mothergeo Documentation

Release 0.0.1

Pat Daburu

Sep 11, 2017
## Contents

1 Development  
   1.1 Python Module Dependencies  
   1.2 Design Considerations  
   1.3 Source Control  
   1.4 How To...  
   1.5 Miscellany  

2 API Documentation  
   2.1 mothergeo  
   2.2 mothergeo.codetools  
   2.3 mothergeo.config  
   2.4 mothergeo.db.postgis.geoalchemy  
   2.5 mothergeo.db.modeling  
   2.6 mothergeo.db.postgis.modeling  
   2.7 mothergeo.i18n  
   2.8 mothergeo.schemas.modeling  
   2.9 mothergeo.schemas.parsing  
   2.10 mothergeo.geometry  

3 Indices and tables  

Python Module Index
Word to your mother.
mothergeo is under active development.

## Python Module Dependencies

The `requirements.txt` file contains this project’s module dependencies. You can install these dependencies using `pip`.

```
pip install -r requirements.txt
```

### requirements.txt

<table>
<thead>
<tr>
<th>Package</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>alabaster</td>
<td>0.7.10</td>
</tr>
<tr>
<td>appdirs</td>
<td>1.4.3</td>
</tr>
<tr>
<td>Babel</td>
<td>2.4.0</td>
</tr>
<tr>
<td>colorama</td>
<td>0.3.7</td>
</tr>
<tr>
<td>docutils</td>
<td>0.13.1</td>
</tr>
<tr>
<td>GeoAlchemy2</td>
<td>0.4.0</td>
</tr>
<tr>
<td>imagesize</td>
<td>0.7.1</td>
</tr>
<tr>
<td>injector</td>
<td>0.13.1</td>
</tr>
<tr>
<td>insensitive-dict</td>
<td>0.1.6</td>
</tr>
<tr>
<td>Jinja2</td>
<td>2.9.6</td>
</tr>
<tr>
<td>MarkupSafe</td>
<td>1.0</td>
</tr>
<tr>
<td>measurement</td>
<td>1.8.0</td>
</tr>
<tr>
<td>numpy</td>
<td>1.12.1</td>
</tr>
<tr>
<td>packaging</td>
<td>16.8</td>
</tr>
<tr>
<td>psycopg2</td>
<td>2.7.1</td>
</tr>
<tr>
<td>Pygments</td>
<td>2.2.0</td>
</tr>
<tr>
<td>pyparsing</td>
<td>2.2.0</td>
</tr>
<tr>
<td>pytz</td>
<td>2017.2</td>
</tr>
<tr>
<td>requests</td>
<td>2.13.0</td>
</tr>
<tr>
<td>six</td>
<td>1.10.0</td>
</tr>
</tbody>
</table>
Design Considerations

Coming soon.

Source Control

Coming soon.

How To...

How to Use an Alternate PyPI (Package Index)

The Python Package Index (PyPI) works great, but there may be times when, for whatever reason, you need to host packages elsewhere. You can set up your own index, or you can use a hosted service like Gemfury.

This article describes, in general terms, some of the things you’ll need to do in your development environment to make your use of an alternate package index a little smoother.

Installing Modules with pip

One of the tricks to installing packages from your alternate repository is telling pip about it.

pip.ini

While you can use command line parameters with pip to indicate the location of your package index server, you can also modify (or create) a special pip configuration file called pip.ini that will allow you install packages from the command line just as you would if you were installing them from the public repositories.

Windows

On Windows, you can place a pip.ini file at %APPDATA%\pip\pip.ini. Use the extra-index-url option to tell pip where your alternate package index lives. If your package index doesn’t support SSL, you can suppress warnings by identifying it as a trusted-host. The example below assumes the name of your server is pypi.mydomain.com and you’re running on non-standard port 8080.

```
[global]
extra-index-url = http://pypi.mydomain.com:8080
```
[install]
trusted-host = pypi.mydomain.com

Linux

Coming soon.

Note: If you are using SSL with a verified certificate, you won’t need the trusted-host directive.

Publishing Modules

This article doesn’t go into much detail on the general process of publishing modules, but we’ll assume that you’re using setuptools to publish.

.pypirc

You can automate the process of publishing your package with distutils by modifying the .pypirc file in your home directory. This file typically contains the common public indexes, but you can also add your alternate index. The example below assumes you’re using Gemfury, but the format will be fundamentally similar regardless of where you’re hosting your repository.

