

 del.icio.us,  Dig IT,  PDF

Contents

1. Start Here
 1. Short Intro To TurboGears
 2. Search TG docs
2. Install TurboGears
3. QuickStart a new project
 1. Create project
 2. Start the project
 3. Make changes
4. Turbogears Overview
 1. Add,remove URL in controller.py
 2. Database settings
 3. Database tables using model.py
 4. Kid template
5. First webapp: Address Book
 1. Database Model
 1. Import Proper Modules
 2. Create Tables
 3. Map Your Table To Python Class
 4. Create Tables in Database
 2. Controls
 3. Template
 4. Save Form
 5. Edit form
6. TurboGears Forms
 1. Widgets and Validators
 1. Create a Form
 2. Create URL
 3. Create template
 4. Process the form
 2. Widgets options
 3. Validators options
 4. Password validators
 5. Validate Widget
7. TurboGears dispaly Widgets
 1. DataGrid
 1. DataGrid and SQLAlchemy Example
 2. DataGrid Example with static content
 2. FastDataGrid
8. SQLAlchemy (SO) quick Overview
 1. dev.conf -database connection
 2. model.py -database model
 1. sqlalchemy features

2. SQLAlchemy save to database
9. Install Turbogears with Apache
 1. Egg deploy
 1. Create an Egg File
 2. dev.cfg app.cfg prod.cfg
 3. Deploy an EGG file
 4. Configure TurboGears prod.cfg
 5. Run the app
 2. Mod_Proxy
 1. Enable mod proxy
 2. Configure apache
 3. Auto start of the app
 4. Server Log
 3. apache and mod_wsgi
 1. Install mod_wsgi
 2. Configure your Turbogears app for mod_wsgi
 3. advance mod_wsgi memory management
 4. Stress test your app
 4. mod_python
 1. Install project file
 2. Test project file
 3. Configure apache2 with your turbogears project
 4. Error
10. Done simple, Go Advance
11. Create a project (LEVEL II)
 1. Introduction
12. SQLAlchemy (SA)
 1. model.py Connection, and setup to existing database
 2. dev.conf and database connection
 1. what needs to be imported
 2. bind to database
 3. create tables
 4. create python data class
 5. map database to python class
 3. create table in sqlalchemy
 4. controller.py, Data passing and conversion
 5. Direct Sql statements to a database
 6. Basic select()
 1. equal, not equal, less then, more then
 2. more or less
 3. and ,or ,not
 4. like, endswith, startswith, between, in
 7. Advenced select()

1. different select:
 sqlalchemy.select()
2. group_by
3. select_by
8. available functions of sqlalchemy
 1. properties of python class and sql
 2. properties of a field name
 3. list of available fields, properties
 4. working with columns
 5. Select() function properties
 6. get() function
9. insert record
10. update record
 1. Display record
11. Flow Control:
 update,submit,save,display
12. Saving table with many columns
13. Saving multiple records
14. sqlalchemy in virtualenv
13. Session Management
 1. Session ID
14. Turbogears API Details
 1. Packages
 2. Modules
 3. Classes
15. Performance
 1. Genshi vs other
 2. maco vs Cheetah
 3. Sqlalchemy vs storm
16. Errors
 1. TgFastData for python 2.5
17. EXTRA
 1. Produce PDF Pages
 2. MyTube with Flex and Turbogears
 3. captcha widgets for a form
 4. Basic Way to display column data from a database
 5. Identity Management
 6. TurboGears with existing MySQL database
 7. Flash status Bar
 8. Tagging with Turbogears
 9. Drag and Drop Sort list
 10. Turbogears and Oracle
 11. Turbogears memory management
 12. unixODBC
 13. Authenticating against an external password source
 14. Embed swf files in your turbogears

- code (IE, Opera, Firefox etc)
18. TurboGears Specifics usage
 1. Use CAPITAL names in widgets and sqlalchemy autoloader
 2. redirect all requests to index
 19. Other Documentation
 20. Common Errors
 1. non-keyword arg after keyword arg
 2. cherrypy._cperror.NotReady: Port not free
 3. ExpatError: not well-formed (invalid token)
 4. Could not assemble any primary key columns for mapped table

Original Document is located here: <http://lucasmanual.com/mywiki/TurboGears>

Keywords: Tutorial, TurboGears, SqlObject, SqlAlchemy, web, python, features, Linux, windows, database, mysql, Examples, web, cherrypy, manual, easy, first time to turbogears, documentation, solutions, fixed, solved, working, pylones, python web framework, TurboGears widgets, TurboGears Forms, TurboGears controller, TurboGears model, TurboGears and sqlalchemy, TurboGears and AJAX, Turbogears Documentation, Turbogears Manual, Turbogears Howto, Turbogears how to, Turbogears faq, trubogears

Start Here

- **Welcome.**
- **Please start from the beginning of this page if you are new to TurboGears.**
- **If you are looking for something specific please go over the menu items.**

Short Intro To TurboGears

If it is your first time working with turbogears, this [TurboGears Article](#) will give you an overview of what is what in turbogears. It will show you how turbogears connects database, templates, javascript via python in one simple framework. Here is a copy of just TurboGears related pages [Turbogears in O3 Magazine, Issue5 \(copy\)](#)

Search TG docs

Just looking for specific information? Look through menu first, then try [TurboGears Search engine](#)

Install TurboGears

For TurboGears to work we need the following items.

1. TurboGears.
2. Sqlite, python bindings for sqlite or mysql and mysql python bindings.

We will use Linux since it is the easiest operating system to work with.

- Install turbogears

```
aptitude update
aptitude install python-turbogears
```

- Install sqlite it will get used especially in really quick tg apps.

```
aptitude install sqlite3
aptitude install python-pysqlite2
```

- Install mysql and/or python binding for mysql

```
aptitude install mysql-server
aptitude install python-mysqldb
```

You are ready to start your turbogears development.

QuickStart a new project

Create project

- This command will create your tg app. It will ask you for your project name, etc.

```
tg-admin quickstart --sqlalchemy
```

- After running the command you should see questions like this:

```
Enter project name: myfirstproject
Enter package name [myfirstproject]
```

- When done it will create folder called 'myfirstproject'

Start the project

- To start a project you just run a start file.
- Go into the turbogears folder and start your project. Replace *myfirstproject* with your

project name.

```
python start-myfirstproject.py
```

- After you start it go to your internet browser and go to this website. Your app is running in a turbogears test server at <http://localhost:8080>

Make changes

Just go into your project folder, inside there should be:

1. dev.cfg - where you set configuration properties for your project.
2. setup.py - which you can use to install your python project into python site-package
3. myfirstproject folder - There is a folder with a same name as your project, inside there are:
 1. templates - which hold all your html templates (welcome.kid)
 2. static - which holds your images, css, and javascript
 3. config- where you will find your app specific configuration.
 4. model.py - which has your data models
 5. controller.py - which has your data manipulations, your urls, etc.

You can start by editing the main page. In my case I go to myfirstproject/templates and in there you can find the welcome.kid . It is an xhtml file so you can change some text and see how they show up immediately in <http://localhost:8080>.

Turbogears Overview

Add,remove URL in controller.py

- Methods in your controllers.py file map directly to URL.
- In your projects' folder there is a file called controllers.py. In there you should see a code that looks like:

```
@expose(template="myfirstproject.templates.welcome")
def index(self):
    import time
    # log.debug("Happy TurboGears Controller Responding For Duty")
    return dict(now=time.ctime())
```

- First Line. You can reach index page which uses templates.welcome when you visit the <http://localhost:8080>
- Second line. You can access this url. To create url with ending hello <http://localhost:8080/hello>, add the following lines.

- Define a function hello and you will need **hello.kid** in your template folder.

```
@expose(template="myfirstproject.templates.hello")
def hello(self):
    import time
    # log.debug("Happy TurboGears Controller Responding For Duty")
    return dict(greeting="Hello from a controller.py")
```

Database settings

- Edit dev.cfg and where you see the sqlalchemy.dburi replace it with your settings.
- You can use the default or comment it out and use some of the other examples that they have provided. mysql, sqlite, etc.

```
sqlalchemy.dburi="sqlite://%(current_dir_uri)s/devdata.sqlite"
```

If you want to use a true database from a start you can use syntax like this for:

```
sqlalchemy.dburi="mysql://username:password@hostname:3306/databasename"
sqlalchemy.dburi="postgres://username:password@hostname/databasename"
sqlalchemy.dburi="oracle://username:password@hostname:1521/sidname"
sqlalchemy.dburi="mssql://username:password@hostname:1433/databasename?driver=TDS"
```

You don't have to provide the port as sqlalchemy knows default ports.

Database tables using model.py

- Now that we know where the database is, lets create few tables.
- Before you start change the first line, because that will allow you to differentiate different components in sqlalchemy.
- Change:

```
from sqlalchemy import *
to
import sqlalchemy
```

- You do it in model.py. Add something similar to this one class.

```
address_table = sqlalchemy.Table('address', metadata,
    sqlalchemy.Column('Address_Sid', sqlalchemy.Integer,
primary_key=True),
    sqlalchemy.Column('FirstName',
sqlalchemy.Unicode(40), nullable=False),
    sqlalchemy.Column('LastName', sqlalchemy.Unicode(40), nullable=False),
    sqlalchemy.Column('MaidenLastName', sqlalchemy.Unicode(40)),
```

```

sqlalchemy.Column('Email', sqlalchemy.Unicode(80), nullable=False),
sqlalchemy.Column('Address', sqlalchemy.Unicode(80), nullable=False),
sqlalchemy.Column('City', sqlalchemy.Unicode(80), nullable=False),
sqlalchemy.Column('State', sqlalchemy.String(2), nullable=False),
sqlalchemy.Column('ZipCode', sqlalchemy.Integer, nullable=False),
sqlalchemy.Column('DOB', sqlalchemy.Date(), nullable=False),
sqlalchemy.Column('CreatedDate', sqlalchemy.Date,
default=datetime.now().date()),
sqlalchemy.Column('CreatedTime', sqlalchemy.Time,
default=datetime.now().time())
)

```

- Save and exit. Now that model.py has its first table lets create it in our database.
- **SqlAlchemy: Database tables are classes, rows are instances, fields are attributes**
- To create actual database in sqlite,mysql,postgres, etc issue this command.

```
tg-admin sql create
```

Kid template

- Kid template is a set of xml files in which you are able to replace text,tags, etc. All kid files need to be a valid xml, xhtml files.
- To replace text in kid template you can do few things.The most simplest one is to define a variable. In this example we define it in template. In normal use this variable would be passed into the template from controller.py.
- Options on replacing text are:

```

py:content      -Replace content inside of a tag <a py:content> replace
this text</a>
py:if           -Do if statement: py:if="x" if x is true display, else it
will remove the tag and its children.
py:for         -Do for loop
py:replace     -Replace whole tag: <a py:replace="<a> my default meta
tag</meta>"\>
py:strip
py:match       -Matches a tag in a template and replaces with the content
you specify.(headers, footers)
py:def

```

- **Example: Replace Text**

```

<?python
myvariable = 'i want this text to replace the variable name in template'
?>
<span py:content="myvariable">Text to be replaced</span>

```

- **Replace from file:**

```
<div py:content="document('header.html')">Replace this text</div>
```

- Replace whole tag. This line will replace the whole meta tag with a value we have it, aka. *So if you use this line to your template, the meta tag won't appear in the final output of the page.*

```
<meta content="text/html";charset=UTF-8" http-equiv="content-type"
py:replace="" />
```

- For loop

```
<ul>
  <li py:for="person in PhoneBook">
    <a href="$person.home_page">
      <span py:content="person.last_name">Name to be
replaced</span>,
      <span py:content="person.first_name">Name to be
replaced</span>
    </a>
```

- If statement:

```
<div py:if="person.first_name and person.last_name" >
  <span py:content="person.last_name">Name to be
replaced</span>,
  <span py:content="person.first_name">Name to be
replaced</span>
</div>
```

or

```
<div py:if="value_of('somefield', None)" py:content="somecontent"></div>
```

or

```
<p py:if="5 * 5 == 25">
  Python seems to be handling multiplication okay.
</p>
```

First webapp: Address Book

- Create a new project.

```
tg-admin quickstart --sqlalchemy
```

- Now open a dev.cfg and add your database information in there, follow the example connection string they have in there.
- Or pick one of these:

```
sqlalchemy.dburi="sqlite://%(current_dir_uri)s/devdata.sqlite"  
sqlalchemy.dburi="mysql://username:password@hostname:3306/databasename"  
sqlalchemy.dburi="postgres://username:password@hostname/databasename"  
sqlalchemy.dburi="oracle://username:password@hostname:1521/sidname"  
sqlalchemy.dburi="mssql://username:password@hostname:1433/databasename?driver=TDS"
```

You don't have to provide the port as sqlalchemy knows default ports.

