



Sahana Eden: Introduction to the Code

5 November 2010, Sahana Camp

Fran Boon

fran@sahanafoundation.org



Emergency

We have to build an application by lunchtime!



As well as being a suite of pre-built applications, Eden is also an Emergency Development Environment: a framework which can be used to rapidly build new ones to meet custom needs (as every emergency is different).



Agenda

- Web2Py execution model
- S3 REST Controller
 - Model
 - Controller
 - View



Web2Py Execution Model

eden module resource

`http://hostname/application/controller/function/args?vars`

1. All **Models** executed every request
 - In alphabetical order
 - Within web2py environment
2. The **Controller** is executed
3. The **View** template is parsed
4. (HTML) page returned to client

Because all models are executed every request, it is important to minimise the amount of time taken to process them.

Alphabetical order is to Manage the dependencies

‘In the Web2Py environment’ means no need to import most of the libraries in ‘gluon/’ & access to the global variables: request, session & response.

First the parts of the Controller outside Functions are executed & then the relevant Function.

Functions with no arguments are automatically visible via URLs

-No need to add a routes entry (unlike other frameworks, like Django)

The Controller or View could also return XML (e.g. RSS or KML), JSON, PDF or XLS.



Models

- Define Tables in Database

Live Migrations:

- Tables created/alterd in database

Deleted Columns / Tables simply Ignored

Sahana Eden is Resource-centric



Resources

- Common Resources from other Models
 - Person
 - Group
 - Location
 - Organisation
 - Site
 - Hospital
 - Shelter

Sahana Eden is Resource-centric



S3 REST Controller

- Args:
 - /update, /create, /delete
- HTTP verbs:
 - GET, PUT (POST), DELETE
- Representations:
 - .html, .json, .xml, .xls, .pdf

Web Services, Mash-ups



Adding a new application

- Vehicle Tracking System
- Name: `vts`
- Resources:
 - vehicle
 - driver
 - location



Model: Define Table

 models/vts.py

```
# Vehicle resource
table = db.define_table("vts_vehicle",
    Field("registration"),
    Field("make"),
    Field("model"),
    Field("supplier"),
)
```

Create a new file in the Models folder called 'vts.py' & type in the text above.

Text after this symbol is a comment & is ignored

If you make a mistake, then the Ticketing system should catch the error & let you know what you did wrong.



Controller: S3 REST

 controllers/vts.py

```
def vehicle():  
    return s3_rest_controller("vts", "vehicle")
```

Create a new file in the Controllers folder called 'vts.py' & type in the text above.



View

None needed at 1st – can RAD without them
& then polish later 😊

Try:



<http://127.0.0.1:8000/eden/vts/vehicle>

1. Create a vehicle.
2. View in different formats:
 - .json, .xml, .xls, .pdf

Note that you'll need to Login to the system before you're allowed to create a resource.

Coffee





Views: S3 REST

- REST has default views
 - create.html
 - display.html
 - list.html
 - list_create.html
 - update.html

These defaults are why we didn't need any views for our application.

Web2Py defaults to `views/controller/function.html`, but S3REST needs a different view per Method



Views

- Python code inside `{{ }}` is parsed server-side
- Views can **extend** & **include** other views
- Extend '**layout**' for overall look/feel

```
{{extend "layout.html"}}
```

No need to learn another macro language

Can **Extend** just a single other file

Can **Include** many others



Views: Custom

- They can also be customised:

 views/vts/vehicle_list_create.html

```
{{extend "layout.html"}}
```

```
{{rheader="Register a new vehicle in the system:"}}
```

```
{{include "_list_create.html"}}
```

Create a new folder in the views folder called 'vts' & inside that put a new file 'vehicle_list_create.html'. Type in the text above.