[distutils]
index-servers =
  pypi
  fury

[pypi]
username=mypypiuser
password=$ecret-Pa$$w0rd

[fury]
repository: https://pypi.fury.io/myfuryusername/
username: $ecret-K3y!
password:

With that in place, you can build and upload your package by identifying the configured index server name.

python setup.py sdist upload -r fury

Note: Remember that the keys and passwords in your .pypirc are secrets, and should be kept away from prying eyes.

How To Document Your Code

Coming Soon.
How To Set Up Your Development Environment

This article describes the steps you can follow to get the mothergeo-py project set up for development.

Get the Source

This project is managed under source control in GitHub, so you’ll need to install git. Once you have that going, getting the latest version of the code is just a matter of cloning the repository into your development directory.

```
  git clone https://github.com/patdaburu/mothergeo-py.git
```

Get the Requirements

The project uses a number of modules that are available from the PyPI package repository. All of the required modules should be listed in the `requirements.txt` file in the root directory, and you can get them using pip.

```
  pip install -r requirements.txt
```

How To Install GDAL/OGR Packages on Ubuntu

GDAL is a translator library for raster and vector geospatial data formats.

OGR Simple Features Library is a C++ open source library (and commandline tools) providing read (and sometimes write) access to a variety of vector file formats including ESRI Shapefiles, S-57, SDTS, PostGIS, Oracle Spatial, and Mapinfo mid/mif and TAB formats.

OGR is a part of the GDAL library.

GDAL/OGR are used in numerous GIS software projects and, lucky for us, there are bindings for python. In fact, you may want to check out the Python GDAL/OGR Cookbook.

This article describes a process you can follow to install GDAL/OGR on Ubuntu.

Before You Begin: Python 3.6

If you are installing the GDAL/OGR packages into a virtual environment based on Python 3.6, you may need to install the python3.6-dev package.

```
  sudo apt-get install python3.6-dev
```

For more information about creating virtual environments on Ubuntu 16.04 LTS, see A Note About Python 3.6 and Ubuntu 16.04 LTS.

Install GDAL/OGR

Much of this section is taken from a really helpful blog post by Sara Safavi. Follow these steps to get GDAL/OGR installed.

To get the latest GDAL/OGR version, add the PPA to your sources, then install the gdal-bin package (this should automatically grab any necessary dependencies, including at least the relevant libgdal version).

```
  sudo add-apt-repository ppa:ubuntugis/ppa && sudo apt-get update
```
Once you add the repository, go ahead and update your source packages.

```
sudo apt-get update
```

Now you should be able to install the GDAL/OGR package.

```
sudo apt-get install gdal-bin
```

To verify the installation, you can run `ogrinfo --version`.

```
ogrinfo
```

## Install GDAL for Python

Before installing the GDAL Python libraries, you'll need to install the GDAL development libraries.

```
sudo apt-get install libgdal-dev
```

You'll also need to export a couple of environment variables for the compiler.

```
export CPLUS_INCLUDE_PATH=/usr/include/gdal
export C_INCLUDE_PATH=/usr/include/gdal
```

Now you can use `pip` to install the Python GDAL bindings.

```
pip install GDAL
```

## Putting It All Together

If you want to run the whole process at once, we've collected all the commands above in the script below.

```
#!/usr/bin/env bash

sudo add-apt-repository ppa:ubuntugis/ppa && sudo apt-get update
sudo apt-get update
sudo apt-get install gdal-bin
sudo apt-get install libgdal-dev
export CPLUS_INCLUDE_PATH=/usr/include/gdal
export C_INCLUDE_PATH=/usr/include/gdal
pip install GDAL
```

## Try It Out

Now that GDAL/OGR is installed, and you can program against it in Python, why not try it out? The code block below is a sample from the Python OGR/GDAL Cookbook that gets all the layers in an Esri file geodatabase.

```
# standard imports
import sys

# import OGR
from osgeo import ogr

# use OGR specific exceptions
ogr.UseExceptions()
```
# get the driver
driver = ogr.GetDriverByName("OpenFileGDB")

# opening the FileGDB
try:
    gdb = driver.Open(gdb_path, 0)
except Exception, e:
    print e
    sys.exit()

# list to store layers' names
featsClassList = []

# parsing layers by index
for featsClass_idx in range(gdb.GetLayerCount()):
   _featsClass = gdb.GetLayerByIndex(featsClass_idx)
    featsClassList.append(_featsClass.GetName())

# sorting
featsClassList.sort()

# printing
for featsClass in featsClassList:
    print featsClass

# clean close
del gdb

Acknowledgements

Thanks to Sara Safavi and Paul Whipp for contributing some of the leg work on this.

