a different format like, all the whitespaces replaced by underscores ("_"), all the CamelCase form will be expanded by underscore (_) characters and it is always lowercase.

There is also the `code` attribute which is simple the uppercase form of `nice_name` if it is not defined differently (i.e set to another value).

tags
a list of Tag objects which shows the related tags to the entity

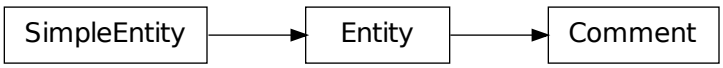
updated_by
gets and sets the User object who has updated this AuditEntity

stalker.core.models.comment

Classes

`Comment`(**kwargs[, body, to])	The Comment data model which derives from the entity.AuditEntity

stalker.core.models.comment.Comment



class `stalker.core.models.comment.`**Comment**(*body=''*, *to=None*, ***kwargs*)
Bases: `stalker.core.models.entity.Entity`

The Comment data model which derives from the entity.AuditEntity

Parameters

- **body** – the body of the comment, it is a string or unicode variable, it can be empty but it is then meaningles to have an empty comment. Anything other than a string or unicode will raise a ValueError.

- **to** – the relation variable, that holds the connection that this comment is related to. it should be an Entity object, any other will raise a ValueError

__init__(*body=''*, *to=None*, ***kwargs*)

Methods

`__init__`(**kwargs[, body, to])	

Attributes

`body`	this is the property that sets and returns the body attribute
`code`	
`created_by`	gets and sets the User object who has created this
`date_created`	gets and sets the datetime.datetime object which shows when
`date_updated`	gets and sets the datetime.datetime object which shows when
`description`	the description of the entity
`name`	the name of the entity
`nice_name`	this is the `nice name` of the SimpleEntity. It has the same
`tags`	a list of Tag objects which shows the related tags to the
`to`	this is the property that sets and returns the to attribute
`updated_by`	gets and sets the User object who has updated this

body
this is the property that sets and returns the body attribute

created_by
gets and sets the User object who has created this AuditEntity

date_created
gets and sets the datetime.datetime object which shows when this object has been created

date_updated
gets and sets the datetime.datetime object which shows when this object has been updated

description
the description of the entity

name
the name of the entity

nice_name
this is the `nice name` of the SimpleEntity. It has the same value with the name (contextually) but with

a different format like, all the whitespaces replaced by underscores ("_"), all the CamelCase form will be expanded by underscore (_) characters and it is always lowercase.

There is also the `code` attribute which is simple the uppercase form of `nice_name` if it is not defined differently (i.e set to another value).

tags
a list of Tag objects which shows the related tags to the entity

to
this is the property that sets and returns the to attribute

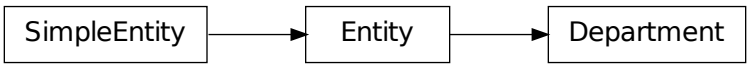
updated_by
gets and sets the User object who has updated this AuditEntity

stalker.core.models.department

Classes

`Department`(**kwargs[, members, lead])	A department holds information about a studios departments.

stalker.core.models.department.Department



class `stalker.core.models.department.`**`Department`**(*members=[ ]*, *lead=None*, ***kwargs*)
Bases: `stalker.core.models.entity.Entity`

A department holds information about a studios departments. The informations that a Department object holds is like:

- The members of the department
- The lead of the department
- and all the other things those are inherited from the AuditEntity class

so creating a department object needs the following parameters:

Parameters

- **members** – it can be an empty list, so one department can be created without any member in it. But this parameter should be a list of User objects.
- **lead** – this is a User object, that holds the lead information, a lead could be in this department but it is not forced to be also a member of the department. So another departments member can be a lead for another department. Lead attribute can not be empty or None.

__init__ (*members*=[ ], *lead*=*None*, ***kwargs*)

Methods

`__init__`(**kwargs[, members, lead])	

Attributes

code	
created_by	gets and sets the User object who has created this
date_created	gets and sets the datetime.datetime object which shows when
date_updated	gets and sets the datetime.datetime object which shows when
description	the description of the entity
lead	lead is the lead of this department, it is a User object
members	members are a list of users representing the members of this
name	the name of the entity
nice_name	this is the `nice name` of the SimpleEntity. It has the same
tags	a list of Tag objects which shows the related tags to the
updated_by	gets and sets the User object who has updated this

created_by
gets and sets the User object who has created this AuditEntity

date_created
gets and sets the datetime.datetime object which shows when this object has been created

date_updated
gets and sets the datetime.datetime object which shows when this object has been updated

description
the description of the entity

lead
lead is the lead of this department, it is a User object

members
members are a list of users representing the members of this department

name
the name of the entity

nice_name
this is the `nice name` of the SimpleEntity. It has the same value with the name (contextually) but with

a different format like, all the whitespaces replaced by underscores ("_"), all the CamelCase form will be expanded by underscore (_) characters and it is always lowercase.

There is also the `code` attribute which is simple the uppercase form of `nice_name` if it is not defined differently (i.e set to another value).

tags
a list of Tag objects which shows the related tags to the entity

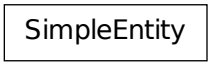
updated_by
gets and sets the User object who has updated this AuditEntity

stalker.core.models.entity

Classes

Entity(**kwargs[, tags])	This is the entity class which is derived from the SimpleEntity and adds
SimpleEntity(1, 13, 18, 37, 26, 1, 13, 18, ...)	The base class of all the others
StatusedEntity(**kwargs[, status_list, status])	This is a normal entity class that derives from Entity and adds status
TypeEntity(**kwargs)	TypeEntity is the entry point for types.

stalker.core.models.entity.SimpleEntity



class stalker.core.models.entity.**SimpleEntity**(*name=None*, *description=''*, *created_by=None*, *updated_by=None*, *date_created=datetime.datetime(2011, 1, 13, 18, 37, 26, 361690)*, *date_updated=datetime.datetime(2011, 1, 13, 18, 37, 26, 361702)*, *code=None*)

Bases: object

The base class of all the others

This class has the basic information about an entity which are the name, the description, tags and the audit information like created_by, updated_by, date_created and date_updated about this entity.

Parameters

- **name** – a string or unicode attribute that holds the name of this entity. it could not be empty, the first letter should be an alphabetic (not alphanumeric) letter and it should not contain any white space at the beggining and at the end of the string
- **description** – a string or unicode attribute that holds the description of this entity object, it could be an empty string, and it could not again have white spaces at the beggining and at the end of the string
- **created_by** – the created_by attribute should contain a User object who is created this object
- **updated_by** – the updated_by attribute should contain a User object who is updated the user lastly. the created_by and updated_by attributes should point the same object if this entity is just created
- **date_created** – the date that this object is created. it should be a time before now
- **date_updated** – this is the date that this object is updated lastly. for newly created entities this is equal to date_created and the date_updated cannot be before date_created
- **code** – this is the code name of this simple entity, can be omitted and it will be set to the uppercase version of the nice_name attribute. it accepts string or unicode values. If both the name and code arguments are given the the code property will be set to code, but in any update to name attribute the code also will be updated to the uppercase form of the nice_name attribute

__init__(*name=None*, *description=''*, *created_by=None*, *updated_by=None*, *date_created=datetime.datetime(2011, 1, 13, 18, 37, 26, 361690)*, *date_updated=datetime.datetime(2011, 1, 13, 18, 37, 26, 361702)*, *code=None*)

Methods

`__init__`(1, 13, 18, 37, 26, 1, 13, 18, 37, 26)

Attributes

`code`	
`created_by`	gets and sets the User object who has created this
`date_created`	gets and sets the datetime.datetime object which shows when
`date_updated`	gets and sets the datetime.datetime object which shows when
`description`	the description of the entity
`name`	the name of the entity
`nice_name`	this is the `nice name` of the SimpleEntity. It has the same
`updated_by`	gets and sets the User object who has updated this

created_by

gets and sets the User object who has created this AuditEntity

date_created
gets and sets the datetime.datetime object which shows when this object has been created

date_updated
gets and sets the datetime.datetime object which shows when this object has been updated

description
the description of the entity

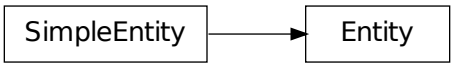
name
the name of the entity

nice_name
this is the `nice name` of the SimpleEntity. It has the same value with the name (contextually) but with a different format like, all the whitespaces replaced by underscores ("_"), all the CamelCase form will be expanded by underscore (_) characters and it is always lowercase.