- You are done with your app connection to a database. SQLAlchemy/Turbogears will take it from here.

Database Model

- Model.py handles the definitions of all your data.
- It is here where you define your tables, data models, anything that needs a definition.

Before we start lets make sure you have all modules imported and you know what is for what.

1. Make sure you imported proper modules.
2. Define your tables. 1.Create and bind your database to python class.

Import Proper Modules

1. You should see an import statements like this(if you don't need identity this list will be shorter):

```
from datetime import datetime    #this imports date and time manipulation  
library  
from turbogears.database import metadata, mapper    #This maps database to  
a python class (more on it soon)  
  
# import some basic SQLAlchemy classes for declaring the data model  
# (see http://www.sqlalchemy.org/docs/04/ormtutorial.html)  
from sqlalchemy import Table, Column, ForeignKey  
from sqlalchemy.orm import relation  
  
# import some datatypes for table columns from SQLAlchemy  
# (see http://www.sqlalchemy.org/docs/04/types.html for more)  
#from sqlalchemy import String, Unicode, Integer, DateTime, Time, Date,  
TEXT  
import sqlalchemy  
from turbogears import identity
```

Create Tables

- Its time to create your table now.
- With sqlalchemy you have choice of either defining your own table or loading already existing table from database.
- The database you are connecting is defined in dev.cfg.

Define a table

```
#Your table definition here.
address_table = sqlalchemy.Table('addressbook', metadata,
    sqlalchemy.Column('Address_Sid', sqlalchemy.Integer,
primary_key=True),
    sqlalchemy.Column('FirstName',
sqlalchemy.Unicode(40), nullable=False),
    sqlalchemy.Column('LastName', sqlalchemy.Unicode(40), nullable=False),
    sqlalchemy.Column('MaidenLastName', sqlalchemy.Unicode(40)),
    sqlalchemy.Column('Email', sqlalchemy.Unicode(80), nullable=False),
    sqlalchemy.Column('Address', sqlalchemy.Unicode(80), nullable=False),
    sqlalchemy.Column('City', sqlalchemy.Unicode(80), nullable=False),
    sqlalchemy.Column('State', sqlalchemy.String(2), nullable=False),
    sqlalchemy.Column('ZipCode', sqlalchemy.Integer, nullable=False),
    sqlalchemy.Column('DOB', sqlalchemy.Date(), nullable=False),
    sqlalchemy.Column('CreatedDate', sqlalchemy.Date,
default=datetime.now().date()),
    sqlalchemy.Column('CreatedTime', sqlalchemy.Time,
default=datetime.now().time())
)
```

- These lines will create an object **address_table** which has definitions of each column and will be used to create table in database.

Map Your Table To Python Class

Create a python class

- Now we map the database table to a python class.
- This basically means that our database will become a python class. We will be able to call its attributes and use it just like you would use normal classes and functions in python.
- Create an empty python class which will hold our data later. Capitalize the first character.

```
#Object Creation
#This is an empty class that will become our data class
class Addressbook(object):
    pass
```

Map python class

- Now map the empty class to database object.

```
#Mapping of Table to Object
addresstable_mapper=mapper(Addressbook, address_table)
```

- This actually connects the database table definition you created and empty python class.
- Now **Addressbook** knows about each column and you can reference it as `Addressbook.City` for example.
- It also knows what type it is and how to retrieve, insert, delete that record.
- Save and Exit.

Note: Sqlalchemy 0.3, assign_mapper has been depreciated and default is to use a mapper. If you are looking for assign_mapper functionality take a look at Elixir Project.

Create Tables in Database

- Final step, and the easiest one.
- Create database by issuing command :

```
tg-admin sql create
```

If you experience problems or errors go back to installation and make sure you have all the components. If that doesn't fix the error check your model.py

Controls

- Controls.py hold all your data transformations. Passing data from one thing to another, and converting it on the way there is done here.
- Since we defined the data structure in our model we can now use it to query database.
- We will pass the results to a kid template that will display the data for us. `template.addresses` will do just that.
- To create a list of addresses add this to controls.py

```
@expose(template="myfirstproject.templates.addresses")
def addresses(self):
    from model import Addressbook
    x=Addressbook.select()
    return dict(addresses=x)
```

- `@expose(template="myfirstproject.templates.addresses")` -- tells which template to pass the results of a function
- `def addresses(self):` -- tells what the URL for the function be. `localhost:8080/addresses`
- `return dict(addresses=x)` --tells what to return. `addresses.kid` will have addresses object available to it to display.

Template

- **templates/address.kid** is where all your data is displayed as xhtml. You tell it how to display it by modifying the template.
- Copy the welcome.kid to addresses.kid and replace everything in body tag with:

```
<body>
<ul>
  <li py:for="record in addresses">
    <a py:content="record.FirstName">first name</a>
    <a py:content="record.LastName">last name</a>
  </li>
</ul>
</body>
```

[addresses is available to the template because the function addresses in controls.py return it.]

Visit: <http://localhost:8080/addresses>

We now finished with displaying addresses. Now its time to create a form so we can insert them.

Save Form

- First we create a controller that will save our address book.
- Make these changes in controll.py
- On top line where you see:

```
from turbogears import controllers, expose
```

add redirect at the end of list.

```
from turbogears import controllers, expose, redirect
```

- Then below root class add this function

```
@expose()
def saveaddressbook(self, first_name, last_name):
    from model import AddressBook
    first_name=first_name
    last_name=last_name
    AddressBook(first_name=first_name, last_name=last_name)
    raise redirect("/addresses")
```

This function expects the first and last name to be passed into it. You can reach this function at localhost:8080/saveaddressbook but it will give you an error if you don't pass first and last name. It will save names using the Addressbook Class from the model. At the end it will redirect to localhost:8080/addresses

Edit form

- Now let's create actual form to input the names:
- First we need a controller, so add this into your controllers.py

```
@expose(template="myfirstproject.templates.form")
def addressbookform(self):
    return dict()
```

This will sent empty dictionary to template form, and will allow you to access localhost:8080/addressbookform

- Now inside templates lets' create form.kid
- Copy welcome.kid to form.kid and replace the whole body with this:

```
<body>
<form NAME="Add Addressbook" METHOD="post" ACTION="/saveaddressbook">
<p>First Name: <input name="first_name"></input></p>
<p>Last Name: <input name="last_name"></input></p>
<p><input type="submit" value="submit"></input></p>
</form>
</body>
```

- Now we are done. go to <http://localhost:8080/addressbookform> and you should see your form.
- After you submit it you should be redirected to <http://localhost:8080/addresses>
- If you experienced an error :

```
ExpatError: mismatched tag: line 13, column 2
```

You need to check your form.kid because it expects valid xhtml code. This error means that your template has some kind of xhtml syntax error.

TurboGears Forms

Widgets and Validators

- Widget is a collection of tools to build html forms automatically for you. Validators is a collection of tools to validate the input from the html form automaticly.

- We start from a beginning with **tg-admin quickstart**, our app will be called addressbook

Create a Form

- In your controller.py add these 4 modules to the import list, widgets, validators, error_handle, validate
- Your line should look similar to this.

```
from turbogears import controllers, expose, widgets, validators,
error_handler, validate
```

- Above root class, lets create class with our form fields.
- This class is what defines the each part of a html form and what needs to be validated when the form is filled.
- Label will be displayed next to the field. By default it is what the field name you gave it ex. **firstname**. If you want it to be changed the displayed name you add **label="First Name"** inside with the rest of the parameters.
- With validators such as money, number,etc empty fields will be still accepted. If you would like to force the field to **not be empty** you have to add(**not_empty=True**) to your definition. Look at syntax of **yourcarvalue** below.
- If you would like to use multiple validators you would use:
validators.all(validators.plaintext(),validators.minLength(15))
- Here are some sample fields and validators just to get you started. See the full list for all available widgets.

```
class AddressBookFields(widgets.WidgetsList):
    firstname = widgets.TextField(validator=validators.NotEmpty)
    description = widgets.TextArea(validator=validators.NotEmpty)
    homepage = widgets.TextField(validator=validators.URL)
    yourcarvalue = widgets.TextField(label="Car Value",
validator=validators.Money(not_empty=True))
    zipcode =
widgets.TextField(label="ZipCode", validator=validators.PostalCode)
```

- Now that you created your widget we will have to initiate it. We initiate it right below the class, otherwise instate it where you need it. We will reference our form by this name, **address_form**.

```
addressbook_form =
widgets.TableForm(fields=AddressBookFields(), submit_text="save address")
```

- You are done **with html form, html form validation aka. widget**.

Below you will find an example of choices you can make when creating widget:

```

class BigAddressBookFields(widgets.WidgetsList):
    firstname = widgets.TextField(validator=validators.NotEmpty)
    description = widgets.TextArea(validator=validators.NotEmpty)
    homepage = widgets.TextField(validator=validators.URL)
    yourcarvalue = widgets.TextField(label="Car Value",
validator=validators.Money(not_empty=True))
    zipcode =
widgets.TextField(label="ZipCode", validator=validators.PostalCode)
    age = widgets.TextField(label="Your Age", validator=validators.Int)
    sex =
widgets.SingleSelectField(label="Gender", validator=validators.NotEmpty,
options=['Female', 'Male'])
    maritalstatus = widgets.SingleSelectField(label="Marital Status",
validator=validators.NotEmpty, options=['Single', 'Married'])
    dob = widgets.CalendarDatePicker(format = '%Y/%m/%d', validator =
validators.DateTimeConverter(format="%Y/%m/%d"))

year=widgets.TextField(attrs={'size':4, 'maxlength':4}, validator=validators.All(val
searchin = widgets.RadioButtonList(label="Search
in", validator=validators.NotEmpty, options=['Option1', 'Option2']))
    searchfor = widgets.RadioButtonList(label="Search
For", validator=validators.NotEmpty, options=['Option1', 'Option2'], default='Option1'
    searchtype = widgets.RadioButtonList(label="Search
Type", validator=validators.NotEmpty,
options=[(1, 'Exact'), (2, 'Slow')], default=1)

```

- You can type widgets in a following format if it makes it easier to read for you:

```

myfield = widgets.TextField(
name='title', label='Title',
attrs={'size': 64, 'maxlength': 64},
validator=v.All(validators.NotEmpty, validators.UnicodeString)
)

```


Create URL

- Lets define a url where our form will be accessible from the internet.
- Again in controller.py Under a root class add this:

```

@expose(template="addressbook.templates.form")
def addressbook(self):
    submit_action = "/addressbookprocess"
    return dict(form=addressbook_form, action=submit_action)

```

- I. First line tells you which template will be used.
- II. Second line tells you what will be the url of this method. 
http://localhost:8080/addressbook
- III. Submit action tells where should the form input be sent.
- IV. Last line returns a dictionary 'form', which is equal to addressbook_form, and we pass an

action as well.