How To Use Mother's logging Module

Mother has her own logging module which you are strongly encouraged to use within the project.

Using the loggable_class Decorator

When you create a new class, you can provide it with a Python logger just by decorating it with the @loggable_class decorator (abbreviated to @loggable in the example). The code sample below will provide a logger property to the decorated class.

```python
import logging
import sys
# (Let's alias the decorator's name for brevity's sake.)
from mothergeo.logging import loggable_class as loggable

# We'll just create a simple test configuration so we can see logging occur.
logging.basicConfig(
    stream=sys.stdout,
    level=logging.DEBUG
)
```
@loggable
class MyUsefulClass(object):
    def my_useful_method(self):
        self.logger.debug("I'm using the logger that was provided by the decorator.")
        print("You should see some logging output.")

if __name__ == "__main__":
    my_useful_object = MyUsefulClass()
    my_useful_method()

Overriding the Default Logger Name

By default, the name of the Python logger a class decorated with the @logger_class decorator creates a logger based on the module in which the class is found, and the name of the class. You can override this behavior by providing a logger_name property on the decorated class, as in the example below.

```python
import logging
import sys
# (Let's alias the decorator's name for brevity's sake.)
from mothergeo.logging import loggable_class as loggable

# We'll just create a simple test configuration so we can see logging occur.
logging.basicConfig(
    stream=sys.stdout,
    level=logging.DEBUG
)

@loggable
class MyUsefulClass(object):
    logger_name = 'alterate.logger.name'  # Override the default logger name formula.
    def my_useful_method(self):
        self.logger.debug("The logger's name should reflect the 'logger_name' property.")
        print("You should see some logging output.")

if __name__ == "__main__":
    my_useful_object = MyUsefulClass()
    my_useful_method()
```

See also:

If you're interested in reading more about logging in Python, have a look at *From the Hitchhiker's Guide: Logging in Python Libraries*.

How To Run Your Own PyPI Server

This article includes some notes we hope will be helpful in setting up your own PyPI server for those times when you need to share modules, but you're not ready to publish them to the rest of the World.
The Server Side

There are a few different ways to host your repository. This article focuses on pypi-server which you can get from the public package index.

The Client Side

Installing Modules with pip

One of the tricks to installing packages from your private repository is telling pip about it. On Windows, you can place a pip.ini file at %APPDATA%\pip\pip.ini

```
[globa1]
extra-index-url = http://<host>:<port>/

[install]
trusted-host = <host>
```

Note: If you are using SSL with a verified certificate, you won’t need the trusted-host directive.

Publishing Updates

To keep life a little simpler, you probably want to modify your .pypirc file to include information about your new repository server. You can do this by adding an alias for the server in your list of index-servers. When you’re finished, your .pypirc file might look something like the one below assuming your give your new repository myownpypi as an alias.

```
[distutils]
index-servers =
    pypi
pypitest
myownpypi

[pypi]
username=<pypi_user>
password=<pypi_password>

[pypitest]
username=<pypitest_user>
password=<pypitest_password>

[myownpypi]
repository: http://<host>:<port>
username: <myownpypi_user>
password: <myownpypi_password>
```

Note: There are refinements to this process and we’ll update this document as we go along.
How to Exclude Folders in PyCharm

If you’re using PyCharm to develop, you may have noticed that it has some pretty righteous searching and refactoring capabilities; however, there are likely to be some folders in your project’s directory tree that contain files you don’t want PyCharm to look at when it comes time to search or perform automatic refactoring. Examples of these directories include:

- the Python virtual environment (because you definitely don’t want to modify the stuff in there);
- the docs directory (because you don’t really need to refactor the stuff in there); and
- the lib directory (because nothing in there should depend on the code you’re writing, right?).

There may be others as well.

PyCharm allows you to exclude directories from consideration when searching and refactoring. You can exclude a directory by right-clicking on it and selecting Mark Directory as → Excluded.

See also:

JetBrains’ website has an article called Configuring Folders Within a Content Root which has additional insights on how and why you might want to configure the folders in the project.

How To Set Up a Virtual Python Environment (Linux)

virtualenv is a tool to create isolated Python environments. You can read more about it in the Virtualenv documentation. This article provides a quick summary to help you set up and use a virtual environment.

A Note About Python 3.6 and Ubuntu 16.04 LTS

If you’re running Ubuntu 16.04 LTS (or and earlier version), Python 3.5 is likely installed by default. Don’t remove it! To get Python 3.6, follow the instructions in this section.

