VISPA

Release 2.0.0

September 15, 2016

Contents

CHAPTER 1

Introduction

1.1 What is VISPA?

1.2 What VISPA is NOT?

1.3 Contents

1.3.1 Installation

Prerequisites

Download Stable Versions

Using *pip* or *easy_install*

Using pip:

```
$ pip install vispa
```

or with easy_install:

```
$ easy_install vispa
```

It is recommended to use *pip* instead of *easy_install*. If you want to download and install VISPA for yourself proceed to the next instructions depending on your platform.

Unix/Mac

You may download the most current version from PyPI

For other releases, browse our download index.

- Unzip/untar the files

- Enter the directory created by the file extraction.
- Type “python setup.py install” to install the VISPA module

Windows

TODO

Next Steps

TODO

Development versions

TODO

1.3.2 VISPA License (GPL v2)

```
                    GNU GENERAL PUBLIC LICENSE
                       Version 2, June 1991

 Copyright (C) 1989, 1991 Free Software Foundation, Inc.,
 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

                            Preamble

  The licenses for most software are designed to take away your
freedom to share and change it.  By contrast, the GNU General Public
License is intended to guarantee your freedom to share and change free
software--to make sure the software is free for all its users.  This
General Public License applies to most of the Free Software
Foundation's software and to any other program whose authors commit to
using it.  (Some other Free Software Foundation software is covered by
the GNU Lesser General Public License instead.)  You can apply it to
your programs, too.

  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
this service if you wish), that you receive source code or can get it
if you want it, that you can change the software or use pieces of it
in new free programs; and that you know you can do these things.

  To protect your rights, we need to make restrictions that forbid
anyone to deny you these rights or to ask you to surrender the rights.
These restrictions translate to certain responsibilities for you if you
distribute copies of the software, or if you modify it.

  For example, if you distribute copies of such a program, whether
gratis or for a fee, you must give the recipients all the rights that
you have.  You must make sure that they, too, receive or can get the
source code.  And you must show them these terms so they know their
```

```
rights.

  We protect your rights with two steps: (1) copyright the software, and
(2) offer you this license which gives you legal permission to copy,
distribute and/or modify the software.

  Also, for each author's protection and ours, we want to make certain
that everyone understands that there is no warranty for this free
software.  If the software is modified by someone else and passed on, we
want its recipients to know that what they have is not the original, so
that any problems introduced by others will not reflect on the original
authors' reputations.

  Finally, any free program is threatened constantly by software
patents.  We wish to avoid the danger that redistributors of a free
program will individually obtain patent licenses, in effect making the
program proprietary.  To prevent this, we have made it clear that any
patent must be licensed for everyone's free use or not licensed at all.

  The precise terms and conditions for copying, distribution and
modification follow.

                    GNU GENERAL PUBLIC LICENSE
   TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

  0. This License applies to any program or other work which contains
a notice placed by the copyright holder saying it may be distributed
under the terms of this General Public License.  The "Program", below,
refers to any such program or work, and a "work based on the Program"
means either the Program or any derivative work under copyright law:
that is to say, a work containing the Program or a portion of it,
either verbatim or with modifications and/or translated into another
language.  (Hereinafter, translation is included without limitation in
the term "modification".)  Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not
covered by this License; they are outside its scope.  The act of
running the Program is not restricted, and the output from the Program
is covered only if its contents constitute a work based on the
Program (independent of having been made by running the Program).
Whether that is true depends on what the Program does.

  1. You may copy and distribute verbatim copies of the Program's
source code as you receive it, in any medium, provided that you
conspicuously and appropriately publish on each copy an appropriate
copyright notice and disclaimer of warranty; keep intact all the
notices that refer to this License and to the absence of any warranty;
and give any other recipients of the Program a copy of this License
along with the Program.

You may charge a fee for the physical act of transferring a copy, and
you may at your option offer warranty protection in exchange for a fee.

  2. You may modify your copy or copies of the Program or any portion
of it, thus forming a work based on the Program, and copy and
distribute such modifications or work under the terms of Section 1
above, provided that you also meet all of these conditions:
```

```
    a) You must cause the modified files to carry prominent notices
    stating that you changed the files and the date of any change.

    b) You must cause any work that you distribute or publish, that in
    whole or in part contains or is derived from the Program or any
    part thereof, to be licensed as a whole at no charge to all third
    parties under the terms of this License.

    c) If the modified program normally reads commands interactively
    when run, you must cause it, when started running for such
    interactive use in the most ordinary way, to print or display an
    announcement including an appropriate copyright notice and a
    notice that there is no warranty (or else, saying that you provide
    a warranty) and that users may redistribute the program under
    these conditions, and telling the user how to view a copy of this
    License.  (Exception: if the Program itself is interactive but
    does not normally print such an announcement, your work based on
    the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole.  If
identifiable sections of that work are not derived from the Program,
and can be reasonably considered independent and separate works in
themselves, then this License, and its terms, do not apply to those
sections when you distribute them as separate works.  But when you
distribute the same sections as part of a whole which is a work based
on the Program, the distribution of the whole must be on the terms of
this License, whose permissions for other licensees extend to the
entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest
your rights to work written entirely by you; rather, the intent is to
exercise the right to control the distribution of derivative or
collective works based on the Program.

In addition, mere aggregation of another work not based on the Program
with the Program (or with a work based on the Program) on a volume of
a storage or distribution medium does not bring the other work under
the scope of this License.

  3. You may copy and distribute the Program (or a work based on it,
under Section 2) in object code or executable form under the terms of
Sections 1 and 2 above provided that you also do one of the following:

    a) Accompany it with the complete corresponding machine-readable
    source code, which must be distributed under the terms of Sections
    1 and 2 above on a medium customarily used for software interchange; or,

    b) Accompany it with a written offer, valid for at least three
    years, to give any third party, for a charge no more than your
    cost of physically performing source distribution, a complete
    machine-readable copy of the corresponding source code, to be
    distributed under the terms of Sections 1 and 2 above on a medium
    customarily used for software interchange; or,

    c) Accompany it with the information you received as to the offer
    to distribute corresponding source code.  (This alternative is
    allowed only for noncommercial distribution and only if you
    received the program in object code or executable form with such
```

```
    an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for
making modifications to it.  For an executable work, complete source
code means all the source code for all modules it contains, plus any
associated interface definition files, plus the scripts used to
control compilation and installation of the executable.  However, as a
special exception, the source code distributed need not include
anything that is normally distributed (in either source or binary
form) with the major components (compiler, kernel, and so on) of the
operating system on which the executable runs, unless that component
itself accompanies the executable.

If distribution of executable or object code is made by offering
access to copy from a designated place, then offering equivalent
access to copy the source code from the same place counts as
distribution of the source code, even though third parties are not
compelled to copy the source along with the object code.

  4. You may not copy, modify, sublicense, or distribute the Program
except as expressly provided under this License.  Any attempt
otherwise to copy, modify, sublicense or distribute the Program is
void, and will automatically terminate your rights under this License.
However, parties who have received copies, or rights, from you under
this License will not have their licenses terminated so long as such
parties remain in full compliance.

  5. You are not required to accept this License, since you have not
signed it.  However, nothing else grants you permission to modify or
distribute the Program or its derivative works.  These actions are
prohibited by law if you do not accept this License.  Therefore, by
modifying or distributing the Program (or any work based on the
Program), you indicate your acceptance of this License to do so, and
all its terms and conditions for copying, distributing or modifying
the Program or works based on it.

  6. Each time you redistribute the Program (or any work based on the
Program), the recipient automatically receives a license from the
original licensor to copy, distribute or modify the Program subject to
these terms and conditions.  You may not impose any further
restrictions on the recipients' exercise of the rights granted herein.
You are not responsible for enforcing compliance by third parties to
this License.

  7. If, as a consequence of a court judgment or allegation of patent
infringement or for any other reason (not limited to patent issues),
conditions are imposed on you (whether by court order, agreement or
otherwise) that contradict the conditions of this License, they do not
excuse you from the conditions of this License.  If you cannot
distribute so as to satisfy simultaneously your obligations under this
License and any other pertinent obligations, then as a consequence you
may not distribute the Program at all.  For example, if a patent
license would not permit royalty-free redistribution of the Program by
all those who receive copies directly or indirectly through you, then
the only way you could satisfy both it and this License would be to
refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under
```

```
any particular circumstance, the balance of the section is intended to
apply and the section as a whole is intended to apply in other
circumstances.

It is not the purpose of this section to induce you to infringe any
patents or other property right claims or to contest validity of any
such claims; this section has the sole purpose of protecting the
integrity of the free software distribution system, which is
implemented by public license practices.  Many people have made
generous contributions to the wide range of software distributed
through that system in reliance on consistent application of that
system; it is up to the author/donor to decide if he or she is willing
to distribute software through any other system and a licensee cannot
impose that choice.

This section is intended to make thoroughly clear what is believed to
be a consequence of the rest of this License.

  8. If the distribution and/or use of the Program is restricted in
certain countries either by patents or by copyrighted interfaces, the
original copyright holder who places the Program under this License
may add an explicit geographical distribution limitation excluding
those countries, so that distribution is permitted only in or among
countries not thus excluded.  In such case, this License incorporates
the limitation as if written in the body of this License.

  9. The Free Software Foundation may publish revised and/or new versions
of the General Public License from time to time.  Such new versions will
be similar in spirit to the present version, but may differ in detail to
address new problems or concerns.

Each version is given a distinguishing version number.  If the Program
specifies a version number of this License which applies to it and "any
later version", you have the option of following the terms and conditions
either of that version or of any later version published by the Free
Software Foundation.  If the Program does not specify a version number of
this License, you may choose any version ever published by the Free Software
Foundation.

  10. If you wish to incorporate parts of the Program into other free
programs whose distribution conditions are different, write to the author
to ask for permission.  For software which is copyrighted by the Free
Software Foundation, write to the Free Software Foundation; we sometimes
make exceptions for this.  Our decision will be guided by the two goals
of preserving the free status of all derivatives of our free software and
of promoting the sharing and reuse of software generally.

                            NO WARRANTY

  11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY
FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW.  EXCEPT WHEN
OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES
PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED
OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.  THE ENTIRE RISK AS
TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU.  SHOULD THE
PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING,
REPAIR OR CORRECTION.
```

```
  12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR
REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES,
INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING
OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED
TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY
YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER
PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE
POSSIBILITY OF SUCH DAMAGES.

                     END OF TERMS AND CONDITIONS

            How to Apply These Terms to Your New Programs

  If you develop a new program, and you want it to be of the greatest
possible use to the public, the best way to achieve this is to make it
free software which everyone can redistribute and change under these terms.

  To do so, attach the following notices to the program.  It is safest
to attach them to the start of each source file to most effectively
convey the exclusion of warranty; and each file should have at least
the "copyright" line and a pointer to where the full notice is found.

    <one line to give the program's name and a brief idea of what it does.>
    Copyright (C) <year>  <name of author>

    This program is free software; you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation; either version 2 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License along
    with this program; if not, write to the Free Software Foundation, Inc.,
    51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this
when it starts in an interactive mode:

    Gnomovision version 69, Copyright (C) year name of author
    Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate
parts of the General Public License.  Of course, the commands you use may
be called something other than `show w' and `show c'; they could even be
mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your
school, if any, to sign a "copyright disclaimer" for the program, if
necessary.  Here is a sample; alter the names:
```

```
  Yoyodyne, Inc., hereby disclaims all copyright interest in the program
  `Gnomovision' (which makes passes at compilers) written by James Hacker.

  <signature of Ty Coon>, 1 April 1989
  Ty Coon, President of Vice

This General Public License does not permit incorporating your program into
proprietary programs.  If your program is a subroutine library, you may
consider it more useful to permit linking proprietary applications with the
library.  If this is what you want to do, use the GNU Lesser General
Public License instead of this License.
```

The starting point for each extenion is a class extending AbstractExtension:

class vispa.server.**AbstractExtension**(*server*)
Base class for Extensions

add_controller(*controller*)
Mount a CherryPy controller using the extension name for path.