Refresh the vehicle page to see the new text: <http://127.0.0.1:8000/eden/vts/vehicle>



Model: SQL constraints

 models/vts.py

```
table = db.define_table("vts_vehicle",
    Field("registration", unique=True),
    Field("make"),
    Field("model"),
    Field("supplier"),
)
```


 **value already in database**

Unique=True is a SQL-level constraint.

Open your vts.py from the models directory & add the extra code above.

Try adding multiple vehicles with the same registration.

Validators are DAL-level constraints which provide server-side validation & some also provide client-side rendering & validation.

These are automatically added with SQL constraints to produce nice error messages instead of tickets.

Validators can also be added manually as:

```
db.vts_vehicle.registration.requires = IS_NOT_IN_DB(...)
```



Model: Field Types



models/vts.py

```
table = db.define_table("vts_vehicle",  
    Field("registration", unique=True),  
    Field("make"),  
    Field("model"),  
    Field("purchase_date", "date"),  
    Field("supplier"),  
)
```

Purchase Date:
2010-10-27

◀	Oct	▶	2010	◀	▶	
Su	Mo	Tu	We	Th	Fr	Sa
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

Fields default to type 'string'.

'date' is the SQL-level field type & also provides the class in the HTML which means we get a date widget

(This comes from the default IS_DATE() Validator)

Add the extra code above to your file & refresh the page, now enter a date.



Model: Default Values

 models/vts.py

```
table = db.define_table("vts_vehicle",
    Field("registration", unique=True),
    Field("make"),
    Field("model"),
    Field("purchase_date", "date",
default=request.utcnow),
    Field("supplier"),
    )
```

Add the extra code above to your file & refresh the page to see the default value



Labels

 models/vts.py

```
...  
db.vts_vehicle.registration.label = T("License Plate")
```

License Plate:

Add this to your model (under the table definition) & try it out.

T() is to Internationalise the string



Controller: Comment



controllers/vts.py

```
def vehicle():  
    db.vts_vehicle.registration.comment = \  
        DIV(_class="tooltip", _title="Help text here")  
  
    return s3_rest_controller("vts", "vehicle")
```

License Plate: *

? HELP

Add the comment to your controller & try it out



How do we Navigate?

- Modules menu
- Modules list on frontpage
- Menu within Module



Enable Module



models/000_config.py

```
deployment_settings.modules = Storage(  
    ...  
    vts = Storage(  
        name_nice = "Vehicle Tracking System",  
        description = "Track vehicles",  
        module_type = 10  
    ),  
    ...  
)
```

Add this text to the file 000_config.py in the models folder. Navigate to the home page to see the module appear both there & on the menu. Try navigating to the module.

Module Type 10 means appears in the 'more' section of the default modules menu. (Most deployments will create a fully-customised menu anyway)



Index page

 controllers/vts.py

```
module = request.controller

def index():
    "Custom View"
    module_name = \
deployment_settings.modules[module].name_nice
    return dict(module_name=module_name)
```

Add this text to your controller file & now open the module.

“Custom View” is a Doc String

One should be added to all functions for automatic documentation generators & interactive browsing of docs

You can try this out in the Interactive Shell: **w2p**

```
execfile("applications/eden/controllers/vts.py",
globals())
help(index)
```



View



views/vts/index.html

```
{{extend "layout.html"}}
{{=H2(T(module_name))}}
<p>This module allows users to track their vehicles</p>
{{=LI(A("List Vehicles", _href=URL(r=request,
f="vehicle")))}}}
```

Create a file called 'index.html' inside the views/vts folder

See plain HTML being interspersed with the server-side parsed Python

Look at another module's index.html

What's bad about this index?



Controller: Menu

 controllers/vts.py

```
response.menu_options = [  
    [T("Vehicles"), False, URL(r=request, f="vehicle"), [  
        [T("List"), False, URL(r=request, f="vehicle")],  
        [T("Add"), False, URL(r=request, f="vehicle",  
args="create")] ]]]
```

Add this text to your controller (outside the functions – e.g. at the top) & see what effect this has on your module.

Lunch