There is also the `code` attribute which is simple the uppercase form of `nice_name` if it is not defined differently (i.e set to another value).

updated_by
gets and sets the User object who has updated this AuditEntity

stalker.core.models.entity.Entity



class `stalker.core.models.entity.`**Entity**(*tags*=[], ***kwargs*)
Bases: `stalker.core.models.entity.SimpleEntity`

This is the entity class which is derived from the SimpleEntity and adds only tags to the list of parameters.

Parameters **tags** – a list of tag objects related to this entity. tags could be an empty list, or when omitted it will be set to an empty list

__init__(*tags*=[], ***kwargs*)

Methods

`__init__`(**kwargs[, tags])

Attributes

code	
created_by	gets and sets the User object who has created this
date_created	gets and sets the datetime.datetime object which shows when
date_updated	gets and sets the datetime.datetime object which shows when
description	the description of the entity
name	the name of the entity
nice_name	this is the nice name of the SimpleEntity. It has the same
tags	a list of Tag objects which shows the related tags to the
updated_by	gets and sets the User object who has updated this

created_by
gets and sets the User object who has created this AuditEntity

date_created
gets and sets the datetime.datetime object which shows when this object has been created

date_updated
gets and sets the datetime.datetime object which shows when this object has been updated

description
the description of the entity

name
the name of the entity

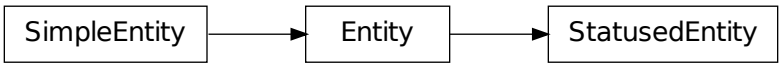
nice_name
this is the nice name of the SimpleEntity. It has the same value with the name (contextually) but with a different format like, all the whitespaces replaced by underscores ("_"), all the CamelCase form will be expanded by underscore (_) characters and it is always lowercase.

There is also the code attribute which is simple the uppercase form of nice_name if it is not defined differently (i.e set to another value).

tags
a list of Tag objects which shows the related tags to the entity

updated_by
gets and sets the User object who has updated this AuditEntity

stalker.core.models.entity.StatusedEntity



class stalker.core.models.entity.**StatusedEntity**(*status_list=[ ]*, *status=0*, ***kwargs*)
Bases: stalker.core.models.entity.Entity

This is a normal entity class that derives from Entity and adds status variables and notes to the parameters list. Any object that needs a status and a corresponding status list should be derived from this class.

Parameters

- **status_list** – this attribute holds a status list object, which shows the possible statuses that this entity could be in. This attribute can not be empty.
- **status** – an integer value which is the index of the status in the status_list attribute. So the value of this attribute couldn't be lower than 0 and higher than the length of the status_list object and nothing other than an integer

__init__(*status_list=[ ]*, *status=0*, ***kwargs*)

Methods

__init__(**kwargs[, status_list, status])

Attributes

code	
created_by	gets and sets the User object who has created this
date_created	gets and sets the datetime.datetime object which shows when
date_updated	gets and sets the datetime.datetime object which shows when
description	the description of the entity
name	the name of the entity
nice_name	this is the nice name of the SimpleEntity. It has the same
status	this is the property that sets and returns the status attribute
status_list	this is the property that sets and returns the status_list attribute
tags	a list of Tag objects which shows the related tags to the
updated_by	gets and sets the User object who has updated this

created_by
gets and sets the User object who has created this AuditEntity

date_created
gets and sets the datetime.datetime object which shows when this object has been created

date_updated
gets and sets the datetime.datetime object which shows when this object has been updated

description
the description of the entity

name
the name of the entity

nice_name
this is the nice name of the SimpleEntity. It has the same value with the name (contextually) but with a different format like, all the whitespaces replaced by underscores ("_"), all the CamelCase form will be expanded by underscore (_) characters and it is always lowercase.

There is also the code attribute which is simple the uppercase form of nice_name if it is not defined differently (i.e set to another value).

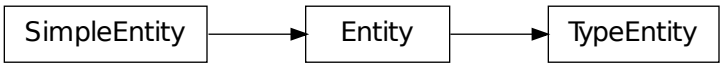
status
this is the property that sets and returns the status attribute

status_list
this is the property that sets and returns the status_list attribute

tags
a list of Tag objects which shows the related tags to the entity

updated_by
gets and sets the User object who has updated this AuditEntity

stalker.core.models.entity.TypeEntity



class stalker.core.models.entity.**TypeEntity**(***kwargs*)

Bases: stalker.core.models.entity.Entity

TypeEntity is the entry point for types.

It is created to group the *Type* objects, so any other classes accepting a `TypeEntity` object can have one of the derived classes, this is done in that way mainly to ease the of creation of only one `TypeTemplate` class and let the others to use this one TypeTemplate class.

It doesn't add any new parameters to it's super.

__init__(***kwargs*)

Methods

`__init__`(**kwargs)	

Attributes

`code`	
`created_by`	gets and sets the User object who has created this
`date_created`	gets and sets the datetime.datetime object which shows when
`date_updated`	gets and sets the datetime.datetime object which shows when
`description`	the description of the entity
`name`	the name of the entity
`nice_name`	this is the `nice name` of the SimpleEntity. It has the same
`tags`	a list of Tag objects which shows the related tags to the
`updated_by`	gets and sets the User object who has updated this

created_by

gets and sets the User object who has created this AuditEntity

date_created

gets and sets the datetime.datetime object which shows when this object has been created

date_updated
gets and sets the datetime.datetime object which shows when this object has been updated

description
the description of the entity

name
the name of the entity

nice_name
this is the `nice name` of the SimpleEntity. It has the same value with the name (contextually) but with a different format like, all the whitespaces replaced by underscores ("_"), all the CamelCase form will be expanded by underscore (_) characters and it is always lowercase.

There is also the `code` attribute which is simple the uppercase form of `nice_name` if it is not defined differently (i.e set to another value).

tags
a list of Tag objects which shows the related tags to the entity

updated_by
gets and sets the User object who has updated this AuditEntity

stalker.core.models.error

Exceptions

`LoginError`(value)	Raised when the login information is not correct or not correlate with

stalker.core.models.error.LoginError

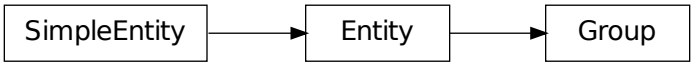
exception `stalker.core.models.error.`**LoginError**(*value*)
Raised when the login information is not correct or not correlate with the data in the database

stalker.core.models.group

Classes

`Group`(**kwargs[, tags])	the group class

stalker.core.models.group.Group



class stalker.core.models.group.**Group**(*tags*=[ ], ***kwargs*)
Bases: stalker.core.models.entity.Entity

the group class

__init__(*tags*=[ ], ***kwargs*)

Methods

__init__(**kwargs[, tags])	

Attributes

code	
created_by	gets and sets the User object who has created this
date_created	gets and sets the datetime.datetime object which shows when
date_updated	gets and sets the datetime.datetime object which shows when
description	the description of the entity
name	the name of the entity
nice_name	this is the nice name of the SimpleEntity. It has the same
tags	a list of Tag objects which shows the related tags to the
updated_by	gets and sets the User object who has updated this

created_by
gets and sets the User object who has created this AuditEntity

date_created
gets and sets the datetime.datetime object which shows when this object has been created

date_updated
gets and sets the datetime.datetime object which shows when this object has been updated

description
the description of the entity

name
the name of the entity

nice_name
this is the nice name of the SimpleEntity. It has the same value with the name (contextually) but with a different format like, all the whitespaces replaced by underscores ("_"), all the CamelCase form will be expanded by underscore (_) characters and it is always lowercase.

