eestec.portal Documentation

Release 0.1

February 09, 2015

1 Table of Contents 3
1.1 Definitions of basic terms 3
1.2 Technical Specification 4
1.3 Developers Guide 11

2 Changelog 23
2.1 0.1 (unreleased) 23

3 License (3-clause BSD) 25

4 Indices and tables 27

Framework Plone 4.2

Bug tracker https://github.com/eestec/eestec.portal/issues

Source https://github.com/eestec/eestec.portal

Documentation http://eestecportal.readthedocs.org/

Code status

Summary

EESTEC.net portal is the main hub for managing of EESTEC Events (workshops, exchanges, etc.), applications to these event, managing Local Committees, managing public content (news, PR, etc.) and an online collaboration space for EESTECers.

Make sure you read the install instructions before attempting to *run buildout* or at least check the list of prerequisites.

Table of Contents

1.1 Definitions of basic terms

1.1.1 EESTEC

Terms that appear in the general EESTEC context and can help understand context and use-cases behind the code in `eestec.portal` package.

Member of EESTEC International A member of EESTEC is always an association.

LC (Local Committee) An association the General Assembly of EESTEC has accepted as a full member. Must organize international events and has a vote in the General Assembly. They can freely apply for all events.

JLC (Junior Local Committee) In this membership level the association must be registered. They have an obligation to try to hold an international event within two years or they will be demoted. They do not have a vote in General Assembly. Promotion to this status is handled by the General Assembly. They can freely apply for all events.

Observer In this membership level the association must be organized. They should register within two years or they are automatically demoted. They do not have a vote in the general assembly. They are promoted to this status by the international Board. They can freely apply for all events.

Observer Candidate Any interested party can apply for this status from the international Board. They can apply for events with the acceptal of VC-EA.

Organizer An LC that is organizing an event.

CP (Contact Person) The person in an LC which handles communcation between local level and international level.

VC-IA (Vice Chairperson for Internal Affairs) The person in the international Board who is responsible for staying bidirectionally in touch with EESTEC member organizations (ie. LCs)

VC-EA (Vice Chairperson for External Affairs) The person in the international Board who is responsible for staying in contact with partner organizations and expansion of EESTEC.

International Board The board to run practical level tasks. Elected by GA in every Congress. The term of the board is from Congress to Congress.

Local Board The board of a member organization (ie. an LC).

Event An event held by a member organization (ie. an LC) into which people from other member organizations can apply.

Event Application Deadline Deadline until members of member organizations (ie. LCs) can apply for an event.

1.1.2 Technical

Terms that appear in more technical contexts and expect the reader to have some development background.

[Plone Object] EestecMember A Plone object representing a single EESTEC member storing his personal information.

[Plone Object] LC A Plone object representing a single local committee. Based on it's workflow state it is defined whether it is a normal LC, or maybe a JLC or Observer. LC is folderish and contains LC specific information such as news, events, etc.

[Plone Object] Event A Plone object representing a single EESTEC event, whether it is a workshop, exchange, ECM, Congres, etc. It contains all relevant information about an event. Event is folderish contains EventApplications. Events are contained in LCs.

[Plone Object] EventApplication A Plone object representing a single event application by an EESTEC member. It can only be contained within a single Event. EventApplication stores a relation to a member and so provides all necessary data about the participant.

1.2 Technical Specification

1.2.1 Rules and Assumptions

Rules and Assumptions

Some rules and guidelines we internally agreed on. Along with assumptions in code.

First page

We assume that there is a Page content type object in root of Plone site with id 'what-is-eestec'. The source of this file should be handled with care as it's divs have special ids, that show one part of content for anonymous and another part for authenticated users. Div with id 'blob' is displayed for anonymous visitors and 'blob-login' is displayed for authenticated members.

Components folder

This is a normal Plone folder with id 'components' residing in root of Plone site. Only managers have direct access to this folder. The folder contains components that are used as building blocks of the site. Example: collections for building footer sitemap, first page footer images, ...

Footer navigation

We have a collection that is listing child objects for every important top-level folder. We display these collections in footer with a little help of Products.ContentWellPortlets.

LCs Folder

This is a normal Plone folder that contains only LC content objects. All LCs must be added to this folder.

LC titles

LC's do not have their status in the name of the LC. Their status is controlled by a custom workflow enabling different states: Inactive, Observer, JLC, LC. The actual presentation title is then build from the status of the LC and it's name. Ergo, for LC name use just the city, do not prefix it with status.

Remember and self-registration

Since Plone 3.3.2 Remeber's self-registration is broken: http://plone.org/products/remember/issues/64

We fixed this by going to http://eestec.net/portal_memberdata/manage -> click tab Security -> uncheck 'Aquire security settings' and check row Anonymous. Save.

Groups

LC Group Each LC has a dedicated group for assigning LC specific permissions on portal object. When administrators create a new LC they must also manually create a group and assign local role "LC" for this group on the LC (this can later on be automatized but we now have more important issues to address). When Member object is created by the registration form this group is automatically assigned local role "LC" over the Member object, and member is added to LC Group (e.g. Ljubljana). This ensures that CPs have permissions over their LC's members.

Administrators Senior IT Team members with managerial permissions.

Board International Board members with special permissions like approving event participants, etc. Boardies also have a dedicated private workspace on the site, where they can share documents, know-how, tasks, ...

