TurboGears2 Cheatsheet

version 0.1

credits: @__amol__ @pna

SQLAlchemy Models

```
from sqlalchemy.orm import relation, backref
from sqlalchemy import ForeignKey, Column
from sqlalchemy.types import *
from myproj.model import DeclarativeBase, DBSession

class SampleModel(DeclarativeBase):
    __tablename__ = 'sample_model'

    id = Column(Integer, autoincrement=True, primary_key=True)
    data = Column(Unicode(255), nullable=False)
```

Basic SQLA declarative model, most common column types are: Boolean, Date, DateTime, Float, Integer, LargeBinary, Numeric (precision=10, length=2), PickleType, String(length=None), Unicode(length=None), UnicodeText

```
DBSession.query(MyModel).filter_by(name='John').first()
```
Gets one row by name or None

```
DBSession.query(MyModel).get(5)
```
Gets one row by primary key or None, works also as a cache: If the row is already inside the Unit of Work the database won't be queried again

```
DBSession.query(MyModel).filter(MyModel.name=='John') \
                        .filter (MyModel.surname!='Doe').all()
```
Gets all the people that are named John but are not Doe

```
DBSession.query(MyModel).filter(MyModel.name.like('John%')).all()
```
Gets all the people that have a name starting with John

```
DBSession.query(MyModel).filter(MyModel.name=='John' |
                                MyModel.name=='Aldo').all()
```
Gets all people that are named John or Aldo

```
DBSession.query(MyModel).offset(50).limit(10).all()
```
Gets 10 models after the 50th model, useful for basic pagination

```
DBSession.query(MyModel).order_by(MyModel.age.desc()).all()
```
Gets all models ordered by descending age

EasyCrudRestController

```
class UsersController(EasyCrudRestController):
    title = "Manage People"
    model = model.User

    __form_options__ = {
        '__hide_fields__' : ['uid'],
        '__omit_fields__' : ['_password']
    }
    __table_options__ = {
        '__omit_fields__' : ['_password']
    }
```

Creates a CRUD controller to edit Users hiding uid field when editing and omitting the _password field both when editing and displaying users table

```
class PhotosController(EasyCrudRestController):
    allow_only = predicates.in_group('photos')
    title = "Manage Photos"
    model = model.Photo
    keep_params = ['gallery']
    __form_options__ = {
        '__hide_fields__' : ['uid', 'author', 'gallery'],
        '__field_widget_types__' : {'image':FileField},
        '__field_validator_types__' : {'image':FieldStorageUploadConverter},
        '__field_widget_args__' : {'author':
            {'default':lambda:request.identity['user'].user_id}}
    }
    __table_options__ = {
        '__omit_fields__' : ['uid', 'author_id', 'gallery_id', 'gallery'],
        '__xml_fields__' : ['image'],
        'image': lambda filler,row: html.literal('‹img src="%s"/›' % row.image.thumb_url)
}
```

Creates a CRUD controller to edit a photo gallery with file upload field for photos, restricts access only to people in photos group and keeps passing around the gallery parameter to avoid losing which gallery we were editing.

Also displays a preview of the image inside the table, image upload is implemented using tgext.datahelpers attachments

Quickstarting Projects

`easy_install -i http://tg.gy/current tg.devtools`	Install Last Stable version of TurboGears
`paster quickstart --noinput projectname`	Create a project with Genshi, SQLAlchemy, ToscaWidgets2 and Authentication
`paster quickstart --jinja --ming projectname`	Create a project with Jinja2 and MongoDB
`paster quickstart --mako --nosa --enable-tw1 projectname`	Create a project without database using Mako and ToscaWidgets1
`paster setup-app development.ini`	Initializes the database creating tables and default data, websetup/bootstrap.py defines the default data [Not required if not using any storage or Auth]
`paster serve development.ini`	Launches application inside development server

Widgets

```
import tw2.core as twc
import tw2.forms as twf
from tg import lurl

class MovieForm(twf.TableForm):
  title = twf.TextField(validator=twc.Required)
  director = twf.TextField(validator=twc.Required)
  director_verify = twf.TextField()
  genres = twf.CheckBoxList(options=['Action', 'Comedy', 'Romance'])

  action = lurl('/save_movie')
  validator = FieldsMatch('director', 'director_verify')
```

Basic tw2 form, with two edit fields and a list of checkbox for genres

Provides also a formwide validator that checks if director and director_verify match.
Validators can either be a tw2.core validator or a formencode one.

