OOpgrade Documentation

Release 0.6.1

GISCE-TI, S.L.

September 09, 2016

Contents

This project, is based on OpenUpgrade, and provides a set of functions to migrate schema and data from our openobject server.

Release v0.6.1

Note: The API is quite unstable due the active development

Contents:

CHAPTER 1

OOpgrade API

The OpenUpgrade library contains all kinds of helper functions for your pre and post scripts, in OpenUpgrade itself or in the migration scripts of your own module (in either major or minor version upgrades). It can be installed with

```
pip install oopgrade
```

and then used in your scripts as

```
from oopgrade import oopgrade
```

1.1 General methods

oopgrade.oopgrade.**load_data**(*cr*, *module_name*, *filename*, *idref=None*, *mode='init'*)

Load an xml or csv data file from your post script. The usual case for this is the occurrence of newly added essential or useful data in the module that is marked with "noupdate='1'" and without "forcecreate='1'" so that it will not be loaded by the usual upgrade mechanism. Leaving the 'mode' argument to its default 'init' will load the data from your migration script.

Theoretically, you could simply load a stock file from the module, but be careful not to reinitialize any data that could have been customized. Preferably, select only the newly added items. Copy these to a file in your migrations directory and load that file. Leave it to the user to actually delete existing resources that are marked with 'noupdate' (other named items will be deleted automatically).

Parameters

- **module_name** – the name of the module
- **filename** – the path to the filename, relative to the module directory.
- **idref** – optional hash with ?id mapping cache?
- **mode** – one of 'init', 'update', 'demo'. Always use 'init' for adding new items from files that are marked with 'noupdate'. Defaults to 'init'.

oopgrade.oopgrade.**rename_columns**(*cr*, *column_spec*)

Rename table columns. Typically called in the pre script.

Parameters **column_spec** – a hash with table keys, with lists of tuples as values. Tuples consist of (old_name, new_name).

oopgrade.oopgrade.**rename_tables**(*cr*, *table_spec*)

Rename tables. Typically called in the pre script. :param column_spec: a list of tuples (old table name, new table name).

oopgrade.oopgrade.**drop_columns**(*cr*, *column_spec*)
Drop columns but perform an additional check if a column exists. This covers the case of function fields that may or may not be stored. Consider that this may not be obvious: an additional module can govern a function fields' store properties.

Parameters **column_spec** – a list of (table, column) tuples

oopgrade.oopgrade.**table_exists**(*cr*, *table*)
Check whether a certain table or view exists

oopgrade.oopgrade.**column_exists**(*cr*, *table*, *column*)
Check whether a certain column exists

oopgrade.oopgrade.**change_column_type**(*cursor*, *column_spec*)

Parameters

- **cr** – Cursor
- **colum_spec** – a hash with table keys, with lists of tuples as values. Tuples consist of (column name, new_def). new_def as in posgresql_language

Returns execute result

oopgrade.oopgrade.**delete_model_workflow**(*cr*, *model*)
Forcefully remove active workflows for obsolete models, to prevent foreign key issues when the orm deletes the model.

oopgrade.oopgrade.**set_defaults**(*cr*, *pool*, *default_spec*, *force=False*)
Set default value. Useful for fields that are newly required. Uses orm, so call from the post script.

Parameters

- **default_spec** – a hash with model names as keys. Values are lists of tuples (field, value). None as a value has a special meaning: it assigns the default value. If this value is provided by a function, the function is called as the user that created the resource.
- **force** – overwrite existing values. To be used for assigning a non- default value (presumably in the case of a new column). The ORM assigns the default value as declared in the model in an earlier stage of the process. Beware of issues with resources loaded from new data that actually do require the model's default, in combination with the post script possible being run multiple times.

oopgrade.oopgrade.**set_stored_function**(*cr*, *obj*, *fields*)
Init newly created stored functions calling the function and storing them to the database.

..note:: Use in the post stage

Parameters

- **cr** – Database cursor
- **obj** – Object
- **fields** – list of fields

oopgrade.oopgrade.**update_module_names**(*cr*, *namespec*)
Deal with changed module names of certified modules in order to prevent 'certificate not unique' error, as well as updating the module reference in the XML id.

Parameters **namespec** – tuple of (old name, new name)

oopgrade.oopgrade.**add_ir_model_fields**(*cr*, *columnspec*)
Typically, new columns on ir_model_fields need to be added in a very early stage in the upgrade process of the base module, in raw sql as they need to be in place before any model gets initialized. Do not use for fields with

additional SQL constraints, such as a reference to another table or the cascade constraint, but craft your own statement taking them into account.

Parameters **columnspec** – tuple of (column name, column type)

oopgrade.oopgrade.**install_modules**(*cursor*, **modules*)
Installs a module.

Parameters

- **cr** – Cursor database
- **module** – The module to install

oopgrade.oopgrade.**get_foreign_keys**(*cursor*, *table*)
Get all the foreign keys from the given table

Returns a dict with column_name as a key and the following keys:

- constraint_name
- table_name
- column_name
- foreign_table_name
- foreign_column_name

Parameters

- **cursor** – Database cursor
- **table** – Table name to get the foreign keys

Returns dict

1.2 Version

class oopgrade.version.**Version**(*version*)
Version class using semantic version parsing

Parameters **version** – version string

bump_major()
Bump a major version: X from X.Y.Z

bump_minor()
Bump a minor version: Y from X.Y.Z

bump_patch()
Bump a patch version: Z from X.Y.Z

info
Get the information parsing the version.

Returns a dict with the keys: major, minor, patch, prerelease and build

1.3 Data migration

class oopgrade.data.**DataMigration**(*content*, *cursor*, *module*, *search_params=None*)

Data Migration class

Parameters

- **content** – XML Content to migrate
- **cursor** – Database cursor
- **module** – OpenObject module name
- **search_params** – Dict where key is the model and value is the list

of fields to do the match.

Example:

```
from oopgrade import DataMigration

dm = DataMigration(xml_content, cursor, 'module_name', search_params={
    'test.model': ['field1', 'field2']
})
dm.migrate()
```

In this case when a record for model *test.model* is found it will use the fields *field1* and *field2* to do the match it will construct a search query as:

```
[
    ('field1', '=', 'content_xml_record_field1'),
    ('field2', '=', 'content_xml_record_field2')
]
```

Note: If no search_params is passed **all** the fields from the xml will be used to create the search params

class oopgrade.data.**DataRecord**(*id*, *model*, *noupdate*, *vals*)

id
Alias for field number 0

model
Alias for field number 1

noupdate
Alias for field number 2

vals
Alias for field number 3

O