Roles

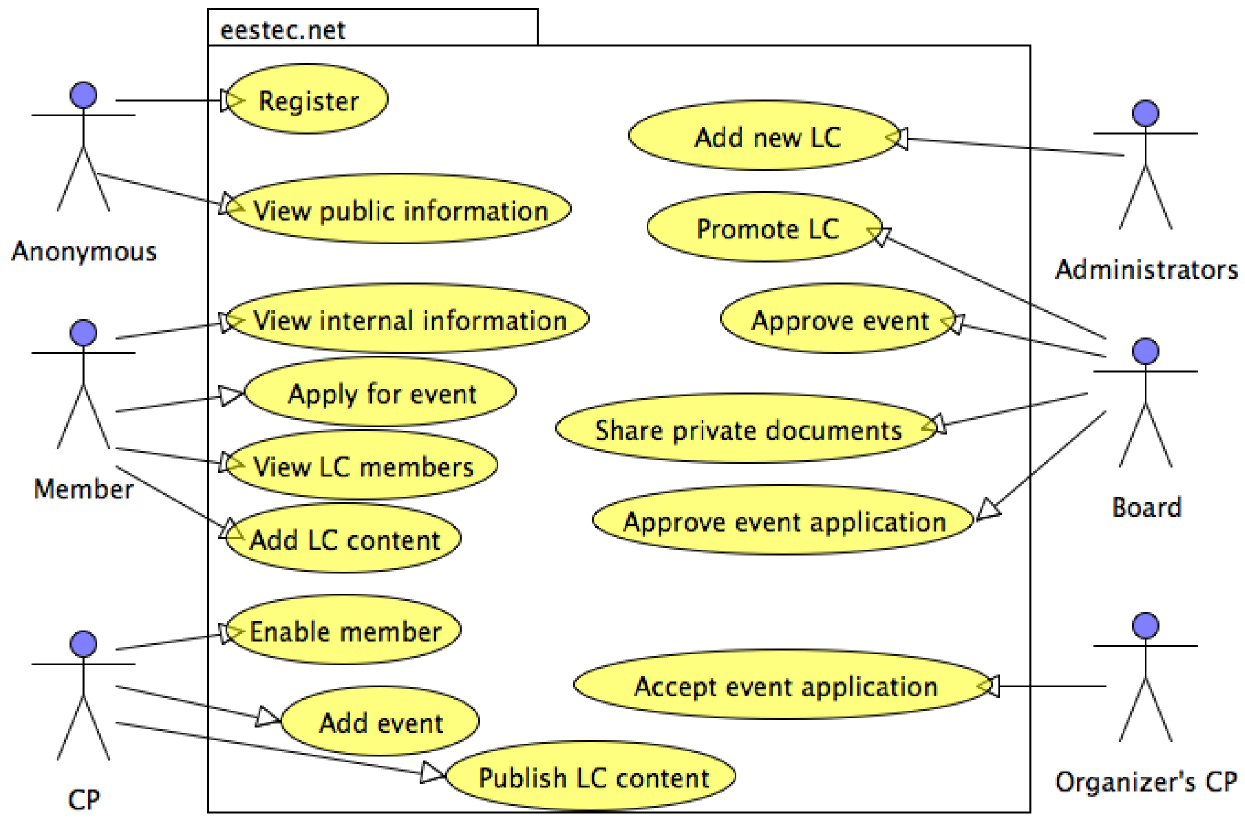
CP A member with this role has CP related permissions: adding events, accepting event applications, managing it's LCs members, etc.

Board A role for the Board group.

LC A role that is assigned on LC object for the LC's "LC Group".

1.2.2 Use case

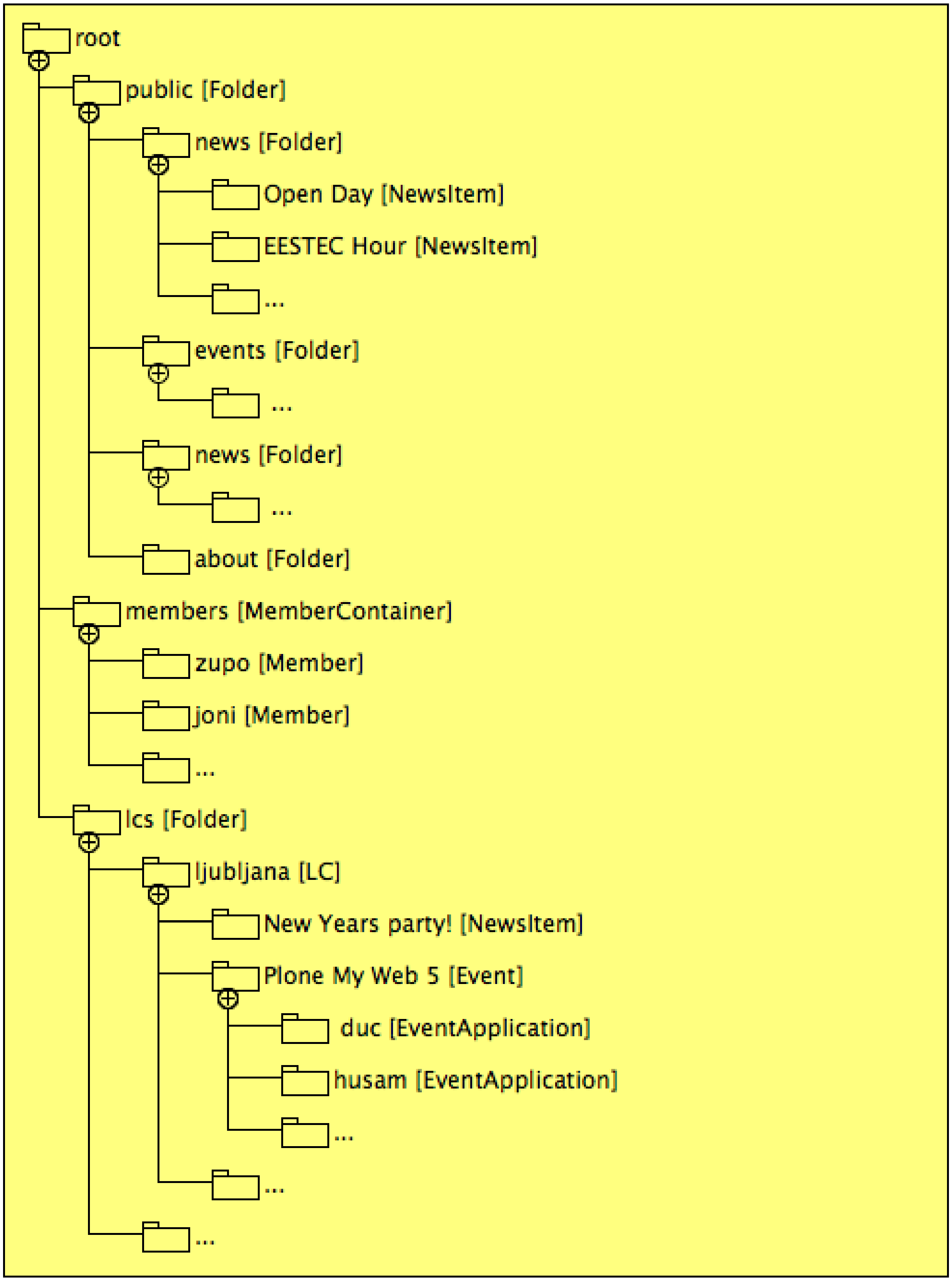
This diagram is displaying activities that different entities can take.



