Bungeni Technical Documentation

Release 0.0.1

Bungeni Dev Team, et al.

Nov 08, 2017

Contents

Welcome to The Bungeni Technical Documentation. Using this guide, a developer, systems administrator or systems analyst should be able to get a grasp of the many facets of Bungeni. The goal of this guide to provide concise, accurate and lots of information about Bungeni, how to use it and customize it.

CHAPTER 1

Getting Started

This part of the documentation focuses on explaining Bungeni, setting up your Python environment. It then proceeds with instructions on how to get Bungeni running seamlessly on the Plone CMS. Instructions on how to make server updates using tools like capistrano are included.

1.1 Picking an Interpreter

Which Python to use?

1.1.1 2.x vs 3.x

tl;dr: Python 2.x is the status quo, Python 3.x is the shiny new thing.

Further Reading

Today

If you're choosing a Python interpreter to use, I *highly* recommend you Use Python 2.7.x, unless you have a strong reason not to.

The Future

As more and more modules get ported over to Python3, the easier it will be for others to use it.

1.1.2 Which Python to Support?

If you're starting work on a new Python module, I recommend you write it for Python 2.5 or 2.6, and add support for Python3 in a later iteration.

1.1.3 Implementations

There are several popular implementations of the Python programming language on different back-ends.

CPython

CPython is the reference implementation of Python, written in C. It compiles Python code to intermediate bytecode which is then interpreted by a virtual machine. When people speak of *Python* they often mean not just the language but also this implementation. It provides the highest level of compatibility with Python packages and C extension modules.

If you are writing open-source Python code and want to reach the widest possible audience, targeting CPython is your best bet. If you need to use any packages that are rely on C extensions for their functionality (eg: numpy) then CPython is your only choice.

Being the reference implementation, all versions of the Python language are available as CPython. Python 3 is only available in a CPython implementation.

PyPy

PyPy is a Python interpreter implemented in a restricted statically-typed subset of the Python language called RPython. The interpreter features a just-in-time compiler and supports multiple back-ends (C, CLI, JVM).

PyPy aims for maximum compatibility with the reference CPython implementation while improving performance.

If you are looking to squeeze more performance out of your Python code, it's worth giving PyPy a try. On a suite of benchmarks, it's current over 5 times faster than CPython.

Currently PyPy supports Python 2.7.

Jython

Jython is a Python implementation that compiles Python code to Java byte code that then executes on a JVM. It has the additional advantage of being able to import and use any Java class the same as a Python module.

If you need to interface with an existing Java codebase or have other reasons to need to write Python code for the JVM, Jython is the best choice.

Currently Jython supports up to Python 2.5.

IronPython

IronPython is an implementation of Python for .NET framework. It can use both Python and .NET framework libraries, and can also expose Python code to other .NET languages.

Python Tools for Visual Studio integrate IronPython directly in to the Visual Studio development environment, making it an ideal choice for Windows developers.

IronPython supports Python 2.7.

- Properly Install Python

1.2 Installing Python on Mac OS X

The latest version of Mac OS X, Lion, **comes with Python 2.7 out of the box**.

You do not need to install or configure anything else to use Python. Having said that, I would strongly recommend that you install the tools and libraries described in the next section before you start building Python applications for real-world use. In particular, you should always install Distribute, as it makes it much easier for you to use other third-party Python libraries.

The version of Python that ships with OS X is great for learning, but it's not good for development. It's slightly out of date, and Apple has made significant changes that can cause hidden bugs.

1.2.1 Doing it Right

Let's install a real version of Python.

First, you'll need to have GCC installed to compile Python. You can either get this from XCode or the smaller OSX-GCC-Installer package.

While Lion comes with a large number of UNIX utilities, those familiar with Linux systems will notice one key component missing: a decent package manager. Homebrew fills this void.

To install Homebrew, simply run:

```
$ ruby -e "$(curl -fsS https://raw.github.com/gist/323731)"
```

Then, insert the hombrew directory at the top of your `PATH` enviornment variable. You can do this by adding the following line at the bottom of your `~/.bashrc` file:

```
export PATH=/usr/local/bin:$PATH
```

Now, we can install Python 2.7:

```
$ brew install python --framework
```

This will take a minute or two. Once that's complete, you'll have to add the new Python scripts directory to your `PATH`:

```
export PATH=/usr/local/share/python:$PATH
```

The `--framework` option tells Homebrew to compile a Framework-style Python build, rather than a UNIX-style build. The outdated version of Python that Snow Leopard comes packaged with is built as a Framework, so this helps avoid some future module installation bugs.

1.2.2 Distribute & Pip

The most crucial third-party Python software of all is Distribute, which extends the packaging and installation facilities provided by the distutils in the standard library. Once you add Distribute to your Python system you can download and install any compliant Python software product with a single command. It also enables you to add this network installation capability to your own Python software with very little work.

Hombrew already installed Distribute for you. Its `easy_install` command is considered by many to be deprecated, so we will install its replacement: **pip**. Pip allows for uninstallation of packages, and is actively maintained, unlike easy_install.

To install pip, simply open a command prompt and run:

```
$ easy_install pip
```

1.2.3 Virtualenv

After Distribute & Pip, the next development tool that you should install is virtualenv. Use pip:

```
$ pip install virtualenv
```

The virtualenv kit provides the ability to create virtual Python environments that do not interfere with either each other, or the main Python installation. If you install virtualenv before you begin coding then you can get into the habit of using it to create completely clean Python environments for each project. This is particularly important for Web development, where each framework and application will have many dependencies.

To set up a new Python environment, change the working directory to where ever you want to store the environment, and run the virtualenv utility in your project's directory:

```
$ virtualenv --distribute venv
```

To use an environment, run `source venv/bin/activate`. Your command prompt will change to show the active environment. Once you have finished working in the current virtual environment, run `deactivate` to restore your settings to normal.

Each new environment automatically includes a copy of `pip`, so that you can setup the third-party libraries and tools that you want to use in that environment. Put your own code within a subdirectory of the environment, however you wish. When you no longer need a particular environment, simply copy your code out of it, and then delete the main directory for the environment.

1.3 Installing Python on Windows

First, download the latest version of Python 2 from the official Website.

The Windows version is provided as an MSI package. To install it manually, just double-click the file. The MSI package format allows Windows administrators to automate installation with their standard tools.

By design, Python installs to a directory with the version number embedded, e.g. `C:\Python27\`, so that you can have multiple versions of Python on the same system without conflicts. Of course, only one interpreter can be the default application for Python file types. It also does not automatically modify the `PATH` environment variable, so that you always have control over which copy of Python is run.

Typing the full path name for a Python interpreter each time quickly gets tedious, so add the directories for your default Python version to the PATH. Assuming that your Python installation is in `C:\Python27\`, add this to your PATH:

```
C:\Python27\;C:\Python27\Scripts\
```

You can do this easily by running the following in `powershell`:

```
[Environment]::SetEnvironmentVariable("Path", "$env:Path;C:\Python27\;
↪C:\Python27\Scripts\", "User")
```

You do not need to install or configure anything else to use Python. Having said that, I would strongly recommend that you install the tools and libraries described in the next section before you start building Python applications for real-world use. In particular, you should always install Distribute, as it makes it much easier for you to use other third-party Python libraries.

1.3.1 Distribute + Pip

The most crucial third-party Python software of all is Distribute, which extends the packaging and installation facilities provided by the distutils in the standard library. Once you add Distribute to your Python system you can download and install any compliant Python software product with a single command. It also enables you to add this network installation capability to your own Python software with very little work.

To obtain the latest version of Distribute for Windows, run the python script available here:
http://python-distribute.org/distribute_setup.py

You'll now have a new command available to you: **easy_install**. It is considered by many to be deprecated, so we will install its replacement: **pip**. Pip allows for uninstallation of packages, and is actively maintained, unlike easy_install.

To install pip, simply open a command prompt and run:

```
> easy_install pip
```

1.3.2 Virtualenv

After Distribute & Pip, the next development tool that you should install is virtualenv. Use pip:

```
> pip install virtualenv
```

The virtualenv kit provides the ability to create virtual Python environments that do not interfere with either each other, or the main Python installation. If you install virtualenv before you begin coding then you can get into the habit of using it to create completely clean Python environments for each project. This is particularly important for Web development, where each framework and application will have many dependencies.

To set up a new Python environment, change the working directory to where ever you want to store the environment, and run the virtualenv utility in your project's directory:

```
> virtualenv --distribute venv
```

To use an environment, run the `activate.bat` batch file in the `Scripts` subdirectory of that environment. Your command prompt will change to show the active environment. Once you have finished working in the current virtual environment, run the `deactivate.bat` batch file to restore your settings to normal.

Each new environment automatically includes a copy of `pip` in the `Scripts` subdirectory, so that you can setup the third-party libraries and tools that you want to use in that environment. Put your own code within a subdirectory of the environment, however you wish. When you no longer need a particular environment, simply copy your code out of it, and then delete the main directory for the environment.

This page is a remixed version of another guide, which is available under the same license.

1.4 Installing Python on Linux

The latest version of Ubuntu, **comes with Python 2.7 out of the box**.

You do not need to install or configure anything else to use Python. Having said that, I would strongly recommend that you install the tools and libraries described in the next section before you start building Python applications for real-world use. In particular, you should always install Distribute, as it makes it much easier for you to use other third-party Python libraries.

1.4.1 Distribute & Pip

The most crucial third-party Python software of all is Distribute, which extends the packaging and installation facilities provided by the distutils in the standard library. Once you add Distribute to your Python system you can download and install any compliant Python software product with a single command. It also enables you to add this network installation capability to your own Python software with very little work.

To obtain the latest version of Distribute for Linux, run the python script available here:
http://python-distribute.org/distribute_setup.py

The new‘‘easy_install‘‘ command you have available is considered by many to be deprecated, so we will install its replacement: **pip**. Pip allows for uninstallation of packages, and is actively maintained, unlike easy_install.

To install pip, simply open a command prompt and run:

```
$ easy_install pip
```

1.4.2 Virtualenv

After Distribute & Pip, the next development tool that you should install is virtualenv. Use pip:

```
$ pip install virtualenv
```

The virtualenv kit provides the ability to create virtual Python environments that do not interfere with either each other, or the main Python installation. If you install virtualenv before you begin coding then you can get into the habit of using it to create completely clean Python environments for each project. This is particularly important for Web development, where each framework and application will have many dependencies.

To set up a new Python environment, change the working directory to where ever you want to store the environment, and run the virtualenv utility in your project's directory:

```
$ virtualenv --distribute venv
```

To use an environment, run `source venv/bin/activate`. Your command prompt will change to show the active environment. Once you have finished working in the current virtual environment, run `deactivate` to restore your settings to normal.

Each new environment automatically includes a copy of `pip`, so that you can setup the third-party libraries and tools that you want to use in that environment. Put your own code within a subdirectory of the

environment, however you wish. When you no longer need a particular environment, simply copy your code out of it, and then delete the main directory for the environment.

This page is a remixed version of another guide, which is available under the same license.

- Properly Install Plone

1.5 Installing Plone (UNIX-based environments)

The latest version of Ubuntu, **comes with Python 2.7 out of the box**.

You do not need to install or configure anything else to use Python. Having said that, I would strongly recommend that you install the tools and libraries described in the next section before you start building Python applications for real-world use. In particular, you should always install Distribute, as it makes it much easier for you to use other third-party Python libraries.

1.5.1 Distribute & Pip

The most crucial third-party Python software of all is Distribute, which extends the packaging and installation facilities provided by the distutils in the standard library. Once you add Distribute to your Python system you can download and install any compliant Python software product with a single command. It also enables you to add this network installation capability to your own Python software with very little work.

To obtain the latest version of Distribute for Linux, run the python script available here:
http://python-distribute.org/distribute_setup.py

The new‘‘easy_install‘‘ command you have available is considered by many to be deprecated, so we will install its replacement: **pip**. Pip allows for uninstallation of packages, and is actively maintained, unlike easy_install.

