feincms3-data Documentation

Release 0.3

Feinheit AG

Sep 19, 2022

CONTENTS

CHAPTER

ONE

WHY

Utilities for loading and dumping database data as JSON.

These utilities (partially) replace Django's built-in `dumpdata` and `loaddata` management commands.

Suppose you want to move data between systems incrementally. In this case it isn't sufficient to only know the data which has been created or updated; you also want to know which data has been deleted in the meantime. Django's `dumpdata` and `loaddata` management commands only support the former case, not the latter. They also do not including dependent objects in the dump.

This package offers utilities and management commands to address these shortcomings.

CHAPTER

TWO

HOW

`pip install feincms3-data`.

Add `feincms3_data` to `INSTALLED_APPS` so that the included management commands are discovered.

Add datasets somewhere describing the models and relationships you want to dump, e.g. in a module named `app.f3datasets`:

```
from feincms3_data.data import (
    specs_for_app_models,
    specs_for_derived_models,
    specs_for_models,
)
from app.dashboard import models as dashboard_models
from app.world import models as world_models


def districts(args):
    pks = [int(arg) for arg in args.split(",") if arg]
    return [
        *specs_for_models(
            [world_models.District],
            {
                "filter": {"pk__in": pks},
                "delete_missing": True,
            },
        ),
        *specs_for_models(
            [world_models.Exercise],
            {
                "filter": {"district__in": pks},
                "delete_missing": True,
            },
        ),
        # All derived non-abstract models which aren't proxies:
        *specs_for_derived_models(
            world_models.ExercisePlugin,
            {
                "filter": {"parent__district__in": pks},
                "delete_missing": True,
            },
        ),
```

(continues on next page)

(continued from previous page)

```
    ]


def datasets():
    return {
        "articles": {
            "specs": lambda args: specs_for_app_models(
                "articles",
                {"delete_missing": True},
            ),
        },
        "pages": {
            "specs": lambda args: specs_for_app_models(
                "pages",
                {"delete_missing": True},
            ),
        },
        "teachingmaterials": {
            "specs": lambda args: specs_for_models(
                [
                    dashboard_models.TeachingMaterialGroup,
                    dashboard_models.TeachingMaterial,
                ],
                {"delete_missing": True},
            ),
        },
        "districts": {
            "specs": districts,
        },
    }
```

Add a setting with the Python module path to the specs function:

```
FEINCMS3_DATA_DATASETS = "app.f3datasets.datasets"
```

Now, to dump e.g. pages you would run:

```
./manage.py f3dumpdata pages > tmp/pages.json
```

To dump the districts with the primary key of 42 and 43 you would run:

```
./manage.py f3dumpdata districts:42,43 > tmp/districts.json
```

The resulting JSON file has three top-level keys:

- `"version": 1`: The version of the dump, because not versioning dumps is a recipe for pain down the road.
- `"specs": [...]`: A list of model specs.
- `"objects": [...]`: A list of model instances; uses the same serializer as Django's `dumpdata`, everything looks the same.

Model specs consist of the following fields:

- `"model"`: The lowercased label (`app_label.model_name`) of a model.

- `"filter"`: A dictionary which can be passed to the `.filter()` queryset method as keyword arguments; used for determining the objects to dump and the objects to remove after loading.
- `"delete_missing"`: This flag makes the loader delete all objects matching `"filter"` which do not exist in the dump.
- `"ignore_missing_m2m"`: A list of field names where deletions of related models should be ignored when restoring. This may be especially useful when only transferring content partially between databases.
- `"save_as_new"`: If present and truish, objects are inserted using new primary keys into the database instead of (potentially) overwriting pre-existing objects.

The dumps can be loaded back into the database by running:

```
./manage.py f3loaddata -v2 tmp/pages.json tmp/districts.json
```

Each dump is processed in an individual transaction. The data is first loaded into the database; at the end, data *matching* the filters but whose primary key wasn't contained in the dump is deleted from the database (if `"delete_missing": True`).

CHAPTER

THREE

CHANGE LOG

- Fixed a crash when using nullable foreign keys to a model which uses `save_as_new`.
- Fix a behavior when using `save_as_new` together with `delete_missing`: When the parent of a model with both flags set is updated, the content is now duplicated (because the old object hadn't been removed and the new one has been saved according to `save_as_new`). Previously, the old object has been removed automatically. Worse, this made is impossible to use `save_as_new` for duplicating top-level objects (their object graph would be removed after inserting the copy). This change may still be backwards incompatible for you though, so better check twice.
- Fixed deletion of missing objects in the presence of protected objects by processing specs in reverse.
- Changed `specs_for_derived_models` to skip proxy models instead of skipping unmanaged models.
- Started deferring constraints and resetting sequences when loading dumps.
- Added Django 4.1.
- Added the `mappers` argument to `dump_specs` which allows changing the serialized representation of models before writing the JSON.
- Changed `load_dump` to also validate specs.
- Replaced `FEINCMS3_DATA_SPECS` with the more flexible and opinionated `FEINCMS3_DATA_DATASETS`.
- Initial release!