`Download UMLet source file.`

1.2.3 Content Structure

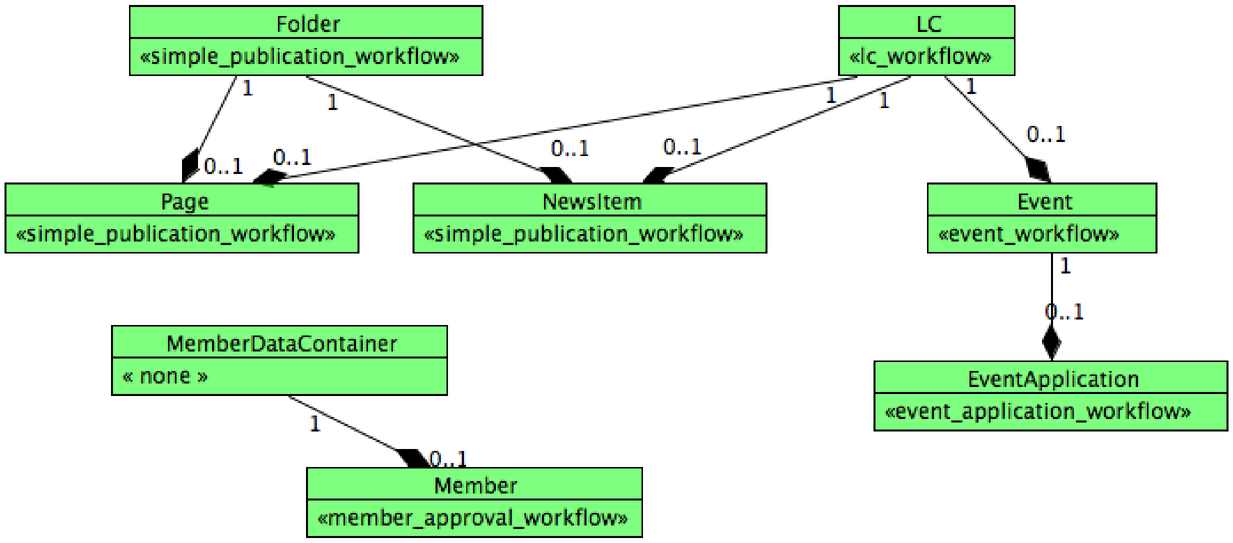
Since Plone uses an object database (ZODB) rather than a traditional relational database (such as MySQL) all content are objects contained within parent objects. It's best if you imagine this as folders and files on you local disc.



Download UMLet source file.

1.2.4 Class diagram

A Class diagram shows which objects we have in our data model and what are their relationships.


Folder
«simple_publication_workflow»
LC
«lc_workflow»
Page
«simple_publication_workflow»
NewsItem
«simple_publication_workflow»
Event
«event_workflow»
MemberDataContainer
« none »
EventApplication
«event_application_workflow»
Member
«member_approval_workflow»
1
0..1

`Download UMLet source file.`

1.2.5 Workflows

Workflows

Detailed descriptions of what workflows we use and the user story behind them.

Prelude

This document is primarily meant for developers on the eestec.net project. It's also a recommended read for power users, such as CPs and the Board, as the reader gains valuable insight of the story behind eestec.net workflows and object relations.

The general purpose of the eestec.net portal is to standardize and enable EESTEC members to apply for events themselves and then track their application online, hence replacing the currently inefficient way of doing this though emails.

