MouseDB Documentation

Release 1.1.1

Dave Bridges, Ph.D.

October 27, 2012

CONTENTS

Contents:

CHAPTER

ONE

MOUSEDB CONCEPTS

Data storage for MouseDB is separated into packages which contain information about animals, and information collected about animals. There is also a separate module for timed matings of animals. This document will describe the basics of how data is stored in each of these modules.

1.1 Animal Module

Animals are tracked as individual entities, and given associations to breeding cages to follow ancestry, and strains. Animals are the main objects in the database, and most other data is linked to and can be accessed from animals.

1.1.1 Animal

Most parameters about an animal are set within the `Animal` object. Here is where the animals strain, breeding, parentage and many other parameters are included. Animals have foreignkey relationships with both `Strain` and `Breeding`, so an animal may only belong to one of each of those. As an example, a mouse cannot come from more than one Breeding set, and cannot belong to more than one strain.

Backcrosses and Generations

For this software, optional tracking of backcrosses and generations is available and is stored as an attribute of an animal. When an inbred cross is made against a pure background, the backcross increases by 1. When a heterozygote cross is made, the generation increases by one. As an example, for every time a mouse in a C57/BL6 background is crossed against a wildtype C57/B6 mouse, the backcross (but not the generation) increases by one. For every time a mutant strain is crosses against itself (either vs a heterozygote or homozygote of that strain), the generation will increase by one. Backcrosses should typically be performed against a separate colony of purebred mouse, rather than against wild-type alleles of the mutant strain.

1.1.2 Breeding Cages

A breeding cage is defined as a set of one or more male and one or more female mice. Because of this, it is not always clear who the precise parentage of an animal is. If the parentage is known, then the Mother and Father fields can be set for a particular animal. In the case of Active, if an End field is specified, then the Active field is set to False. In the case of Cage, if a Cage is provided, and animals are specified under Male or Females for a Breeding object, then the Cage field for those animals is set to that of the breeding cage. The same is true for both Rack and Rack Position.

1.1.3 Strains

A strain is a set of mice with a similar genetics. Importantly strains are separated from Backgrounds. For example, one might have mice with the genotype ob/ob but these mice may be in either a C57-Black6 or a mixed background. This difference is set at the individual animal level. The result of this is that a query for a particular strain may then need to be filtered to a specific background.

1.2 Data Module

Data (or measurements) can be stored for any type of measurement. Conceptually, several pieces of data belong to an experiment (for example several mice are measured at some time) and several experiments belong to a study. Measurements can be stored independent of experiments and experiments can be performed outside of the context of a study. It is however, perfered that measurements are stored within an experiment and experiments are stored within studies as this will greatly facilitate the organization of the data.

1.2.1 Studies

In general studies are a collection of experiments. These can be grouped together on the basis of animals and/or treatment groups. A study must have at least one treatment group, which defines the animals and their conditions.

1.2.2 Experiments

An experiment is a collection of measurements for a given set of animals. In general, an experiment is defined as a number of measurements take in a given day.

1.2.3 Measurements

A measurement is an animal, an assay and a measurement value. It can be associated with an experiment, or can stand alone as an individual value. Measurements can be viewed in the context of a study, an experiment, a treatment group or an animal by going to the appropriate page.

1.3 Timed Matings Module

Timed matings are a specific type of breeding set. Generally, for these experiments a mating cage is set up and pregnancy is defined by a plug event. Based on this information, the age of an embryo can be estimated. When a breeding cage is defined, one option is to set this cage as a timed mating cage (ie Timed_Mating=True). If this is the case, then a plug event can be registered and recorded for this mating set. If the mother gives birth then this cage is implicitly set as a normal breeding cage.

1.4 Groups Module

This app defines generic Group and License information for a particular installation of MouseDB. Because every page on this site identifies both the Group and data restrictions, at a minimum, group information must be provided upon installation (see installation instructions).

CHAPTER

TWO

MOUSEDB INSTALLATION

2.1 Configuration

MouseDB requires both a database and a webserver to be set up. Ideally, the database should be hosted separately from the webserver and MouseDB installation, but this is not necessary, as both can be used from the same server. If you are using a remote server for the database, it is best to set up a user for this database that can only be accessed from the webserver. If you want to set up several installations (ie for different users or different laboratories), you need separate databases and MouseDB installations for each. You will also need to set up the webserver with different addresses for each installation.

2.2 Software Dependencies

1. **Python**. Requires Version 2.6, is not yet compatible with Python 3.0. Download from http://www.python.org/download/.
2. **MouseDB source code**. Download from one of the following:

1. Using **pip or easy_install**. If setuptools (available at http://pypi.python.org/pypi/setuptools) is installed type **pip install mousedb** at a command prompt.
2. http://github.com/davebridges/mousedb/downloads for the current release. If you will not be contributing to the code, download from here.
3. http://github.com/davebridges/mousedb for the source code via Git. If you might contribute code to the project use the source code.

Downloading and/or unzipping will create a directory named mousedb. You can update to the newest revision at any time either using git or downloading and re-installing the newer version. Changing or updating software versions will not alter any saved data, but you will have to update the localsettings.py file (described below).

3. **Database software**. Recommended to use mysql, available at http://dev.mysql.com/downloads/mysql/ . It is also possible to use SQLite, PostgreSQL, MySQL, or Oracle. See http://docs.djangoproject.com/en/1.2/topics/install/#database-installation for more information. You will also need the python bindings for your database. If using MySQL python-mysql will be installed below.
4. **Webserver**. Apache is recommended, available at http://www.apache.org/dyn/closer.cgi . It is also possible to use FastCGI, SCGI, or AJP. See http://docs.djangoproject.com/en/1.2/howto/deployment/ for more details. The recommended way to use Apache is to download and enable mod_wsgi. See http://code.google.com/p/modwsgi/ for more details.

2.3 Installation

1. Navigate into mousedb folder
2. Run **python setup.py install** to get dependencies. If you installed via pip, this step is not necessary (but wont hurt). This will install the dependencies South, mysql-python and django-ajax-selects.
3. Run **python bootstrap.py** to get the correct version of Django and to set up an isolated environment. This step may take a few minutes.
4. Run **bin\buildout** to generate django, test and wsgi scripts. This step may take a few minutes.

2.4 Database Setup

1. Create a new database. Check the documentation for your database software for the appropriate syntax for this step. You need to record the user, password, host and database name. If you are using SQLite this step is not required.
2. Go to mousedbsrcmousedblocalsettings_empty.py and edit the settings:

- ENGINE: Choose one of 'django.db.backends.postgresql_psycopg2','django.db.backends.postgresql', 'django.db.backends.mysql', 'django.db.backends.sqlite3', 'django.db.backends.oracle' depending on the database software used.
- NAME: database name
- USER: database user
- PASSWORD: database password
- HOST: database host

3. Save this file as **localsettings.py** in the same folder as localsettings_empty.py
4. Migrate into first mousedb directory and enter *django syncdb*. When prompted create a superuser (who will have all availabler permissions) and a password for this user.

2.5 Web Server Setup

You need to set up a server to serve both the django installation and saved files. For the saved files. I recommend using apache for both. The preferred setup is to use Apache2 with mod_wsgi. See http://code.google.com/p/modwsgi/wiki/InstallationInstructions for instructions on using mod_wsgi. The following is a httpd.conf example where the code is placed in **/usr/src/mousedb**:

```
Alias /robots.txt /usr/src/mousedb/src/mousedb/media/robots.txt
Alias /favicon.ico /usr/src/mousedb/src/mousedb/media/favicon.ico

Alias /mousedb-media/ /usr/src/mousedb/src/mousedb/media/
<Directory /usr/src/mousedb/src/mousedb/media>
     Order deny,allow
     Allow from all
</Directory>

Alias /static/ /usr/src/mousedb/src/mousedb/static/
<Directory /usr/src/mousedb/src/mousedb/static>
     Order deny,allow
     Allow from all
```

```
</Directory>

<Directory /usr/src/mousedb/bin>
     Order deny,allow
     Allow from all
</Directory>
WSGIScriptAlias /mousedb /usr/src/mousedb/bin/django.wsgi
```

If you want to restrict access to these files, change the Allow from all directive to specific domains or ip addresses (for example Allow from 192.168.0.0/99 would allow from 192.168.0.0 to 192.168.0.99)

2.6 Enabling of South for Future Migrations

Schema updates will utilize south as a way to alter database tables. This must be enabled initially by entering the following commands from /mousedb/bin:

```
django schemamigration animal --initial
django schemamigration data --initial
django schemamigration groups --initial
django schemamigration timed_mating --initial
django syncdb
django migrate
```

Future schema changes (se the UPGRADE_NOTES.rst file for whether this is necessary) are accomplished by entering:

```
django schemamigration <INDICATED_APP> --auto
django migrate <INDICATED_APP>
```

2.7 Final Configuration and User Setup

- Go to *servername/mousedb/admin/groups/group/1* and name your research group and select a license if desired
- Go to *servername/mousedb/admin/auth/users/* and create users, selecting usernames, full names, password (or have the user set the password) and then choose group permissions.

2.8 Testing

From the mousedb directory run **bin\test** or **bin\django test** to run the test suite. See https://github.com/davebridges/mousedb/wiki/Known-Issues—Test-Suite for known issues. Report any additional errors at the issue page at https://github.com/davebridges/mousedb/issues.

CHAPTER

THREE

USERS AND RESTRICTION

All pages in this database are restricted to logged-in users. Users are defined using the standard Django `User` objects. It is also recommended that data is secured by only allowing access of specific IP addresses. For more details on this see the documentation for your webserver software (for example, for Apache see here http://httpd.apache.org/docs/2.2/howto/access.html). Each database should have at least one superuser, and that user can create and designate permissions for other users. When a user does not have the permissions to view a page or to edit something, the link to that page will not be visible and if they enter the address, they will be redirected to a login page.