Create template

- Now let's create a form.kid in the template folder
- In Template folder copy welcome.kid and create form.kid
- Replace the <body> with this:

```
<body>
${form(action=action)}
</body>
```

Process the form

- Our form will be submitted to /addressbookprocess. In controller.py we need to create that url. Again Under a root class add this function.

```
@expose()
@error_handler(addressbook)
@validate(form=addressbook_form)
def addressbookprocess(self, **kwargs):
    return dict(kwargs=kwargs)
```

- Error handler will catch the error and if one exists it will pass it back to where it came from. In our case addressbook.
- Validate will validate the form.
- Addressbookprocess will process the form. We will save the content later, for now we pass values to our function via **kwargs (key word arguments)
- FYI. *name **syntax gets all the parameters and puts them in list. You can reference the list as *name[0] for first argument, etc.**
- Now if you wanted to save your addressbook follow the same process that we discussed in previous example. Look at the Database model paragraph and save form paragraph.

Widgets options

- To find options for widgets. Start python, and type

```
import turbogears
```

- Now type

```
dir(turbogears.widgets)
```

- You will get a list of available functions. You should see.

```
[ 'AjaxGrid', 'AutoCompleteField', 'Button', 'CSSLink', 'CSSSource',
  'CalendarDat
ePicker', 'CalendarDateTimePicker', 'CalendarLangFileLink', 'CheckBox',
  'CheckBo
xList', 'CompoundFormField', 'CompoundInputWidget', 'CompoundWidget',
  'DataGrid'
, 'FieldSet', 'FileField', 'Form', 'FormField', 'FormFieldsContainer',
  'HiddenFi
eld', 'ImageButton', 'InputWidget', 'JSLink', 'JSSource', 'JumpMenu',
  'Label', '
Link', 'LinkRemoteFunction', 'ListForm', 'LocalizableJSLink',
  'MultipleSelectFie
ld', 'PaginateDataGrid', 'PasswordField', 'RPC', 'RadioButtonList',
  'RemoteForm'
, 'RepeatingFieldSet', 'RepeatingFormField', 'RepeatingInputWidget',
  'ResetButto
n', 'Resource', 'SelectionField', 'SingleSelectField', 'Source',
  'SubmitButton',
  'SyntaxHighlighter', 'Tabber', 'TableForm', 'TextArea', 'TextField',
  'URLLink',
  'Widget', 'WidgetDescription', 'WidgetsList', '__builtins__', '__doc__',
  '__fil
e__', '__name__', '__path__', 'all_widgets', 'base', 'big_widgets',
  'datagrid',
  'forms', 'i18n', 'js_location', 'links', 'load_widgets', 'meta',
  'mochikit', 're
gister_static_directory', 'rpc', 'set_with_self', 'static']
```

- This means you could for example work with datagrid. (🌍)
<http://www.checkandshare.com/blog/?p=24>

Validators options

- We follow similar step with validator. After opening python

```
import turbogears
dir(turbogears.validators)
```

- You should see.

```
[ 'All', 'Any', 'Bool', 'ConfirmType', 'Constant', 'CreditCardExpires',
  'CreditCa
rdSecurityCode', 'CreditCardValidator', 'DateConverter',
  'DateTimeConverter', 'D
ateValidator', 'DictConverter', 'Email', 'Empty', 'FancyValidator',
  'FieldStorag
eUploadConverter', 'FieldsMatch', 'FileUploadKeeper', 'ForEach',
  'FormValidator'
, 'IndexListConverter', 'Int', 'Invalid', 'JSONValidator', 'MaxLength',
  'MinLeng
```

```
th', 'Money', 'MultipleSelection', 'NoDefault', 'NotEmpty', 'Number',
'OneOf', '
PhoneNumber', 'PlainText', 'PostalCode', 'Regex', 'RequireIfMissing',
'RequireIf
Present', 'Schema', 'Set', 'SignedString', 'StateProvince', 'String',
'StringBoo
l', 'StringBoolean', 'StripField', 'TgFancyValidator', 'TimeConverter',
'URL', '
UnicodeString', 'Validator', 'Wrapper', '_', '__builtin__',
'__builtins__', '_d
oc_', '__file__', '__name__', '_findall', '_illegal_s', 'cgi',
'datetime', 'for
mat', 'jsonify', 'pkg_resources', 're', 'simplejson',
'strftime_before1900', 'ti
me', 'turbogears', 'util', 'validators', 'warnings']
```

Password validators

- This would mostly be used to see if the password in two fields are the same.

```
class PasswordFields(widgets.WidgetsDecalration):
    passwd = widgets.PasswordField(validators=validators.NotEmpty())
    passwd2 =
widgets.PasswordField(validators=validators.UnicodeString())
class PasswordSchema(fromencode.schema.Schema):
    chained_validators=[validators.FieldsMatch('passwd', 'passwd2')]

form = TableForm(fields=PasswordFields(), validator=PasswordSchema)
```

If you are building a registration forms you might want to try: [Registration for TG](#). It is a set of tools to help you register users.

Validate Widget

- Make sure your controller now imports proper modules from turbogears

```
from turbogears import redirect, widgets, validators, error_handler,
validate, flash
```

- Define your widget definition class and create object you will call later

```
class AddressBookFields(widgets.WidgetsList):
    firstname = widgets.TextField(validator=validators.NotEmpty)
    description = widgets.TextArea(validator=validators.NotEmpty)
    homepage = widgets.TextField(validator=validators.URL)

# Create object and set submit button text
address_form =
widgets.TableForm(fields=AddressBookFields(), submit_text="save address")
```

- Create controller function for your widget

```
@expose(template="addressbook.templates.addressupdate")
def addressupdate(self):
    submit_action = "/addresssave"
    return dict(form=address_form, action=submit_action)
```

- Add widget form to the template

```
<html>
<body>
<p> ${form(action=action)} </p>
</body>
</html>
```

- Handle error, validate and process, and inform about results from the widget

```
#@expose(template="addressbook.templates.addresssave")
@expose()
@error_handler(addressupdate)
@validate(form=address_form)
def addresssave(self, **kwargs):
    flash('Addressbook Information are Saved')
    #You can save the information stored in dictionary kwargs to your
    database, process data, etc
    # return dict(kwargs=kwargs) or raise redirect("/index") or
    return other information.
    return dict(kwargs=kwargs)
```

TurboGears display Widgets

DataGrid

- DataGrid allows you to display data in a grid like way.
- DataGrid requires you to describe what you're going to display.
- [TG Docs on DataGrid](#)
- DataGrid's template is very simple.

DataGrid and SQLAlchemy Example

- DataGrid displays our data in a Grid Like table. It calls attributes of a class like this: myclass.FirstName, myclass.LastName. You get this type of structure when you deal with sqlalchemy and assign_mapper, so I assume you want to display some data from sqlalchemy here.
- We first need to create a data grid description. DataGrid will use it to display what we

want in order we want it to display.

```
from turbogears.widgets import DataGrid

myusers_datagrid = DataGrid(fields=[
    ('DisplayedColumnName1', 'LastName'),
    ('DisplayedColumnName2', 'FirstName'),
    ('DisplayedColumnName3', 'PhoneNumber'),
])
```

- Each tuple ('DisplayedColumnName1', 'LastName'), defines single column in the table. DisplayedColumnName1 **parameter defines how your whole column will be called. The LastName is an attributes' name of the class you will be trying to get the data out of. Ex. myclass.LastName.**
- So now we need to pass a list of the data we want to display. I already have a sqlalchemy table mapped to Users **using assing_mapper.**

```
mydatagrid=myusers_datagrid.display(Users)
retrun dict(mydatagrid=mydatagrid)
```

- In your template just add this where you want to display your data:

```
<span py:content="mydatagrid">datagrid will appear here</span>
```

- You are done. Wasn't that easy?

DataGrid Example with static content

- Let say you want to display something on the bottom of your page in a datagrid like manner.
- Add this in your model.py file

```
from turbogears.widgets import DataGrid
```

- In Model.py we will add our data model. For now we just need to display some semi-static data in a grid so we will handcode the data fields and some values.

```
mycustom_widget = DataGrid(fields=[
    ('NumberOfPayments', lambda row=row[0]),
    ('Due Each Month', lambda row=row[1]),
])

mydefaults=[(6,100), (2,300)]
```

- This initiates a datagrid instance that has 2 columns: numberofpayments and due each month.

- We create a list of some numbers that we will passed into this instance. In order to retrieve the values from the list we use lambda which acts like a anonymous function. If you are not familiar to lambda this is how it would look in normal python way: `def somefunction(row): return row[0]`
- In controller.py under your function you want to display the data add this:

```
from model import *
mydatagrid=mycustome_widget.display(mydefaults)
return dict(mydatagrid=mydatagrid)
```

- We pass our list of defaults to display() function of our widget.
- We now need to return mydatagrid to our template.
- Now in your template add this anywhere you want to display the data.:

```
<span py:content="mydatagrid">datagrid will appear here</span>
```

FastDataGrid

- FastDataGrid is sophisticated enough to figure out how to display an arbitrary results from a select statment instance that was done when you do .select() on you SQLAlchemy's model object.
- FastDataGrid is a seperate module for Turbogears and it needs to be downloaded from turbogears website.

```
sudo easy_install -f http://turbogears.org/download/ TGFastData
```

- <http://turbogears.org/widgets/tutorials/DataGridWidget.html>

SQLObject (SO) quick Overview

- Lets start the project

```
tg-admin quickstart
```

dev.conf -database connection

- We have database done already and it is in mysql
- In dev.cfg we connect to our database mysql connection

```
sqlobject.dburi="mysql://username:password@localhost:3306/databasename"
```

model.py -database model

- In model.py we tell sql objects to use our existing database. We also tell it what is the primary key.
- sqlmeta class is a class that holds columns, columnLists, columnDefinitions, joins, indexes, joinDefinitions, indexDefinitions. Information in these are used to properly connect and use database.

```
#Create database object from a database
class mainapplication(SQLObject):
    _idName = "App_Sid"
    class sqlmeta:
        fromDatabase = True
```

- `_idName` tells sql objects that our primary key is not called id **but it is called App_Sid short for Application system ID.**
- `fromDatabase` tell sql object that we should use definitions of column names from a database.

sqlobject features

Design

- **When you design a database tables try to always follow Third Normal Form 3NF. It will make your app much easier to maintain and expend.**
- http://en.wikipedia.org/wiki/Third_normal_form

Naming convention:

- **Change naming style of tables and columns by setting a style attribute:**
- `MixedCaseUnderscoreStyle`
- `MixedCaseStyle`
- Example:

```
class Addressbook(SQLObject):
    class sqlmeta:
        style=MixedCaseStyle()
```

- Also `style=MixedCaseStyle(longOD=True)` is a possibility.