There is also the code attribute which is simple the uppercase form of nice_name if it is not defined differently (i.e set to another value).

tags
a list of Tag objects which shows the related tags to the entity

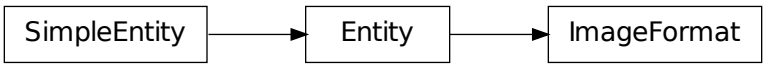
updated_by
gets and sets the User object who has updated this AuditEntity

stalker.core.models.imageFormat

Classes

ImageFormat(**kwargs[, width, height, ...])	the image format class

stalker.core.models.imageFormat.ImageFormat



class stalker.core.models.imageFormat.**ImageFormat**(*width=None*, *height=None*, *pixel_aspect=1.0*, *print_resolution=300*, ***kwargs*)

Bases: stalker.core.models.entity.Entity

the image format class

adds up this parameters to the SimpleEntity:

Parameters

- **width** – the width of the format, it cannot be zero or negative, if a float number is given it will be converted to integer
- **height** – the height of the format, it cannot be zero or negative, if a float number is given it will be converted to integer
- **pixel_aspect** – the pixel aspect ratio of the current ImageFormat object, it can not be zero or negative, and if given as an integer it will be converted to a float, the default value is 1.0
- **print_resolution** – the print resolution of the ImageFormat given as DPI (dot-per-inch). It can not be zero or negative

__init__(*width=None*, *height=None*, *pixel_aspect=1.0*, *print_resolution=300*, ***kwargs*)

Methods

__init__(**kwargs[, width, height, ...])	

Attributes

code	
created_by	gets and sets the User object who has created this
date_created	gets and sets the datetime.datetime object which shows when
date_updated	gets and sets the datetime.datetime object which shows when
description	the description of the entity
device_aspect	returns the device aspect
height	this is a property to set and get the height of the
name	the name of the entity
nice_name	this is the nice name of the SimpleEntity. It has the same
pixel_aspect	this is a property to set and get the pixel_aspect of the
print_resolution	this is a property to set and get the print_resolution of the
tags	a list of Tag objects which shows the related tags to the
updated_by	gets and sets the User object who has updated this
width	this is a property to set and get the width of the

created_by
gets and sets the User object who has created this AuditEntity

date_created
gets and sets the datetime.datetime object which shows when this object has been created

date_updated
gets and sets the datetime.datetime object which shows when this object has been updated

description
the description of the entity

device_aspect
returns the device aspect

because the device_aspect is calculated from the width/height*pixel formula, this property is read-only.

height
this is a property to set and get the height of the image_format

- the height should be set to a positif non-zero integer
- integers are also accepted but will be converted to float
- for improper inputs the object will raise a ValueError

name
the name of the entity

nice_name
this is the `nice name` of the SimpleEntity. It has the same value with the name (contextually) but with a different format like, all the whitespaces replaced by underscores ("_"), all the CamelCase form will be expanded by underscore (_) characters and it is always lowercase.

There is also the `code` attribute which is simple the uppercase form of `nice_name` if it is not defined differently (i.e set to another value).

pixel_aspect
this is a property to set and get the pixel_aspect of the ImageFormat

- the pixel_aspect should be set to a positif non-zero float
- integers are also accepted but will be converted to float
- for improper inputs the object will raise a ValueError

print_resolution
this is a property to set and get the print_resolution of the ImageFormat

- it should be set to a positif non-zero float or integer
- integers are also accepted but will be converted to float
- for improper inputs the object will raise a ValueError

tags
a list of Tag objects which shows the related tags to the entity

updated_by
gets and sets the User object who has updated this AuditEntity

width
this is a property to set and get the width of the image_format

- the width should be set to a positif non-zero integer
- integers are also accepted but will be converted to float
- for improper inputs the object will raise a ValueError

stalker.core.models.link

Classes

`Link`(**kwargs[, path, filename, type])	Holds data about external links.

stalker.core.models.link.Link

class stalker.core.models.link.**Link**(*path=''*, *filename=''*, *type=None*, ***kwargs*)
Bases: stalker.core.models.entity.Entity

Holds data about external links.

Links are all about to give some external information to the current entity (external to the database, so it can be something on the Repository or in the Web). The link type is defined by the LinkType object and it can be anything like *General*, *File*, *Folder*, *WebPage*, *Image*, *ImageSequence*, *Movie*, *Text* etc. (you can also use multiple Tag objects to adding more information, and filtering back). Again it is defined by the needs of the studio.

Parameters

- **path** – The Path to the link, it can be a path to a file in the file system, or a web page. Setting path to None or an empty string is not accepted and causes a ValueError to be raised.
- **filename** – The file name part of the link url, for file sequences use "#" in place of the numerator (Nuke style). Setting filename to None or an empty string is not accepted and causes a ValueError to be raised.
- **type_** – The type of the link. It should be an instance of LinkType, the type can not be None or anything other than a LinkType object. Specifies the link type, can be an LinkType with name Image, Movie/Video, Sound etc.

__init__(*path=''*, *filename=''*, *type=None*, ***kwargs*)

Methods

__init__(**kwargs[, path, filename, type])

Attributes

code	
created_by	gets and sets the User object who has created this
date_created	gets and sets the datetime.datetime object which shows when
date_updated	gets and sets the datetime.datetime object which shows when
description	the description of the entity
filename	the filename part of the url to the link, it can not be None or
name	the name of the entity
nice_name	this is the `nice name` of the SimpleEntity. It has the same
path	the path part of the url to the link, it can not be None or an
tags	a list of Tag objects which shows the related tags to the
type	the type of the link, it should be a
updated_by	gets and sets the User object who has updated this

created_by
gets and sets the User object who has created this AuditEntity

date_created
gets and sets the datetime.datetime object which shows when this object has been created

date_updated
gets and sets the datetime.datetime object which shows when this object has been updated

description
the description of the entity

filename
the filename part of the url to the link, it can not be None or an empty string, it should be a string or unicode

name
the name of the entity

nice_name
this is the `nice name` of the SimpleEntity. It has the same value with the name (contextually) but with a different format like, all the whitespaces replaced by underscores ("_"), all the CamelCase form will be expanded by underscore (_) characters and it is always lowercase.

There is also the `code` attribute which is simple the uppercase form of `nice_name` if it is not defined differently (i.e set to another value).

path
the path part of the url to the link, it can not be None or an empty string, it should be a string or unicode

tags
a list of Tag objects which shows the related tags to the entity

type
the type of the link, it should be a `LinkType` object and it can not be None

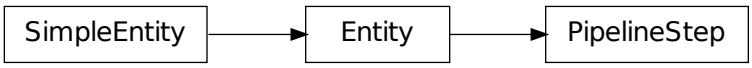
updated_by
gets and sets the User object who has updated this AuditEntity

stalker.core.models.pipelineStep

Classes

`PipelineStep`(**kwargs)	A PipelineStep object represents the general pipeline steps which are

stalker.core.models.pipelineStep.PipelineStep



class `stalker.core.models.pipelineStep.`**`PipelineStep`**(***kwargs*)
Bases: `stalker.core.models.entity.Entity`

A PipelineStep object represents the general pipeline steps which are used around the studio. A couple of examples are:

- Design
- Model
- Rig
- Fur
- Shading
- Previs
- Match Move
- Animation

etc.