To install pip, simply open a command prompt and run:

```
$ easy_install pip
```

1.5.2 Virtualenv

After Distribute & Pip, the next development tool that you should install is virtualenv. Use pip:

```
$ pip install virtualenv
```

The virtualenv kit provides the ability to create virtual Python environments that do not interfere with either each other, or the main Python installation. If you install virtualenv before you begin coding then you can get into the habit of using it to create completely clean Python environments for each project. This is particularly important for Web development, where each framework and application will have many dependencies.

To set up a new Python environment, change the working directory to where ever you want to store the environment, and run the virtualenv utility in your project's directory:

```
$ virtualenv --distribute venv
```

To use an environment, run `source venv/bin/activate`. Your command prompt will change to show the active environment. Once you have finished working in the current virtual environment, run `deactivate` to restore your settings to normal.

Each new environment automatically includes a copy of `pip`, so that you can setup the third-party libraries and tools that you want to use in that environment. Put your own code within a subdirectory of the environment, however you wish. When you no longer need a particular environment, simply copy your code out of it, and then delete the main directory for the environment.

CHAPTER 2

Installing Bungeni

2.1 About Bungeni

2.1.1 Explaining Bungeni's existence

Bungeni 2.0 is a parliamentary and Legislative Information System that aims at making Parliaments more open and accessible to citizens virtually allowing them "inside Parliament" or "Bungeni" the Kiswahili word for "inside Parliament".

Bungeni is a collaborative software development initiative based on open standards, AKOMA NTOSO and open source applications that provieds a leading solution for drafting, managing, consolidating and publishing legislative and other parliamentary documents.

Bungeni is an initiative of "Africa i-Parliament Action plan" a programme of UN/DESA.

2.2 Getting the Source Code

2.2.1 Using Subversion

You can get access to Bungeni's stable release with subversion.

Getting source code

Here's what getting the source code using subversion involves:

```
svn checkout http//bungeni-portal.googlecode.com/svn/bungeni.main/trunk bugeni-portal-
↪read-only
```

2.3 Getting Help

For more information or to ask questions about Bungeni, check out:

- Bungeni Documentation
- IRC: #bungeni on freenode.net

CHAPTER 3

Architecture

3.1 Software Architecture Overview

3.1.1 Scope

The scope of this section is to help a developer to understand the deployment configuration and to provide an understanding of how to modify the existing features and to create new ones. The system is composed of many different modules and each requires knowledge of the used technology, this section provides a global vision and does not attempt to try to explain everything.

3.1.2 Description

To describe the architecture of Bungeni, UML will be used. Some snippets of code will be provided where necessary to describe bash file configurations, python code, paster and deliverance configuration files.

3.2 Software Requirements

3.2.1 Operating System

Bungeni is easily run on Linux using the Ubuntu distro (11.04 and 10.04, 10.04 being recommended).

3.2.2 Middleware

Main Components

The following section describes the more programmatic views used by the Bungeni Portal.

Capistrano

Capistrano is a utillity and framework for executing commands in parallel on multiple remote machines via SSH. It uses a simple DSL (*Domain Specific Language*) that allows you to define *tasks*, which may be applied to machines in certain roles. It also supports tunnelling connections vai some gateway machine to allow operations to be performed behind VPN's and firewalls.

Capistrano was originally designed to simplify and automate deployment of web applications to distributed environments, and originally came bundled with a set of tasks designed for deploying Rails applications. Read the docs.

Supervisor

Supervisor is a client/server system that allows its users to monitor and control a number of processes on UNIX-like operating systems. It is responsible for starting programs at its own invocation, responding to commands from clients, restarting crashed or exited subprocesses, logging its subprocess `stdout` and `stderr` output, and generating and handling *events* corresponding to points in subprocess lifetimes. It provides a web user interface to view and control process status.

The above two components are part of the deployment system.

Paster

Python Paste is a set of libraries to deploy WSGI applications, it covers all aspect of a CGI appliation: testing, dispatcher, authentication, debugging and deployment. Specifically the Paste Deployment is a system for finding and configuring WSGI applications and servers, it provides a single, simple function (loadapp) for loading a WSGI application from a configuration file or a Python Egg. The usual way to deploy a WSGI application is to use `paster serve`, this command line counterpart to serve an application using a *Paste Deploy* configuration file.

Deliverance

Deliverance is a tool to theme HTML, applying a consistent style to applications and static files regardless of how they are implemented, and separating site-wide styling from application-level templating. Deliverance takes the HTML from a source then applies a *theme* to the HTML using something similar to XSLT transforms (but without restrictions).

Zope 2 and 3

Zope is an open source web application server primarily written in the Python programming language. It features a transactional object database which can store ot only content and custom data, but also dynamic HTML templates, scripts, a search engine, and relational database (RDBMS) connection and code. It features a strong through-the-web development model, allowing you to update your website from anywhere in the world. Zope also features a tightly integrated security model.

BlueBream or Zope 3 is a rewrite by Zope developers of the Zope web application server. It is built on top of the Zope Tool Kit (**ZTK**). The project tries to create a more developer-friendly and flexible platform for programming web applications. The original intent of BlueBream was to become a replacement for Zope 2, however this did not happen as planned. Instead Zope 2 continued to make up the majority of new Zope deployments, mostly due to the popularity of Plone. At this moment, many component of BlueBream are used in Zope 2 and in other frameworks, and the community uses the term ZTK to define a set of libraries used as a basis for Zope3, Zope2, bfg and Grok frameworks.

The main innovation of BlueBream is the *comonent architecture*, which allows structuring code into small, composable units with introspectable interfaces, configurable through the ZCML files.

PostgresSQL

PostgresSQL is a powerful, open source object relational database system (**ORDBMS**), developed at the University of California at Berkeley Computer Science Department. It has more than 15 years of active development and a proven architecture that has earned it a strong reputation for reliability, and correctness. As an enterprise class database, PostgresSQL boasts of sophisticated features such as Multi-Version Concurrency Control (**MVCC**), point in time recovery, tablespaces, asynchronous replication, nested transactions (savepoints), online/hot backups, a sophisticated query planner/optimizer, and write ahead logging for fault tolerance.

SQLAlchemy

SQLAlchemy is the Python SQL toolkit and Object Relational Mapper (**ORM**) that gives application developers the full power and flexibility of SQL. It provides a full suit of well known enterprise-level persistence patterns, designed for efficient and high-performing database access, adapted into a simple and Pythonic DSL (*domain specific language*). SQLAlchemy doesn't view database as a collection of tables; it sees them as relational algebra engines. Its ORM enables classes to be mapped against database tables in more than one way. SQL constructs don't just select from just tables (you can also select from joins, subqueries, and uniions). Thus database relationships and domain object models can be cleanly decoupled from the beginning.

Xapian

Xapian is an Open Source Search Engine Library. It is a highly adaptable toolkit which allows developers to easily index and integrate advanced search facilities to their own applications. It supporst the Probabilistic Information Retrieval model and also supporst a rich set of boolean query operators.

Plone

Plone is a powerful, flexible Content Management System (CMS) that is easy to install, use and extend. Plone lets non-technical people create maintain information using only a web browser. The main use of Plone is a basis for websites or intranets because of its modular nature which helps the customization of all aspects. Plone is a product that runs on the Zope 2 application server, so it shares the core functionalities like a components-based architecture, security and scalability.

Theming component

3.2.3 Theming Component & Dispatcher

The system uses Deliverance to add a common theme to BungeniPortal and BungeniCMS. Deliverance is integrated in teh *paster* middleware, so it is a *WSGI* application. Usually these type of applications are referred as pipeline components. Deliverance receives responses from the applications mapped in the *dispatch* section then transform the HTML on the basis of the `rules.xml` file. In `portal/deploy.ini` there is:

```
[pipeline:main]

pipeline = deliverance

dispatch
```

The paster application is launched from *supervisord* , see `supervisord.conf` in the section [`program:portal`].

The configuration for Deliverance is:

```
[filter:deliverance]

use = egg:bungeni.portal #deliverance

## use rule_file_host here since that's the internal server:port for deliverance rule_
↪uri =

http://%(rule_file_host)s/static/themes/rules.xml
```

The paster configuration "egg:bungeni.portal#deliverance" is related to the declaration in the `setup.py` of bungeni.portal egg:

```
entry_points = """

[paste.filter_app_factory]

deliverance = bungeni.portal.middleware:make_deliverance_middleware

[paste.app_factory]

static = bungeni.portal.app:make_static_serving_app
"""
```

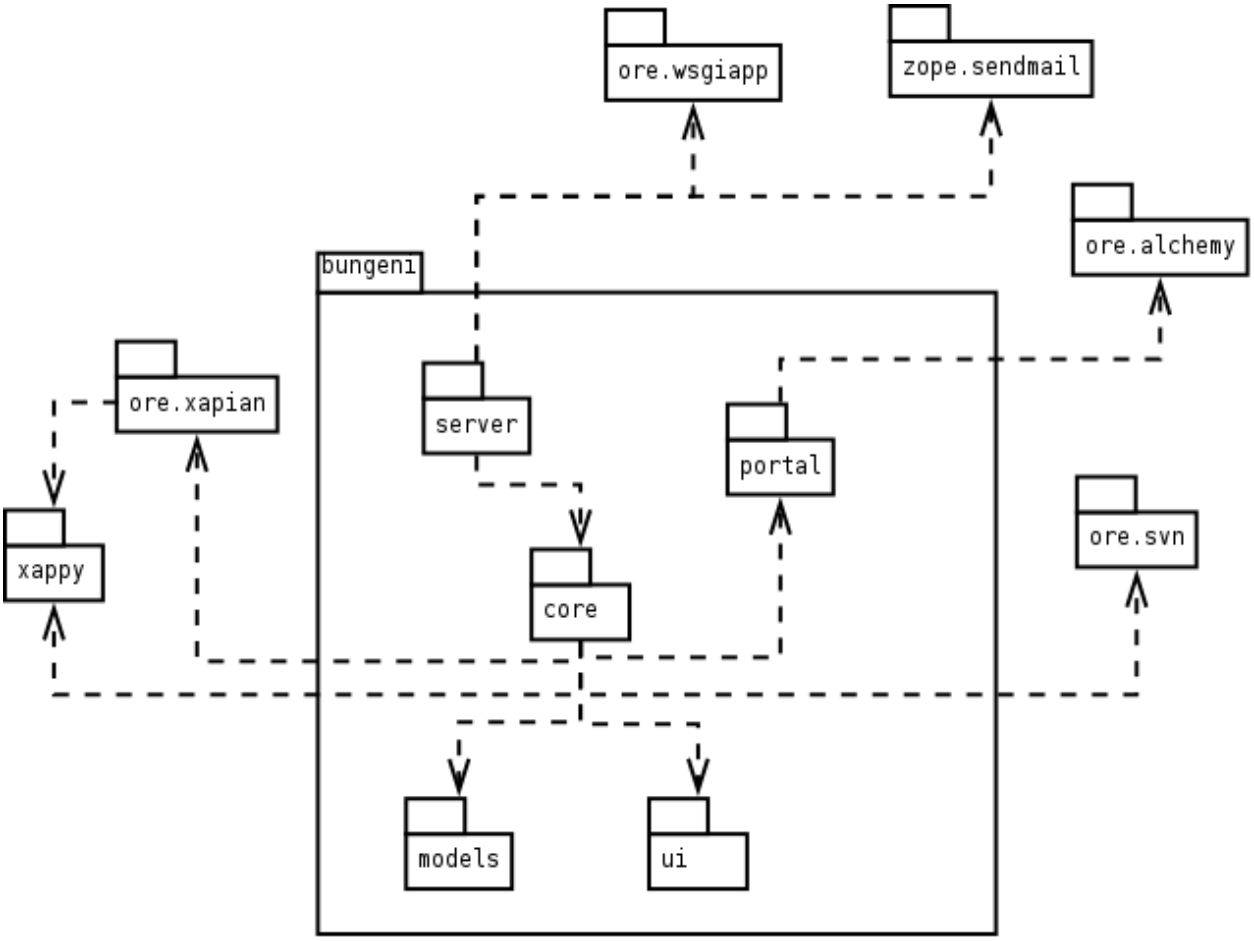
`make_deliverance_middleware` is the factory method generating the real Deliverance app. When instantiated the `rule_uri` parameter is passed to the factory.