Lazy Update and Caching of connection object

```
cacheValues - sets the number of cache values
lazyUpdate - will waits with an update until sync command is issued
```

```
sync() -sync with database
syncUpdate() -syncs and updates database
```

sql attributes

- **You use these for creating functions that can be accessed from multiple places.(ex. calculate total fee)**
- You can use these prefixes to crate your attributes: `_del_`, `_doc_`, `_get_`, `_set_`,

```
def _get_totalfee(self):
    return eval(self.fees)
```

- Above function will create an attribute `_totalfee`

Transactions

- **Transactions allows you to control the saving of the data in 2 steps all success or all failed.**
- Change database connection.
- If 'conn' is your database connection. You do:

```
#This creates transaction object
trans = conn.transaction()
#set connection for your table
for x in mySqlClass:
    x._connection=trans
```

- Use transaction to change one database and insert into another

```
def sell(item):
    try:
        books=BookStore.selectBy(book=item)[0]
        books.quantity-= 1
        SoldBooks(item=item)
        trans.commit()
    except Exception, e:
        trans.rollback()
        trans.begin()
```

- If you wouldn't use transactions you could end up with book being subtracted from a quantity and not added to SoldBooks. Transaction will prevent that error happening.

SQLObject save to database

Example1

```
class PhoneBook (SQLObject) :
    first_name=StringCol (alternateID=True, length=30)
    last_name=StringCol ()
    home_page=StringCol (length=255)
```

#Simple save.

```
PhoneBook(first_name="John",last_name="Smith",home_page="www.example.com") }}
```

Install Turbogears with Apache

Egg deploy

[Instructions for mod_proxy and Turbogears were tested and are working](#)

Create an Egg File

- To create an egg file that you can transfer to other system you can do.

```
cd /projects/foo
python setup.py bdist_egg
```

- The file shows up in:

```
/projects/foo/dist/foo-1.0-py2.4.egg
```

dev.cfg app.cfg prod.cfg

- Turbo gears has 3 configuration files. dev.cfg, prod.cfg, app.cfg

```
dev.cfg -Has location configuration of a development database
prod.cfg -Has location configuration of a production database
app.cfg -Has custom configuration that your might need for your project
app, such as template, engine, encodings, etc.
```

- Location:

```
app.cfg is located in PROJECTNAME/config/app.cfg
dev.cfg is located in PROJECTNAME.dev.cfg
prod.cfg has a sample file in PROJECTNAME/sample-prod.cfg which you need
to modify for production environment.
```

Deploy an EGG file

- After you copy the selected egg that you have just build, we will install it on production server.
- Copy the prod.cfg to:

```
/var/www/foo/prod.cfg
```

- Edit it.
- Add the proper sql and server settings based on production server setup.
- Then copy the egg file into the eggs folder and install it.

```
cd /var/www/foo/eggs
easy_install foo-1.0-py2.4.egg
```

- [OPTIONAL]If this is different machine you will need to create a database. To create users on mysql do something like this:

```
cd /var/www/foo
mysqladmin -h localhost -u root -p create foodb
tg-admin sql create --egg foo
```

Configure TurboGears prod.cfg

- In prod.cfg add

```
server.socket_port=8080
```

- and uncomment

```
base_url_filter.on = True
base_url_filter.use_x_forwarded_host = True
```

Run the app

```
# run the server
/usr/bin/foo-start.py /var/www/foo/prod.cfg
```

- If you want the server to run all the time

```
/usr/bin/foo-start.py /var/www/foo/prod.cfg &
```

- To kill it you would have to do `ps -A|grep foo`, **look for id and then** kill 1234

Mod_Proxy

Enable mod proxy

- Now that you got the your app installed, lets enable it to work with apache2 and turbogears

```
domain: www.example.com
Turbogears URL: http://www.example.com/webapp/
Turbogears working on port 8080
Configuration File: prod.cfg
```

- On Debian you need to install apache and modproxy

```
aptitude update
aptitude install apache2
aptitude install libapache2-mod-proxy-html
```

- Enable modules:

```
a2enmod proxy
a2enmod proxy_http
```

Configure apache

- Add a file like this:

```
/etc/apache2/conf.d/webapp
```

- Inside add these lines:

```
ProxyPreserveHost on
<Proxy *>
  Order allow,deny
  Allow from all
</Proxy>
ProxyPass /webapp/ http://127.0.0.1:8080/webapp/
ProxyPassReverse /webapp/ http://127.0.0.1:8080/webapp/
ProxyPass /static/ http://127.0.0.1:8080/static/
ProxyPassReverse /static/ http://127.0.0.1:8080/static/
```

- Then:

```
/etc/init.d/apache2 force-reload
```

Done.

Auto start of the app

- For now you can try adding this to the root cron tab. This will ensure your web application start at boot time.

```
@reboot /usr/bin/python /usr/bin/start-webapp.py
/usr/local/bin/webapp/prod.cfg > /var/log/webapp.log &
```

Below instructions did not work for me. I will leave them for reference. Check the `mod_wsgi` instructions.

- To auto start the app if it failes create an auto start script in `cgi-bin`
- `/usr/lib/cgi-bin/webapp-autostart.cgi`
- `chmod 711 /usr/lib/cgi-bin/webapp-autostart.cgi`
- Inside the file add these lines:

```
#!/usr/bin/python
print "Content-type: text/html\r\n"
print """<html><head><META HTTP-EQUIV="Refresh" CONTENT="4;
URL=/webapp/"></head><body>Restarting site ...<a href="/webapp/">click
here<a></body></html>"""
import os
import sys
os.setpgid(os.getpid(), 0)
os.system(sys.executable + ' /usr/bin/start-webapp.py
/usr/local/bin/prod.cfg &')
```

- Replace `webapp` with your app's name and replace `'usr/local/bin/prod.cfg'` with a location of your `prod.cfg` file
- Now configure apache by adding these lines to the file we initially configured: `/etc/apache2/conf.d/webapp`

```
<Location /recall>
# Adjust these lines and uncomment to restart the server if it isn't
already running
ErrorDocument 503 /cgi-bin/webapp-autostart.cgi
ErrorDocument 502 /cgi-bin/webapp-autostart.cgi
</Location>
```

Server Log

What about a server log? How can I tell it to use `/var/log/apache2/webapp.log` How can I tell it to rotate the same way apache does it?

apache and `mod_wsgi`

- <http://code.google.com/p/modwsgi/wiki/IntegrationWithTurboGears>

Instructions for `mod_wsgi` and `Turbogears` were tested and are working

Install mod_wsgi

- Check if your version has this package available:

```
aptitude install libapache2-mod-wsgi
```

- If this worked go to the next section. If not continue reading.
- If not, Download mod_wsgi 1.3 [download](#)
- Untar it.

```
tar -xzvf mod_wsgi-1.3.tar.gz
```

- Install python dev ,and apache dev modules so we can compile the program.(apache assumes threaded here)

```
aptitude update
aptitude install python-dev
aptitude install apache2-dev
aptitude install gcc
```

(A number of Python binary packages for Linux systems are not compiled with shared library support enabled.How can I tell?)

- Now Configure your mod_wsgi

```
cd mod_wsgi-1.3
./configure
make
make install
make clean
```

- Now let's enable this module in apache configuration files.
- Create apache load file:

```
vi /etc/apache2/mods-available/mod_wsgi.load
```

- Add this line:

```
LoadModule wsgi_module /usr/lib/apache2/modules/mod_wsgi.so
```

- Now enable mod_wsgi

```
a2enmod mod_wsgi
/etc/init.d/apache2 force-reload
```

Configure your TurboGears app for mod_wsgi

1. My application will be called 'myfirstapp'. Replace it with your application name.
2. We will keep our project in /usr/local/turbogears/myfirstapp. You would copy your tg app to /usr/local/turbogears/.
3. File structure will look like this:

```
File Structure for your app:
/usr/local/turbogears/
-----myfirstapp/ (all files from tg app)
-----myfirstapp/myfirstapp/ (main configuration files, controller.py
and model.py etc)
-----myfirstapp/myfirstapp/static/ (all static files, css,images)
-----myfirstapp/prod.cfg
-----myfirstapp/apache/myfirstapp.wsgi
-----myfirstapp/server.log
and
/etc/apache2/conf.d/myfirstapp
```

- Lets configure our application.
- Create folder called turbogears in /usr/local/

```
mkdir /usr/local/turbogears
```

- Copy your turbogears app inside that folder

```
cp /home/lucas/project/myfirstapp /usr/local/turbogears
or
scp -r /home/lucas/projects/myfirstapp
username@example.com/usr/local/turbogears/
```

- Go inside myfirstapp folder you just created and create folder called apache:

```
mkdir /usr/local/turbogears/myfirstapp/apache
```

- Inside apache folder you created, create a file called myfirstapp.wsgi

```
vi /usr/local/turbogears/myfirstapp/apache/myfirstapp.wsgi
```

- Inside myfirstapp.wsgi put these lines:

```
import sys
sys.path.append('/usr/local/turbogears/myfirstapp')
sys.stdout = sys.stderr

import os
#os.environ['PYTHON_EGG_CACHE'] = '/usr/local/turbogears/myfirstapp'
```

```

import atexit
import cherrypy
import cherrypy._cpwsgi
import turbogears

turbogears.update_config(configfile="/usr/local/turbogears/myfirstapp/prod.cfg",
modulename="myfirstapp.config")
turbogears.config.update({'global': {'server.environment':
'production'}})
turbogears.config.update({'global': {'autoreload.on': False}})
turbogears.config.update({'global': {'server.log_to_screen': False}})

#For non root mounted app:
turbogears.config.update({'global': {'server.webpath': '/myfirstapp'}})

import myfirstapp.controllers

cherrypy.root = myfirstapp.controllers.Root()

if cherrypy.server.state == 0:
    atexit.register(cherrypy.server.stop)
    cherrypy.server.start(init_only=True, server_class=None)

#For root mounted app
#application = cherrypy._cpwsgi.wsgiAppi

#For none-root mounted app
def application(environ, start_response):
    environ['SCRIPT_NAME'] = ''
    return cherrypy._cpwsgi.wsgiApp(environ, start_response)

```

- Now inside /usr/local/turbogears/myfirstapp you should have prod.cfg. If you don't copy sample-prod.cfg to prod.cfg and add all necessary configuration inside of it (database, etc)
- Don't forget to issue tg-admin sql create **if you need to create some databases.**
- Inside prod.cfg there is a line that says: args="('server.log',)"
- If you want to enable this log then change it to **args="('/usr/local/turbogears/myfirstapp/server.log',)"**
- If you want to disable it since you have apache logs you have to comment out two lines(args and class):

```

[[[access_out]]]
# set the filename as the first argument below
#args="('server.log',)"
#class='FileHandler'
level='INFO'
formatter='message_only'

```

- If you are using mysql you will need to add a pool_recycle to prod.cfg
- Right below your mysql connection string 'sqlalchemy.dburi=...' add:

```
sqlalchemy.pool_recycle = 3600
```

- Make sure your app has the same permissions as the apache server:

```
chown -R www-data:www-data /usr/local/turbogears/myfirstapp
```

- Now tell apache about your app.
- Create a file:

```
vi /etc/apache2/conf.d/myfirstapp
```

- Add these lines in it:

```
Alias /static /usr/local/turbogears/myfirstapp/myfirstapp/static

WSGIScriptAlias /myfirstapp
/usr/local/turbogears/myfirstapp/apache/myfirstapp.wsgi

<Directory /usr/local/turbogears/myfirstapp/apache>
Order deny,allow
Allow from all
</Directory>
```

- When done restart apache

```
/etc/init.d/apache2 restart
```

- You are done. Now go to your website 'www.example.com/myfirstapp'
- [Warning] If you are converting from proxy style hosting of your app make sure you remove old app from python. If you don't then the app in site-package will be run instead of the one /usr/local/turbogears. Use following command to do it.

```
easy_install -m myfirstapp
```

advance mod_wsgi memory management

ProcessesAndThreading

- Find out what apache version you are running.

```
/usr/sbin/apache2 -V
```

- You should see:

```
Server MPM:      Prefork
threaded:       no
```

```
forked:      yes (variable process count)
```

1. apache prefork MPM = no multi threaded turbogears application
2. apache worker MPM = multi threaded turbogears application

Overall, use of 'worker' MPM will result in less child processes needing to be created, but resource usage of individual child processes will be greater. On modern computer systems, the 'worker' MPM would in general be the preferred MPM to use and should if possible be used in preference to the 'prefork' MPM.