Parameters **controller** – filename relative to extension directory

add_workspace_directoy(*directory='workspace'*)
Add files to be transferred to the worker.

Parameters **directoy** – directory relative to extension directory

clear_workspace_instance(*name*, *key=None*, *user=None*, *workspace=None*, *db=None*)

config()

create_topic(*topic=''*, *view_id=None*)

dependencies()
Return a list of Extension names this Extension depends on.

get_workspace_instance(*name*, *key=None*, *user=None*, *workspace=None*, *db=None*, ***kwargs*)

name()
Return the name of the Extension. This name is used as part of the URL.

setup()
Setup the extension.

A minimal example looks like this:

```
from vispa.server import AbstractExtension

class MyExtension(AbstractExtension):

   def name(self):
      return "myext"

   def dependencies(self):
     return []

   def setup(self):
     pass
```

Directories from which extensions are loaded:

- vispa/extensions

- $(var_dir)/extensions
- global packages starting with *vispa_*

In our example, the code above could be placed in a *__init__.py* file in a global package/directory named *vispa_myext-1.0*. By default VISPA loads all extensions it finds, but extensions listed in *vispa.ini*, section *extensions*, option *ignore* will be ignored:

```
[extensions]
ignore = myext
```

CHAPTER 2

Controller

AbstractController

class vispa.controller.**AbstractController**(*mount_static=True*)

cache(*workspace_id*, *key*)

convert(*value*, *flag*)

get(*key*, **args*, ***kwargs*)

mount_static(*path=None*, *url='static'*)

release()

release_database()

release_session()

set_cache(*workspace_id*, *item*, *key=None*)

add_controller

every exposed function is visible

parameters need to be present or 404, or default

cherrypy.tools.ajax()

return values, sring, objects

available cherrypy.request variables

CHAPTER 3

Workspace

self.get_workspace_instance()

CHAPTER 4

Client

TODO

CHAPTER 5

User Guide

5.1 User Management

VISPA's user management is designed for collaborative work on different scales. If provides content management as well as a server-side permission system.

5.1.1 Groups

First of all, it is useful to cluster users together to treat them as one unit. This is done by groups, which are just collections of users as well as other groups. Users can join and leave groups following three different stages of privacy:

- public: open for everyone
- protected: protected by password
- private: new users must be confirmed

Groups are organized by selected users, called managers, who can add and remove members as well as changing the groups name, privacy, password etc.

5.1.2 Projects

Projects are the points, where users and groups are connected to content. Therefore, a project consists of several users and groups as well as a collection of items. In order to control the access to the content, a project assigns permissions via roles to each member (see below). Similar to groups, projects are organized by selected users, called managers.

5.1.3 Workgroups

Workgroups are a lightweight version of projects. They have also content in the form of items but only users are members, no groups. Furthermore, there is no explicit handling of permissions because every user has full access to the workgroups content. There are also managers, which organize the membership of the users.

5.1.4 Roles and Permissions

Permissions are descriptive rights, which allow certain actions of the user. They are collected into roles, which users and groups can fulfill in context of a project. Permissions are abstract rights like 'read items of project' and not

concrete ones like ‘read item X of project Y’. Thus, permissions are usable in every project but get their full context not before being assigned to a certain user or group inside one.

5.1.5 Example

In order to bring light to the user managements concept, here is an example of how it can be used. Let’s assume there is a lecture ‘theoretical physics 101’, which wants to use VISPA. First of all, we look at the people, who belong to the lecture. There we have Professor Einstein, several graduates for the tutorials and many students (amongst others Peter, Paul and Mary). Thus, to make it simpler, we have two groups: * ‘theoretical physics 101 - tutors’ * ‘theoretical physics 101 - students’ which consist of the corresponding people. The tutors group should be private, because only a few dedicated persons belong to them while the students group could be public or protected by a password, which is given to the students during the first lecture. Furthermore, we need a project: ‘theoretical physics 101’. Of course, the lecture has some content e.g. slides or homework sheets and solutions, which can be added to the project as items. So far, we have:

- ‘theoretical physics 101’
  - members:
    - user: Professor Einstein
    - group: ‘theoretical physics 101 - tutors’
      - ...
    - group: ‘theoretical physics 101 - students’
      - Peter
      - Paul
      - Mary
      - ...
  - content:
    - lecture slides
    - homework sheets
    - homework solutions

Now, we need to control, who can do what. Therefore, we use different roles. These could be:

- role 1
  - read slides
  - read sheets
- role 2
  - read slides
  - read sheets
  - read solutions
- role 3
  - read slides
  - read sheets
  - read solutions

  - write solution

Thus, we can assign the roles as follows:

- Professor Einstein:
  - role 3
- ‘theoretical physics 101 - tutors’:
  - role 2
- ‘theoretical physics 101 - students’:
  - role 1

Thus, the access to the different resources is controlled. Additionally, Peter, Paul and Mary want to work on their homework together. Therefore, they can create their own workgroup, which consists only of them and can be utilized e.g. to exchange some files.

5.1.6 Global permissions

In addition to a concept as in the example above, VISPA is capable of managing global permissions. This is done with a global project, by default called ‘VISPA’. A global Permission is e.g. the right to use a certain extension.

CHAPTER 6

Deployment Guide

6.1 Optimized Static Resource

VISPA uses require.js to load JavaScript, CSS and Template files. In normal or development mode each file is loaded dynamically using the AMD loader. Since the VISPA frontend consists of many files, this i quite some overhead. Require.js provides the means to merge and minify all the files using the r.js optimizer. The basic idea is to provide the optimized files instead of the original files, so not path or other flags need to be changed.

Note: The optimizer currently requires that all extensions are placed in the vispa/extensions folder. Symlinks are possible though.

The following steps are required to use optimized builds in VISPA:

1. Install the npm packages:

```
$ npm install
```

1. Add all extensions that shall be optimized in build.js. Add another module, like the filebrowser extension:

```
{
    name: 'extensions/file/static/js/extension',
    exclude: ['vispa/config', 'vispa/vispa']
},
```

1. Run the optimizer. All static files are copied to the build folder.

```
$ node_modules/.bin/r.js -o build.js
```

1. Set the build_path variable in the [web] section in the vispa.ini. Use a relative path to the vispa package folder or an absolute path.

```
[web]
build_path = ../build
```

6.2 User Management

The concept of VISPA's user management is explained in the User Guide. VISPA is able to automatically setup the user management when the server is ramped up. It is highly recommended to use the auto setup in order to use the full capability of the user management. The auto setup can be configured via the vispa.ini. The vispa.ini.sample contains a full example of how such a setup should look:

```
[usermanagement]
# do auto setup?
autosetup = True
# the role/permission setup of the extensions can be accepted or not
accept_extensions = True
# name of the global project
global_project = VISPA
# default user and guest group
user_group = default_user
guest_group = default_guest
# three roles, e.g. with ascending rank
role_1 = Student
role_2 = Tutor
role_3 = Manager
# list of permissions that must exist
permissions = ["project.read_items", "project.create_items", "project.edit_items", "project.delete_it
# assignment of permissions to default roles. permissions must exists (use line above)
role_1_permissions = ["project.read_items"]
role_2_permissions = ["project.read_items", "project.create_items"]
role_3_permissions = ["project.read_items", "project.create_items", "project.edit_items", "project.de
# assignment of roles to user and guest group
user_group_roles = [1]
guest_group_roles = []
```

The comments explain the different options. For a successful setup it is important, that all options are implemented. Otherwise, the automatic setup fails and nothing is committed to the database. It is remarkable, that the permissions "project.*_items" are hard coded into the user management controller and needed for the access to the items of a project.

6.3 Upgrade the Database

VISPA uses alembic to perform database migrations. Per default VISPA upgrades automatically to the latest version of the database schema (head). In a production setup this may not be desirable. To disable automatic database migration set *use_alembic* option in the *alembic* section in the vispa.ini file to *False*:

```
[alembic]
use_alembic = False
```

NOTE: In case of an SQLite database, alembic is never used by VISPA, because SQLite does not support all necessary database migration steps.

6.3.1 Manual Upgrade

To manually upgrade the database schema, you also need to set the *sqlalchemy_url* option in the *database* section:

```
[database]
sqlalchemy.url = sqlite:///var/db/vispa.db

[alembic]
use_alembic = False
script_location = vispa/models/alembic
```

Now you can use the alembic tool to migrate the database:

```
$ alembic -c conf/vispa.ini upgrade head
```

NOTE: The manual upgrade can also be performed on SQLite database, but it can cause errors due to the mentioned limitations of the SQLite dialect. In such a case, the database must deleted and recreated without alembic, as done by VISPA.

CHAPTER 7

Developer Guide

7.1 Extensions

Extensions add functionality to the the VISPA Platform. They provide new views in the browser and new functionality on the server.

7.1.1 Overview

Extensions contain code to be executed on the client browser, HTML, CSS and Javascript, on the VISPA Server, Python, and on the workspaces, also Python.

7.1.2 Tutorial

TODO

7.1.3 Server

The main entry point is the AbstractExtension class in an extension module. Extension modules are either activated in the config file, or by its name: vispa_<name>.

7.1.4 Signals

In VISPA important events are communicated by messages. Events concerning the webserver are communicated using the CherryPy bus. VISPA internal events are send through the VISPA bus. For example, each time a user is logged in, an event with the topic “user.login” is sent. The only parameter is the model.User object:

vispa.publish(“user.login”, user_object)

To receive the message, you need to subscribe to the topic:

vispa.subscribe(“user.login”, login_callback)

Where login_callback is a function which is called with the parameters provided to the publish call. In the case of “user.login”, the user object:

def login_callback(user): print “a new user logged in:”, user.name

Currently the following topics are published by VISPA:

exit, stop, bus.session_added, bus.session_removed, bus.all_user_sessions_removed

7.1.5 Client

Folderstructure:

- myextension/
  - __init__.py
  - controller.py
  - extension.py
  - myextension.ini
  - workspace/
    - ...
  - static/
    - js/
      - myextension.js
      - myfile.js
      - ...
    - css/
      - myextension.css/less
    - templates/
      - ...

RequireJs usage: The namespace is "vispa/myextension/myfile". Vendor modules are simply identified by their name, e.g. "jquery" or "ace". There is no need to register the extension yourself.

7.1.6 Workspace

TODO

7.1.7 User Management

The concept of VISPA's user management is explained in the User Guide. VISPA has an AJAX controller, which gives a simple and convenient access to the user management. The general syntax is:

```
ExtensionView.METHOD("/ajax/um/some_usermanagement_function", ...)
```

Thus, it is a simple AJAX request, e.g.:

```
ExtensionView.GET("/ajax/um/user_get_groups", {}, ...)
```

It is recommended to use the UM controller also on server side python code, because it automatically checks dependencies and permissions. Such a call looks like:

```
Server.controller.ajax.um.some_usermanagement_function(...)
```

For a full documentation of the user management controller look into the class reference.

