Cromlech Documentation

Release .1

Dolmen & Cromlech team

April 29, 2012

CONTENTS

Contents:

CHAPTER

ONE

ABOUT CROMLECH

Cromlech is the namecode of a toolkit based on the ZCA, Martian and a tiny subset of Grok's grokcore packages.

The project and motivation was born from an extensive experience with Zope and Grok, in the domain of custom WEB application development.

The focus was : a very small and clean dependencies list, a traceable and understandable publication process, the use of well defined and precise structure that don't overdo their part. Long story short : the full control of the stack. The base philosophy is : spend your time coding, not trying to understand and parameter. If it's not there, do it yourself.

1.1 What is Cromlech ?

Cromlech is divided in small packages that are dedicated to defining components. To the Cromlech Team, Zope has shown the limit of the Component Architecture provoked by the shipping and deep intertwinement of the definition and the implementation. In order to deliver a truly pluggable and clean system, the Cromlech Team took the decision to strictly separate the implementation and the definition.

This is the reason to be of the 2 main Cromlech packages.

1.1.1 cromlech.io

cromlech.io defines the primitive and utmost basic entities of an application : the request, the response and the actors linked to it, namedly the publication and the application itself.

1.1.2 cromlech.browser

cromlech.browser is the backbone of a webapplication. It defines an extensive set of publisheable actors, from the forms and views to the sub-components, known in Zope/Grok as viewlets, viewletmanagers.

This package is more or less similar in the idea to *zope.browser*, that tried and failed in separating the component definition of the actual implementation. It failed by the lack of responsivity of the community and the already monolithic nature of most zope packages.

1.2 Does Cromlech do something else ?

Actually, yes, it does. The previous statement was not completly true : we do ship some implementations, but not along with the definition. Some Cromlech packages do provide framework-level components and others, merely utilities or 'fundamental' middlewares.

1.2.1 Unessential definitions

- cromlech.container

1.2.2 implementations

- cromlech.dawnlight
- cromlech.zodb
- cromlech.webob

1.2.3 middlewares

- cromlech.security
- cromlech.beaker

1.2.4 utilities

- cromlech.configuration
- cromlech.wsgi

1.3 Dolmen and the application-level components

1.3.1 dolmen.layout

x

1.3.2 dolmen.view

x

1.3.3 dolmen.viewlet

x

1.3.4 dolmen.message

x

1.3.5 dolmen.content

x

1.3.6 dolmen.menu

x

CHAPTER

TWO

CROMLECH DEMO

The cromlech demo is a good way to start to understand what cromlech provides.

It is quite a dense demo as it tries to show very different things that can be done with cromlech.

It was not conceived for normal framework users, but to developpers eagers to understand how the framework work and how it does not close you in a silo but tries to keeps setup and requirements simple at the same time.

2.1 Installation

Here is how to install CromlechDemo on a debian linux.

2.1.1 prerequisite

We need some basic packages:

```
$ apt-get install python python-dev python-virtualenv
```

2.1.2 Getting source

Clone repo and use branch zeam.setup:

```
$ git clone git://devel.dolmen-project.org/CromlechDemo.git
$ cd CromlechDemo
$ git checkout zeam.setup
```

We will do it in an isolated virtualenv to avoid conflicts:

```
$ virtualenv --no-site-packages .
$ . bin/activate
```

Note: if python 3 is your default python version you may use `-p /usr/bin/python2.7` (or python2.6)

Boostrap and build:

```
(CromlechDemo)$ python bootstrap.py
(CromlechDemo)$ setup install
```

2.1.3 Running

Now you can run the command that will launch the uWSGI server:

```
$ bin/demoapp
```

Now point your browser to http://localhost:8080/ and there you are !

2.2 This way, please!

Here is what you can see on the CromlechDemo.

First it is serve by uWSGI but could have been equally served by **Paste_** (easier at development time).

It use a simple Twitter Bootstrap based skin.

2.2.1 ZODB Application

The first page you land on is part of the ZODB application.

ZODB is an object database that helps building application with complex relations between entities without worrying about persitency too much.

There is not much to bee seen as the database is empty. But you can use *Add to folder* to add an Author. After adding an author you may add books using *Add to folder* on the author page. You can also edit or delete items.

The *Book list* shows you a simple JSON view, typical from what you could except from a webservice.

Also note that each item is rendered using a so called *view* (the central part) where as general menu, footer and so on are provided by a *layout*. The hello world banner is a so called *viewlet*, a fragment of view.

As you can see it's a hierarchical store. You can see that url follows the containment as a path in a filesystem. This is usually called a *traversing* based routing (you find stuff traversing objects).

2.2.2 Traject Application (SQL Database)

The upper menu permits you to go to the *traject application* located at http://localhost:8080/traject

This application use a SQL database (here a simple **sqlite_** one). Routing (finding the object to display at a certain url) is done using traject. ORM is SQLAlchemy.

Once again you are presented with an author / book database. You can, add, edit, delete items. See how urls now follows tables names.

2.2.3 Basic application

Basic page at http://localhost:8080/basic is there to show how cromlech plays well with wsgi only parts and does not require to enter in a particular model. It does not even use any layout, it's raw html.

2.2.4 Raw skin

A hidden feature is a raw view skin, a skin which as no layout, no menu, just raw content from the skin. This is accessible adding `++request_type++raw` at the begining of your path, as in http://localhost:8080/++request_type++raw/ It works everywhere.

Note that the links does not preserve the *raw* marker as this part of url would normally be hidden to user and added by the HTTP Server on, say, a raw.mysite.com address.

2.3 Diving into the code