- So if you are a huge site you use worker if you are a small-medium site you use default prefork.

You are left with a choice of embeded or daemon mode for your wsgi application

- "Embedded mode is the best mode for very large sites as it is able to harness the feature of Apache to scale up to meet demand by creating more child processes to handle requests and reap those processes when no longer required."
- "When using 'daemon' mode of mod_wsgi, each process group can be individually configured so as to run in a manner similar to either 'prefork', 'worker' MPMs for Apache. This is achieved by controlling the number of processes and threads within each process using the 'processes' and 'threads' options of the WSGIDaemonProcess directive."

Unlike the normal Apache child processes when 'embedded' mode of mod_wsgi is used, the configuration as to the number of daemon processes within a process group is fixed with the settings you provide. That is, when the server experiences additional load, no more daemon processes are created than what is defined. You should therefore always plan ahead and make sure the number of processes and threads defined is adequate to cope with the expected load.

So now the question is what is max your system can cope with and what we should start our processes and thread count.

- To run in daemon mode change the apache settings from:

```
Alias /static /usr/local/turbogears/myfirstapp/myfirstapp/static

WSGIScriptAlias /myfirstapp
/usr/local/turbogears/myfirstapp/apache/myfirstapp.wsgi

<Directory /usr/local/turbogears/myfirstapp/apache>
Order deny,allow
Allow from all
</Directory>
```

to

```
Alias /static /usr/local/turbogears/myfirstapp/myfirstapp/static

WSGIDaemonProcess myfirstapp threads=10 processes=3
WSGIProcessGroup myfirstapp

WSGIScriptAlias /myfirstapp
/usr/local/turbogears/myfirstapp/apache/myfirstapp.wsgi

<Directory /usr/local/turbogears/myfirstapp/apache>
Order deny,allow
Allow from all
</Directory>
```

Stress test your app

- If you would like to stress test you app:
- We will use a program <http://httpd.apache.org/docs/2.0/programs/ab.html> It should already be installed

```
/usr/sbin/ab -n 1000 -w http://localhost/myfirstapp/ >> test.htm
12544 www-data 25 0 129m 21m 4604 S 33.3 4.3 0:12.08 apache2
12585 www-data 25 0 129m 21m 4604 S 32.6 4.3 0:10.75 apache2
12564 www-data 25 0 129m 21m 4604 S 33.3 4.3 0:12.08 apache2
```

then try with:

```
/usr/sbin/ab -n 10000 -w http://localhost/myfirstapp/ >> test2.htm
```

- You should see results like:

```
Requests per second: 10.83
Transfer rate: 67.04 kb/s received
Connection Times (ms)
      min      avg      max
Connect:    0         0         0
Processing: 87        91       214
Total:     87        91       214
```

mod_python

Below instructions did not work for me. I'll leave it for reference. Check the [mod_wsgi instructions](#)

- Did not work for me but If i did for you please update appropriate section.

Based on: http://docs.turbogears.org/1.0/mod_python * If you don't have apache2, mod_python, you will need to install it.

```
aptitude update
aptitude install apache2
aptitude install libapache2-mod-python
```

- Download a modpython_gateway from: http://projects.amor.org/misc/svn/modpython_gateway.py and put it in `/usr/lib/python2.4/site-package/`

```
mv /home/lucas/Desktop/modpython_gateway.py
/usr/lib/python2.4/site-package/
```

- Change owner if necessary

```
chown root:root /usr/lib/python2.4/site-package/modpython_gateway.py
```

Install project file

- Now create a file called PROJECTNAME_modpython.py and put it inside the same site-package folder. In my case i would create a file called myfirstproject_modpython.py
- Inside of that file place this text. Change to reflect your project name

```
import pkg_resources
pkg_resources.require("TurboGears")

import cherrypy
import turbogears

turbogears.update_config(modulename="myifirstproject.config")
turbogears.update_config(configfile="/home/lucas/web/myfirstproject/myfirstproject

from myfirstproject.controllers import Root

cherrypy.root = Root()
cherrypy.server.start(initOnly=True, serverClass=None)

def fixuphandler(req):
    return 0
```

- Now inside your folder directory issue a command:

```
python setup.py install
```

If you get an error like this. You will need to login as root `su root` or use `--`:

```
Perhaps your account does not have write access to this directory?
```

You might have to use different installation directory by specifying `--install-dir` **in the**

command.

Test project file

- To see if it was installed type:

```
cd
python
```

inside of python type import PROJECTNAME. **So in my case i would do:**

```
import myfirstproject
```

Configure apache2 with your turbogears project

- Create a file with your project name in /etc/apache2/sites-enabled/. In my case i would do:

```
vi /etc/apache2/sites-enabled/myfirstproject
```

- And insert this text but replace project name with your project

```
<Location /projectname>
    SetHandler python-program
    PythonHandler modpython_gateway::handler
    PythonOption wsgi.application cherrypy._cpwsgi::wsgiApp
    PythonFixupHandler projectname_modpython
    PythonDebug on
</Location>
```

- Restart apache

```
/etc/init.d/apache2 restart
```

- Check out your website at: <http://localhost/myfirstproject/>

Error

```
File "/var/lib/python-support/python2.4/turbogears/config.py", line 98,
in _get_loggers
    raise ConfigError("Logger %s references unknown "
ConfigError: Logger access references unknown handler access_out
```

Done simple, Go Advance

- This should make your realize how fast and easy you can create your turbogears app, or how easy you can extend your desktop application.

Create a project (LEVEL II)

Introduction

- Extend your existing application to web
- Use existing database
- Create forms
- Validate forms
- Create process
- Save data
- Verify data
- Confirm final submission

SqlAlchemy (SA)

- You can find full documentation on sqlalchemy at their website, but this section will get you up to speed.
- If you need to use sqlalchemy on its own try this [sqlalchemy tutorial](#)
- Start new project with sqlalchemy prefix.

```
tg-admin quickstart --sqlalchemy
```

model.py Connection, and setup to existing database

- We will use existing database!

dev.conf and database connection

- We have database done already and it is in mysql
- In dev.cfg we connect to our database mysql connection

```
sqlalchemy.dburi="mysql://username:password@localhost:3306/databasename"
```

- If you start from scratch refer to http://www.sqlalchemy.org/docs/03/tutorial.html#tutorial_schemasql_table_creating for information on how to create your own database fields here.

what needs to be imported

- Modify the import lines in model.py **so that they look like:**

```
from turbogears.database import metadata, session, bind_meta_data
from sqlalchemy.ext.associationmapper import association_mapper
import sqlalchemy
```

bind to database

- Now bind the connection to your database in model.py. This function will bind to the connection you have setup at dev.conf

```
bind_meta_data()
```

create tables

- Create an instance for your first table.
- Add this to your model.py **file. Replace yourtablename with the actual table name from your database.**

```
#Creating table sctructure from our database.
yourtablename_table = sqlalchemy.Table('yourtablename', metadata,
autoload=True)
```

- This line will create an object which will store your definitions of table.

create python data class

- Now we map the database table to a python class that we will use to manipulate our data. This basically means that our database will become a python class that we can call its attributes etc.
- Create an empty python class which will hold our data later. Capitalize the first character.
- Add this to model.py

```
#Object Creation
#This is an empty class that will become our data class
class Yourtablename(object):
    pass
```

map database to python class

- Map the empty class to database object. This actually connects the database to our class that we just created. It tells the class what fields, attributes, functions it allows.

```
#Mapping of Table to Object
yourtablename_mapper=assign_mapper(session.context,Yourtablename,yourtablename_tab
```

- You are done with model.py
- Your final model.py should look something like this. In this case I am using table called binder

```
from turbogears.database import metadata, session, bind_meta_data
from sqlalchemy.ext.assignmapper import assign_mapper
import sqlalchemy

#Binding sqlalchemy to a turbogears database
bind_meta_data()

#[Mysql option]Fixed in next release. Table creation, ticket 482
#from turbogears import database
#engine=database.get_engine()
#tables = []
#for name in engine.execute("SHOW TABLES"):
#    tables[name] = sqlalchemy.Table(name, metadata, autoload=True)

#Using manual table creation
binder_table = sqlalchemy.Table('binder', metadata, autoload=True)
bdriver_table = sqlalchemy.Table('bdriver', metadata, autoload=True)
bcoverage_table = sqlalchemy.Table('bcoverage', metadata, autoload=True)
bvehicle_table = sqlalchemy.Table('bvehicle', metadata, autoload=True)
bviolation_table = sqlalchemy.Table('bviolation', metadata,
autoload=True)
bdiscschg_table = sqlalchemy.Table('bdiscschg', metadata, autoload=True)

#Object Creation
#This is an empty class that will become our data class
class Binder(object):
    pass
class bdriver(object):
    pass
class bcoverage(object):
    pass
class bvehicle(object):
    pass
class bviolation(object):
    pass
class bdiscschg(object):
    pass

#Mapping of Table to Object
```

```
bindermapper=assign_mapper(session.context,Binder,binder_table)
bdrivermapper=assign_mapper(session.context,bdriver,bdriver_table)
bvehiclemapper=assign_mapper(session.context,bvehicle,bvehicle_table)
bcoveragemapper=assign_mapper(session.context,bcoverage,bcoverage_table)
bviolationmapper=assign_mapper(session.context,bviolation,bviolation_table)
bdiscschmapper=assign_mapper(session.context,bdiscschg,bdiscschg_table)
```

When Done Don't forget to run this command in root directory of the project.

```
tg-admin sql create
```

create table in sqlalchemy

- Here are sqlalchemy data types: [Sqlalchemy Data Types](#)

Capitalized refers to a SQL standard type, whereas non-capitalized is a "generic" type that may resolve differently on different database backends.

```
module sqlalchemy.types

    * class BLOB(Binary)
    * class BOOLEAN(Boolean)
    * class Binary(TypeEngine)
    * class Boolean(TypeEngine)
    * class CHAR(String)
    * class CLOB(TEXT)
    * class DATE(Date)
    * class DATETIME(DateTime)
    * class DECIMAL(Numeric)
    * class Date(TypeEngine)
    * class DateTime(TypeEngine)
    * class FLOAT(Float)
    * class Float(Numeric)
    * class INT(Integer)
    * class Integer(TypeEngine)
    * class Interval(TypeDecorator)
    * class NCHAR(Unicode)
    * class NUMERIC(Numeric)
    * class Numeric(TypeEngine)
    * class PickleType(MutableType, TypeDecorator)
    * class SMALLINT(SmallInteger)
    * class SmallInteger(Integer)
    * class String(Concatenable, TypeEngine)
    * class TEXT(String)
    * class TIME(Time)
    * class TIMESTAMP(DateTime)
    * class Time(TypeEngine)
    * class TypeDecorator(AbstractType)
```

```
* class TypeEngine (AbstractType)
* class Unicode (String)
* class VARCHAR (String)
```