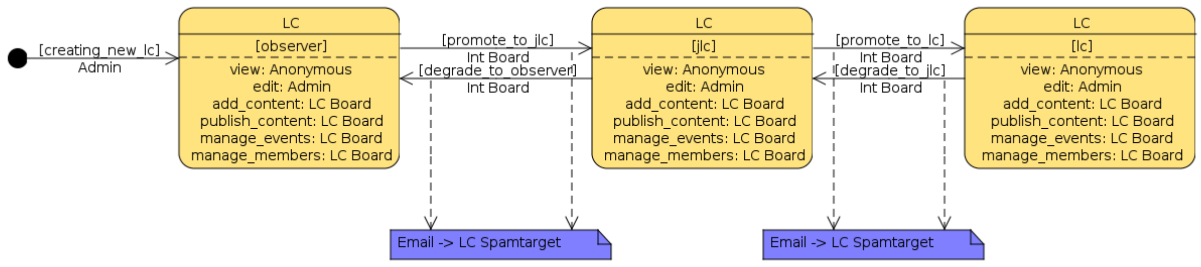
Note: Diagrams are made in UMLet. Their source files are below each of their screenshots.

Note: By default, workflow specifications for Plone are done in XML which get kind of hard to read when they grow. Luckily we can use a great tool called 'collective.wtf' that enables us to do workflow specification in CSV format in your favorite spreadsheet editor (Calc, Excel, etc.) making specifications much more readable and developer-friendly. Read more about it here (TODO: link).

LC workflow

We have different types of LCs: normal, junior, observer. Each of these types have some specific behaviour and permissions bind to them. To avoid creating separate content types for each of these types of LCs we'll rather use workflows and one content type "LC".

LC object starts it's life cycle in state "pending" with only administrators being able to add new LCs to the site. Then administrator will check if all LCs fields are correctly set and will move LC's state to a new one, either "LC", "JLC" or "Observer". Each of these workflow states will have special permissions bindings (Observers cannot add Events, ...).



`Download UMLet source file.`

TODO: LC can only be created by dev.

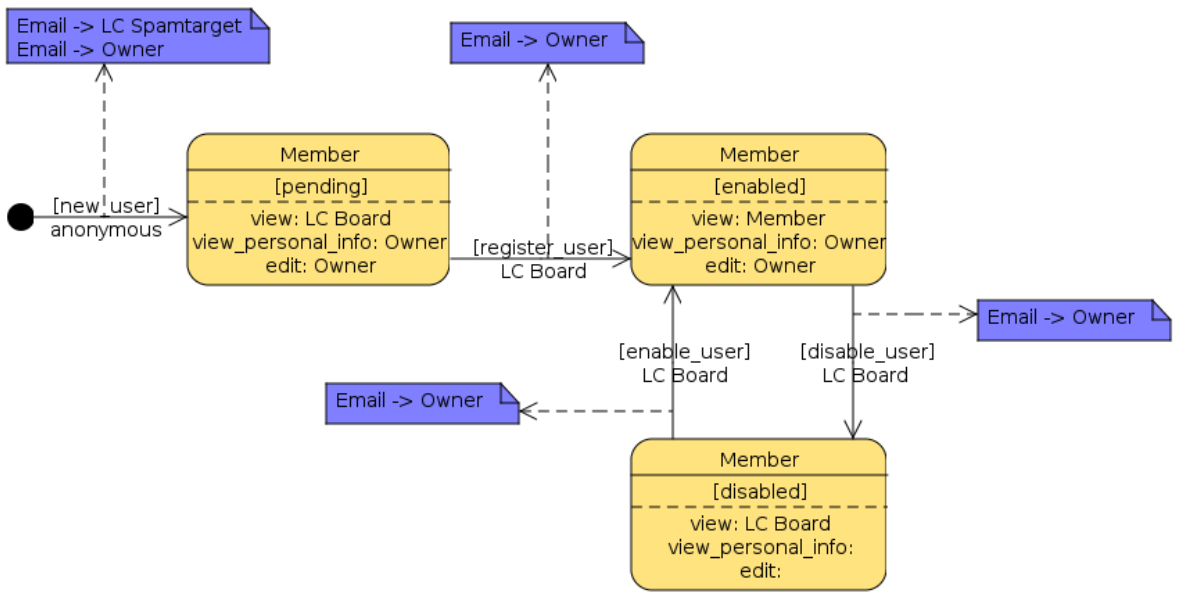
LC Content workflow

All LC members can add default Plone content into their LC folder (news, pages, files, images, etc.). This content is not immediately visible by non-LC members and needs to be published first. CPs can publish such content with a single click. We'll use default Plone publication workflow for this. TODO: link to plone.org explaining default workflows

Member workflow

We use Membrane/Remember framework for advanced member handling in Plone. Consequently it's much easier to add new fields to a member object and bind workflows to them. Since we need a machinery for confirming newly registered members we'll again use workflows. The default workflow that comes with Remember already supports this kind of user story and only minor customization is needed to suit our needs.

Member object starts it's life cycle in state pending with anonymous user filling out the registration form. Based on LC selection on the registration form, respective CP is given permissions over the newly created Member object and can now approve it by moving it to state "active". Upon approving, member receives an activation email and can now login to the site. There is one more state, which we'll have to add manually, the "alumni" state. CP is again responsible for transition of member from "active" to "alumni". All states also have a "disable" transition which disables the member, denying login for this member. Round trip from "disabled" back to "active" state is also supported.



