invenio-upgrader Documentation

Release 0.1.0

CERN

August 19, 2015

1 API **3**
1.1 Recommendations for writing upgrades . . . 3
1.2 Upgrade modules . . . 3
1.3 Upgrade dependency graph . . . 4
1.4 Models . . . 4

2 Contents: **5**

3 Indices and tables **7**

Python Module Index **9**

Upgrader engine for Invenio modules.

This is an experimental development preview release.

- Free software: GPLv2 license
- Documentation: https://invenio-upgrader.readthedocs.org.

CHAPTER 1

API

Usage (via `inveniomanage`).

```
$ inveniomanage upgrader create recipe -p invenio_search
$ inveniomanage upgrader create release -r invenio -p invenio.base
$ inveniomanage upgrader show applied
$ inveniomanage upgrader show pending
$ inveniomanage upgrader check
$ inveniomanage upgrader run
```

1.1 Recommendations for writing upgrades

- An upgrade must be self-contained. **DO NOT IMPORT ANYTHING** from Invenio unless absolutely necessary. Reasons: 1) If a it depends on other Invenio modules, then their API must be very stable and backwards-compatible. Otherwise when an upgrade is applied two years later, the Invenio function might have evolved and the upgrade will fail.
- Once an upgrade have been committed, no fiddling is allowed afterwards (i.e. no semantic changes). If you want to correct a mistake, make an new upgrade instead.
- All upgrades must depend on a previous upgrade (except for your first upgrade).
- For every software release, make a `<repository>_release_<x>_<y>_<z>.py` that depends on all upgrades between the previous release and the new, so future upgrades can depend on this upgrade. The command `inveniomanage upgrader create release` can help you with this.
- Upgrades may query for user input, but must be able to run in unattended mode when `--yes-i-know option` is being used, thus good defaults/guessing should be used.

1.2 Upgrade modules

Upgrades are implemented as normal Python modules. They must implement the methods `do_upgrade()` and `info()` and contain a list variable `depends_on`. Optionally they may implement the methods `estimate()`, `pre_upgrade()`, `post_upgrade()`.

The upgrade engine will search for upgrades in all packages listed in `current_app.config['PACKAGES']`.

1.3 Upgrade dependency graph

The upgrades form a *dependency graph* that must be without cycles (i.e. a DAG). The upgrade engine supports having several independent graphs (normally one per module). The graphs are defined using via a filename prefix using the pattern (`<module name>_<date>_<name>.py`).

The upgrade engine will run upgrades in topological order (i.e upgrades will be run respecting the dependency graph). The engine will detect cycles in the graph and will refuse to run any upgrades until the cycles have been broken.

`invenio_upgrader.`**`populate_existing_upgrades`**(*sender*, *yes_i_know=False*, *drop=True*, ***kwargs*)

Populate existing upgrades.

1.4 Models

Upgrade engine database model for keeping log of which upgrades have been applied to to a given database.

CHAPTER 2

Contents:

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

i

invenio_upgrader, 3
invenio_upgrader.models, 4

Index

I

invenio_upgrader (module), 3
invenio_upgrader.models (module), 4

P

populate_existing_upgrades() (in module invenio_upgrader), 4