VeilFS Documentation

Release 1.0

VeilFS

January 26, 2014

Contents

Part I

VeilClient

Documents

CHAPTER 1

About

VeilClient is a part of a meta file system, called VeilFS, which unifies access to different storage systems and provides a POSIX compatible interface.

1.1 Goals

The main goal of VeilClient is to provision a file system to different, heterogeneous storage systems, which will work in the user space, e.g. Lustre, GPFS, DPM, iRODS. VeilClient intends to reduce the complexity of accessing various storage systems by providing a standard, POSIX compatible interface. Furthermore, storage systems connected to VeilClient can be geographically distributed, and operated by different organizations. The end user may operate on data from the storage systems as if they were stored at a local file system.

1.2 Getting Started

VeilClient is built with CMake. More informations about compiling the project in "Compilation" section. Sources are put in 'src'. The 'src' includes subdirectories: 'fuse' and 'helpers'. The 'fuse' directory contains sources of the user space file system while the 'helpers' directory includes storage helpers used by FUSE.

1.3 Prerequisites

In order to compile the project, you need to have fallowing additional libraries, its headers and all its prerequisites in include/ld path: Also you need cmake 2.8+.

- fuse
- protobuf
- ssl
- crypto
- boost (>= 1.49)
- ltdl
- globus_common

- globus_oldgaa
- globus_openssl
- globus_gsi_callback
- globus_gsi_cert_utils
- globus_gsi_sysconfig
- globus_proxy_ssl
- globus_gsi_proxy_core
- globus_gsi_credential
- globus_openssl_error

Use this command to install the required dependency packages:

- Debian/Ubuntu Dependencies (.deb packages):

```
apt-get install libprotobuf-dev libfuse-dev fuse libboost-dev libglobus-*-dev libtool
```

- RHEL/CentOS/Fedora Dependencies (.rpm packages):

```
yum install fuse fuse-libs fuse-devel protobuf-devel openssl-devel cmake28 boost-devel boost
```

1.4 Compilation

1.4.1 "The fast way"

If you just need an RPM package, you can just type:

```
make -s rpm
```

If there was no errors, you will get list of generated packages (rpm or dep).

1.4.2 "The standard way"

VeilClient uses cmake as a build tool thus building process is same as for most cmake-based projects. However you can use Makefile-proxy with following interface: (Note that -s flag is optional - it's a silent mode which makes output much prettier, because it leaves only cmakes' stdout/stderr)

Configure

```
make -s configure
```

Build

```
make -s build
```

Install

```
make -s install
```

RPM/DEB packages

```
make -s rpm
```

Testing

There are two testing targets:

```
make -s test
```

which has summarized output (per test case) and:

```
make -s cunit
```

which shows detailed test results.

1.5 Using VeilClient

1.5.1 Configuration

First of all you should tune up some config settings. Configuration file can be found in {INSTALL_PREFIX}/etc/veilFuse.conf. In most linux distros deafault {INSTALL_PREFIX} is /usr/local. Configuration options are described in configuration file itself. In most cases you want to stick with default values although there are 2 options that requires special care:

- cluster_hostname - hostname of VeilCluster used by client
- peer_certificate_file - path to proxy certificate (.pem file) used in SSL session. Paths are relative to HOME env unless absolute path is specified.

You don't edit this global config file if you don't want to. You can also create new file, type options that shall be overriden and pass '–config=/path/to/your/config/file' option while starting VeilClient. Also its possible to override options by setting env variable with the same name (only uppercase):

```
CLUSTER_HOSTNAME="some.hostname.com" veilFuse /mount/point
```

1.5.2 Mounting the filesystem

Prerequisites

In order to use VeilClient, you need to have fallowing additional libraries in ld path:

- libfuse
- libprotobuf
- libssl

Use this command to install the required dependency packages:

- Debian/Ubuntu Dependencies (.deb packages):

```
apt-get install libprotobuf libfuse fuse
```

- RHEL/CentOS/Fedora Dependencies (.rpm packages):

```
yum install fuse fuse-libs protobuf openssl
```

Starting

In order to mount VeilFS just enter:

```
veilFuse /mount/point
```

Additionally you can add '-d' option which enables debug mode. In debug mode application will remain running, displaing all logs and debug informations, also in this mode ctrl+c unmount filesystem. If not in debug mode, application will go background as daemon.

1.5.3 Unmounting the filesystem

If veilFuse was started with '-d' option, just hit ctrl+c. If not:

```
fusermount -u /mount/point
```

1.6 Support

For more information visit project Confluence or write to 'wrzeszcz@agh.edu.pl ('wrzeszcz@agh.edu.pl)'.

Submodules

CHAPTER 2

Classes

Documents

2.1 veil::client::AutoLock

class **veil::client::AutoLock**

The *AutoLock* class.

Object of this class provides auto-unlocking *ReadWriteLock*.

Public Functions

AutoLock(*ReadWriteLock* & lock, *LockType* type)

Create auto *ReadWriteLock*.

Also locks for read/write depending on *LockType* param.

~AutoLock()

Unlocks wrapped *ReadWriteLock* lock, but only if it wasn't released manually.

void **changeType**(*LockType* type)

Changle lock type.

Releases current lock and locks for diffrent operation type.

void **release**()

Release current lock.

void **lock**()

Locks current lock.

Private Members

ReadWriteLock & **m_lock**

The wrapped lock.

LockType **m_type**

Type of this lock.

See

LockType

bool **m_released**

Flag saying if lock is released right now.

2.2 veil::client::Config

class **veil::client::Config**

The *Config*.

Parses config files and provides safe access to configuration map.

Public Functions

std::string **getFuseID**()

Returns current FuseID.

void **negotiateFuseID**(time_t delay = 0)

Starts FuseID negotiation process.

Parameters

- `delay` - Since this is async actions, you can specify execution delay in seconds.

std::string **getString**(std::string opt)

Returns string value of requested option.

Before using this function you should check is option exists, but it's not required.

See

Config::isSet

int **getInt**(std::string opt)

Returns int value of requested option.

Before using this function you should check is option exists, but it's not required.

See

Config::isSet

double **getDouble**(std::string opt)

Returns double value of requested option.

Before using this function you should check is option exists, but it's not required.

See

Config::isSet

bool **getBool**(std::string opt)

Returns boolean value of requested option.

Before using this function you should check is option exists, but it's not required.

See

Config::isSet

Warning

If given opition wasn't set, you'll get empty object of given type T (T())

bool **isSet**(std::string opt)

Checks if given option is set.

See

Config::getValue

void **setGlobalConfigFile**(std::string path)

Sets path to global config file.

See

Config::parseConfig

void **setUserConfigFile**(std::string path)

Sets path to user config file.

See

Config::parseConfig

void **setEnv**()

Saves current CWD and HOME env viariables. This is required as FUSE changes them after non-debug start. This is also done automatically in *Config::Config*.

bool **parseConfig**()

Parses config from files set by *Config::setGlobalConfigFile* and *Config::setUserConfigFile*.

User config overides global settings. If user config declares all required options, global config file isn't required, otherwise it has exists.

Config()

~Config()

Public Static Functions

std::string **absPathRelToCWD**(std::string path)

Converts relative path, to absolute using CWD env as base prefix.

std::string **absPathRelToHOME**(std::string path)

Converts relative path, to absolute using HOME env as base prefix.

void **putEnv**(std::string name, std::string value)

Saves given env variable.

Protected Functions

template < typename T >
T **get**(std::string opt)

Internal implementation of *Config::getValue*.

See

Config::getValue

template < typename T >
T **getValue**(std::string opt)

Returns type-specialized value of given config option.

bool **runTask**(*TaskID* taskId, std::string arg0, std::string arg1, std::string arg3)

Task runner derived from *ISchedulable*.

See

ISchedulable::runTask

void **setupDefaults**()

Protected Attributes

bool **defaultsLoaded**

ReadWriteLock **m_access**

std::string **m_globalConfigPath**

Path to global config file.

See

Config::setGlobalConfigFile

std::string **m_userConfigPath**

Path to user config file.

See

Config::setUserConfigFile

YAML::Node **m_globalNode**

Global config object.

YAML::Node **m_userNode**

User config object.

YAML::Node **m_envNode**

Temp config object used to manipulate env settings.

YAML::Node **m_defaultsNode**

Default configs.

Protected Static Attributes

std::string **m_requiredOpts**[]

Array containing required options names.

std::string **m_envCWD**

Saved CWD env variable.

std::string **m_envHOME**

Saved HOME env variable.

std::map< std::string, std::string > **m_envAll**

All saved env variables.

2.3 veil::client::FslogicProxy

class **veil::client::FslogicProxy**

The *FslogicProxy* class.

This class provides proxy-methods that runs their correspondent cluster-fslogic methods. Each object of *FslogicProxy* has its own connection pool, so in order to always use already opened connections there should be only one instance of this class unless you really need connection isolation between successive cluster requests.

Public Functions

FslogicProxy()

~FslogicProxy()

bool **getFileAttr**(std::string logicName, protocol::fuse_messages::FileAttr & attr)

Downloads file attributes from cluster.

bool **getFileLocation**(std::string logicName, protocol::fuse_messages::FileLocation & location)

Downloads file location info.

bool **getNewFileLocation**(std::string logicName, mode_t mode, protocol::fuse_messages::FileLocation & location)

Query cluser to create new file in DB and get its real location.

int **renewFileLocation**(std::string logicName)

Try to renew location validity for given file.

bool **getFileChildren**(std::string dirLogicName, uint32_t children_num, uint32_t offset, std::vector< std::string > & childrenNames)

List files in given folder.

std::string **renameFile**(std::string fromLogicName, std::string toLogicName)

Rename/move file to new location.

std::string **createDir**(std::string logicName, mode_t mode)

Create directory.

std::string **deleteFile**(std::string logicName)

Delete given file.

bool **sendFileNotUsed**(std::string logicName)

Inform cluster that file isnt used anymore.

std::string **changeFilePerms**(std::string path, mode_t mode)

Change file permissions.

std::string **createLink**(std::string from, std::string to)

Creates symbolic link "from" to file "to".

std::pair< std::string, std::string > **getLink**(std::string path)

Gets path pointed by link.

std::string **updateTimes**(std::string path, time_t atime = 0, time_t mtime = 0, time_t ctime = 0)

Updates *time meta attributes for specific file.

std::string **changeFileOwner**(std::string path, uid_t uid, std::string uname = "")

Updates file's owner.

std::string **changeFileGroup**(std::string path, gid_t gid, std::string gname = "")

Updates file's group owner.

void **pingCluster**(std::string nth)

bool **runTask**(*TaskID* taskId, std::string arg0, std::string arg1, std::string arg3)

Task runner derived from *ISchedulable*.

See

ISchedulable::runTask

Protected Functions

bool **sendFuseReceiveAnswer**(const google::protobuf::Message & fMsg, google::protobuf::Message & response)

Sends and receives given protobuf message.

High level method used to send serialized protobuf message to cluster and return its response as given by reference object. Both message and response types have to be subtype of FuseMessage.

Return

true only if received response message is initialized (see google::protobuff::Message::IsInitialized())

std::string **sendFuseReceiveAtom**(const google::protobuf::Message & fMsg)

Sends given protobuf message and receives atom.

This method is simalar to *FslogicProxy::sendFuseReceiveAnswer* But receives simple atom cluster response.

See

FslogicProxy::sendFuseReceiveAnswer

Protected Attributes

boost::shared_ptr< *MessageBuilder* > **m_messageBuilder**

MessageBuilder used to construct cluster packets.

2.4 veil::client::ISchedulable

class **veil::client::ISchedulable**

The *ISchedulable* interface.

Deriving from this interface gives possibility to schedule and run async, delayed tasks using *JobScheduler*.

See

JobScheduler::addTask

Public Type

TaskID enum

The TaskID enum.

Values:

- TASK_CLEAR_FILE_ATTR -
- TASK_SEND_FILE_NOT_USED -
- TASK_RENEW_LOCATION_MAPPING -
- TASK_REMOVE_EXPIRED_LOCATON_MAPPING -
- TASK_PING_CLUSTER -
- TASK_ASYNC_GET_FILE_LOCATION -
- TASK_ASYNC_READDIR -
- TASK_ASYNC_GETATTR -
- TASK_ASYNC_UPDATE_TIMES -
- TASK_CLEAR_ATTR -
- TASK_CONNECTION_HANDSHAKE -
- TASK_LAST_ID -

Public Functions

ISchedulable()

~ISchedulable()

Interface destructor.

bool **runTask**(*TaskID* taskId, std::string arg0, std::string arg1, std::string arg3)

Callback which are called by *JobScheduler* when requested.

See

JobScheduler

2.5 veil::client::JobScheduler

class **veil::client::JobScheduler**

The *JobScheduler* class Objects of this class are living daemons (threads) with their own run queue from which they are precessing tasks.

Public Functions

JobScheduler()

~JobScheduler()

bool **hasTask**(*ISchedulable::TaskID* task)

void **addTask**(*Job* job)

Insert (register) new task to run queue.

Inserted task shall run when current time passes its *Job::when*.

See

::Job

void **deleteJobs**(*ISchedulable* * subject, *ISchedulable::TaskID* task)

Deletes all jobs registred by given object.

Used mainly when *ISchedulable* object is destructed.

Protected Functions

void **schedulerMain**()

Thread main loop.

Checks run queue and runs tasks when needed.

void **runJob**(*Job* job)

Starts given task.

See

JobScheduler::schedulerMain

void **startDaemon**()

Starts/restarts daemon.

Protected Attributes

std::priority_queue< *Job* > **m_jobQueue**

Run queue.

pthread_t **m_daemon**

Thread ID.

pthread_mutex_t **m_mutex**

Mutex used to synchronize access to *JobScheduler::m_jobQueue*.

pthread_cond_t **m_queueCond**

Condition used to synchronize access to *JobScheduler::m_jobQueue*.

Protected Static Functions

void * **schedulerMainWrapper**(void * arg)

C wrapper used to start *JobScheduler::schedulerMain*.

2.6 veil::client::MessageBuilder

class **veil::client::MessageBuilder**

The *MessageBuilder* class.

This class can be used to build protobuf messages used to communicate with cluster. Theres encode and decode method for each base message type used by VeilClient. Arguments matches proto specification of their messages.

Public Functions

MessageBuilder()

~MessageBuilder()

protocol::fuse_messages::FuseMessage **createFuseMessage**(std::string id, std::string messageType, std::string messageInput)

protocol::communication_protocol::ClusterMsg **createClusterMessage**(std::string moduleName, std::string messageType, std::string answerType, std::string answerDecoderName, bool synch, std::string input)

protocol::communication_protocol::ClusterMsg **createClusterMessage**(std::string moduleName, std::string messageType, std::string answerType, std::string answerDecoderName, bool synch)

protocol::communication_protocol::ClusterMsg **packFuseMessage**(std::string messageType, std::string answerType, std::string answerDecoderName, std::string messageInput)

protocol::fuse_messages::FuseMessage **decodeFuseAnswer**(protocol::communication_protocol::Answer & answer)

std::string **decodeAtomAnswer**(protocol::communication_protocol::Answer & answer)

2.7 veil::client::MetaCache

class **veil::client::MetaCache**

Class responsible for caching file attributes.

See

VeilFS::getattr

Public Functions

MetaCache()

~MetaCache()

void **addAttr**(std::string path, struct stat & attr)

Cache given attributes Expiration time can be set using configuration file.

void **clearAttrs**()

Clear whole cache.

bool **getAttr**(std::string path, struct stat * attr)

Gets file attributes from cache.

Return

Bool saying if operation succeed and stat struct was filled with data

Parameters

- `stat` - Pointer to stat structure that should be filled with data from cache

void **clearAttr**(std::string path)

Remove cache for given file.

bool **updateTimes**(std::string path, time_t atime = 0, time_t mtime = 0, time_t ctime = 0)

Update *time meta attributes for specific file in cache. Returns true if cache was updated or false if given file wasn't found in cache.

bool **updateSize**(std::string path, size_t size)

Update size meta attribute for specific file in cache. Returns true if cache was updated or false if given file wasn't found in cache.

bool **runTask**(*TaskID* taskId, std::string arg0, std::string arg1, std::string arg3)

Task runner derived from *ISchedulable*.

See

ISchedulable::runTask

Protected Attributes

std::map< std::string, std::pair< time_t, struct stat > > **m_statMap**

This is the cache map.

Value of this std::map is std::pair containing expiration time of attributes and stat struct itself

ReadWriteLock **m_statMapLock**

Lock used to synchronize access to *MetaCache::m_statMap*.

2.8 veil::client::PushListener

class **veil::client::PushListener**

Public Functions

PushListener()

~PushListener()

void **onMessage**(const protocol::communication_protocol::Answer msg)

Input callback. This method should be registered in connection object. This is the source of all processed messages.

int **subscribe**(*listener_fun* fun)

Register callback function.

Each registered by this method function will be called for every incoming PUSH message. Registered callback has to return bool value which tells if subscription shall remain active (false - callback will be removed).

Return

ID of subscription that can be used to unsubscribe manually.

void **unsubscribe**(int subId)

Remove previously added callback.

Parameters

- `subId` - shall match the ID returned by *PushListener::subscribe*

void **onChannelError**(const protocol::communication_protocol::Answer & msg)

Callback called fo each non-ok Answer from cluster.

Protected Functions

void **mainLoop**()

Worker thread's loop.

Protected Attributes

int **m_currentSubId**

bool **m_isRunning**

boost::thread **m_worker**

boost::condition **m_queueCond**

boost::mutex **m_queueMutex**

boost::unordered_map< int, *listener_fun* > **m_listeners**

Listeners callbacks.

std::list< protocol::communication_protocol::Answer > **m_msgQueue**

Message inbox.

2.9 veil::client::ReadWriteLock

class **veil::client::ReadWriteLock**

The *ReadWriteLock* class.

Solves the "library problem" (fast thread-safe read access) *ReadWriteLock* should not be used directly. Use *AutoLock* wrapper instead, for automatic lock release (using RAII)

Public Functions

ReadWriteLock()

~ReadWriteLock()

void **readLock**()

Locks for read operation.

void **readUnlock**()

Unlocks read lock.

void **writeLock**()

Locks for write operation.

void **writeUnlock**()

< Unlocks write lock

Private Functions

DECLARE_MUTEX_FOR(m_class)

DECLARE_MUTEX_FOR(m_res)

Private Members

pthread_cond_t **m_classCond**

int **m_readers**

How many readers are in critical section right now.

int **m_writers**

How many writers are waitning in queue to access critical section.

int **m_fairness**

How many readers was in critical section since last writer left.

2.10 veil::client::StorageMapper

class **veil::client::StorageMapper**

Public Functions

StorageMapper(boost::shared_ptr< *FslogicProxy* > fslogicProxy)

~StorageMapper()

std::pair< *locationInfo* , *storageInfo* > **getLocationInfo**(std::string logical_name, bool useCluster = false)

Gets file location information along with storage info for storage heleper's calls.

Return

std::pair of *locationInfo* and *storageInfo* structs for this file

Parameters

- `logical_name` - File path (relative to *VeilFS* mount point)
- `useCluster` - Specify if the method should use cache only (deafault) or try quering cluster.

std::string **findLocation**(std::string logicalName)

Query cluster about file location and instert it to cache.

See

StorageMapper::addLocation

void **addLocation**(std::string logicalName, protocol::fuse_messages::FileLocation location)

Cache given file location.

Insert to file location cache new FileLocation received from cluster.

void **openFile**(std::string logicalName)

Increases open file count for specified file.

See

locationInfo::opened

void **releaseFile**(std::string logicalName)

Decreases open file count for specified file.

See

locationInfo::opened

bool **runTask**(*TaskID* taskId, std::string arg0, std::string arg1, std::string arg3)

Task runner derived from *ISchedulable*.

See

ISchedulable::runTask

Protected Attributes

std::map< int, *storageInfo* > **m_storageMapping**

Contains storage info accessd by its ID.

See

storageInfo

ReadWriteLock **m_storageMappingLock**

Lock used while operating on *StorageMapper::m_storageMapping*.

See

StorageMapper::m_storageMapping

std::map< std::string, *locationInfo* > **m_fileMapping**

Contains storage info accessd by its ID.

See

storageInfo

ReadWriteLock **m_fileMappingLock**

Lock used while operationg on *StorageMapper::m_fileMapping*.

See

StorageMapper::m_fileMapping

boost::shared_ptr< *FslogicProxy* > **m_fslogic**

Reference to *FslogicProxy* instance.

See

VeilFS::m_fslogic

2.11 veil::client::VeilException

class **veil::client::VeilException**

Base class of all VeilClient exceptions.

Public Functions

VeilException()

Default, empty constuctor.

VeilException(std::string veilError, std::string logMsg = "")

VeilException main constructor.

See

VeilFS::translateError

Parameters

- `veilError` - POSIX error name. Should be compatibile with VeilFS::translateError
- `logMsg` - Human readable excpetion reason message.

~VeilException()

const char * **what**()

Returns *VeilException::m_logMessage*.

std::string **veilError**()

Returns *VeilException::m_veilError*.

Private Members

std::string **m_logMessage**

Human readable excpetion reason message.

std::string **m_veilError**

POSIX error name.

This error string should be compatibile with VeilFS::translateError.

See

VeilFS::translateError

2.12 veil::client::VeilFS

class **veil::client::VeilFS**

The *VeilFS* main class.

This class contains FUSE all callbacks, so it basically is an heart of the filesystem. Technically *VeilFS* is an singleton created on programm start and registred in FUSE daemon.

Public Functions

VeilFS(std::string path, boost::shared_ptr< *Config* > cnf, boost::shared_ptr< *JobScheduler* > scheduler, boost::shared_ptr< *FslogicProxy* > fslogic, boost::shared_ptr< *MetaCache* > metaCache, boost::shared_ptr< *StorageMapper* > mapper, boost::shared_ptr< helpers::StorageHelperFactory > sh_factory)

VeilFS constructor.

