Generv – how to use it

Generv is a simple Python script that will, with a little guidance, create a controller and a set of views from a basic model, and as created these will provide a working application based on web2py.

1. Installation and Set up

Create a web2py project using whatever method you wish with whatever name you choose. Then install Generv, full installation and requirements for Generv are given in the Appendix and you must carry out those steps before you try to use the generator.

2. Create the initial model

Run generv.py, it has a simple command line interface and it asks:

enter application name:

enter the name of the application you gave when you created the project

enter model name:

enter the name of the model that you want to create (use lower case)

M, V or C:

enter M

an initial model file will be created in the /modules sub-directory of the project using the lower-case class name, e.g. fred.py

Edit the model .py file to add the Field definitions in the define_table() call as in standard web2py. Note that a statement to set the tablename attribute is added also:

self.tablename = <name of table>

Add any generv-specific attributes such as 'hasOne', 'hasMany'.

NOTE: the current version of generv has some diagnostic output to the console, please ignore this, it will be removed in a future version. Any errors will be reported to the console.

Special rules for hasOne

The owner model must contain a field with the same name as the owned model and defined as an integer; this field will hold the id of the owned model record.

The owned model must have a field with the name '<model>_name' which is used within the owner's views for identification, rather than simply show the owned record id.

Example:
model supplier has a one to one with model address
in /modules/supplier.py

```
c
define_table('supplier',
...
Field('address', 'integer', label='Address', default=0),
Field('description', length=512, label='Description'),
...
```

```
)

self.hasOne = ['address']
```

and in /modules/address.py

```
define_table('address',
...
Field('address_name', 'string', label='Address'),

...
)

self.isOne = 1
```

Special rules for hasMany

The owner model only refers to the owned record via the hasMany setting.

The owned model must have a field with the name '<model>_name' which is used within the owner's views rather than simply show the owned record id.

The owned model must contain fields named 'owner_type' and 'owner_id' which are used to hold the appropriate values relating to the owner record

Example:
model supplier has one to many with model contact
in modules/supplier.py

```
define_table('supplier',
...
)

self.hasMany = ['contact']
```

in modules/contact.py

```
define_table('contact',
...
Field('contact_name', 'string', label='Contact'),
Field('ownerType', 'string', label='Owner Type'),
Field('ownerId', 'integer', label='Owner'),
...
)

self.isMany = 1
```

3. Generate the controller

Run generv.py:
please enter application name:

enter the name of the application
enter model name:
enter the name of the model for which you want a controller (use lower case)
create model (M), view (V) or controller (C):
enter C

The model's controller .py file will be created in the /controllers sub-directory. Any existing file will be overwritten without warning.

4. Generate the views

Run generv.py:
enter application name:
enter the name of the application
enter model name:
enter the name of the model for which you want the views (use lower case)
M, V or C:
enter V

The appropriate set of views will be created, if the model's /views/modelname sub-directory does not exist it will be created. Any existing files will be overwritten without warning.

5. Run the application

As with any web2py application, simply use the URL:

http://127.0.0.1:8000/<application>/<model>/index.html

and the application will load with a index page for the model without any data but with a 'Create' button and you can go from there.

6. Restrictions and warnings.

1. There is no validation at all on any data entered in a data entry page and invalid data may be stored in the database and subsequently cause errors when attempting to list the data, this is particularly important with integer fields where it appears that non-integer values may be stored and then cause the application to crash on the next attempt to display any data (readily seen in the error log).

2. Only 'string' and 'integer' data items are supported

3. There are a couple of Javascript Alerts that crept in accidentally, they were added for diagnostic purposes and will be removed, just click OK on them as they have no other impact.

4. There are a number of diagnostic messages sent to the console, just ignore them although in the case of any failures they may contain useful information.

5. Clarification for selecting from one to one or one to many Selections

When faced with a popup contain a list of records from which you are supposed to select a record you should click on the record id field of the required record. Because the event listener is set up dynamically there is no automatic change to the mouse icon when hovering over that field and I haven't made the appropriate update as yet (in practice the record id field will probably be removed and the link moved the the name field, after all that should be more meaningful to the user)

7. Short-term developments

1. Add validation

Basic validation will be added to ensure that at least the values accepted are of the correct form with later upgrading to fully implement any user-specified validations.

2. Support for all common data types/HTML types

Strictly speaking the initial Generv only handles text input boxes and full support for booleans – check boxes, enumerated lists - selection lists, dates and times will be added shortly.

3. Adding control for fields to be shown in list actions (and select)

Initially all fields are shown everywhere but in particular in the list and select (for the owned side of a relationship) the user should be able to specify the required fields, to keep the list short and meaningful

4. Adding control for field ordering on views

At the moment the order in all views is simply the order found in the model class but this should be user definable. If (3) above deals with the displayed list actions then in this case we will deal with the create, amend and show views.

5. Restricting create option for owned models

Where there is one to one or one to many relationships where the user accesses the owner they also have full create options for the owned side. There will be an attribute(s) added so that the user may be prohibited from creating the owned records and thus would be forced to only select from a list of existing records (e.g. if there was a one to one for 'colour' and rather than give the user the option of making up, say, somewhat fanciful names for the colour they could be restricted to only selecting from those hues already set up).

6. Offer overwrite option on file generation

Currently Generv just overwrites when it creates files, this will be replaced by a check and offer: 'file already exists, shall I overwrite or do you want to enter new file name or abandon ?'

7. Generalising Generv

Currently there is a number of hard-coded, shall we say, assumptions in Generv, some which need manual amendment, and these will be removed and, if necessary, replaced by a small configuration file.

Appendix

Installation

For the moment Generv contains hard-coding that assumes that it has been installed in the application's root directory. The installation and the manual changes given below will need to be made once only for each application.

Generv is supplied as a Zip containing:

```
/gener ------   generv.py

               /templates  ------   model.py
                                    smodel.py
                                    controller.py
                                    hasOne.py
                                    isOne.py
                                    hasManyShow.py
```

and should be installed in the root level of your project

Generv make use of a few standard Python libraries but these should not be an issue.

1. Dependencies

It assumes that jQuery is available and also needs HTML Kickstart (see www.99lime.com/elements/) and jBox (see http://stephanwagner.me/jBox/get_started), it seems best to install these in the project's /static sub-directory and you would normally add lines as below to the project's layout.html file:

```
<link rel="stylesheet" type="text/css" href="{{=URL('static','css/kickstart.css')}}"
media="all" />
<link rel="stylesheet" type="text/css" href="{{=URL('static','css/style.css')}}"
media="all" />
<link rel="stylesheet" type="text/css" href="{{=URL('static','css/jBox.css')}}"
media="all" />

<script type="text/javascript" src="{{=URL('static','js/kickstart.js')}}"></script>
<script type="text/javascript" src="{{=URL('static','js/jBox.js')}}"></script>
```

then you must manually amend the following files:

2. amend the new application's models/db.py to include

from gluon import current
current.db = db

say, at the end of the file. The model class methods use 'db'.

3. amend generv.py

at about line 263 where it references the 'extend' within the index.html

```
    index_str.append("""\
{{extend 's3layout.html'}}
<script>
```

inserting your choice of layout file in the 'extend' statement. This is the only place that the file is referred to, of course this layout file must be amended as in (1) above.

And finally:

4. copy smodel.py

Copy gener/templates/smodel.py into the application's /modules directory

And then you should be ready to use Generv.