3.1 Creating New Users

Create users by going to **../mousedb/admin/auth/user/add/** and filling in both pages of the form. Permissions are set by going to **../mousedb/admin/auth/user/** selecting the user and manually moving the permissions from the box on the left to the box on the right. If you want the user to have access to the admin site, select staff. Only select superuser if you want that user to have all permissions. Users can change passwords on the administration site as well.

3.2 Removing Users

Remove inactive users by selecting their username from the **../mousedb/admin/auth/user/** page and deselecting the active box. Only delete a user if it was generated mistakenly and that the particular user had not been used to edit any data.

CHAPTER

FOUR

ANIMAL DATA ENTRY

4.1 Newborn Mice or Newly Weaned Mice

1. Go to Breeding Cages Tab
2. Click on Add/Wean Pups Button
3. Each row is a new animal. If you accidentaly enter an extra animal, check off the delete box then submit.
4. Leave extra lines blank if you have less than 10 mice to enter
5. If you need to enter more than 10 mice, enter the first ten and submit them. Go back and enter up to 10 more animals (10 more blank spaces will appear)

4.2 Newborn Mice

1. Enter Breeding Cage under Cage
2. Enter Strain
3. Enter Background (normally Mixed or C57BL/6-BA unless from the LY breeding cages in which case it is C57BL/6-LY5.2)
4. Enter Birthdate in format YYYY-MM-DD
5. Enter Generation and Backcross

4.3 Weaning Mice

1. If not previously entered, enter data as if newborn mice
2. Enter gender
3. Enter Wean Date in format YYYY-MM-DD
4. Enter new Cage number for Cage

4.4 Cage Changes (Not Weaning)

1. Find mouse either from animal list or strain list

2. Click the edit mouse button
3. Change the Cage, Rack and Rack Position as Necessary

4.5 Genotyping or Ear Tagging

1. Find mouse either from animal list or strain list, or through breeding cage
2. Click the edit mouse button or the Eartag/Genotype/Cage Change/Death Button
3. Enter the Ear Tag and/or select the Genotype from the Pull Down List

4.6 Marking Mice as Dead

4.6.1 Dead Mice (Single Mouse)

1. Find mouse from animal list or strain list
2. Click the edit mouse button
3. Enter the death date in format YYYY-MM-DD
4. Choose Cause of Death from Pull Down List

4.6.2 Dead Mice (Several Mice)

1. Find mice from breeding cages
2. Click the Eartag/Genotype/Cage Change/Death Button
3. Enter the death date in format YYYY-MM-DD
4. Choose the Cause of Death from Pull Down List

CHAPTER

FIVE

STUDIES AND EXPERIMENTAL SETUP

Set up a new study at /mousedb/admin/data/study/ selecting animals

You must put a description and select animals in one or more treatment groups

If you have more than 2 treatment groups save the first two, then two more empty slots will appear. For animals, click on the magnifying glass then find the animal in that treatment group and click on the MouseID. The number displayed now in that field will not be the MouseID, but don't worry its just a different number to describe the mouse. To add more animals, click on the magnifying glass again and select the next animal. There should be now two numbers, separated by commas in this field. Repeat to fill all your treatment groups. You must enter a diet and environment for each treatment. The other fields are optional, and should only be used if appropriate. Ensure for pharmaceutical, you include a saline treatment group.

CHAPTER

SIX

MEASUREMENT ENTRY

6.1 Studies

If this measurement is part of a study (ie a group of experiments) then click on the plus sign beside the study field and enter in the details about the study and treatment groups. Unfortunately until i can figure out how to filter the treatment group animals in the admin interface, at each of the subsequent steps you will see all the animals in the database (soon hopefully it will only be the ones as part of the study group).

6.2 Experiment Details

- Pick experiment date, feeding state and resarchers
- Pick animals used in this experiment (the search box will filter results)
- Fasting state, time, injections, concentration, experimentID and notes are all optional

6.3 Measurements

- There is room to enter 14 measurements. If you need more rows, enter the first 14 and select "Save and Continue Editing" and 14 more blank spots will appear.
- Each row is a measurement, so if you have glucose and weight for some animal that is two rows entered.
- For animals, click on the magnifying glass then find the animal in that treatment group and click on the MouseID. The number displayed now in that field will not be the MouseID, but don't worry its just a different number to describe the mouse.
- For values, the standard units (defined by each assay) are mg for weights, mg/dL for glucose and pg/mL for insulin). You must enter integers here (no decimal places). If you have several measurements (ie several glucose readings during a GTT, enter them all in one measurement row, separated by commas and *NO spaces*).

CHAPTER

SEVEN

AUTOMATED DOCUMENTATION

7.1 Animals Application

The animal app contains and controls the display of data about animals.

Animals are tracked as individual entities, and given associations to breeding cages to follow ancestry, and strains.

7.1.1 Animal

Most parameters about an animal are set within the animal object. Here is where the animals strain, breeding, parentage and many other parameters are included. Animals have foreignkey relationships with both Strain and Breeding, so an animal may only belong to one of each of those. As an example, a mouse cannot come from more than one Breeding set, and cannot belong to more than one strain.

Backcrosses and Generations

For this software, optional tracking of backcrosses and generations is available and is stored as an attribute of an animal. When an inbred cross is made against a pure background, the backcross increases by 1. When a heterozygote cross is made, the generation increases by one. As an example, for every time a mouse in a C57/BL6 background is crossed against a wildtype C57/B6 mouse, the backcross (but not the generation) increases by one. For every time a mutant strain is crosses against itself (either vs a heterozygote or homozygote of that strain), the generation will increase by one. Backcrosses should typically be performed against a separate colony of purebred mouse, rather than against wild-type alleles of the mutant strain.

7.1.2 Breeding Cages

A breeding cage is defined as a set of one or more male and one or more female mice. Because of this, it is not always clear who the precise parentage of an animal is. If the parentage is known, then the Mother and Father fields can be set for a particular animal.

7.1.3 Strains

A strain is a set of mice with a similar genetics. Importantly strains are separated from Backgrounds. For example, one might have mice with the genotype ob/ob but these mice may be in either a C57-Black6 or a mixed background. This difference is set at the individual animal level. The result of this is that a query for a particular strain may then need to be filtered to a specific background.

7.1.4 Animal Data Models

This module describes the Strain, Animal, Breeding and Cage data models.

This module stores all data regarding a particular laboratory animal. Information about experimental data and timed matings are stored in the data and timed_matings packages. This module describes the database structure for each data model.

class `mousedb.animal.models.`**`Animal`**(**args*, ***kwargs*)

A data model describing an animal.

This data model describes a wide variety of parameters of an experimental animal. This model is linked to the Strain. If the parentage of a mouse is known, this can be identified (the breeding set may not be clear on this matter). Mice are automatically marked as not alive when a Death date is provided and the object is saved. Strain, Background and Genotype are required fields. By default, querysets are ordered first by strain then by MouseID.

`age`()

Calculates the animals age, relative to the current date (if alive) or the date of death (if not).

`breeding_female_location_type`()

This attribute defines whether a female's current location is the same as the breeding cage to which it belongs.

This attribute is used to color breeding table entries such that male mice which are currently in a different cage can quickly be identified. The location is relative to the first breeding cage an animal is assigned to.

`breeding_male_location_type`()

This attribute defines whether a male's current location is the same as the breeding cage to which it belongs.

This attribute is used to color breeding table entries such that male mice which are currently in a different cage can quickly be identified. The location is relative to the first breeding cage an animal is assigned to.

`save`()

The save method for Animal class is over-ridden to set Alive=False when a Death date is entered. This is not the case for a cause of death.

class `mousedb.animal.models.`**`Breeding`**(**args*, ***kwargs*)

This data model stores information about a particular breeding set

A breeding set may contain one ore more males and females and must be defined via the progeny strain. For example, in the case of generating a new strain, the strain indicates the new strain not the parental strains. A breeding cage is defined as one male with one or more females. If the breeding set is part of a timed mating experiment, then Timed_Mating must be selected. Breeding cages are automatically inactivated upon saving when a End date is provided. The only required field is Strain. By default, querysets are ordered by Strain, then Start.

`duration`()

Calculates the breeding cage's duration.

This is relative to the current date (if alive) or the date of inactivation (if not). The duration is formatted in days.

`male_breeding_location_type`()

This attribute defines whether a breeding male's current location is the same as the breeding cage.

This attribute is used to color breeding table entries such that male mice which are currently in a different cage can quickly be identified.

`save`()

The save function for a breeding cage has to automatic over-rides, Active and the Cage for the Breeder.

In the case of Active, if an End field is specified, then the Active field is set to False. In the case of Cage, if a Cage is provided, and animals are specified under Male or Females for a Breeding object, then the Cage field for those animals is set to that of the breeding cage. The same is true for both Rack and Rack Position.

unweaned()

This attribute generates a queryset of unweaned animals for this breeding cage. It is filtered for only Alive animals.

class `mousedb.animal.models.`**Strain**(**args*, ***kwargs*)

A data model describing a mouse strain.

This is separate from the background of a mouse. For example a ob/ob mouse on a mixed or a black-6 background still have the same strain. The background is defined in the animal and breeding cages. Strain and Strain_slug are required.

get_absolute_url(**moreargs*, ***morekwargs*)

For a Strain object, the permalinked absolute url is */strain/strain-slug*.

7.1.5 Animal App Views and URLs

This package contains all url dispatchers for the animal app.

It is broken down into animal, strain, breeding, cage and date url dispatchers for clarity. Each of these takes a different subfolder directive (ie the animal module is for */animal...* requests.

Breeding URLS

This URLconf defines the routing of pages for breeding objects.

This includes generic views for breeding, breeding details and create, change, search and delete breeding cages.

Cage URLS

Date URLS

This package is the url dispatcher for date views.

This urlconf takes a url in the form **/date/...** and can generate a general archive, yearly archive or monthly archive.

Mouse URLS

This url dispatcher for animal objects

controls views in the form */mouse...*, */mice...* or */animal...*

This includes create, update, delete views as well as animal-list, and animal-list all and animal-multiple-new.