The user management delivers a cherrypy tool, which can be used to check for global permissions. For further information look into the class reference.

The user management also delivers a simple interface for automatically setting up permissions and roles the extension needs. Therefore, the myextension.ini can be used (look above for folder structure). The myextension.ini is loaded an server ramp up and the extensions can define a set of permission it needs as well as an assignment of the permissions to three default roles. The myextension.ini itself is structured as follows:

```
[usermanagement]
# list of permissions that must exist
permissions = ["myextension.permission1", "myextension.permission2", "myextension.permission3"]
# assignment of permissions to default roles. permissions must exists (use line above)
role_1_permissions = ["myextension.permission1"]
role_2_permissions = ["myextension.permission1", "myextension.permission2"]
role_3_permissions = ["myextension.permission1", "myextension.permission2", "myextension.permission3"
```

The prefix “myextensions” for the permissions is a namespace which should be used by convention.

7.2 Create Database Migration Script

alembic -c <path-to-vispa.ini> revision -m “commit message” –autogenerate

CHAPTER 8

API Reference

TODO: Intro to API reference

8.1 Workspace Handling

8.2 Shared classes

8.2.1 `Emitter()`

8.2.2 `UrlHandler()`

8.2.3 `Socket()`

class `Socket` (*url*[, *options*])
text

Arguments

- **`url`** (*string*) – Url.
- **`options`** (*string*) – Options.

CHAPTER 9

Python Reference Manual

9.1 vispa package

9.1.1 Subpackages

vispa.controller package

Submodules

vispa.controller.ajax module

class vispa.controller.ajax.**AjaxController**(*root*)

Bases: *vispa.controller.AbstractController*

addworkspace(*name*, *host*, *login*, *key=None*, *cmd=None*)

connectworkspace(*wid*, *password=None*)

deleteworkspace(*wid*)

disconnectworkspace(*wid*)

editworkspace(*wid*, *name=None*, *host=None*, *login=None*, *key=None*, *cmd=None*)

feedback(*content*, *anonymous*)

forgotpassword(*username*)

getjson(*key*, *wid=None*)

getworkspacedata(*wid=None*)

localuser()

login(*username*, *password*)

register(*username*, *email*)

setjson(*key*, *value*, *wid=None*)

setpassword(*hash*, *password*)

vispa.controller.bus module

class vispa.controller.bus.**BusController**
Bases: *vispa.controller.AbstractController*

index(**args*, ***kwargs*)

poll(*timeoutms=10000*)

send(**args*, ***kwargs*)

vispa.controller.error module

class vispa.controller.error.**ErrorController**
Bases: object

This class provides custom error-catching functions that are inserted into the cherrypy config.

STATUSMAP = {'404': 'filenotfound', '403': 'forbidden', '401': 'unauthorized', '400': 'badrequest', '500': 'internalserver

TMPL = "<html><head><meta http-equiv='refresh' content='0;url=%s' /></head></html>"

get_error_data()

index(**args*, ***kwargs*)

vispa.controller.filesystem module

class vispa.controller.filesystem.**FSAjaxController**(*mount_static=True*)
Bases: *vispa.controller.AbstractController*

checkpermissions(*path*)

compress(*paths*, *path*, *name=''*, *isTmp=False*)

createfile(*path*, *name*)

createfolder(*path*, *name*)

decompress(*file*)

exists(*path*, *filetype=None*)

expand(*path*)

filecount(*path*, *watch_id=None*)

filelist(*path*, *filefilter=None*, *reverse=False*, *watch_id=None*)

getfacl(*path*)
Get the facl of a certain path.

Parameters **path** – path of interest

Returns list of tuples with (type, user, mode[, default mode])

getfile(*path*, *watch_id=None*, *utf8=False*)

getsuggestions(*path*, *length=10*, *append_hidden=True*)

getworkspaceini(*request*, *fail_on_missing=False*)

isbrowserfile(*path*)

move(*source*, *destination*)

paste(*path*, *paths*, *cut*)

remove(*path*)

rename(*path*, *name*, *new_name*)

savefile(*path*, *content*, *watch_id=None*, *utf8=False*)

setfacl(*path*, *type*, *name*, *mode*, *remove=False*, *recursive=False*, *default=False*)
Set the facl entry for a certain user or group. On remote side, the commandline tool 'setfacl' is used, so this method delivers an interface to that.

Parameters

- **path** – path of interest
- **type** – type, either 'user', 'group', 'mask' or 'other'
- **name** – name of the user or group of interest
- **mode** – mode to be set
- **remove** – remove all extended facl entries (option '-x')
- **recursive** – apply changes to all files and directories recursivley (option '-R')
- **default** – edit the default values of an facl entry (option '-d')

Raises AjaxException if type is not 'user', 'group', 'mask' or 'other'

setworkspaceini(*request*)

unwatch(*watch_id=None*)

upload(**args*, ***kwargs*)

watch(*path*, *watch_id*)

class vispa.controller.filesystem.**FSController**
Bases: *vispa.controller.AbstractController*

getfile(*path*, *download=None*, *deleteoncomplete=None*, ***kwargs*)

thumbnail(*path*, *width=100*, *height=100*, ***kwargs*)

vispa.controller.root module

class vispa.controller.root.**RootController**(*server*)
Bases: *vispa.controller.AbstractController*

graceful_shutdown(**args*, ***kwargs*)

guest_login(**args*, ***kwargs*)

index(**args*, ***kwargs*)

login(**args*, ***kwargs*)

logout(*path='/'*)

mount_extension_controller(*mountpoint*, *controller*)

password(*hash*, **args*, ***kwargs*)

status(**args*, ***kwargs*)

workspace_data(*workspace=None*, *keys=None*)

vispa.controller.usermanagement module

class vispa.controller.usermanagement.**UMAjaxController**(*mount_static=True*)
Bases: *vispa.controller.AbstractController*

The UMAjaxController inherits all methods, which are neccessery for the user management. Most of the function names are chosen self explanantory, while the first word corresponds to the object of interest and the rest to the action performed on the object, e.g. user_get_groups returns the groups of the user, who created the request. If the returned Objects are (lists of) database classes, they are converted into python/JSON objects with the values of interest as properties, e.g.: user_get_groups() returns a list of objects with "name" and "status" as properties.

group_add_child_group(*parent_group*, *child_group*)
Add child group to parent group.

Parent_group concerning parent group

Child_group concerning child group

Raises AjaxException

group_add_manager(*group*, *user*)
Add manager to group.

Parameters

- **group** – concerning group
- **user** – user to add as manager

Raises AjaxException

group_add_parent_group(*parent_group*, *child_group*)
Add parent group to child group.

Parent_group concerning parent group

Child_group concerning child group

Raises AjaxException

group_add_user(*group*, *user*)
Add user to group

Parameters

- **group** – concerning group
- **user** – user to be added

Raises AjaxException

group_confirm_child_group(*parent_group*, *child_group*)
Confirm child group in parent group.

Parent_group concerning parent group

Child_group concerning child group

Raises AjaxException

group_confirm_user(*group*, *user*)
Confirm user.

Parameters

- **group** – concerning group
- **user** – user to be confirmed

Raises AjaxException

group_create(*name*, *privacy=0*, *password=''*)
Create group.

Parameters **name** – name of group

Privacy privacy

Password password

Raises AjaxException

group_delete(*group*)
Delete group.

Parameters **group** – concerning group

Raises AjaxException

group_enter_parent_group(*parent_group*, *child_group*, *password=''*)
Enter parent group as child group.

Parent_group concerning parent group

Child_group concerning child group

Parameters **password** – password if parent group is protected

Raises AjaxException

group_get(*group*)
Get group by its name.

Parameters **group** – name of the group

Returns dict with "name", "privacy" and "status"

Raises AjaxException

group_get_all()
Get all groups.

Returns list of dict with "name", "privacy" and "status"

group_get_child_groups(*group*, *recursion_depth=-1*)
Get child groups of group.

Parameters **group** – concerning group

Recursion_depth recursion depth for getting child groups

Returns list of dict with "name" and "status" of membership

Raises AjaxException

group_get_managers(*group*)
Get managers of group.

Parameters **group** – concerning group

Returns list of strings (names of managers)

Raises AjaxException

group_get_parent_groups(*group*, *recursion_depth=-1*)
Get parent groups of group.

Parameters **group** – concerning group

Recursion_depth recursion depth for getting parent groups

Returns list of dict with "name" and "status" of membership

Raises AjaxException

group_get_users(*group*, *recursion_depth=-1*)
Get users of group.

Parameters

- **group** – concerning group
- **recursion_depth** – recursion_depth for getting users

Returns list of dict with "name" and "status" and membership

Raises AjaxException

group_leave_parent_group(*parent_group*, *child_group*)
Leave parent group as child group.

Parent_group concerning parent group

Child_group concerning child group

Raises AjaxException

group_remove_child_group(*parent_group*, *child_group*)
Remove child group from parent group.

Parent_group concerning parent group

Child_group concerning child group

Raises AjaxException

group_remove_manager(*group*, *manager*)
Remove manager from group.

Parameters

- **group** – concerning group
- **manager** – manager to be removed

Raises AjaxException

group_remove_user(*group*, *user*)
Remove user.

Parameters

- **group** – concerning group
- **user** – user to be removed

Raises AjaxException

group_rename(*group*, *name*)
Rename group.

Parameters

- **group** – concerning group
- **name** – new name

Raises AjaxException

group_set_password(*group*, *password*)
Set password of group.

Parameters

- **group** – concerning group
- **password** – new password

Raises AjaxException

group_set_privacy(*group*, *privacy*)
Set privacy of group.

Parameters

- **group** – concerning group
- **privacy** – new privacy

Raises AjaxException

group_set_status(*group*, *status*)
Set status of group.

Parameters

- **group** – concerning group
- **status** – new status

permission_create(*name*)
Create permission.

Parameters **name** – name of new permission

Raises AjaxException

permission_delete(*permission*)
Delete permission.

Parameters **permission** – concerning permission

Raises AjaxException

permission_get_all()
Get all permissions.

Returns list of strings

Raises AjaxException

permission_rename(*permission*, *name*)
Rename permission.

Parameters

- **permission** – concerning permission
- **name** – new name of permission

Raises AjaxException

project_add_group(*project*, *group*)
Add group to project.

Parameters

- **project** – concerning project
- **group** – group to be added

Raises AjaxException

project_add_manager(*project*, *user*)
Add manager to project.

Parameters

- **project** – concerning project
- **user** – user to be added as manager

Raises AjaxException

project_add_user(*project*, *user*)
Add user to project.

Parameters

- **project** – concerning project
- **user** – user to be added

Raises AjaxException

project_create(*name*)
Create project.

Parameters **name** – name of new project

Raises AjaxException

project_delete(*project*)
Delete project.

Parameters **project** – concerning project

Raises AjaxException

project_get_all()
Get all projects.

Returns list of dict with “name” and “status”

Raises AjaxException

project_get_groups(*project*)
Get groups of project.

Parameters **project** – concerning project

Returns list of dict with “name”

Raises AjaxException

project_get_managers(*project*)
Get managers of project.

Parameters **project** – concerning project

Returns list of dict with “name”

Raises AjaxException

project_get_roles_of_group(*project*, *group*)
Get roles of group inside project.

Parameters

- **project** – concerning project
- **group** – concerning group

Returns list of strings

Raises AjaxException

project_get_roles_of_user(*project*, *user*)
Get roles of user inside project.

Parameters

- **project** – concerning project
- **user** – concerning user

Returns list of strings

Raises AjaxException

project_get_users(*project*)
Get users of project.