~VeilFS()

int **access**(const char * path, int mask)

access FUSE callback.

Not implemented yet.

int **getattr**(const char * path, struct stat * statbuf, bool fuse_ctx = true)

getattr FUSE callback.

See

int **readlink**(const char * path, char * link, size_t size)

readlink FUSE callback.

Not implemented yet.

See

int **mknod**(const char * path, mode_t mode, dev_t dev)

mknod FUSE callback.

See

int **mkdir**(const char * path, mode_t mode)

mkdir FUSE callback.

See

int **unlink**(const char * path)

unlink FUSE callback.

See

int **rmdir**(const char * path)

rmdir FUSE callback.

See

int **symlink**(const char * path, const char * link)

symlink FUSE callback.

Not implemented yet.

See

int **rename**(const char * path, const char * newpath)

rename FUSE callback.

See

int **link**(const char * path, const char * newpath)

link FUSE callback.

Not implemented yet.

See

int **chmod**(const char * path, mode_t mode)

chmod FUSE callback.

See

int **chown**(const char * path, uid_t uid, gid_t gid)

chown FUSE callback.

Not implemented yet.

See

int **truncate**(const char * path, off_t newSize)

truncate FUSE callback.

See

int **utime**(const char * path, struct utimbuf * ubuf)

utime FUSE callback.

Not implemented yet.

See

int **open**(const char * path, struct fuse_file_info * fileInfo)

open FUSE callback.

See

int **read**(const char * path, char * buf, size_t size, off_t offset, struct fuse_file_info * fileInfo)

read FUSE callback.

See

int **write**(const char * path, const char * buf, size_t size, off_t offset, struct fuse_file_info * fileInfo)

write FUSE callback.

See

int **statfs**(const char * path, struct statvfs * statInfo)

statfs FUSE callback.

Not implemented yet.

See

int **flush**(const char * path, struct fuse_file_info * fileInfo)

flush FUSE callback.

Not implemented yet.

See

int **release**(const char * path, struct fuse_file_info * fileInfo)

release FUSE callback.

See

int **fsync**(const char * path, int datasync, struct fuse_file_info * fi)

fsync FUSE callback.

Not implemented yet.

See

int **setxattr**(const char * path, const char * name, const char * value, size_t size, int flags)

setxattr FUSE callback.

Not implemented yet.

See

int **getxattr**(const char * path, const char * name, char * value, size_t size)

getxattr FUSE callback.

Not implemented yet.

See

int **listxattr**(const char * path, char * list, size_t size)

listxattr FUSE callback.

Not implemented yet.

See

int **removexattr**(const char * path, const char * name)

removexattr FUSE callback.

Not implemented yet.

See

int **opendir**(const char * path, struct fuse_file_info * fileInfo)

opendir FUSE callback.

Not implemented yet.

See

int **readdir**(const char * path, void * buf, fuse_fill_dir_t filler, off_t offset, struct fuse_file_info * fileInfo)

readdir FUSE callback.

See

int **releasedir**(const char * path, struct fuse_file_info * fileInfo)

releasedir FUSE callback.

Not implemented yet.

See

int **fsyncdir**(const char * path, int datasync, struct fuse_file_info * fileInfo)

fsyncdir FUSE callback.

Not implemented yet.

See

int **init**(struct fuse_conn_info * conn)

init FUSE callback.

See

bool **runTask**(*TaskID* taskId, std::string arg0, std::string arg1, std::string arg3)

Task runner derived from *ISchedulable*.

See

ISchedulable::runTask

Public Static Functions

boost::shared_ptr< *JobScheduler* > **getScheduler**(*TaskID* taskId = TASK_LAST_ID)

Returns *JobScheduler* assigned to this object.

boost::shared_ptr< *Config* > **getConfig**()

Returns *Config* assigned to this object.

boost::shared_ptr< SimpleConnectionPool > **getConnectionPool**()

boost::shared_ptr< *PushListener* > **getPushListener**()

void **addScheduler**(boost::shared_ptr< *JobScheduler* > injected)

Sets *JobScheduler* object.

void **setConfig**(boost::shared_ptr< *Config* > injected)

Sets *Config* object.

void **setConnectionPool**(boost::shared_ptr< SimpleConnectionPool > injected)

void **staticDestroy**()

Private Members

std::string **m_root**

Filesystem root directory.

uid_t **m_uid**

Filesystem owner's effective uid.

gid_t **m_gid**

Filesystem owner's effective gid.

uid_t **m_ruid**

Filesystem root real uid.

gid_t **m_rgid**

Filesystem root real gid.

boost::shared_ptr< *FslogicProxy* > **m_fslogic**

FslogicProxy instance.

boost::shared_ptr< *StorageMapper* > **m_storageMapper**

StorageMapper instance.

boost::shared_ptr< *MetaCache* > **m_metaCache**

MetaCache instance.

boost::shared_ptr< helpers::StorageHelperFactory > **m_shFactory**

Storage Helpers Factory instance.

std::map< std::string, std::pair< std::string, time_t > > **m_linkCache**

Simple links cache.

ReadWriteLock **m_linkCacheLock**

Private Static Attributes

ReadWriteLock **m_schedulerPoolLock**

std::list< boost::shared_ptr< *JobScheduler* > > **m_jobSchedulers**

JobScheduler instances.

boost::shared_ptr< *Config* > **m_config**

Config instance.

boost::shared_ptr< SimpleConnectionPool > **m_connectionPool**

boost::shared_ptr< *PushListener* > **m_pushListener**

CHAPTER 3

Files

Source code

3.1 gsiHandler.cc

Author

Rafal Slota

Copyright

(C) 2013 ACK CYFRONET AGH

This software is released under the MIT Licence.

Defines

PROXY_INIT

globus-proxy-utils program names

PROXY_INFO

X509_USER_CERT_ENV

X509_USER_KEY_ENV

GLOBUS_P12_PATH

GLOBUS_PEM_CERT_PATH

GLOBUS_PEM_KEY_PATH

MSG_DEBUG_INFO

CRYPTO_FREE(M, X)

STDOUT_MAX_LEN

namespace **veil**

namespace **client**

namespace **gsi**

Functions

string **findUserCert**()

string **findUserKey**()

3.2 jobScheduler.cc

Author

Rafal Slota

Copyright

(C) 2013 ACK CYFRONET AGH

This software is released under the MIT Licence.

Defines

PTHREAD_CMD(M)

Run command M, fetch error code and if error occured restart daemon.

namespace **veil**

namespace **client**

3.3 messageBuilder.cc

Author

Beata Skiba

Copyright

(C) 2013 ACK CYFRONET AGH

This software is released under the MIT Licence.

Functions

string **tolower**(string input)

namespace **veil**

namespace **client**

3.4 veilfs.cc

Author

Rafal Slota

Copyright

(C) 2013 ACK CYFRONET AGH

This software is released under the MIT Licence.

Defines

SH_RUN(NAME, ARGS, FUN)

Runs FUN on NAME storage helper with constructed with ARGS. Return value is avaiable in 'int sh_return'.

RETURN_IF_ERROR(X)

If given veilError does not produce POSIX 0 return code, interrupt execution by returning POSIX error code.

GET_LOCATION_INFO(PATH)

Fetch locationInfo and storageInfo for given file.

On success - lInfo and sInfo variables will be set. On error - POSIX error code will be returned, interrupting code execution.

PARENT(X)

Get parent path (as string)

namespace **veil**

namespace **client**

3.5 veilFuse.cc

Author

Rafal Slota

Copyright

(C) 2013 ACK CYFRONET AGH

This software is released under the MIT Licence.

Functions

int **wrap_access**(const char * path, int mask)

int **wrap_getattr**(const char * path, struct stat * statbuf)

int **wrap_readlink**(const char * path, char * link, size_t size)

int **wrap_mknod**(const char * path, mode_t mode, dev_t dev)

int **wrap_mkdir**(const char * path, mode_t mode)

int **wrap_unlink**(const char * path)

int **wrap_rmdir**(const char * path)

int **wrap_symlink**(const char * path, const char * link)

int **wrap_rename**(const char * path, const char * newpath)

int **wrap_link**(const char * path, const char * newpath)

int **wrap_chmod**(const char * path, mode_t mode)

int **wrap_chown**(const char * path, uid_t uid, gid_t gid)

int **wrap_truncate**(const char * path, off_t newSize)

int **wrap_utime**(const char * path, struct utimbuf * ubuf)

int **wrap_open**(const char * path, struct fuse_file_info * fileInfo)

int **wrap_read**(const char * path, char * buf, size_t size, off_t offset, struct fuse_file_info * fileInfo)

int **wrap_write**(const char * path, const char * buf, size_t size, off_t offset, struct fuse_file_info * fileInfo)

int **wrap_statfs**(const char * path, struct statvfs * statInfo)

int **wrap_flush**(const char * path, struct fuse_file_info * fileInfo)

int **wrap_release**(const char * path, struct fuse_file_info * fileInfo)

int **wrap_fsync**(const char * path, int datasync, struct fuse_file_info * fi)

int **wrap_readdir**(const char * path, void * buf, fuse_fill_dir_t filler, off_t offset, struct fuse_file_info * fileInfo)

int **wrap_opendir**(const char * path, struct fuse_file_info * fileInfo)

int **wrap_releasedir**(const char * path, struct fuse_file_info * fileInfo)

int **wrap_fsyncdir**(const char * path, int datasync, struct fuse_file_info * fileInfo)

int **vfs_opt_proc**(void * data, const char * arg, int key, struct fuse_args * outargs)

void **oper_init**()

void **fuse_init**()

int **main**(int argc, char * argv, char * envp)

Variables

boost::shared_ptr< *VeilFS* > **VeilAppObject**

Main application object (filesystem state)

struct fuse_operations **vfs_oper**

Headers

3.6 config.h

Defines

CLUSTER_HOSTNAME_OPT

Config option names.

CLUSTER_PORT_OPT

LOG_DIR_OPT

PEER_CERTIFICATE_FILE_OPT

ENABLE_ATTR_CACHE_OPT

ATTR_CACHE_EXPIRATION_TIME_OPT

ENABLE_LOCATION_CACHE_OPT

ENABLE_ENV_OPTION_OVERRIDE

FUSE_ID_OPT

CLUSTER_PING_INTERVAL_OPT

JOBSCHEDULER_THREADS_OPT

ALIVE_META_CONNECTIONS_COUNT_OPT

ALIVE_DATA_CONNECTIONS_COUNT_OPT

ENABLE_DIR_PREFETCH_OPT

ENABLE_PARALLEL_GETATTR_OPT

FUSE_OPT_PREFIX

Prefix for all env variables that will be send to cluster.

PROTOCOL_VERSION

ATTR_DEFAULT_EXPIRATION_TIME

DECLARE_DEFAULT(KEY, VALUE)

namespace **veil**

namespace **client**

class **Config**

The *Config*.

Parses config files and provides safe access to configuration map.

Public Functions

std::string **getFuseID**()

Returns current FuseID.

void **negotiateFuseID**(time_t delay = 0)

Starts FuseID negotiation process.

Parameters

- `delay` - Since this is async actions, you can specify execution delay in seconds.

std::string **getString**(std::string opt)

Returns string value of requested option.

Before using this function you should check is option exists, but it's not required.

See

Config::isSet

int **getInt**(std::string opt)

Returns int value of requested option.

Before using this function you should check is option exists, but it's not required.

See

Config::isSet

double **getDouble**(std::string opt)

Returns double value of requested option.

Before using this function you should check is option exists, but it's not required.

See

Config::isSet

bool **getBool**(std::string opt)

Returns boolean value of requested option.

Before using this function you should check is option exists, but it's not required.

See

Config::isSet

Warning

If given opition wasn't set, you'll get empty object of given type T (T())

bool **isSet**(std::string opt)

Checks if given option is set.

See

Config::getValue

void **setGlobalConfigFile**(std::string path)

Sets path to global config file.

See

Config::parseConfig

void **setUserConfigFile**(std::string path)

Sets path to user config file.

See

Config::parseConfig

void **setEnv**()

Saves current CWD and HOME env viariables. This is required as FUSE changes them after non-debug start. This is also done automatically in *Config::Config*.

bool **parseConfig**()

Parses config from files set by *Config::setGlobalConfigFile* and *Config::setUserConfigFile*.

User config overides global settings. If user config declares all required options, global config file isn't required, otherwise it has exists.

Config()

~Config()

Public Static Functions

std::string **absPathRelToCWD**(std::string path)

Converts relative path, to absolute using CWD env as base prefix.

std::string **absPathRelToHOME**(std::string path)

Converts relative path, to absolute using HOME env as base prefix.

void **putEnv**(std::string name, std::string value)

Saves given env variable.

Protected Functions

template < typename T >
T **get**(std::string opt)

Internal implementation of *Config::getValue*.

See

Config::getValue

template < typename T >
T **getValue**(std::string opt)

Returns type-specialized value of given config option.

bool **runTask**(*TaskID* taskId, std::string arg0, std::string arg1, std::string arg3)

Task runner derived from *ISchedulable*.

See

ISchedulable::runTask

void **setupDefaults**()

Protected Attributes

bool **defaultsLoaded**

ReadWriteLock **m_access**

std::string **m_globalConfigPath**

Path to global config file.

See

Config::setGlobalConfigFile

std::string **m_userConfigPath**

Path to user config file.

See

Config::setUserConfigFile

YAML::Node **m_globalNode**

Global config object.

YAML::Node **m_userNode**

User config object.

YAML::Node **m_envNode**

Temp config object used to manipulate env settings.

YAML::Node **m_defaultsNode**

Default configs.

Protected Static Attributes

std::string **m_requiredOpts**[]

Array containing required options names.

std::string **m_envCWD**

Saved CWD env variable.

std::string **m_envHOME**

Saved HOME env variable.

std::map< std::string, std::string > **m_envAll**

All saved env variables.

Friends

friend class **IVeilFactory**

namespace **utils**

Functions

template < typename T >
std::string **toString**(T in)

template < typename T >
T **fromString**(std::string in)

3.7 fslogicProxy.h

Author

Beata Skiba

Rafal Slota

Copyright

(C) 2013 ACK CYFRONET AGH

This software is released under the MIT Licence.

Defines

RENEW_LOCATION_MAPPING_TIME

How many secs before expiration should focation mapping be renewed.

GET_FILE_ATTR

GET_FILE_LOCATION

GET_NEW_FILE_LOCATION

RENEW_FILE_LOCATION

GET_FILE_CHILDREN

FILE_CHILDREN

FILE_ATTR

FILE_LOCATION

CREATE_DIR

DELETE_FILE

FILE_NOT_USED

RENAME_FILE

CHANGE_FILE_PERMS

FILE_LOCATION_VALIDITY

CREATE_LINK

GET_LINK

LINK_INFO

FUSE_MESSAGE

ATOM

FSLOGIC

ACTION_NOT_ALLOWED

ACTION_FAILED

namespace **veil**

namespace **client**

class **FslogicProxy**

The *FslogicProxy* class.

This class provides proxy-methods that runs their correspondent cluster-fslogic methods. Each object of *FslogicProxy* has its own connection pool, so in order to always use already opened connections there should be only one instance of this class unless you really need connection isolation between successive cluster requests.

Public Functions

FslogicProxy()

~FslogicProxy()

bool **getFileAttr**(std::string logicName,
protocol::fuse_messages::FileAttr & attr)

Downloads file attributes from cluster.

bool **getFileLocation**(std::string logicName,
protocol::fuse_messages::FileLocation & location)

Downloads file location info.

bool **getNewFileLocation**(std::string logicName, mode_t mode,
protocol::fuse_messages::FileLocation & location)

Query cluser to create new file in DB and get its real location.

int **renewFileLocation**(std::string logicName)

Try to renew location validity for given file.

bool **getFileChildren**(std::string dirLogicName, uint32_t children_num, uint32_t offset, std::vector< std::string > & childrenNames)

List files in given folder.

std::string **renameFile**(std::string fromLogicName, std::string toLogicName)

Rename/move file to new location.

std::string **createDir**(std::string logicName, mode_t mode)

Create directory.

std::string **deleteFile**(std::string logicName)

Delete given file.

bool **sendFileNotUsed**(std::string logicName)

Inform cluster that file isnt used anymore.

std::string **changeFilePerms**(std::string path, mode_t mode)

Change file permissions.

std::string **createLink**(std::string from, std::string to)

Creates symbolic link "from" to file "to".

std::pair< std::string, std::string > **getLink**(std::string path)

Gets path pointed by link.

std::string **updateTimes**(std::string path, time_t atime = 0, time_t mtime = 0, time_t ctime = 0)

Updates *time meta attributes for specific file.

std::string **changeFileOwner**(std::string path, uid_t uid, std::string uname = "")

Updates file's owner.

std::string **changeFileGroup**(std::string path, gid_t gid, std::string gname = "")

Updates file's group owner.

void **pingCluster**(std::string nth)

bool **runTask**(*TaskID* taskId, std::string arg0, std::string arg1, std::string arg3)

Task runner derived from *ISchedulable*.

See

ISchedulable::runTask

Protected Functions

bool **sendFuseReceiveAnswer**(const google::protobuf::Message & fMsg, google::protobuf::Message & response)

Sends and receives given protobuf message.

High level method used to send serialized protobuf message to cluster and return its response as given by reference object. Both message and response types have to be subtype of FuseMessage.

Return

true only if received response message is initialized (see google::protobuff::Message::IsInitialized())

std::string **sendFuseReceiveAtom**(const google::protobuf::Message & fMsg)

Sends given protobuf message and receives atom.

This method is simalar to *FslogicProxy::sendFuseReceiveAnswer* But receives simple atom cluster response.

See

FslogicProxy::sendFuseReceiveAnswer

Protected Attributes

boost::shared_ptr< *MessageBuilder* > **m_messageBuilder**

MessageBuilder used to construct cluster packets.

3.8 gsiHandler.h

Defines

GSI_INIT_COMMAND

GSI_INFO_COMMAND

BASE_DOMAIN

namespace **veil**

namespace **client**

namespace **gsi**

Functions

bool **validateProxyConfig**()

bool **validateProxyCert**()

std::string **getProxyCertPath**()

std::string **getClusterHostname**()

Variables

bool **debug**

3.9 lock.h

Author

Rafal Slota

Copyright

(C) 2013 ACK CYFRONET AGH

This software is released under the MIT Licence.

Defines

FAIRNESS_LEVEL

Defines how many readers (minimum) should be processed after each writer (which has priority)

MUTEX(X)

Constructs name for mutex.

DECLARE_MUTEX_FOR(X)

Declare mutex with given name.

INIT_MUTEX_FOR(X)

Initialize mutex with given name as PTHREAD_MUTEX_RECURSIVE.

namespace **veil**

namespace **client**

Enums

LockType enum

See

AutoLock::AutoLock

Values:

- `READ_LOCK` -
- `WRITE_LOCK` -

class **ReadWriteLock**

The *ReadWriteLock* class.

Solves the "library problem" (fast thread-safe read access) *ReadWriteLock* should not be used directly. Use *AutoLock* wrapper instead, for automatic lock release (using RAII)

Public Functions

ReadWriteLock()

~ReadWriteLock()

void **readLock**()

Locks for read operation.

void **readUnlock**()

Unlocks read lock.

void **writeLock**()

Locks for write operation.

void **writeUnlock**()

< Unlocks write lock

Private Functions

DECLARE_MUTEX_FOR(m_class)

DECLARE_MUTEX_FOR(m_res)

Private Members

pthread_cond_t **m_classCond**

int **m_readers**

How many readers are in critical section right now.

int **m_writers**

How many writers are waitning in queue to access critical section.

int **m_fairness**

How many readers was in critical section since last writer left.

class **AutoLock**

The *AutoLock* class.

Object of this class provides auto-unlocking *ReadWriteLock*.

Public Functions

AutoLock(*ReadWriteLock* & lock, *LockType* type)

Create auto *ReadWriteLock*.

Also locks for read/write depending on *LockType* param.

~AutoLock()

Unlocks wrapped *ReadWriteLock* lock, but only if it wasn't released manually.

void **changeType**(*LockType* type)

Changle lock type.

Releases current lock and locks for diffrent operation type.

void **release**()

Release current lock.

void **lock**()

Locks current lock.

Private Members

ReadWriteLock & **m_lock**

The wrapped lock.

LockType **m_type**

Type of this lock.

See

LockType

bool **m_released**

Flag saying if lock is released right now.

3.10 messageBuilder.h

Author

Beata Skiba

Copyright

(C) 2013 ACK CYFRONET AGH

This software is released under the MIT Licence.

Defines

FUSE_MESSAGES

COMMUNICATION_PROTOCOL

GET_FILE_LOCATION

FUSE_MESSAGE

FSLOGIC

namespace **veil**

namespace **client**

class **MessageBuilder**

The *MessageBuilder* class.

This class can be used to build protobuf messages used to communicate with cluster. Theres encode and decode method for each base message type used by VeilClient. Arguments matches proto specification of their messages.

Public Functions

MessageBuilder()

~MessageBuilder()

protocol::fuse_messages::FuseMessage **createFuseMessage**(std::string id, std::string messageType, std::string messageInput)

protocol::communication_protocol::ClusterMsg **createClusterMessage**(std::string moduleName, std::string messageType, std::string answerType, std::string answerDecoderName, bool synch, std::string input)

protocol::communication_protocol::ClusterMsg **createClusterMessage**(std::string moduleName, std::string messageType, std::string answerType, std::string answerDecoderName, bool synch)

protocol::communication_protocol::ClusterMsg **packFuseMessage**(std::string messageType, std::string answerType, std::string answerDecoderName, std::string messageInput)

protocol::fuse_messages::FuseMessage **decodeFuseAnswer**(protocol::communication_protocol::Answer & answer)

std::string **decodeAtomAnswer**(protocol::communication_protocol::Answer & answer)

3.11 veilfs.h

Author

Rafal Slota

Copyright

(C) 2013 ACK CYFRONET AGH

This software is released under the MIT Licence.