- Example Sqlalchemy Table

```
address_table = sqlalchemy.Table('address', metadata,
    sqlalchemy.Column('Address_Sid', sqlalchemy.Integer,
primary_key=True),
    sqlalchemy.Column('FirstName',
sqlalchemy.Unicode(40), nullable=False),
    sqlalchemy.Column('LastName', sqlalchemy.Unicode(40), nullable=False),
    sqlalchemy.Column('MaidenLastName', sqlalchemy.Unicode(40)),
    sqlalchemy.Column('Email', sqlalchemy.Unicode(80), nullable=False),
    sqlalchemy.Column('Address', sqlalchemy.Unicode(80), nullable=False),
    sqlalchemy.Column('City', sqlalchemy.Unicode(80), nullable=False),
    sqlalchemy.Column('State', sqlalchemy.String(2), nullable=False),
    sqlalchemy.Column('ZipCode', sqlalchemy.Integer, nullable=False),
    sqlalchemy.Column('DOB', sqlalchemy.Date(), nullable=False),
    sqlalchemy.Column('CreateDate', sqlalchemy.Date,
default=datetime.now().date()),
    sqlalchemy.Column('CreatedTime', sqlalchemy.Time,
default=datetime.now().time())
)
```

controller.py, Data passing and conversion

In controller you you to add this to import lines if they don't have it included:

```
from quote import model
```

Direct Sql statements to a database

- You can use direct sql statements to get the data you need.

```
from turbogears import database
engine=database.get_engine()
engine.execute("SHOW TABLES"):
```

Basic select()

- There are two ways you can get data from database via sqlalchemy, via Query or via Select. The "Query class should not be confused with the Select class, which defines database SELECT operations at the SQL (non-ORM) level. Query differs from Select in that it returns ORM-mapped objects and interacts with an ORM session, whereas the Select construct interacts directly with the database to return iterable result sets." 🌍 10

- The simplest form of select is:

```
s=Users.select()
```

- Here we select all from that table. aka select * from Users

equal, not equal, less then, more then

- To select partucular record

```
s1=Users.select(Users.c.LASTNAME=='Smith')
```

more or less

- Less or More

```
s1=Users.select(Users.c.AGE < 40)
```

and ,or ,not

- Use two fields And or or

```
s1=Users.select((Users.c.LASTNAME=='Smith') &
(Users.c.FIRSTNAME=='John'))
s1=Users.select((Users.c.LASTNAME=='Smith') |
(Users.c.FIRSTNAME=='John'))
```

- Or you want to select everybody except for Smith

```
s1=Users.select(~(Users.c.LASTNAME=='Smith'))
```

- As you can see we are using a Users.select() **function to get these values. As a parameter we are giving a combinations of Users.c.FIELDNAME**

like, endswith, startswith, between, in

- You can also use other ways to find your results, but this time we will use a build in functions on that FIELDNAME that we are using.(s1=Users.select(Users.c.FIELDNAME.SOMEFUNCTION) These include:

```
s1=Users.select(Users.c.AGE.between(18,21))
s1=Users.select(Users.c.LASTNAME.like('%a'))
s1=Users.select(Users.c.FaxNumber.like('%'+str(kwargs['FaxNumber'])+'%'))
s1=Users.select(Users.c.LASTNAME.endswith('th'))
```

```
s1=Users.select(Users.c.LASTNAME.startswith('Mr'))
s1=Users.select(Users.c.LASTNAME.in_('Smith','Johnson'))
```

- As you saw we used functions: like, endswith, startswith, between, in_ functions there are few more but you have to see for yourself how they work.
- More like examples:

```
s1=Users.select(Users.c.LASTNAME.like('%'+kwargs['LASTNAME']+'%'))
s1=Users.select(Users.c.LASTNAME.like('%'+str(kwargs['USERID'])+'%'))
```

Advanced select()

different select: sqlalchemy.select()

- Now that you have a handle on this select() function. There is another select function that is part of sqlalchemy, but that function needs to be executed.

```
import sqlalchemy
s2=sqlalchemy.select(Users.c.LASTNAME=='Smith')
s3=s2.execute()
```

- The actual going to a database happens when you issue a execute()
- If you don't want to select all fields you can tell select which fields you want. sqlalchemy.select().execute()

```
s1=sqlalchemy.select(
[Users.c.LAST,Users.c.FIRST,Users.c.Age],(Users.c.LASTNAME=='Smith')).execute()
```

group_by

- With this select you are able to group single fields and do a group by queries.
- To select a single column you do:

```
x=sqlalchemy.select([Users.c.AGE], Users.c.LASTNAME=='Smith')
x2=x.execute()
```

- To Select a single column and group it by some column you do:

```
x2=sqlalchemy.select([Users.c.AGE], group_by=[Users.c.AGE])
x2=x.execute()
```

- If you want to combine where and group by you do:

```
x2=sqlalchemy.select([Users.c.AGE],
Users.c.LASTNAME=='Smith',group_by=[Users.c.AGE])
x2=x.execute()
```

- If you are looking for a advanced group_by read: [advanced group by logic](#)

select_by

- Another select function os select_by in which you can specify multiple field names.

```
User.select_by(User_sid=100, last_name='Smith')
```

available functions of sqlalchemy

- [FYI]These are available functions and/or attributes of a python class. We have used some of these.

properties of python class and sql

- From this list we used select(),select_by(), flush()

```
dir(User)
['IDCOLUMN', 'COLUMN1',
'WHATEVER', 'THE', 'COLUMN', 'NAME', 'IS', 'IN', 'THE', 'DATABASE',
'__class__', '__delattr__', '__dict__', '__doc__', '__getattr__',
'__hash__', '__init__', '__module__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__setattr__', '__str__', '__weakref__',
'__state__',
'c', 'count', 'count_by', 'delete', 'expire', 'expunge', 'flush', 'get',
'get_by', 'join_to', 'join_via', 'mapper', 'merge', 'refresh', 'save',
'save_or_update', 'select', 'select_by', 'selectone', 'update']
```

properties of a field name

- These are propertis of fields. Fields like Users.c.LASTNAME
- We've used like,endswith,startswith,in_like.
- These are the rest of the function that you could choose from

```
>>> dir(Users.c.LASTNAME)
['_ColumnClause__label', '_ColumnElement__orig_set',
'_Column__originating_column', '_SchemaItem__case_sensitive', '__add__',
'__and__', '__class__', '__delattr__', '__dict__', '__div__', '__doc__',
'__eq__', '__ge__', '__getattr__', '__gt__', '__hash__', '__init__',
'__invert__', '__le__', '__lt__', '__mod__', '__module__', '__mul__',
'__ne__', '__new__', '__or__', '__reduce__', '__reduce_ex__', '__repr__',
'__setattr__', '__str__', '__sub__', '__truediv__', '__weakref__',
```

```
'_bind_param', '_case_sens', '_case_sensitive_setting', '_check_literal',
'_compare', '_compare_self', '_compare_type', '_derived_metadata',
'_determine_case_sensitive', '_find_engine', '_foreign_keys',
'_get_case_sensitive', '_get_engine', '_get_from_objects', '_get_label',
'_get_orig_set', '_get_parent', '_group_parenthesized', '_init_items',
'_is_oid', '_label', '_make_proxy', '_negate', '_one_fkey', '_operate',
'_primary_key', '_process_from_dict', '_selectable',
'_set_casing_strategy', '_set_orig_set', '_set_parent',
'accept_schema_visitor', 'accept_visitor', 'append_foreign_key', 'args',
'autoincrement', 'between', 'case_sensitive', 'columns', 'compare',
'compile', 'constraints', 'copy', 'copy_container', 'default',
'distinct', 'endswith', 'engine', 'execute', 'foreign_key',
'foreign_keys', 'get_engine', 'in_', 'index', 'key', 'label', 'like',
'metadata', 'name', 'nullable', 'onupdate', 'op', 'orig_set', 'parens',
'primary_key', 'quote', 'scalar', 'select', 'shares_lineage',
'startswith', 'table', 'to_selectable', 'type', 'unique']
```

list of available fields, proprties

- **TABLENAME.chas names of the columns.**
- For example you could

```
>>> print User.c.has_key('FIRSTNAME')
True
>>> print User.c.has_key('firstname')
False
```

- To get a column name in actual database, you could get it by doing:

```
>>> print User.c.FIRSTNAME
User.FIRSTNAME
```

- Where **User is a table name and FIRSTNAME is a column name**
- These are available properties that you could use:

```
dir(User.c)
['_OrderedProperties__data', '__add__', '__class__', '__contains__',
'__delattr__', '__delitem__', '__dict__', '__doc__', '__getattr__',
'__getattribute__', '__getitem__', '__hash__', '__init__', '__iter__',
'__len__', '__module__', '__new__', '__reduce__', '__reduce_ex__',
'__repr__', '__setattr__', '__setitem__', '__str__', '__weakref__',
'_get_data', 'clear', 'get', 'has_key', 'keys']
```

working with columns

- You can work with single column by using these attributes:

```
>>> dir(User.c.FIRSTNAME)
['_ColumnClause__label', '_ColumnElement__orig_set',
'_Column__originating_column', '_SchemaItem__case_sensitive', '__add__',
 '__and__', '__class__', '__delattr__', '__dict__', '__div__', '__doc__',
 '__eq__', '__ge__', '__getattr__', '__gt__', '__hash__', '__init__',
 '__invert__', '__le__', '__lt__', '__mod__', '__module__', '__mul__',
 '__ne__', '__new__', '__or__', '__reduce__', '__reduce_ex__', '__repr__',
 '__setattr__', '__str__', '__sub__', '__truediv__', '__weakref__',
'_bind_param', '_case_sens', '_case_sensitive_setting', '_check_literal',
'_compare', '_compare_self', '_compare_type', '_derived_metadata',
'_determine_case_sensitive', '_find_engine', '_foreign_keys',
'_get_case_sensitive', '_get_engine', '_get_from_objects', '_get_label',
'_get_orig_set', '_get_parent', '_group_parenthesized', '_init_items',
'_is_oid', '_label', '_make_proxy', '_negate', '_one_fkey', '_operate',
'_primary_key', '_process_from_dict', '_selectable',
'_set_casing_strategy', '_set_orig_set', '_set_parent',
'accept_schema_visitor', 'accept_visitor', 'append_foreign_key', 'args',
'autoincrement', 'between', 'case_sensitive', 'columns', 'compare',
'compile', 'constraints', 'copy', 'copy_container', 'default',
'distinct', 'endswith', 'engine', 'execute', 'foreign_key',
'foreign_keys', 'get_engine', 'in_', 'index', 'key', 'label', 'like',
'metadata', 'name', 'nullable', 'onupdate', 'op', 'orig_set', 'parens',
'primary_key', 'quote', 'scalar', 'select', 'shares_lineage',
'startswith', 'table', 'to_selectable', 'type', 'unique', 'use_labels']
```

Select() function properties

- If you want to select all columns and rows you can do:

```
all=User.select()
```

- When you do that all becomes a list
- You can access first element by

```
all[0]      #To get the whole object
or
all[0].FIRSTNAME  #To get actual value
or
all[0].keys() #To get a list of all column names for that row
or
for column in all[0].c.keys():
    print getattr(all[0],column)
```

- Where 0 tells its a first record.

get() function

- If you have a primary key, 1,2,3,4,.....you could issue this command

```
one=User.get(55) #This will get all columns for primary key = 55
one_lastname= User.get(55).LASTNAME #This will just get one column for
key = 55
```