Strain URLS

This module is the url dispatcher for strain related views.

It takes the root */strain...* and generates strain-list, strain-new, strain-edit, strain-delete, strain-detail and strain-detail-all views from animal.views.

ToDo List URLS

URL configuration file for todo subpages.

This controls any page **/mousedb/todo/...** and sends it to the appropriate views.

Views

These views define template redirects for the animal app.

This module contains all views for this app as class based views.

class `mousedb.animal.views.`**`AnimalCreate`**`(`***kwargs*`)`

This class generates the new `Animal` view (animal-new).

This permission restricted view takes a url in the form */animal/new* and generates an empty animal_form.html. This view is restricted to those with the animal.create_animal permission.

`dispatch``(`**args*, ***kwargs*`)`

This decorator sets this view to have restricted permissions.

`form_class`

alias of `AnimalForm`

`model`

alias of `Animal`

class `mousedb.animal.views.`**`AnimalDelete`**`(`***kwargs*`)`

This class generates the delete `Animal` view (animal-delete).

This permission restricted view takes a url in the form */animal/#/delete* and passes that object to the confirm_delete.html page. This view is restricted to those with the animal.delete_animal permission.

`dispatch``(`**args*, ***kwargs*`)`

This decorator sets this view to have restricted permissions.

`model`

alias of `Animal`

class `mousedb.animal.views.`**`AnimalDetail`**`(`***kwargs*`)`

This view displays specific details about an `Animal` object as animal-detail.

It takes a request in the form *animal/(id)*, *mice/(id)* or *mouse/(id)* and renders the detail page for that mouse. The request is defined for id not MouseID (or barcode) because this allows for details to be displayed for mice without barcode identification. Therefore care must be taken that animal/4932 is id=4932 and not barcode=4932. The animal name is defined at the top of the page. This page is restricted to logged-in users.

`model`

alias of `Animal`

class `mousedb.animal.views.`**`AnimalList`**`(`***kwargs*`)`

This view generates a list of `Animal` objects as animal-list

This view responds to a url in the form */animal* It sends a variable animal containing all animals to animal_list.html. This view is login protected.

`dispatch``(`**args*, ***kwargs*`)`

This decorator sets this view to have restricted permissions.

`model`

alias of `Animal`

class `mousedb.animal.views.`**`AnimalListAlive`**(***kwargs*)

This view generates a list of alive `Animal` objects.

This view subclasses `AnimalList` The main use for this view is to take a url in the form */animal/all* and to return a list of all alive animals to animal_list.html in the context animal. It also adds an extra context variable, "list type" as Alive. This view is login protected.

`get_context_data`(***kwargs*)

This add in the context of list_type and returns this as Alive.

class `mousedb.animal.views.`**`AnimalMonthArchive`**(***kwargs*)

This view generates a list of animals born within the specified year.

It takes a url in the form of **/date/####** where #### is the four digit code of the year. This view is restricted to logged in users.

`dispatch`(**args*, ***kwargs*)

This decorator sets this view to have restricted permissions.

`model`

alias of `Animal`

class `mousedb.animal.views.`**`AnimalUpdate`**(***kwargs*)

This class generates the update `Animal` view (animal-edit).

This permission restricted view takes a url in the form */animal/#/edit* and generates a animal_form.html with that object. This view is restricted to those with the animal.update_animal permission.

`dispatch`(**args*, ***kwargs*)

This decorator sets this view to have restricted permissions.

`form_class`

alias of `AnimalForm`

`model`

alias of `Animal`

class `mousedb.animal.views.`**`AnimalYearArchive`**(***kwargs*)

This view generates a list of animals born within the specified year.

It takes a url in the form of **/date/####** where #### is the four digit code of the year. This view is restricted to logged in users.

`dispatch`(**args*, ***kwargs*)

This decorator sets this view to have restricted permissions.

`model`

alias of `Animal`

class `mousedb.animal.views.`**`BreedingCreate`**(***kwargs*)

This class generates the breeding-new view.

This permission restricted view takes a url in the form */breeding/new* and generates an empty plugevents_form.html.

`dispatch`(**args*, ***kwargs*)

This decorator sets this view to have restricted permissions.

`form_class`

alias of `BreedingForm`

`model`

alias of `Breeding`

class `mousedb.animal.views.`**`BreedingDelete`**`(`***kwargs*`)`

This class generates the breeding-delete view.

This permission restricted view takes a url in the form */breeding/#/delete* and passes that object to the confirm_delete.html page.

`dispatch``(`**args*, ***kwargs*`)`

This decorator sets this view to have restricted permissions.

`model`

alias of `Breeding`

class `mousedb.animal.views.`**`BreedingDetail`**`(`***kwargs*`)`

This view displays specific details about a `Breeding` object.

It takes a request in the form */breeding/(breeding_id)* and renders the detail page for that breeding set. The breeding_id refers to the background id of the `Breeding` object, and not the breeding cage barcode. This page is restricted to logged-in users.

`model`

alias of `Breeding`

class `mousedb.animal.views.`**`BreedingList`**`(`***kwargs*`)`

This class generates an object list for active `Breeding` objects.

This login protected view takes all `Breeding` objects and sends them to strain_list.html as a strain_list dictionary. It also passes a strain_list_alive and cages dictionary to show the numbers for total cages and total strains. The url for this view is */strain/*

`get_context_data``(`***kwargs*`)`

This adds into the context of breeding_type and sets it to Active.

class `mousedb.animal.views.`**`BreedingListAll`**`(`***kwargs*`)`

This class generates a view for all `Breeding` objects.

This class is a subclass of `BreedingList`, changing the queryset and the breeding_type context.

`get_context_data``(`***kwargs*`)`

This add in the context of breeding_type and sets it to All.

class `mousedb.animal.views.`**`BreedingListTimedMating`**`(`***kwargs*`)`

This class generates a view for `Breeding` objects, showing only timed mating cages.

This class is a subclass of `BreedingList`, changing the queryset and the breeding_type context.

`get_context_data``(`***kwargs*`)`

This add in the context of breeding_type and sets it to Timed_Matings.

class `mousedb.animal.views.`**`BreedingSearch`**`(`***kwargs*`)`

This class generates a view for breeding objects, showing the results of a search query for cage number.

This class is a subclass of `BreedingList`, changing the queryset and the breeding_type context as well as providing the search query and search results if available.

`get_context_data``(`***kwargs*`)`

This add in the context of breeding_type and sets it to Search it also returns the query and the queryset.

class `mousedb.animal.views.`**`BreedingUpdate`**`(`***kwargs*`)`

This class generates the breeding-edit view.

This permission restricted view takes a url in the form */breeding/#/edit* and generates a breeding_form.html with that object.

dispatch(**args*, ***kwargs*)
This decorator sets this view to have restricted permissions.

form_class
alias of `BreedingForm`

model
alias of `Breeding`

class `mousedb.animal.views.`**CrossTypeAnimalList**(***kwargs*)
This view filters animal objects for a particular strain showing only the results of a particular breeding type.

This view takes a url in the form **/strain/<strain_slug>/<breeding_type>** and filters `Animal` objects on that basis, via both of those keyword arguments. Notably breeding_type has to be the display value not the key value of CROSS_TYPE. This view is a subclass of `AnimalList`.

get_context_data(***kwargs*)
This add in the context of list_type and returns this as whatever the crosstype was.

queryset()
This function sets the queryset according to the keyword arguments. For the crosstype, the input value is the the display value of CROSS_TYPE. This is done because the spaces in HET vs HET are not recognized. Therefore the queryset must be matched exactly (ie by case so Intercross not intercross). The function also filters the strain by the strain_slug keyword argument.

class `mousedb.animal.views.`**EarTagList**(***kwargs*)
This view is for showing animals which need to be eartagged.

This view is a subclass of `AnimalList`. This list shows animals that do not have an eartag (MouseID) and are older than the age set by WEAN_AGE in localsettings.py (default is 14 days). It takes a view **/todo/eartag**. This view is login protected.

class `mousedb.animal.views.`**GenotypeList**(***kwargs*)
This view is for showing animals which need to be genotyped.

This view is a subclass of `AnimalList`. This list shows animals that do not have a genotype (ie N.D. or ?) and are older than GENOTYPE_AGE as designated in localsettings.py (default is 21 days). It takes a view **/todo/genotype**. This view is login protected.

class `mousedb.animal.views.`**NoCageList**(***kwargs*)
This view is for showing animals which need to have a cage entered.

This list shows animals that have no cage number and are alive. This view is a subclass of `AnimalList` It takes a view **/todo/no_cage**. This view is login protected.

class `mousedb.animal.views.`**NoRackList**(***kwargs*)
This view is for showing animals which need to have a cage entered.

This list shows animals that have no cage number and are alive. This view is a subclass of `AnimalList` It takes a view **/todo/no_rack**. This view is login protected.

class `mousedb.animal.views.`**StrainCreate**(***kwargs*)
This class generates the new `Strain` view (strain-new).

This permission restricted view takes a url in the form */strain/new* and generates an empty strain_form.html. This view is restricted to those with the animal.create_strain permission.

dispatch(**args*, ***kwargs*)
This decorator sets this view to have restricted permissions.

model
alias of `Strain`

class `mousedb.animal.views.`**`StrainDelete`**(***kwargs*)

This class generates the delete `Strain` view (strain-delete).

This permission restricted view takes a url in the form */strain/#/delete* and passes that object to the confirm_delete.html page. This view is restricted to those with the animal.delete_strain permission.

`dispatch`(**args*, ***kwargs*)

This decorator sets this view to have restricted permissions.

`model`

alias of `Strain`

class `mousedb.animal.views.`**`StrainDetail`**(***kwargs*)

This view displays specific details about a `Strain` object showing *only current* related objects.

It takes a request in the form *strain/(strain_slug)/* and renders the detail page for that `Strain`. This view also passes along a dictionary of alive animals belonging to that strain. This page is restricted to logged-in users.