Defines

GLOBAL_CONFIG_FILE

The name of default global config file.

CONFIG_ARGV_OPT_NAME

The command line pattern used to find user config path ARGV.

DIR_BATCH_SIZE

How many dirent should be fetch from cluster at once.

Note that each opendir syscall will query at least DIR_BATCH_SIZE dirents

namespace **veil**

namespace **client**

class **VeilFS**

The *VeilFS* main class.

This class contains FUSE all callbacks, so it basically is an heart of the filesystem. Technically *VeilFS* is an singleton created on programm start and registred in FUSE daemon.

Public Functions

VeilFS(std::string path, boost::shared_ptr< *Config* > cnf, boost::shared_ptr< *JobScheduler* > scheduler, boost::shared_ptr< *FslogicProxy* > fslogic, boost::shared_ptr< *MetaCache* > metaCache, boost::shared_ptr< *StorageMapper* > mapper, boost::shared_ptr< helpers::StorageHelperFactory > sh_factory)

VeilFS constructor.

~VeilFS()

int **access**(const char * path, int mask)

access FUSE callback.

Not implemented yet.

int **getattr**(const char * path, struct stat * statbuf, bool fuse_ctx = true)

getattr FUSE callback.

See

int **readlink**(const char * path, char * link, size_t size)

readlink FUSE callback.

Not implemented yet.

See

int **mknod**(const char * path, mode_t mode, dev_t dev)

mknod FUSE callback.

See

int **mkdir**(const char * path, mode_t mode)

mkdir FUSE callback.

See

int **unlink**(const char * path)

unlink FUSE callback.

See

int **rmdir**(const char * path)

rmdir FUSE callback.

See

int **symlink**(const char * path, const char * link)

symlink FUSE callback.

Not implemented yet.

See

int **rename**(const char * path, const char * newpath)

rename FUSE callback.

See

int **link**(const char * path, const char * newpath)

link FUSE callback.

Not implemented yet.

See

int **chmod**(const char * path, mode_t mode)

chmod FUSE callback.

See

int **chown**(const char * path, uid_t uid, gid_t gid)

chown FUSE callback.

Not implemented yet.

See

int **truncate**(const char * path, off_t newSize)

truncate FUSE callback.

See

int **utime**(const char * path, struct utimbuf * ubuf)

utime FUSE callback.

Not implemented yet.

See

int **open**(const char * path, struct fuse_file_info * fileInfo)

open FUSE callback.

See

int **read**(const char * path, char * buf, size_t size, off_t offset, struct fuse_file_info * fileInfo)

read FUSE callback.

See

int **write**(const char * path, const char * buf, size_t size, off_t offset, struct fuse_file_info * fileInfo)

write FUSE callback.

See

int **statfs**(const char * path, struct statvfs * statInfo)

statfs FUSE callback.

Not implemented yet.

See

int **flush**(const char * path, struct fuse_file_info * fileInfo)

flush FUSE callback.

Not implemented yet.

See

int **release**(const char * path, struct fuse_file_info * fileInfo)

release FUSE callback.

See

int **fsync**(const char * path, int datasync, struct fuse_file_info * fi)

fsync FUSE callback.

Not implemented yet.

See

int **setxattr**(const char * path, const char * name, const char * value, size_t size, int flags)

setxattr FUSE callback.

Not implemented yet.

See

int **getxattr**(const char * path, const char * name, char * value, size_t size)

getxattr FUSE callback.

Not implemented yet.

See

int **listxattr**(const char * path, char * list, size_t size)

listxattr FUSE callback.

Not implemented yet.

See

int **removexattr**(const char * path, const char * name)

removexattr FUSE callback.

Not implemented yet.

See

int **opendir**(const char * path, struct fuse_file_info * fileInfo)

opendir FUSE callback.

Not implemented yet.

See

int **readdir**(const char * path, void * buf, fuse_fill_dir_t filler, off_t offset, struct fuse_file_info * fileInfo)

readdir FUSE callback.

See

int **releasedir**(const char * path, struct fuse_file_info * fileInfo)

releasedir FUSE callback.

Not implemented yet.

See

int **fsyncdir**(const char * path, int datasync, struct fuse_file_info * fileInfo)

fsyncdir FUSE callback.

Not implemented yet.

See

int **init**(struct fuse_conn_info * conn)

init FUSE callback.

See

bool **runTask**(*TaskID* taskId, std::string arg0, std::string arg1, std::string arg3)

Task runner derived from *ISchedulable*.

See

ISchedulable::runTask

Public Static Functions

boost::shared_ptr< *JobScheduler* > **getScheduler**(*TaskID* taskId = TASK_LAST_ID)

Returns *JobScheduler* assigned to this object.

boost::shared_ptr< *Config* > **getConfig**()

Returns *Config* assigned to this object.

boost::shared_ptr< SimpleConnectionPool > **getConnectionPool**()

boost::shared_ptr< *PushListener* > **getPushListener**()

void **addScheduler**(boost::shared_ptr< *JobScheduler* > injected)

Sets *JobScheduler* object.

void **setConfig**(boost::shared_ptr< *Config* > injected)

Sets *Config* object.

void **setConnectionPool**(boost::shared_ptr< SimpleConnectionPool > injected)

void **staticDestroy**()

Private Members

std::string **m_root**

Filesystem root directory.

uid_t **m_uid**

Filesystem owner's effective uid.

gid_t **m_gid**

Filesystem owner's effective gid.

uid_t **m_ruid**

Filesystem root real uid.

gid_t **m_rgid**

Filesystem root real gid.

boost::shared_ptr< *FslogicProxy* > **m_fslogic**

FslogicProxy instance.

boost::shared_ptr< *StorageMapper* > **m_storageMapper**

StorageMapper instance.

boost::shared_ptr< *MetaCache* > **m_metaCache**

MetaCache instance.

boost::shared_ptr< helpers::StorageHelperFactory > **m_shFactory**

Storage Helpers Factory instance.

std::map< std::string, std::pair< std::string, time_t > > **m_linkCache**

Simple links cache.

ReadWriteLock **m_linkCacheLock**

Private Static Attributes

ReadWriteLock **m_schedulerPoolLock**

std::list< boost::shared_ptr< *JobScheduler* > > **m_jobSchedulers**

JobScheduler instances.

boost::shared_ptr< *Config* > **m_config**

Config instance.

boost::shared_ptr< SimpleConnectionPool > **m_connectionPool**

boost::shared_ptr< *PushListener* > **m_pushListener**

Globus proxy utils

3.12 fmemopen.c

Typedefs

typedef struct *fmem* **fmem_t**

Functions

int **readfn**(void * handler, char * buf, int size)

int **writefn**(void * handler, const char * buf, int size)

fpos_t **seekfn**(void * handler, fpos_t offset, int whence)

int **closefn**(void * handler)

FILE * **fmemopen**(void * buf, size_t size, const char * mode)

A BSD port of the fmemopen Linux method using funopen.

man docs for fmemopen:

man docs for funopen:

This method is ported from ingenuitas' python-tesseract project.

You must call fclose on the returned file pointer or memory will be leaked.

```
@param buf The data that will be used to back the FILE* methods. Must be at least
           @c size bytes.
@param size The size of the @c buf data.
@param mode The permitted stream operation modes.
@returns A pointer that can be used in the fread/fwrite/fseek/fclose family of metho
         If a failure occurred NULL will be returned.
```

class **fmem**

Public Members

size_t **pos**

size_t **size**

char * **buffer**

3.13 fmemopen.h

Functions

FILE * **fmemopen**(void * buf, size_t size, const char * mode)

A BSD port of the fmemopen Linux method using funopen.

man docs for fmemopen:

man docs for funopen:

This method is ported from ingenuitas' python-tesseract project.

You must call fclose on the returned file pointer or memory will be leaked.

```
@param buf The data that will be used to back the FILE* methods. Must be at least
          @c size bytes.
@param size The size of the @c buf data.
@param mode The permitted stream operation modes.
@returns A pointer that can be used in the fread/fwrite/fseek/fclose family of metho
         If a failure occurred NULL will be returned.
```

3.14 globus_stdio_ui.c

Functions

int **globus_l_stdio_ui_activate**(void)

int **globus_l_stdio_ui_deactivate**(void)

int **globus_l_stdio_ui_read**(UI * ui, UI_STRING * uis)

int **globus_l_stdio_ui_write**(UI * ui, UI_STRING * uis)

Variables

globus_module_descriptor_t **globus_i_stdio_ui_module**

const UI_METHOD * **globus_l_stdio_ui_old_method**

UI_METHOD * **globus_l_stdio_ui_method**

3.15 globus_stdio_ui.h

Defines

GLOBUS_STDIO_UI_MODULE

Variables

globus_module_descriptor_t **globus_i_stdio_ui_module**

3.16 grid_proxy_info.c

Defines

SHORT_USAGE_FORMAT

args_show_short_help()

args_show_full_usage()

args_error_message(errmsg)

args_error(argp, errmsg)

GLOBUS_I_GSI_PROXY_UTILS_PRINT_ERROR

STATUS_OK

STATUS_EXPIRED

STATUS_NOT_FOUND

STATUS_CANT_LOAD

STATUS_NO_NAME

STATUS_BAD_OPTS

STATUS_INTERNAL

Functions

int **globus_i_gsi_proxy_utils_print_error**(globus_result_t result, int debug, const char * filename, int line, const char * fmt, ...)

int **proxy_info**(int argc, char ** argv, FILE * _vout, FILE * _verr)

This function replaces standard "main" of grid-proxy-info tool.

First two arguments are standard "main's" arugments. The others are used to capture standard output and standard error steams.

Variables

int **debug**

char * **LONG_USAGE**

3.17 grid_proxy_init.c

Defines

SHORT_USAGE_FORMAT

args_show_short_help()

args_show_full_usage()

args_error_message(errmsg)

args_error(argval, errmsg)

args_verify_next(argnum, argval, errmsg)

Functions

int **globus_i_gsi_proxy_utils_print_error**(globus_result_t result, int debug, const char * filename, int line, const char * fmt, ...)

int **globus_i_gsi_proxy_utils_pwstdin_callback**(char * buf, int num, int w)

void **globus_i_gsi_proxy_utils_key_gen_callback**(int p, int n, void * dummy)

int **globus_l_gsi_proxy_utils_extension_callback**(globus_gsi_callback_data_t callback_data, X509_EXTENSION * extension)

int **proxy_init**(int argc, char ** argv, FILE * _vout, FILE * _verr, char const * _pass, int _passLen)

This function replaces standard "main" of grid-proxy-init tool.

First two arguments are standard "main's" arugments. The others are used to capture standard output and standard error steams and pass cached certificate passphrase.

Variables

int **quiet**

int **debug**

FILE * **vout**

stdout and stderr substitutions used in grid-proxy-utils

FILE * **verr**

char const * **certPasswd**

int **passwdLen**

char * **LONG_USAGE**

3.18 gsi_utils.h

Author

Rafal Slota

Copyright

Functions

int **proxy_init**(int argc, char ** argv, FILE * _vout, FILE * _verr, char const * _pass, int _passLen)

This function replaces standard "main" of grid-proxy-init tool.

First two arguments are standard "main's" arugments. The others are used to capture standard output and standard error steams and pass cached certificate passphrase.

int **proxy_info**(int argc, char ** argv, FILE * _vout, FILE * _verr)

This function replaces standard "main" of grid-proxy-info tool.

First two arguments are standard "main's" arugments. The others are used to capture standard output and standard error steams.

Variables

FILE * **vout**

stdout and stderr substitutions used in grid-proxy-utils

FILE * **verr**

CHAPTER 4

Namespaces

Documents

4.1 veil::client::gsi

bool **debug**

bool **validateProxyConfig**()

bool **validateProxyCert**()

std::string **getProxyCertPath**()

std::string **getClusterHostname**()

string **findUserCert**()

string **findUserKey**()

4.2 veil::client

LockType enum

See

AutoLock::AutoLock

Values:

- READ_LOCK -
- WRITE_LOCK -

typedef boost::function< bool(const protocol::communication_protocol::Answer &)> **listener_fun**

typedef struct *veil::client::locationInfo* **locationInfo**

Structure containing file mapping base information.

typedef struct *veil::client::storageInfo* **storageInfo**

4.3 veil::client::utils

template < typename T >
std::string **toString**(T in)

template < typename T >
T **fromString**(std::string in)

CHAPTER 5

Structs

Documents

5.1 fmem

struct **fmem**

Public Members

size_t **pos**

size_t **size**

char * **buffer**

5.2 veil::client::Job

struct **veil::client::Job**

#include <include/jobScheduler.h>

The *Job* struct Used by *JobScheduler* to describe scheduled tasks.

Public Functions

Job(time_t when, boost::shared_ptr< *ISchedulable* > subject, *ISchedulable::TaskID* task, std::string arg0 = “”, std::string arg1 = “”, std::string arg2 = “”)

bool **operator<**(const *Job* & other)

Compare operator for priority queue.

It compares tasks by it’s *Job::when* field

bool **operator==**(const *Job* & other)

Compares equal all fields excpet *Job::when*.

Public Members

time_t **when**

Time when *Job* should be processed.

boost::shared_ptr< *ISchedulable* > **subject**

Pointer to object being context of *Job* execution.

ISchedulable::TaskID **task**

ID of task.

See

ISchedulable::TaskID

std::string **arg0**

Task's first argument.

std::string **arg1**

Task's second argument.

std::string **arg2**

Task's third argument.

Default constructor

5.3 veil::client::locationInfo

struct **veil::client::locationInfo**
#include <include/storageMapper.h>

Structure containing file mapping base information.

Public Functions

bool **isValid**()

Public Members

int **storageId**

Storage identificator.

See

StorageMapper::m_storageMapping

std::string **fileId**

File identificator.

This ID should be recognisable by *storage helper*. Most commonly it's just file path relative to the *storage*

See

StorageMapper::m_fileMapping

time_t **validTo**

Mapping expiration time.

int **opened**

How many files are currently opened using this mapping.

5.4 veil::client::storageInfo

struct **veil::client::storageInfo**
#include <include/storageMapper.h>

Public Functions

bool **isValid**()

Public Members

time_t **last_updated**

Last update time.

std::string **storageHelperName**

Name of storage helper.

See

StorageHelperFactory::getStorageHelper

std::vector< std::string > **storageHelperArgs**

Arguments for storage helper.

See

StorageHelperFactory::getStorageHelper

Part II

VeilCluster

Documents

CHAPTER 6

About

VeilCluster is a part of VeilFS system that unifies access to files stored at heterogeneous data storage systems that belong to geographically distributed organizations.

6.1 Goals

The main goal of VeilCluster is to provision a self-scalable cluster, which manages the VeilFS system in a single data centre, i.e. it stores meta-data about actual users' data from the data centre, decides how to distribute users' files among available storage systems, and executes data management rules, which can be defined by administrators or users.

6.2 Getting Started

This is a short tutorial how to start VeilCluster on a single machine.

6.2.1 Prerequisites

In order to compile the project, you need to have the following libraries:

- libglobus_gsi_callback
- libglobus_common
- libssl
- libfuse
- libboost - filesystem, thread, random, system (version >= 1.49)

Use the following command to install the required dependency packages:

- RHEL/CentOS/Fedora Dependencies (.rpm packages):

```
yum install globus-proxy-utils globus-gsi-callback-devel fuse-libs fuse-devel cmake28 boost-

ln -s /usr/lib64/globus/include/globus_config.h /usr/include/globus
```

6.2.2 VeilCluster RPM installation and configuration

After installing prerequisites, you can install the VeilCluster RPM.

```
rpm -i veil-<version>.rpm
```

VeilCluster setup scripts can be executed with the following command:

```
veil_setup
```

The script will guide you through the setup procedure of the VeilCluster. To start the work you should start at least one instance of the CCM component and one worker component instance.

A sample session of starting VeilCluster from scratch is as follows:

```
[root@plgsl63 veilcluster]# veil_setup

*** Veil SETUP ***
~ Nodes configured on this machine will use its hostname: @172.16.67.219
(!) Make sure it is resolvable by other hosts in the network
==> What do you want to do?
 [1] Manage database nodes
 [2] Manage veil nodes
 [3] Exit
> Your choice: 2
~ Each machine can only host a single worker or a ccm + worker pair.
==> What do you want to do?
 [1] Set up a new cluster
 [2] Extend existing cluster
 [3] Go back
> Your choice: 1
~ Installing a new cluster beside a running one may cause unpredictable behaviour.
~ It is required that all database nodes are installed prior to the cluster.
==> Do you wish to continue?
 [1] Yes
 [2] No
> Your choice: 1
==> List ALL running database nodes, delimiting them with commas (no spaces) [eg. db1@host.net,d
==> The cluster will use ONLY the nodes specified now.
> Running DB nodes: db@172.16.67.219
==> Connection to following database nodes has been confirmed:
 - db@172.16.67.219
==> Following nodes will be installed:
 - ccm@172.16.67.219
 - worker@172.16.67.219
==> Confirm:
 [1] Continue
 [2] Go back
> Your choice: 1
~ Installing ccm@172.16.67.219...
~ Installing worker@172.16.67.219...
~ Starting node(s)...
```

To check if the VeilCluster is running you can attach to the CCM component and check heartbeat messages:

```
[root@plgsl63 veilcluster]# /opt/veil/nodes/ccm/bin/veil_cluster_node attach
Attaching to /tmp//opt/veil/nodes/ccm/erlang.pipe.1 (^D to exit)

(search)`':
```

```
(ccm@172.16.67.219)1> 11:07:00.454 [info] Heart beat on node: ccm@172.16.67.219: sent; connectic
11:07:00.454 [info] Heart beat on node: ccm@172.16.67.219: answered, new state_num: 4

(ccm@172.16.67.219)1> 11:07:09.989 [info] Cluster state ok
```

After starting the nodes you have a complete VeilCluster running :) You can now proceed to VeilClient installation.

Note:

- To have a fully working VeilFS installation, we should also start a BigCouch instance on the same machine, with its cookie set to 'veil_cluster_node' and hostname set to 'db'.

6.3 Support

For more information visit project Confluence or write to 'wrzeszcz@agh.edu.pl ('wrzeszcz@agh.edu.pl)'.

CHAPTER 7

Development

This document intends to provide essential information for VeilCluster developers. General information about VeilCluster usage can be found in the *About* file.

7.1 Repo layout

- c_src # TODO uzupelnic
- cacerts # TODO uzupelnic
- config # TODO uzupelnic
- include # TODO uzupelnic
- releases # folder for generated VeilCluster releases
- src
  - cluster_elements # includes directories that contain code of Erlang modules that enable cluster management and host 'proper' modules of VeilFS
  - veil_modules # includes directories that contain code of 'proper' modules of VeilFS
  - proto # definitions of protocol buffer messages used by clients during the communication with VeilCluster
- test
- test_distributed # TODO uzupelnic - czy to nie moze byc w 'test' ???
- ... # othe VeilCluster files

7.2 Src

Directly in the 'src' directory only files needed to start application can be put.

The 'cluster_elements' includes source of Erlang modules, which are responsible for load balancing, spawning processes for requests etc. This allows implementation of 'proper' modules using sequential code.

The 'veil_modules' includes directories that contain code of 'proper' modules of VeilFS. Each 'proper' module will work inside of 'worker_host' (one of 'cluster_elements') so it must implement 'worker_plugin_behaviour' defined in 'worker_plugin_behaviour.erl' file in this directory.

7.3 Tests

It should contain the same subdirectories as 'src'. Each test name should be constructed as follows: 'name of tested file'*tests.erl, e.g. 'node*manager_tests.erl' should contain functions that test code from 'node_manager.erl' file.

Eunit is used during tests so each test file should:

- include "eunit/include/eunit.hrl",
- use proper functions names (name of each test function must end with '_test'),
- use compilation control macros (code between '-ifdef(TEST).' and '-endif.').

7.4 Releases

To create new release, version must be changed in both 'src/veil_cluster_node.app.src' and 'releases/reltool.conf'.

Node:

- Documentation is generated automatically using edoc so it should use tags defined by it.

7.5 Development - using Makefile to generate single releases and test environments of veil cluster nodes

7.5.1 Generating and managing a single node release for development purposes

The script 'gen_dev' generates a single node release for testing purposes. It uses 'veil_cluster' script to set the configuration.

Every node (worker or CCM) requires information about all CCMs running in the cluster. Hence to generate release of a node it is required to specify the following set of arguments:

```
-name node_name@host -main_ccm main_ccm_node@host [-opt_ccms opt_ccm_node1@host opt_ccm_node2@hc
```

- The expression after -name specifies the node for which the release will be generated. It can be one of CCMs listed later on.
- The expression after -main_ccm specifies the node name of the main CCM.
- The expression after -opt_ccms specifies the list of optional CCMs. These arguments are not mandatory.
- The expression after -db_nodes specifies the list of all DBMS nodes.

The above argument string can be either placed in file 'gen_dev.args' located in the root directory or passed to Makefile.

Optionally, one of following can be added to arguments:

```
-no_compile     -> skips code compilation straight to release generation
-no_generate    -> skips code compilation and release generation (creates a copy of existing rel
```

7.5.2 Generating a release

```
~$  make gen_test_node args="-name node_name@host -main_ccm main_ccm_node@host -opt_ccms opt_ccm

~$  make gen_test_node_from_file
```

After either of these operations, the release will be placed in releases/test_cluster/node_name, with all the environment variables set up.