- If you have a compound (composite) key you could use

```
one=User.get((55,3)) #This will get all columns for primary key1 = 55,
key2=3

*To Access the column with get you do similar thing as we did in example
above.
{{{
one=User.get(55)
one.LASTNAME
one.FIRSTNAME
or
for column in one.c.keys():
    print getattr(one,column)
```

- Access the column names and values in a same way.

insert record

- The easiest way to save your data is to do the following:

```
x=model.User()
x.LASTNAME='Smith'
x.FIRSTNAME='John'
x.AGE=29
x.save()
x.flush() #This will actually save it to a database
```

- You initiate an object. Then you set its attributes and then you save. You can create and save another object. When you are ready to save you call the flush()

update record

- One of the fastest way to update existing record is to get the oldone. Change the records and save it back.
- We do it like this:

```
y=Bdriver.get(343)
y.LASTNAME='Johns'
y.FIRSTNAME='Jimmy'
y.flush()
```

- If you are getting a compaund key you do the following on get:

```
y=Bdriver.get((343,3))
```

Display record

- To display all fields, you can do the following:

```
@expose(template="quote.templates.all")
def index(self):
    x=model.Binder.select()
    return dict(binders=x)
```

- Now copy welcom.kid to all.kid and make sure it has these lines:

```
<div py:for="binder in binders">
  <span py:content="binder.BINDER_SID">bidner_sid</span>
  <span py:content="binder.LAST">last name </span>
  <span py:content="binder.FIRST">first name</span>
</div>
```

Flow Control: update,submit,save,display

- When you have 2 forms and you need to pass save and move on to the next form here is a way to do it.
- Let say your controller has these 4 functions. updatepage1, savepage1, updatepage2, savepage2

```
class Root(controllers.RootController):
    @expose(template="quote.templates.welcome")
    def index(self):
        pass
    @expose(template="myapp.templates.updatepage1")
    def updatepage1(self):
        submit_action = "/savepage1"
        return dict(form=page1_form, action=submit_action)
    @expose()
    @error_handler(updatepage1)
    @validate(form=page1_form)
    def savepage1(self, **kwargs):
        .....
        #Move to next screen. Option 1 raise redirect
        flash('Policy Information Saved')
        raise turbogears.redirect(url("updatepage2",tg_errors=None))
        #option 2 Return function self.page2
        #return self.updatepage2()
    @expose(template="quote.templates.updatepage2")
    def upddriver(self, data=''):
        submit_action = "/savepage2"
        return dict(form=page2_form, action=submit_action)
```

```
@expose()
@error_handler(updatepage2)
@validate(form=page2_form)
def savedriver(self, **kwargs):
    .....
    #Redirect in a similar way as in savepage1
```

Saving table with many columns

- In order to save multiple columns from a table you can use python setattr function.

```
tableinstance=model.Yourtable()
attribute=''
for key in kwargs.keys():
    attribute=upper(key)
    setattr(tableinstance,attribute,kwargs[key])
tableinstance.flush()
```

- Otherwise you can initiate them one by one.

```
tableinstance=model.Yourtable() tableinstance.fieldname='some value' tableinstance.flush()
```

Saving multiple records

- If you come across a problem where you need to save multiple records that have little differences in them you can do something like this.
- You will need to use a pure sqlalchemy. Assigning_mapper,elixir won't help you here. No extra configuration needed just different statements.
- In your model.py you should have something like this:

```
import sqlalchemy
#Binding sqlalchemy to a turbogears database
bind_meta_data()

#Fixed in next release. Table creation, ticket 482
#from turbogears import database
#engine=database.get_engine()
#tables = []
#for name in engine.execute("SHOW TABLES"):
#    tables[name] = sqlalchemy.Table(name, metadata, autoload=True)

#Using manual table creation
user_table = sqlalchemy.Table('User', metadata, autoload=True)
#Some other code that deals with assign mapper
```

- We will use the user_table as our saving point.
- In Controller you do:

```

usertable=model.user_table.insert()
recordstosave=[]
#Do your for loop on a dictionary from a form widget
for key,value in kwargs.items():
    new={}
    new['GROUP_SID']=primarysomevariable
    new['USER_SID']=somevalue
    #Defaults
    new['FACTOR']=0
    new['RATE']=0
    new['BASE']=0
    if key=='GROUP1' and value==True:
        new['group']=1
        recordstosave.append(new)
    elif key=='GROUP2' and value==True:
        new['group']=1
        recordstosave.append(new)
usertabletable.execute(recordstosave)

```

sqlalchemy in virtualenv

- If you would like to set up sqlalchemy in virtual enviroment here are instructinons on how to do it.

```

aptitude install python-virtualenv
cd
python /usr/lib/python2.4/site-packages/virtualenv.py ENV

```

- Your setup is ready to run. You have 2 options now.
- You can use it while virtual environment is activated(recommended) or when its not activated.

Not Activated

- Add your virtual path to PYTHONPATH:

```

export PYTHONPATH=/home/lucas/ENV/lib/python2.4/site-packages/
echo $PYTHONPATH

```

- Download the your program or svn, and install it:

```

mkdir tmp
cd tmp
svn checkout http://svn.sqlalchemy.org/sqlalchemy/trunk sqlalchemy
cd sqlalchemy
python setup.py install --prefix /home/lucas/ENV

```

- To start python you can do:

```
/home/lucas/ENV/bin/python
```

Activate virtual Environment

- Activated Python Path.
- This setup will have all the virtual environment files.
- Point your bash to a new virtual environment:

```
source ENV/bin/activate
```

- You should see your bash changed to (ENV)lucas@debianlaptop:

Deactivate Virtual Environment

- To deactivate it when you are done do:

```
deactivate
```

Use Activated Virtual Environment

- To install additional packages after activating your new virtual environment, you can run command like:

```
easy_install Elixir-0.5.2-py2.4.egg  
or  
easy_install SQLAlchemy  
or  
easy_install SQLAlchemy==dev # for svn trunk  
easy_install SQLAlchemy==0.4.5 # whatever version
```

- This way you use python like you normally would, but in reality you are pointing to virtual environment.

Session Managment

Session ID

- In dev.cfg enable sessions.

```
session_filter.on = True
```

- This will create session for each request. This session is a dictionary which you can access by `cherry.py.session`
- In your controller add this to import lines

```
import cherrypy
```

- `cherrypy.session` holds a variable called `_id` which you has a unique session id key.

```
cherrypy.session['_id']
```

Turbogears API Details

<http://tg.maetico.com/api/> Modules it covers:

Packages

```
turbogears
turbogears.command
turbogears.view
turbogears.view.templates
turbogears.widgets
turbogears.widgets.templates
turbogears.widgets.tests
```

Modules

```
turbogears.command.base
turbogears.command.i18n
turbogears.command.info
turbogears.command.quickstart
turbogears.config
turbogears.controllers
turbogears.database
turbogears.scheduler
turbogears.validators
turbogears.view.base
turbogears.widgets.base
turbogears.widgets.big_widgets
turbogears.widgets.datagrid
turbogears.widgets.forms
turbogears.widgets.i18n
turbogears.widgets.links
turbogears.widgets.meta
turbogears.widgets.rpc
turbogears.widgets.tests.test_datagrid
turbogears.widgets.tests.test_forms
turbogears.widgets.tests.test_link_inclusion
turbogears.widgets.tests.test_nested_form_controllers
turbogears.widgets.tests.test_nested_widgets
turbogears.widgets.tests.test_new_validation
```

```
turbogears.widgets.tests.test_request_related_features
turbogears.widgets.tests.test_widgets
```

Classes

```
turbogears.command.i18n.InternationalizationTool
turbogears.command.info.InfoCommand
turbogears.command.quickstart.BaseTemplate
turbogears.command.quickstart.quickstart
turbogears.command.quickstart.TGBig
turbogears.command.quickstart.TGTemplate
turbogears.command.quickstart.TGWidgetTemplate
turbogears.command.quickstart.TurbogearsTemplate
turbogears.command.quickstart.update
turbogears.controllers.Controller
turbogears.controllers.RootController
turbogears.database.AutoConnectHub
turbogears.database.EndTransactionsFilter
turbogears.database.PackageHub
turbogears.scheduler.DayTaskRescheduler
turbogears.scheduler.ForkedIntervalTask
turbogears.scheduler.ForkedMonthdayTask
turbogears.scheduler.ForkedScheduler
turbogears.scheduler.ForkedTaskMixin
turbogears.scheduler.ForkedWeekdayTask
turbogears.scheduler.IntervalTask
turbogears.scheduler.MonthdayTask
turbogears.scheduler.Scheduler
turbogears.scheduler.Task
turbogears.scheduler.ThreadedIntervalTask
turbogears.scheduler.ThreadedMonthdayTask
turbogears.scheduler.ThreadedScheduler
turbogears.scheduler.ThreadedTaskMixin
turbogears.scheduler.ThreadedWeekdayTask
turbogears.scheduler.WeekdayTask
turbogears.validators.DateTimeFieldConverter
turbogears.validators.FieldStorageUploadConverter
turbogears.validators.JSONValidator
turbogears.validators.Money
turbogears.validators.MultipleSelection
turbogears.validators.Number
turbogears.validators.Schema
turbogears.validators.UnicodeString
turbogears.view.base.cycle
turbogears.view.base.DeprecatedDictWrapper
turbogears.view.base.DeprecatedVariableProviders
turbogears.view.base.MetaDeprecatedVariableProviders
turbogears.view.base.UserAgent
turbogears.widgets.base.CompoundWidget
turbogears.widgets.base.CSSLink
turbogears.widgets.base.CSSSource
turbogears.widgets.base.JSLink
```

```
turbogears.widgets.base.JSSource
turbogears.widgets.base.Link
turbogears.widgets.base.Resource
turbogears.widgets.base.Source
turbogears.widgets.base.Widget
turbogears.widgets.base.WidgetDescription
turbogears.widgets.base.WidgetsDeclaration
turbogears.widgets.base.WidgetsList
turbogears.widgets.big_widgets.AjaxGrid
turbogears.widgets.big_widgets.AutoCompleteField
turbogears.widgets.big_widgets.CalendarDatePicker
turbogears.widgets.big_widgets.CalendarDateTimePicker
turbogears.widgets.big_widgets.LinkRemoteFunction
turbogears.widgets.big_widgets.RemoteForm
turbogears.widgets.big_widgets.URLLink
turbogears.widgets.datagrid.DataGrid
turbogears.widgets.datagrid.PaginateDataGrid
turbogears.widgets.forms.Button
turbogears.widgets.forms.CheckBox
turbogears.widgets.forms.CheckBoxList
turbogears.widgets.forms.CompoundFormField
turbogears.widgets.forms.CompoundInputWidget
turbogears.widgets.forms.FieldSet
turbogears.widgets.forms.FileField
turbogears.widgets.forms.Form
turbogears.widgets.forms.FormField
turbogears.widgets.forms.FormFieldsContainer
turbogears.widgets.forms.HiddenField
turbogears.widgets.forms.ImageButton
turbogears.widgets.forms.InputWidget
turbogears.widgets.forms.Label
turbogears.widgets.forms.ListForm
turbogears.widgets.forms.MultipleSelectField
turbogears.widgets.forms.PasswordField
turbogears.widgets.forms.RadioButtonList
turbogears.widgets.forms.RepeatingFieldSet
turbogears.widgets.forms.RepeatingFormField
turbogears.widgets.forms.RepeatingInputWidget
turbogears.widgets.forms.ResetButton
turbogears.widgets.forms.SelectionField
turbogears.widgets.forms.SingleSelectField
turbogears.widgets.forms.SubmitButton
turbogears.widgets.forms.TableForm
turbogears.widgets.forms.TextArea
turbogears.widgets.forms.TextField
turbogears.widgets.i18n.CalendarLangFileLink
turbogears.widgets.i18n.LocalizableJSLink
turbogears.widgets.links.JumpMenu
turbogears.widgets.links.SyntaxHighlighter
turbogears.widgets.links.Tabber
turbogears.widgets.meta.MetaWidget
turbogears.widgets.rpc.RPC
turbogears.widgets.tests.test_datagrid.Foo
turbogears.widgets.tests.test_datagrid.TestDataGrid
```

```

turbogears.widgets.tests.test_datagrid.User
turbogears.widgets.tests.test_forms.CallableCounter
turbogears.widgets.tests.test_forms.NestedController
turbogears.widgets.tests.test_forms.Request
turbogears.widgets.tests.test_forms.w1
turbogears.widgets.tests.test_forms.w2
turbogears.widgets.tests.test_nested_form_controllers.MyRoot
turbogears.widgets.tests.test_nested_widgets.InnerSchema
turbogears.widgets.tests.test_nested_widgets.MiddleSchema
turbogears.widgets.tests.test_nested_widgets.OuterSchema
turbogears.widgets.tests.test_nested_widgets.Request
turbogears.widgets.tests.test_nested_widgets.TestNestedSchemaValidators
turbogears.widgets.tests.test_nested_widgets.TestNestedWidgets
turbogears.widgets.tests.test_nested_widgets.TestNestedWidgetsWMixedValidation
turbogears.widgets.tests.test_nested_widgets.TestNestedWidgetsWSchemaValidation
turbogears.widgets.tests.test_nested_widgets.TestSchema
turbogears.widgets.tests.test_new_validation.NotSoSimpleSchema
turbogears.widgets.tests.test_new_validation.Request
turbogears.widgets.tests.test_new_validation.SimpleFields
turbogears.widgets.tests.test_new_validation.SimpleFieldSet
turbogears.widgets.tests.test_new_validation.SimpleForm
turbogears.widgets.tests.test_new_validation.SimpleSchema
turbogears.widgets.tests.test_new_validation.TestNestedForm
turbogears.widgets.tests.test_new_validation.TestSimpleForm
turbogears.widgets.tests.test_widgets.A
turbogears.widgets.tests.test_widgets.B
turbogears.widgets.tests.test_widgets.C
turbogears.widgets.tests.test_widgets.Fields
turbogears.widgets.tests.test_widgets.Fields
turbogears.widgets.tests.test_widgets.FieldsSchema
turbogears.widgets.tests.test_widgets.FieldsSchema
turbogears.widgets.tests.test_widgets.Request
turbogears.widgets.tests.test_widgets.TestParams
turbogears.widgets.tests.test_widgets.TestSchemaValidation
turbogears.widgets.tests.test_widgets.TestSchemaValidationWithChildWidgetsValidato