`get_context_data`(***kwargs*)

This add in the context of strain_list_alive (which filters for only alive animals and active) and cages which filters for the number of current cages.

class `mousedb.animal.views.`**`StrainDetailAll`**(***kwargs*)

This view displays specific details about a `Strain` showing *all* related objects.

This view subclasses `StrainDetail` and modifies the get_context_data to show associated active objects. It takes a request in the form *strain/(strain_slug)/all* and renders the detail page for that `Strain`. This view also passes along a dictionary of alive animals belonging to that `Strain`. This page is restricted to logged-in users.

`get_context_data`(***kwargs*)

This adds into the context of strain_list_all (which filters for all alive `Animal` objects and active cages) and cages which filters for the number of current cages.

class `mousedb.animal.views.`**`StrainList`**(***kwargs*)

This class generates an object list for `Strain` objects.

This login protected view takes all `Strain` objects and sends them to strain_list.html as a strain_list dictionary. It also passes a strain_list_alive and cages dictionary to show the numbers for total cages and total strains. The url for this view is **/strain/**

`get_context_data`(***kwargs*)

This add in the context of strain_list_alive (which filters for all alive animals) and cages which filters for the number of current cages.

`model`

alias of `Strain`

class `mousedb.animal.views.`**`StrainUpdate`**(***kwargs*)

This class generates the update `Strain` view (strain-edit).

This permission restricted view takes a url in the form */strain/#/edit* and generates a strain_form.html with that object. This view is restricted to those with the animal.update_strain permission.

`dispatch`(**args*, ***kwargs*)

This decorator sets this view to have restricted permissions.

`model`

alias of `Strain`

class `mousedb.animal.views.`**`WeanList`**(***kwargs*)

This view is for showing animals which need to be weaned.

This list shows animals that need to be weaned. This view is a subclass of `AnimalList` filtering for animals that are older than the WEAN_AGE and are alive. It takes a view **/todo/wean**. This view is login protected.

`mousedb.animal.views.breeding_change(request, *args, **kwargs)`

This view is used to generate a form by which to change pups which belong to a particular breeding set.

This view typically is used to modify existing pups. This might include marking animals as sacrificed, entering genotype or marking information or entering movement of mice to another cage. It is used to show and modify several animals at once. It takes a request in the form /breeding/(breeding_id)/change/ and returns a form specific to the breeding set defined in breeding_id. breeding_id is the background identification number of the breeding set and does not refer to the barcode of any breeding cage. This view returns a formset in which one row represents one animal. To add extra animals to a breeding set use /breeding/(breeding_id)/pups/. This view is restricted to those with the permission animal.change_animal.

`mousedb.animal.views.breeding_pups(request, *args, **kwargs)`

This view is used to generate a form by which to add pups which belong to a particular breeding set.

This view is intended to be used to add initial information about pups, including eartag, genotype, gender and birth or weaning information. It takes a request in the form /breeding/(breeding_id)/pups/ and returns a form specific to the breeding set defined in breeding_id. breeding_id is the background identification number of the breeding set and does not refer to the barcode of any breeding cage. This view is restricted to those with the permission animal.add_animal.

`mousedb.animal.views.breeding_wean(request, *args, **kwargs)`

This view is used to generate a form by which to wean pups which belong to a particular breeding set.

This view typically is used to wean existing pups. This includes the MouseID, Cage, Markings, Gender and Wean Date fields. For other fields use the breeding-change page. It takes a request in the form /breeding/(breeding_id)/wean/ and returns a form specific to the breeding set defined in breeding_id. breeding_id is the background identification number of the breeding set and does not refer to the barcode of any breeding cage. This view returns a formset in which one row represents one animal. To add extra animals to a breeding set use /breeding/(breeding_id)/pups/. This view is restricted to those with the permission animal.change_animal.

`mousedb.animal.views.date_archive_year(request)`

This view will generate a table of the number of mice born on an annual basis.

This view is associated with the url name archive-home, and returns an dictionary of a date and a animal count.

`mousedb.animal.views.multiple_breeding_pups(request, breeding_id)`

This view is used to enter multiple animals at the same time from a breeding cage.

It will generate a form containing animal information and a number of mice. It is intended to create several identical anin

This view requres an input of a breeding_id to generate the correct form.

`mousedb.animal.views.multiple_pups(request)`

This view is used to enter multiple animals at the same time.

It will generate a form containing animal information and a number of mice. It is intended to create several identical animals with the same attributes.

`mousedb.animal.views.todo(request, *args, **kwargs)`

This view generates a summary of the todo lists.

The login restricted view passes lists for ear tagging, genotyping and weaning and passes them to the template todo.html.

Administrative Interface

Admin site settings for the animal app.

class `mousedb.animal.admin.`**`AnimalAdmin`**(*model*, *admin_site*)

Provides parameters for animal objects within the admin interface.

`mark_estimated_death`(*request*, *queryset*)

An admin action for marking several animals as sacrificed.

This action sets the selected animals as Alive=False, Death=today and Cause_of_Death as Estimated. To use other paramters, mice muse be individually marked as sacrificed. This admin action also shows as the output the number of mice sacrificed.

`mark_sacrificed`(*request*, *queryset*)

An admin action for marking several animals as sacrificed.

This action sets the selected animals as Alive=False, Death=today and Cause_of_Death as sacrificed. To use other paramters, mice muse be individually marked as sacrificed. This admin action also shows as the output the number of mice sacrificed.

class `mousedb.animal.admin.`**`AnimalInline`**(*parent_model*, *admin_site*)

Provides an inline tabular formset for animal objects.

Currently used with the breeding admin page.

`model`

alias of `Animal`

class `mousedb.animal.admin.`**`BreedingAdmin`**(*model*, *admin_site*)

Settings in the admin interface for dealing with Breeding objects.

This interface also includes an form for adding objects associated with this breeding cage.

`mark_deactivated`(*request*, *queryset*)

An admin action for marking several cages as inactive.

This action sets the selected cages as Active=False and Death=today. This admin action also shows as the output the number of mice sacrificed.

class `mousedb.animal.admin.`**`StrainAdmin`**(*model*, *admin_site*)

Settings in the admin interface for dealing with Strain objects.

7.1.6 Animal App Unit Tests

This file contains tests for the animal application.

These tests will verify generation and function of a new breeding, strain and animal object.

class `mousedb.animal.tests.`**`AnimalModelTests`**(*methodName='runTest'*)

Tests the model attributes of Animal objects contained in the animal app.

`setUp`()

Instantiate the test client.

`tearDown`()

Depopulate created model instances from test database.

`test_animal_unicode`()

This is a test for creating a new animal object, with only the minimum fields being entered. It then tests that the correct unicode representation is being generated.

`test_create_animal_minimal`()

This is a test for creating a new animal object, with only the minimum fields being entered

class `mousedb.animal.tests.`**`AnimalViewTests`**(*methodName='runTest'*)

Tests the views associated with animal objects.

setUp()
Instantiate the test client. Creates a test user.

tearDown()
Depopulate created model instances from test database.

test_animal_delete()
This test checks the view which displays an animal deletion page.

It checks for the correct templates and status code and that the animal is passed correctly to the context.

test_animal_detail()
This tests the animal-detail view, ensuring that templates are loaded correctly.

This view uses a user with superuser permissions so does not test the permission levels for this view.

test_animal_edit()
This test checks the view which displays a animal edit page.

It checks for the correct templates and status code and that the animal is passed correctly to the context.

test_animal_list()
This test checks the view which displays a breeding list page showing active animals. It checks for the correct templates and status code.

test_animal_list_all()
This test checks the view which displays a breeding list page showing all animals. It checks for the correct templates and status code.

test_animal_new()
This test checks the view which displays a new animal.

It checks for the correct templates and status code.

class mousedb.animal.tests.**BreedingModelTests**(*methodName='runTest'*)
Tests the model attributes of Breeding objects contained in the animal app.

setUp()
Instantiate the test client.

tearDown()
Depopulate created model instances from test database.

test_autoset_active_state()
This is a test for creating a new breeding object, with only the minimum being entered. That object is then tested for the active state being automatically set when a End date is specified.

test_create_breeding_minimal()
This is a test for creating a new breeding object, with only the minimum being entered.

test_duration()
This test verifies that the duration is set correctly.

test_study_absolute_url()
This test verifies that the absolute url of a breeding object is set correctly.

test_unweaned()
This is a test for the unweaned animal list. It creates several animals for a breeding object and tests that they are tagged as unweaned. They are then weaned and retested to be tagged as not unweaned. This test is incomplete.

class mousedb.animal.tests.**BreedingViewTests**(*methodName='runTest'*)
These are tests for views based on Breeding objects. Included are tests for breeding list (active and all), details, create, update and delete pages as well as for the timed mating lists.

test_breeding_delete()
: This test checks the view which displays a breeding detail page. It checks for the correct templates and status code.

test_breeding_detail()
: This test checks the view which displays a breeding detail page. It checks for the correct templates and status code.

test_breeding_edit()
: This test checks the view which displays a breeding edit page. It checks for the correct templates and status code.

test_breeding_list()
: This test checks the view which displays a breeding list page showing active breeding cages. It checks for the correct templates and status code.

test_breeding_list_all()
: This test checks the view which displays a breeding list page, for all the cages. It checks for the correct templates and status code.

test_breeding_new()
: This test checks the view which displays a new breeding page. It checks for the correct templates and status code.

test_timed_mating_list()
: This test checks the view which displays a breeding list page, for all the cages. It checks for the correct templates and status code.

class mousedb.animal.tests.**CageViewTests**(*methodName='runTest'*)
: These are tests for views based on animal objects as directed by cage urls. Included are tests for cage-list, cage-list-all and cage-detail

test_cage_detail()
: This test checks the view which displays a animal list page showing all animals with a specified cage number. It checks for the correct templates and status code.

test_cage_list()
: This test checks the view which displays a cage list page showing all animals. It checks for the correct templates and status code.

class mousedb.animal.tests.**DateViewTests**(*methodName='runTest'*)
: These are tests for views based on animal objects as directed by date based urls. Included are tests for archive-home, archive-month and archive-year

test_archive_home()
: This test checks the view which displays a summary of the birthdates of animals. It checks for the correct templates and status code.

test_archive_month()
: This test checks the view which displays a list of the animals, filtered by month. It checks for the correct templates and status code.

test_archive_year()
: This test checks the view which displays a list of the animals, filtered by year. It checks for the correct templates and status code.

class mousedb.animal.tests.**StrainModelTests**(*methodName='runTest'*)
: Tests the model attributes of Strain objects contained in the animal app.

setUp()
: Instantiate the test client. Creates a test user.

tearDown()
Depopulate created model instances from test database.

test_create_strain_all()
This is a test for creating a new strain object, with only all fields being entered

test_create_strain_minimal()
This is a test for creating a new strain object, with only the minimum fields being entered

test_strain_absolute_url()
This is a test for creating a new strain object, then testing absolute url.

test_strain_unicode()
This is a test for creating a new strain object, then testing the unicode representation of the strain.

class mousedb.animal.tests.**StrainViewTests**(*methodName='runTest'*)
Test the views contained in the animal app relating to Strain objects.

setUp()
Instantiate the test client. Creates a test user.

tearDown()
Depopulate created model instances from test database.

test_strain_crosstype()
This tests the strain-crosstype view, ensuring that templates are loaded correctly.