7.5.3 Starting and managing generated packages

```
~$  make start_node node="node_name"              -> starts the node called 'node_name' as a da

~$  make attach_to_node node="node_name"          -> attaches to the running node called 'node_

~$  make start_node_console node="node_name"      -> combines the two above
```

Note, that it only works for packages in releases/test_cluster/ - those created with 'gen_dev' or 'gen_test'

7.5.4 Generating a local test environment

The script 'gen_test' simplifies setting up a bunch of cluster nodes for testing. It uses both 'gen_dev' and 'veil_cluster' scripts. To generate a testing environment proper arguments must be passed to the script:

```
-workers worker1@host worker2@host ... -main_ccm main_ccm_node@host [-opt_ccms opt_ccm_node1@hos
```

- The expression after -workers specifies the list of workers in the cluster.
- The expression after -main_ccm specifies the node name of the main CCM.
- The expression after -opt_ccms specifies the list of optional CCMs in the cluster (not mandatory).
- The expression after -db_nodes specifies the list of all DBMS nodes.

Again, these arguments can be obtained from 'gen_test.args' or passed via args to Makefile.

7.5.5 Generating releases for the whole cluster

```
~$  make gen_test_env args="-workers worker1@host -main_ccm main_ccm_node@host -opt_ccms opt_ccm

~$  make gen_test_env_from_file
```

Both of these commands produce a release for each node in corresponding directories (/releases/test_cluster/).

7.5.6 Starting previously generated cluster

```
~$  make start_test_env args="-workers worker1@host -main_ccm main_ccm_node@host -opt_ccms opt_c

~$  make start_test_env_from_file
```

It is important that the same args are passed to Makefile or remain in the .args file. This is the way for the script to know which release packages need to be started.

7.5.7 Generating and immediately starting the whole cluster

```
~$  make gen_start_test_env args="-workers worker1@host -main_ccm main_ccm_node@host -opt_ccms c

~$  make gen_start_test_env_from_file
```

Every node can be started independently with use of 'start_node', 'attach_to_node' and 'start_node_console' make targets.

7.5.8 Compilation and Releases from Sources

To build a working release of VeilCluster from scratch, type the following commands:

```
~$  make test
~$  make generate
```

After this step we should have a 'veil_cluster_node' folder in the 'releases' folder.

7.5.9 Execution

Now, we are ready to start CCM and worker processes:

```
~$  cp -R releases/veil_cluster_node releases/veil_cluster_node_worker
~$  ./releases/veil_cluster_node/bin/veil_cluster -name ccm@127.0.0.1 -main_ccm ccm@127.0.0.1 -c
~$  ./releases/veil_cluster_node_worker/bin/veil_cluster -name worker@127.0.0.1 -main_ccm ccm@12

~$  curl https://127.0.0.1:8000
```

If everything went correct, two processes should be started and a default web site of the VeilCluster should be available.

Note:

- To have a fully working VeilFS installation, we should also start a BigCouch instance on the same machine, with its cookie set to 'veil_cluster_node' and hostname set to 'db'.

7.5.10 Useful make commands

```
~$  make compile # compilation

~$  make test # compilation & execution of unit tests

~$  make generate # compilation & creation of release:

~$  make docs # documentation generation

~$  make PREV="name of directory with previous release" upgrade # generation of package for hot

~$ make dialyzer # starting 'dialyzer' in order to analyze binaries in ./ebin:
```

Note:

- in Linux you should use './rebar' instead of 'rebar'
- rebar is built with Erlang/OTP R16B - if you have other Erlang version installed, it may not work properly (in this case download rebar sources and rebuilt rebar)

7.6 Release management

After generation of a release package, configuration files contain default parameters. The script 'veil_cluster', that comes with the package (in the 'bin' directory) is used to set up and start a release.

7.6.1 Prerequisites

Firstly, the user must have execution rights on both './bin/veil_cluster' and './bin/veil_cluster_node' scripts.

// TODO to along veil_cluster czy w config ??? Secondly, 'config.args' file must be present in the 'config' directory (along with 'veil_cluster' script).

7.6.2 Setting parameters

List of parameters that can be set:

```
name       -> name of the node (erlang long name)
main_ccm   -> main CCM name (of the cluster this node operates in)
opt_ccms   -> list of optional CCMs (this parameter is not mandatory)
db_nodes   -> list of DBMS nodes
```

Primarily, these parameters are retrieved from 'config.args' file. It should contain these parameters in the following manner:

```
<parameter_name>: <parameter_value>
```

Another way is passing these parameters via command line arguments. In this case the syntax is:

```
./veil_cluster -<parameter1_name> <parameter1_value> -<parameter2_name> <parameter2_value> ...
```

NOTE:

- parameters passed via command line OVERRIDE those in 'config.args' file

- "command line way" can specify any subset of parameters, for instance:
  - './veil_cluster' will cause the script to use all parameters from 'config.args'
  - './veil_cluster -name somename@host.net (somename@host.net)' will cause the script to use parameters from 'config.args' except node name
  - './veil_cluster -opt_ccms' (no opt_ccms value specified) will override opt_ccms from 'config.args' with null value
- (both) parameter order can be arbitrary
- (both) multiple values (eg. DBMS nodes) are passed as a space-delimited list (eg. -db_nodes dbnode1@host.net (dbnode1@host.net) dbnode2@host.net (dbnode2@host.net))
- (both) parameter values can't contain spaces or hyphens
- (command line) parameter names are preceded by a hyphen; '-'

7.6.3 Starting a release

There are three options which are used to start a release. They are passed along with other arguments to the 'veil_cluster' script.

```
-start    -> the script will perform the configuration and then start the node as a daemon
-attach   -> the script will skip configuration and try to attach to a running node with an erla
-console  -> the script will perform the configuration and then start the node with an erlang sh
```

If none of these arguments occur, the script will terminate after setting up the configuration.

7.6.4 Full example of usage

```
~$  ./veil_cluster -name mynode@host.net -main_ccm ccmnode@host.net -console
```

Above command will configure the release according to 'config.args', except for name and main_ccm which will be modified corresponding to command line arguments. Then, the node will be started presenting to the user with an erlang shell.

7.7 Support

For more information visit project Confluence or write to 'wrzeszcz@agh.edu.pl ('wrzeszcz@agh.edu.pl)'.

CHAPTER 8

Installation

Using current RPM version it is possible to install application's nodes, Bigcouch database and configure storage. Undermentioned instructions are valid for all Red Hat Linux distributions, especially for Scientific Linux.

8.1 Dependencies

```
$ yum install js-devel libicu libicu-devel openssl openssl-devel python python-devel rpm-bui
```

8.2 System preparation

In order to make our machine recognisable for other machines in the network it is required to set its hostname properly. It is recommended to set hostname to IP address, so that modifications of /etc/hosts file are not necessary.

```
$ vi /etc/sysconfig/network -> set HOSTNAME=172.16.67.X -> save file
$ hostname 172.16.67.X -> (instant effect, system reboot is not necessary)
```

Moreover it is required to unlock ports in iptables:

```
$ service iptables stop
$ chkconfig --del iptables (turn off service permanently - necessary if machine will be rebo
```

8.3 RPM build and installation

After project cloning:

```
$ make rpm (create .rpm in ./releases/veil-0.0.6-1.x86_64.rpm)
$ yum localinstall ./releases/veil-0.0.6-1.x86_64.rpm (start RPM installation)
$ veil_setup (start installation script)
```

Warning:

- database needs to be installed before cluster
- during cluster installation it is required to give full database node name (e.g. db@172.16.67.143 (db@172.16.67.143))
- **during cluster installation in storage definition phase:**
  - give storage mount point used by cluster to store data (e.g. defined users or groups)
  - define optional storages used by fuse client groups, for each of them give mount point and name of fuse client group that is supposed to use it
- during node addition to existing cluster it is only required to give IP address of any working node (e.g. 172.16.67.143)

8.4 Useful commands

```
$ /opt/veil/nodes/ccm/bin/veil_cluster_node attach (connect to local CCM. Warning! to termir
$ /opt/veil/nodes/worker/bin/veil_cluster_node attach (connect to local worker. Warning! to
$ yum remove veil (clean deletion of RPM)
```

8.5 Vocabulary

- *database node* - installed Bigcouch instance
- *veil node* - installed CCM and worker or only worker
- *db cluster* - connected and cooperating database nodes group
- *veil cluster, cluster, veil* - connected and cooperating veil nodes group
- *storage* - it is composed of group name that use it and mount point
- *group name* - fuse client group name that use storage
- *storage directory* - storage mount point

Submodules

CHAPTER 9

REST API

Documents

9.1 Authentication

In brief authentication proceed through following steps:

- Client opens HTTPS connection and sends his certificate (in most cases proxy certificate) and all other certificates required to associate his certificate with certificate authorities present on cluster.
- Each HTTPS connection is accepted regardless of sent certificates. However all certificates are saved in connection session.
- If connection requires authentication, and in case of REST requests it does, immediately after connection is established sent certificates are validated using Grid Security Infrastructure.
- If sent certificate fails verification or is not not present in database, connection is closed.
- If sent certificate passes verification its distinguished name is associated with single entry in users database and since than client can access cluster resources.

How to create proxy certificate?

To create proxy certificate run *grid-proxy-init* script with argument *-out path_to_proxy_cert*. This script can be found at root directory on cluster and requires presence of main PLGrid certificate in */root/.globus* directory.

Example:

```
sh grid-proxy-init -out proxy_cert
```

How to use proxy certificate?

Proxy certificate can be used with any computer software that provides data transfer using HTTPS protocol and supports HTTPS certificates. In following example content of *dir* directory will be retrieved using HTTP GET request and curl (http://curl.haxx.se/) tool.

Example:

```
curl -i -k --cert proxy_cert -X GET https://example.com/rest/latest/files/dir
```

> **Warning:** VeilFS REST API doesn't work with every version of **curl** tool. It has to be linked to *OpenSSL* library. To check it use *curl –version* command.

9.2 Errors & Validation

If a request fails due to client error, the resource will return an HTTP response code in the 40x range. These can be broadly categorised into:

HTTP Code	Description
400 (Bad Request) 401 (Unauthorized) 403 (Forbidden) 404 (Not Found) 405 (Method Not Allowed)	

9.3 Getting started

REST API is based on open standards, you can use any web development language or command line tool capable of generating an HTTP request to access the API.

9.4 Resources

REST-ish endpoint for interacting with VeilFS.

9.4.1 Index

9.4.2 /rest/latest/files

Methods

`GET /rest/latest/files`

Retrieve content of root directory and return as a list of names of files and subdirectories.

Response Headers

- **Content-Type** – application/json

Status Codes

- **200** – OK
- **500** – Internal Server Error

Example request:

```
GET /rest/latest/files HTTP/1.1
Header: "content-type: application/json"
Host: example.com
```

Example responses:

```
HTTP/1.1 200 OK
connection: close
server: Cowboy
date: Sun, 05 Jan 2014 16:34:54 GMT
content-length: 12
Access-Control-Allow-Origin: *
content-type: application/json

["dir1","dir2","groups","file.txt"]
```

9.4.3 /rest/latest/files/(path)

Methods

GET /rest/latest/files/ (*path*)

Retrieve content of specified file or directory. For path to an existing file this request returns its content. For path to an existing directory this request returns list of names of contained files and subdirectories.

Parameters

- **path** (*string*) – path to file or directory

Response Headers

- **Content-Type** – application/json for path to directory
- **Content-Type** – MIME type for path to file

Status Codes

- **200** – OK
- **404** – Not Found
- **500** – Internal Server Error

Example request:

```
GET /rest/latest/files/dir1/dir2 HTTP/1.1
Header: "content-type: application/json"
Host: example.com
```

Example responses:

```
HTTP/1.1 200 OK
connection: close
server: Cowboy
date: Sun, 05 Jan 2014 16:34:54 GMT
content-length: 12
Access-Control-Allow-Origin: *
content-type: application/json

["file.txt"]
```

POST /rest/latest/files/ (*path*)

Upload data to specified path using multipart method. Path has to be a valid file system path, that is it can't contain regular file as a subdirectory. Specified path can't exist and will be created automatically.

Parameters
- **path** (*string*) – path where file will be uploaded

Request Headers
- **Content-Type** – multipart/form-data

Response Headers
- **Content-Type** – application/octet-stream

Status Codes
- **100** – Continue
- **204** – No Content
- **422** – Unprocessable Entity

An example curl (http://curl.haxx.se/) request to upload file 'file.txt', that is located in current directory, to remote location */dir/file.txt* would be:

```
curl -i -k --cert proxy_cert -X POST -H "content-type: multipart/form-data" -F "file=@fi
```

Example request:

```
POST /rest/latest/files/dir/file.txt HTTP/1.1
Host: example.com
Header: "content-type: multipart/form-data"
Data: "file=@file.txt"
```

Example responses:

```
HTTP/1.1 100 Continue

HTTP/1.1 204 No Content
connection: close
server: Cowboy
date: Fri, 24 Jan 2014 08:43:05 GMT
content-length: 0
Access-Control-Allow-Origin: *
content-type: application/octet-stream
```

PUT /rest/latest/files/ (*path*)

Upload data to specified path using multipart method. Path has to be a valid file system path, that is it can't contain regular file as a subdirectory. If specified path doesn't exist it will be created automatically. If specified path exists it will be overwritten.

Parameters
- **path** (*string*) – path where file will be uploaded

Request Headers
- **Content-Type** – multipart/form-data

Response Headers
- **Content-Type** – application/octet-stream

Status Codes
- **100** – Continue
- **204** – No Content
- **422** – Unprocessable Entity

An example curl (http://curl.haxx.se/) request to upload file 'file.txt', that is located in current directory, to remote location */dir/file.txt* would be:

```
curl -i -k --cert proxy_cert -X PUT -H "content-type: multipart/form-data" -F "file=@fil
```

Example request:

```
PUT /rest/latest/files/dir/file.txt HTTP/1.1
Host: example.com
Header: "content-type: multipart/form-data"
Data: "file=@file.txt"
```

Example responses:

```
HTTP/1.1 100 Continue
```

```
HTTP/1.1 204 No Content
connection: close
server: Cowboy
date: Fri, 24 Jan 2014 08:43:05 GMT
content-length: 0
Access-Control-Allow-Origin: *
content-type: application/octet-stream
```

DELETE /rest/latest/files/ (*path*)

Delete regular file at specified path if it exists.

Parameters

- **path** (*string*) – path to file or directory

Response Headers

- **Content-Type** – application/json
- **Content-Type** – application/octet-stream

Status Codes

- **204** – No Content
- **404** – Not Found

Example request:

```
DELETE /rest/latest/files/dir/file.txt HTTP/1.1
Header: "content-type: application/json"
Host: example.com
```

Example responses:

```
HTTP/1.1 404 Not Found
connection: close
server: Cowboy
date: Fri, 24 Jan 2014 08:50:18 GMT
content-length: 0
Access-Control-Allow-Origin: *
content-type: application/json
```

9.4.4 /rest/latest/attrs/(path)

Methods

GET /rest/latest/attrs/ (*path*)

Retrieve attributes of specified file or directory as a record of structure *{property : value}*.

Fields of returned record:

- file protection mode
- file owner user ID
- file owner group ID
- file last access time

- file last modification time
- file or inode last change time
- file type
- file owner user name
- file owner group name

Parameters
- **path** (*string*) – path to file or directory

Response Headers
- **Content-Type** – application/json

Status Codes
- **200** – OK
- **404** – Not Found
- **500** – Internal Server Error

Example request:

```
GET /rest/latest/attrs/dir1/dir2 HTTP/1.1
Header: "content-type: application/json"
Host: example.com
```

Example responses:

```
HTTP/1.1 200 OK
connection: close
server: Cowboy
date: Sun, 05 Jan 2014 17:17:39 GMT
content-length: 157
Access-Control-Allow-Origin: *
content-type: application/json

{"mode":8,"uid":20000,"gid":20000,"atime":1388937272,"mtime":1388937283,"ctime":13889372
```

9.4.5 /rest/latest/shares

Methods

GET /rest/latest/shares

Retrieve shared files as a list of globally unique identifiers.

Response Headers
- **Content-Type** – application/json

Status Codes
- **200** – OK
- **404** – Not Found
- **500** – Internal Server Error

Example request:

```
GET /rest/latest/files/shares HTTP/1.1
Header: "content-type: application/json"
Host: example.com
```

Example responses:

```
HTTP/1.1 200 OK
connection: close
server: Cowboy
date: Sun, 05 Jan 2014 17:47:00 GMT
content-length: 36
Access-Control-Allow-Origin: *
content-type: application/json

["04ef3c62ea0cdba9cd2ac1a860835efe"]
```

POST /rest/latest/shares

Share existing file. This request adds specified file to a list of shared files.

Parameters

- **path** (*string*) – path to file to be shared

Request Headers

- **Content-Type** – application/json

Response Headers

- **Content-Type** – application/json
- **Location** – redirect link to shared file

Status Codes

- **303** – See Other
- **422** – Unprocessable Entity
- **500** – Internal Server Error

An example curl (http://curl.haxx.se/) request to share file */dir/file.txt* would be:

```
curl -i -k --cert proxy_cert -H "content-type: application/json" -X POST https://example
```

Example request:

```
POST /rest/latest/files/shares HTTP/1.1
Host: example.com
Header: "content-type: application/json"
Data: "dir/file.txt"
```

Example responses:

```
HTTP/1.1 303 See Other
connection: close
server: Cowboy
date: Sun, 05 Jan 2014 18:38:17 GMT
content-length: 0
Access-Control-Allow-Origin: *
content-type: application/json
location: https://example.com/share/04ef3d726a2554f240bb15bf4cfa2a13
```

9.4.6 /rest/latest/shares/(guid)

Methods

GET /rest/latest/shares/ (*guid*)

Retrieve shared file details as a record of structure *{property : value}*.

Fields of returned record:

- shared file path
- shared file download url

Parameters
- **guid** (*string*) – shared files globally unique identifier

Response Headers
- **Content-Type** – application/json

Status Codes
- **200** – OK
- **404** – Not Found
- **500** – Internal Server Error

Example request:

```
GET /rest/latest/files/shares/04ef3c62ea0cdba9cd2ac1a860835efe HTTP/1.1
Header: "content-type: application/json"
Host: example.com
```

Example responses:

```
HTTP/1.1 200 OK
connection: close
server: Cowboy
date: Sun, 05 Jan 2014 17:52:16 GMT
content-length: 108
Access-Control-Allow-Origin: *
content-type: application/json

{"file_path":"dir/file.txt","download_path":"https://example.com/share/04ef3c62ea0cdba9c
```

DELETE /rest/latest/shares/ (*guid*)

Stop sharing existing file. This request removes specified file from a list of shared files.

Parameters
- **guid** (*string*) – shared files globally unique identifier

Response Headers
- **Content-Type** – application/json

Status Codes
- **204** – No Content
- **405** – Method Not Allowed
- **500** – Internal Server Error

Example request:

```
DELETE /rest/latest/files/shares/04ef3c62ea0cdba9cd2ac1a860835efe HTTP/1.1
Header: "content-type: application/json"
Host: example.com
```

Example responses:

```
HTTP/1.1 204 No Content
connection: close
server: Cowboy
date: Sun, 05 Jan 2014 17:58:05 GMT
content-length: 0
Access-Control-Allow-Origin: *
content-type: application/json
```

9.5 Structure of the REST URIs

VeilFS's REST APIs provide access to resources (data entities) via URI paths. To use a REST API, your application will make an HTTP request and parse the response. The VeilFS REST API uses JSON as its communication format, and the standard HTTP methods like GET, PUT, POST and DELETE. URIs for VeilFS's REST API resource have the following structure:

```
http://host:port/rest/api-version/path/to/resource
```

Where *api-version* can be of following:

- number - which stands for an exact API version
- latest - which stands for most recent, stable version

We strongly recommend usage of the latter option especially due to introduction of new features and elimination of bugs in system.

Example URI that would retrieve content of *dir* directory.

```
https://example.com/rest/latest/files/dir
```

For more information see a full list of available *resources*.

CHAPTER 10

C modules

Documents

10.1 GridProxyVerify

```
GridProxyVerify is an library that provides GSI proxy certificates verification based on Globus
```

10.1.1 Files

```
* **grid\_proxy\_verify.c** - This file can be compiled into independent static/dynamic library
    - Insert any peer/peerCA/CA/CRL certificates into current verification context
    - Proceed with verification of certificates added earlier
    - Retrieve verification errors

    Basically it provides simple GSI validation that can be used through very simple API

* **gpv_nif.c** - This file contains Erlang NIF interface for grid\_proxy\_verify library. It ca
    For more info check _cluster\_elements/request\_dispatcher/gpv\_nif.erl_ file or just its @d
```

10.1.2 Prerequisites

```
* libglobus_gsi_callback
* libglobus_common
* libssl

Use this command to install the required dependency packages:

* Debian/Ubuntu Dependencies (.deb packages):

        apt-get install libglobus-gsi-callback-dev

* RHEL/CentOS/Fedora Dependencies (.rpm packages):

        yum install globus-gsi-callback-devel

Additionally if you want to use NIF API, you need to have Erlang NIF libraries and headers. Norm
```

10.1.3 Compilation

The easiest way of compiling this lib is to use **rebar**. Rebar automatically adds required by
Unfortunately you still need to explicate specify _-lglobus_gsi_callback -lglobus_common -lss
your _rebar.config_ file, like in [this](https://github.com/basho/rebar/blob/master/rebar.config

10.2 VeilHelpersNIF

VeilHelpersNIF is an NIF wrapper for VeilHelpers library.

10.2.1 Files

* **veilhelpers_nif.cc** - NIF wrapper for IStorageHelper interface (see VeilHelper source for m

* **term_translator.cc** - Helper functions used to translate erlang terms to c/c++ types

10.2.2 Prerequisites

All you need are Erlang NIF libraries and headers. Normally they are shipped with Erlang.

