Introduction to TurboGears

Mike Pirnat

Clepy Meeting: 11/07/2005

http://www.pirnat.com/geek/

What, Another Python Framework??

“You're not a Real Python Programmer until you've written your own web templating toolkit.”

-- Moshe Zadka

“Stop writing kewl new frameworks! Help improve the top few frameworks so they can become a best of breed.”

-- Michelle Levesque

What is TurboGears?

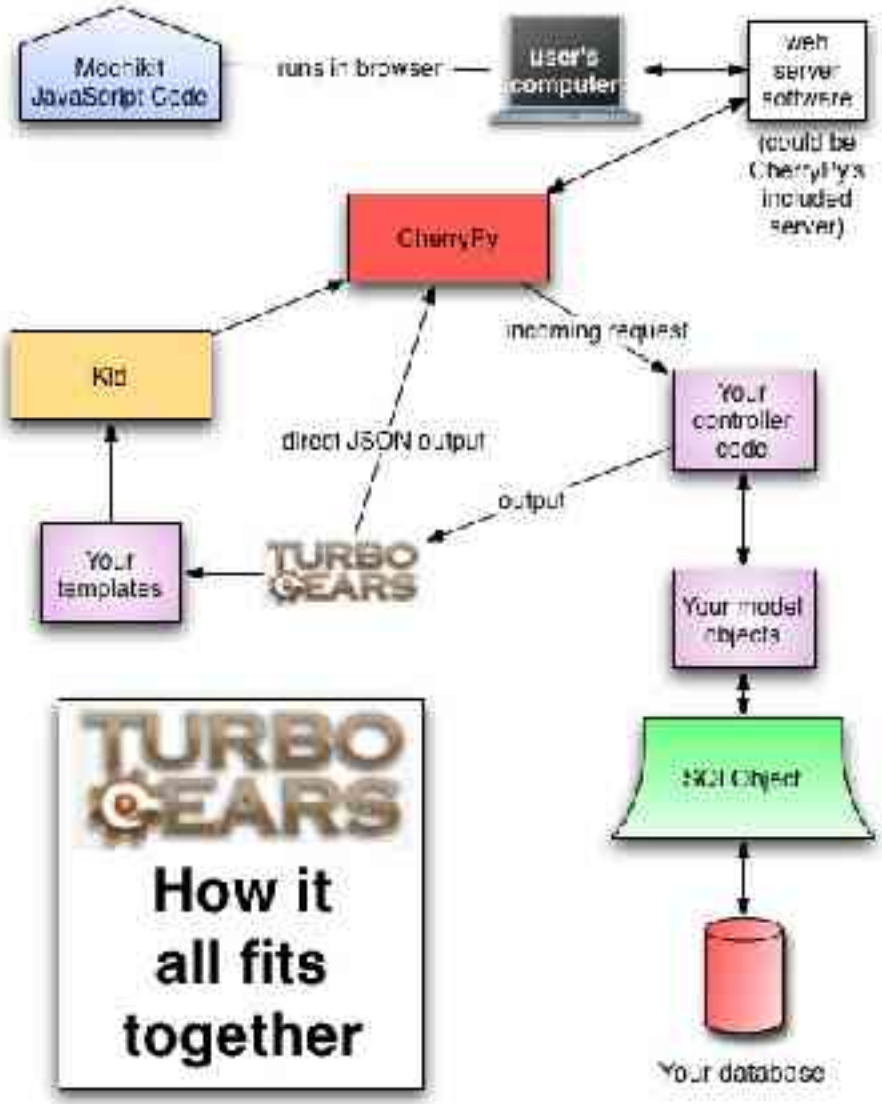
- Python “megaframework” for web apps
- Pythonic answer to Ruby on Rails
- Rapid web development
- Rapidly growing community—over 500 users on the mailing list since late September!
- Not afraid of “Not Invented Here”
- Under heavy development
- Shiny and new... or is it?