Add the PPA

Run the following command to add the Python 3.6 PPA.

```bash
sudo add-apt-repository ppa:jonathonf/python-3.6
```

Check for Updates and Install

Check for updates and install Python 3.6 via the following commands.

```bash
sudo apt-get update
sudo apt-get install python3.6
```

Now you have three Python version, use python to run version 2.7, python3 for version 3.5, and python3.6 for version 3.6.

For more information on this subject, check out Jim’s article How to Install Python 3.6.1 in Ubuntu 16.04 LTS.
Create a Virtual Python Environment

`cd` to your project directory and run `virtualenv` to create the new virtual environment.

The following commands will create a new virtual environment under `my-project/my-venv`.

```
cd my-project
virtualenv --python python3.6 venv
```

Activate the Environment

Now that we have a virtual environment, we need to activate it.

```
source venv/bin/activate
```

After you activate the environment, your command prompt will be modified to reflect the change.

Add Libraries and Create a requirements.txt File

After you activate the virtual environment, you can add packages to it using `pip`. You can also create a description of your dependencies using `pip`.

The following command creates a file called `requirements.txt` that enumerates the installed packages.

```
pip freeze > requirements.txt
```

This file can then be used by collaborators to update virtual environments using the following command.

```
pip install -r requirements.txt
```

Deactivate the Environment

To return to normal system settings, use the `deactivate` command.

```
deactivate
```

After you issue this command, you’ll notice that the command prompt returns to normal.

Acknowledgments

Much of this article is taken from The Hitchhiker's Guide to Python. Go buy a copy right now.

How To Set Up a Virtual Python Environment (Windows)

`virtualenv` is a tool to create isolated Python environments. You can read more about it in the Virtualenv documentation. This article provides a quick summary to help you set up and use a virtual environment.
Where’s My Python?

Sometimes the trickiest part of setting up a virtual environment on Windows is finding your python distribution. If the installer didn’t add it to your PATH variable, you may have to go looking. If you downloaded and installed python from python.org and accepted all the defaults during installation, python.exe may be found in one of the following locations:

**64-bit (Preferred)**

```
C:\Users\%username%\AppData\Local\Programs\Python\Python36\python.exe
```

**32-bit**

```
C:\Users\%username%\AppData\Local\Programs\Python\Python36-32\python.exe
```

Install `virtualenv`

If you try to run `virtualenv` and find it isn’t present, you can install it using `pip`.

```
pip install virtualenv
```

`virtualenv.exe` will likely now be found in your python installation directory under the `Scripts` subdirectory.

Create a Virtual Python Environment

`cd` to your project directory and run `virtualenv` to create the new virtual environment.

The following commands will create a new virtual environment under `my-project/my-venv`.

```
cd my-project
virtualenv --python C:\Path\To\Python\python.exe venv
```

**Note:** If Windows cannot find `virtualenv.exe`, see `Install virtualenv`. You can either add the executable’s home directory to your PATH variable, or just include the full path in your command line. If you aren’t sure where `python.exe` is installed, see `Where’s My Python?`.

Activate the Environment

Now that we have a virtual environment, we need to activate it.

```
.\venv\Scripts\activate
```

After you activate the environment, your command prompt will be modified to reflect the change.
Add Libraries and Create a requirements.txt File

After you activate the virtual environment, you can add packages to it using `pip`. You can also create a description of your dependencies using `pip`.

The following command creates a file called `requirements.txt` that enumerates the installed packages.

```bash
pip freeze > requirements.txt
```

This file can then be used by collaborators to update virtual environments using the following command.

```bash
pip install -r requirements.txt
```

Deactivate the Environment

To return to normal system settings, use the `deactivate` command.

```bash
deactivate
```

After you issue this command, you’ll notice that the command prompt returns to normal.

Acknowledgments

Much of this article is taken from The Hitchhiker’s Guide to Python. Go buy a copy right now.

Miscellany

From the Hitchhiker’s Guide: Logging in Python Libraries

**Note:** If you’re wondering how Mother likes to handle logging, take a look at *How To Use Mother’s logging Module*.

It is, of course, desirable for library modules to perform logging. However, we generally want to maintain consistency and allow the consuming application to perform logging configuration. This article details a strategy for achieving those goals.