10.2.3 Compilation

The wrapper should be compiled as a part of cluster using **rebar**, because rebar automatically

CHAPTER 11

Erlang modules

Documents

11.1 veil_cluster_node_app

Authors Michal Wrzeszcz

Copyright This software is released under the *MIT License*.

Descritpion It is the main module of application. It lunches supervisor which then initializes appropriate components of node.

Behaviours application (http://www.erlang.org/doc/man/application.html)

11.1.1 Function Index

- *start/2*
- *stop/1*

11.1.2 Function Details

`start` (*_StartType :: any()*, *_StartArgs :: any()*) → Result

- **Error:** {already_started, pid()} | {shutdown, term()} | term()
- **Result:** {ok, pid()} | ignore | {error, Error}

Starts application by supervisor initialization.

`stop` (*_State :: any()*) → Result

- **Result:** ok

Stops application.

11.2 veil_cluster_node_sup

Authors Michal Wrzeszcz

Copyright This software is released under the *MIT License*.

Descritpion It is the main supervisor. It starts (as it child) node manager which initializes node.

Behaviours supervisor (http://www.erlang.org/doc/man/supervisor.html)

11.2.1 Function Index

- *init/1*
- *start_link/1*

11.2.2 Function Details

init (*Args :: term()*) → Result

- **ChildSpec:** {Id :: term(), StartFunc, RestartPolicy, Type :: worker | supervisor, Modules}
- **Modules:** [module()] | dynamic
- **RestartPolicy:** permanent | transient | temporary
- **RestartStrategy:** one_for_all | one_for_one | rest_for_one | simple_one_for_one
- **Result:** {ok, {SupervisionPolicy, [ChildSpec]}} | ignore
- **StartFunc:** {M :: module(), F :: atom(), A :: [term()] | undefined}
- **SupervisionPolicy:** {RestartStrategy, MaxR :: non_neg_integer(), MaxT :: pos_integer()}

  supervisor:init/1

start_link (*Args :: term()*) → Result

- **Error:** {already_started, pid()} | {shutdown, term()} | term()
- **Result:** {ok, pid()} | ignore | {error, Error}

Starts application supervisor

Submodules

11.3 Cluster elements

Documents

11.3.1 cluster_manager

Authors Michal Wrzeszcz

Copyright This software is released under the *MIT License*.

Descritpion This module coordinates central cluster.

Behaviours gen_server (http://www.erlang.org/doc/man/gen_server.html)

Function Index

Function Details

code_change (*OldVsn*, *State :: term()*, *Extra :: term()*) → Result

- **OldVsn:** Vsn | {down, Vsn}
- **Result:** {ok, NewState :: term()} | {error, Reason :: term()}
- **Vsn:** term()

gen_server:code_change/3

handle_call (*Request :: term()*, *From :: {pid()*, *Tag :: term()}*, *State :: term()*) → Result

- **NewState:** term()
- **Reason:** term()
- **Reply:** term()
- **Result:** {reply, Reply, NewState} | {reply, Reply, NewState, Timeout} | {reply, Reply, NewState, hibernate} | {noreply, NewState} | {noreply, NewState, Timeout} | {noreply, NewState, hibernate} | {stop, Reason, Reply, NewState} | {stop, Reason, NewState}
- **Timeout:** non_neg_integer() | infinity

gen_server:handle_call/3

handle_cast (*Request :: term()*, *State :: term()*) → Result

- **NewState:** term()
- **Result:** {noreply, NewState} | {noreply, NewState, Timeout} | {noreply, NewState, hibernate} | {stop, Reason :: term(), NewState}
- **Timeout:** non_neg_integer() | infinity

gen_server:handle_cast/2

handle_info (*Info :: timeout | term()*, *State :: term()*) → Result

- **NewState:** term()
- **Result:** {noreply, NewState} | {noreply, NewState, Timeout} | {noreply, NewState, hibernate} | {stop, Reason :: term(), NewState}
- **Timeout:** non_neg_integer() | infinity

gen_server:handle_info/2

init(*Args :: term()*) → Result

- **Result:** {ok, State} | {ok, State, Timeout} | {ok, State, hibernate} | {stop, Reason :: term()} | ignore
- **State:** term()
- **Timeout:** non_neg_integer() | infinity

gen_server:init/1

monitoring_loop(*Flag*) → ok

- **Flag:** on | off

Loop that monitors if nodes are alive.

monitoring_loop(*Flag*, *Nodes*) → ok

- **Flag:** on | off
- **Nodes:** list()

Beginning of loop that monitors if nodes are alive.

start_link() → Result

- **Error:** {already_started, Pid} | term()
- **Pid:** pid()
- **Result:** {ok,Pid} | ignore | {error,Error}

Starts cluster manager

start_link(*Mode*) → Result

- **Error:** {already_started,Pid} | term()
- **Mode:** test | normal
- **Pid:** pid()
- **Result:** {ok,Pid} | ignore | {error,Error}

Starts cluster manager

start_monitoring_loop(*Flag*, *Nodes*) → ok

- **Flag:** on | off
- **Nodes:** list()

Starts loop that monitors if nodes are alive.

stop() → ok

Stops the server

terminate(*Reason*, *State :: term()*) -> *Any :: term()*

- **Reason:** normal | shutdown | {shutdown, term()} | term()

gen_server:terminate/2

11.3.2 node_manager

Authors Michal Wrzeszcz

Copyright This software is released under the *MIT License*.

Descritpion This module is a gen_server that coordinates the life cycle of node. It starts/stops appropriate services (according to node type) and communicates with ccm (if node works as worker). Node can be ccm or worker. However, worker_hosts can be also started at ccm nodes.

Behaviours gen_server (http://www.erlang.org/doc/man/gen_server.html)

Function Index

Function Details

addCallback (*State :: term()*, *FuseId :: string()*, *Pid :: pid()*) → NewState

- **NewState:** list()

Adds callback to fuse.

check_vsn () → Result

- **Result:** term()

Checks application version

code_change (*OldVsn*, *State :: term()*, *Extra :: term()*) → Result

- **OldVsn:** Vsn | {down, Vsn}
- **Result:** {ok, NewState :: term()} | {error, Reason :: term()}
- **Vsn:** term()

gen_server:code_change/3

delete_callback (*State :: term()*, *FuseId :: string()*, *Pid :: pid()*) → Result

- **NewState:** term()
- **Result:** {NewState, fuse_not_found | fuse_deleted | pid_not_found | pid_deleted}

Deletes callback

get_callback (*State :: term()*, *FuseId :: string()*) → Result

- **Result:** non | pid()

Gets callback to fuse (if there are more than one callback it chooses one).

handle_call (*Request :: term()*, *From :: {pid()*, *Tag :: term()}*, *State :: term()*) → Result

- **NewState:** term()
- **Reason:** term()
- **Reply:** term()
- **Result:** {reply, Reply, NewState} | {reply, Reply, NewState, Timeout} | {reply, Reply, NewState, hibernate} | {noreply, NewState} | {noreply, NewState, Timeout} | {noreply, NewState, hibernate} | {stop, Reason, Reply, NewState} | {stop, Reason, NewState}
- **Timeout:** non_neg_integer() | infinity

  gen_server:handle_call/3

handle_cast (*Request :: term()*, *State :: term()*) → Result

- **NewState:** term()
- **Result:** {noreply, NewState} | {noreply, NewState, Timeout} | {noreply, NewState, hibernate} | {stop, Reason :: term(), NewState}
- **Timeout:** non_neg_integer() | infinity

  gen_server:handle_cast/2

handle_info (*Info :: timeout | term()*, *State :: term()*) → Result

- **NewState:** term()
- **Result:** {noreply, NewState} | {noreply, NewState, Timeout} | {noreply, NewState, hibernate} | {stop, Reason :: term(), NewState}
- **Timeout:** non_neg_integer() | infinity

  gen_server:handle_info/2

init (*Args :: term()*) → Result

- **Result:** {ok, State} | {ok, State, Timeout} | {ok, State, hibernate} | {stop, Reason :: term()} | ignore
- **State:** term()
- **Timeout:** non_neg_integer() | infinity

  gen_server:init/1

start_link (*Type*) → Result

- **Error:** {already_started,Pid} | term()
- **Pid:** pid()

- **Result:** {ok,Pid} | ignore | {error,Error}
- **Type:** test_worker | worker | ccm

Starts the server

stop() → ok

Stops the server

terminate(*Reason*, *State :: term()) -> Any :: term*()

- **Reason:** normal | shutdown | {shutdown, term()} | term()

  gen_server:terminate/2

11.3.3 worker_host

Authors Michal Wrzeszcz

Copyright This software is released under the *MIT License*.

Descritpion This module hosts all VeilFS modules (fslogic, cluster_rengin etc.). It makes it easy to manage modules and provides some basic functionality for its plug-ins (VeilFS modules) e.g. requests management.

Behaviours gen_server (http://www.erlang.org/doc/man/gen_server.html)

Function Index

Function Details

clear_cache (*Cache :: term()*) → ok

Clears chosen caches at all nodes

code_change (*OldVsn*, *State :: term()*, *Extra :: term()*) → Result

- **OldVsn:** Vsn | {down, Vsn}
- **Result:** {ok, NewState :: term()} | {error, Reason :: term()}
- **Vsn:** term()

gen_server:code_change/3

create_simple_cache (*Name :: atom()*) → Result

- **Result:** ok | error_during_cache_registration

Creates simple cache.

create_simple_cache (*Name :: atom()*, *CacheLoop*, *ClearFun :: term()*) → Result

- **CacheLoop:** integer() | atom()
- **Result:** ok | error_during_cache_registration | loop_time_not_a_number_error

Creates simple cache.

create_simple_cache (*Name :: atom()*, *CacheLoop*, *ClearFun :: term()*, *StrongCacheConnection :: boolean()*) → Result

- **CacheLoop:** integer() | atom()
- **Result:** ok | error_during_cache_registration | loop_time_not_a_number_error

Creates simple cache.

create_simple_cache (*Name :: atom()*, *CacheLoop*, *ClearFun :: term()*, *StrongCacheConnection :: boolean()*, *Pid :: pid()*) → Result

- **CacheLoop:** integer() | atom()
- **Result:** ok | error_during_cache_registration | loop_time_not_a_number_error

Creates simple cache.

generate_sub_proc_list ([*{Name :: atom()*, *MaxDepth :: integer()*, *MaxWidth :: integer()*, *ProcFun :: term()*, *MapFun :: term()}*]) → Result

- **Result:** list()

Generates the list that describes sub procs.

generate_sub_proc_list (*Name :: atom()*, *MaxDepth :: integer()*, *MaxWidth :: integer()*, *ProcFun :: term()*, *MapFun :: term()*) → Result

- **Result:** list()

Generates the list that describes sub procs.

handle_call (*Request :: term()*, *From :: {pid()*, *Tag :: term()}*, *State :: term()*) → Result

- **NewState:** term()
- **Reason:** term()

- **Reply:** term()
- **Result:** {reply, Reply, NewState} | {reply, Reply, NewState, Timeout} | {reply, Reply, NewState, hibernate} | {noreply, NewState} | {noreply, NewState, Timeout} | {noreply, NewState, hibernate} | {stop, Reason, Reply, NewState} | {stop, Reason, NewState}
- **Timeout:** non_neg_integer() | infinity

gen_server:handle_call/3

handle_cast (*Request :: term()*, *State :: term()*) → Result

- **NewState:** term()
- **Result:** {noreply, NewState} | {noreply, NewState, Timeout} | {noreply, NewState, hibernate} | {stop, Reason :: term(), NewState}
- **Timeout:** non_neg_integer() | infinity

gen_server:handle_cast/2

handle_info (*Info :: timeout | term()*, *State :: term()*) → Result

- **NewState:** term()
- **Result:** {noreply, NewState} | {noreply, NewState, Timeout} | {noreply, NewState, hibernate} | {stop, Reason :: term(), NewState}
- **Timeout:** non_neg_integer() | infinity

gen_server:handle_info/2

init (*Args :: term()*) → Result

- **Result:** {ok, State} | {ok, State, Timeout} | {ok, State, hibernate} | {stop, Reason :: term()} | ignore
- **State:** term()
- **Timeout:** non_neg_integer() | infinity

gen_server:init/1

start_link (*PlugIn*, *PlugInArgs*, *LoadMemorySize*) → Result

- **Error:** {already_started,Pid} | term()
- **LoadMemorySize:** integer()
- **Pid:** pid()
- **PlugIn:** atom()
- **PlugInArgs:** any()
- **Result:** {ok,Pid} | ignore | {error,Error}

Starts host with apropriate plug-in

start_sub_proc (*Name :: atom()*, *MaxDepth :: integer()*, *MaxWidth :: integer()*, *ProcFun :: term()*, *MapFun :: term()*) → Result

- **Result:** pid()

Starts sub proc

stop (*PlugIn*) → ok

- **PlugIn:** atom()

Stops the server

terminate(*Reason*, *State :: term()*) -> *Any :: term()*

- **Reason:** normal | shutdown | {shutdown, term()} | term()

gen_server:terminate/2

Submodules

11.3.4 Request dispatcher

Documents

gsi_handler

Authors Rafal Slota

Copyright This software is released under the *MIT License*.

Descritpion This module manages GSI validation

Function Index

- *call/3*
- *find_eec_cert/3*
- *init/0*
- *is_proxy_certificate/1*
- *load_certs/1*
- *proxy_subject/1*
- *update_crls/1*
- *verify_callback/3*

Function Details

call(*Module :: atom()*, *Method :: atom()*, *Args :: [term()]*) -> *ok* | *no_return()*

Calls apply(Module, Method, Args) on one of started slave nodes. If slave node is down, initializes restart procedure and tries to use another node. However is all nodes are down, error is returned and GSI action is interrupted (e.g. peer verification fails).

find_eec_cert(*CurrentOtp :: #'OTPCertificate'{}*, *Chain :: [#'OTPCertificate'{}]*, *IsProxy :: boolean()*) -> *{ok, #'OTPCertificate'{}}* | *no_return()*

For given proxy certificate returns its EEC

init() → ok

Initializes GSI Handler. This method should be called once, before using any other method from this module.

is_proxy_certificate(*OtpCert :: #'OTPCertificate'{}*) -> *boolean()*

Checks is given OTP Certificate has an proxy extension

load_certs (*CADir :: string()) -> ok | no_return()*

Loads all PEM encoded CA certificates from given directory along with their CRL certificates (if any). Note that CRL certificates should also be PEM encoded and the CRL filename should match their CA filename but with '.crl' extension.

proxy_subject (*OtpCert :: #'OTPCertificate'{}*) → {rdnSequence, [#'AttributeTypeAndValue'{}]}

Returns subject of given certificate. If proxy certificate is given, EEC subject is returned.

update_crls (*CADir :: string()) -> ok | no_return()*

Updates CRL certificates based on their distribution point (x509 CA extension). Not yet fully implemented.

verify_callback (*OtpCert :: #'OTPCertificate'{}, Status :: term(), Certs :: [#'OTPCertificate'{}]*) → {valid, UserState :: any()} | {fail, Reason :: term()}

This method is an registered callback, called foreach peer certificate. This callback saves whole certificate chain in GSI ETS based state for further use.

gsi_nif

Authors Rafal Slota

Copyright This software is released under the *MIT License*.

Descritpion This module loads GSI NIF libs

Function Index

- *start/1*
- *verify_cert_c/4*

Function Details

start (*Prefix :: string()*) → ok | {error, Reason :: term()}

This method loads NIF library into erlang VM. This should be used once before using any other method in this module.

verify_cert_c (*PeerCert :: binary(), PeerChain :: [binary()], CAChain :: [binary()], CRLs :: [binary()]*) → {ok, 1} | {ok, 0, Errno :: integer()} | {error, Reason :: term()}

This method validates peer certificate. All input certificates should be DER encoded. {ok, 1} result means that peer is valid, {ok, 0, Errno} means its not. Errno is a raw errno returned by openssl/globus library {error, Reason} on the other hand is returned only when something went horribly wrong during validation.

logger

Authors Lukasz Opiola

Copyright This software is released under the *MIT License*.

Descritpion This module handles log dispatching.

Function Index

Function Details

dispatch_log(*LoglevelAsInt :: integer(), Metadata :: [tuple()], Format :: string(), Args :: string(), IncludeStacktrace :: boolean()*) → ok | {error, lager_not_running}

Logs the log locally (it will be intercepted by central_logging_backend and sent to central sink)

get_console_loglevel() -> *integer*()

Returns current console loglevel

get_current_loglevel() -> *integer*()

Returns current loglevel as set in application's env

get_default_loglevel() -> *integer*()

Returns default loglevel as set in application's env

get_include_stacktrace() -> *boolean*()

Returns get_include_stacktrace env value

loglevel_atom_to_int(*LoglevelAsAtom :: atom()*) -> *integer*()

Returns loglevel number associated with loglevel name

loglevel_int_to_atom(*LoglevelAsInt :: integer()*) -> *atom*()

Returns loglevel name associated with loglevel number

parse_process_info(*ProcessInfo :: tuple()*) → [tuple()]

Changes standard 'process_info' tuple into metadata proplist

set_console_loglevel(*Loglevel :: integer() | atom()*) → ok | {error, badarg}

Changes current console loglevel to desired. Argument can be loglevel as int or atom 'default' atom can be used to set it back to default - default is what is defined in sys.config

set_include_stacktrace (*boolean()*) → ok | {error, badarg}

Changes include_stacktrace env to true or false

set_loglevel (*Loglevel :: integer() | atom()*) → ok | {error, badarg}

Changes current global loglevel to desired. Argument can be loglevel as int or atom 'default' atom can be used to set it back to default

should_log (*LoglevelAsInt :: integer()*) -> *boolean()*

Determines if logs with provided loglevel should be logged or discarded

records_translator

Authors Michal Wrzeszcz

Copyright This software is released under the *MIT License*.

Descritpion This module is able to do additional translation of record decoded using protocol buffer e.g. it can change record "atom" to Erlang atom type.

Function Index

- *translate/2*
- *translate_to_record/1*

Function Details

translate (*Record :: tuple()*, *DecoderName :: string()*) → Result

- **Result:** term()

Translates record to simpler terms if possible.

translate_to_record (*Value :: term()*) → Result

- **Result:** tuple() | term()

Translates term to record if possible.

request_dispatcher

Authors Michal Wrzeszcz

Copyright This software is released under the *MIT License*.

Descritpion This module forwards client's requests to appropriate worker_hosts.

Behaviours gen_server (http://www.erlang.org/doc/man/gen_server.html)

Function Index

Function Details

code_change (*OldVsn*, *State :: term()*, *Extra :: term()*) → Result

- **OldVsn:** Vsn | {down, Vsn}
- **Result:** {ok, NewState :: term()} | {error, Reason :: term()}
- **Vsn:** term()

gen_server:code_change/3

handle_call (*Request :: term()*, *From :: {pid()*, *Tag :: term()}*, *State :: term()*) → Result

- **NewState:** term()
- **Reason:** term()
- **Reply:** term()
- **Result:** {reply, Reply, NewState} | {reply, Reply, NewState, Timeout} | {reply, Reply, NewState, hibernate} | {noreply, NewState} | {noreply, NewState, Timeout} | {noreply, NewState, hibernate} | {stop, Reason, Reply, NewState} | {stop, Reason, NewState}
- **Timeout:** non_neg_integer() | infinity

gen_server:handle_call/3

handle_cast (*Request :: term()*, *State :: term()*) → Result

- **NewState:** term()
- **Result:** {noreply, NewState} | {noreply, NewState, Timeout} | {noreply, NewState, hibernate} | {stop, Reason :: term(), NewState}
- **Timeout:** non_neg_integer() | infinity

gen_server:handle_cast/2

handle_info (*Info :: timeout | term()*, *State :: term()*) → Result

- **NewState:** term()
- **Result:** {noreply, NewState} | {noreply, NewState, Timeout} | {noreply, NewState, hibernate} | {stop, Reason :: term(), NewState}

- **Timeout:** non_neg_integer() | infinity

  gen_server:handle_info/2

init (*Args :: term()*) → Result

- **Result:** {ok, State} | {ok, State, Timeout} | {ok, State, hibernate} | {stop, Reason :: term()} | ignore
- **State:** term()
- **Timeout:** non_neg_integer() | infinity

  gen_server:init/1

send_to_fuse (*FuseId :: string()*, *Message :: term()*, *MessageDecoder :: string()*) → Result

- **Result:** callback_node_not_found | node_manager_error | dispatcher_error | ok | term()

Sends message to fuse

start_link () → Result

- **Error:** {already_started,Pid} | term()
- **Pid:** pid()
- **Result:** {ok,Pid} | ignore | {error,Error}

Starts the server

stop () → ok

Stops the server

terminate (*Reason*, *State :: term()*) -> *Any :: term()*

- **Reason:** normal | shutdown | {shutdown, term()} | term()

  gen_server:terminate/2

ws_handler

Authors Michal Wrzeszcz

Copyright This software is released under the *MIT License*.

Descritpion This module forwards requests from socket to dispatcher.

Function Index

- *init/3*
- *websocket_handle/3*
- *websocket_info/3*
- *websocket_init/3*
- *websocket_terminate/3*

Function Details

init (*Proto :: term()*, *Req :: term()*, *Opts :: term()*) → {upgrade, protocol, cowboy_websocket}

Switches protocol to WebSocket

websocket_handle (*{Type :: atom()*, *Data :: term()}*, *Req*, *State*) → {reply, {Type :: atom(), Data :: term()}, Req, State} | {ok, Req, State} | {shutdown, Req, State}

- **Req:** term()
- **State:** #hander_state{}

Cowboy's webscoket_handle callback. Binary data was received on socket. For more information please refer Cowboy's user manual.

websocket_info (*Msg :: term()*, *Req*, *State*) → {reply, {Type :: atom(), Data :: term()}, Req, State} | {ok, Req, State} | {shutdown, Req, State}

- **Req:** term()
- **State:** #hander_state{}

Cowboy's webscoket_info callback. Erlang message received. For more information please refer Cowboy's user manual.

websocket_init (*TransportName :: atom()*, *Req :: term()*, *Opts :: list()*) → {ok, Req :: term(), State :: term()} | {shutdown, Req :: term()}

Cowboy's webscoket_init callback. Initialize connection, proceed with TLS-GSI authentication. If GSI validation fails, connection will be closed. Currently validation is handled by Globus NIF library loaded on erlang slave nodes.

websocket_terminate (*Reason :: term()*, *Req*, *State*) → ok

- **Req:** term()
- **State:** #hander_state{}

Cowboy's webscoket_info callback. Connection was closed. For more information please refer Cowboy's user manual.