Parameters **project** – concerning project

Returns list of dict with “name”

Raises AjaxException

project_remove_group(*project*, *group*)
Remove group from project.

Parameters

- **project** – concerning project
- **group** – group to be removed

Raises AjaxException

project_remove_manager(*project*, *manager*)
Remove manager from project.

Parameters

- **project** – concerning project
- **manager** – manager to be removed

Raises AjaxException

project_remove_user(*project*, *user*)
Remove user from project.

Parameters

- **project** – concerning project
- **user** – user to be removed

Raises AjaxException

project_rename(*project*, *name*)
Rename project.

Parameters

- **project** – concerning project
- **name** – new name of project

Raises AjaxException

project_set_roles_of_group(*project*, *group*, *roles*)
Set roles of group inside project.

Parameters

- **project** – concerning project
- **group** – concerning group
- **roles** – JSON encoded list of roles with "name" and "assignment" bool

Raises AjaxException

project_set_roles_of_user(*project*, *user*, *roles*)
Set roles of user inside project.

Parameters

- **project** – concerning project
- **user** – concerning user
- **roles** – JSON encoded list of roles with "name" and "assignment" bool

Raises AjaxException

project_set_status(*project*, *status*)
Set status of project.

Parameters

- **project** – concerning project
- **status** – new status of project

Raises AjaxException

role_create(*name*)
Create role.

Parameters **name** – name of new role

Raises AjaxException

role_delete(*role*)
Delete role.

Parameters **role** – concerning role

Raises AjaxException

role_get_all()
Get all roles.

Returns list of strings

role_get_permissions(*role*)
Get permissions of role.

Parameters **role** – concerning role

Returns list of strings

Raises AjaxException

role_rename(*role*, *name*)
Rename role.

Parameters

- **role** – concerning role
- **name** – new name of role

Raises AjaxException

role_set_permissions(*role*, *permissions*)
Set permissions of role.

Parameters

- **role** – concerning role
- **permissions** – JSON encoded list of permissions with "name" and "assignment" bool

Raises AjaxException

user_enter_group(*group*, *password=''*)
Enter group as current user.

Parameters

- **group** – concerning group
- **password** – password for protected groups

Raises AjaxException

user_get_groups()
Get groups of the current user.

Returns list of dict with "name" and "status"

user_get_managed_groups()
Get managed groups of current user.

Returns list of dict with "name", "privacy" and "status"

user_get_managed_projects()
Get managed projects of current user.

Returns list of dict with "name" and "status"

user_get_managed_workgroups()
Get managed workgroups of current user.

Returns list of dict with "name"

user_get_permissions(*project*)
Get permissions of current user in project.

Parameters **project** – concerning project

Returns list of strings

Raises AjaxException

user_get_projects()
Get projects of current user.

Returns list of dict with "name" and "status"

user_get_roles(*project*)
Get roles of current user in project.

Parameters **project** – concerning project

Returns list of dict with "name"

Raises AjaxException

user_get_workgroups()
Get workgroups of current user.

Returns list of dict with "name"

user_leave_group(*group*)
Leave group as current user.

Parameters **group** – concerning group

Raises AjaxException

user_leave_workgroup(*workgroup*)
Leave workgroup as current user.

Raises AjaxException

workgroup_add_manager(*workgroup*, *user*)
Add manager to workgroup.

Parameters

- **workgroup** – concerning workgroup
- **user** – user to be added as manager

Raises AjaxException

workgroup_add_user(*workgroup*, *user*)
Add user to workgroup.

Parameters

- **workgroup** – concerning workgroup
- **user** – user to be added

Raises AjaxException

workgroup_create(*name*)
Create workgroup.

Parameters **name** – name of new workgroup

Raises AjaxException

workgroup_delete(*workgroup*)
Delete workgroup.

Parameters **workgroup** – workgroup to be deleted

Raises AjaxException

workgroup_get_managers(*workgroup*)
Get managers of workgroup.

Parameters **workgroup** – concerning workgroup

Returns list of dict with "name"

Raises AjaxException

workgroup_get_users(*workgroup*)
Get users of workgroup.

Parameters **workgroup** – concerning workgroup

Returns list of dict with "name"

Raises AjaxException

workgroup_remove_manager(*workgroup*, *manager*)
Remove manager from workgroup.

Parameters

- **workgroup** – concerning workgroup
- **manager** – manager to be removed

Raises AjaxException

workgroup_remove_user(*workgroup*, *user*)
Remove user from workgroup.

Parameters

- **workgroup** – concerning workgroup
- **user** – user to be removed

Raises AjaxException

workgroup_rename(*workgroup*, *name*)
Rename workgroup.

Parameters

- **workgroup** – concerning workgroup
- **name** – new name of workgroup

Raises AjaxException

Module contents

class vispa.controller.**AbstractController**(*mount_static=True*)
Bases: object

cache(*workspace_id*, *key*)

convert(*value*, *flag*)

get(*key*, **args*, ***kwargs*)

mount_static(*path=None*, *url='static'*)

release()

release_database()

release_session()

set_cache(*workspace_id*, *item*, *key=None*)

class vispa.controller.**StaticController**(*path*)
Bases: object

vispa.controller.**strongly_expire**(*func*)
Decorator that sends headers that instruct browsers and proxies not to cache.

vispa.extensions package

Subpackages

vispa.extensions.codeeditor package

Subpackages

vispa.extensions.codeeditor.workspace package

Module contents

class vispa.extensions.codeeditor.workspace.**CodeEditorRpc**(*window_id*, *view_id*)

BURST_BUFFER = 8000

BURST_DELAY = 0.25

MAX_BURST = 1000

MAX_RATE = 2000

SIGTEM_SIGKILL_DELAY = 0.1

abort()

close()

runningjob()

start(*cmd*, *base*)

vispa.extensions.codeeditor.workspace.**expand**(*path*)

Submodules

vispa.extensions.codeeditor.controller module

class vispa.extensions.codeeditor.controller.**EditorController**(*mount_static=True*)
Bases: *vispa.controller.AbstractController*

abort()

close()

execute(*cmd*, *base*)

getrpc()

runningjob()

Module contents

class vispa.extensions.codeeditor.**CodeEditorExtension**(*server*)

Bases: *vispa.server.AbstractExtension*

dependencies()

name()

setup()

vispa.extensions.core package

Module contents

class vispa.extensions.core.**CoreController**(*mount_static=True*)

Bases: *vispa.controller.AbstractController*

class vispa.extensions.core.**CoreExtension**(*server*)

Bases: *vispa.server.AbstractExtension*

dependencies()

name()

setup()

vispa.extensions.demo package

Subpackages

vispa.extensions.demo.workspace package

Module contents

class vispa.extensions.demo.workspace.**DemoRpc**

ls(*path*)

Submodules

vispa.extensions.demo.controller module

class vispa.extensions.demo.controller.**DemoController**(*mount_static=True*)

Bases: *vispa.controller.AbstractController*

ls(*path=None*)

Module contents

class vispa.extensions.demo.**DemoExtension**(*server*)

Bases: *vispa.server.AbstractExtension*

dependencies()

name()

setup()

vispa.extensions.dummy package

Subpackages

vispa.extensions.dummy.workspace package

Module contents

class vispa.extensions.dummy.workspace.**DummyRpc**

dummy()

wait(*cb*)

class vispa.extensions.dummy.workspace.**Scheduler**

Submodules

vispa.extensions.dummy.controller module

class vispa.extensions.dummy.controller.**DummyController**(*extension*)

Bases: *vispa.controller.AbstractController*

data()

failure(*msg=None*)

sigtest(*o*, *l*, *i*, *s*)

Module contents

class vispa.extensions.dummy.**DummyExtension**(*server*)

Bases: *vispa.server.AbstractExtension*

dependencies()

name()

setup()

vispa.extensions.file package

Submodules

vispa.extensions.file.controller module

class vispa.extensions.file.controller.**FileController**

Bases: *vispa.controller.AbstractController*

Module contents

class vispa.extensions.file.**FileBrowserController**(*mount_static=True*)

Bases: *vispa.controller.AbstractController*

class vispa.extensions.file.**FileBrowserExtension**(*server*)

Bases: *vispa.server.AbstractExtension*

dependencies()

name()

setup()

vispa.extensions.gallery package

Module contents

class vispa.extensions.gallery.**GalleryController**(*mount_static=True*)
Bases: *vispa.controller.AbstractController*

class vispa.extensions.gallery.**GalleryExtension**(*server*)
Bases: *vispa.server.AbstractExtension*

dependencies()

name()

setup()

vispa.extensions.ldap-export package

Module contents

vispa.extensions.terminal package

Subpackages

vispa.extensions.terminal.workspace package

Module contents

class vispa.extensions.terminal.workspace.**Terminal**
Bases: object

close()

communicate(*input_data*, *timeout=0.05*)

open(*window_id*, *view_id*, *shell=None*)

read(*timeout=0.05*, *buffer_size=16384*)

resize(*w*, *h*)

write(*input_data*)

Module contents

class vispa.extensions.terminal.**TerminalController**(*mount_static=True*)
Bases: *vispa.controller.AbstractController*

close(*tid*)

communicate(*tid*, *input_data*)

open()

read(*tid*, *timeout=10*)

resize(*tid*, *w*, *h*)

write(*tid*, *input_data*)

class vispa.extensions.terminal.**TerminalExtension**(*server*)
Bases: *vispa.server.AbstractExtension*

dependencies()

name()

setup()

Module contents

vispa.models package

Subpackages

vispa.models.alembic package

Submodules

vispa.models.alembic.env module

Module contents
vispa.models.alembic.**migrate**(*db*, *revision='head'*)

Submodules

vispa.models.group module

class vispa.models.group.**Group_User_Assoc**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

The Group_User_Assoc object is a association object representing the membership of a user in a group. In addition to the membership itself, it has a status flag, which indicates, whether the membership is confirmed or not.

CONFIRMED = 0

UNCONFIRMED = 1

group_id

status

user

user_id

class vispa.models.group.**Group_Group_Assoc**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

The Group_Group_Assoc object is a association object representing the membership of one group in another, called child group and parnet group. In addition to the membership itself, it has a status flag, which indicates whether the membership is confirmed or not.

CONFIRMED = 0

UNCONFIRMED = 1

child_group

child_group_id

parent_group_id

status

class vispa.models.group.**Group**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

A group is a collection of users and other groups. Every group has an id, a unique name and a creation timestamp. Furthermore there is a privacy integer as follows:

> 0 - public: the group and its members can be seen by everyone and everyone can join 1 - protected: the group is visible, members only for other members, joining via request to group manager 2 - private: group and members are invisible, joining via password

A group is organized by managers, who can edit the name, privacy etc. as well as the memberships of users and group. Additionally, the managers can join the group into a parentgroup.

ACTIVE = 1

DELETED = 2

INACTIVE = 0

PRIVATE = 2

PROTECTED = 1

PUBLIC = 0

add_child_group(*session*, *child_group*, *confirmed=1*)
Add child group to parent group. Loops in groups are permitted. The necessary Group_Group_Assoc object is added to the database.

Parameters

- **session** – current session of database
- **child_group** (Group) – concerning child group

Raises TypeError of child_group is not instance of Group

Raises Exception if a loop is detected

Raises Exception if child_group is already in the group

add_manager(*user*)
Add manager to group.

Parameters **user** (User) – user which has to be added

Raises TypeError if user is not instance of User

Raises Exception if user is already manager of the group,

add_user(*session*, *user*, *confirmed=1*)
Add user to group. The necessary Group_User_Assoc object is added to the database. By default, the membership is unconfirmed.