Doesn't add any new parameter for its parent class.

`__init__`(***kwargs*)

Methods

`__init__`(**kwargs)	

Attributes

code	
created_by	gets and sets the User object who has created this
date_created	gets and sets the datetime.datetime object which shows when
date_updated	gets and sets the datetime.datetime object which shows when
description	the description of the entity
name	the name of the entity
nice_name	this is the nice name of the SimpleEntity. It has the same
tags	a list of Tag objects which shows the related tags to the
updated_by	gets and sets the User object who has updated this

created_by
gets and sets the User object who has created this AuditEntity

date_created
gets and sets the datetime.datetime object which shows when this object has been created

date_updated
gets and sets the datetime.datetime object which shows when this object has been updated

description
the description of the entity

name
the name of the entity

nice_name
this is the nice name of the SimpleEntity. It has the same value with the name (contextually) but with a different format like, all the whitespaces replaced by underscores ("_"), all the CamelCase form will be expanded by underscore (_) characters and it is always lowercase.

There is also the code attribute which is simple the uppercase form of nice_name if it is not defined differently (i.e set to another value).

tags
a list of Tag objects which shows the related tags to the entity

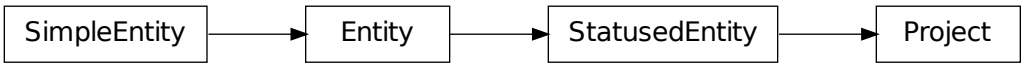
updated_by
gets and sets the User object who has updated this AuditEntity

stalker.core.models.project

Classes

Project(**kwargs[, status_list, status])	the project class

stalker.core.models.project.Project



class stalker.core.models.project.**Project**(*status_list=[ ]*, *status=0*, ***kwargs*)

Bases: stalker.core.models.entity.StatusedEntity

the project class

__init__(*status_list=[ ]*, *status=0*, ***kwargs*)

Methods

__init__(**kwargs[, status_list, status])

Attributes

code	
created_by	gets and sets the User object who has created this
date_created	gets and sets the datetime.datetime object which shows when
date_updated	gets and sets the datetime.datetime object which shows when
description	the description of the entity
name	the name of the entity
nice_name	this is the nice name of the SimpleEntity. It has the same
status	this is the property that sets and returns the status attribute
status_list	this is the property that sets and returns the status_list attribute
tags	a list of Tag objects which shows the related tags to the
updated_by	gets and sets the User object who has updated this

created_by

gets and sets the User object who has created this AuditEntity

date_created

gets and sets the datetime.datetime object which shows when this object has been created

date_updated
gets and sets the datetime.datetime object which shows when this object has been updated

description
the description of the entity

name
the name of the entity

nice_name
this is the `nice name` of the SimpleEntity. It has the same value with the name (contextually) but with a different format like, all the whitespaces replaced by underscores ("_"), all the CamelCase form will be expanded by underscore (_) characters and it is always lowercase.

There is also the `code` attribute which is simple the uppercase form of `nice_name` if it is not defined differently (i.e set to another value).

status
this is the property that sets and returns the status attribute

status_list
this is the property that sets and returns the status_list attribute

tags
a list of Tag objects which shows the related tags to the entity

updated_by
gets and sets the User object who has updated this AuditEntity

stalker.core.models.repository

Classes

`Repository`(**kwargs[, linux_path, ...])	Repository is a class to hold repository server data.

stalker.core.models.repository.Repository

class `stalker.core.models.repository.`**Repository**(*linux_path=''*, *windows_path=''*, *osx_path=''*, ***kwargs*)

Bases: `stalker.core.models.entity.Entity`

Repository is a class to hold repository server data. A repository is a network share that all users have access to.

A studio can create several repositories, for example, one for movie projects and one for commercial projects.

A repository also defines the default paths for linux, windows and mac fileshares.

Parameters

- **linux_path** – shows the linux path of the repository root, should be a string
- **osx_path** – shows the mac osx path of the repository root, should be a string
- **windows_path** – shows the windows path of the repository root, should be a string

__init__(*linux_path=''*, *windows_path=''*, *osx_path=''*, ***kwargs*)

Methods

`__init__`(**kwargs[, linux_path, ...])	

Attributes

`code`	
`created_by`	gets and sets the User object who has created this
`date_created`	gets and sets the datetime.datetime object which shows when
`date_updated`	gets and sets the datetime.datetime object which shows when
`description`	the description of the entity
`linux_path`	property that helps to set and get linux_path values
`name`	the name of the entity
`nice_name`	this is the `nice name` of the SimpleEntity. It has the same
`osx_path`	property that helps to set and get osx_path values
`path`	property that helps to get path for the current os
`tags`	a list of Tag objects which shows the related tags to the
`updated_by`	gets and sets the User object who has updated this
`windows_path`	property that helps to set and get windows_path values

created_by
gets and sets the User object who has created this AuditEntity

date_created
gets and sets the datetime.datetime object which shows when this object has been created

date_updated
gets and sets the datetime.datetime object which shows when this object has been updated

description
the description of the entity

linux_path
property that helps to set and get linux_path values

name
the name of the entity

nice_name
this is the `nice name` of the SimpleEntity. It has the same value with the name (contextually) but with a different format like, all the whitespaces replaced by underscores ("_"), all the CamelCase form will be expanded by underscore (_) characters and it is always lowercase.

There is also the `code` attribute which is simple the uppercase form of `nice_name` if it is not defined differently (i.e set to another value).

osx_path
property that helps to set and get osx_path values

path
property that helps to get path for the current os

tags
a list of Tag objects which shows the related tags to the entity

updated_by
gets and sets the User object who has updated this AuditEntity

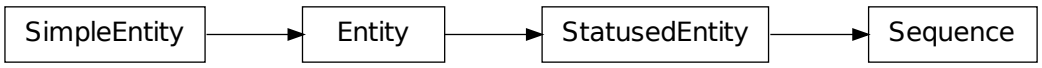
windows_path
property that helps to set and get windows_path values

stalker.core.models.sequence

Classes

`Sequence`(**kwargs[, status_list, status])	the sequence class

stalker.core.models.sequence.Sequence



class `stalker.core.models.sequence.`**Sequence**(*status_list=[ ]*, *status=0*, ***kwargs*)
Bases: `stalker.core.models.entity.StatusedEntity`

the sequence class

__init__(*status_list=[ ]*, *status=0*, ***kwargs*)

Methods

`__init__`(**kwargs[, status_list, status])

Attributes

code	
created_by	gets and sets the User object who has created this
date_created	gets and sets the datetime.datetime object which shows when
date_updated	gets and sets the datetime.datetime object which shows when
description	the description of the entity
name	the name of the entity
nice_name	this is the `nice name` of the SimpleEntity. It has the same
status	this is the property that sets and returns the status attribute
status_list	this is the property that sets and returns the status_list attribute
tags	a list of Tag objects which shows the related tags to the
updated_by	gets and sets the User object who has updated this

created_by
gets and sets the User object who has created this AuditEntity

date_created
gets and sets the datetime.datetime object which shows when this object has been created

date_updated
gets and sets the datetime.datetime object which shows when this object has been updated

description
the description of the entity

name
the name of the entity

nice_name
this is the `nice name` of the SimpleEntity. It has the same value with the name (contextually) but with a different format like, all the whitespaces replaced by underscores ("_"), all the CamelCase form will be expanded by underscore (_) characters and it is always lowercase.