This view uses a user with superuser permissions so does not test the permission levels for this view.

test_strain_delete()
This tests the strain-delete view, ensuring that templates are loaded correctly.

This view uses a user with superuser permissions so does not test the permission levels for this view.

test_strain_detail()
This tests the strain-detail view, ensuring that templates are loaded correctly.

This view uses a user with superuser permissions so does not test the permission levels for this view.

test_strain_detail_all()
This tests the strain-detail-all view, ensuring that templates are loaded correctly.

This view uses a user with superuser permissions so does not test the permission levels for this view.

test_strain_edit()
This tests the strain-edit view, ensuring that templates are loaded correctly.

This view uses a user with superuser permissions so does not test the permission levels for this view.

test_strain_list()
This tests the strain-list view, ensuring that templates are loaded correctly.

This view uses a user with superuser permissions so does not test the permission levels for this view.

test_strain_new()
This tests the strain-new view, ensuring that templates are loaded correctly.

This view uses a user with superuser permissions so does not test the permission levels for this view.

class mousedb.animal.tests.**ToDoViewTests**(*methodName='runTest'*)
Tests the views associated with animal objects for the three todo lists.

setUp()
Instantiate the test client. Creates a test user.

tearDown()
Depopulate created model instances from test database.

test_eartag_list()
This test checks the view which displays an animal list page showing animals which need to be eartagged. It checks for the correct templates and status code.

test_genotype_list()
This test checks the view which displays an animal list page showing animals which need to be genotyped. It checks for the correct templates and status code.

test_no_cage_list()
This test checks the view which displays an animal list page showing animals which need to have a cage entered. It checks for the correct templates and status code.

test_no_rack_list()
This test checks the view which displays an animal list page showing animals which need to have a rack entered. It checks for the correct templates and status code.

test_todo_home()
This test checks the view which displays a summary of the todo lists. It checks for the correct templates and status code.

test_wean_list()
This test checks the view which displays an animal list page showing animals which need to be weaned. It checks for the correct templates and status code.

7.1.7 Animal App Utilities

Forms for use in manipulating objects in the animal app.

class mousedb.animal.forms.**AnimalForm**(*data=None, files=None, auto_id='id_%s', prefix=None, initial=None, error_class=<class 'django.forms.util.ErrorList'>, label_suffix=':', empty_permitted=False, instance=None*)
This modelform provides fields for modifying animal data.

This form also automatically loads javascript and css for the datepicker jquery-ui widget. It also includes auto

class mousedb.animal.forms.**BreedingForm**(*data=None, files=None, auto_id='id_%s', prefix=None, initial=None, error_class=<class 'django.forms.util.ErrorList'>, label_suffix=':', empty_permitted=False, instance=None*)
This form provides most fields for creating and entring breeding cage data.

This form is used from the url /mousedb/breeding/new and is a generic create view. This view includes a datepicker widget for Stat and End dates and autocomplete fields for the Females and Male fields

class mousedb.animal.forms.**MultipleAnimalForm**(*data=None, files=None, auto_id='id_%s', prefix=None, initial=None, error_class=<class 'django.forms.util.ErrorList'>, label_suffix=':', empty_permitted=False, instance=None*)
This modelform provides fields for entering multiple identical copies of a set of mice.

This form only includes the required fields Background and Strain.

class `mousedb.animal.forms.`**`MultipleBreedingAnimalForm`**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.util.ErrorList'>*, *label_suffix=':'*, *empty_permitted=False*, *instance=None*)

This modelform provides fields for entering multiple pups within a breeding set.

The only fields presented are Born, Weaned, Gender and Count. Several other fields will be automatically entered based on the Breeding Set entries.

This is a configuration file for the ajax lookups for the animal app.

See http://code.google.com/p/django-ajax-selects/ for information about configuring the ajax lookups.

class `mousedb.animal.lookups.`**`AnimalLookup`**

This is the generic lookup for animals.

It is to be used for all animal requests and directs to the 'animal' channel.

`format_item`(*animal*)

the display of a currently selected object in the area below the search box. html is OK

`format_result`(*animal*)

This controls the display of the dropdown menu.

This is set to show the unicode name of the animal, as well as its Cage and its Markings.

`get_objects`(*ids*)

given a list of ids, return the objects ordered as you would like them on the admin page. this is for displaying the currently selected items (in the case of a ManyToMany field)

`get_query`(*q*, *request*)

This sets up the query for the lookup.

The lookup searches the MouseID or the id (database identifier) for the mouse.

class `mousedb.animal.lookups.`**`AnimalLookupFemale`**

This is the generic lookup for animals.

It is to be used for all animal requests and directs to the 'animal-female' channel.

`format_item`(*animal*)

the display of a currently selected object in the area below the search box. html is OK

`format_result`(*animal*)

This controls the display of the dropdown menu.

This is set to show the unicode name of the animal, as well as its Cage and its Markings.

`get_objects`(*ids*)

given a list of ids, return the objects ordered as you would like them on the admin page. this is for displaying the currently selected items (in the case of a ManyToMany field)

`get_query`(*q*, *request*)

This sets up the query for the lookup.

The lookup searches the MouseID or the id (database identifier) for the mouse. It then filters this set for only females.

class `mousedb.animal.lookups.`**`AnimalLookupMale`**

This is the generic lookup for animals.

It is to be used for all animal requests and directs to the 'animal-male' channel.

format_item (*animal*)
: the display of a currently selected object in the area below the search box. html is OK

format_result (*animal*)
: This controls the display of the dropdown menu.

  This is set to show the unicode name of the animal, as well as its Cage and its Markings.

get_objects (*ids*)
: given a list of ids, return the objects ordered as you would like them on the admin page. this is for displaying the currently selected items (in the case of a ManyToMany field)

get_query (*q*, *request*)
: This sets up the query for the lookup.

  The lookup searches the MouseID or the id (database identifier) for the mouse. It then filters this set for only males.

7.2 Data Application

The data module describes the conditions and collection of data regarding experimental animals.

Data (or `Measurement`) can be stored for any type of `Experiment`. Conceptually, several pieces of data belong to an experiment (for example several `Animal` are measured at some time) and several `Experiment` belong to a `Study`.

Measurements can be stored independent of experiments and experiments can be performed outside of the context of a study. It is however, perfered that measurements are stored within an experiment and experiments are stored within studies as this will greatly facilitate the organization of the data.

7.2.1 Studies

In general studies are a collection of experiments. This is a large class which can contain several `Experiment` objects. Within a study two classes control how data is stored, `Treatment` and `Cohort`.

7.2.2 Treatments

A `Treatment` is a group of `Animal` that are grouped together based on some manipulation, which could be a `Pharmaceutical`, `Diet`, `Implantation` or some other manipulation. Generally a `Study` will have two or more `Treatment` groups.

7.2.3 Cohorts

For the same `Treatment` groups, within a `Study`, there may be several independent replicates of those treatment. A replicate are called `Cohort`. For example, there could be a `Treatment` in which some mice get a particular `Diet`, while other mice get a diffferent `Diet`. These two `Treatment` groups are designated within a cohort. Later on, using different `Animal` sets, this `Study` will be repeated. While the `Treatment` groupings are the same, the different series' of `Experiment` will be separated by being part of a different `Cohort`.

7.2.4 Experiments

An experiment is a collection of measurements for a given set of animals. In general, an experiment is defined as a number of measurements take in a given day.

7.2.5 Measurements

A measurement is an animal, an assay and a measurement value. It can be associated with an experiment, or can stand alone as an individual value. Measurements can be viewed in the context of a study, an experiment, a treatment group or an animal by going to the appropriate page.

7.2.6 Data API Access

This package controls API access to the `data` app.

Overview

The API for the `data` application provides data on measurements and their associated data types. A list of all endpoints and links to their schemas are available at **http://yourserver.org/api/v1/** where yourserver is specific to your installation. There are four access points, each of which is available using GET requests only:

- measurements are available at the endpoint **http://yourserver.org/api/v1/data/**
- assays are available at the endpoint **http://yourserver.org/api/v1/assay/**
- experiments are available at the endpoint **http://yourserver.org/api/v1/experiment/**
- studies are available at the endpoint **http://yourserver.org/api/v1/study/**

The data can be provided as either a group of objects or as a single object. Currently for all requests, no authentication is required.

The entire API schema is available from each endpoint at:

```
/schema/?format=xml
/schema/?format=json
```

For example **http://yourserver.org/api/v1/study/schema/?format=json**.

Sample Code

Either group requests or single object requests can be served depending on if the primary key is provided. The request URI has several parts including the servername, the api version (currently v1) then the item type (data, assay, experiment or study). There must be a trailing slash before the request parameters (which are after a **?** sign and separated by a **&** sign).