Parameters

- **session** – current session of database
- **user** (User) – user which has to be added

Raises TypeError if user is not instance of User

Raises Exception if user already in group

static all (*session*)
Get all groups.

Parameters **session** – current session of database

Returns list of all Group objects

child_groups

confirm_child_group (*child_group*)
Confirm a child group in a private parent group.

Parameters **child_group** (Group) – concerning child group

Raises TypeError of child_group is not instance of Group

Raises Exception if child_group is not in the group

confirm_user (*user*)
Confirm a user in a private group.

Parameters **user** (User) – concerning user

Raises TypeError if user is not instance of User

Raises Exception if user not in group

created

delete ()
Delete group. Internally, the delete flag is set, its not deleted from the database.

static get (*session*, *group*)
Get a group by name or id. If the group parameter is an instance of Group, it is directly returned.

Parameters

- **session** – current session of database
- **group** – name or id to look for. if group is instance of Group, group is returned

Returns Group

Raises Exception if group parameter invalid or no group can be found

static get_by_id (*session*, *gid*)
Get a group by its id.

Parameters

- **session** – current session of database
- **gid** – id to look for

Returns Group or None if inexistent

static get_by_name (*session*, *name*)
Get a group by its name.

Parameters

- **session** – current session of database
- **name** – name to look for

Returns Group or None if inexistent

get_child_groups(*recursion_depth=-1*)
Get child groups of a parent group. This function works as get_users regarding the recursion depth.

Parameters **recursion_depth** – number of steps for recursion

Returns Set of Group_Group_Assoc objects

get_managers()
Get managers of the group.

Returns list of User objects, which are managers of the group

static get_or_create_by_name(*session*, *name*, *privacy=0*, *password=''*)
Get a group by name or create it as public group, if it does not exists.

Parameters

- **session** – current session of database
- **name** – name of the group

Returns Group

Raises Exception if no group with name exists and either name or privacy are invalid

get_parent_groups(*recursion_depth=-1*)
Get parent groups (confirmed and unconfirmed) of a child group. This function works as get_child_groups but in opposite direction. The recursion is only done for confirmed memberships.

Parameters **recursion_depth** – number of steps for recursion

Returns Set of Group_Group_Assoc objects

get_projects()
Returns active projects of the group and all (confirmed) parent groups.

Returns Set of Project_Group_Assoc objects

get_users(*recursion_depth=-1*)
Get all users of the group. The recursion depth for the child groups can be given as additional argument. E.g. 0 is only the group itself, 1 also includes all direct child groups. -1 belongs to infinite recursion depth. Notice: since loops in groups are permitted, there is only a finite number of recusions for a finite number of groups.

Parameters **recursion_depth** – number of recursions for child groups

Returns Set of Group_User_Assoc objects

id

managers

name

password

privacy

remove_child_group(*session*, *child_group*)
Remove child group from parent group. The concerning Group_Group_Assoc object is deleted form the database.

Parameters

- **session** – current session of database
- **child_group** (Group) – concerning child group

Raises TypeError of child_group is not instance of Group

Raises Exception if child_group is not in the group

remove_manager(*manager*)
Remove manager from group.

Parameters **manager** (User) – manager which has to be removed

Raises TypeError if manager is not instance of User

Raises Exception if manager is not manager of the group

remove_user(*session*, *user*)
Remove user from group. The concerning Group_User_Assoc object is deleted from the database.

Parameters

- **session** – current session of database
- **user** (User) – user which has to be removed

Raises TypeError if user is not instance of User

Raises Exception if user not in group

rename(*session*, *newname*)
Rename group.

Parameters

- **session** – current session of database, used to validate new name
- **newname** – new name of the group

Raises Exception if newname is invalid or already exitent

set_password(*password*)
Set the password of a group.

Parameters **password** – new password

Raises Exception if password is invalid

set_privacy(*privacy*)
Set privacy of group.

Parameters **privacy** – new privacy of the group (0, 1, or 2)

Raises Exception if privacy is invalid

set_status(*status*)
Set status of group.

Parameters **status** – new status of the group, either 0 for inactive or 1 for active

Raises Exception if status is invalid

status

users

vispa.models.jsondata module

class vispa.models.jsondata.**JSONData**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

get_info_data()

static get_item(*db*, *user_id*, *key*, *workspace_id*, *create=False*)

static get_value(*db*, *user_id*, *key*, *workspace_id*)

static get_values_by_key(*db*, *user_id*, *key=None*)

id

key

static set_value(*db*, *user_id*, *key*, *workspace_id*, *value*)

timestamp

user_id

value

workspace_id

vispa.models.project module

class vispa.models.project.**Project_User_Assoc**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

The Project_User_Assoc object is an association object, which connects a project and a user. It has a many-to-many relationship to roles, which gives the user roles inside the project. For the association table, the project and the user id are used.

get_permissions()
Get the permissions of this Project User connection.

Returns Set of Permission objects

project_id

roles

user

user_id

class vispa.models.project.**Project_Group_Assoc**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

The Project_Group_Assoc object is an association object, which connects a project and a group. It has a many-to-many relationship to roles, which gives the group roles inside the project. For the association table, the project and the group id are used.

get_permissions()
Get the permissions of this Project Group connection.

Returns Set of Permission objects

group

group_id

project_id

roles

class vispa.models.project.**Project**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

A project connects users and groups to some content (ProjectItems) and also assigns permissions to them via roles. Permissions can be e.g. read and write rights on the content.

ACTIVE = 1

DELETED = 2

INACTIVE = 0

add_group(*session*, *group*)
Adds group to project without any roles. The necessary Project_Group_Assoc object is added to the database.

Parameters

- **session** – current session of database
- **group** (Group) – concerning group

Raises TypeError if group is not instance of Group

add_manager(*user*)
Adds new manager to project.

Parameters **user** (User) – concerning user

Raises TypeError if user is not instance of User

add_roles_to_group(*group*, *roles*)
Add role to group.

Parameters

- **group** (Group) – concerning group
- **roles** (*list of Role objects*) – list of roles

Raises TypeError if type if group or roles is invalid

Raises Exception if group not in project

add_roles_to_user(*user*, *roles*)
Add role to a user.

Parameters

- **user** (User) – concerning user
- **roles** (*list of Role objects*) – list of roles

Raises TypeError if type if user or roles is invalid

Raises Exception if user not in project

add_user(*session*, *user*)
Adds user to project without any roles.

Parameters

- **session** – current session of database
- **user** (User) – concerning user

Raises TypeError if user is not instance of User

Raises Exception if user already in project

static all(*session*)
Returns all existing projects.

Parameters **session** – current session of database

Returns list of Project objects

static create(*session*, *name*)
Create new project.

Parameters

- **session** – current session of database
- **name** – name of the project

Raises Exception if name is invalid or project with same name already exists

created

delete()
Delete project. Internally, the delete flag is set, its not really deleted.

static get(*session*, *project*)
Get a project by name. If the project parameter is an instance of Project, it is directly returned.

Parameters

- **session** – current session of database
- **project** – name to look for. If project is instance of Project it is directly returned

Returns Project object

Raises Exception if project parameter is invalid or no project can be found

static get_by_id(*session*, *gid*)
Get a project by its id.

Parameters

- **session** – current session of database
- **gid** – given id, which is looked for

Returns Project object or None if inexistent

static get_by_name(*session*, *name*)
Get a project by its name.

Parameters

- **session** – current session of database
- **name** – name which is looked for

Returns Project object or None if inexistent

get_groups()
Get groups of project.

Returns list of Project_Group_Assoc objects

get_items(*itemtype=None*)
Get ProjectItems.

Parameters **itemtype** – optional selector on the item type

Returns list of ProjectItem objects

get_managers()
Get managers of project.

Returns list of User objects

static get_or_create_by_name(*session*, *name*)
Get or create a project by its name.

Parameters

- **session** – current session of database
- **name** – name of the project

Returns Project

get_roles_of_group(*group*)
Return roles of group.

Parameters **group** (Group) – concerning group

Returns list of Role objects

Raises TypeError if group is not instance of Group

Raises Exception if group not in project

get_roles_of_user(*user*)
Return roles of user.

Parameters **user** (User) – concerning user

Returns list of Role objects

Raises TypeError if user is not instance of User

Raises Exception if user not in project

get_users()
Returns users of project.

Returns list of Project_User_Assoc objects

groups

has_group(*group*)
Check if group is already in project.

Parameters **group** (Group) – concerning group

Returns bool whether group is in project

Raises TypeError if group is not instance of Group

id

items

managers

name

remove_group(*session*, *group*)
Removes group from Project.

Parameters

- **session** – current session of database
- **group** (Group) – concerning group

Raises TypeError if group is not instance of Group

Raises Exception if group is not in project

remove_manager(*manager*)
Removes manager from Project.

Parameters **manager** (User) – concerning manager

Raises TypeError if manager is not instance of User

Raises Exception if manager is not manager of project

remove_user(*session*, *user*)
Removes user from Project. The concerning Project_User_Assoc is deleted from the database.

Parameters

- **session** – current session of database
- **user** (User) – concerning user

Raises TypeError if user is not instance of User

Raises Exception if user not in project

rename(*session*, *newname*)
Rename project.

Parameters

- **session** – current session of database, used to validate new name
- **newname** – new name of the project

Raises Exception if newname is invalid or project with newname already exists

set_roles_of_group(*group*, *roles*)
Sets the roles of a group in a project.

Parameters

- **group** (Group) – concerning group
- **roles** (*list of Role objects*) – list of roles

Raises TypeError if type if group or roles is invalid

Raises Exception if group not in project

set_roles_of_user(*user*, *roles*)
Sets the roles of a user in a project.

Parameters

- **user** (User) – concerning user
- **roles** (*list of Role objects*) – list of roles

Raises TypeError if type if user or roles is invalid

Raises Exception if user not in project

set_status(*status*)
Set status of project.

Parameters **status** – new status of the project, either 0 for inactive or 1 for active

Raises Exception if status is invalid

status

users

class vispa.models.project.**ProjectItem**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

A project item represents the actual content of a project. It is simply an object consisting of a type (e.g. file) and its content (e.g. the path of the file)

content

static create(*session*, *project*, *itemtype*, *content*)
Create an project item. The item is added to the database.

Parameters

- **session** – current session of database
- **project** – project of the new item
- **itemtype** – type of the item
- **content** – content of the item

delete(*session*)
Delete the item. It is removed from the database.

Parameters **session** – current session of database

static get(*session*, *item*)
Get a ProjectItem by its id. If the item is already a ProjectItem object, it is directly returned.

Parameters

- **session** – current session of database
- **item** – id to look for. if type of item is ProjectItem, item is returned

Returs ProjectItem

Raises Exception if item is invalid or no item can be found

static get_by_id(*session*, *gid*)
Get a ProjectItem by its id.

Parameters

- **session** – current session of database
- **gid** – given id, which is looked for

Returns ProjectItem or None if inexistent

get_project()
Get the project of the item.

Returns Project

id

itemtype

project_id

set_content(*content*)
Set the content of the item.

Parameters **content** – new content

vispa.models.role module

class vispa.models.role.**Permission**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

A Permission object is self-explanatory a permission for a user. It is only characerized by its name, which explains its meaning.

static all(*session*)
Get all existing permissions.

Parameters **session** – current session of database

Returns list of Permission objects

static create(*session*, *name*)
Create new permission.