There is also the `code` attribute which is simple the uppercase form of `nice_name` if it is not defined differently (i.e set to another value).

status
this is the property that sets and returns the status attribute

status_list
this is the property that sets and returns the status_list attribute

tags
a list of Tag objects which shows the related tags to the entity

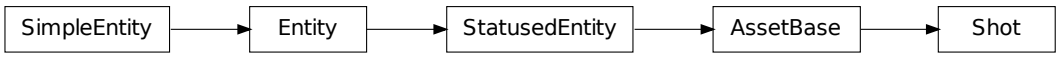
updated_by
gets and sets the User object who has updated this AuditEntity

stalker.core.models.shot

Classes

Shot(**kwargs[, status_list, status])	The Shot class to manage Shot data.

stalker.core.models.shot.Shot



class stalker.core.models.shot.**Shot**(*status_list=[]*, *status=0*, ***kwargs*)
Bases: stalker.core.models.assetBase.AssetBase

The Shot class to manage Shot data.

__init__(*status_list=[]*, *status=0*, ***kwargs*)

Methods

__init__(**kwargs[, status_list, status])	

Attributes

code	
created_by	gets and sets the User object who has created this
date_created	gets and sets the datetime.datetime object which shows when
date_updated	gets and sets the datetime.datetime object which shows when
description	the description of the entity
name	the name of the entity
nice_name	this is the nice name of the SimpleEntity. It has the same
status	this is the property that sets and returns the status attribute
status_list	this is the property that sets and returns the status_list attribute
tags	a list of Tag objects which shows the related tags to the
updated_by	gets and sets the User object who has updated this

created_by
gets and sets the User object who has created this AuditEntity

date_created
gets and sets the datetime.datetime object which shows when this object has been created

date_updated
gets and sets the datetime.datetime object which shows when this object has been updated

description
the description of the entity

name
the name of the entity

nice_name
this is the nice name of the SimpleEntity. It has the same value with the name (contextually) but with a different format like, all the whitespaces replaced by underscores ("_"), all the CamelCase form will be expanded by underscore (_) characters and it is always lowercase.

There is also the code attribute which is simple the uppercase form of nice_name if it is not defined differently (i.e set to another value).

status
this is the property that sets and returns the status attribute

status_list
this is the property that sets and returns the status_list attribute

tags
a list of Tag objects which shows the related tags to the entity

updated_by
gets and sets the User object who has updated this AuditEntity

stalker.core.models.status

Classes

Status(**kwargs[, short_name, thumbnail])	The Status class
StatusList(**kwargs[, statuses])	the list version of the Status

stalker.core.models.status.Status

class stalker.core.models.status.**Status**(*short_name=None*, *thumbnail=None*, ***kwargs*)

Bases: stalker.core.models.entity.Entity

The Status class

Parameters **short_name** – the short_name of the status name, keep it as simple as possible, the string will be formated to have all upper-case and no white spaces at the beggining and at the end of the attribute

__init__(*short_name=None*, *thumbnail=None*, ***kwargs*)

Methods

__init__(**kwargs[, short_name, thumbnail])	

Attributes

code	
created_by	gets and sets the User object who has created this
date_created	gets and sets the datetime.datetime object which shows when
date_updated	gets and sets the datetime.datetime object which shows when
description	the description of the entity
name	the name of the entity
nice_name	this is the nice name of the SimpleEntity. It has the same
short_name	returns the short_name property
tags	a list of Tag objects which shows the related tags to the
updated_by	gets and sets the User object who has updated this

created_by
gets and sets the User object who has created this AuditEntity

date_created
gets and sets the datetime.datetime object which shows when this object has been created

date_updated
gets and sets the datetime.datetime object which shows when this object has been updated

description
the description of the entity

name
the name of the entity

nice_name
this is the nice name of the SimpleEntity. It has the same value with the name (contextually) but with a different format like, all the whitespaces replaced by underscores ("_"), all the CamelCase form will be expanded by underscore (_) characters and it is always lowercase.

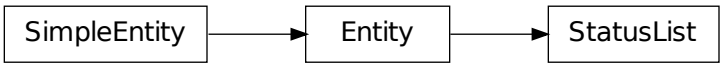
There is also the code attribute which is simple the uppercase form of nice_name if it is not defined differently (i.e set to another value).

short_name
returns the short_name property

tags
a list of Tag objects which shows the related tags to the entity

updated_by
gets and sets the User object who has updated this AuditEntity

stalker.core.models.status.StatusList



class stalker.core.models.status.**StatusList**(*statuses*=[ ], ***kwargs*)
Bases: stalker.core.models.entity.Entity

the list version of the Status

Holds multiple statuses to be used as a choice list for several other classes

Parameters **statuses** – this is a list of status objects, so you can prepare different StatusList objects for different kind of entities

__init__(*statuses*=[ ], ***kwargs*)

Methods

__init__(**kwargs[, statuses])	

Attributes

code	
created_by	gets and sets the User object who has created this
date_created	gets and sets the datetime.datetime object which shows when
date_updated	gets and sets the datetime.datetime object which shows when
description	the description of the entity
name	the name of the entity
nice_name	this is the nice name of the SimpleEntity. It has the same
statuses	this is the property that sets and returns the statuses, or
tags	a list of Tag objects which shows the related tags to the
updated_by	gets and sets the User object who has updated this

created_by
gets and sets the User object who has created this AuditEntity

date_created
gets and sets the datetime.datetime object which shows when this object has been created

date_updated
gets and sets the datetime.datetime object which shows when this object has been updated

description
the description of the entity

name
the name of the entity

nice_name
this is the `nice name` of the SimpleEntity. It has the same value with the name (contextually) but with a different format like, all the whitespaces replaced by underscores ("_"), all the CamelCase form will be expanded by underscore (_) characters and it is always lowercase.

There is also the `code` attribute which is simple the uppercase form of `nice_name` if it is not defined differently (i.e set to another value).

statuses
this is the property that sets and returns the statuses, or namely the status list of this StatusList object

tags
a list of Tag objects which shows the related tags to the entity

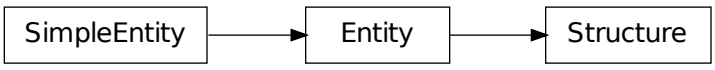
updated_by
gets and sets the User object who has updated this AuditEntity

stalker.core.models.structure

Classes

`Structure`(**kwargs[, project_template, ...])	A structure object is the place to hold data about how the physical

stalker.core.models.structure.Structure



class `stalker.core.models.structure.`**`Structure`**(*project_template=''*, *asset_templates=[]*, *reference_templates=[]*, ***kwargs*)

Bases: `stalker.core.models.entity.Entity`

A structure object is the place to hold data about how the physical files are arranged in the `Repository`.

Parameters

- **project_template** – it is a string holding several lines of text showing the folder structure of the project. Whenever a project is created, folders are created by looking at this folder template.

  The template string can have Jinja2 directives. These variables are given to the template engine:

  – *project*: holds the current `Project` object using this structure, so you can use {{project.code}} or {{project.sequences}} kind of variables in the Jinja2 template

- **asset_templates** – holds `TypeTemplate` objects with an `AssetType` connected to its *type* attribute, which can help specifying templates based on the related `AssetType` object.

  Testing a second paragraph addition.

- **reference_templates** – holds `TypeTemplate` objects, which can help specifying templates based on the given `LinkType` object

This templates are used in creation of Project folder structure and also while interacting with the assets and references in the current `Project`. You can create one project structure for *Commmercials* and another project structure for *Movies* and another one for *Print* projects etc. and can reuse them with new projects.