Add @validate(MovieForm, error_handler=new_movie) to controllers to validate the form and go to new_movie if it fails

Genshi

```
<html xmlns:py="http://genshi.edgewall.org/">
  <head>
    <title>${page_title}</title>
  </head>
  <body>
    <h1 py:if="header">${header}</h1>
    <div py:replace="page_body">this is the body</div>
  </body>
</html>
```

Basic Genshi Template, genshi provides additional attributes with namespace **py:** to manage template behavior.

Templating

`py:if="condition"`	Tag and its content will be removed if the condition is False
`py:strip="condition"`	Tag but not its content will be removed if condition is True
`py:choose="a-value"`	Like a switch statement, tags inside the one having the py:choose attribute can use py:when and py:otherwise to control when they must be removed or available
`py:for="item in itemlist"`	Content of the tag will be looped for each item in itemlist
`py:content="expression"`	Content of the tag will be replaced with the value of expression
`py:replace="expression"`	Tag and its content will be replaced with the value of expression
`py:attrs="a-dict"`	An attribute for each entry in the dictionary will be added to the tag with name and value equal to dictionary keys and values
`${Markup(expression)}`	Expression won't be escaped, prefer Markup to XML.

Blocks & Inheritance

`<xi:include href="otherpage" />`	Content of otherpage will be included inside current page
`py:def="function(params)"`	Tag with py:def will be exposed as a python function, calling the function inside ${} will put the content of the tag in place of the ${} expression. This permits to implement blocks
`py:match="XPath expression"`	Tag with py:match attribute will replace any tag that matches the provided XPath expression. This permits to implement template inheritance
`py:attrs="select('@*')"`	When added to a tag with py:match, the tag will inherit all the attributes of the tag matched by the py:match expression
`${select("*|text()")}`	When added inside a tag with py:match the tag will replace the expression with the content of the tag matched by the py:match expression

Sprox

Tables

TableForm	__base_widget_type__	What widget to use for the form
SAWidgetSelector	__widget_selector_type__	What class to use for widget selection
SAValidatorSelector	__validator_selector_type__	What class to use for validator selection
[]	__require_fields__	Specifies which fields are required
FALSE	__check_if_unique__	Set this to True for "new" forms. This causes Sprox to check if there is an existing record in the database which matches the field data
{}	__field_validators__	A dictionary of validators indexed by fieldname
{}	__field_validator_types__	Types of validators to use for each field (allow sprox to set the attribute of the validators)
None	__base_validator__	A validator to attach to the form
None	__validator_selector__	What object to use to select field validators
FieldsMetadata	__metadata_type__	What metadata type to use to get schema info on this object
None	__dropdown_field_names__	List or dict of names to use for discovery of field names for dropdowns (None uses sprox default names) a dict provides field-level granularity
{}	__field_widgets__	A dictionary of widgets to replace the ones that would be chosen by the selector
{}	__field_widget_types__	A dictionary of types of widgets, allowing sprox to determine the widget args
None	__widget_selector__	An instantiated object to use for widget selection
None	__entity__ (__model__)	Entity used for metadata extraction
None	__field_order__	A list of ordered field names
[]	__hide_fields__	Fields marked as hidden
{}	__add_fields__	Additional fields to add to the config
[]	__disable_fields__	Field marked as disabled
[]	__omit_fields__	Fields removed from the field list
None	__limit_fields__	Limit the field list to this
{}	__field_attrs__	Attr parmater to set to the field