```

Performance

Genshi vs other

-  Genshi Performance

maco vs Cheetah

from maco website:

Mako: 1.10 ms	Myghty: 4.52 ms
Cheetah: 1.10 ms	Genshi: 11.46 ms
Django: 2.74 ms	Kid: 14.54 ms

Sqlalchemy vs storm

- [Sqlalchemy vs Strom](#)

Errors

TgFastData for python 2.5

- Currently the Tgfast data in download section of TurboGears is not up to date so we need to install it from subversion repository
- You should have subversion installed if not aptitude install subversion

```
easy_install http://svn.turbogears.org/projects/FastData/trunk
```

EXTRA

Produce PDF Pages

- <http://achieviewith.us/public/articles/2007/02/21/produce-pdf-pages-with-turbogears-cheetah-and-i>
- Follow along with the rest of this document to see how to produce PDF documents with the help of TurboGears, Cheetah, and ReportLab. We will generate form letters in response to a job opening, and in Part II we will print mailing labels for our letters. (excel)
- [Optional] http://www.tinyerp.com/component/option,com_joomlaxplorer/Itemid,132/
-> <http://www.tinyerp.com/demonstration.html>

MyTube with Flex and Turbogears

- <http://codedemigod.com/node/11>
- Upload a few more videos, and then visit the simplest address(<http://localhost:8080/simplest>). This is enough to show off Python and TurboGears and hopefully get you to consider them in your projects.

captcha widgets for a form

- <http://code.google.com/p/tgcaptcha/>
- TGCaptcha is a TurboGears widget that provides an easy way to incorporate a captcha as

part a form in an attempt to reduce spam or malicious activity.

Basic Way to display column data from a database

- http://groups.google.com/group/turbogears/browse_thread/thread/86bbb9be7782060b/6252e65c30

Identity Management

- <http://docs.turbogears.org/1.0/IdentityManagement>
- Many database Web Apps need to identify users. For example, you log in to access your Gmail account, or to use Amazon bookstore services. TurboGears therefore provides an identity system that you can use in your applications. This system supports both authentication and authorization functions, and allows role-based access control by assigning users to groups.

TurboGears with existing MySQL database

- http://groups.google.com/group/turbogears/browse_thread/thread/69e4b982b78987db/4214fb7e55 -talks and links to sqlalchemy ways of doing it
- <http://www.sqlalchemy.org/trac/wiki/UsageRecipes/AutoCode> - This is a way to automatically create table definitions from an existing database.
- http://www.sqlalchemy.org/docs/metadata.html#metadata_tables_reflecting
- <http://sqlobject.org/SQLObject.html#automatic-class-generation>
- For SqlObject just do this in model.py:

```
from sqlobject import *  
  
class Book(SQLObject):  
    class sqlmeta:  
        fromDatabase = True
```

Flash status Bar

- http://www.splee.co.uk/2005/11/23/fancy-status-messages-using-tg_flash/

Tagging with TurboGears

- <http://www.thesamet.com/blog/2006/11/17/tutorial-how-to-implement-tagging-with-turbogears-and-sql>

Drag and Drop Sort list

<http://wiki.script.aculo.us/scriptaculous/show/SortableListsDemo>

Turbogears and Oracle

<http://www.oracle.com/technology/pub/articles/rubio-python-turbogears.html>

Turbogears memory management

- These apply to memory leaks in turbogears apps with sqlalchemy, but you should be able to use these strategies in sqlalchemy.

<http://psychicorigami.com/2007/10/27/a-little-sqlobject-performance-guide/>

<http://psychicorigami.com/2007/12/16/using-raw-sql-with-sqlobject-and-keeping-the-object-y-goodness>

unixODBC

- Install odbc, and mssql drivers (tdsodbc)

```
aptitude update
aptitude install tdsodbc unixodbc
```

- Create a driver template.

```
vi /etc/freetds/tds.driver.template
```

- Inside add :

```
[TDS]
Description      = FreeTDS Driver for Linux & MSSQL on Win32
Driver           = /usr/lib/odbc/libtdsodbc.so
Setup            = /usr/lib/odbc/libtdsS.so
```

- Now dsn file:

```
vi /etc/freetds/tds.dsn.template
```

- And inside add:

```
[DSN_NAME]
Description      = Description of you DSN connection.
Driver           = TDS
```

```
Trace           = No
Database        = DefaultDatabase [replace with your database name]
Server          = mysqlserver.inter.net [replace with your SQL server's
host,ip]
Port           = 1433 [replace with the port that SQL is listening on]
```

- Now run this command to add your driver to unixodbc

```
odbcinst -i -d -f /etc/freetds/tds.driver.template
```

- Now you add dsn. You have 2 options.
- Add system DSN for all users

```
odbcinst -i -s -l -f /etc/freetds/tds.dsn.template
```

- Add dsn to a user your are running a command on:

```
odbcinst -i -s -f /etc/freetds/tds.dsn.template
```

- Now test your connection:

```
isql -v DSN_NAME username password
```

- You should get something like

```
*****
* unixODBC - isql                               *
*****
* Syntax                                         *
*                                               *
.
.
.
more
.
```

Authenticating against an external password source



<http://docs.turbogears.org/1.0/IdentityManagement#authenticating-against-an-external-password-source>

Embed swf files in your turbogears code (IE, Opera, Firefox etc)

- <http://code.google.com/p/swfobject/>

Turbogears Specifics usage

Use CAPITAL names in widgets and sqlalchemy autoload

- I am querying the database using sqlalchemy

```
d=model.Useraddress()
x=d.get_by(USER_SID=345,ADDRESS_SID=2)
```

Then I pass x(the corresponding record) to my template

```
return dict(address=address_form,value=x, action=submit_action)
```

But when I want to prefill the widget form with values from sqlalchemy using:

```
<p><span>${address(value=value, action=action)}</span></p>
```

The values are not set Am I missing something here or values from sqlalchemy query do not correspond to widget fields?

It seems as it does but it needs to be the same case.

```
So I went into my widget class and I replaced the
"City" with "CITY" which caused the value to fill in.
```

It is the same case when saving from widget to sqlalchemy. If your widget doesn't have CAPITAL names then you have to convert them.

```
def saveuser(self,**kwargs):
    from string import upper
    b.USER_SID=(int(max_id)+1)
    #Assigning rest of the keys
    for widgetkey in kwargs.keys():
        j=upper(widgetkey)
        setattr(b,j,kwargs[widgetkey])
```

CONCLUSION: When using sqlalchemy and widgets we need to name our attributes in the widget using CAPITAL letters. when we use sqlalchemy autoload function.

sqlalchemy:

```
user_table = sqlalchemy.Table('user', metadata, autoload=True)
```

widget:

```
class User(widgets.WidgetsList):
    LASTNAME=widgets.TextField(label='Last
Name', validator=validators.NotEmpty)
    FIRSTNAME=widgets.TextField(label='First
Name', validator=validators.NotEmpty)
```

WAS THIS INTENDED? If not should it be documented somewhere or changed? 1-15-08. update. I just loaded my table a table with all lowercase and I didn't need all capital letters. There has to be some kind of exception.

redirect all requests to index

- Add this to your controller. This is a default function that will return index. You can change self.index to return other function.

```
@expose()
def default(self, *args, **kw):
    return self.index(*args, **kw)
```

Other Documentation

1. [Turbogears by IBM](#)
2. [Turbogears by Oracle](#)

Common Errors

non-keyword arg after keyword arg

```
SyntaxError: non-keyword arg after keyword arg
```

- This means that you should switch the order of the arguments

BAD:

```
x2=sqlalchemy.select([Users.c.AGE],
group_by=[Users.c.AGE],Users.c.LASTNAME=='Smith')
```

GOOD:

```
x2=sqlalchemy.select([Users.c.AGE],
Users.c.LASTNAME=='Smith',group_by=[Users.c.AGE])
```

cherrypy._cperror.NotReady: Port not free

- This means you are running something else on that port
- In dev.cfg, uncomment the

```
server.socket_port=8080
```

and change it to

```
server.socket_port=8081
```

then python start_yourproject.py

ExpatError: not well-formed (invalid token)

- If you include links that have "&" in it you will have to replace it with:

```
Replace "&" with "&amp;"
<a
href="http://www.google.com/search?hl=en&q=schools+near+60630&btnG=Google+Search">
(bad)
<a
href="http://www.google.com/search?hl=en&amp;q=schools+near+60630&amp;btnG=Search"
(good)
```

Could not assemble any primary key columns for mapped table

-  Could not assemble any primary key columns for mapped table

MyWiki: TurboGears (last edited 2008-04-14 18:25:59 by LukaszSzybalski)