Dispatcher

"egg:Paste#urlmap" is a standard component of the Paste framework. It maps the URLs to applications providing the same features of a rewrite rule or proxy rule in Apache. For more information about `urlmap` refer to: `http://pythonpaste.org/deploy <http://pythonpaste.org/deploy>`

Bungeni Portal

This diagram shows the main components of BungeniPortal: Diagram 5: Packages in BungeniPortal



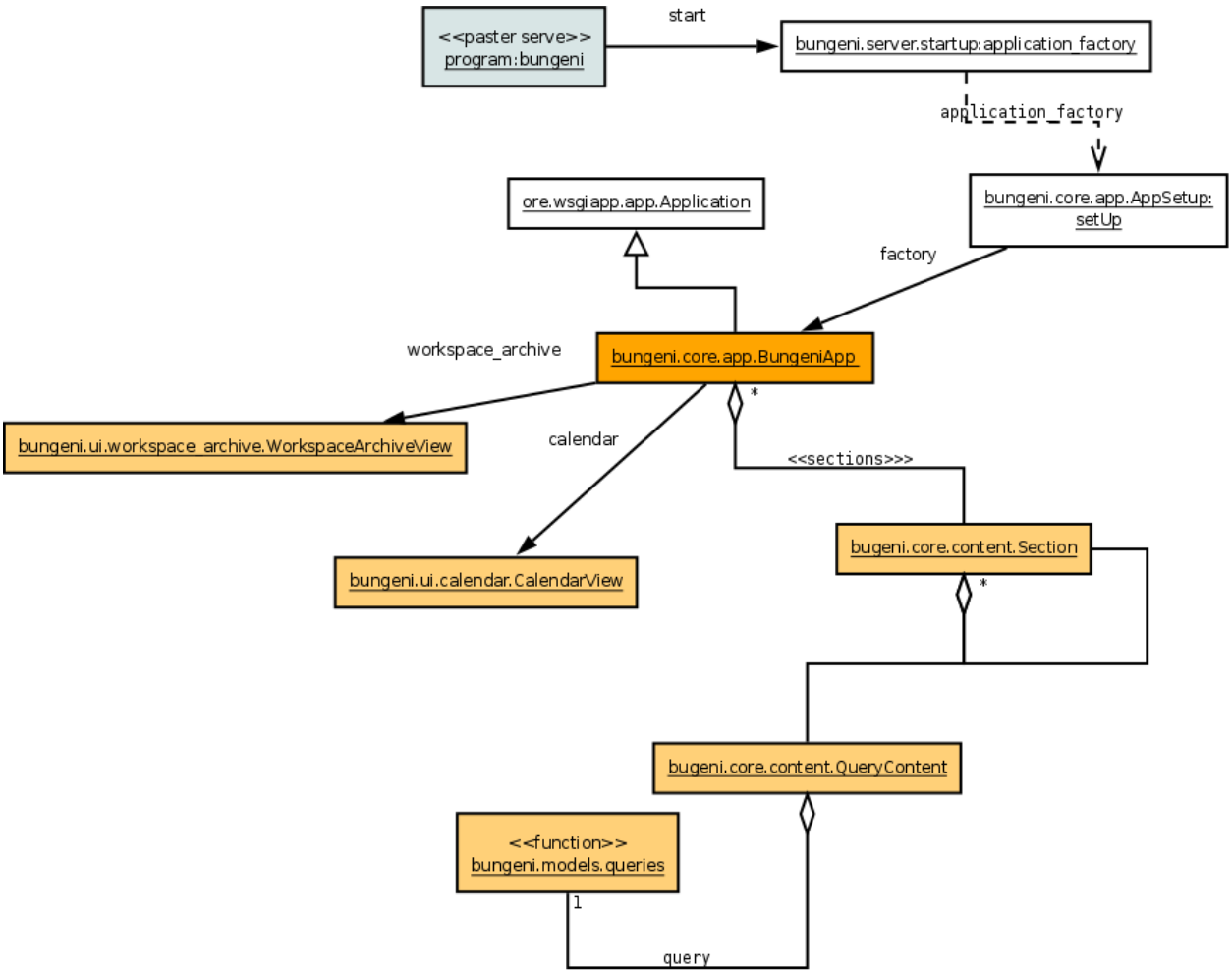
bungeni.server

The application is based on Zope3; this package contains the configuration of the libraries (what is included and excluded to reduce the startup time) and the function `application_factory` that is used by `paster serve` command to launch the application; it depends on `ore.wsgiapp` that allows bootstrapping a Zope3 environment as a *wsgi* application without a ZODB backend. This package contains also the utility **SMTP Mailer** used for sending e-mail.

bungeni.core

This package contains the application started from the `application_factory` entry point of `bungeni.server` and the contents creating the sections of the portal. The following diagram shows the main classes involved:

Diagram 6: bungeni.core interation



bungeni.core.app

The main class is AppSetup that is the factory adapter for the BungeniApp (IBungeniApplication). As the name stated it setups the application:

- create indexes for each content and add these to the `indexer` object that warps around Xapian i.e. using a file system storage for the index catalog:

```
<buildoutpath>
/parts/index
```

- add to the application object the names bound to the functionalities. The application context is a dictionary-like object so for example the 'business' link is added as a key:

```
business = self.context['business'] = Section( title=_(u"Business"),
                        description=_(u"Daily operations of the parliament."),
                        default_name = u"whats-on")
```

The sections are based on four types of classes.

- `bungeni.core.content.Section`: is an `OrderedContainer`, a Zope3 class modelling a folder in which the contents container are maintained in order. For example, 'Business', 'Members', 'Archive' are Section contents. Note that usually the OrderedContainers are Persistent objects (in Zope sense) but in this case they are not stored at all.

- `bungeni.core.content.QueryContent`: a function that performs a SQL query is attached to this object, see `bungeni.model.queries` module. For example the "committees" and "bills" under business are QueryContent::

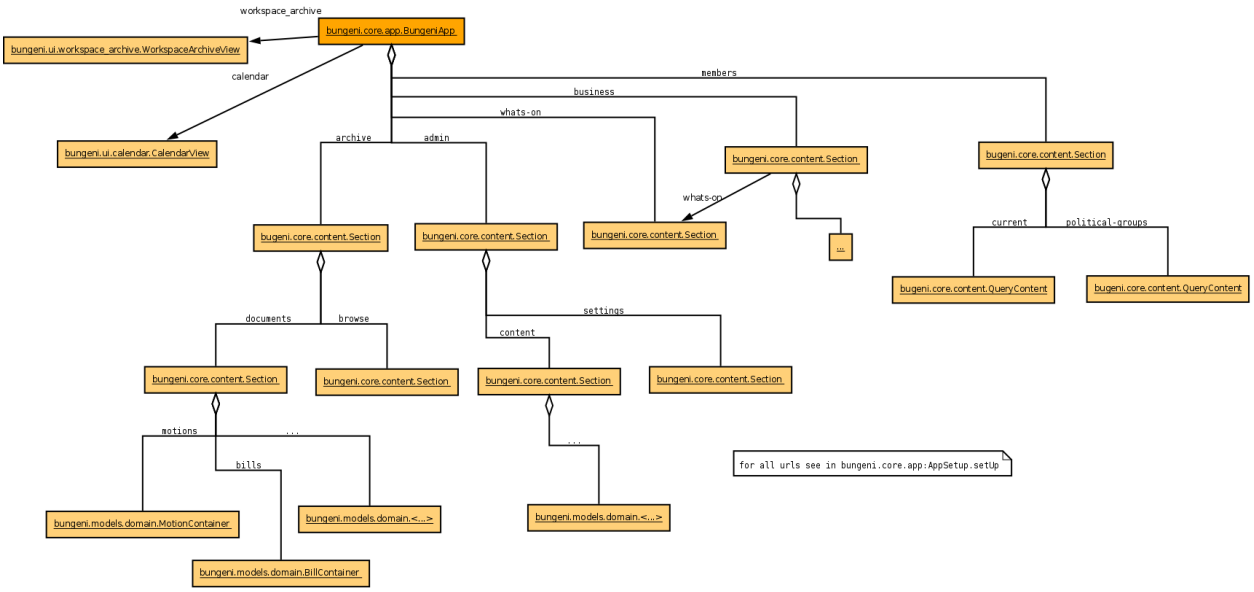
```
business[u"committees"] = QueryContent(
            container_getter(get_current_parliament, 'committees'),
            title = _(u"Committees"),
            marker = interfaces.ICommitteeAddContext,
            description = _(u"View committees created by the current parliament.
↪"))
```

- `bungeni.ui.calendar.CalendarView`: is a browser page that provides a calendar, see `bungeni.ui`
- `bungeni.ui.workspace.archive.WorkspaceArchiveView`: is the user/member workspace, see `bungeni.ui`

Below is a tree that shows the contents based on `bungeni.models.domain`, they are the objects that mapped to tables and rows in the RDBMS and accesed through the SQLAlchemy ORM. For example the section '*bills*' is a BillContainer, a folderish object, and contains `Bill` from `bungeni.models.domain`.

Note `domain.Container` is autogenerated by SQLAlchemy. Here is a partial diagram showing the objects and the relations with URLs.

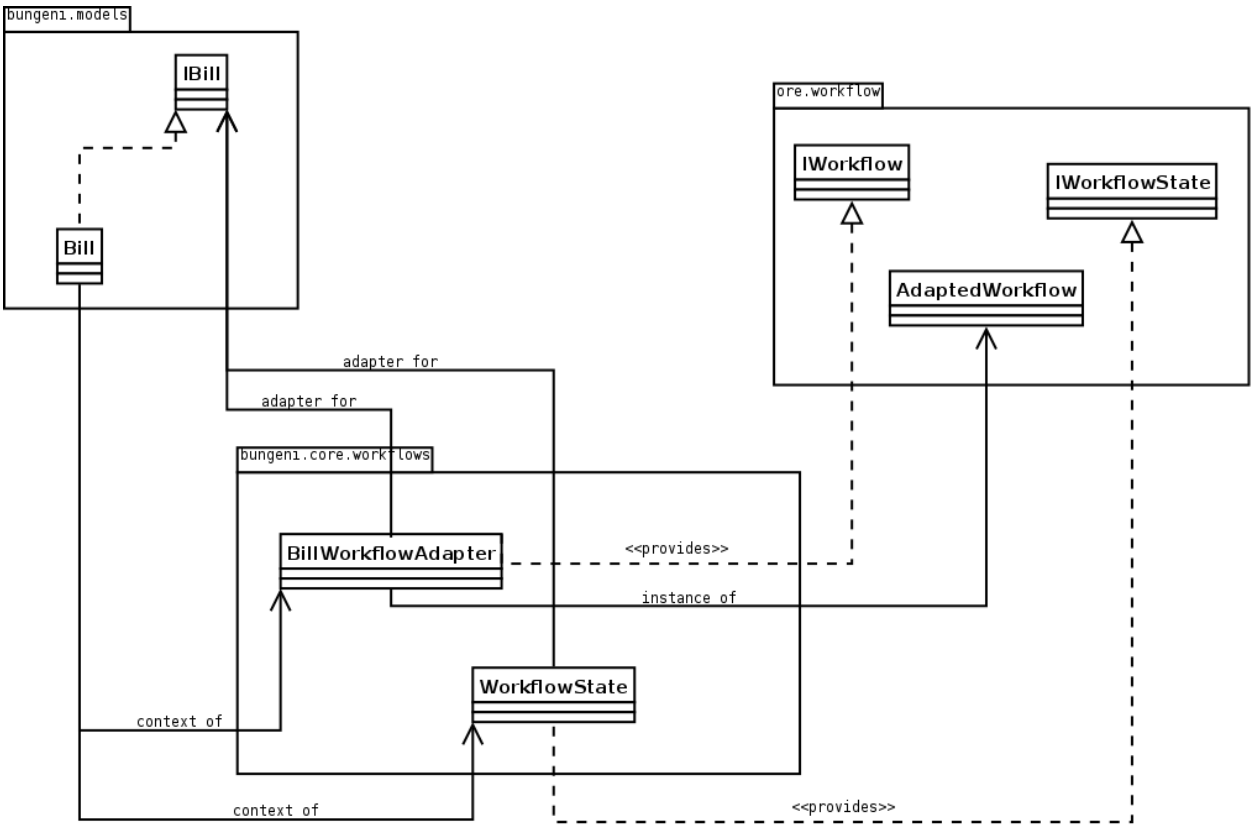
Diagram 7: Bungeni package integration



bungeni.core.workflows

In `bungeni.core.workflows` there are configurations, factories and definitions of workflows used in the site. A workflow is tied to a `bungeni.model.domain` content through a configuration basd on the interface. An example of the implmentation for `Bill` content looks like (see: `bungeni.models`):

Diagram 8: Workflows in `bungeni.core`



In `configure.zcml` of `bungeni.core.workflows` for Bill there is:

```
<adapter

for = "bungeni.models.interfaces.IBill"
provides = "ore.workflow.interfaces.IWorkflow"
factory = ".adaptors.BillWorkflowAdapter" />
```

This means that `BillWorkflowAdapter` is the constructor of workflow for the class implementing the `IBill` interface. The operation is done through the `load_workflow` method passed to the `AdaptedWorkflow` class (*not shown in the diagram*), it reads the `bill.xml` file containing the description of the workflow in terms of states and transitions, and then generates the workflow object. In a similar way, the state of workflow is managed by the `WorkflowState` class, it provides the access to the state attribute in a `Bill` object; with this attribute the engine is able to determine the possible transitions to other states.