For a collection of objects

For a collection of measurements you can request:

```
http://yourserver.org/api/v1/data/?format=json
```

This would return all measurements in the database. This would return the response with two JSON objects, **meta** and **objects**. The **format=json** parameter is default for curl and not required but is necessary for browser requests. The meta object contains fields for the **limit**, **next**, **offset**, **previous** and **total_count** for the series of objects requested. The objects portion is an array of the returned object fields (see the details below for each API). Note the id field of an object. This is used for retrieving a single object. Collections can also be filtered based on several request parameters (see below for details for each API):

```
http://yourserver.org/api/v1/data/ #returns all data in JSON format
http://yourserver.org/api/v1/experiment/set/1;3/?format=json #returns the experiments with id numbers
```

For a single object

To retrieve a single item you need to know the primary key of the object. This can be found from the id parameter of a collection (see above) or from the actual object page. You can retrieve details about a single assay with a call such as:

```
http://yourserver.org/api/v1/assay/2/?format=json
```

In this case **2** is the primary key (or id field) of the assay in question.

Reference for Measurement API

The measurement API is available at the endpoint **/api/v1/data/**

Request Parameters

The following are the potential request variables, all of which are optional. The default format is JSON, but this can be set as XML if required. If viewing by web browser ?format=json must be specified By default 20 items are returned but you can increase this to all by setting limit=0.

Parameter	Potential Values
format	**json** or **xml**
limit	**0** for all, any other number

Response Values

The response (in either JSON or XML) provides the following fields for each object (or for the only object in the case of a single object request).

Field	Explanation	Sample Value
id	the id of the measurement	1
resource_uri	the URI to request details about a measurement	/api/v1/data/1/
values	the measurement, or measurement(s)	423

Reference for the Assay API

The assay API is available at the endpoint **/api/v1/assay/**

Request Parameters

The following are the potential request variables, all of which are optional. The default format is JSON, but this can be set as XML if required. If viewing by web browser ?format=json must be specified By default 20 items are returned but you can increase this to all by setting limit=0.

Parameter	Potential Values
format	**json** or **xml**
limit	**0** for all, any other number

Response Values

The response (in either JSON or XML) provides the following fields for each object (or for the only object in the case of a single object request).

Field	Explanation	Sample Value
id	the id of the measurement	3
resource_uri	the URI to request details about a measurement	/api/v1/assay/3/
assay	the name of the assay	Body Weight
assay_slug	the slugified name of the assay (can be used for filtering)	body_weight
measurement_units	the units for the measurment	mg/dL
notes	notes regarding this assay	Some text values

Reference for the Experiment API

The experiment API is available at the endpoint **/api/v1/experiment/**

Request Parameters

The following are the potential request variables, all of which are optional. The default format is JSON, but this can be set as XML if required. If viewing by web browser ?format=json must be specified By default 20 items are returned but you can increase this to all by setting limit=0.

Parameter	Potential Values
format	**json** or **xml**
limit	**0** for all, any other number

Response Values

The response (in either JSON or XML) provides the following fields for each object (or for the only object in the case of a single object request).

Field	Explanation	Sample Value
id	the id of the measurement	5
resource_uri	the URI to request details about a measurement	/api/v1/experiment/5/
concentration	the concentration of the injection (if done)	1mU/kg
date	the date of the experiment	2012-09-28
experimentID	the optional experimentID number	DB-2012-09-28
fasting_time	the duration of the animal fast (if done) in hours	16
feeding_state	whether the animals were fed or fasted	fed OR fasted
injection	the injection (if done)	Insulin, Glucose, Pyruvate or Glucagon
notes	notes regarding this experiment	Some text values
time	the time of day the assay was done (24h format)	16:00

Reference for the Study API

The study API is available at the endpoint **/api/v1/study/**

Request Parameters

The following are the potential request variables, all of which are optional. The default format is JSON but this can be set as XML if required. If viewing by web browser ?format=json must be specified By default 20 items are returned but you can increase this to all by setting limit=0.

Parameter	Potential Values
format	**json** or **xml**
limit	**0** for all, any other number

Response Values

The response (in either JSON or XML) provides the following fields for each object (or for the only object in the case of a single object request).

Field	Explanation	Sample Value
id	the id of the measurement	6
resource_uri	the URI to request details about a measurement	/api/v1/study/6/
description	the description of the study	some text
notes	some notes about the study	some text
start_date	the optional starting date of the study	2012-07-23
stop _date	the optional end date of the study	2012-09-27

class `mousedb.data.api.`**`AssayResource`**(*api_name=None*)

This generates the API resource for `Assay` objects.

It returns all assays in the database.

class **`Meta`**

The API serves all `Assay` objects in the database..

`object_class`

alias of `Assay`

class `mousedb.data.api.`**`ExperimentResource`**(*api_name=None*)

This generates the API resource for `Experiment` objects.

It returns all experiments in the database.

class Meta
The API serves all `Experiment` objects in the database..

object_class
alias of `Experiment`

class `mousedb.data.api.`**`MeasurementAssayResource`**(*api_name=None*)
This generates serves `Assay` objects.

This is a limited dataset for use in MeasurementResource calls.

class Meta
The API serves all `Assay` objects in the database..

class `mousedb.data.api.`**`MeasurementResource`**(*api_name=None*)
This generates the API resource for `Measurement` objects.

It returns all measurements in the database.

class Meta
The API serves all `Measurement` objects in the database..

object_class
alias of `Measurement`

class `mousedb.data.api.`**`StudyResource`**(*api_name=None*)
This generates the API resource for `Study` objects.

It returns all studies in the database.

class Meta
The API serves all `Study` objects in the database..

object_class
alias of `Study`

7.2.7 Data Models

class `mousedb.data.models.`**`Assay`**(**args*, ***kwargs*)
Assay(id, assay, assay_slug, notes, measurement_units)

class `mousedb.data.models.`**`Cohort`**(**args*, ***kwargs*)
A Cohort is a group of `Animals`.

Generally a cohort is an experimental replicate of a `Treatment` as part of a `Study`. Cohorts are also generally defined by starting and ending dates. A cohort would generally comprise both `Treatment` groups being compared. The required fields are **name** (which must be unique to this cohort) and **animals**.

`get_absolute_url`(**moreargs*, ***morekwargs*)
The url for a treatment-detail is **/cohort/<id>**.

`save`(**args*, ***kwargs*)
The slug field is auto-populated during the save from the name field.

class `mousedb.data.models.`**`Diet`**(**args*, ***kwargs*)
Diet(id, vendor_id, description, product_id, fat_content, protein_content, carb_content, irradiated, notes)

class `mousedb.data.models.`**`Environment`**(**args*, ***kwargs*)
Environment(id, building, room, temperature, humidity, notes)

class `mousedb.data.models.`**`Experiment`**(**args*, ***kwargs*)
This class describes Experiment objects.

This object describes the aspects of an experiment done on several `Animal` objects. This object describes the conditions under which the experiment ws done. The results of the experiment are contained in `Measurement` objects and grouped together in `Treatment` objects. Experiments may or may not be part of a `Study`. The required fields for this object are date, `Researchers` and feeding state. The optional fields are notes, time, experimentID, fasting_time, injection and concentration and the `Study`.

get_absolute_url(**moreargs*, ***morekwargs*)
The absolute url of an experiment is */mousedb/experiment/id </mousedb/experiment/id>*.

class `mousedb.data.models.`**Implantation**(**args*, ***kwargs*)
Implantation(id, implant, vendor_id, product_id, notes)

class `mousedb.data.models.`**Measurement**(**args*, ***kwargs*)
Measurement(id, animal_id, experiment_id, assay_id, values)

class `mousedb.data.models.`**Pharmaceutical**(**args*, ***kwargs*)
This class defines a drug treatment.

Each object is specific to a particular vendor, dose and mode of delivery. For other doses, generate additional Pharmaceutical objects. The required fields are **drug**, **dose**, **recurrance**, **mode** and **vendor**.

get_absolute_url(**moreargs*, ***morekwargs*)
The absolute url of a `Pharmaceutical` is **/parameter/pharmaceutical/<id>**.

class `mousedb.data.models.`**Researcher**(**args*, ***kwargs*)
Researcher(id, first_name, last_name, name_slug, email, active)

class `mousedb.data.models.`**Study**(**args*, ***kwargs*)
Study(id, description, start_date, stop_date, notes)

class `mousedb.data.models.`**Transplantation**(**args*, ***kwargs*)
Transplantation(id, tissue, transplant_date, notes)

class `mousedb.data.models.`**Treatment**(**args*, ***kwargs*)
This model defines the groupings of mice for an experiment.

The purpose of treatment groups is to associate together animals which are treated similarly in a study. A treatment group is generally defined by its specific conditions, including:

- `Diet`
- `Environment`
- `Implantation` (optional)
- `Transplantation` (optional)
- `Pharmaceutical` (optional)

The required fields are the associated `Study`, the name (treatment), the `Animal` objects in this group and the `Researcher` objects in the group. There is also an optional notes field.

get_absolute_url(**moreargs*, ***morekwargs*)
The url for a treatment-detail is **/treatment/<id>**.

class `mousedb.data.models.`**Vendor**(**args*, ***kwargs*)
Vendor(id, vendor, website, email, ordering, notes)

7.2.8 Data Views and URLs

Cohort URLS

Url redirections for `Cohort` objects.

This includes generic create, update, delete, list and detail views. The views for these urls are defined in the module.

Experiment URLS

Parameter URLS

Study URLS

This URLconf defines the routing of pages for study objects.

This includes generic views for study list, study details and create, change and delete studies.

Treatment URLS

Url redirections for treatment objects.

This includes gneeric create, update, delete, list and detail views. These are restricted by login required (for detail and list) and appropriate permissions for forms.

Views

class `mousedb.data.views.`**`CohortCreate`**`(`***kwargs*`)`

This view generates a form for creating a `Cohort` object.