A Bit of Logging Boilerplate

From *The Hitchhiker’s Guide to Python*, Chapter 4:

Do not add any handlers other than `NullHandler` to your library’s loggers. Place the following code in your project’s top-level `__init__.py`.

```python
# Set default logging handler to avoid "No handler found" warnings.
import logging
try:  # Python 2.7+
    from logging import NullHandler
except ImportError:
    class NullHandler(logging.Handler):
        def emit(self, record):
            pass
```
logging.getLogger(__name__).addHandler(NullHandler())

**Acknowledgments**

Much of this article is taken from *The Hitchhiker’s Guide to Python*. Go buy a copy right now.
Read the docs. (This change is inserted really just to demonstrate the PyCharm VCS integration.)

mothergeo

GIS handling that cares.

mothergeo.codetools

Helpful utilities from mother.

class mothergeo.codetools.Dicts
    Bases: object

    This is a utility class that wants to help you work with dict types.

    __init__
    Initialize self. See help(type(self)) for accurate signature.

    static try_get (obj: dict, key: str, default=None) → mothergeo.codetools.TryGetResult
    Try to retrieve a value from a dict that may, or may not be present.

    Parameters

    - obj (dict) – the object that defines the value
    - key (str) – the key whose value you want
    - default – the value that will be returned if no value is defined for the key

    Returns a named tuple that indicates whether or not the key was defined, and the value

    Return type TryGetResult
class mothergeo.codetools.Enums
    Bases: object

    This is a utility class that wants to help you work with Enum types.

    __init__
    Initialize self. See help(type(self)) for accurate signature.

    static from_name (enum_cls, name: str) → enum.Enum
    Get an enumeration member value from its name.

    Parameters
    • enum_cls (class) – the Enum class
    • name (str) – the enumeration member’s name

    Returns the enumeration member

    Return type Enum

class mothergeo.codetools.Iters
    Bases: object

    This is a utility class that wants to help you work with iterators.

    __init__
    Initialize self. See help(type(self)) for accurate signature.

    static count (it: typing.Iterator) → int
    Get the number of items in an iterator.

    Parameters it (iter) – the iterator

    Returns the number of items in the iterator

    Return type int

    static get_item_at (it: typing.Iterator, index: int)
    Get the item at a given index from an iterator.

    Parameters
    • it (iter) – the iterator
    • index (int) – the index of the item you want

    Returns the item at the specified index

class mothergeo.codetools.TryGetResult
    Bases: mothergeo.codetools.TryResult

    This is a named tuple that contains a bool result that indicates success or failure of a “try-get” operation, and the value retrieved.

    __init__
    Initialize self. See help(type(self)) for accurate signature.

    count (value) → integer – return number of occurrences of value

    index (value[, start[, stop]]) → integer – return first index of value.
    Raises ValueError if the value is not present.

    result
    Alias for field number 0
```
value
   Alias for field number 1

mothergeo.config

May I take your order, please?

class mothergeo.config.ConfigurationManager
   Bases: object

   Use a configuration manager to keep track of the configurable aspects of the system.

   __init__()

   get (section: str, option: str, fallback: str = None) → str
      Get a configured value.

      Parameters
      • section (str) – the name of the configuration section
      • option (str) – the name of the configuration option
      • fallback (str) – the value to return if no configured value is found

      Returns  the configured value

      Return type  str

load (config_info: str)
   Load configuration information.

      Parameters  config_info (str) – the configuration information (most likely a path to a file or something similar)

map_env_variable (section: str, option: str, env_var: str)
   Map an data_store variable to a configuration option.

      Parameters
      • section (str) – the configuration section
      • option (str) – the configuration option name
      • env_var (str) – the name of the data_store variable

set (section: str, option: str, value: str)
   Set a configuration value.

      Parameters
      • section (str) – the configuration section
      • option (str) – the configuration option name
      • value (str) – the new value

unmap_env_variable (section: str, option: str) → str
   Map an data_store variable to a configuration option.

      Parameters
      • section (str) – the configuration section
      • option (str) – the configuration option name
```
Returns the unmapped data_store variable name (if any)

Return type str or None

mothergeo.db.postgis.geoalchemy

mothergeo.db.modeling

mothergeo.db.postgis.modeling

Tools for working with data models in PostGIS.

mothergeo.i18n

Think locally, act globally. These are tools to help with internationalization.

class mothergeo.i18n.I18nPack(initialdata: dict = None)
   Bases: collections.UserDict

   This is a dictionary of translatable strings.