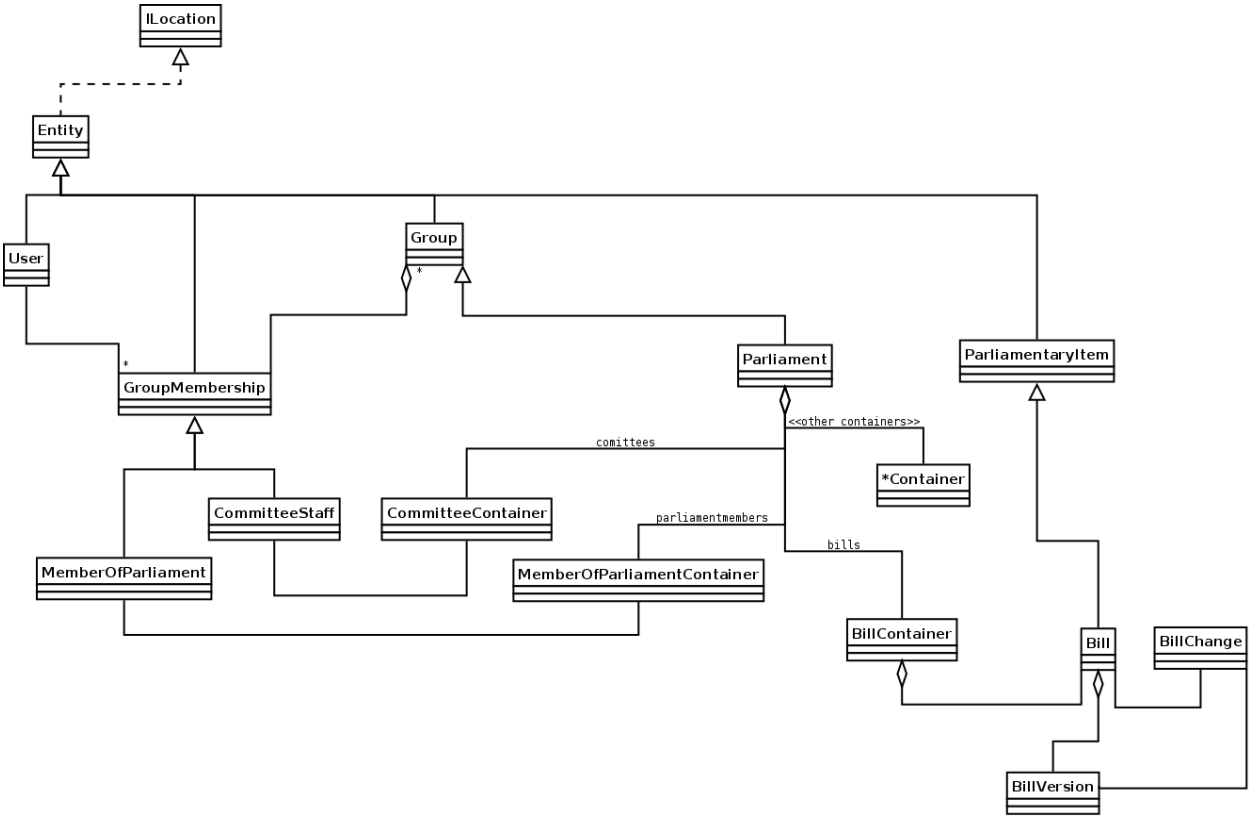
For an explanation about entity based workflow engines see: http://www.zope.org/Members/hathawsh/DCWorkflow_docs/default/DCWorkflow_doc.pdf and `workflow.txt` in http://pypi.python.org/pypi/ore.workflow package.

bungeni.models

The main module of this package is 'domain', the module is rather complicated so this document reports only part of the inner classes to show the general structure. The base class is `Entity`: the main scope is to provide `ILocation` interface that is used to declare the parent and the name of the object inside this parent container. The second important class is `Parliament`, the root of the system, contains the containers of committees, members, bills etc.

Below a partial view of the module:

Diagram 9: `bungeni.models`



As example of implementations on Parliament object there is the 'bills' name that is a `BillContainer` container `Bills` objects. The `Bill` has two other classes associated:

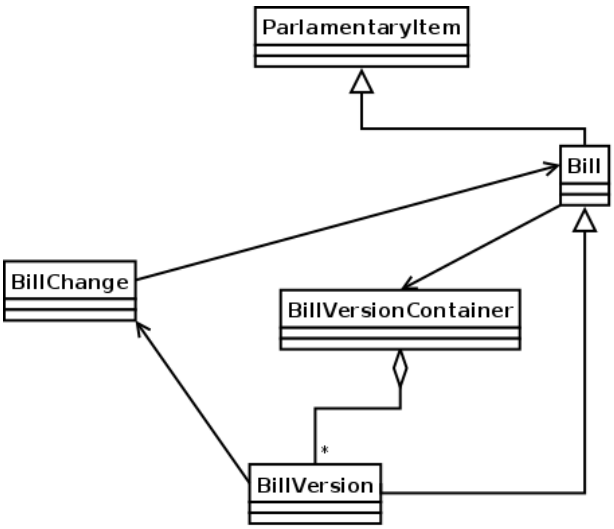
- `BillVersion`: it is a reference to a `Bill` object
- `BillChange`: it is an operation done on an object

The tables for `Bill`, `BillVersion` and `BillChange` are generated in `bungeni.models.schema` module, e.g.:

```
bills = rdb.Table("bills", metadata,
        rdb.Column("bill_id", rdb.Integer, rdb.ForeignKey('parliamentary_items.
↪parliamentary'), primary_key=True),
        rdb.Column("bill_type_id", rdb.Integer, rdb.ForeignKey('bill_types.bill_type_
↪id'), nullable = False),
        rdb.Column("ministry_id", rdb.Integer, rdb.ForeignKey('groups.group_id')),
        rdb.Column("identifier", rdb.Integer),
        rdb.Column("summary", rdb.UnicodeText),
        rdb.Column("publication_date", rdb.Date),
        )
bill_changes = make_changes_table(bills, metadata)
bill_versions = make_versions_table(bills, metadata, parliamentary_items)
```

The table for versions and changes are generated through the methods `make_changes_table` and `make_version_table`, they create new tables with specific fields, in particular `bills_versions` contains the columns of the `parliament_items` table. Below the relation between these classes (excerpt from classes and RDB table definitions):

Diagram 10: Relation between content type tables



Class Diagram

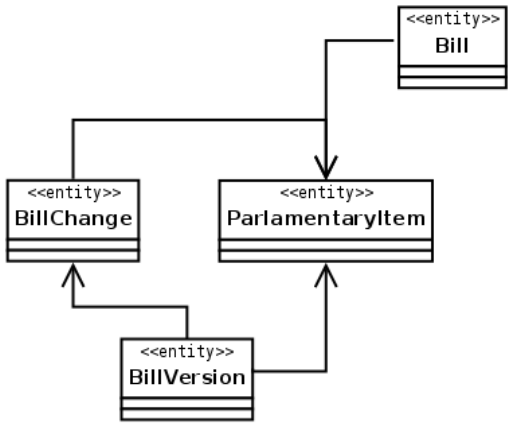


Table diagram

Each `Bill` object is a `ParliamentaryItem` in terms of RDB, this means that for each row in `Bill` table, a row is created in `ParliamentaryItem` table. If you modify a Bill object you are modifying a `ParliamentaryItem` row, then the old values of this record are copied in a new `BillVersion` row and it generates a new `BillChange` row.

bungeni.ui

The following packages are provide the interface for various parts of the BungeniPortal:

- forms package: content and container broswer views
- calendar pakcage: contains the code to manage events associated to parliamentary items.
- `workspaces.py` (and other modules): manage the access to items of a member for parliament.
- `workflow.py` (and other modules): manage the interface to access the workflow functionalities.
- `versions.py` (and others): provide the functionalities to access the item versions.

Bungeni CMS

At the moment of writing this document, Bungeni is powered by vanilla Plone with a minimal setup for the IA (*Information Architecture*). Some portal sections are folder but in the final integration with Deliverance they are mapped on BungeniPortal URLs. Refer to http://www.plone.org for more information about Plone. The version used is 3.3.3.

3.3 Minimal Hardware Requirements

3.3.1 Processor

3.3.2 Memory

3.3.3 Disk Space

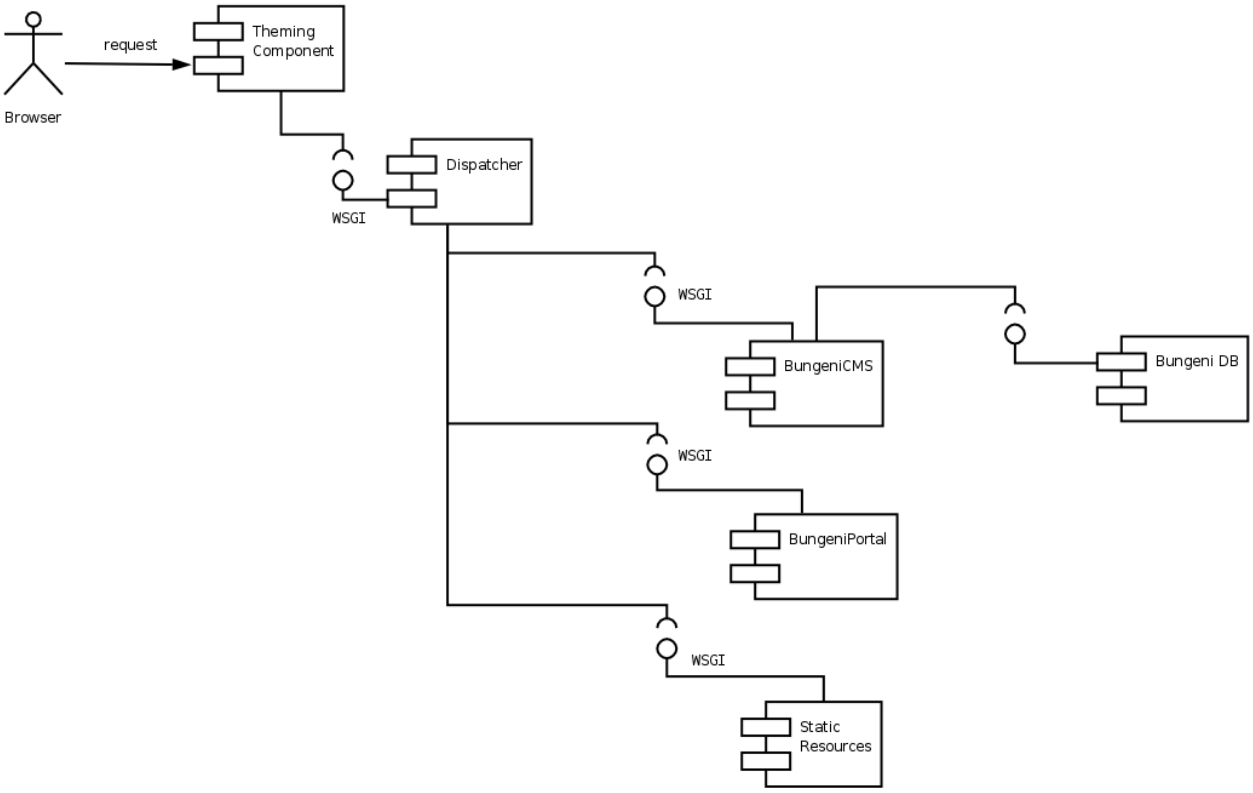
3.4 Logical Views

3.4.1 Logical Elements

The main logical elements are described below:

- Theming component: this provides a coherent look and feel to the BungeniPortal and BungeniCMS.
- Dispatcher redirects the incoming requests to the correct URL inside the main applications.
- BugneniPortal provides the parliament functionalities.
- BugeniDB is a relational DB, it stores the data of BungeniPortal
- BungeniCMS contains the general materials, it manages the content that is stored in the system. BungeniCMS uses an integrated object DB (not shown in the diagram)
- Static resources are the images and CSS files which are served straight from the server's file system.