__init__(*project_template=''*, *asset_templates=[ ]*, *reference_templates=[ ]*, ***kwargs*)

Methods

`__init__`(**kwargs[, project_template, ...])

Attributes

asset_templates	A list of
code	
created_by	gets and sets the User object who has created this
date_created	gets and sets the datetime.datetime object which shows when
date_updated	gets and sets the datetime.datetime object which shows when
description	the description of the entity
name	the name of the entity
nice_name	this is the nice name of the SimpleEntity. It has the same
project_template	A string which shows the folder structure of the current project.
reference_templates	A list of
tags	a list of Tag objects which shows the related tags to the
updated_by	gets and sets the User object who has updated this

asset_templates
A list of TypeTemplate objects which gives information about the Asset Version file placements

created_by
gets and sets the User object who has created this AuditEntity

date_created
gets and sets the datetime.datetime object which shows when this object has been created

date_updated
gets and sets the datetime.datetime object which shows when this object has been updated

description
the description of the entity

name
the name of the entity

nice_name
this is the nice name of the SimpleEntity. It has the same value with the name (contextually) but with a different format like, all the whitespaces replaced by underscores ("_"), all the CamelCase form will be expanded by underscore (_) characters and it is always lowercase.

There is also the code attribute which is simple the uppercase form of nice_name if it is not defined differently (i.e set to another value).

project_template
A string which shows the folder structure of the current project. It can have Jinja2 directives. See the documentation of Structure object for more information

reference_templates
A list of TypeTemplate objects which gives information about the placement of references to entities

tags
a list of Tag objects which shows the related tags to the entity

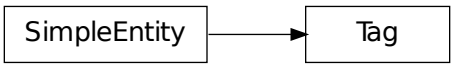
updated_by
gets and sets the User object who has updated this AuditEntity

stalker.core.models.tag

Classes

Tag(**kwargs)	the tag class

stalker.core.models.tag.Tag



class stalker.core.models.tag.**Tag**(***kwargs*)
Bases: stalker.core.models.entity.SimpleEntity

the tag class

__init__(***kwargs*)

Methods

__init__(**kwargs)	

Attributes

code	
created_by	gets and sets the User object who has created this
date_created	gets and sets the datetime.datetime object which shows when
date_updated	gets and sets the datetime.datetime object which shows when
description	the description of the entity
name	the name of the entity
nice_name	this is the nice name of the SimpleEntity. It has the same
updated_by	gets and sets the User object who has updated this

created_by
gets and sets the User object who has created this AuditEntity

date_created
gets and sets the datetime.datetime object which shows when this object has been created

date_updated
gets and sets the datetime.datetime object which shows when this object has been updated

description
the description of the entity

name
the name of the entity

nice_name
this is the nice name of the SimpleEntity. It has the same value with the name (contextually) but with a different format like, all the whitespaces replaced by underscores ("_"), all the CamelCase form will be expanded by underscore (_) characters and it is always lowercase.

There is also the code attribute which is simple the uppercase form of nice_name if it is not defined differently (i.e set to another value).

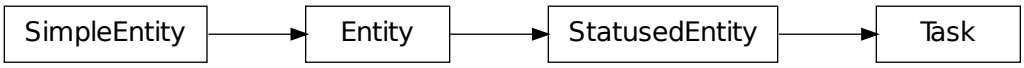
updated_by
gets and sets the User object who has updated this AuditEntity

stalker.core.models.task

Classes

Task(**kwargs[, status_list, status])	the task class

stalker.core.models.task.Task



class stalker.core.models.task.**Task**(*status_list=[ ]*, *status=0*, ***kwargs*)

Bases: stalker.core.models.entity.StatusedEntity

the task class

__init__(*status_list=[ ]*, *status=0*, ***kwargs*)

Methods

__init__(**kwargs[, status_list, status])

Attributes

code	
created_by	gets and sets the User object who has created this
date_created	gets and sets the datetime.datetime object which shows when
date_updated	gets and sets the datetime.datetime object which shows when
description	the description of the entity
name	the name of the entity
nice_name	this is the nice name of the SimpleEntity. It has the same
status	this is the property that sets and returns the status attribute
status_list	this is the property that sets and returns the status_list attribute
tags	a list of Tag objects which shows the related tags to the
updated_by	gets and sets the User object who has updated this

created_by

gets and sets the User object who has created this AuditEntity

date_created

gets and sets the datetime.datetime object which shows when this object has been created

date_updated
gets and sets the datetime.datetime object which shows when this object has been updated

description
the description of the entity

name
the name of the entity

nice_name
this is the `nice name` of the SimpleEntity. It has the same value with the name (contextually) but with a different format like, all the whitespaces replaced by underscores ("_"), all the CamelCase form will be expanded by underscore (_) characters and it is always lowercase.

There is also the `code` attribute which is simple the uppercase form of `nice_name` if it is not defined differently (i.e set to another value).

status
this is the property that sets and returns the status attribute

status_list
this is the property that sets and returns the status_list attribute

tags
a list of Tag objects which shows the related tags to the entity

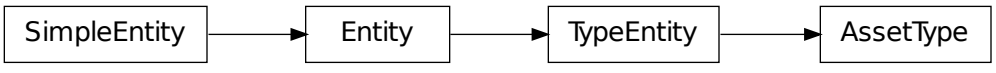
updated_by
gets and sets the User object who has updated this AuditEntity

stalker.core.models.types

Classes

`AssetType`(**kwargs[, steps])	The AssetType class holds the information about the asset type.
`LinkType`(**kwargs)	The type of `Link` is hold in LinkType
`ProjectType`(**kwargs)	Helps to create different type of
`TypeTemplate`(**kwargs[, path_code, ...])	The TypeTemplate model holds templates for Types.

stalker.core.models.types.AssetType



class `stalker.core.models.types.`**`AssetType`**(*steps*=[], ***kwargs*)
Bases: `stalker.core.models.entity.TypeEntity`

The AssetType class holds the information about the asset type.

One asset type object has information about the pipeline steps that this type of asset needs.

So for example one can create a "Chracter" asset type and then link "Design", "Modeling", "Rig", "Shading" pipeline steps to this asset type object. And then have a "Environment" asset type and then just link "Design", "Modeling", "Shading" pipeline steps to it.

Parameters **steps** – This is a list of PipelineStep objects.

__init__(*steps*=[ ], ***kwargs*)

Methods

`__init__`(**kwargs[, steps])	

Attributes

`code`	
`created_by`	gets and sets the User object who has created this
`date_created`	gets and sets the datetime.datetime object which shows when
`date_updated`	gets and sets the datetime.datetime object which shows when
`description`	the description of the entity
`name`	the name of the entity
`nice_name`	this is the `nice name` of the SimpleEntity. It has the same
`steps`	this is the property that lets you set and get steps attribute
`tags`	a list of Tag objects which shows the related tags to the
`updated_by`	gets and sets the User object who has updated this

created_by
gets and sets the User object who has created this AuditEntity

date_created
gets and sets the datetime.datetime object which shows when this object has been created

date_updated
gets and sets the datetime.datetime object which shows when this object has been updated

description
the description of the entity

name
the name of the entity

nice_name
this is the `nice name` of the SimpleEntity. It has the same value with the name (contextually) but with a different format like, all the whitespaces replaced by underscores ("_"), all the CamelCase form will be expanded by underscore (_) characters and it is always lowercase.