`Download UMLet source file.`

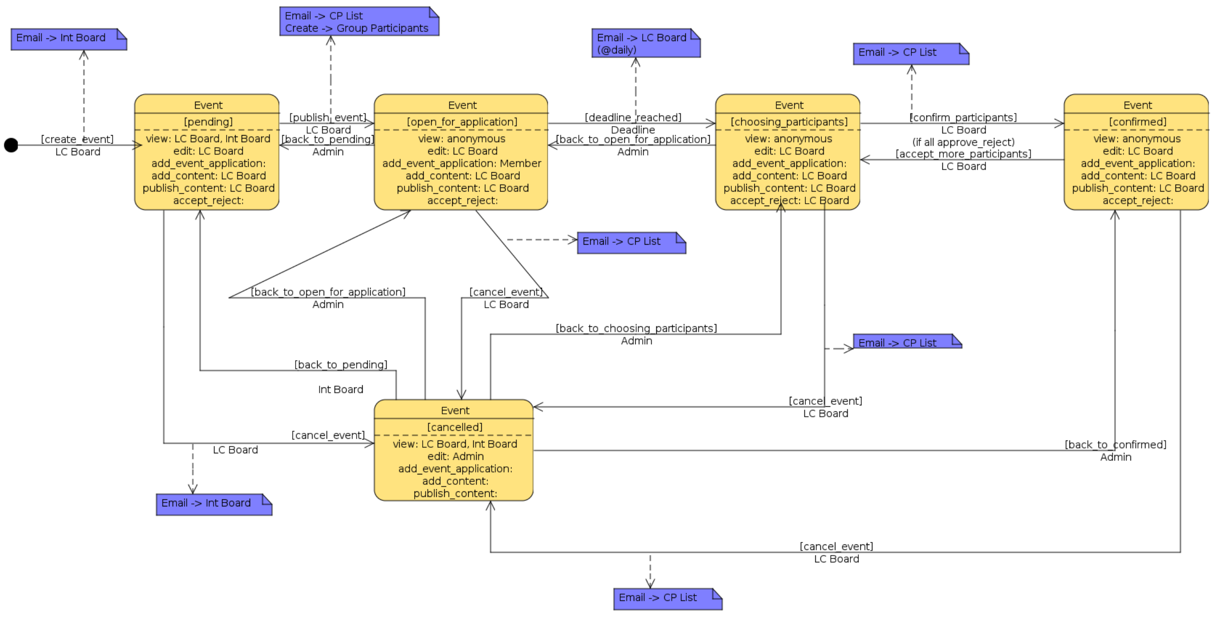
Event and EventApplication workflow

Now here's where the fun starts :). Events start their life cycle in state pending with CP adding a new Event object into it's LC object. Then VC-IA comes along and confirms the Event which is now in state "open_for_applications". An automatic email is sent to CP mailing list to inform everybody that we have a new Event members can apply to. Members also see this Event in the "upcoming events" listing. Upon opening a single Event members are presented with all information about an Event and an "apply for this event" button. This button creates an EventApplication in this Event. Organizer's CP and Member's CP are both notified about this via email.

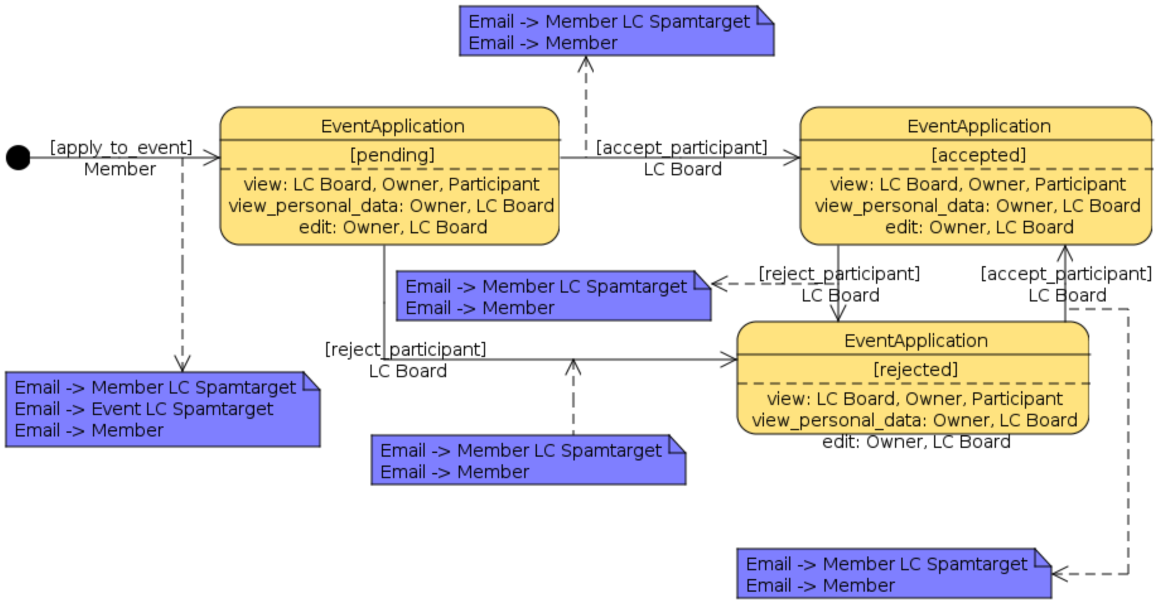
Members are able to apply until the official application deadline. After the deadline, Organizer's CP will "accept" or "reject" each pending EventApplication.

When Organizer is happy with accepted participants he puts the Event into next workflow state, 'board_approval'. If Event is not put into board approval within 24 hours from the official deadline they are sent an email reminder to do so. VC-IA is notified of this.

An email is sent to VC-IA requesting him/her to login to the site and approve/reject all event applications for this Event. When VC-IA is happy with the list of approved participants, he/she moves the Event into state "confirmed". All members, member's CPs and organizer CP are notified that Event is confirmed and their application is approved.



Download UMLet source file.