TurboGears Philosophy

- Create great web apps faster, easier, and more fun
- Use the best available existing components
- Combine in an easy-to-use package
- Deliver an excellent out-of-box experience
- MVC == good
- Eliminate tedium with abstraction, tools
- TurboGears provides the “glue” so that you don't have to make your own

TurboGears Components

- Database ORM access: SQLObject
- Application server: CherryPy
- Template engine: Kid
- Javascript/AJAX library: MochiKit
- Automated testing: TestGears, Nose


Mochikit
JavaScript Code
runs in browser
user's
computer
web
server
software
(could be
CherryPy's
included
server)
CherryPy
Kid
incoming request
Your
controller
code
direct JSON output
output
Your
templates
TURBO
GEARS
Your model
objects
SQL Object
Your database
TURBO
GEARS
How it
all fits
together

TurboGears Kickstarts Your App

- Create a new project and directory structure with `tg-admin quickstart`
- Start your server with `python` *`PROJECT`*`-start.py` (created by quickstart in the root of your app)
- Your install is now up and running at: http://localhost:8080/!
- Set up your `dev.cfg` (created automatically) with database connection string
- Define your data model using SQLObject subclasses, then run `tg-admin sql create`
- Then the fun part begins!

SQLObject

- Object Relational Manager
- Database tables are classes, rows are instances, fields are attributes
- Eliminates need to write SQL by hand (mostly)
- Use with MySQL, PostgreSQL, sqlite, Firebird
- Sybase and Oracle support in progress
- Can generate DB from Python, or Python from DB!

SQLObject

- Supports joins, one-to-many, and many-to-many relationships
- Supports transactions
- Can do lazy updates to minimize DB chatter
- Result sets are lazily evaluated generators
- Result sets can be sliced => different SQL
- Caches objects retrieved from DB
- Can still do full-on SQL if needed
- http://sqlobject.org/

SQLObject: Example

```
from sqlobject import *
from datetime import datetime

class Person(SQLObject):
    firstName = StringCol(length=100)
    middleInitial = StringCol(length=1, default=None)
    lastName = StringCol(length=100)
    lastContact = DateTimeCol(default=datetime.now)

# create a new record
p = Person(firstName="John", lastName="Doe")

# get an existing record
p1 = Person.get(1)

# select multiple records
peeps = Person.select(Person.q.firstName=="John")
```

CherryPy

- Pythonic, objected-oriented web app framework
- Maps URL request to a Python method using decorators
- CGI variables are mapped to method arguments
- Does not support SSL natively; put it behind Apache with mod_proxy or mod_python to allow secure traffic
- http://www.cherrypy.org

CherryPy: Example

```
import cherrypy

class MyRoot:

    @cherrypy.expose()
    def index(self, who=”World”):
        return “Hello, %s!” % who
```

CherryPy/TurboGears: Examples

```
import cherrypy

class MyRoot:

    @cherrypy.expose()
    def index(self, who="World"):
        return "Hello, %s!" % who
```

```
import turbogears
from turbogears import controllers

class MyRoot(controllers.Root):

    @turbogears.expose(html="foo")
    def index(self, who="World"):
        return dict(who=who)
```

Kid

- Templates are well-formed XML
- Attribute language similar to Zope's TAL
- Compiled to Python byte-code
- Processed using a pull-style parser based on ElementTree
- Supports template inheritance and XSLT-like matching
- Designer-friendly: viewable in browser
- http://kid.lesscode.org

Kid: Example

```
<?python
title = "A Kid Test Document"
fruits = ["apple", "orange", "kiwi", "M&M"]
from platform import system
?>
<html xmlns:py="http://purl.org/kid/ns#">
  <head>
    <title py:content="title">This is
replaced.</title>
  </head>
  <body>
    <p>These are some of my favorite
fruits:</p>
    <ul>
      <li py:for="fruit in fruits">
        I like ${fruit}s
      </li>
    </ul>
    <p py:if="system() == 'Linux'">
      Good for you!
    </p>
  </body>
</html>
```

```
<?xml version="1.0" encoding="utf-8"?>
<html>
  <head>
    <title>A Kid Test Document</title>
  </head>
  <body>
    <p>These are some of my favorite
fruits:</p>
    <ul>
      <li>I like apples</li>
      <li>I like oranges</li>
      <li>I like kiwis</li>
      <li>I like M&amp;amp;Ms</li>
    </ul>
    <p>
      Good for you!
    </p>
  </body>
</html>
```