Diagram 1: Logical Elements



Theme Delivering

This component provides a coherent look-and-feel across all the applications in the site. The HTML coming from a source (e.g. Bungeni Portal or BungeniCMS) is re-written based on a "theme" which is a static HTML page: the component extracts the parts from the page and fills the empty spaces in the static template. This operation is done using a set of rules based on an XPath syntax.

Dispatcher

This component simply calls the application using a mapping between URLs and apps.

BungeniPortal

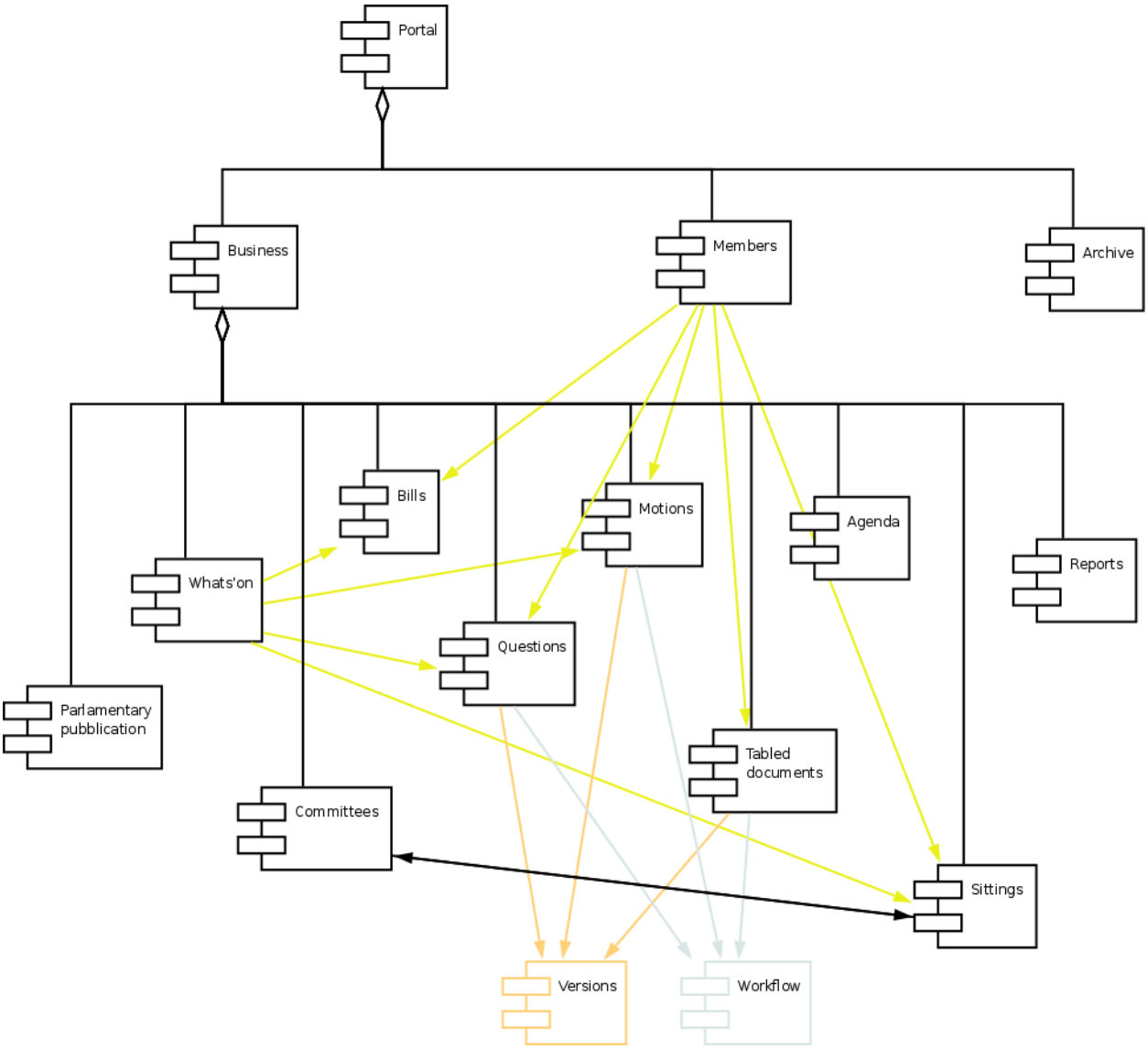
This is the application that provides the specific parliament features, it can be broken up into the following sections:

- *Business:* in this area there are the daily operations of the various parliament activities.
- *What's on:* an overview of the daily operations of the parliament
- *Committees:* list of committees, for each one there is metadata about the matter of discussion, membership and sittings.
- *Bills:* list of bills and metadata for each bill. Actions are provided to version the bill and access the workflow associated with the bill.

- *Questions:* list of questions and associated metadata about the matter of discussion, membership and sittings.
- *Motions:* list of motions and associated metadata. Workflow and versioning actions are provided.
- *Tabled documents:* list of tabled documents and metadata. Workflow and versioning actions are provided.
- *Agenda items:* this is a list of agenda items and metadata.
- *Sittings:* calendar showing the sittings of the plenary and the committees.
- *Parliamentary publications:* this is a list of publications and information; these publications are the reports that come out of sittings.
- *Members:* in this section, one can search for information about members of parliament (MPs)
- *Member of parliament:* general information such as name and election date.
- *Personal Info:* a complete biography of the member.
- *Offices held:* information about offices in which the member has a title
- *Parliamentary activities:* a list of content workflows the member has participated in. e.g. questions created by the member or motions moved by the member.
- *Archive:* access to current and historical activities of the parliament, the categories are:
  - Parliaments
  - Political groups
  - Committees
  - Governments
- *Workspace:* This is available for members of parliament and for clerks. This provides access to to the most relevant and current information for the user in a single page. e.g. for the Member of Parliament - the following
- *Administration:* This is an administration section provided to the Admin. This is used for adding parliaments, new users, closing parliaments, entering preliminary metatdata etc.

The following diagram shows the logical components of the BungeniPortal.

Diagram 2: Logical Components of BungeniPortal



The versions and workflow functionalities provide traversals into the content; for example in a motion there are the links to past workflow states and older versions of the motion - allowing the user to browse not just the current state of the motion but also the full audited history of the motion.

From some sections a user can reach contents in other sections. An example of this is the "Parliamentary activities" tab of a member, it is possible to take a look at a bill moved by that MP (member of parliament).

BungeniCMS

This is the content management system part of the portal, it provides a set of functionalities which are designed to:

- Allow a large number of people of contribute to and share stored contents
- Control access to contents, based on permissions
- User roles or group membership define what each user can do (not only edit and view).
- Improve communication between users using comments on contents.
- Publication workflow and versioning support

The CMS contains various contents: documents, events, news, pictures, files are the main types. The information architecture is organized in a tree structure, at this moment it looks as:

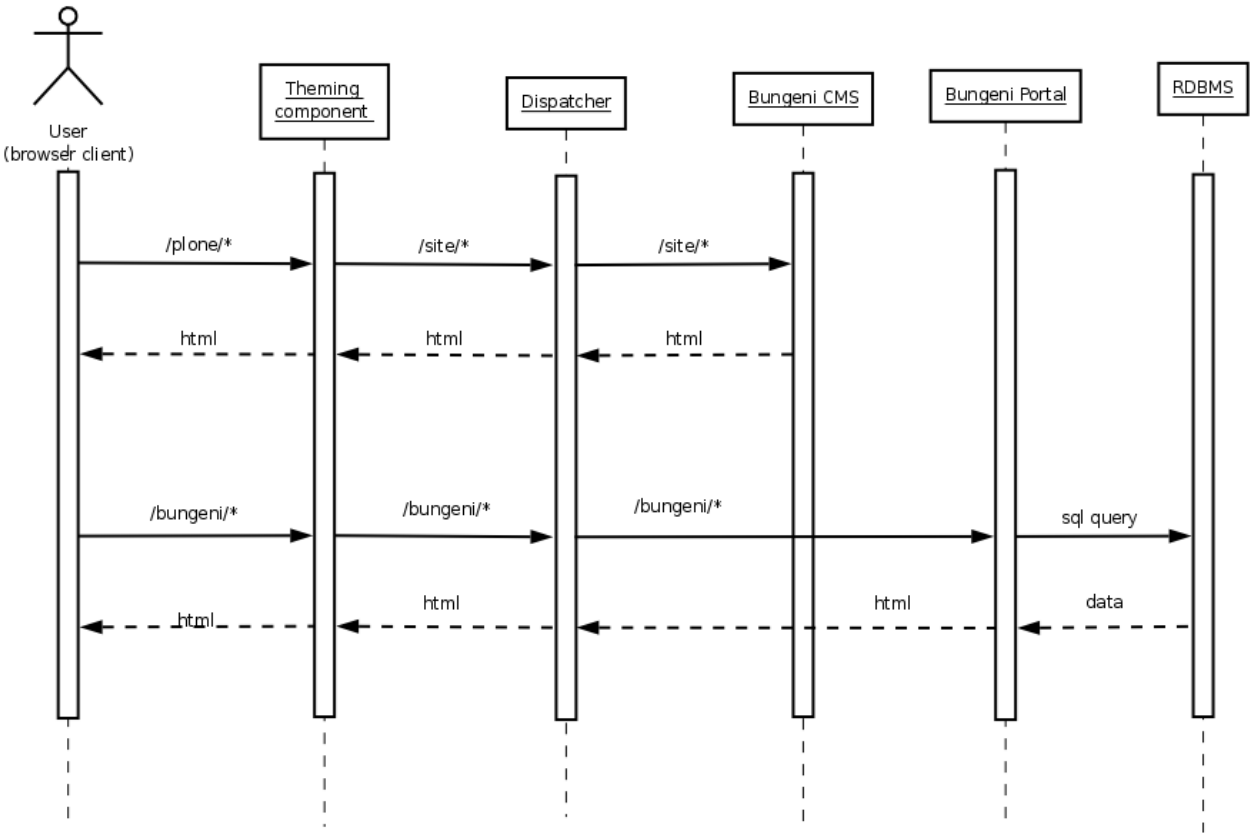
- How we work
- Rules and regulations
- How parliament works
- Seating plan
- Administrative
- Reference material
- History of parliament
- Online resources
- Useful links
- Picture gallery
- Have your say
- Vote in the election
- Become an member of parliament
- Present a petition
- Visit parliament

This is the base structure but subject to changes due to the specificity of each parliaments.