Email -> Member LC Spamtarget
Email -> Member
[apply_to_event]
Member
EventApplication
[pending]
view: LC Board, Owner, Participant
view_personal_data: Owner, LC Board
edit: Owner, LC Board
[accept_participant]
LC Board
EventApplication
[accepted]
view: LC Board, Owner, Participant
view_personal_data: Owner, LC Board
edit: Owner, LC Board
Email -> Member LC Spamtarget
Email -> Member
[reject_participant]
LC Board
[accept_participant]
LC Board
[reject_participant]
LC Board
EventApplication
[rejected]
view: LC Board, Owner, Participant
view_personal_data: Owner, LC Board
edit: Owner, LC Board
Email -> Member LC Spamtarget
Email -> Event LC Spamtarget
Email -> Member
Email -> Member LC Spamtarget
Email -> Member
Email -> Member LC Spamtarget
Email -> Member

Download UMLet source file.

1.3 Developers Guide

1.3.1 Conventions

Rules and guidelines on syntax style, development process, repository workflow, etc.

Conventions

Line length

All Python code in this package should be PEP8 valid. However, we don't strictly enforce the 80-char line length rule. It is encouraged to have your code formatted in 80-char lines, but somewhere it's just more readable to break this rule for a few characters. Long and descriptive test method names are a good example of this.

Note: Configuring your editor to display a line at 80th column helps a lot here and saves time.

Note: The line length rules also applies to non-python source files, such as documentation .rst files.

About imports

1. Don't use * to import *everything* from a module.
2. Don't use commas to import multiple stuff on a single line.
3. Don't use relative paths.

```
from collective.table.local import add_row
from collective.table.local import delete_rows
from collective.table.local import update_cell
```

instead of

```
from collective.table.local import *
from collective.table.local import add_row, delete_rows
from .local import update_cell
```

Sort imports

As another imports stylistic guide: Imports of code from other modules should always be alphabetically sorted with no empty lines between imports. The only exception to this rule is to keep one empty line between a group of `from x import y` and a group of `import y` imports.

```
from collective.table.tests.base import TableIntegrationTestCase
from plone.app.testing import login

import os
```

instead of

```
import os

from plone.app.testing import login
from collective.table.tests.base import TableIntegrationTestCase
```

Commit checklist

Before every commit you should:

- Run *Unit tests*.

- Run *Syntax validation*.
- Add an entry to *Changelog* (if applicable).
- Add/modify *Sphinx Documentation* (if applicable).

Note: All syntax checks and all tests can be run with a single command:

```
$ make tests
```

Unit tests

Un-tested code is broken code.

For every feature you add to the codebase you must also add tests for it. Also write a test for every bug you fix to ensure it doesn't crop up again in the future.

You run tests like this:

```
$ bin/test
```

To speed things up, you can choose to run only some tests, not all at once. Use the `-t` to filter out tests and run only those that match the expression.

```
# run only setup tests
$ bin/test -t test_setup
```

Syntax validation

All Python source code should be *PEP-8* valid and checked for syntax errors. The tools used for this are *flake8* and *zptlint*.

To validate your source code, run the following commands:

```
$ bin/flake8 src/eestec/portal
$ for pt in `find src/eestec/portal/ -name "*.pt"` ; do bin/zptlint $pt; done

# or just this one (also runs all unit tests)
# make tests
```

Note: It pays off to invest a little time to make your editor run *flake8* on a file every time you save that file. Saves lots of time in the long run.

Changelog

Feature-level changes to code are tracked inside `docs/HISTORY.txt`. Examples:

- added feature X
- removed Y
- fixed bug Z

Add an entry every time you add/remove a feature, fix a bug, etc.

Sphinx Documentation

Un-documented code is broken code.

For every feature you add to the codebase you should also add documentation for it to `docs/`.

After adding/modifying documentation, re-build *Sphinx* and check how it is displayed:

```
$ make docs
$ open docs/html/index.html
```

Documentation is automatically generated from these source files every time you push your code to GitHub. The post-commit hook is handled by ReadTheDocs and the results are visible at http://eestecportal.readthedocs.org/.

Travis Continuous Integration

On every push to GitHub, Travis runs all tests/syntax validation checks and reports failures (if there are any) to `it@eestec.net` mailinglist and to the `#ngep` IRC channel.

Travis is configured with the `.travis.yml` file located in the root of `eestec.portal` package.

Git workflow & branching model

We only have one Python package for the entire portal, `eestec.portal`, version controled by Git on https://github.com/eestec/eestec.portal.

Git repository has the following layout:

- **feature branches**: all development for new features must be done in dedicated branches, normaly one branch per feature,
- **master branch**: when features get completed they are merged into the master branch; bugfixes are commited directly on the master branch,
- **tags**: whenever we deploy code to production we tag the repository so we can later re-trace our steps and revert broken deployments if necessary.

Hooks:

- On every change to the `master` branch, our *Staging environment* gets re-deployed.
- On every new tag, our *Production environment* gets re-deployed.

1.3.2 Local development environment

Setting up and using the local development environment.

Local development environment

This section is meant for developers on the eestec.net project. It's purpose is to guide them through the steps needed to start contributing.

Prerequisites

System libraries First let's look at 'system' libraries and applications that are normally installed with your OS packet manager, such as apt, aptitude, yum, etc.:

- `libxml2` - an xml parser written in C
- `libxslt` - XSLT library written in C
- `pcre` - Perl regex libraly
- `git` - version control system.
- `gcc` - the GNU Compiler Collection.
- `g++` - the C++ extensions for gcc.
- `GNU make` - the fundamental build-control tool.
- `GNU tar` - the (un)archiving tool for extracting downloaded archives.
- `bzip2` and `gzip` decompression packages - `gzip` is nearly standard, however some platforms will require that `bzip2` be installed.
- `Python 2.7` - Plone 4.2 does NOT work with other Python version so you need this exact version.
- development headers for libxml2, libxslt and python

Python tools Then you'll also need to install some Python specific tools:

- easy_install - the Python packaging system (download http://peak.telecommunity.com/dist/ez_setup.py and run `sudo python ez_setup.py`.
- virtualenv - a tool that assists in creating isolated Python working environments.