Parameters

- **session** – current session of database
- **name** – name of the permission

Raises Exception if name is invalid or already in use

created

delete(*session*)
Delete permission. It is also deleted from the database.

Parameters **session** – current session of database

static get(*session*, *permission*)
Get a permissionby name. If permission is already a Permission object, it is directly returned.

Parameters

- **session** – current session of database
- **permission** – name to look for. if type of permission is Permission, it is returned

Returns Permission

Raises Exception if permission parameter is invalid or no permission can be found

static get_by_id(*session*, *gid*)
Get a permission by its id.

Parameters

- **session** – current session of database
- **gid** – given id, which is looked for

Returns Permission or None

static get_by_name(*session*, *name*)
Get a permission by its name.

Parameters

- **session** – current session of database
- **name** – given name, which is looked for

Returns Permission or None

static get_or_create_by_name(*session*, *name*)
Get or create a permission by name.

Parameters

- **session** – current session of database
- **name** – name of the permission

Returns Permission

id

name

rename(*session*, *newname*)
Rename Permission.

Parameters

- **session** – current session of database
- **newname** – new name of the permission

Raises Exception if newname is already in use

class vispa.models.role.**Role**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

A Role is a collection of permissions (Permission objects).

add_permissions(*permissions*)
Add a list of permissions to the role.

Parameters **permissions** (*list of Permission objects*) – list of permissions

Raises TypeError if type of permissions is invalid

static all(*session*)
Get all existing permissions.

Parameters **session** – current session of database

Returns list of Role objects

static create(*session*, *name*)
Create new role.

Parameters

- **session** – current session of database
- **name** – name of the role

Raises Exception if name is invalid or already in use

created

delete(*session*)
Delete role. It is also deleted from the database.

Parameters **session** – current session of database

static get(*session*, *role*)
Get a role by name. If role is already a Role object, it is directly returned.

Parameters

- **session** – current session of database
- **role** – name to look for. if type of role is Role, role is returned

Returns Role

Raises Exception if role parameter is invalid or no role can be found

static get_by_id(*session*, *gid*)
Get a role by its id.

Parameters

- **session** – current session of database
- **gid** – given id, which is looked for

Returns Role or None

static get_by_name(*session*, *name*)
Get a role by its name.

Parameters

- **session** – current session of database
- **name** – given name, which is looked for

Returns Role or None

static get_or_create_by_name(*session*, *name*)
Get or create a role by name.

Parameters

- **session** – current session of database
- **name** – name of the role

Returns Role

id

name

permissions

rename(*session*, *newname*)
Rename role.

Parameters

- **session** – current session of database
- **newname** – new name of the role

Raises Exception if newname is already in use

set_permissions(*permissions*)
Sets the permissions of a role.

Parameters **permissions** (*list of Permission objects*) – list of permissions

Raises TypeError if type of permissions is invalid

vispa.models.user module

class vispa.models.user.**User**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

ACTIVE = 1

FORBIDDEN_NAMES = ['data', 'guest', 'global', 'user', 'delete', 'select', 'insert', 'update', 'drop']

INACTIVE = 0

MIN_PW_LENGTH = 8

NAME_CHARS = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ01234567890_-+'

NAME_LENGTH = [6, 30]

PASSWORD_RESET_DELAY = 30

active()

static all(*session*)

created

email

static forgot_password(*db*, *name_or_mail*)

static generate_hash(*length=10*)

static get(*session*, *user*)

static get_by_email(*session*, *email*)

static get_by_hash(*session*, *hash*)

static get_by_id(*session*, *uid*)

static get_by_name(*session*, *name*)

get_groups()
Get groups of a user. Only active groups are returned.

Returns list of Group_User_Assoc objects

get_managed_groups()
Get managed groups of the user. Only not deleted groups are returned.

Returns list of Group objects

get_managed_projects()
Get managed projects of the user.

Returns list of Project objects

get_managed_workgroups()
Get managed workgroups of the user.

Returns list of Workgroup objects

static get_or_create_by_name(*session*, *name*, ***kwargs*)

get_permissions(*project*)
Get permissions of the user in the given project.

Parameters **project** (Project) – concerning project

Returns list of Permission objects

Raises TypeError if project is not instance of Project

Raises Exception if user not in project

get_projects()
Get active projects of the user.

Returns Set of Project_User_Assoc objects, Set of Project_Group_Assoc objects

get_roles(*project*)
Get roles of the user in the given project.

Parameters **project** (Project) – concerning project

Returns list of Role objects

Raises TypeError if project is not instance of Project

Raises Exception if user not in project

get_workgroups()
Get workgroups of the user.

Returns list of Workgroup objects

static guest_login(*session*)

has_permission(*permissions*, *project*)
Check if the user has the given permissions in the given project

Parameters

- **permission** (*Permission or list of Permission*) – permission or list of permissions to be checked
- **project** (Project) – concerning project

Returns bool is all permissions are present

Raises TypeError if project is not instance of Project

Raises Exception if user not in project

hash

id

static is_active(*session*, *uid*)

is_in_workgroup(*workgroup*)
Check if the user is in the workgroup, either user or manager.

Parameters **workgroup** (Workgroup) – concerning workgroup

Returns bool whether user is manager or user of workgroup

Raises TypeError if workgrop is not instance of Workgroup

last_password_reset

last_request

static login(*session*, *username*, *password*)

name

password

static register(*session*, *name*, *email*)

static send_registration_mail(*name*, *email*, *hash*)

serveradmin

static set_password(*db*, *hash*, *password*)

status

static update_last_request(*session*, *uid*)

vispa.models.workgroup module

class vispa.models.workgroup.**Workgroup**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

A workgroup is a lighter version of a project. It only contains users, no groups, and there is no explicit management of roles and permission, which means, all users have the same rights. The memberships are organized by managers.

add_manager(*user*)
Add manager to workgroup.

Parameters **user** (User) – user which has to be added as manager

Raises TypeError if user is not instance of User

Raises Exception if user is already manager of workgroup

add_user(*user*)
Add user to workgroup.

Parameters **user** (User) – user which has to be added

Raises TypeError if user is not instance of User

Raises Exception if user is already in workgroup

static all(*session*)
Get all existing workgroups.

Parameters **session** – current session of database

Returns list of Workgroup objects

static create(*session*, *name*)
Create new workgroup.

Parameters

- **session** – current session of database
- **name** – name of the workgroup

Raises Exception if parameter name is invalid or already in use

created

delete(*session*)
Delete workgroup. It is also deleted from the database.

Parameters **session** – current session of database

static get(*session*, *workgroup*)
Returns workgroup, whichs name or id is given. If the parameter workgroup is already an instance of Workroup it is directly returned.

Parameters

- **session** – current session of database
- **workgroup** – name or id to look for. If workgroup is instance of Workroup, it is directly returned

Returns Workgroup

Raises Exception if parameter workgroup is invalid of no workgroup can be found

static get_by_id(*session*, *gid*)
Get a workgroup by its id.

Parameters **session** – current session of database

Para gid id, which is looked for

Returns Workgroup or None

static get_by_name(*session*, *name*)
Get a workgroup by its name.

Parameters **session** – current session of database

Para name name, which is looked for

Returns Workgroup or None

get_items(*itemtype=None*)
Get WorkgroupItem.

Parameters **itemtype** – optional selector on the item type

Returns list of WorkgroupItem objects

get_managers()
Get managers of workgroup.

Returns list of User objects

get_users()
Get users of the workgroup.

Returns list of User objects

id

items

managers

name

remove_manager(*manager*)
Remove manager from workgroup.

Parameters **manager** (User) – manager which has to be removed

Raises TypeError if manager is not instance of User

Raises Exception if manager is not manager of workgroup

remove_user(*user*)
Remove user from workgroup.

Parameters **user** (User) – user which has to be removed

Raises TypeError if user is not instance of User

Raises Exception if user is not in workgroup

rename(*session*, *newname*)
Rename workgroup.

Parameters

- **session** – current session of database
- **newname** – new name of the workgroup

Raises Exception if newname is already in use

users

class vispa.models.workgroup.**WorkgroupItem**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

A workgroup item represents the actual content of a workgroup. It is simply an object consisting of a type (e.g. file) and its content (e.g. the path of the file)

content

static create(*session*, *workgroup*, *itemtype*, *content*)
Create an workgroup item. The item is added to the database.

Parameters

- **session** – current session of database
- **workgroup** – workgroup of the new item
- **itemtype** – type of the item
- **content** – content of the item

delete(*session*)
Delete the item. It is also deleted from the database.

Parameters **session** – current session of database

static get(*session*, *item*)
Returns item, whichs id given. If the parameter item is an instance of WorkgroupItem object, it is directly returned.

Parameters

- **session** – current session of database
- **item** – id to look for. If item is an instance of WorkgroupItem, item is directly returned

Returns WorkgroupItem

Raises Exception if parameter item is invalid or no WorkgroupItem can be found

static get_by_id(*session*, *gid*)
Get a workgroup item by its id.

Parameters

- **session** – current session of database

- **gid** – given id, which is looked for

Returns WorkgroupItem or None

get_workgroup()
Get the workgroup of the item.

Returns Workgroup

id

itemtype

set_content(*content*)
Set the content of the item.

Parameters **content** – new content

workgroup_id

vispa.models.workspace module

class vispa.models.workspace.**Workspace**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

KEYS = ['id', 'user_id', 'name', 'host', 'login', 'command', 'created', 'auto_connect', 'key']

static add(*db*, *user*, *name*, *host*, *login*, *key=None*, *command=None*, *add=True*)

auto_connect

can_edit(*user*)

command

created

static get_by_id(*db*, *id*)

static get_user_workspace_count(*db*, *user*)

static get_user_workspaces(*db*, *user*)

has_access(*user*)

host

id

is_valid()

key

login

login_credentials

make_dict(*keys=None*)

name

static remove(*db*, *id*)

static update(*db*, *id*, ***kwargs*)

user_id

Module contents

vispa.plugins package

Submodules

vispa.plugins.template module

class vispa.plugins.template.**MakoPlugin**(*bus*, *base_dir=None*, *module_dir=None*, *collection_size=50*, *encoding='utf-8'*)

Bases: cherrypy.process.plugins.SimplePlugin

lookup_template(*name*)

start()

stop()

Module contents

vispa.remote package

Subpackages

vispa.remote.fsmonitor package

Submodules

vispa.remote.fsmonitor.common module

class vispa.remote.fsmonitor.common.**FSEvent**(*watch*, *action*, *name=''*)