3.4.2 Logical relationships

The following diagram shows how the different parts of the system communicate with each other:

Diagram 3: Logical Relationships



The request is passed from the 'theming component' to the dispatcher that call the designated application; the returned html is processed from 'theming component' and release to the user. In this diagram is missing the 'paster server' component that provides main access to the web server and manage the *wsgi* messages among the parts. As shown the components are for the most not dependent upon each other: the 'theming component' an the Dispatcher merge backend applications, the BungeniCMS can work without the others as the BungeniPortal (in this case however there is an explicit need for the RDBMS to store and retrive data).

3.4.3 Deployment of logical elements to hardware components

The starter point is the *supervisord* configuration: supervisord.conf (a file with structure similar to Microsoft Windows INI files). From this file you can see which services compose the system and how they are started:

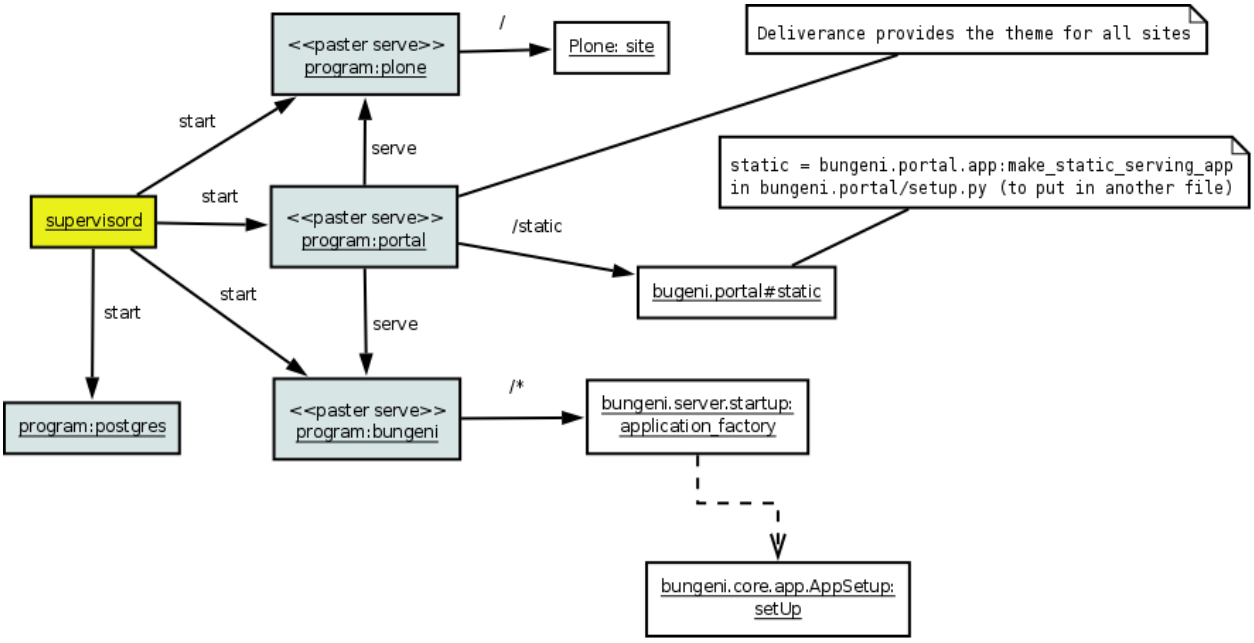


Diagram 4: Deployment

The sections are:

- program:portal
- program:plone
- program:bungeni
- program:postgres
- program:openoffice

program:portal

Specify how all web servers and applications are reachable, they are served through Paster. Paster is a two-level command and the second level is pluggable, for Bungeni *serve* that the `serve` command is used, which is a script to *serve* applications based on the WSGI interface (*similar to CGI*) using the http protocol (see http://pythonpaste.org/script/developer.html). The configuration of *portal* is in portal/deploy.ini: the *main* section defines a pipeline which filters requests through deliverance and serving a lot of urls: see [pipeline:main] then [filter:deliverance] and [composite:dispatch] sections. Deliverance provides a uniform theme to all applications (**'http://deliverance.openplans.org/index.html'**_), it intercepts the pages and via a set of rules applies a common look-and-feel. IN the *dispatch* section you can see the url mapped:

- / = plone
- /plone = plone

these are provided from server specified in [program:plone] of supervisord.conf

- /static = static

this is provided directly as a *wsgi* service from the module bungeni.portal#static

program:bungeni

Bungeni Portal is served through paster with `deploy.ini`, the mapped urls are `'/'` and `'/cache'`. On `'/'` there is the real portal, a pipeline of `repoze.who` (WSGI authentication middleware) and bungeni.server (code in src/bungeni.server). This one uses `ore.wsgiapp` and `site.zcml` is the Zope 3 instance configuration.

The names provided from bungeni sections are managed from `bungeni.ui`: the names/urls are implemented as browser pages or menu actions: see the configuration in `src/bungeni.ui/bungeni/ui/menu.zcml` (this requires understanding of zcml zope technology).

program:plone

BungeniCMS is based on Plone and it is served through paster (that is unusual for Plone) with the configuration file `plone/etc/deploy.ini`. The paster configuration is a pipeline of various middleware at end of which there is Zope2. The BungeniCMS is the *site* instance of Plone in the root of Zope.

program:postgres

The configuration to start up the PostgresSQL database server.

3.5 Architecture Polices and Mechanisms

3.5.1 Policies and Mechanisms

General Policies and Mechanisms

Localization (i18n)

For localization, *zope.i18n* library is used. This package provides a way to register translations files for domains and languages. An example of this library being used in Python is shown here

```
from zope.i18nmessageid import MessageFactory
_ = MessageFactory = MessageFactory('bungeni.ui')
#
# some more code
#
title = _(u'label_translate', default=u'Language:')
```

or

```
from zope.i18n import translate
#
# more code
#
text = translate(self.body)
```

See the multilingual seciont for use in page templates and the format of translations file.

Multilingual UI or Templating

The multilingual feature for the UI is provided by the i18n feature integrated in the Zope Page Template system. For example in `bungeni/ui/viewlets/templates/whatson.pt` there is::

```
<div metal:fill-slot="body" i18n:domain="bungeni.ui">
<h1><a href="/business" i18n:translate="">What's on</a></h1>
```

i18n:domain specifies the translation domain because the same string can be present in different domains. The *i18n:translate* command instruct the engine to translate the text in the tag based on the browser language. The translations are provided from `.po` files stored in the software packages. For example, for `bungeni.ui` there is the folder `locales/en/LC_MESSAGES` containing the domain `bungeni.ui.po` and the compiled version

bungeni.ui.mo. The format of files is very simple. It contains a metadata part and then the translations in the form of (message id, message string) couple::

```
"Project-Id-Version: \n"

"POT-Creation-Date: Thu Jan 14 13:13:54 2010\n"

"PO-Revision-Date: 2010-01-14 13:26+0300\n"

"Last-Translator: Christian Ledermann <christian@parliaments.info>\n" "Language-Team:␣
↪\n"

"MIME-Version: 1.0\n"

"Content-Type: text/plain; charset=UTF-8\n"

"Content-Transfer-Encoding: 8bit\n"

#

#: src/bungeni.ui/bungeni/ui/audit.py:20

msgid "action"

msgstr "Action"

...
```

As stated above the templating system is based on Zope Page Templates, refer to http://zpt.sourceforge.net for an overview of the software. The main feature of this templating system is the separation between the namespace of commands and the namespace of HTML, this way the HTML is still valid and the TAL (Template Attribute Language) commands aren't interfering with it it, for example::

```
<title tal:content="here/title">Page Title</title>
```

where *tal:content* rewrite the tag text. For further explanation: http://docs.zope.org/zope2/zope2book/ZPT.htmlhttp://docs.zope.org/zope2/zope2book/ZPT.html

Storage

For storing the data of BungeniPortal, a well known RDB databases, PostgresSQL, was chosen. The BungeniCMS, on the other hand, uses the usual Zope setup with ZODB, a tested Objec oriented database.

Security

Both BungeniPortal and BungeniCMS are based on Zope, it uses a strong authentication/authorization system based on permissions and roles. The programmer can assign a permission to a function or a method of a class using the zcml configuration or with functions (as in Plone) such as declareProtected, declarePrivate and declarePublic. After that the programmer creates a set of roles and assigns the permissions to roles, then the roles are assigned to users and groups, either in zcml or through the ZMI interface in Plone (the setup is stored on file system). The publisher of Zope checks the user permissions before rendering a page or calling a method, raising an Unauthorized exception in case the user hasn't got the required permission.

Information Architecture

The information architecture is very parliament specific.

System management and monitoring

Supervisord is used to manage the server's services (status, start, stop and restart).

Policies and Mechanisms for Quality Requirements

There are no policies or mechanisms for quality requirements right now.

3.6 References

- Bungeni http://www.bungeni.org
- repoze.bfg: http://bfg.repoze.org
- Paster: http://pythonpaste.org/ and http://pythonpaste.org/script/
- Zope: http://www.zope.org
- Buildout: http://pypi.python.org/pypi/zc.buildout
- Plone: http://www.plone.org
- Deliverance: http://deliverance.openplans.org/philosophy.html
- Postgres: http://www.postgresql.org/
- SQLAlchemy: http://www.sqlalchemy.org/
- Subversion: http://subversion.tigris.org/
- Xapian: http://xapian.org/
- supervisord: http://supervisord.org/
- **Capistrano:**
  - http://www.capify.org/index.php/Capistrano
  - http://www.capify.org/index.php/Getting_Started

CHAPTER 4

Systems Administration

4.1 Database Server Administration

4.1.1 Installing a Database Server (PostgresSQL)

4.1.2 Using and Administering PostgresSQL

4.2 Fabric

Fabric is a library for simplifying system administration tasks. Fabric is more focused on application level tasks such as deployment.

Install Fabric:

```
$ pip install fabric
```

The following code will create two tasks that we can use: `memory_usage` and `deploy`. The former will output the memory usage on each machine. The latter will ssh into each server, cd to our project directory, activate the virtual environment, pull the newest codebase, and restart the application server.

```
from fabric.api import cd, env, prefix, run, task

env.hosts = ['my_server1', 'my_server2']

@task
def memory_usage():
    run('free -m')

@task
def deploy():
    with cd('/var/www/project-env/project'):
        with prefix('. ../bin/activate'):
```

```
            run('git pull')
            run('touch app.wsgi')
```

With the previous code saved in a file named fabfile.py, we can check memory usage with:

```
$ fab memory_usage
[my_server1] Executing task 'memory'
[my_server1] run: free -m
[my_server1] out:              total     used     free   shared  buffers   cached
[my_server1] out: Mem:          6964     1897     5067        0      166      222
[my_server1] out: -/+ buffers/cache:     1509     5455
[my_server1] out: Swap:            0        0        0

[my_server2] Executing task 'memory'
[my_server2] run: free -m
[my_server2] out:              total     used     free   shared  buffers   cached
[my_server2] out: Mem:          1666      902      764        0      180      572
[my_server2] out: -/+ buffers/cache:      148     1517
[my_server2] out: Swap:          895        1      894
```

and we can deploy with:

```
$ fab deploy
```

Additional features include parallel execution, interaction with remote programs, and host grouping.

4.2.1 Bungeni Fabric files

Bungeni fabric deployment files can be found in the following repositories:

- https://github.com/bungeni/bungeni-fab
- http://bungeni-portal.googlecode.com/svn/fabric/branches/

4.3 Using Capistrano

CHAPTER 5

Customizing Bungeni and Writing Awesome Code

This part of the guide focuses on best practices for writing Python code.

5.1 Structuring Your Project

Structuring your project properly is extremely important.

Todo

Fill in "Structuring Your Project" stub

5.1.1 Structure is Key

5.1.2 Vendorizing Dependencies

5.1.3 Runners

5.1.4 Further Reading

5.2 Code Style

5.2.1 Idioms

Idiomatic Python code is often referred to as being *pythonic*.

Zen of Python

Also known as PEP 20, the guiding principles for Python's design.

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

See http://stackoverflow.com/questions/228181/the-zen-of-python for some examples.

PEP 8

PEP 8 is the de-facto code style guide for Python.