   Example

   >>> pack = I18nPack({
   'alpha': 'apple',
   'beta': 'banana',
   'gamma': 'grapes'
   })
   >>> pack.add_translation('alpha', '', 'ja_jp')
   >>> pack.add_translation('beta', '', 'ja_jp')

See also:

I18nPack.add_translation()

__init__(initialdata: dict = None)

   Parameters initialdata (dict) – a dictionary you can use to seed this package

   add_translation (key: str, translation: str, locale: str = None)
   Add a translation to this package.

   Parameters

   • key (str) – the key used to retrieve the translated value
   • translation (str) – the translation you want people to read
   • locale (str) – the locale in which the translation can be understood

   clear() → None. Remove all items from D.

copy()

fromkeys (iterable, value=None)

get (k[, d]) → D[k] if k in D, else d. d defaults to None.
**items**() → a set-like object providing a view on D’s items

**keys**() → a set-like object providing a view on D’s keys

**pop**(k[, d]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

**popitem**() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

**set_translations**(translations: typing.Dict[str, str], locale: str = None)
Update the pack with a complete set of translations.

Parameters

- translations (dict[str, str]) – the translations for the specified locale
- locale (str) – the locale in which the translations would be understood (or None to set the defaults)

Raises **ValueError** – if translations is not a dict

**setdefault**(k[, d]) → D.get(k,d), also set D[k]=d if k not in D

**update**(E[, **F]) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

**values**() → an object providing a view on D’s values

mothergeo.i18n.current_locale = None
The current locale (None indicates the default locale.)

mothergeo.i18n localize(s: str) → str
Localize a string.

Parameters s (str) – the original string

Returns

mothergeo.schemas.modeling

mothergeo.schemas.parsing

mothergeo.geometry
CHAPTER 3

Indices and tables

- genindex
- modindex
- search
m

mothergeo, 17
mothergeo.codetools, 17
mothergeo.config, 19
mothergeo.db.postgis.modeling, 20
mothergeo.i18n, 20
Symbols

__init__ (mothergeo.codetools.Dicts attribute), 17
__init__ (mothergeo.codetools.Enums attribute), 18
__init__ (mothergeo.codetools.Iters attribute), 18
__init__ (mothergeo.codetools.TryGetResult attribute), 18
__init__() (mothergeo.config.ConfigurationManager method), 19
__init__() (mothergeo.i18n.I18nPack method), 20
add_translation() (mothergeo.i18n.I18nPack method), 20
ConfigurationManager (class in mothergeo.config), 19
clear() (mothergeo.i18n.I18nPack method), 20
copy() (mothergeo.i18n.I18nPack method), 20
count() (mothergeo.codetools.Iters static method), 18
count() (mothergeo.codetools.TryGetResult method), 18
current_locale (in module mothergeo.i18n), 21
D
dicts (class in mothergeo.codetools), 17
E
Enums (class in mothergeo.codetools), 17
F
from_name() (mothergeo.codetools.Enums static method), 18
fromkeys() (mothergeo.i18n.I18nPack method), 20
G
get() (mothergeo.config.ConfigurationManager method), 19
get() (mothergeo.i18n.I18nPack method), 20
get_item_at() (mothergeo.codetools.Iters static method), 18
I
I18nPack (class in mothergeo.i18n), 20
index() (mothergeo.codetools.TryGetResult method), 18
items() (mothergeo.i18n.I18nPack method), 20
Iters (class in mothergeo.codetools), 18
K
keys() (mothergeo.i18n.I18nPack method), 21
L
load() (mothergeo.config.ConfigurationManager method), 19
localize() (in module mothergeo.i18n), 21
M
map_env_variable() (mothergeo.config.ConfigurationManager method), 19
mothergeo (module), 17
mothergeo.codetools (module), 17
mothergeo.config (module), 19
mothergeo.db.postgis.modeling (module), 20
mothergeo.i18n (module), 20
P
pop() (mothergeo.i18n.I18nPack method), 21
popitem() (mothergeo.i18n.I18nPack method), 21
R
result (mothergeo.codetools.TryGetResult attribute), 18
S
set() (mothergeo.config.ConfigurationManager method), 19
set_translations() (mothergeo.i18n.I18nPack method), 21
setdefault() (mothergeo.i18n.I18nPack method), 21
T
try_get() (mothergeo.codetools.Dicts static method), 17
TryGetResult (class in mothergeo.codetools), 18

U
unmap_env_variable() (mothergeo.config.ConfigurationManager method), 19
update() (mothergeo.i18n.I18nPack method), 21

V
value (mothergeo.codetools.TryGetResult attribute), 18
values() (mothergeo.i18n.I18nPack method), 21