Kid: Template Inheritance

- Master template defines a “template function”

```
<div py:def=”header(title)”>
  <h1>${title}</h1>
</div>
```

- Individual templates extend the master and call the function:

```
<html py:extends=”'common.kid'”> ...
<div py:replace=”header('Foo!')”>This will be
  replaced</div>
...
</html>
```

Kid: Matching

- Individual templates extend from master
- Master template defines a matching rule:

```
<body py:match="item.tag=='{http://www.w3.org/1999/xhtml}
body'">
  <h1>${title}</h1>
  <div py:replace="item[:]">
```

MochiKit

- Pythonic JavaScript library
- “Makes JavaScript suck less”
- Well-documented
- Reliable (lots of automated tests)
- Single JS import to have access to all features
- Features galore...
- http://mochikit.com

MochiKit: Features

- Asynchronous tasks
- DOM manipulation
- Color abstraction
- Date and time
- String formatting
- Iteration (ala Python's itertools)
- Logging
- Interactive JavaScript shell
- Visual effects

MochiKit: Sorting & Accessing

```
myObjectArray = [
  {“a”: 3, “b”: 2},
  {“a”: 1, “b”: 2}
];

// sort by the “a” property
myObjectArray.sort(keyComparator(“a”));

// get just the “a” values out into an array
sortedAValues = map(itemgetter(“a”), myObjectArray);
```

MochiKit: DOM Manipulation

```
var rows = [
  [“dataA1”, “dataA2”, “dataA3”],
  [“dataB1”, “dataB2”, “dataB3”]
];

row_display = function(row) {
  return TR(null, map(partial(TD, null), row));
}

var newTable = TABLE({'class': 'prettytable'},
  THEAD(null, row_display([“head1”, “head2”, “head3”])),
  TFOOT(null, row_display([“foot1”, “foot2”, “foot3”])),
  TBODY(null, map(row_display, rows)));

swapDOM(oldTable, newTable);
```

Testing: TestGears and Nose

- TurboGears up to 0.8: TestGears
- TurboGears 0.9+ will use Nose
- Both are unittest extensions that mimic the behavior and ease-of-use of py.test
- Automated test discovery and execution
- More info:
  - http://www.turbogears.org/testgears
  - http://somethingaboutorange.com/mrl/projects/nose

TestGears

- Where to put tests? Generally in a “tests” package in your project
- Running tests? At the top level directory of your project, run: `python setup.py testgears`
- Testing methods:
  - Directly
  - Simulating a web request

TestGears: Direct Testing

```
from turbogears.tests import util
from myproject.controllers import Root
import cherrypy

def test_indextitle():
    """Index method should return a title."""
    cherrypy.root = Root()
    output = util.call(cherrypy.root.index)
    assert output['title'] == 'foobar'
    assert output.has_key('now')
```

TestGears: Simulating a Web Request

```
from turbogears import util
from myproject.controllers import Root
import cherrypy

def test_indextitle_in_template():
    """Index template should have a title."""
    cherrypy.root = Root()
    util.createRequest('/')
    assert "<title>Foobar</title>" in \
            cherrypy.response.body[0]
```

The Infamous 20-Minute Wiki

- How do we put components together?
- How quickly can we make an app?
- Would you believe... a usable wiki in 20 minutes?
- “Screencast” inspired by the Ruby on Rails introductory video
  - http://www.turbogears.org/docs/wiki20/20MinuteWiki.mov
  - http://www.turbogears.org/docs/wiki20/index.html

Current “Hot Topics”

- CRUD
- Identity
- Installation, Upgrading, Deployment
- Kid issues
- SQLObject issues
- Forms: Validation & Generation

Hot Topics: CRUD

- CRUD = Create, Retrieve, Update, Delete
- TG wanted to include “free CRUD” ala Django and Rails
- Several groups started in their own directions
- When suddenly... along came CatWalk!
  - Manage and interact with your TG models from a browser
  - http://checkandshare.com/catwalk/
  - Now included with TurboGears SVN
  - Will be bundled in 0.9+

Hot Topics: Identity

- Jeff Watkins is producing an identity framework for TurboGears
- Should be ready for TG 0.9
- User, group, permission scheme
- IP-based access control
- Should be interchangeable with other preferred/required identity models which implement a consistent interface
- http://newburyportion.com/nerd/2005/10/identity-management-for-turbogears