11.4 Veil modules

Documents

11.4.1 cluster_rengine

Authors Michal Wrzeszcz

Copyright This software is released under the *MIT License*.

Descritpion This module implements worker_plugin_behaviour to provide the functionality of rule engine which triggers admin/user defined rules when events appears.

Behaviours *worker_plugin_behaviour*

Function Index

- *cleanup/0*
- *handle/2*
- *init/1*

Function Details

11.4.2 gateway

Authors Michal Wrzeszcz

Copyright This software is released under the *MIT License*.

Descritpion This module implements worker_plugin_behaviour to provide gateway functionality (transfer of files between data centers).

Behaviours *worker_plugin_behaviour*

Function Index

- *cleanup/0*
- *handle/2*
- *init/1*

Function Details

11.4.3 remote_files_manager

Authors Michal Wrzeszcz

Copyright This software is released under the *MIT License*.

Descritpion This module implements worker_plugin_behaviour to provide functionality of remote files manager.

Behaviours *worker_plugin_behaviour*

Function Index

- *cleanup/0*
- *handle/2*
- *init/1*

Function Details

cleanup() → ok

worker_plugin_behaviour callback cleanup/0

handle(*ProtocolVersion :: term()*, *Request :: term()*) → Result

- **Result:** term()

worker_plugin_behaviour callback handle/1. Processes standard worker requests (e.g. ping) and requests from FUSE.

init(*Args :: term()*) -> *list()*

worker_plugin_behaviour callback init/1

11.4.4 rtransfer

Authors Michal Wrzeszcz

Copyright This software is released under the *MIT License*.

Descritpion This module implements worker_plugin_behaviour to provide remote transfer manager functionality (gateways' management).

Behaviours *worker_plugin_behaviour*

Function Index

- *cleanup/0*
- *handle/2*
- *init/1*

Function Details

11.4.5 rule_manager

Authors Michal Wrzeszcz

Copyright This software is released under the *MIT License*.

Descritpion This module implements worker_plugin_behaviour to provide functionality of rule engines manager (update of rules in all types of rule engines).

Behaviours *worker_plugin_behaviour*

Function Index

- *cleanup/0*
- *handle/2*
- *init/1*

Function Details

11.4.6 worker_plugin_behaviour

Authors Michal Wrzeszcz

Copyright This software is released under the *MIT License*.

Descritpion It is the behaviour of each application component. All components defined in subdirectories of veil_modules must implement it.

Function Index

- *behaviour_info/1*

Function Details

behaviour_info(*Arg*) → Result

- **Arg:** callbacks | Other
- **Fun_def:** tuple()
- **Other:** any()
- **Result:** [Fun_def] | undefined

Defines behaviour

Submodules

11.4.7 Central logger

Documents

central_logger

Authors Lukasz Opiola

Copyright This software is released under the *MIT License*.

Descritpion This module implements worker_plugin_behaviour to provide central logging functionalities.

Behaviours *worker_plugin_behaviour*

Function Index

- *cleanup/0*
- *handle/2*
- *init/1*

Function Details

cleanup() → Result

- **Error:** timeout | term()
- **Result:** ok | {error, Error}

worker_plugin_behaviour callback cleanup/0 Reconfigures lager back to standard

handle(*ProtocolVersion :: term()*, *Request*) → Result

- **Error:** term()
- **Message:** string()
- **Metadata:** list()
- **Request:** ping | get_version | {subscribe, Subscriber} | {unsubscribe, Subscriber} | {dispatch_log, Message, Timestamp, Severity, Metadata}
- **Response:** term()
- **Result:** ok | {ok, Response} | {error, Error} | pong | Version
- **Severity:** atom()
- **Subscriber:** pid()
- **Timestamp:** term()
- **Version:** term()

worker_plugin_behaviour callback handle/1

init(*Args :: term()*) → Result

- **Result:** {ok, term()}

worker_plugin_behaviour callback init/1 Sets up the worker for propagating logs to CMT sessions and configures lager trace files.

central_logging_backend

Authors Lukasz Opiola

Copyright This software is released under the *MIT License*.

Descritpion This is a gen_event module (lager backend), responsible for intercepting logs and sending them to central sink via request_dispatcher.

Behaviours gen_event (http://www.erlang.org/doc/man/gen_event.html)

Function Index

- *terminate/2*

Function Details

code_change (*OldVsn*, *State :: term()*, *Extra :: term()*) → Result

- **OldVsn:** Vsn | {down, Vsn}
- **Result:** {ok, NewState :: term()} | {error, Reason :: term()}
- **Vsn:** term()

gen_event:code_change/3

handle_call (*Request :: term()*, *State :: term()*) → Result

- **NewState:** term()
- **Reply:** term()
- **Result:** {ok, Reply, NewState}

gen_event:handle_call/2

handle_event (*Request :: term()*, *State :: term()*) → Result

- **NewState:** term()
- **Result:** {ok, NewState}

gen_event:handle_event/2

handle_info (*Info :: term()*, *State :: term()*) → Result

- **NewState:** term()
- **Result:** {ok, NewState}

gen_event:handle_info/2

init (*Args :: term()*) → Result

- **Result:** {ok, term()}

gen_event callback init/1 Called after installing this handler into lager_event. Returns its loglevel ({mask, 255}) as Status.

terminate (*Reason*, *State :: term()) -> Any :: term()*

- **Reason:** normal | shutdown | {shutdown, term()} | term()

gen_event:terminate/2

11.4.8 Control panel

Documents

control_panel

Authors Lukasz Opiola

Copyright This software is released under the *MIT License*.

Descritpion This module implements worker_plugin_behaviour callbacks. It is responsible for setting up cowboy listener and registering nitrogen_handler as the handler.

Behaviours *worker_plugin_behaviour*

Function Index

- *cleanup/0*
- *handle/2*
- *init/1*

Function Details

cleanup() → Result

- **Error:** timeout | term()
- **Result:** ok | {error, Error}

worker_plugin_behaviour callback cleanup/0 Stops cowboy listener and terminates

handle(*ProtocolVersion :: term()*, *Request*) → Result

- **Error:** term()
- **Request:** ping | get_version
- **Response:** term()
- **Result:** ok | {ok, Response} | {error, Error} | pong | Version
- **Version:** term()

worker_plugin_behaviour callback handle/1

init(*Args :: term()*) → Result

- **Error:** term()
- **Result:** ok | {error, Error}

worker_plugin_behaviour callback init/1 Sets up cowboy dispatch with nitrogen handler and starts cowboy service on desired port.

rest_routes

Authors Lukasz Opiola

Copyright This software is released under the *MIT License*.

Descritpion This module provides mapping of rest subpaths to erlang modules that will end up handling REST requests.

Function Index

- *route/1*

Function Details

route([*binary()*]) → {atom(), binary()}
:noindex:
This function returns handler module and resource ID based on REST request path. The argument is a list of binaries - result of splitting request subpath on "/". Subpath is all that occurs after "" /rest/ /"" in request path. Should return a tuple: - the module that will be called to handle requested REST resource (atom) - resource id or undefined if none was specified (binary or atom (undefined)) or undefined if no module was matched

Submodules

Handlers

Documents

file_transfer_handler

Authors Lukasz Opiola

Copyright This software is released under the *MIT License*.

Descritpion This module processes file download and upload requests. After validating them it conducts streaming to or from a socket or makes nitrogen display a proper error page.

Function Index

- *get_download_buffer_size/0*
- *handle_rest_upload/3*
- *handle_upload_request/2*
- *maybe_handle_request/1*

Function Details

multipart_data(*Req*) → {headers, cowboy:http_headers(), Req} | {body, binary(), Req} | {end_of_part | eof, Req}

- **Req:** req()

Asserts the validity of mutlipart POST request and proceeds with parsing and writing its data to a file at specified path. Returns true for successful upload anf false otherwise.

handle_upload_request(*Req :: req()*, *record()*) → {ok, Params :: [tuple()], Files :: [#uploaded_file{}]} | {error, incorrect_session}

Asserts the validity of mutlipart POST request and proceeds with parsing or returns an error. Returns list of parsed filed values and file body.

maybe_handle_request(*Req :: req()*) → {boolean(), NewReq :: req()}

Intercepts the request if its a user content or shared file download request or doeas nothinf otherwise. Checks its validity and serves the file or redirects to error page. Should be called on a request coming directly to nitrogen_handler.

nagios_handler

Authors Lukasz Opiola

Copyright This software is released under the *MIT License*.

Descritpion This module handles Nagios monitoring requests.

Function Index

- *handle/2*
- *init/3*
- *terminate/3*

Function Details

handle (*term()*, *term()*) → {ok, term(), term()}

Handles a request producing an XML response

init (*any()*, *term()*, *any()*) → {ok, term(), []}

Cowboy handler callback, no state is required

terminate (*term()*, *term()*, *term()*) → ok

Cowboy handler callback, no cleanup needed

nitrogen_handler

Authors Lukasz Opiola

Copyright This software is released under the *MIT License*.

Descritpion This is the callback module for cowboy to handle requests by passing them to the nitrogen engine

Function Index

- *handle/2*
- *init/3*
- *terminate/3*

Function Details

handle (*Request*, *Options*) → Result

- **Options:** term()
- **Request:** term()
- **Response:** term()
- **Result:** {ok, Response, Options}

Handles a request producing a response with use of Nitrogen engine or a file stream response

init (*Protocol*, *Request :: term()*, *Options :: term()*) → Result

- **Protocol:** {Transport :: term(), http}
- **Result:** {ok, Request :: term(), #state{}}

Initializes a request-response procedure

terminate(*Reason*, *Request*, *State*) → Result

- **Reason:** term()
- **Request:** term()
- **Result:** ok
- **State:** term()

Cowboy handler callback

rest_handler

Authors Lukasz Opiola

Copyright This software is released under the *MIT License*.

Descritpion This is a cowboy handler module, implementing cowboy_rest interface.
It handles REST requests by routing them to proper rest module.

Function Index

- *allowed_methods/2*
- *content_types_accepted/2*
- *content_types_provided/2*
- *delete_resource/2*
- *get_resource/2*
- *handle_json_data/2*
- *handle_multipart_data/2*
- *handle_urlencoded_data/2*
- *init/3*
- *resource_exists/2*
- *rest_init/2*

Function Details

allowed_methods(*req()*, *#state{}*) → {[binary()], req(), #state{}}

Cowboy callback function Returns methods that are allowed for request URL. Will call allowed_methods/2 from rest_module_behaviour.

content_types_accepted(*req()*, *#state{}*) → {term(), req(), #state{}}

Cowboy callback function Returns content-types that are accepted by REST handler and what functions should be used to process the requests.

content_types_provided(*req()*, *#state{}*) → {[binary()], req(), #state{}}

Cowboy callback function Returns content types that can be provided for the request. Will call content_types_provided/1|2 from rest_module_behaviour.

delete_resource(*req()*, *#state{}*) → {term(), req(), #state{}}

Cowboy callback function Handles DELETE requests. Will call delete/2 from rest_module_behaviour.

get_resource(*req()*, *#state{}*) → {term(), req(), #state{}}

Cowboy callback function Handles GET requests. Will call get/1|2 from rest_module_behaviour.

handle_json_data(*req()*, *#state{}*) → {boolean(), req(), #state{}}

Function handling "application/json" requests.

handle_multipart_data(*req()*, *#state{}*) → {boolean(), req(), #state{}}

Function handling "multipart/form-data" requests.

handle_urlencoded_data(*req()*, *#state{}*) → {boolean(), req(), #state{}}

Function handling "application/x-www-form-urlencoded" requests.

init(*any()*, *any()*, *any()*) → {upgrade, protocol, cowboy_rest}

Cowboy callback function Imposes a cowboy upgrade protocol to cowboy_rest - this module is now treated as REST module by cowboy.

resource_exists(*req()*, *#state{}*) → {boolean(), req(), #state{}}

Cowboy callback function Determines if resource identified by URL exists. Will call exists/2 from rest_module_behaviour.

rest_init(*req()*, *term()*) → {ok, req(), term()} | {shutdown, req()}

Cowboy callback function Called right after protocol upgrade to init the request context. Will shut down the connection if the peer doesn't provide a valid proxy certificate.

veil_multipart_bridge

Authors Lukasz Opiola

Copyright This software is released under the *MIT License*.

Descritpion This is a custom multipart bridge. It checks if a request is a multipart POST requests, checks its validity and passes control to file_transfer_handler.

Function Index

- *parse/1*

Function Details

parse(*ReqBridge :: record()*) → {ok, not_multipart} | {ok, Params :: [tuple()], Files :: [#uploaded_file{}]} | {error, any()}

Try to parse the request encapsulated in request bridge if it is a multipart POST request.

Rest modules

Documents

rest_attrs

Authors Lukasz Opiola

Copyright This software is released under the *MIT License*.

Descritpion This module implements rest_module_behaviour and handles all REST requests directed at /rest/attrs/(path). It returns file attributes, if possible.

Behaviours *rest_module_behaviour*

Function Index

Function Details

allowed_methods(*req()*, *binary()*, *binary()*) → {[binary()], req()}

Should return list of methods that are allowed and directed at specific Id. e.g.: if Id =:= undefined -> '[< >, < >]' if Id /= undefined -> '[< >, < >, < >]'

content_types_provided(*req()*, *binary()*) → {[binary()], req()}

Should return list of provided content-types without specified ID (e.g. ".../rest/resource/"). Should take into account different types of methods (PUT, GET etc.), if needed. Should return empty list if method is not supported.

content_types_provided(*req()*, *binary()*, *binary()*) → {[binary()], req()}

Should return list of provided content-types with specified ID (e.g. ".../rest/resource/some_id"). Should take into account different types of methods (PUT, GET etc.), if needed. Should return empty list if method is not supported.

delete(*req()*, *binary()*, *binary()*) → {boolean(), req()}

Will be called for DELETE request on given ID. Should try to remove specified resource and return true/false indicating the result. Should always return false if the method is not supported.

exists(*req()*, *binary()*, *binary()*) → {boolean(), req()}

Should return whether resource specified by given ID exists. Will be called for GET, PUT and DELETE when ID is contained in the URL.

get (*req()*, *binary()*) → {term() | {stream, integer(), function()} | halt, req(), req()}

Will be called for GET request without specified ID (e.g. ".../rest/resource/"). Should return one of the following: 1. ResponseBody, of the same type as content_types_provided/1 returned for this request 2. Cowboy type stream function, serving content of the same type as content_types_provided/1 returned for this request 3. 'halt' atom if method is not supported

get (*req()*, *binary()*, *binary()*) → {term() | {stream, integer(), function()} | halt, req(), req()}

Will be called for GET request with specified ID (e.g. ".../rest/resource/some_id"). Should return one of the following: 1. ResponseBody, of the same type as content_types_provided/2 returned for this request 2. Cowboy type stream function, serving content of the same type as content_types_provided/2 returned for this request 3. 'halt' atom if method is not supported

post (*req()*, *binary()*, *binary()*, *term()*) → {boolean() | {true, binary()}, req()}

Will be called for POST request, after the request has been validated. Should handle the request and return true/false indicating the result. Should always return false if the method is not supported. Returning {true, URL} will cause the reply to contain 201 redirect to given URL.

put (*req()*, *binary()*, *binary()*, *term()*) → {boolean(), req()}

Will be called for PUT request on given ID, after the request has been validated. Should handle the request and return true/false indicating the result. Should always return false if the method is not supported.

validate (*req()*, *binary()*, *binary()*, *term()*) → {boolean(), req()}

Should return true/false depending on whether the request is valid in terms of the handling module. Will be called before POST or PUT, should discard unprocessable requests.

rest_files

Authors Lukasz Opiola

Copyright This software is released under the *MIT License*.

Descritpion This module implements rest_module_behaviour and handles all REST requests directed at /rest/files/(path). Essentially, it serves user content (files) via HTTP.

Behaviours *rest_module_behaviour*

Function Index

- *handle_multipart_data/4*
- *post/4*
- *put/4*
- *validate/4*

Function Details

allowed_methods (*req()*, *binary()*, *binary()*) → {[binary()], req()}

Should return list of methods that are allowed and directed at specific Id. e.g.: if Id =:= undefined -> ''[<>, <>]" if Id /= undefined -> ''[<>, <>, <>]''

content_types_provided (*req()*, *binary()*) → {[binary()], req()}

Should return list of provided content-types without specified ID (e.g. ".../rest/resource/"). Should take into account different types of methods (PUT, GET etc.), if needed. Should return empty list if method is not supported. If there is no id, only dirs can be listed -> application/json.

content_types_provided (*req()*, *binary()*, *binary()*) → {[binary()], req()}

Should return list of provided content-types with specified ID (e.g. ".../rest/resource/some_id"). Should take into account different types of methods (PUT, GET etc.), if needed. Should return empty list if method is not supported. Id is a dir -> application/json Id is a regular file -> ' ' Id does not exist -> []

delete (*req()*, *binary()*, *binary()*) → {boolean(), req()}

Will be called for DELETE request on given ID. Should try to remove specified resource and return true/false indicating the result. Should always return false if the method is not supported.

exists (*req()*, *binary()*, *binary()*) → {boolean(), req()}

Should return whether resource specified by given ID exists. Will be called for GET, PUT and DELETE when ID is contained in the URL.

get (*req()*, *binary()*) → {term() | {stream, integer(), function()} | halt, req(), req()}

Will be called for GET request without specified ID (e.g. ".../rest/resource/"). Should return one of the following: 1. ResponseBody, of the same type as content_types_provided/1 returned for this request 2. Cowboy type stream function, serving content of the same type as content_types_provided/1 returned for this request 3. 'halt' atom if method is not supported

get (*req()*, *binary()*, *binary()*) → {term() | {stream, integer(), function()} | halt, req(), req()}

Will be called for GET request with specified ID (e.g. ".../rest/resource/some_id"). Should return one of the following: 1. ResponseBody, of the same type as content_types_provided/2 returned for this request 2. Cowboy type stream function, serving content of the same type as content_types_provided/2 returned for this request 3. 'halt' atom if method is not supported

handle_multipart_data (*req()*, *binary()*, *binary()*, *term()*) → {boolean(), req()}

Optional callback to handle multipart requests. Data should be streamed in handling module with use of cowboy_multipart module. Method can be '<> or <>'. Should handle the request and return true/false indicating the result. Should always return false if the method is not supported.

post (*req()*, *binary()*, *binary()*, *term()*) → {boolean() | {true, binary()}, req()}

Will be called for POST request, after the request has been validated. Should handle the request and return true/false indicating the result. Should always return false if the method is not supported. Returning {true, URL} will cause the reply to contain 201 redirect to given URL.

put (*req()*, *binary()*, *binary()*, *term()*) → {boolean(), req()}

Will be called for PUT request on given ID, after the request has been validated. Should handle the request and return true/false indicating the result. Should always return false if the method is not supported.

validate (*req()*, *binary()*, *binary()*, *term()*) → {boolean(), req()}

Should return true/false depending on whether the request is valid in terms of the handling module. Will be called before POST or PUT, should discard unprocessable requests.

rest_module_behaviour

Authors Lukasz Opiola

Copyright This software is released under the *MIT License*.

Descritpion This behaviour should be implemented by modules handling REST requests. It ensures the presence of required callbacks.

Function Index

- *behaviour_info/1*

Function Details

behaviour_info (*Arg*) → Result

- **Arg:** callbacks | Other
- **Fun_def:** tuple()
- **Other:** any()
- **Result:** [Fun_def] | undefined

Defines the behaviour (lists the callbacks and their arity)

rest_shares

Authors Lukasz Opiola

Copyright This software is released under the *MIT License*.

Descritpion This module implements rest_module_behaviour and handles all REST requests directed at /rest/shares/(path) (creating and deleting shares, listing shares and retrieving share details).