Bases: object

Access = 1

All = 511

Attrib = 4

Create = 8

Delete = 16

DeleteSelf = 32

Modify = 2

MoveFrom = 64

MoveSelf = 256

MoveTo = 128

action_name

action_names = {16: 'delete', 1: 'access', 2: 'modify', 4: 'attrib', 32: 'delete self', 8: 'create', 64: 'move from', 128: 'm

path

user

exception vispa.remote.fsmonitor.common.**FSMonitorError**
Bases: exceptions.Exception

exception vispa.remote.fsmonitor.common.**FSMonitorOSError**
Bases: exceptions.OSError, *vispa.remote.fsmonitor.common.FSMonitorError*

vispa.remote.fsmonitor.linux module

class vispa.remote.fsmonitor.linux.**FSMonitor**
Bases: object

add_dir_watch(*path*, *flags=511*, *user=None*)

add_file_watch(*path*, *flags=511*, *user=None*)

close()

disable_watch(*watch*)

enable_watch(*watch*, *enable=True*)

read_events(*timeout=None*)

remove_all_watches()

remove_watch(*watch*)

watches

class vispa.remote.fsmonitor.linux.**FSMonitorWatch**(*wd*, *path*, *flags*, *user*)
Bases: object

vispa.remote.fsmonitor.linux.**convert_flags**(*flags*)

vispa.remote.fsmonitor.linux.**parse_events**(*s*)

vispa.remote.fsmonitor.polling module

class vispa.remote.fsmonitor.polling.**FSMonitor**
Bases: object

add_dir_watch(*path*, *flags=511*, *user=None*)

add_file_watch(*path*, *flags=511*, *user=None*)

disable_watch(*watch*)

enable_watch(*watch*, *enable=True*)

read_events(*timeout=None*)

remove_all_watches()

remove_watch(*watch*)

watches

class vispa.remote.fsmonitor.polling.**FSMonitorDirWatch**(*path*, *flags*, *user*)
Bases: object

delstate()

getstate()

classmethod new_state(*path*)

setstate(*state*)

state

class vispa.remote.fsmonitor.polling.**FSMonitorFileWatch**(*path*, *flags*, *user*)
Bases: object

delstate()

getstate()

classmethod new_state(*path*)

setstate(*state*)

state

class vispa.remote.fsmonitor.polling.**FSMonitorWatch**(*path*, *flags*, *user*)
Bases: object

vispa.remote.fsmonitor.polling.**get_dir_contents**(*path*)

vispa.remote.fsmonitor.polling.**round_fs_resolution**(*t*)

vispa.remote.fsmonitor.win32 module

Module contents

class vispa.remote.fsmonitor.**FSMonitor**
Bases: object

add_dir_watch(*path*, *flags=511*, *user=None*)

add_file_watch(*path*, *flags=511*, *user=None*)

close()

disable_watch(*watch*)

enable_watch(*watch*, *enable=True*)

read_events(*timeout=None*)

remove_all_watches()

remove_watch(*watch*)

watches

class vispa.remote.fsmonitor.**FSMonitorThread**(*callback=None*)
Bases: threading.Thread

add_dir_watch(*path*, *flags=511*, *user=None*)

add_file_watch(*path*, *flags=511*, *user=None*)

read_events()

remove_all_watches()

remove_watch(*watch*)

run()

stop()

exception vispa.remote.fsmonitor.**FSMonitorError**
Bases: exceptions.Exception

exception vispa.remote.fsmonitor.**FSMonitorOSError**
Bases: exceptions.OSError, *vispa.remote.fsmonitor.common.FSMonitorError*

class vispa.remote.fsmonitor.**FSEvent**(*watch*, *action*, *name=''*)
Bases: object

Access = 1

All = 511

Attrib = 4

Create = 8

Delete = 16

DeleteSelf = 32

Modify = 2

MoveFrom = 64

MoveSelf = 256

MoveTo = 128

action_name

action_names = {16: 'delete', 1: 'access', 2: 'modify', 4: 'attrib', 32: 'delete self', 8: 'create', 64: 'move from', 128: 'm

path

user

Submodules

vispa.remote.filesystem module

class vispa.remote.filesystem.**FileSystem**(*userid*, *workspaceid*)
Bases: object

ADDITIONAL_MIMES = {'root': 'text/plain', 'pxlio': 'text/plain'}

BROWSER_EXTENSIONS = ['png', 'jpg', 'jpeg', 'bmp']

FILE_EXTENSIONS = ['png', 'jpg', 'jpeg', 'bmp', 'ps', 'eps', 'pdf', 'txt', 'xml', 'py', 'c', 'cpp', 'root', 'pxlio']

GLOBAL_WORKSPACE_CONF = '/etc/vispa/workspace.ini'

PRIVATE_WORKSPACE_CONF = '~/.vispa/workspace.ini'

checkPermissions(*path*, *permission=2*)

check_file_extension(*path*, *extensions=[]*)

close()

compress(*paths*, *path*, *name*, *is_tmp=False*)

create_file(*path*, *name*)

create_folder(*path*, *name*)

cut_slashs(*path*)

decompress(*path*)

exists(*path*, *type=None*)

expand(*path*)

get_file(*path*, *binary=False*, *utf8=False*, *window_id=None*, *view_id=None*, *watch_id=None*, *max_size=20*)

get_file_content(*path*, *offset=0*, *length=None*)

get_file_count(*path*, *window_id=None*, *view_id=None*, *watch_id=None*)

get_file_list(*path*, *filter=None*, *reverse=False*, *hide_hidden=True*, *encode_json=True*, *window_id=None*, *view_id=None*, *watch_id=None*)

get_mime_type(*filepath*)

get_mtime(*path*)

get_suggestions(*path*, *length=1*, *append_hidden=True*, *encode_json=True*)

get_workspaceini(*request*, *fail_on_missing=False*)

getfacl(*path*)

handle_file_name_collision(*name*, *path*)

is_browser_file(*path*)

move(*source*, *destination*)

paste(*path*, *fullsrc*, *cut*)

remove(*path*)

rename(*path*, *name*, *new_name*, *force=False*)

save_file(*path*, *content*, *force=True*, *binary=False*, *utf8=False*, *window_id=None*, *view_id=None*, *watch_id=None*)

save_file_content(*filename*, *content*, *path=None*, *force=True*, *append=False*)

set_workspaceini(*request*)

setfacl(*path*, *type*, *name*, *mode*, *remove=False*, *recursive=False*, *default=False*)

setup(*basedir=None*)

stat(*path*)

thumbnail(*path*, *width=100*, *height=100*, *sharpen=True*)

unwatch(*window_id*, *view_id*, *watch_id=None*)

watch(*path*, *window_id*, *view_id*, *watch_id*, *pattern=None*, *reverse=False*, *hide_hidden=True*)

class vispa.remote.filesystem.**WatchService**
Bases: object

stop()

subscribe(*id*, *path*, *pattern=None*, *reverse=False*, *hide_hidden=True*)

unsubscribe(*id*)

class vispa.remote.filesystem.**WatchSubscriber**(*service*, *id*)
Bases: object

EVENT_DELAYS = {'modify': [1.0, 0.2], 'change': [1.0, 0.1]}

MAX_INLINE_SUBJECTS = 10

MAX_SUBJECT_NAMES = 25

bind(*path*)

destroy()

emit(*event*)

flush(*force=False*)

process(*event*, *subject=''*)

unbind()

update(*path*, *pattern=''*, *reverse=False*, *hide_hidden=True*)

vispa.remote.filesystem.**file_compare**(*a*, *b*)

vispa.remote.filesystem.**get_file_info**(*base*, *name*)

vispa.remote.filesystem.**string_compare**(*a*, *b*)

vispa.remote.helper module

class vispa.remote.helper.**UTF8Buffer**

Bases: object

Buffers incoming UTF8 encoded data, and prevents chunks from being created in the middle of a multi byte sequence.

fill(*data*)

Fill the buffer with the given *data*.

passThru(*data*, *count=None*)

A convenience function for adding data to the buffer and reading up to *count* of it again. If *count* is *None* the current buffer length is used.

read(*count=None*)

Get a maximum of *count* bytes from the buffer. UTF8 multibyte characters will not be broken apart. If *count* is *None* the current buffer length is used.

Module contents

exception vispa.remote.**AjaxException**(*message*, *code=None*, *alert=True*)

Bases: exceptions.Exception

AjaxException that is handled by the ajax tool and that can be raised in controller methods. *message* is the error message to show. *code* should be an integer that represents a specific type of exception. If *code* is *None* and *message* is an integer representing a http status code, the error message is set to the standard error message for that http error. If *alert* is *True*, the message is shown in a dialog in the GUI.

vispa.remote.**raise_ajax**(*fn=None*, ***kwargs*)

Decorator that transforms raised exceptions into AjaxExceptions.

vispa.remote.**send_topic**(*topic*, *data=None*, *window_id=None*, *user_id=None*)

vispa.tools package

Submodules

vispa.tools.ajax module

Definition of the vispa ajax tool.

class vispa.tools.ajax.**AjaxTool**
Bases: cherrypy._cptools.Tool

Ajax tool that takes the output of a wrapper inner function and returns a json encoded dictionary containing the following entries: - code: A response code, basically an appropriate http status. The status of the

cherrypy response object is also set to this value. Thus, the code is 200 if there were no errors.

- data: The attached payload with an arbitrary type.
- message: In case of an error, i.e. code != 200, an additional error message. On success, i.e. code = 200, this should be empty.
- alert: A boolean that desribes whether the error message should be shown in the GUI using a dialog or something similar.

callable(*encoded=False*)
The callable of this tool. If *encoded* is *True*, the return value of the wrapped inner function is expected to be already json encoded. Otherwise, it will be encoded using `json.dumps()`.

vispa.tools.db module

class vispa.tools.db.**SqlAlchemyTool**(*engine*)
Bases: cherrypy._cptools.Tool

bind_session()
Attaches a session to the request's scope by requesting the SA plugin to bind a session to the SA engine.

commit_transaction()
Commits the current transaction or rolls back if an error occurs. Removes the session handle from the request's scope.

vispa.tools.device module

class vispa.tools.device.**DeviceTool**
Bases: cherrypy._cptools.Tool

devicematch(*agent*, *device='all'*)

get_device_name(*agent*)

groupmatch(*agent*, *group='all'*)

vispa.tools.json_parameters module

class vispa.tools.json_parameters.**JsonParameters**
Bases: cherrypy._cptools.Tool

before_handler()

vispa.tools.method module

Definition of the vispa method tool.

class vispa.tools.method.**MethodTool**
Bases: cherrypy._cptools.Tool

Basically, the method tool implements the same functionality as cherrypy's built-in "allow" tool, but in addition, this tool is compliant to our ajax tool.

callable(*accept=None*, *reject=None*, *ajax=True*)
Actual tool logic. Checks whether a request is sent with a valid HTTP method using *accept* and *reject*. Both of them can be strings or iterables of strings. When the request not accepted and *ajax* is *True*, the ajax tool is used to send an error. Otherwise, a cherrypy.HTTPError is raised.

vispa.tools.parameters module

Cherrypy tool that moves parameters that start with an underscore from the querystring to request.private_params.

class vispa.tools.parameters.**PrivateParameterFilter**
Bases: cherrypy._cptools.Tool

before_handler()

vispa.tools.permission module

class vispa.tools.permission.**PermissionTool**
Bases: cherrypy._cptools.Tool

use from config file:

[/path/to/protected/resource] tools.permission.on = True tools.permission.requiredPermissions = ['myextension.read', 'myextension.write']

use as decorator: @cherrypy.expose @cherrypy.tools.permission(requiredPermissions=['myextension.resources']) def resource(self):

return "Hello, %s!" % cherrypy.request.login

Parameters

- **requiredPermissions** – string or iterable of strings, which the user must all have in the global project. If the user management is not automatically setup, the tool does nothing
- **ignoreInexistentPermissions** – when true, requiredPermissions are ignored if they do not exist

vispa.tools.status module

class vispa.tools.status.**StatusMonitor**
Bases: cherrypy._cptools.Tool

Register the status of each thread.

callable()

unregister()
Unregister the current thread.

class vispa.tools.status.**ThreadStatus**(*threadid*)
Bases: object

end = None

idle_time()

last_req_time()

start = None

url = None

vispa.tools.template module

class vispa.tools.template.**MakoTool**(*common_data=None*)
Bases: cherrypy._cptools.Tool

callable(*template=None*, *common_data=None*, ***kwargs*)

vispa.tools.user module

Definition of the vispa user tool.