> PEP 8

There exists a command-line program, *pep8* that can check your code for conformance.

```
pip install pep8
```

Simply run it on a file or series of files and get a report of any violations

```
$ pep8 optparse.py
optparse.py:69:11: E401 multiple imports on one line
optparse.py:77:1: E302 expected 2 blank lines, found 1
optparse.py:88:5: E301 expected 1 blank line, found 0
optparse.py:222:34: W602 deprecated form of raising exception
optparse.py:347:31: E211 whitespace before '('
optparse.py:357:17: E201 whitespace after '{'
optparse.py:472:29: E221 multiple spaces before operator
optparse.py:544:21: W601 .has_key() is deprecated, use 'in'
```

Conforming your style to PEP 8 is generally a good idea and helps make code a lot more consistent when working on projects with other developers.

5.3 Documenting Your Code

Documenting your code is extremely important. It is debatebly even more important than testing.

5.3.1 The Basics

Code Comments

Information regarding code comments is taken from PEP 008 (http://www.python.org/dev/peps/pep-0008/). Block comment styling should be used when commenting out multiple lines of code.:

```
Block comments generally apply to some (or all) code that follows them,
and are indented to the same level as that code.  Each line of a block
comment starts with a # and a single space (unless it is indented text
inside the comment).
Paragraphs inside a block comment are separated by a line containing a
single #.
```

Inline comments are used for individual lines and should be used sparingly.:

```
An inline comment is a comment on the same line as a statement.  Inline
comments should be separated by at least two spaces from the statement.
They should start with a # and a single space.
Inline comments are unnecessary and in fact distracting if they state
the obvious.  Don't do this:
    x = x + 1                 # Increment x
But sometimes, this is useful: ::
    x = x + 1                 # Compensate for border
```

Doc Strings

PEP 257 is the primary reference for docstrings. (http://www.python.org/dev/peps/pep-0257/)

There are two types of docstrings, one-line and multi-line. Their names should be fairly self explanatory. One-line docstrings:

```
def kos_root():
    """Return the pathname of the KOS root directory."""
    global _kos_root
    if _kos_root: return _kos_root
    ...
```

Multi-line docstrings:

```
def complex(real=0.0, imag=0.0):
    """Form a complex number.

    Keyword arguments:
    real -- the real part (default 0.0)
    imag -- the imaginary part (default 0.0)

    """
    if imag == 0.0 and real == 0.0: return complex_zero
    ...
```

Sphinx

Sphinx is a tool which converts documentation in the *reStructuredText* markup language into a range of output formats including HTML, LaTeX (for printable PDF versions), manual pages and plain text.

Note: This Guide is built with Sphinx

reStructuredText

Most Python documentation is written with reStructuredText. The reStructuredText Primer and the reStructuredText Quick Reference should help you familiarize yourself with its syntax.

5.3.2 Other Tools

that old thing

pocco / docco / shocco

Ronn

5.4 Testing Your Code

Testing your code is very important.

5.4.1 The Basics

Unittest

Unittest is the batteries-included test module in the Python standard library. Its API will be familiar to anyone who has used any of the JUnit/nUnit/CppUnit series of tools.

Creating testcases is accomplished by subclassing a TestCase base class

```
import unittest

def fun(x):
    return x + 1

class MyTest(unittest.TestCase):
    def test(self):
        self.assertEqual(fun(3), 4)
```

As of Python 2.7 unittest also includes its own test discovery mechanisms.

unittest in the standard library documentation

Doctest

The doctest module searches for pieces of text that look like interactive Python sessions, and then executes those sessions to verify that they work exactly as shown.

5.4.2 Tools

py.test

py.test is a no-boilerplate alternative to Python's standard unittest module.

```
$ pip install pytest
```

Despite being a fully-featured and extensible test tool it boasts a simple syntax. Creating a test suite is as easy as writing a module with a couple of functions

```
# content of test_sample.py
def func(x):
    return x + 1

def test_answer():
    assert func(3) == 5
```

and then running the *py.test* command

```
$ py.test
=========================== test session starts ============================
platform darwin -- Python 2.7.1 -- pytest-2.2.1
collecting ... collected 1 items

test_sample.py F

================================= FAILURES =================================
_____________________________________ test_answer _____________________________________

    def test_answer():
>       assert func(3) == 5
E       assert 4 == 5
E        +  where 4 = func(3)

test_sample.py:5: AssertionError
========================== 1 failed in 0.02 seconds ========================
```

far less work than would be required for the equivalent functionality with the unittest module!

py.test

Nose

nose extends unittest to make testing easier.

```
$ pip install nose
```

nose provides automatic test discovery to save you the hassle of manually creating test suites. It also provides numerous plugins for features such as xUnit-compatible test output, coverage reporting, and test selection.

nose

tox

tox is a tool for automating test environment management and testing against multiple interpreter configurations

```
$ pip install tox
```

tox allows you to configure complicatated multi-parameter test matrices via a simple ini-style configuration file.

tox

Unittest2

unittest2 is a a backport of Python 2.7's unittest module which has an improved API and better assertions over the one available in previous versions of Python.

If you're using Python 2.6 or below, you can install it with pip

```
$ pip install unittest2
```

You may want to import the module under the name unittest to make porting code to newer versions of the module easier in the future

```
import unittest2 as unittest

class MyTest(unittest.TestCase):
    ...
```

This way if you ever switch to a newer python version and no longer need the unittest2 module, you can simply change the import in your test module without the need to change any other code.

unittest2

5.5 Choosing a License

Open source.

Todo

Fill in License stub

5.5.1 Non-Restrictive

PSFL

MIT / BSD / ISC

MIT (X11)

New BSD

ISC

Apache

5.5.2 Restrictive

LGPL

GPL

GPLv2

GPLv3

CHAPTER 6

Development Guide

This part of the guide focuses on tool and module advice based on different scenarios and workflows availed when you use the Bungeni portal.

6.1 Database Schema diagram

6.2 The Authentication Stack

6.3 Model Layer

6.3.1 The models in the application

6.3.2 Roles

6.3.3 Workflows

6.3.4 Workspaces

6.4 Views

6.4.1 URL Scheme

6.4.2 View functions

6.4.3 Custom views

6.4.4 Viewlet managers

6.5 Forms

6.5.1 Form actions and menus

6.5.2 Validation

6.6 Templating

6.7 Sessions

6.8 Caching

6.9 Internationalization and Localization

6.10 API Guide

6.10.1 Core modules in Bungeni

CHAPTER 7

Testing Guide

This part of the guide documents how tests are run on Bungeni.

7.1 How to run Bungeni unit tests

7.2 How to do automated functional and integration tests

7.3 Troubleshooting

CHAPTER 8

Development Environment

8.1 Your Development Environment

8.1.1 Text Editors

Just about anything which can edit plain text will work for writing Python code, however, using a more powerful editor may make your life a bit easier.

VIM

There exist a couple of plugins and settings for the VIM editor to aid python development. If you only develop in Python, a good start is to set the default settings for indentation and linewrapping to values compliant with PEP8:

```
set textwidth=79
set shiftwidth=4
set tabstop=4
set expandtab
set softtabstop=4
set shiftround
```

With these settings newlines are inserted after 79 characters and indentation is set to 4 spaces per tab. If you also use VIM for other languages, there is a handy plugin at indent, which handles indentation settings for python source files. Additionally there is also a handy syntax plugin at syntax featuring some improvements over the syntax file included in VIM 6.1.

These plugins supply you with a basic environment for developing in Python. However in order to improve the programming flow we also want to continually check for PEP8 compliance and check syntax. Luckily there exist PEP8 and Pyflakes to do this for you. If your VIM is compiled with *+python* you can also utilize some very handy plugins to do these checks from within the editor. For PEP8 checking install vim-pep8. Now you can map the vim function *Pep8()* to any hotkey or action you want. Similarly for pyflakes you can install vim-pyflakes. Now you can map *Pyflakes()* like the PEP8 function and have it called quickly. Both plugins will display errors in a quickfix list and provide an easy way to jump to the corresponding line. A very handy setting is calling these functions whenever a buffer is saved. In order to do this, enter the following lines into your vimrc:

```
autocmd BufWritePost *.py call Pyflakes()
autocmd BufWritePost *.py call Pep8()
```

Todo

add supertab notes

TextMate

"TextMate brings Apple's approach to operating systems into the world of text editors. By bridging UNIX underpinnings and GUI, TextMate cherry-picks the best of both worlds to the benefit of expert scripters and novice users alike."

Sublime Text

"Sublime Text is a sophisticated text editor for code, html and prose. You'll love the slick user interface and extraordinary features."

Sublime Text has excellent support for editing Python code and uses Python for its plugin API.

Sublime Text 2 is currently in beta.

8.1.2 IDEs

PyCharm / IntelliJ IDEA

PyCharm is developed by JetBrains, also known for IntelliJ IDEA. Both share the same code base and most of PyCharm's features can be brought to IntelliJ with the free Python Plug-In.

Eclipse

The most popular Eclipse plugin for Python development is Aptana's PyDev.

Komodo IDE

Komodo IDE is developed by ActiveState and is a commerical IDE for Windows, Mac and Linux.

Spyder

Spyder an IDE specifically geared toward working with scientific python libraries (namely Scipy). Includes integration with pyflakes, pylint, and rope.

Spyder is open-source (free), offers code completion, syntax highlighting, class and function browser, and object inspection.

8.1.3 Interpreter Tools

virtualenv

Virtualenv is a tool to keep the dependencies required by different projects in separate places, by creating virtual Python environments for them. It solves the "Project X depends on version 1.x but, Project Y needs 4.x" dilemma and keeps your global site-packages directory clean and manageable.

virtualenv creates a folder which contains all the necessary executables to contain the packages that a Python project would need. An example workflow is given.

Install virtualenv:

```
$ pip install virtualenv
```

Create a virtual environment for a project:

```
$ cd my_project
$ virtualenv venv
```

`virtualenv venv` will create a folder in the currect directory which will contain the Python executable files, and a copy of the `pip` library which you can use to install other packages. The name of the virtual environment (in this case, it was `venv`) can be anything; omitting the name will place the files in the current directory instead.

In order the start using the virtual environment, run:

```
$ source venv/bin/activate
```

The name of the current virtual environment will now appear on the left of the prompt (e.g. `(venv)Your-Computer:your_project UserName$`) to let you know that it's active. From now on, any package that you install using `pip` will be placed in the venv folder, isolated from the global Python installation. Install packages as usual:

```
$ pip install requests
```

To stop using an environment simply type `deactivate`. To remove the environment, just remove the directory it was installed into. (In this case, it would be `rm -rf venv`).

Other Notes

Running `virtualenv` with the option `--no-site-packages` will not include the packages that are installed globally. This can be useful for keeping the package list clean in case it needs to be accessed later.

In order to keep your environment consistent, it's a good idea to "freeze" the current state of the environment packages. To do this, run

```
$ pip freeze > requirements.txt
```

This will create a `requirements.txt` file, which contains a simple list of all the packages in the current environment, and their respective versions. Later, when a different developer (or you, if you need to re- create the environment) can install the same packages, with the same versions by running

```
$ pip install -r requirements.txt
```

This can help ensure consistency across installations, across deployments, and across developers.

Lastly, remember to exclude the virtual environment folder from source control by adding it to the ignore list.

virtualenvwrapper

Virtualenvwrapper makes virtualenv a pleasure to use by wrapping the command line API with a nicer CLI.

```
$ pip install virtualenvwrapper
```

Put this into your *~/.bash_profile* (Linux/Mac) file:

```
$ export VIRTUALENVWRAPPER_VIRTUALENV_ARGS='--no-site-packages'
```

This will prevent your virtualenvs from relying on your (global) site packages directory, so that they are completely separate..