This view is restricted to logged in users with the create-cohort permission. It is generated when the url **/cohort/new** is requested.

`model`

alias of `Cohort`

class `mousedb.data.views.`**`CohortDelete`**`(`***kwargs*`)`

This view generates a view for deleting `Cohort` objects.

This view is restricted to logged in users with the delete-cohort permision. It passes an object **cohort** when the url **/cohort/<slug>/delete** is requested.

`model`

alias of `Cohort`

class `mousedb.data.views.`**`CohortDetail`**`(`***kwargs*`)`

This view generates details about a `Cohort` object.

This view is restricted to logged in users. It passes an object **cohort** when the url **/cohort/<slug>** is requested.

`model`

alias of `Cohort`

class `mousedb.data.views.`**`CohortList`**`(`***kwargs*`)`

This view generates list of `Cohort` objects.

This view is restricted to logged in users. It passes an object **cohort_list** when the url **/cohort** is requested.

model
alias of Cohort

class mousedb.data.views.**CohortUpdate**(***kwargs*)
This view generates a form for updating Cohort objects.

This view is restricted to logged in users with the update-cohort permision. It passes an object **object** when the url **/cohort/<slug>/edit** is requested.

model
alias of Cohort

class mousedb.data.views.**PharmaceuticalCreate**(***kwargs*)
This view generates a form for creating a Pharmaceutical object.

This view is restricted to logged in users with the create-pharmaceutical permission. It is generated when the url **/parameter/pharmaceutical/new** is requested.

model
alias of Pharmaceutical

class mousedb.data.views.**PharmaceuticalDelete**(***kwargs*)
This view generates a view for deleting Pharmaceutical objects.

This view is restricted to logged in users with the delete-pharmaceutical permision. It passes an object **pharmaceutical** when the url **/parameter/pharmaceutical/<id>/delete** is requested.

model
alias of Pharmaceutical

class mousedb.data.views.**PharmaceuticalDetail**(***kwargs*)
This view generates details about a Pharmaceutical object.

This view is restricted to logged in users. It passes an object **pharmaceutical** when the url **/parameter/pharmaceutical/<id>** is requested.

model
alias of Pharmaceutical

class mousedb.data.views.**PharmaceuticalList**(***kwargs*)
This view generates list of Pharmaceutical objects.

This view is restricted to logged in users. It passes an object **pharmaceutical** when the url **/parameter/pharmaceutical** is requested.

model
alias of Pharmaceutical

class mousedb.data.views.**PharmaceuticalUpdate**(***kwargs*)
This view generates a form for updating Pharmaceutical objects.

This view is restricted to logged in users with the update-pharmaceutical permision. It passes an object **pharmaceutical** when the url **/parameter/pharmaceutical/<id>/edit** is requested.

model
alias of Pharmaceutical

class mousedb.data.views.**TreatmentDetail**(***kwargs*)
This view generates details about a Treatment object.

This view is restricted to logged in users. It passes an object **treatment** when the url **/treatment/<pk#>** is requested.

model
alias of `Treatment`

class `mousedb.data.views.`**`TreatmentList`**(***kwargs*)
This view generatea list of a `Treatment` objects.

This view is restricted to logged in users. It passes an object **treatment_list** when the url **/treatment** is requested.

model
alias of `Treatment`

`mousedb.data.views.`**`add_measurement`**(*request*, **args*, ***kwargs*)
This is a view to display a form to add single measurements to an experiment.

It calls the object MeasurementForm, which has an autocomplete field for animal.

`mousedb.data.views.`**`aging_csv`**(*request*)
This view generates a csv output file of all animal data for use in aging analysis.

The view writes to a csv table the animal, strain, genotype, age (in days), and cause of death.

`mousedb.data.views.`**`all_data_csv`**(*request*)
This view generates a csv output of all data for a strain.

`mousedb.data.views.`**`experiment_details_csv`**(*request*, *experiment_id*)
This view generates a csv output file of an experiment.

The view writes to a csv table the animal, genotype, age (in days), assay and values.

`mousedb.data.views.`**`litters_csv`**(*request*)
This view generates a csv output file of all animal data for use in litter analysis.

The view writes to a csv table the birthdate, breeding cage and strain.

Administrative Interface

7.2.9 Data Unit Tests

This file contains tests for the data application.

These tests will verify generation of new experiment, measurement, assay, researcher, study, treatment, vendor, diet, environment, implantation, transplantation and pharnaceutical objects.

class `mousedb.data.tests.`**`BasicTestCase`**(*methodName='runTest'*)
This class factors out the TestcCase setup and Teardown code.

setUp()
Instantiate the test client.

tearDown()
Depopulate created model instances from test database.

class `mousedb.data.tests.`**`CohortModelTests`**(*methodName='runTest'*)
These tests test the functionality of `Cohort` objects.

test_cohort_absolute_url()
This tests the absolute_url generation of a `Cohort`.

test_cohort_unicode()
This tests the unicode representation of a `Cohort`.

test_create_cohort_all()
This test creates a `Cohort` with all information entered.

test_create_cohort_minimum()
This test creates a `Cohort` with the required information only.

class `mousedb.data.tests.`**CohortViewTests**(*methodName='runTest'*)
This class tests the views for `Cohort` objects.

test_cohort_list()
This tests the cohort-list view, ensuring that templates are loaded correctly.

This view uses a user with superuser permissions so does not test the permission levels for this view.

test_cohort_view()
This tests the cohort-view view, ensuring that templates are loaded correctly.

This view uses a user with superuser permissions so does not test the permission levels for this view.

test_cohort_view_create()
This tests the cohort-new view, ensuring that templates are loaded correctly.

This view uses a user with superuser permissions so does not test the permission levels for this view.

test_cohort_view_delete()
This tests the cohort-delete view, ensuring that templates are loaded correctly.

This view uses a user with superuser permissions so does not test the permission levels for this view.

test_cohort_view_edit()
This tests the cohort-edit view, ensuring that templates are loaded correctly.

This view uses a user with superuser permissions so does not test the permission levels for this view.

class `mousedb.data.tests.`**PharmaceuticalModelTests**(*methodName='runTest'*)
These tests test the functionality of `Pharmaceutical` objects.

test_create_pharmaceutical_all()
This test creates a `Pharmaceutical` with all information entered.

test_create_pharmaceutical_minimum()
This test creates a `Pharmaceutical` with the required information only.

test_pharmaceutical_absolute_url()
This tests the absolute_url generation of a `Pharmaceutical`.

test_pharmaceutical_unicode()
This tests the unicode representation of a `Pharmaceutical`.

class `mousedb.data.tests.`**PharmaceuticalViewTests**(*methodName='runTest'*)
This class tests the views for `Pharmaceutical` objects.

test_pharmaceutical_list()
This tests the pharmaceutical-list view, ensuring that templates are loaded correctly.

This view uses a user with superuser permissions so does not test the permission levels for this view.

test_pharmaceutical_view()
This tests the pharmaceutical-view view, ensuring that templates are loaded correctly.

This view uses a user with superuser permissions so does not test the permission levels for this view.

test_pharmaceutical_view_create()
This tests the pharmaceutical-new view, ensuring that templates are loaded correctly.

This view uses a user with superuser permissions so does not test the permission levels for this view.

test_pharmaceutical_view_delete()
This tests the pharmaceutical-delete view, ensuring that templates are loaded correctly.

This view uses a user with superuser permissions so does not test the permission levels for this view.

test_pharmaceutical_view_edit()
This tests the pharmaceutical-edit view, ensuring that templates are loaded correctly.

This view uses a user with superuser permissions so does not test the permission levels for this view.

class mousedb.data.tests.**StudyModelTests**(*methodName='runTest'*)
Test the creation and modification of Study objects.

test_create_studey_detailed()
This is a test for creating a new study object, with all fields being entered. It also verifies that unicode is set correctly. This test is dependent on the ability to create a new Strain object (see animal.tests.StrainModelTests.test_create_minimal_strain).

test_create_study_minimal()
This is a test for creating a new study object, with only the minimum being entered. It also verifies that unicode is set correctly.

test_study_absolute_url()
This test verifies that the absolute url of a study object is set correctly. This study is dependend on a positive result on test_create_study_minimal.

class mousedb.data.tests.**StudyViewTests**(*methodName='runTest'*)
These tests test the views associated with Study objects.

test_study_delete()
This test checks the view which displays a study detail page. It checks for the correct templates and status code.

test_study_detail()
This test checks the view which displays a study detail page. It checks for the correct templates and status code.

test_study_edit()
This test checks the view which displays a study edit page. It checks for the correct templates and status code.

test_study_list()
This test checks the status code, and templates for study lists.

test_study_new()
This test checks the view which displays a study creation page. It checks for the correct templates and status code.

class mousedb.data.tests.**TreatmentModelTests**(*methodName='runTest'*)
These tests test the functionality of Treatment objects.

test_create_treatment_all()
This test creates a Treatment with all information entered.

test_create_treatment_minimum()
This test creates a Treatment with the required information only.

test_treatment_absolute_url()
This tests the absolute_url generation of a Treatment.

test_treatment_unicode()
This tests the unicode representation of a Treatment.

class `mousedb.data.tests.`**`TreatmentViewTests`**(*methodName='runTest'*)

These tests test the views associated with Treatment objects.

`test_treatment_detail`()

This test checks the view which displays a treatment-detail page.

It checks for the correct templates and status code.

`test_treatment_list`()

This tests the treatment-list view, ensuring that templates are loaded correctly.

This view uses a user with superuser permissions so does not test the permission levels for this view.

7.2.10 Data Utilities

class `mousedb.data.forms.`**`ExperimentForm`**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.util.ErrorList'>*, *label_suffix=':'*, *empty_permitted=False*, *instance=None*)

This is the configuration for the experiment form.

This form is used to set up and modify an experiment. It uses a datepicker widget for the date.

class `mousedb.data.forms.`**`MeasurementForm`**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.util.ErrorList'>*, *label_suffix=':'*, *empty_permitted=False*, *instance=None*)

Form definition for adding and editing measurements.