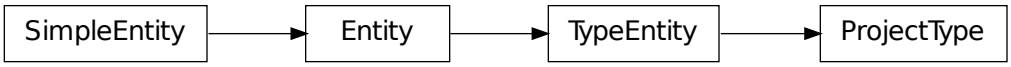
There is also the `code` attribute which is simple the uppercase form of `nice_name` if it is not defined differently (i.e set to another value).

steps
this is the property that lets you set and get steps attribute

tags
a list of Tag objects which shows the related tags to the entity

updated_by
gets and sets the User object who has updated this AuditEntity

stalker.core.models.types.ProjectType



class `stalker.core.models.types.`**`ProjectType`**(***kwargs*)
Bases: `stalker.core.models.entity.TypeEntity`

Helps to create different type of `Project` objects.

Can be used to create different type projects like Commercial, Movie, Still etc.

`__init__`(***kwargs*)

Methods

`__init__`(**kwargs)

Attributes

code	
created_by	gets and sets the User object who has created this
date_created	gets and sets the datetime.datetime object which shows when
date_updated	gets and sets the datetime.datetime object which shows when
description	the description of the entity
name	the name of the entity
nice_name	this is the nice name of the SimpleEntity. It has the same
tags	a list of Tag objects which shows the related tags to the
updated_by	gets and sets the User object who has updated this

created_by
gets and sets the User object who has created this AuditEntity

date_created
gets and sets the datetime.datetime object which shows when this object has been created

date_updated
gets and sets the datetime.datetime object which shows when this object has been updated

description
the description of the entity

name
the name of the entity

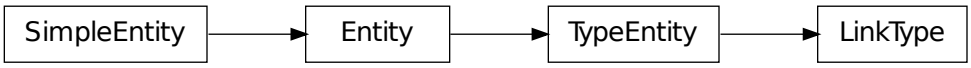
nice_name
this is the nice name of the SimpleEntity. It has the same value with the name (contextually) but with a different format like, all the whitespaces replaced by underscores ("_"), all the CamelCase form will be expanded by underscore (_) characters and it is always lowercase.

There is also the code attribute which is simple the uppercase form of nice_name if it is not defined differently (i.e set to another value).

tags
a list of Tag objects which shows the related tags to the entity

updated_by
gets and sets the User object who has updated this AuditEntity

stalker.core.models.types.LinkType



class stalker.core.models.types.**LinkType**(***kwargs*)
Bases: stalker.core.models.entity.TypeEntity

The type of Link is hold in LinkType objects.

LinkType objects hold the type of the link and it is generaly used by Project to sort things out. See Project object documentation for details.

__init__(***kwargs*)

Methods

__init__(**kwargs)

Attributes

code	
created_by	gets and sets the User object who has created this
date_created	gets and sets the datetime.datetime object which shows when
date_updated	gets and sets the datetime.datetime object which shows when
description	the description of the entity
name	the name of the entity
nice_name	this is the nice name of the SimpleEntity. It has the same
tags	a list of Tag objects which shows the related tags to the
updated_by	gets and sets the User object who has updated this

created_by
gets and sets the User object who has created this AuditEntity

date_created
gets and sets the datetime.datetime object which shows when this object has been created

date_updated
gets and sets the datetime.datetime object which shows when this object has been updated

description
the description of the entity

name
the name of the entity

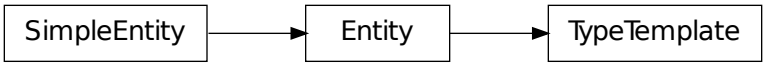
nice_name
this is the `nice name` of the SimpleEntity. It has the same value with the name (contextually) but with a different format like, all the whitespaces replaced by underscores ("_"), all the CamelCase form will be expanded by underscore (_) characters and it is always lowercase.

There is also the `code` attribute which is simple the uppercase form of `nice_name` if it is not defined differently (i.e set to another value).

tags
a list of Tag objects which shows the related tags to the entity

updated_by
gets and sets the User object who has updated this AuditEntity

stalker.core.models.types.TypeTemplate



class `stalker.core.models.types.`**`TypeTemplate`**(*path_code=''*, *file_code=''*, *type=None*, ***kwargs*)

Bases: `stalker.core.models.entity.Entity`

The TypeTemplate model holds templates for Types.

TypeTemplate objects help to specify where to place a file related to `TypeEntity` objects and its derived classes.

The first very important usage of TypeTemplates is to place asset `Version`'s to proper places inside a `Project`'s `Structure`.

Parameters

- **path_code** – The Jinja2 template code which specifies the path of the given item. It is relative to the project root which is in general {{repository.path}}/{{project.code}}/
- **file_code** – The Jinja2 template code which specifies the file name of the given item
- **type_** – A `TypeEntity` object or any other class which is derived from TypeEntity.

Examples:

A template for asset versions can have this parameters:

```
from stalker import db
from satlker.db import auth
from stalker.core.models import types, pipelineStep
```

```
# setup the default database
db.setup()

# store the query method for ease of use
session = db.session
query = db.session.query

# login to the system as admin
admin = auth.login("admin", "admin")

# create a couple of variables
path_code = "ASSETS/{{asset_type.name}}/{{pipeline_step.code}}"

file_code = "{{asset.name}}_{{take.name}}_{{asset_type.name}}_v{{version.version_number}}"

# create a pipeline step object
modelingStep = pipelineStep.PipelineStep(
    name="Modeling",
    code="MODEL",
    description="The modeling step of the asset",
    created_by=admin
)

# create a "Character" AssetType with only one step
typeObj = types.AssetType(
    name="Character",
    description="this is the character asset type",
    created_by=admin,
    steps=[modelingStep]
)

# now create our TypeTemplate
char_template = types.TypeTemplate(
    name="Character",
    description="this is the template which explains how to place Character assets",
    path_code=path_code,
    file_code=file_code,
    type=typeObj,
)

# assign this type template to the structure of the project with id=101
myProject = query(project.Project).filter_by(id=101).first()

# append the type template to the structures' asset templates
myProject.structure.asset_templates.append(char_template)

session.commit()
```

Now with the code above, whenever a new `Version` created for a **Character** asset, Stalker will automatically place the related file to a certain folder and with a certain file name defined by the template. For example the above template should render something like below for Windows:

```
|- M:\\PROJECTS  --> {{repository.path}}
 |- PRENSESIN_UYKUSU  --> {{project.code}}
  |- ASSETS  --> "ASSETS"
   |- Character  --> {{asset_type.name}}
    |- Olum  --> {{asset.name}}
     |- MODEL  --> {{pipeline_step.code}}
```

```
|- Olum_MAIN_MODEL_v001.ma --> {{asset.name}}_{{take.name}}_{{asset_type.name}}_v{{versi
```

And one of the good side is you can create a version from Linux, Windows or OSX all the paths will be correctly handled by Stalker.

__init__(*path_code=''*, *file_code=''*, *type=None*, ***kwargs*)

Methods

`__init__`(**kwargs[, path_code, file_code, type])	

Attributes

`code`	
`created_by`	gets and sets the User object who has created this
`date_created`	gets and sets the datetime.datetime object which shows when
`date_updated`	gets and sets the datetime.datetime object which shows when
`description`	the description of the entity
`file_code`	this is the property that helps you assign values to
`name`	the name of the entity
`nice_name`	this is the `nice name` of the SimpleEntity. It has the same
`path_code`	this is the property that helps you assign values to
`tags`	a list of Tag objects which shows the related tags to the
`type`	the target type this template should work on, should be an
`updated_by`	gets and sets the User object who has updated this

created_by
gets and sets the User object who has created this AuditEntity

date_created
gets and sets the datetime.datetime object which shows when this object has been created

date_updated
gets and sets the datetime.datetime object which shows when this object has been updated

description
the description of the entity

file_code
this is the property that helps you assign values to file_code attribute

name
the name of the entity

nice_name
this is the `nice name` of the SimpleEntity. It has the same value with the name (contextually) but with a different format like, all the whitespaces replaced by underscores ("_"), all the CamelCase form will be expanded by underscore (_) characters and it is always lowercase.