Code style guide We use the `plone.api` style guide so please read it and use it: http://ploneapi.readthedocs.org/en/latest/contribute/conventions.html

Further information If you experience problems read through the following links as almost all of the above steps are required for a default Plone development environment:

- http://plone.org/documentation/tutorial/buildout
- http://pypi.python.org/pypi/zc.buildout/
- http://pypi.python.org/pypi/setuptools
- http://plone.org/documentation/manual/installing-plone

If you are an OS X user, you first need a working Python implementation (the one that comes with the operating system is broken). Use https://github.com/collective/buildout.python and be happy. Also applicable to other OSes, if getting a working Python proves a challenge.

Creating the development environment

Go to your home folder or a folder you use for development and *clone* latest `eestec.portal` code:

```
[you@local ~]$ cd <your_work_folder>
[you@local work]$ git clone https://github.com/eestec/eestec.portal.git
```

Now *cd* into the newly created directory and create an isolated python environment and build the development environment.

```
[you@local work]$ cd eestec.portal
[you@local eestec.portal]$ make
```

Internally, this uses *zc.buildout* to build Zope and any other servers we might need, fetches all dependencies and installs them, generates config files and scripts, prepares deployment tools and much more. Read more about buildout at http://plone.org/documentation/tutorial/buildout:

While buildout is running go make some tea. When you run it for the first time it needs a couple of minutes to finish preparing your development environment. More if you have a slower Internet connection.

After `make` is finished your development environment is ready! You have more make command on your disposal:

```
[you@local eestec.portal]$ make tests  # run all unit tests
[you@local eestec.portal]$ make docs  # generate documentation
[you@local eestec.portal]$ make clean && make  # start from scratch
```

Starting the portal

Let's start Zope - the application server. There are several ways to start Zope. For development purposes we'll use the 'foreground' mode which starts Zope in console's foreground so you can immediately see all debug messages and use the Python Debugger to interactively debug your code:

```
[you@local eestec.portal]$ bin/instance fg
```

Once Zope has started you need to add a Plone site. Open up a browser and point it to `http://localhost:8080/@@plone-addsite?site_id=Plone`. Username is `admin`, password is also `admin`. Check the `eestec.portal` checkbox in the *Add-ons* list and click `Create Plone Site`.

There you go, a local installation of the EESTEC portal on your laptop. Go nuts with it!

You can also run our *Unit tests* or perform *Syntax validation*.

Adding initial content

If you don't have a ZODB to work with, as in, you are starting with a fresh install of Plone, follow this steps to add some content to your site so you can see what your code does:

1. Follow the *Starting the portal* instructions above.
2. Use the `Add new ...` drop down menu to add a new `LC`.
3. Once inside the new LC, use the `Add new ...` drop down menu again, this time to add a new `Event`.
4. Once inside the new Event, use the `Add new ...` drop down menu for the last time, now to add a new `Event Application`.
5. Use the `State:` drop-down menu to play around with different items' workflow states.

Working on an issue

Out GitHub account contains a list of open issues. Click on one that is labeled with a green `entry-level` tag. If the issue description says `No one is assigned` it means no-one is already working on it and you can claim it as your own. Click on the button next to the text and make yourself the one assigned for this issue.

Based on our *Git workflow & branching model* all new features must be developed in separate git branches. So if you are not doing a simple bugfix, but rather adding new features/enhancements, you should create a *feature branch*. This way your work is kept in an isolated place where you can receive feedback on it, improve it, etc. Once we are happy with your implementation, your branch gets merged into *master* at which point everyone else starts using your code.

```
[you@local eestec.portal]$ git checkout master  # go to master branch
[you@local eestec.portal]$ git checkout -b issue_17  # create a feature branch
# replace 17 with the issue number you are working on

# change code here

[you@local eestec.portal]$ git add -p && git commit  # commit my changes
[you@local eestec.portal]$ git push origin issue_17  # push my branch to GitHub
# at this point other can see your changes but they don't get effected by
them; in other words, others can comment on your code without your code
changing their development environments
```

Read more about Git branching at http://learn.github.com/p/branching.html. Also, to make your git nicer, we have a *Unit tests* chapter in *Tips & Tricks*.

Also please add your name to the Changelog

Once you are done with your work and you would like us to merge your changes into master, go to GitHub to do a *pull request*. Open a browser and point it to https://github.com/eestec/eestec.portal/tree/issue_<ISSUE_NUMBER>. There you should see a `Pull Request` button. Click on it, wrote some text what you did and anything else you would like to tell the on who will merge your branch, and finally click `Send pull request`. Now wait that someone comes by and merges your branch (don't do it yourself, even if you have permissions to do so).

An example pull request text:

```
Please merge my branch that resolves issue #13.
```

1.3.3 Remote development environment

Description of the remote development environment and how to use it for remote development in cases when local development environment is not feasible.

Remote development environment

TODO: Using remote development environment @ Hetzner

1.3.4 Staging environment

Description of our staging environment and how to deploy code to staging.