This form is used for adding or modifying single measurements from within an experiment. It has an autocomplete field for animal.

class `mousedb.data.forms.`**`StudyExperimentForm`**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.util.ErrorList'>*, *label_suffix=':'*, *empty_permitted=False*, *instance=None*)

This is the configuration for a study form (From an experiment).

This hides the study field which will be automatically set upon save.

class `mousedb.data.forms.`**`StudyForm`**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.util.ErrorList'>*, *label_suffix=':'*, *empty_permitted=False*, *instance=None*)

This is the configuration for the study form.

This form is used to create and modify studies. It uses an autocomplete widget for the animals.

class `mousedb.data.forms.`**`TreatmentForm`**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.util.ErrorList'>*, *label_suffix=':'*, *empty_permitted=False*, *instance=None*)

Form class for study treatment groups.

In the case of studies, animals are defined in the treatment group rather than in the study group. A treatment consists of a study, a set of animals and the conditions which define that treatment. This includes related fields for environment, diet, implants and transplants.

7.3 Groups Application

This package defines the Group application. This app defines generic Group and License information for a particular installation of MouseDB. Because every page on this site identifies both the Group and data restrictions, at a minimum, group information must be provided upon installation (see installation instructions).

7.3.1 Groups Data Models

class `mousedb.groups.models.`**`Group`**`(`**args*, ***kwargs*`)`
This defines the data structure for the Group model.

The only required field is group. All other fields (group_slug, group_url, license, contact_title, contact_first, contact_last and contact_email) are optional.

class `mousedb.groups.models.`**`License`**`(`**args*, ***kwargs*`)`
This defines the data structure for the License model.

The only required field is license. If the contents of this installation are being made available using some licencing criteria this can either be defined in the notes field, or in an external website.

7.3.2 Groups App Views and URLs

Views

Administrative Interface

class `mousedb.groups.admin.`**`GroupAdmin`**`(`*model*, *admin_site*`)`
Defines the admin interface for Groups.

Currently set as default.

class `mousedb.groups.admin.`**`LicenseAdmin`**`(`*model*, *admin_site*`)`
Defines the admin interface for Licences.

Currently set as default.

class `mousedb.groups.admin.`**`LogEntryAdmin`**`(`*model*, *admin_site*`)`
Defines the admin interface for the LogEntry objects.

7.3.3 Groups App Unit Tests

This file contains tests for the groups application.

These tests will verify generation of a new group and license object.

class `mousedb.groups.tests.`**`GroupsModelTests`**`(`*methodName='runTest'*`)`
Test the models contained in the 'groups' app.

`setUp``()`
Instantiate the test client.

`tearDown``()`
Depopulate created model instances from test database.

`test_create_group_all_fields``()`
This is a test for creating a new group object, with all fields being entered, except license.

test_create_group_minimal()
This is a test for creating a new group object, with only the minimum being entered.

test_create_license_all_fields()
This is a test for creating a new license object, with all fields being entered.

test_create_license_minimal()
This is a test for creating a new license object, with only the minimum being entered.

7.4 Timed Mating Application

This package defines the timed_mating app.

Timed matings are a specific type of breeding set. Generally, for these experiments a mating cage is set up and pregnancy is defined by a plug event. Based on this information, the age of an embryo can be estimated. When a breeding cage is defined, one option is to set this cage as a timed mating cage (ie Timed_Mating=True). If this is the case, then a plug event can be registered and recorded for this mating set. If the mother gives birth then this cage is implicitly set as a normal breeding cage.

7.4.1 Timed Mating Data Models

This defines the data model for the timed_mating app.

Currently the only data model is for PlugEvents.

class `mousedb.timed_mating.models.`**PlugEvents**(**args*, ***kwargs*)
This defines the model for PlugEvents.

A PlugEvent requires a date. All other fields are optional. Upon observation of a plug event, the PlugDate, Breeding Cage, Femalem, Male, Researcher and Notes can be set. Upon sacrifice of the mother, then genotyped alive and dead embryos can be entered, along with the SacrificeDate, Researcher and Notes.

get_absolute_url(**moreargs*, ***morekwargs*)
The permalink for a plugevent is /mousedb/timed_mating/plugs/**id**.

save()
Over-rides the default save function for PlugEvents.

If a sacrifice date is set for an object in this model, then Active is set to False.

7.4.2 Timed Mating App Views and URLs

This urlconf sets the directions for the timed_mating app.

It takes a url in the form of /plug/something and sends it to the appropriate view class or function.

Views

This package defines custom views for the timed_mating application.

Currently all views are generic CRUD views except for the view in which a plug event is defined from a breeding cage.

class `mousedb.timed_mating.views.`**`PlugEventsCreate`**`(`*`**kwargs`*`)`

This class generates the plugevents-new view.

This permission restricted view takes a url in the form **/plugs/new** and generates an empty plugevents_form.html.

`model`

alias of `PlugEvents`

class `mousedb.timed_mating.views.`**`PlugEventsDelete`**`(`*`**kwargs`*`)`

This class generates the plugevents-delete view.

This permission restricted view takes a url in the form **/plugs/#/delete** and passes that object to the confirm_delete.html page.

`model`

alias of `PlugEvents`

class `mousedb.timed_mating.views.`**`PlugEventsDetail`**`(`*`**kwargs`*`)`

This class generates the plugevents-detail view.

This login protected takes a url in the form **/plugs/1** for plug event id=1 and passes a **plug** object to plugevents_detail.html

`model`

alias of `PlugEvents`

class `mousedb.timed_mating.views.`**`PlugEventsList`**`(`*`**kwargs`*`)`

This class generates an object list for PlugEvent objects.

This login protected view takes all PlugEvents objects and sends them to plugevents_list.html as a plug_list dictionary. The url for this view is **/plugs/**

`model`

alias of `PlugEvents`

class `mousedb.timed_mating.views.`**`PlugEventsListStrain`**`(`*`**kwargs`*`)`

This class generates a strain filtered list for Plug Event objects.

This is a subclass of PlugEventsList and returns as context_object_name plug_events_list to plugevents_list.html. It takes a named argument (strain) which is a Strain_slug and filters based on that strain.

`get_queryset``()`

The queryset is over-ridden to show only plug events in which the strain matches the breeding strain.

class `mousedb.timed_mating.views.`**`PlugEventsUpdate`**`(`*`**kwargs`*`)`

This class generates the plugevents-edit view.

This permission restricted view takes a url in the form **/plugs/#/edit** and generates a plugevents_form.html with that object.

`model`

alias of `PlugEvents`

`mousedb.timed_mating.views.`**`breeding_plugevent`**`(`*`request`*, *`*args`*, *`**kwargs`*`)`

This view defines a form for adding new plug events from a breeding cage.

This form requires a breeding_id from a breeding set and restricts the PlugFemale and PlugMale to animals that are defined in that breeding cage.

Administrative Interface

Settings to control the admin interface for the timed_mating app.

This file defines a PlugEventsAdmin object to enter parameters about individual plug events/

class `mousedb.timed_mating.admin.`**`PlugEventsAdmin`**(*model*, *admin_site*)
    This class defines the admin interface for the PlugEvents model.

7.4.3 Timed Mating App Unit Tests

This file contains tests for the timed_mating application.

These tests will verify generation of a new PlugEvent object.

class `mousedb.timed_mating.tests.`**`Timed_MatingModelTests`**(*methodName='runTest'*)
    Test the models contained in the 'timed_mating' app.

`setUp`()
    Instantiate the test client. Creates a test user.

`tearDown`()
    Depopulate created model instances from test database.

`test_create_plugevent_minimal`()
    This is a test for creating a new PlugEvent object, with only the minimum being entered.

`test_create_plugevent_most_fields`()
    This is a test for creating a new PlugEvent object.

    This test uses a Breeding, PlugDate, PlugMale and PlugFemale field.

`test_set_plugevent_inactive`()
    This is a test for the automatic inactivation of a cage when the SacrificeDate is entered.

class `mousedb.timed_mating.tests.`**`Timed_MatingViewTests`**(*methodName='runTest'*)
    Test the views contained in the 'timed_mating' app.

`setUp`()
    Instantiate the test client. Creates a test user.

`tearDown`()
    Depopulate created model instances from test database.

`test_breeding_plugevent_new`()
    This tests the breeding-plugevent-new view, ensuring that templates are loaded correctly.

    This view uses a user with superuser permissions so does not test the permission levels for this view.

`test_plugevent_delete`()
    This tests the plugevent-delete view, ensuring that templates are loaded correctly.

    This view uses a user with superuser permissions so does not test the permission levels for this view.

`test_plugevent_detail`()
    This tests the plugevent-detail view, ensuring that templates are loaded correctly.

    This view uses a user with superuser permissions so does not test the permission levels for this view.

`test_plugevent_edit`()
    This tests the plugevent-edit view, ensuring that templates are loaded correctly.

    This view uses a user with superuser permissions so does not test the permission levels for this view.

test_plugevent_list()
This tests the plugevent-list view, ensuring that templates are loaded correctly.

This view uses a user with superuser permissions so does not test the permission levels for this view.

test_plugevent_list_strain()
This tests the plugevent-list-strain view, ensuring that templates are loaded correctly.

This view uses a user with superuser permissions so does not test the permission levels for this view.

test_plugevent_new()
This tests the plugevent-new view, ensuring that templates are loaded correctly.

This view uses a user with superuser permissions so does not test the permission levels for this view.

7.4.4 Timed Mating Utilities

This package describes forms used by the Timed Mating app.

class mousedb.timed_mating.forms.**BreedingPlugForm**(*data=None, files=None, auto_id='id_%s', prefix=None, initial=None, error_class=<class 'django.forms.util.ErrorList'>, label_suffix=':', empty_permitted=False, instance=None*)
This form is used to enter Plug Events from a specific breeding cage.

CHAPTER

EIGHT

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

INDEX