There is also the `code` attribute which is simple the uppercase form of `nice_name` if it is not defined differently (i.e set to another value).

path_code
this is the property that helps you assign values to path_code attribute

tags
a list of Tag objects which shows the related tags to the entity

type
the target type this template should work on, should be an instance of `TypeEntity`

updated_by
gets and sets the User object who has updated this AuditEntity

stalker.core.models.user

Classes

`User`(**kwargs[, department, email, ...])	The user class is designed to hold data about a User in the system.

stalker.core.models.user.User

class `stalker.core.models.user.`**User**(*department=None*, *email=''*, *first_name=''*, *last_name=''*, *login_name=''*, *password=''*, *permission_groups=[]*, *projects=[]*, *projects_lead=[]*, *sequences_lead=[]*, *tasks=[]*, *last_login=None*, ***kwargs*)

Bases: `stalker.core.models.entity.Entity`

The user class is designed to hold data about a User in the system.

Parameters

- **email** – holds the e-mail of the user, should be in [part1]@[part2] format
- **last_login** – it is a datetime.datetime object holds the last login date of the user (not implemented yet)

- **login_name** – it is the login name of the user, it should be all lower case. Giving a string or unicode that has uppercase letters, it will be converted to lower case. It can not be an empty string or None and it can not contain any white space inside. login_name parameter is a synonym for *name*, while creating a User object you don't need to specify both of them, one is enough and if the two is given *name* will be used.
- **first_name** – it is the first name of the user, must be a string or unicode, middle name also can be added here, so it accepts white-spaces in the variable, but it will truncate the white spaces at the beginin and at the end of the variable and it can not be empty or None
- **last_name** – it is the last name of the user, must be a string or unicode, again it can not contain any white spaces at the beggining and at the end of the variable and it can be an empty string or None
- **department** – it is the department of the current user. It should be a Department object. One user can only be listed in one department. A user is allowed to have no department to make it easy to create a new user and create the department and assign the user it later.
- **password** – it is the password of the user, can contain any character and it should be scrambled by using the key from the system preferences
- **permission_groups** – it is a list of permission groups that this user is belong to
- **tasks** – it is a list of Task objects which holds the tasks that this user has been assigned to
- **projects** – it is a list of Project objects which holds the projects that this user is a part of
- **projects_lead** – it is a list of Project objects that this user is the leader of, it is for back refefrencing purposes
- **sequences_lead** – it is a list of Sequence objects that this user is the leader of, it is for back referencing purposes

__init__(*department=None*, *email=''*, *first_name=''*, *last_name=''*, *login_name=''*, *password=''*, *permission_groups=[]*, *projects=[]*, *projects_lead=[]*, *sequences_lead=[]*, *tasks=[]*, *last_login=None*, ***kwargs*)

Methods

__init__(**kwargs[, department, email, ...])

Attributes

code	
created_by	gets and sets the User object who has created this
date_created	gets and sets the datetime.datetime object which shows when
date_updated	gets and sets the datetime.datetime object which shows when
department	department of the user, it is a
description	the description of the entity
email	email of the user, accepts strings or unicodes
first_name	first name of the user, accepts string or unicode
last_login	last login time of the user as a datetime.datetime instance
last_name	last name of the user, accepts string or unicode
login_name	login name of the user, accepts string or unicode, also sets
name	the name of the user object, it is the synonym for the
nice_name	this is the nice name of the SimpleEntity. It has the same
password	password of the user, it is scrambled before stored in the
permission_groups	permission groups that this users is a member of, accepts
projects	projects those the current user assigned to, accepts
projects_lead	projects lead by this current user, accepts
sequences_lead	sequences lead by this user, accpets
tags	a list of Tag objects which shows the related tags to the
tasks	tasks assigned to the current user, accepts
updated_by	gets and sets the User object who has updated this

created_by
gets and sets the User object who has created this AuditEntity

date_created
gets and sets the datetime.datetime object which shows when this object has been created

date_updated
gets and sets the datetime.datetime object which shows when this object has been updated

department
department of the user, it is a Department object

description
the description of the entity

email
email of the user, accepts strings or unicodes

first_name
first name of the user, accepts string or unicode

last_login
last login time of the user as a datetime.datetime instance

last_name
last name of the user, accepts string or unicode

login_name
login name of the user, accepts string or unicode, also sets the name attribute

name
the name of the user object, it is the synonym for the login_name

nice_name
this is the `nice name` of the SimpleEntity. It has the same value with the name (contextually) but with a different format like, all the whitespaces replaced by underscores ("_"), all the CamelCase form will be expanded by underscore (_) characters and it is always lowercase.

There is also the `code` attribute which is simple the uppercase form of `nice_name` if it is not defined differently (i.e set to another value).

password
password of the user, it is scrambled before stored in the _password attribute

permission_groups
permission groups that this users is a member of, accepts `Group` object

projects
projects those the current user assigned to, accepts `Project` object

projects_lead
projects lead by this current user, accepts `Project` object

sequences_lead
sequences lead by this user, accpets `Sequence` objects

tags
a list of Tag objects which shows the related tags to the entity

tasks
tasks assigned to the current user, accepts `Task` objects

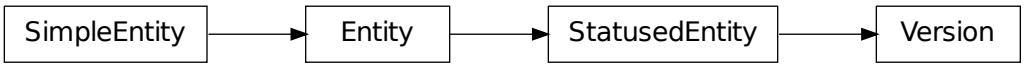
updated_by
gets and sets the User object who has updated this AuditEntity

stalker.core.models.version

Classes

`Version`(**kwargs[, status_list, status])	The Version class is the connection of Assets to versions of that asset.

stalker.core.models.version.Version



class stalker.core.models.version.**Version**(*status_list=[ ]*, *status=0*, ***kwargs*)
Bases: stalker.core.models.entity.StatusedEntity

The Version class is the connection of Assets to versions of that asset. So it connects the Assets to file system, and manages the files as versions.

__init__(*status_list=[ ]*, *status=0*, ***kwargs*)

Methods

__init__(**kwargs[, status_list, status])

Attributes

code	
created_by	gets and sets the User object who has created this
date_created	gets and sets the datetime.datetime object which shows when
date_updated	gets and sets the datetime.datetime object which shows when
description	the description of the entity
name	the name of the entity
nice_name	this is the nice name of the SimpleEntity. It has the same
status	this is the property that sets and returns the status attribute
status_list	this is the property that sets and returns the status_list attribute
tags	a list of Tag objects which shows the related tags to the
updated_by	gets and sets the User object who has updated this

created_by
gets and sets the User object who has created this AuditEntity

date_created
gets and sets the datetime.datetime object which shows when this object has been created

date_updated
gets and sets the datetime.datetime object which shows when this object has been updated

description
the description of the entity

name
the name of the entity

nice_name
this is the `nice name` of the SimpleEntity. It has the same value with the name (contextually) but with a different format like, all the whitespaces replaced by underscores ("_"), all the CamelCase form will be expanded by underscore (_) characters and it is always lowercase.

There is also the `code` attribute which is simple the uppercase form of `nice_name` if it is not defined differently (i.e set to another value).

status
this is the property that sets and returns the status attribute

status_list
this is the property that sets and returns the status_list attribute

tags
a list of Tag objects which shows the related tags to the entity

updated_by
gets and sets the User object who has updated this AuditEntity

1.2.7 Indices and tables

- *genindex*
- *search*

PYTHON MODULE INDEX

S