Forms

SproxDataGrid	__base_widget_type__	Base widget for fields to go into
{}	__headers__	A dictonay of field/header pairs
{}	__column_widths__	A dictionay of field/width(string) pairs
'10em'	__default_column_width__	Header size to use when not specified in __column_widths__
[]	__xml_fields__	Fields whos values should show as html
{}	__field_widgets__	A dictionary of widgets to replace the ones that would be chosen by the selector
{}	__field_widget_types__	A dictionary of types of widgets, allowing sprox to determine the widget args
None	__widget_selector__	An instantiated object to use for widget selection
None	__entity__ (__model__)	Entity used for metadata extraction
None	__field_order__	A list of ordered field names
[]	__hide_fields__	Fields marked as hidden
{}	__add_fields__	Additional fields to add to the config
[]	__disable_fields__	Field marked as disabled
[]	__omit_fields__	Fields removed from the field list
None	__limit_fields__	Limit the field list to this
{}	__field_attrs__	attr parmater to set to the field

i18n

python setup.py extract_messages	Extract all the translatable strings from your project's files and generate a "pot" file in the i18n folder of your application
python setup.py init_catalog -l zh_tw	Create a translation catalog for your language
python setup.py compile_catalog	Compile the catalog after editing i18n/[country code]/LC_MESSAGES/[project-name].po

This cheatsheet as of now is based on TurboGears 2.2 for additional information and details please consider RTFM on http://docs.turbogears.org

This was made with passion for lazy and forgetful people... you owe us a beer

Decorators

@with_trailing_slash	This decorator allows you to ensure that the URL ends in "/".
@https	Ensure that the decorated method is always called with https.
@without_trailing_slash	This decorator allows you to ensure that the URL does not end in "/".
@after_render	A callable to be run after the template is rendered.
@allow_only(predicate)	TurboGears controller wide protector authorization for a predicate
@before_call	A callable to be run before the controller method is called.
@before_render	A callable to be run before the template is rendered.
@before_validate	A callable to be run before validation is performed.
@cached_property	Cached property executing the getter only once.
@expose(...)	Register attributes on the decorated function. Allowed parameters (template, content_type, exclude_names, custom_format, render_params, inherit)
@paginate(name, ...)	Paginate a given collection. This decorator is mainly exposing the functionality of webhelpers.paginate().
@require(predicate)	TurboGears-specific action protector.
@validate(...)	Registers which validators ought to be applied
@with_engine(...)	Decorator to force usage of a specific database engine in TurboGears SQLAlchemy BalancedSession

Hooks

```
def on_startup():
    print 'hello, startup world'

def on_shutdown():
    print 'hello, shutdown world'

def before_render(remainder, params, output):
    print 'system wide before render'

# ... (base_config init code)

base_config.register_hook('startup', on_startup)
base_config.register_hook('shutdown', on_shutdown)
base_config.register_hook('before_render', before_render)
```

TurboGears allows you to attach callables to a wide set of events. Most of those are available as both controller events and system wide events.
To register a system wide event you can use the register_hook method of the base_config object in your app_cfg.py file

```
from tg.decorators import before_render

def before_render_cb(remainder, params, output):
    print 'Going to render', output

class MyController(TGController):
    @expose()
    @before_render(before_render_cb)
    def index(self, *args, **kw):
        return dict(page='index')
```

To register controller based hooks you can use the event decorators

App wide

startup()	Called when the application starts
shutdown()	Called when the application exits
before_config(app) → app	Called after constructing the application
after_config(app) → app	Called after finishing setting everything up
before_validate(remainder, params)	Called before performing validation
before_call(remainder, params)	Called after valdation, before calling the actual controller method
before_render(remainder, params, output)	Called before rendering a controller template, output is the controller return value
after_render(response)	Called after finishing rendering a controller template

credits: @__amol__ @pna

TurboGears2 Cheatsheet

version 0.1