class vispa.tools.user.**UserTool**
Bases: cherrypy._cptools.Tool

The user tool checks whether the session contains the field "user_id". If it exists, a reference to the corresponding user is stored as "cherrypy.request.user". Otherwise, the request is either redirected or a 401 error is returned using the ajax tool.

vispa.tools.workspace module

class vispa.tools.workspace.**WorkspaceTool**
Bases: cherrypy._cptools.Tool

before_handler(***conf*)

Module contents

9.1.2 Submodules

9.1.3 vispa.browser module

vispa.browser.**append_to_session**(*key*, *value*)
Adds a value to a list in the session

vispa.browser.**client_agent**()

vispa.browser.**client_ip**()

vispa.browser.**client_referer**()

vispa.browser.**delete_cookie**(*key*)

vispa.browser.**delete_session**()

vispa.browser.**get_cookie**(*key*)

vispa.browser.**get_session_value**(*key*)

vispa.browser.**has_session_value**(*key*)

vispa.browser.**set_cookie**(*key*, *value=''*, *age=None*, *path='/'*, *version=None*)

vispa.browser.**set_session_value**(*key*, *value=None*)

vispa.browser.**update_to_session**(*key*, *data*)

9.1.4 vispa.rest module

Dispatcher and controller for implementing a REST api in cherrypy.

class vispa.rest.**RESTDispatcher**(*dispatch_method_name=None*, *translate='x00x01x02x03x04x05x06x07x08tnx0bx0crx0ex0fx10x11x12x13x14x15x16x17x18x19 _____0123456789_____ABCDEFGHIJKLMNOPQRSTUVWXYZ_____abcdefghijklmnopqrstuvwxyz____x7fx80x81x82x83x84x*)

Bases: cherrypy._cpdispatch.Dispatcher

Dispatches request to handler functions that are decorated with resolvers. Use it in conjunction with the RESTController class. Example:

```
class Controller(RESTController):

    @POST("user/:id(^\d+$)/name/:name")
    def set_user_name_by_id(self, id, new_name):
        pass

    @POST("user/:name/name/:name")
    def set_user_name_by_name(self, name, new_name):
        pass
```

Calls to POST /user/4/name/tom will be routed to the first handler, calls to POST /user/tom/name/tim to the second one. Thus, the order of handler definitions is important for the dispatcher to take precedence rules into account.

find_handler(*path*)

Implements cherrypy.dispatch.Dispatcher.find_handler(path).

find_node(*path*)

Trails the handler tree to find the correct RESTController instance.

class vispa.rest.**RESTController**

Bases: object

RESTController that should be used in conjunction the a RESTDispatcher. It's only task is to prepare handlers at the end of its initialization.

vispa.rest.**GET**(*fmt*)

vispa.rest.**POST**(*fmt*)

vispa.rest.**DELETE**(*fmt*)

9.1.5 vispa.server module

class vispa.server.**AbstractExtension**(*server*)
Bases: object

Base class for Extensions

add_controller(*controller*)
Mount a CherryPy controller using the extension name for path.

Parameters **controller** – filename relative to extension directory

add_workspace_directoy(*directory='workspace'*)
Add files to be transferred to the worker.

Parameters **directoy** – directory relative to extension directory

clear_workspace_instance(*name*, *key=None*, *user=None*, *workspace=None*, *db=None*)

config()

create_topic(*topic=''*, *view_id=None*)

dependencies()
Return a list of Extension names this Extension depends on.

get_workspace_instance(*name*, *key=None*, *user=None*, *workspace=None*, *db=None*, ***kwargs*)

name()
Return the name of the Extension. This name is used as part of the URL.

setup()
Setup the extension.

class vispa.server.**Server**(***kwargs*)
Bases: object

application()

extension(*name*)

extensions()

run()

start()

9.1.6 vispa.socketbus module

class vispa.socketbus.**Bus**

received_message(*msg*, *window_id*)

send(*window_id=None*, *user_id=None*, *data=None*, *except_sessions=None*, *binary=False*, *encode_json=True*, *broadcast=False*)

send_topic(*topic*, **args*, ***kwargs*)

subscribe(*topic*, *handler*)

unsubscribe(*topic*, *handler=None*)

class vispa.socketbus.**CleanerThread**(**kargs*, ***kwargs*)
Bases: threading.Thread

run()

class vispa.socketbus.**PollingPublisher**(*window_id*, *user_id*)

fetch(*timeout=None*)

received_message(*msg*)

send(*data*, *binary=False*, *timeout=None*)

class vispa.socketbus.**SocketPublisher**

vispa.socketbus.**add_session**(*window_id*, *user_id*, *publisher*)

vispa.socketbus.**get_polling_publisher**(*window_id*, *user_id*)

vispa.socketbus.**remove_session**(*window_id*, *delay=False*)

9.1.7 vispa.url module

Functions to translate a relative path into a valid url

vispa.url.**clean**(*url*)
Clean the url, multiple fix slashes and strip whitespaces

vispa.url.**dynamic**(**parts*, ***kwargs*)
Create an absolute URL to non static content, e.g. controllers

vispa.url.**join**(**args*)
Join all parameters into one clean url path.

vispa.url.**static**(**parts*, ***kwargs*)
Create an absolute URL to static content, e.g. images

9.1.8 vispa.version module

9.1.9 vispa.workspace module

Functions to connect and control connections to workers

class vispa.workspace.**Connection**(*userid*, *workspaceid*, *host*, ***kwargs*)
Bases: object

CONNECTED = 2

CONNECTING = 1

DISCONNECTED = 0

DISCONNECTING = 3

STATUS_MESSAGES = {0: 'disconnected', 1: 'connecting', 2: 'connected', 3: 'disconnecting'}

active()

close()

connected(*timeout=10*)

errors()

host()

open(***kwargs*)

poll()

rpyc()

send_status()

static **send_workspace_status**(*userid*, *workspaceid*, *status*)

status(*new_status=None*)

stdin()

stdout()

tempdir()

class vispa.workspace.**ConnectionPool**
Bases: object

SERVE_CONNECTION_INTERVAL = 0.2

clear(*user=None*, *workspace=None*)

connect(*user*, *workspace*, *password=None*)

connections_of_user(*user*)
Returns a list of tuples: (workspaceId, connection)

get(*user*, *workspace*, ***kwargs*)

get_workspace_connections(*user*, *workspaces*)
return a list of tuples with the workspace and the assosiated connection or None

class vispa.workspace.**InstancePool**
Bases: object

clear(*user=None*, *workspace=None*, *classname=None*, *key=None*)

get(*_user*, *_workspace*, *classname=None*, *key=None*, *init_args=None*, ***kwargs*)

class vispa.workspace.**LocalConnectionFeeder**
Bases: threading.Thread

callbacks = {}

run()

class vispa.workspace.**LocalConnectionImpl**(*command*, ***kwargs*)
Bases: object

close()

print_stderr(*timeout=0*)

set_on_feed(*cb*)

stream()

writeln(*data*)

class vispa.workspace.**LoggingService**(*conn*)
Bases: rpyc.core.service.Service

exposed_log(*name*, *level*, *msg*)

exposed_send(**args*, ***kwargs*)

class vispa.workspace.**SSHConnectionImpl**(*command*, *address*, *missing_host_key_policy='warning'*, ***kwargs*)

Bases: object

An SSH transport for Pushy, which uses Paramiko.

TIMEOUT = 30

address2host_port(*host*)

close()

print_stderr()

set_on_feed(*cb*)

stream()

writeln(*data*)

class vispa.workspace.**WrappedChannelFile**(*file_*, *how*)

Bases: object

close()

fileno()

vispa.workspace.**add_directory_files**(*local*, *remote*, ***kwargs*)

vispa.workspace.**add_package_files**(*pkg*, *target=None*)

vispa.workspace.**add_remote_file**(*localfile*, *remotefile*)

vispa.workspace.**add_remote_files**(*files*)

vispa.workspace.**clear_instance**(*classname*, *key=None*, *user=None*, *workspace=None*, *db=None*)

Remove all instances of type *classname* from the pool

vispa.workspace.**connect**(*workspace*, *user=None*, *db=None*, *password=None*)

Connect the selected workspace.

vispa.workspace.**directory_files**(*local*, *remote*, ***kwargs*)

vispa.workspace.**disconnect**(*workspace=None*, *user=None*, *db=None*)

Disconnect the selected workspace and remove all pooled instances.

vispa.workspace.**disconnect_all**(*user=None*, *db=None*)

Disconnect all workspaces and pooled instances of the selected user.

vispa.workspace.**get_instance**(*classname*, *key=None*, *user=None*, *workspace=None*, *db=None*, *init_args=None*)

Returns a pooled reference to a class instance of type *classname*.

vispa.workspace.**module**(*modulename*, *user=None*, *workspace=None*, *db=None*)

Returns a reference to a remote module

vispa.workspace.**package_files**(*pkg*, *target_path*, ***kwargs*)

9.1.10 vispa.wsgi module

vispa.wsgi.**application**(*environ*, *start_response*)

9.1.11 Module contents

Basic functionality for the VISPA platform

exception vispa.**AjaxException**(*message*, *code=None*, *alert=True*)
Bases: exceptions.Exception

AjaxException that is handled by the ajax tool and that can be raised in controller methods. *message* is the error message to show. *code* should be an integer that represents a specific type of exception. If *code* is *None* and *message* is an integer representing a http status code, the error message is set to the standard error message for that http error. If *alert* is *True*, the message is shown in a dialog in the GUI.

class vispa.**Netstat**
Bases: object

Parses /dev/net/tcp to determine which user owns the specified port

static get_socket_owner(*local_ip*, *local_port*, *remote_ip*, *remote_port*)
Returns the system id of the local user which established the specifiec connection

Parameters

- **local_ip** – ip address of the local peer
- **local_ip** – port the local peer
- **remote_ip** – ip address of the remote peer
- **remote_ip** – port the remote peer

Return type user id of the user who opened this port or None

class vispa.**VispaConfigParser**(*defaults=None*, *dict_type=<class 'collections.OrderedDict'>*, *allow_no_value=False*)
Bases: ConfigParser.SafeConfigParser

vispa.**codepath**(**args*)

vispa.**configpath**(**args*)

vispa.**datapath**(**args*)
returns the path relative to the datapath

vispa.**dump_thread_status**(*f=None*)

vispa.**dump_thread_status_on_signal**(*signal*, *stack*)

vispa.**exception_string**()

vispa.**fire_callback**(*topic*, **args*, ***kwargs*)

vispa.**log_exception**()

vispa.**publish**(*topic*, **args*, ***kwargs*)

vispa.**register_callback**(*topic*, *callback*)

vispa.**send_mail**(*addr*, *subject=''*, *content=''*, *sender_addr=None*, *smtp_host=None*, *smtp_port=None*)
Send an email.

All arguments should be Unicode strings (plain ASCII works as well).

Only the real name part of sender and recipient addresses may contain non-ASCII characters.

The email will be properly MIME encoded and delivered though SMTP to localhost port 25. This is easy to change if you want something different.

The charset of the email will be the first one out of US-ASCII, ISO-8859-1 and UTF-8 that can represent all the characters occurring in the email.

vispa.**set_codepath**(*p*)

vispa.**set_configpath**(*p*)

vispa.**set_datapath**(*p*)

vispa.**setup_thread_dump**()

vispa.**subscribe**(*topic*, *callback*)

vispa.**thread_stacktraces**()

V

A

B

C

D

E

F

G

H

I

J

K

L

M

N

R

S

T

U

V

W