Staging environment

Auto-deploying to staging

On every change to the `master` branch, our development server builds a staging environment with latest code on fresh live data. This is done with a cronjob invoked script that does the following:

1. Check if there were any changes from last time.

2. Get latest changes.
3. Run `bin/buildout -c staging`.
4. Rsync `Data.fs` from the production server.
5. Rsync `blobstorage` from the production server.
6. Restart Zope.

Note: The script is invoked every 5 minutes.

Note: There is no need to purge the staging environment for every re-build as we don't need to test the entire build process – Travis already does that for us.

TODO: How is staging server actually set up? Permissions TODO: crontab script that I describe below doesn't exist yet

1.3.5 Production environment

Description of our production environment and how to deploy code to production.

Production environment

Auto-deploying to production

Whenever a new tag is detected in our Git repository, our production server fetches the latest tag and deploys it. This is done with a cronjob invoked script that does the following:

1. Check if there are any new tags.
2. Backup `Data.fs` and `blobstorage`.
3. Get latest tag.
4. Run `bin/buildout -c production`.
5. Restart all Zope servers.

Note: The script is invoked every 5 minutes.

TODO: How is production server actually set up? Permissions? What OS? TODO: crontab script that I describe below doesn't exist yet

Deployment checklist

As seen above, deployment to production is invoked by simply creating a Git tag. To ensure everything goes smoothly, follow these steps:

- Checkout the code locally and run all tests.
- Bump version in `version.txt` and make sure `HISTORY.txt` has been updated.
- Commit and push all changes.
- Confirm that the code works fine on live data on staging server.

- Create & push a tag: `git tag -a v0.14` and `git push --tags`.
- Verify that production deployment went well.

Reverting a bad deployment

If a deployment goes bad you can easily revert to the previous tag and pre-deployment database snapshot by running the following Fabric script:

1.3.6 Tips & tricks

Random development and deployment tips & tricks.

Tips & Tricks

Tunneling to ETHZ server

Sometimes you need direct access to services running on ETHZ server. All servers are running on local address `127.0.0.2`, and the ports for them are as follows:

```
zope1       = 8091
zope2       = 8092
zope3       = 8093
zope4       = 8094
zope4_debug = 8099 (needs to be manually started with ``bin/zope_debug fg``)
zeo         = 8090
haproxy     = 8080
supervisor  = 9000
```

So, to access (for example) `zope_debug` use:

> $ ssh eestecwm@galen.ee.ethz.ch -L 8099:127.0.0.2:8099

Then open `http://localhost:8099` in your browser and you will directly access the service on this port.

Setting up Git

Git is a very useful tool, especially when you configure it to your needs. Here are a couple of tips.

Enhanced git prompt

Do one (or more) of the following:

- http://clalance.blogspot.com/2011/10/git-bash-prompts-and-tab-completion.html
- http://en.newinstance.it/2010/05/23/git-autocompletion-and-enhanced-bash-prompt/
- http://gitready.com/advanced/2009/02/05/bash-auto-completion.html

Example of `~/.gitconfig`

```
[user]
    name = John Smith
    email = john.smith@gmail.com
[diff "cfg"]
    funcname = ^\\(\\[.*\\].*\\)$
[color]
    diff = auto
    status = auto
    branch = auto
[alias]
    st = status
    ci = commit
    br = branch
    co = checkout
[core]
    excludesfile = /home/jsmith/.gitignore
    editor = nano
[github]
    user = jsmith
    token = <token_here>
```

Example of `~/.gitignore`

```
# Compiled source #
###################
*.com
*.class
*.dll
*.exe
*.o
*.so
*.lo
*.la
*.rej
*.pyc
*.pyo

# Packages #
############
# it's better to unpack these files and commit the raw source
# git has its own built in compression methods
*.7z
*.dmg
*.gz
*.iso
*.jar
*.rar
*.tar
*.zip

# Logs and databases #
######################
*.log
*.sql
*.sqlite

# OS generated files #
######################
```

```
.DS_Store
.DS_Store?
ehthumbs.db
Icon?
Thumbs.db

# Python projects related #
###########################
*.egg-info
Makefile
.egg-info.installed.cfg
*.pt.py
*.cpt.py
*.zpt.py
*.html.py
*.egg
```

1.3.7 Source documentation

List of all Python modules in the `eestec.portal` package and their inline documentation.

Source Documentation

List of all Python modules in the `eestec.portal` package and their inline documentation.

TODO: use sphinx.ext.autodoc to document code when we add it

CHAPTER 2

Changelog

2.1 0.1 (unreleased)

- Memberdata fields. [fakedrake]
- Event and EventApplication content types. [iElectric]
- LC content type and forms. [ibi, brodul, andrejpan]
- Theme skeleton. [vilmoss]
- Bootstraping the package. [iElectric, zupo]

CHAPTER 3

License (3-clause BSD)

Copyright (c) 2012, EESTEC International. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of EESTEC International nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL EESTEC INTERNATIONAL BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

CHAPTER 4

Indices and tables

- *genindex*
- *modindex*
- *search*

Symbols

[Plone Object] EestecMember, **4**
[Plone Object] Event, **4**
[Plone Object] EventApplication, **4**
[Plone Object] LC, **4**

C

CP (Contact Person), **3**

E

Event, **3**
Event Application Deadline, **3**

I

International Board, **3**

J

JLC (Junior Local Committee), **3**

L

LC (Local Committee), **3**
Local Board, **3**

M

Member of EESTEC International, **3**

O

Observer, **3**
Observer Candidate, **3**
Organizer, **3**

V

VC-EA (Vice Chairperson for External Affairs), **3**
VC-IA (Vice Chairperson for Internal Affairs), **3**