Behaviours *rest_module_behaviour*

Function Index

Function Details

allowed_methods(*req()*, *binary()*, *binary()*) → {[binary()], req()}

Should return list of methods that are allowed and directed at specific Id. e.g.: if Id =:= undefined -> '[< >, < >]' if Id /= undefined -> '[< >, < >, < >]'

content_types_provided(*req()*, *binary()*) → {[binary()], req()}

Should return list of provided content-types without specified ID (e.g. ".../rest/resource/"). Should take into account different types of methods (PUT, GET etc.), if needed. Should return empty list if method is not supported.

content_types_provided(*req()*, *binary()*, *binary()*) → {[binary()], req()}

Should return list of provided content-types with specified ID (e.g. ".../rest/resource/some_id"). Should take into account different types of methods (PUT, GET etc.), if needed. Should return empty list if method is not supported.

delete(*req()*, *binary()*, *binary()*) → {boolean(), req()}

Will be called for DELETE request on given ID. Should try to remove specified resource and return true/false indicating the result. Should always return false if the method is not supported.

exists(*req()*, *binary()*, *binary()*) → {boolean(), req()}

Should return whether resource specified by given ID exists. Will be called for GET, PUT and DELETE when ID is contained in the URL.

get(*req()*, *binary()*) → {term() | {stream, integer(), function()} | halt, req(), req()}

Will be called for GET request without specified ID (e.g. ".../rest/resource/"). Should return one of the following: 1. ResponseBody, of the same type as content_types_provided/1 returned for this request 2. Cowboy type stream function, serving content of the same type as content_types_provided/1 returned for this request 3. 'halt' atom if method is not supported

get(*req()*, *binary()*, *binary()*) → {term() | {stream, integer(), function()} | halt, req(), req()}

Will be called for GET request with specified ID (e.g. ".../rest/resource/some_id"). Should return one of the following: 1. ResponseBody, of the same type as content_types_provided/2

returned for this request 2. Cowboy type stream function, serving content of the same type as content_types_provided/2 returned for this request 3. 'halt' atom if method is not supported

post (*req()*, *binary()*, *binary()*, *term()*) → {boolean() | {true, binary()}, req()}

Will be called for POST request, after the request has been validated. Should handle the request and return true/false indicating the result. Should always return false if the method is not supported. Returning {true, URL} will cause the reply to contain 201 redirect to given URL.

put (*req()*, *binary()*, *binary()*, *term()*) → {boolean(), req()}

Will be called for PUT request on given ID, after the request has been validated. Should handle the request and return true/false indicating the result. Should always return false if the method is not supported.

validate (*req()*, *binary()*, *binary()*, *term()*) → {boolean(), req()}

Should return true/false depending on whether the request is valid in terms of the handling module. Will be called before POST or PUT, should discard unprocessable requests. No need to check if file exists as the same will be done in post method

Utils

Documents

openid_utils

Authors Lukasz Opiola

Copyright This software is released under the *MIT License*.

Descritpion This is a simple library used to establish an OpenID authentication. It needs nitrogen and ibrowse to run.

Function Index

- *get_login_url/2*
- *nitrogen_prepare_validation_parameters/0*
- *nitrogen_retrieve_user_info/0*
- *validate_openid_login/1*

Function Details

get_login_url (*HostName :: string()*, *RedirectParams :: string()*) → Result

- **Result:** ok | {error, endpoint_unavailable}

  Produces an URL with proper GET parameters, used to redirect the user to OpenID Provider login page. RedirectParams are parameters concatenated to return_to field.

nitrogen_prepare_validation_parameters () → Result

- **Error:** invalid_request
- **Result:** {string(), string()} | {error, Error}

This function retrieves endpoint URL and parameters from redirection URL created by OpenID provider. They are later used as arguments to validate_openid_login() function. Must be called from within nitrogen page context to work, precisely from openid redirection page.

nitrogen_retrieve_user_info() → Result

- **Error:** invalid_request
- **Result:** {ok, list()} | {error, Error}

This function retrieves user info from parameters of redirection URL created by OpenID provider. They are returned as a proplist and later used to authenticate a user in the system. Must be called from within nitrogen page context to work, precisely from openid redirection page.

rest_utils

Authors Lukasz Opiola

Copyright This software is released under the *MIT License*.

Descritpion This module provides convinience functions designed for REST handling modules.

Function Index

- *decode_from_json/1*
- *encode_to_json/1*
- *map/2*
- *unmap/3*

Function Details

decode_from_json(*term()) -> binary()*

Convinience function that convert JSON binary to an erlang term.

encode_to_json(*term()) -> binary()*

Convinience function that convert an erlang term to JSON, producing binary result. Possible terms, can be nested: {struct, Props} - Props is a structure as a proplist, e.g.: [{id, 13}, {message, “mess”}] {Props} - alias for above {array, Array} - Array is a list, e.g.: [13, “mess”]

map(*record()*[, *atom()*]) → [tuple()]

Converts a record to JSON conversion-ready tuple list. For this input: RecordToMap = #some_record{id=123, message=”mess”} Fields = [id, message] The function will produce: [{id, 123},{message, “mess”}] NOTE: rest_utils.hrl contains convienience macro that will include this function in code with Fields listed for given record name

unmap([*tuple()*], *record()*[, *atom()*]) → [tuple()]

Converts a tuple list resulting from JSON to erlang translation into an erlang record. Reverse process to map/2. For this input: Proplist = [{id, 123},{message, “mess”}] RecordTuple = #some_record{} Fields = [id, message] The function will produce: #some_record{id=123,

message="mess"} NOTE: rest_utils.hrl contains convienience macro that will include this function in code with Fields listed for given record name

Web gui

Documents

gui_utils

Authors Lukasz Opiola

Copyright This software is released under the *MIT License*.

Descritpion This file contains useful functions commonly used in control_panel modules.

Function Index

- *apply_or_redirect/3*
- *apply_or_redirect/4*
- *can_view_logs/0*
- *dn_and_storage_defined/0*
- *empty_page/0*
- *get_requested_hostname/0*
- *get_user_dn/0*
- *logotype_footer/1*
- *storage_defined/0*
- *top_menu/1*
- *top_menu/2*
- *user_logged_in/0*

Function Details

apply_or_redirect (*Module :: atom*, *Fun :: atom*, *boolean()) -> boolean()*

Checks if the client has right to do the operation (is logged in and possibly has a certificate DN defined). If so, it executes the code.

apply_or_redirect (*Module :: atom*, *Fun :: atom*, *Args :: list()*, *boolean()) -> boolean()*

Checks if the client has right to do the operation (is logged in and possibly has a certificate DN defined). If so, it executes the code.

can_view_logs () -> *boolean()*

Determines if current user is allowed to view cluster logs.

dn_and_storage_defined () -> *boolean()*

Convienience function to check both conditions.

get_requested_hostname() -> *string*()

Returns the hostname requested by the client.

get_user_dn() -> *string*()

Returns user's DN retrieved from his session state.

logotype_footer(*MarginTop :: integer()*) -> *list*()

Convienience function to render logotype footer, coming after page content.

storage_defined() -> *boolean*()

Checks if any storage is defined in the database.

top_menu(*ActiveTabID :: any()*) -> *list*()

Convienience function to render top menu in GUI pages. Item with ActiveTabID will be highlighted as active.

top_menu(*ActiveTabID :: any()*, *SubMenuBody :: any()*) -> *list*()

Convienience function to render top menu in GUI pages. Item with ActiveTabID will be highlighted as active. Submenu body (list of nitrogen elements) will be concatenated below the main menu.

user_logged_in() -> *boolean*()

Checks if the client has a valid login session.

page_about

Authors Krzysztof Trzepla

Copyright This software is released under the *MIT License*.

Descritpion This file contains Nitrogen website code

page_contact_support

Authors Lukasz Opiola

Copyright This software is released under the *MIT License*.

Descritpion This file contains Nitrogen website code

page_error

Authors Lukasz Opiola

Copyright This software is released under the *MIT License*.

Descritpion This file contains Nitrogen website code

page_events

Authors Lukasz Opiola

Copyright This software is released under the *MIT License*.

Descritpion This file contains Nitrogen website code

page_file_manager

Authors Lukasz Opiola

Copyright This software is released under the *MIT License*.

Descritpion This file contains Nitrogen website code

page_index

Authors Lukasz Opiola

Copyright This software is released under the *MIT License*.

Descritpion This file contains Nitrogen website code

page_login

Authors Lukasz Opiola

Copyright This software is released under the *MIT License*.

Descritpion This file contains Nitrogen website code

page_logout

Authors Lukasz Opiola

Copyright This software is released under the *MIT License*.

Descritpion This file contains Nitrogen website code

page_logs

Authors Lukasz Opiola

Copyright This software is released under the *MIT License*.

Descritpion This file contains Nitrogen website code

page_manage_account

Authors Lukasz Opiola

Copyright This software is released under the *MIT License*.

Descritpion This file contains Nitrogen website code

page_rules_composer

Authors Lukasz Opiola

Copyright This software is released under the *MIT License*.

Descritpion This file contains Nitrogen website code

page_rules_simulator

Authors Lukasz Opiola

Copyright This software is released under the *MIT License*.

Descritpion This file contains Nitrogen website code

page_rules_viewer

Authors Lukasz Opiola

Copyright This software is released under the *MIT License*.

Descritpion This file contains Nitrogen website code

page_shared_files

Authors Lukasz Opiola

Copyright This software is released under the *MIT License*.

Descritpion This file contains Nitrogen website code

page_system_state

Authors Lukasz Opiola

Copyright This software is released under the *MIT License*.

Descritpion This file contains Nitrogen website code

page_validate_login

Authors Lukasz Opiola

Copyright This software is released under the *MIT License*.

Descritpion This file contains Nitrogen website code

web_404

Authors Lukasz Opiola

Copyright This software is released under the *MIT License*.

Descritpion This file contains Nitrogen website code

Submodules

Elements

Documents

bootstrap_button

Authors Lukasz Opiola

Copyright This software is released under the *MIT License*.

Descritpion This file contains element definition for bootstrap-compliant button

Function Index

- *reflect/0*
- *render_element/1*

Function Details

bootstrap_checkbox

Authors Lukasz Opiola

Copyright This software is released under the *MIT License*.

Descritpion This file contains element definition for bootstrap-compliant checkbox

Function Index

- *reflect/0*
- *render_element/1*

Function Details

form

Authors Lukasz Opiola

Copyright This software is released under the *MIT License*.

Descritpion This file contains element definition for simplest HTML form

Function Index

- *reflect/0*
- *render_element/1*

Function Details

veil_upload

Authors Lukasz Opiola

Copyright This software is released under the *MIT License*.

Descritpion This file contains element definition for advanced file upload form

Function Index

- *event/1*
- *reflect/0*
- *render_element/1*

Function Details

11.4.9 Dao

Documents

dao

Authors Rafal Slota

Copyright This software is released under the *MIT License*.

Descritpion This module implements *worker_plugin_behaviour* callbacks and contains utility API methods. DAO API functions are implemented in DAO submodules like: *dao_cluster*, *dao_vfs*. All DAO API functions Should not be used directly, use *dao:handle/2* instead. Module :: atom() is module suffix (prefix is 'dao'), MethodName :: atom() is the method name and ListOfArgs :: [term()] is list of argument for the method. If you want to call utility methods from this module - use Module = utils See *dao:handle/2* for more details.

Behaviours *worker_plugin_behaviour*

Function Index

- *cleanup/0*
- *doc_to_term/1*
- *get_record/1*
- *handle/2*
- *init/1*
- *init_storage/0*

- *list_records/2*
- *load_view_def/2*
- *remove_record/1*
- *save_record/1*
- *set_db/1*

Function Details

cleanup() → Result

- **Error:** timeout | term()
- **Result:** ok | {error, Error}

worker_plugin_behaviour callback cleanup/0

doc_to_term(*Field :: term()) -> term()*)

Converts given valid BigCouch document body into erlang term(). If document contains saved record which is a valid record (see is_valid_record/1), then structure of the returned record will be updated

get_record(*Id :: atom() | string()) -> {ok, #veil_document{record :: tuple()}} | {error, Error :: term()} | no_return()*)

Retrieves record with UUID = Id from DB. Returns whole #veil_document record containing UUID, Revision Info and demanded record inside. #veil_document{}.uuid and #veil_document{}.rev_info should not be ever changed. You can strip wrappers if you do not need them using API functions of dao_lib module. See #veil_document{} structure for more info. Should not be used directly, use *dao:handle/2* instead.

handle(*ProtocolVersion :: term(), Request*) → Result

- **Args:** list()
- **Error:** term()
- **Method:** atom()
- **Request:** {Method, Args} | {Mod :: atom(), Method, Args} | ping | get_version
- **Response:** term()
- **Result:** ok | {ok, Response} | {error, Error} | pong | Version
- **Version:** term()

worker_plugin_behaviour callback handle/1. All {Module, Method, Args} requests (second argument), executes Method with Args in {@type ({@type) dao_Module} module, but with one exception: If Module = utils, then dao module will be used. E.g calling dao:handle(_, {vfs, some_method, [some_arg]}) will call dao_vfs:some_method(some_arg) but calling dao:handle(_, {utils, some_method, [some_arg]}) will call dao:some_method(some_arg) You can omit Module atom in order to use default module which is dao_cluster. E.g calling dao:handle(_, {some_method, [some_arg]}) will call dao_cluster:some_method(some_arg) Additionally all exceptions from called API method will be caught and converted into {error, Exception} tuple. E.g. calling handle(_, {save_record, [Id, Rec]}) will execute dao_cluster:save_record(Id, Rec) and normalize return value.

init(*Args :: term()*) → Result

- **Error:** term()
- **Result:** ok | {error, Error}

worker_plugin_behaviour callback init/1

init_storage() → ok | {error, Error :: term()}

Inserts storage defined during worker instalation to database (if db already has defined storage, the function only replaces StorageConfigFile with that definition)

list_records(*ViewInfo :: #view_info{}*, *QueryArgs :: #view_query_args{}*) → {ok, QueryResult :: #view_result{}} | {error, term()}

Executes view query and parses returned result into #view_result{} record. Strings from #view_query_args{} are not transformed by *dao_helper:name/1*, the caller has to do it by himself.

load_view_def(*Name :: string()*, *Type :: map | reduce*) -> *string()*

Loads view definition from file.

remove_record(*Id :: atom() | uuid()*) → ok | {error, Error :: term()}

Removes record with given UUID from DB Should not be used directly, use *dao:handle/2* instead.

save_record(*term() | #veil_document{uuid :: string(), rev_info :: term(), record :: term(), force_update :: boolean()}*) -> *{ok, DocId :: string()} | {error, conflict} | no_return()*

Saves record to DB. Argument has to be either Record :: term() which will be saved with random UUID as completely new document or #veil_document record. If #veil_document record is passed caller may set UUID and revision info in order to update this record in DB. If you got #veil_document{} via *dao:get_record/1*, uuid and rev_info are in place and you shouldn't touch them Should not be used directly, use *dao:handle/2* instead.

set_db(*DbName :: string()*) → ok

Sets current working database name

dao_cluster

Authors Rafal Slota

Copyright This software is released under the *MIT License*.

Descritpion This module gives high level DB API which contain veil cluster specific utility methods. All DAO API functions should not be called directly. Call dao:handle(_, {cluster, MethodName, ListOfArgs) instead. See *dao:handle/2* for more details.

Function Index

- *clear_sessions/0*
- *clear_state/0*
- *clear_state/1*
- *close_connection/1*
- *close_fuse_session/1*

Function Details

clear_sessions () -> *ok* | *no_return*()

Cleanups old, unused sessions from DB. Each session which is expired is checked. If there is at least one active connection for the session, its expire time will be extended. Otherwise it will be deleted. Should not be used directly, use *dao:handle/2* instead.

clear_state () -> *ok* | *no_return*()

Removes cluster state with Id = cluster_state Should not be used directly, use *dao:handle/2* instead.

clear_state (*Id :: atom()*) -> *ok* | *no_return*()

Removes cluster state with given Id Should not be used directly, use *dao:handle/2* instead.

close_connection (*SessID :: uuid()*) -> *ok* | *no_return*()

Removes connection_info record with given SessID form DB and tries to close it. This method should not be used unless connection exists. Otherwise it will fail with exception error. Should not be used directly, use *dao:handle/2* instead.

close_fuse_session (*FuseId :: uuid()*) -> *ok* | *no_return*()

Removes fuse_session record with given FuseID form DB and cache. Also deletes all connections that belongs to this session and tries to close them. Should not be used directly, use *dao:handle/2* instead.

get_connection_info (*SessID :: uuid()*) -> *{ok, #veil_document{}}* | *no_return*()

Gets connection_info record with given SessID form DB. Should not be used directly, use *dao:handle/2* instead.

get_fuse_session (*FuseId :: uuid(), {stale, update_before | ok})* -> *{ok, #veil_document{}}* | *no_return*()

Gets fuse_session record with given FuseID form DB. Second argument shall be either {stale, update_before} (will update cache before getting value) or {stale, ok} which is default - get value from cache. Default behaviour can be also achieved by ommiting second argument. Should not be used directly, use *dao:handle/2* instead.

get_state() → {ok, term()} | {error, any()}

Retrieves cluster state with ID = cluster_state from DB. Should not be used directly, use *dao:handle/2* instead.

get_state(*Id :: atom()*) → {ok, term()} | {error, any()}

Retrieves cluster state with UUID = Id from DB. Should not be used directly, use *dao:handle/2* instead.

list_connection_info(*{by_session_id, SessID :: uuid()}*) → {ok, [#veil_document{}]} | {error, any()}

Lists connection_info records using given select condition. Current implementeation supports fallowing selects: {by_session_id, SessID} - select all connections that belongs to session with ID - SessID Should not be used directly, use *dao:handle/2* instead.

list_fuse_sessions(*{by_valid_to, Time :: non_neg_integer()}*) → {ok, [#veil_document{}]} | {error, any()}

Lists fuse_session records using given select condition. Current implementeation supports fallowing selects: {by_valid_to, Time} - select all records whose 'valid_to' field is >= Time Should not be used directly, use *dao:handle/2* instead.

remove_connection_info(*SessID :: uuid()) -> ok | no_return*()

Removes connection_info record with given SessID form DB. Should not be used directly, use *dao:handle/2* instead.

remove_fuse_session(*FuseId :: uuid()) -> ok | no_return*()

Removes fuse_session record with given FuseID form DB and cache. Should not be used directly, use *dao:handle/2* instead.

save_connection_info(*#connection_info{} | #veil_document{}) -> {ok, Id :: string()} | no_return*()

Saves connection_info record to DB. Should not be used directly, use *dao:handle/2* instead.

save_fuse_session(*#fuse_session{} | #veil_document{}) -> {ok, Id :: string()} | no_return*()

Saves fuse_session record to DB. If #fuse_session.valid_to field is not valid (i.e. its value is less then current timestamp) it will be set to default value (specified in application config). Should not be used directly, use *dao:handle/2* instead.

save_state(*Rec :: tuple()) -> {ok, Id :: string()} | no_return*()

Saves cluster state Rec to DB with ID = cluster_state. Should not be used directly, use *dao:handle/2* instead.

save_state(*Id :: atom(), Rec :: tuple()) -> {ok, Id :: string()} | no_return*()

Saves cluster state Rec to DB with ID = Id. Should not be used directly, use *dao:handle/2* instead.

dao_helper

Authors Rafal Slota

Copyright This software is released under the *MIT License*.

Descritpion Low level BigCouch DB API

Function Index

Function Details

create_db (*DbName :: string()*) → ok | {error, term()}

Creates db named DbName. If db already does nothing and returns 'ok'

create_db (*DbName :: string(), Opts :: [Option]*) → ok | {error, term()}

- **Option:** atom() | {atom(), term()}

Creates db named DbName with Opts. Options can include values for q and n, for example {q, "8"} and {n, "3"}, which control how many shards to split a database into and how many nodes each doc is copied to respectively.

create_view(*DbName :: string()*, *DesignName :: string()*, *ViewName :: string()*, *Map :: string()*, *Reduce :: string()*, *DesignVersion :: integer()*) → [ok | {error, term()}]

Creates view with given Map and Reduce function. When Reduce = "", reduce function won't be created

delete_db(*DbName :: string()*) → ok | {error, database_does_not_exist} | {error, term()}

Deletes db named DbName

delete_db(*DbName :: string(), Opts :: [Option]*) → ok | {error, database_does_not_exist} | {error, term()}

- **Option:** atom() | {atom(), term()}

Deletes db named DbName

delete_doc(*DbName :: string()*, *DocID :: string()*) → ok | {error, missing} | {error, deleted} | {error, term()}

Deletes doc from db

delete_docs(*DbName :: string()*[, *DocID :: string()*]) → [ok | {error, term()}]

Deletes list of docs from db

gen_uuid(*) -> string()*

Generates UUID with CouchDBs 'utc_random' algorithm

get_db_info(*DbName :: string()*) → {ok, [{instance_start_time, binary()} | {doc_count, non_neg_integer()} | {doc_del_count, non_neg_integer()} | {purge_seq, non_neg_integer()} | {compact_running, boolean()} | {disk_size, non_neg_integer()} | {disk_format_version, pos_integer()}]} | {error, database_does_not_exist} | {error, term()}

Returns db info for the given DbName

get_doc_count(*DbName :: string()*) → {ok, non_neg_integer()} | {error, database_does_not_exist} | {error, term()}

Returns doc count for given DbName

insert_doc(*DbName :: string()*, *Doc :: #doc{}*) → {ok, {RevNum :: non_neg_integer(), RevBin :: binary()}} | {error, conflict} | {error, term()}

Inserts doc to db

insert_doc(*DbName :: string(), Doc :: #doc{}, Opts :: [Option]*) → {ok, {RevNum :: non_neg_integer(), RevBin :: binary()}} | {error, conflict} | {error, term()}

- **Option:** atom() | {atom(), term()}

Inserts doc to db

insert_docs(*DbName :: string()*[, *Doc :: #doc{}*]) → {ok, term()} | {error, term()}

Inserts list of docs to db

insert_docs(*DbName :: string(), [Doc :: #doc{}], Opts :: [Option]*) → {ok, term()} | {error, term()}

- **Option:** atom() | {atom(), term()}

Inserts list of docs to db

list_dbs() → {ok, [string()]} | {error, term()}

Lists all dbs

list_dbs(*Prefix :: string()*) → {ok, [string()]} | {error, term()}

Lists all dbs that starts with Prefix

name(*Name :: string() | atom()) -> binary()*

Converts string/atom to binary

open_design_doc(*DbName :: string()*, *DesignName :: string()*) → {ok, #doc{}} | {error, {not_found, missing | deleted}} | {error, term()}

Returns design document with a given design doc name

open_doc(*DbName :: string()*, *DocID :: string()*) → {ok, #doc{}} | {error, {not_found, missing | deleted}} | {error, term()}

Returns document with a given DocID

open_doc(*DbName :: string(), DocID :: string(), Opts :: [Option]*) → {ok, #doc{}} | {error, {not_found, missing | deleted}} | {error, term()}

- **Option:** atom() | {atom(), term()}

Returns document with a given DocID

query_view(*DbName :: string()*, *DesignName :: string()*, *ViewName :: string()*) → {ok, QueryResult :: term()} | {error, term()}

Execute a given view with default set of arguments. Check record #view_query_args for details.

query_view(*DbName :: string()*, *DesignName :: string()*, *ViewName :: string()*, *QueryArgs :: #view_query_args{}*) → {ok, QueryResult :: term()} | {error, term()}

Execute a given view. There are many additional query args that can be passed to a view, see query args for details.

revision(*RevInfo :: term()) -> term()*

Normalize revision info

dao_hosts

Authors Rafal Slota

Copyright This software is released under the *MIT License*.