Hot Topics: Installation & Upgrading

- Initial installation is easy thanks to Easy Install:

```
sudo python ez_setup.py -f http://www.turbogears.org/download/index.html \
    --script-dir /usr/local/bin TurboGears
```

- TG and friends are distributed as Python Eggs
- Upgrading has been tricky, often requiring updates to the update framework in order to get the latest versions installed correctly
- Eggs and other package systems not well reconciled yet; potential for conflicts and problems; Internet connection required!
- Switching from release to SVN is tricky

Hot Topics: Deployment

- Non-root installation requires special procedures (makes it difficult to deploy in a budget hosting environment)
- Virtual hosting solution is yet to be determined (multiple sites on the same server)
- Combining multiple TG apps into a single site is yet TBD

Hot Topics: Kid Issues

- Errors in the template that cause it to be invalid XML will kill the CherryPy instance!
- HTML comments in a master template cause Kid to barf when rendering individual templates
- HTML entities that aren't a standard part of XML break Kid (unless you get clever)
- Plain text output will be in 0.7

Hot Topics: SQLObject Issues

- Creating tables with foreign-key dependencies on each other is tricky due to alphabetical order of creation; fixed in the forthcoming 0.8
- Awkward for efficiently accessing multi-table data sets
- SQLObject is not designed for statistics/reporting
- SQLObject can be too active in database activity
- Effort underway to examine Hibernate (a very powerful open source ORM from Java) and beef up SQLObject
- API cleanup is scheduled for the 0.8 release

Hot Topics: Forms & Validation

- Current state of form validation sucks
- FormEncode validators are part of the `@turbogears.expose()` decorator; cumbersome for large forms
- If validation fails, a special method named `validation_error()` will be called with arguments:
  - String with the name of the original method
  - Dictionary of arguments passed to the original method
  - List of validation exceptions that were raised
- No `validation_error()` => exception! Boom!

Hot Topics: Forms & Validation

- Coming soon: widgets!
- Widgets live in Python code, rendered by Kid templates
- Forms will be a particular kind of widget, made up of other widgets:

```
myform = TableForm(widgets=[
    widgets.TextField(“name”),
    widgets.TextField(“address”),
    widgets.TextField(“age”, default=0, \
        validator=validators.int())])
```

Hot Topics: Forms & Validation

- Or by popular demand, a form might instead be a class instead of an instance:

```
class myform(TableForm):
    name = widgets.TextField(“name”)
    address = widgets.TextField(“address”)
    age = widgets.TextField(default=0, \
        validator=validators.int())
```

Hot Topics: Forms & Validation

- Display a form using a form widget:

```
@turbogears.expose(html="foo")
def index(self):
    return dict(form=myform)

<div py:replace="form.insert(action="save")"/>
```

- Validate a form using a form widget:

```
@turbogears.expose(input_form=myform)
def save(self, name, address, age, has_errors):
    if has_errors:
        return self.index() # or do whatever
```

Hot Topics: Forms & Generation

- Recently up for discussion: generating the rendered form based on the widgets as a “kick-start” to building templates
- Output would probably require customization before being deployed
- Automatic generation would give a designer a starting point without having to build the form up from nothing
- What happens when the Python changes—regenerate and re-customize the form?

Dive On In!

- Starting points on the web:
  - http://www.turbogears.org/docs/
  - http://www.turbogears.org/docs/gettingstarted.html
  - http://www.turbogears.org/docs/wiki20.html
- Google group: http://groups.google.com/group/turbogears/
- #turbogears channel on irc.freenode.net
- Aggregate blog: http://planet.turbogears.org

Contributing

- Sprints—Ann Arbor, online, PyCon
- How to contribute?
  - http://www.turbogears.org/community/contributing.html
- Contributions should be written to PEP8
  - http://www.python.org/peps/pep-0008.html
- Wiki & ticket system: http://trac.turbogears.org

Op/Ed

- Not ready for prime time—wait for 1.0 if you need to do serious work (if you can)
- Maturing rapidly, very exciting!
- TG is definitely on the “short list” for how to do web apps in Python—Michelle's hypothetical Brian will be happy!
- Good opportunity to shape a major project