8.1.4 Other Tools

IPython

IPython provides a rich toolkit to help you make the most out of using Python interactively. Its main components are:

- Powerful Python shells (terminal- and Qt-based).
- A web-based notebook with the same core features but support for rich media, text, code, mathematical expressions and inline plots.
- Support for interactive data visualization and use of GUI toolkits.
- Flexible, embeddable interpreters to load into your own projects.
- Tools for high level and interactive parallel computing.

```
$ pip install ipython
```

BPython

bpython is an alternative interface to the Python interpreter for Unix-like operating systems. It has the following features:

- In-line syntax highlighting.
- Readline-like autocomplete with suggestions displayed as you type.
- Expected parameter list for any Python function.
- "Rewind" function to pop the last line of code from memory and re-evaluate.
- Send entered code off to a pastebin.
- Save entered code to a file.
- Auto-indentation.
- Python 3 support.

```
$ pip install bpython
```

8.2 Virtual Environments

A Virtual Environment, put simply, is an isolated working copy of Python which allows you to work on a specific project without worry of affecting other projects.

For example, you can work on a project which requires Django 1.3 while also maintaining a project which requires Django 1.0.

8.2.1 virtualenv

virtualenv is a tool to create isolated Python environments.

Install it via pip:

```
$ pip install virtualenv
```

Basic Usage

1. Create a virtual environment:

```
$ virtualenv venv
```

This creates a copy of Python in whichever directory you ran the command in, placing it in a folder named `venv`.

2. To begin using the virtual environment, it needs to be activated:

```
$ source venv/bin/activate
```

You can then begin installing any new modules without affecting the system default Python or other virtual environments.

3. If you are done working in the virtual environment for the moment, you can deactivate it:

```
$ deactivate
```

This puts you back to the system's default Python interpreter with all its installed libraries.

To delete a virtual environment, just delete its folder.

After a while, though, you might end up with a lot of virtual environments littered across your system, and its possible you'll forget their names or where they were placed.

8.2.2 virtualenvwrapper

virtualenvwrapper provides a set of commands which makes working with virtual environments much more pleasant. It also places all your virtual environments in one place.

To install (make sure **virtualenv** is already installed):

```
$ pip install virtualenvwrapper
$ export WORKON_HOME=~/Envs
$ source /usr/local/bin/virtualenvwrapper.sh
```

(Full virtualenvwrapper install instructions.)

Basic Usage

1. Create a virtual environment:

```
$ mkvirtualenv venv
```

This creates the `venv` folder inside `~/Envs`.

2. Work on a virtual environment:

```
$ workon venv
```

virtualenvwrapper provides tab-completion on environment names. It really helps when you have a lot of environments and have trouble remembering their names. `workon` also deactivates whatever environment you are currently in, so you can quickly switch between environments.

3. Deactivating is still the same:

```
$ deactivate
```

4. To delete:

```
$ rmvirtualenv venv
```

Other useful commands

`lsvirtualenv` List all of the environments.

`cdvirtualenv` Navigate into the directory of the currently activated virtual environment, so you can browse its `site-packages`, for example.

`cdsitepackages` Like the above, but directly into `site-packages` directory.

`lssitepackages` Shows contents of `site-packages` directory.

Full list of virtualenvwrapper commands.

CHAPTER 9

Tutorials

CHAPTER 10

Additional Notes

This part of the guide, which is mostly prose, begins with some background information about Python, then focuses on next steps.

10.1 Introduction

Todo

write a general blurb introducing the Python language

10.1.1 About This Guide

Purpose

The Hitchhiker's Guide to Python exists to provide both novice and expert Python developers a best-practice handbook to the installation, configuration, and usage of Python on a daily basis.

By the Community

This guide is architected and maintained by Kenneth Reitz in an open fashion. This is a community-driven effort that serves one purpose: to serve the community.

For the Community

All contributions to the Guide are welcome, from Pythonistas of all levels. If you think there's a gap in what the Guide covers, fork the Guide on GitHub and submit a pull request. Contributions are welcome from everyone, whether they're an old hand or a first-time Pythonista, and the authors to the Guide will gladly help if you have any questions about the appropriateness, completeness, or accuracy of a contribution.

To get started working on The Hitchhiker's Guide, see the *Contribute* page.

10.2 The Community

10.2.1 BDFL

Guido van Rossum, the creator of Python, is often referred to as the BDFL — the Benevolent Dictator For Life.

10.2.2 Python Software Foundation

The mission of the Python Software Foundation is to promote, protect, and advance the Python programming language, and to support and facilitate the growth of a diverse and international community of Python programmers.

Learn More about the PSF.

10.2.3 PEPs

PEPs are *Python Enhancement Proposals*. They describe changes to Python itself, or the standards around it.

There are three different types of PEPs (as defined by PEP1):

Standards Describes a new feature or implementation.

Informational Describes a design issue, general guidelines, or information to the community.

Process Describes a process related to Python.

Notable PEPs

There are a few PEPs that could be considered required reading:

- **PEP8: The Python Style Guide.** Read this. All of it. Follow it.
- **PEP20: The Zen of Python.** A list of 19 statements that briefly explain the philosophy behind Python.
- **PEP257: Docstring Conventions.** Gives guidelines for semantics and conventions associated with Python docstrings.

You can read more at The PEP Index.

Submitting a PEP

PEPs are peer-reviewed and accepted/rejected after much discussion. Anyone can write and submit a PEP for review.

Here's an overview of the PEP acceptance workflow:

10.2.4 Python Conferences

The major events for the Python community are developer conferences. The two most notable conferences are PyCon, which is held in the US, and its European sibling, EuroPython.

A comprehensive list of conferences is maintained at pycon.org.

10.2.5 Python User Groups

User Groups are where a bunch of Python developers meet to present or talk about Python topics of interest. A list of local user groups is maintained at the Python Software Foundation Wiki.

10.2.6 Plone Community

#TODO: write about the Plone community

10.3 Learning Python

10.3.1 Beginner

Learn Python Interactive Tutorial

Learnpython.org is an easy non-intimidating way to get introduced to python. The website takes the same approach used on the popular Try Ruby website, it has an interactive python interpreter built into the site that allows you to go through the lessons without having to install Python locally.

> Learn Python

Learn Python the Hard Way

This is an excellent beginner programmer's guide to Python. It covers "hello world" from the console to the web.

> Learn Python the Hard Way

Crash into Python

Also known as *Python for Programmers with 3 Hours*, this guide gives experienced developers from other languages a crash course on Python.

> Crash into Python

Dive Into Python 3

Dive Into Python 3 is a good book for those ready to jump in to Python 3. It's a good read if you are moving from Python 2 to 3 or if you already have some experience programming in another language.

> Dive Into Python 3

Think Python: How to Think Like a Computer Scientist

Think Python attempts to give an introduction to basic concepts in computer science through the use of the python language. The focus was to create a book with plenty of exercises, minimal jargon and a section in each chapter devoted to the subject of debugging.

While exploring the various features available in the python language the author weaves in various design patterns and best practices.

The book also includes several case studies which have the reader explore the topics discussed in the book in greater detail by applying those topics to real-world examples. Case studies include assignments in GUI and Markov Analysis.

Think Python

10.3.2 Advanced

Pro Python

Todo

Write about Pro Python

Expert Python Programming

Todo

Write about Expert Python Programming

10.4 Documentation

10.4.1 Official Documentation

The official Python Language and Library documentation can be found here:

- Python 2.x
- Python 3.x

10.4.2 Read the Docs

Read the Docs is a popular community project, providing a single location for all documentation of popular and even more exotic Python modules.

Read the Docs

10.5 News

10.5.1 Planet Python

This is an aggregate of Python news from a growing number of developers.

Planet Python

10.5.2 /r/python

/r/python is the Reddit Python community where users contribute and vote on Python-related news.

/r/python

10.5.3 Python Weekly

Python Weekly is a free weekly newsletter featuring curated news, articles, new releases, jobs, etc. related to Python.

Python Weekly

Contibution notes and legal information are here (for those interested).

10.6 Contribute

Python-guide is under active development, and contributors are welcome.

If you have a feature request, suggestion, or bug report, please open a new issue on GitHub. To submit patches, please send a pull request on GitHub. Once your changes get merged back in, you'll automatically be added to the Contributors List.

10.6.1 Style Guide

For all contributions, please follow the *The Guide Style Guide*.

10.6.2 Todo List

If you'd like to contribute, there's plenty to do. Here's a short todo list.

- Establish "use this" vs "alternatives are...." recommendations

---

Todo

add supertab notes

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/bungeni-portal/checkouts/latest/docs/dev/env.rst, line 58.)

Todo

write a general blurb introducing the Python language

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/bungeni-portal/checkouts/latest/docs/intro/duction.rst, line 4.)

Todo

Write about Pro Python

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/bungeni-portal/checkouts/latest/docs/intro/learning.rst, line 62.)

Todo

Write about Expert Python Programming

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/bungeni-portal/checkouts/latest/docs/intro/learning.rst, line 67.)

Todo

Determine License

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/bungeni-portal/checkouts/latest/docs/notes/license.rst, line 4.)

Todo

Fill in License stub

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/bungeni-portal/checkouts/latest/docs/writing/license.rst, line 7.)

Todo

Fill in "Structuring Your Project" stub

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/bungeni-portal/checkouts/latest/docs/writing/structure.rst, line 6.)

10.7 License

Todo

Determine License

10.8 The Guide Style Guide

As with all documentation, having a consistent formating helps make the document more understandable. In order to make The Guide easier to digest, all contributions should fit within the rules of this style guide where approriate.

The Guide is written as *reStructuredText*.

Note: Parts of The Guide may not yet match this style guide. Feel free to update those parts to by in sync with The Guide Style Guide

---

Note: On any page of the rendered HTML you can click "Show Source" to see how authors have styled the page.

10.8.1 Relevancy

Stride to keep any contributions relevant to the *purpose of The Guide*.

- Avoid including too much information on subjects that don't directly relate to Python development.
- Prefer to link to other sources if the information is already out there. Be sure to describe what and why you are linking.
- Cite references where needed.
- If a subject isn't directly relevant to Python, but useful in conjuction with Python (ex: Git, Github, Databases), reference by linking to useful resouces and describe why it's useful to Python.
- When in doubt, ask.

10.8.2 Headings

Use the following styles for headings.

Chapter title:

```
#########
Chapter 1
#########
```

Page title:

```
==================
Time is an Illusion
==================
```

Section headings:

```
Lunchtime Doubly So
-------------------
```

Sub section headings:

```
Very Deep
~~~~~~~~~
```

10.8.3 Prose

Wrap text lines at 78 characters. Where necessary, lines may exceed 78 characters, especially if wrapping would make the source text more difficult to read.

10.8.4 Code Examples

Wrap all code examples at 70 characters to avoid horizontal scrollbars.

Command line examples:

```
.. code-block:: console

    $ run command --help
    $ ls ..
```

Be sure to include the `$` prefix before each line.

Python interpreter examples:

```
Label the example::

.. code-block:: python

    >>> import this
```

Python examples:

```
Descriptive title::

.. code-block:: python

    def get_answer():
        return 42
```

10.8.5 Externally Linking

- Prefer labels for well known subjects (ex: proper nouns) when linking:

```
Sphinx_ is used to document Python.

.. _Sphinx: http://sphinx.pocoo.org
```

- Prefer to use descriptive labels with inline links instead of leaving bare links:

```
Read the `Sphinx Tutorial <http://sphinx.pocoo.org/tutorial.html>`_
```

- Avoid using labels such as “click here”, “this”, etc. preferring decriptive labels (SEO worthy) instead.

10.8.6 Linking to Sections in The Guide

To cross-reference other parts of this documentation, use the :ref: keyword and labels.

To make reference labels more clear and unique, always add a `-ref` suffix:

```
.. _some-section-ref:

Some Section
------------
```

10.8.7 Notes and Warnings

Make use of the appropriate admonitions directives when making notes.

Notes:

```
.. note::
    The Hitchhiker's Guide to the Galaxy has a few things to say
    on the subject of towels. A towel, it says, is about the most
    massively useful thing an interstellar hitch hiker can have.
```

Warnings:

```
.. warning:: DON'T PANIC
```

10.8.8 TODOs

Please mark any incomplete areas of The Guide with a todo directive. To avoid cluttering the *Todo List*, use a single `todo` for stub documents or large incomplete sections.

```
.. todo::
    Learn the Ultimate Answer to the Ultimate Question
    of Life, The Universe, and Everything
```