Descritpion This module manages hosts and connections to VeilFS DB

Function Index

- *insert/1*
- *reactivate/1*
- *store_exec/2*

Function Details

ban (*Host :: atom()*) → ok | {error, timeout}

Bans db host name (lowers its priority while selecting host in get_host/0) Ban will be cleaned after DEFAULT_BAN_TIME ms or while store refresh

ban (*Host :: atom()*, *BanTime :: integer()*) → ok | {error, no_host} | {error, timeout}

Bans db host name (lowers its priority while selecting host in get_host/0) Ban will be cleaned after BanTime ms or while store refresh If given Host is already banned, nothing happens and will return 'ok' If given Host wasn't inserted, returns {error, no_host}

call (*Method :: atom(), Args :: [term()]*) → term() | {error, rpc_retry_limit_exceeded}

Calls fabric:Method with Args on random db host. Random host will be assigned to the calling process and will be used with subsequent calls

call (*Module :: atom(), Method :: atom(), Args :: [Arg :: term()]*) → term() | Error

- **Error:** {error, rpc_retry_limit_exceeded} | {error, term()}

Same as call/2, but with custom Module

delete (*Host :: atom()*) → ok | {error, timeout}

Deletes db host name from store (host pool)

insert (*Host :: atom()*) → ok | {error, timeout}

Inserts db host name into store (host pool)

reactivate (*Host :: atom()*) → ok | {error, no_host} | {error, timeout}

Reactivate banned db host name If given Host wasn't banned, nothing happens and will return 'ok' If given Host wasn't inserted, returns {error, no_host}

store_exec (*sequential*, *Msg :: term()*) → ok | {error, Error :: term()}

Executes Msg. Caller must ensure that this method won't be used concurrently. Currently this method is used as part of internal module implementation, although it has to be exported because it's called by gen_server (which ensures it's sequential call).

dao_json

Authors Rafal Slota

Copyright This software is released under the *MIT License*.

Descritpion BigCouch document management module (very simple - BigCouch specific - JSON creator)

Function Index

Function Details

get_field(*DocOrObj*, *Name :: string()*) → any() | {error, not_found} | {error, invalid_object}

- **DocOrObj:** #doc{} | json_object()

Returns field's value from given document or JSON object

get_fields(*DocOrObj*) → [{FieldName :: string(), FieldValue :: any()}] | {error, not_found} | {error, invalid_object}

- **DocOrObj:** #doc{} | json_object()

Returns field's values from given document or JSON object

mk_bin(*Term :: term()*) -> *binary()*

Converts given term to binary form used by BigCouch/CouchDB

mk_doc(*Id :: string()*) → #doc{}

Returns new BigCouch document with given Id

mk_field(*DocOrObj*, *Name :: string()*, *Value :: term()*) → DocOrObj

- **DocOrObj:** #doc{} | json_object()

Inserts new field into given document or JSON object

mk_fields(*DocOrObj*[, *Names :: string()*][, *Values :: term()*]) → DocOrObj

- **DocOrObj:** #doc{} | json_object()

Inserts new fields into given document or JSON object

mk_obj() → {[]}

Returns empty json object structure used by BigCouch/CouchDB

mk_str(*Str :: string() | atom()*) -> *binary()*

Converts given string to binary form used by BigCouch/CouchDB

reverse_fields (*DocOrObj*) → DocOrObj

- **DocOrObj:** #doc{} | json_object()

Reverses fields in given document or JSON object

rm_field (*DocOrObj*, *Name :: string()*) → DocOrObj

- **DocOrObj:** #doc{} | json_object()

Removes field from given document or JSON object

rm_fields (*DocOrObj*[, *Name :: string()*]) → DocOrObj

- **DocOrObj:** #doc{} | json_object()

Removes fields from given document or JSON object

dao_lib

Authors Rafal Slota

Copyright This software is released under the *MIT License*.

Descritpion DAO helper/utility functional methods. Those can be used in other modules bypassing worker_host and gen_server.

Function Index

- *apply/4*
- *apply/5*
- *strip_wrappers/1*
- *wrap_record/1*

Function Details

apply (*Module :: module(), Method :: atom() | {synch, atom()} | {asynch, atom()}, Args :: [term()], ProtocolVersion :: number()*) → any() | {error, worker_not_found}

Same as apply/5 but with default Timeout

apply (*Module :: module(), Method :: atom() | {synch, atom()} | {asynch, atom()}, Args :: [term()], ProtocolVersion :: number(), Timeout :: pos_integer()*) → any() | {error, worker_not_found} | {error, timeout}

Behaves similar to erlang:apply/3 but works only with DAO worker . Method calls are made through random gen_server. Method should be tuple {synch, Method} or {asynch, Method} but if its simple atom(), {synch, Method} is assumed Timeout argument defines how long should this method wait for response

strip_wrappers (*VeilDocOrList :: #veil_document{} | [#veil_document{}]*) → tuple() | [tuple()]

Strips #veil_document{} wrapper. Argument can be either an #veil_document{} or list of #veil_document{} Alternatively arguments can be passed as {ok, Arg} tuple. Its convenient because most DAO methods formats return value formatted that way If the argument cannot be converted (e.g. error tuple is passed), this method returns it unchanged. This method is designed for use as wrapper for "get_*"-like DAO methods. E.g.

dao_lib:strip_wrappers(dao_vfs:get_file({absolute_path, "/foo/bar"})) See *dao:save_record/1* and *dao:get_record/1* for more details about #veil_document{} wrapper.

wrap_record(*Record :: tuple()*) → #veil_document{}

Wraps given erlang record with #veil_document{} wrapper. See *dao:save_record/1* and *dao:get_record/1* for more details about #veil_document{} wrapper.

dao_share

Authors Michał Wrzeszcz

Copyright This software is released under the *MIT License*.

Descritpion This module provides high level DB API for handling file sharing.

Function Index

- *get_file_share/1*
- *remove_file_share/1*
- *save_file_share/1*

Function Details

get_file_share(*Key:: {file, File :: uuid()} | {user, User :: uuid()} | {uuid, UUID :: uuid()}) -> {ok, file_share_doc()} | {ok, [file_share_doc()]} | {error, any()} | no_return()*

Gets info about file sharing from db by share_id, file name or user uuid. 1 Non-error return value is always {ok, #veil_document{record = #share_desc}. See *dao:save_record/1* and *dao:get_record/1* for more details about #veil_document{} wrapper. Should not be used directly, use *dao:handle/2* instead (See *dao:handle/2* for more details).

remove_file_share(*Key:: {file, File :: uuid()} | {user, User :: uuid()} | {uuid, UUID :: uuid()}) -> {error, any()} | no_return()*

Removes info about file sharing from DB by share_id, file name or user uuid. Should not be used directly, use *dao:handle/2* instead (See *dao:handle/2* for more details).

save_file_share(*Share :: file_share_info() | file_share_doc()) -> {ok, file_share()} | {error, any()} | no_return()*

Saves info about file sharing to DB. Argument should be either #share_desc{} record (if you want to save it as new document) or #veil_document{} that wraps #share_desc{} if you want to update descriptor in DB. See *dao:save_record/1* and *dao:get_record/1* for more details about #veil_document{} wrapper. Should not be used directly, use *dao:handle/2* instead (See *dao:handle/2* for more details).

dao_users

Authors Lukasz Opiola

Copyright This software is released under the *MIT License*.

Descritpion This module provides high level DB API for handling user documents.

Function Index

- *get_files_number/2*
- *get_user/1*
- *remove_user/1*
- *save_user/1*

Function Details

get_user(*Key:: {login, Login :: string()} | {email, Email :: string()} | {uuid, UUID :: uuid()} | {dn, DN :: string()}) -> {ok, user_doc()} | {error, any()} | no_return()*

Gets user from DB by login, e-mail, uuid or dn. Non-error return value is always {ok, #veil_document{record = #user}. See *dao:save_record/1* and *dao:get_record/1* for more details about #veil_document{} wrapper. Should not be used directly, use *dao:handle/2* instead (See *dao:handle/2* for more details).

remove_user(*Key:: {login, Login :: string()} | {email, Email :: string()} | {uuid, UUID :: uuid()} | {dn, DN :: string()}) -> {error, any()} | no_return()*

Removes user from DB by login, e-mail, uuid or dn. Should not be used directly, use *dao:handle/2* instead (See *dao:handle/2* for more details).

save_user(*User :: user_info() | user_doc()) -> {ok, user()} | {error, any()} | no_return()*

Saves user to DB. Argument should be either #user{} record (if you want to save it as new document) or #veil_document{} that wraps #user{} if you want to update descriptor in DB. See *dao:save_record/1* and *dao:get_record/1* for more details about #veil_document{} wrapper. Should not be used directly, use *dao:handle/2* instead (See *dao:handle/2* for more details).

dao_vfs

Authors Rafal Slota

Copyright This software is released under the *MIT License*.

Descritpion This module gives high level DB API which contain veil file system specific methods. All DAO API functions should not be called directly. Call dao:handle(_, {vfs, MethodName, ListOfArgs) instead. See dao:handle/2 for more details.

Function Index

- *count_subdirs/1*
- *find_files/1*
- *get_descriptor/1*
- *get_file/1*
- *get_file_meta/1*
- *get_path_info/1*

Function Details

count_subdirs (*{uuid, UUID :: uuid()}*) → {ok, non_neg_integer()}

Returns number of first level subdirectories for specified directory.

find_files (*FileCriteria :: file_criteria()) -> {ok, [file_doc()]} | no_return()*

Returns list of uuids of files that matches to criteria passed as FileCriteria Current implementation does not support specifying ctime and mtime at the same time, other combinations of criterias are supported.

get_descriptor (*Fd :: fd()) -> {ok, fd_doc()} | {error, any()} | no_return()*

Gets file descriptor from DB. Argument should be uuid() of #file_descriptor record Non-error return value is always {ok, #veil_document{record = #file_descriptor}. See *dao:save_record/1* and *dao:get_record/1* for more details about #veil_document{} wrapper. Should not be used directly, use *dao:handle/2* instead (See *dao:handle/2* for more details).

get_file (*File :: file()) -> {ok, file_doc()} | {error, any()} | no_return()*

Gets file from DB. Argument should be file() - see dao_types.hrl for more details Should not be used directly, use *dao:handle/2* instead (See *dao:handle/2* for more details).

get_file_meta (*Fd :: fd()) -> {ok, fd_doc()} | {error, any()} | no_return()*

Gets file meta from DB. Argument should be uuid() of #file_meta record Non-error return value is always {ok, #veil_document{record = #file_meta}. See *dao:save_record/1* and *dao:get_record/1* for more details about #veil_document{} wrapper. Should not be used directly, use *dao:handle/2* instead (See *dao:handle/2* for more details).

get_path_info (*File :: file_path()) -> {ok, [file_doc()]} | {error, any()} | no_return()*

Gets all files existing in given path from DB. Argument should be file_path() - see dao_types.hrl for more details Similar to get_file/1 but returns list containing file_doc() for every file within given path(), not only the last one Should not be used directly, use *dao:handle/2* instead (See *dao:handle/2* for more details).

get_storage (*{uuid, DocUUID :: uuid()} | {id, StorageID :: integer()}) -> {ok, storage_doc()} | {error, any()} | no_return()*

Gets storage info from DB. Argument should be uuid() of storage document or ID of storage. Non-error return value is always {ok, #veil_document{record = #storage_info{}}. See *dao:save_record/1* and *dao:get_record/1* for more details about #veil_document{} wrapper. Should not be used directly, use *dao:handle/2* instead (See *dao:handle/2* for more details).

list_descriptors (*MatchCriteria :: fd_select(), N :: pos_integer(), Offset :: non_neg_integer()) -> {ok, fd_doc()} | {error, any()} | no_return()*

Lists file descriptor from DB. First argument is a two-element tuple containing type of resource used to filter descriptors and resource itself Currently only {by_file, File :: file()} is supported. Second argument limits number of rows returned. 3rd argument sets offset of query (skips first Offset rows) Non-error return value is always {ok, [#veil_document{record = #file_descriptor]}. See *dao:save_record/1* and *dao:get_record/1* for more details about #veil_document{} wrapper. Should not be used directly, use *dao:handle/2* instead (See *dao:handle/2* for more details).

list_dir (*Dir :: file(), N :: pos_integer(), Offset :: non_neg_integer()*) → {ok, [file_doc()]}

Lists N files from specified directory starting from Offset. Non-error return value is always list of #veil_document{record = #file{}} records. Should not be used directly, use dao:handle/2 instead (See dao:handle/2 for more details).

list_storage (*) -> {ok, [storage_doc()]} | no_return()*

Lists all storage docs. Non-error return value is always list of #veil_document{record = #storage_info{}} records. Should not be used directly, use dao:handle/2 instead (See dao:handle/2 for more details).

lock_file (*UserID :: string(), FileID :: string(), Mode :: write | read*) → not_yet_implemented

Puts a read/write lock on specified file owned by specified user. Should not be used directly, use dao:handle/2 instead (See dao:handle/2 for more details). Not yet implemented. This is placeholder/template method only!

remove_descriptor (*Fd :: fd() | fd_select()) -> ok | {error, any()} | no_return()*

Removes file descriptor from DB. Argument should be uuid() of #file_descriptor or same as in *dao_vfs:list_descriptors/3* . Should not be used directly, use *dao:handle/2* instead (See *dao:handle/2* for more details).

remove_file (*File :: file()) -> ok | {error, any()} | no_return()*

Removes file from DB. Argument should be file() - see dao_types.hrl for more details Should not be used directly, use *dao:handle/2* instead (See *dao:handle/2* for more details).

remove_file_meta (*FMeta :: uuid()) -> ok | {error, any()} | no_return()*

Removes file_meta from DB. Argument should be uuid() of veil_document - see dao_types.hrl for more details Should not be used directly, use *dao:handle/2* instead (See *dao:handle/2* for more details).

remove_storage (*{uuid, DocUUID :: uuid()} | {id, StorageID :: integer()}) -> ok | {error, any()} | no_return()*

Removes storage info from DB. Argument should be uuid() of storage document or ID of storage Should not be used directly, use *dao:handle/2* instead (See *dao:handle/2* for more details).

rename_file (*File :: file(), NewName :: string()) -> {ok, NewUUID :: uuid()} | no_return()*

Renames specified file to NewName. Should not be used directly, use dao:handle/2 instead (See dao:handle/2 for more details).

save_descriptor (*Fd :: fd_info() | fd_doc()) -> {ok, uuid()} | {error, any()} | no_return()*

Saves file descriptor to DB. Argument should be either #file_descriptor{} record (if you want to save it as new document) or #veil_document{} that wraps #file_descriptor{} if you want to update descriptor in DB. See *dao:save_record/1* and *dao:get_record/1* for more details about #veil_document{} wrapper. Should not be used directly, use *dao:handle/2* instead (See *dao:handle/2* for more details).

save_file (*File :: file_info() | file_doc()) -> {ok, uuid()} | {error, any()} | no_return()*

Saves file to DB. Argument should be either #file{} record (if you want to save it as new document) or #veil_document{} that wraps #file{} if you want to update file in DB. See *dao:save_record/1* and *dao:get_record/1* for more details about #veil_document{} wrapper. Should not be used directly, use *dao:handle/2* instead (See *dao:handle/2* for more details).

save_file_meta (*FMeta :: #file_meta{} | #veil_document{}) -> {ok, uuid()} | {error, any()} | no_return()*

Saves file_meta to DB. Argument should be either #file_meta{} record (if you want to save it as new document) or #veil_document{} that wraps #file_meta{} if you want to update file meta in DB. See *dao:save_record/1* and *dao:get_record/1* for more details about #veil_document{} wrapper. Should not be used directly, use *dao:handle/2* instead (See *dao:handle/2* for more details).

save_new_file (*FilePath :: string(), File :: file_info()) -> {ok, uuid()} | {error, any()} | no_return()*

Saves new file to DB See *dao:save_record/1* and *dao:get_record/1* for more details about #veil_document{} wrapper. Should not be used directly, use *dao:handle/2* instead (See *dao:handle/2* for more details).

save_storage (*Storage :: #storage_info{} | #veil_document{}) -> {ok, uuid()} | {error, any()} | no_return()*

Saves storage info to DB. Argument should be either #storage_info{} record (if you want to save it as new document) or #veil_document{} that wraps #storage_info{} if you want to update storage info in DB. See *dao:save_record/1* and *dao:get_record/1* for more details about #veil_document{} wrapper. Should not be used directly, use *dao:handle/2* instead (See *dao:handle/2* for more details).

unlock_file (*UserID :: string(), FileID :: string(), Mode :: write | read*) → not_yet_implemented

Takes off a read/write lock on specified file owned by specified user. Should not be used directly, use dao:handle/2 instead (See dao:handle/2 for more details). Not yet implemented. This is placeholder/template method only!

user_logic

Authors Lukasz Opiola

Copyright This software is released under the *MIT License*.

Descritpion This module provides methods for managing users in the system.

Function Index

Function Details

`create_user` (*Login*, *Name*, *Teams*, *Email*, *DnList*) → Result

- **DnList:** [string()]
- **Email:** string()
- **Login:** string()
- **Name:** string()
- **Result:** {ok, user_doc()} | {error, any()}
- **Teams:** string()

Creates a user in the DB.

`extract_dn_from_cert` (*PemBin :: binary()*) → {rdnSequence, [#'AttributeTypeAndValue'{}]} | {error, Reason}

- **Reason:** proxy_ceertificate | self_signed | extraction_failed

Processes a .pem certificate and extracts subject (DN) part, returning it as an rdnSequence. Returns an error if: - fails to extract DN - certificate is a proxy certificate -> {error, proxy_ceertificate} - certificate is self-signed -> {error, self_signed}

get_dn_list (*User*) → Result

- **Result:** string()
- **User:** user_doc()

  Convinience function to get DN list from #veil_document encapsulating #user record.

get_email_list (*User*) → Result

- **Result:** string()
- **User:** user_doc()

  Convinience function to get e-mail list from #veil_document encapsulating #user record.

get_login (*User*) → Result

- **Result:** string()
- **User:** user_doc()

  Convinience function to get user login from #veil_document encapsulating #user record.

get_name (*User*) → Result

- **Result:** string()
- **User:** user_doc()

  Convinience function to get user name from #veil_document encapsulating #user record.

get_team_names (*UserQuery :: term()) -> [string()] | no_return()*

Returns list of group/team names for given user. UserQuery shall be either #user{} record or query compatible with user_logic:get_user/1. The method assumes that user exists therefore will fail with exception when it doesnt.

get_teams (*User*) → Result

- **Result:** string()
- **User:** user_doc()

  Convinience function to get user teams from #veil_document encapsulating #user record.

get_user (*Key :: {login, Login :: string()} | {email, Email :: string()} | {uuid, UUID :: user()} | {dn, DN :: string()} | {rdnSequence, [#'AttributeTypeAndValue'{}]}*) → {ok, user_doc()} | {error, any()}

:noindex:

Retrieves user from DB by login, email, uuid, DN or rdnSequence proplist. Returns veil_document wrapping a #user record.

invert_dn_string (*string()) -> string()*

Inverts the sequence of entries in a DN string.

oid_code_to_shortname (*term()) -> string() | no_return()*

Converts erlang-like OID code to OpenSSL short name.

rdn_sequence_to_dn_string(*[#'AttributeTypeAndValue'{}]) -> string() | no_return()*)

Converts rdnSequence to DN string so that it can be compared to another DN.

remove_user(*Key :: {login, Login :: string()} | {email, Email :: string()} | {uuid, UUID :: user()} | {dn, DN :: string()} | {rdnSequence, [#'AttributeTypeAndValue'{}]}*) → Result :: ok | {error, any()}

:noindex:

Removes user from DB by login.

shortname_to_oid_code(*string()) -> term() | no_return()*)

Converts OpenSSL short name to erlang-like OID code.

sign_in(*Proplist*) → Result

- **Proplist:** list()
- **Result:** {string(), user_doc()}

This function should be called after a user has logged in via OpenID. It looks the user up in database by login. If he is not there, it creates a proper document. If the user already exists, synchronization is made between document in the database and info received from OpenID provider.

update_dn_list(*User*, *NewDnList*) → Result

- **NewDnList:** [string()]
- **Result:** {ok, user_doc()} | {error, any()}
- **User:** user_doc()

Update #veil_document encapsulating #user record with new DN list and save it to DB.

update_email_list(*User*, *NewEmailList*) → Result

- **NewEmailList:** [string()]
- **Result:** {ok, user_doc()} | {error, any()}
- **User:** user_doc()

Update #veil_document encapsulating #user record with new e-mail list and save it to DB.

update_teams(*User*, *NewTeams*) → Result

- **NewTeams:** string()
- **Result:** {ok, user_doc()} | {error, any()}
- **User:** user_doc()

Update #veil_document encapsulating #user record with new teams and save it to DB.

11.4.10 Dns

Documents

dns_ranch_tcp_handler

Authors Bartosz Polnik

Copyright This software is released under the *MIT License*.

Descritpion This module is responsible for handling tcp aspects of dns protocol.

Behaviours ranch_protocol (https://github.com/extend/ranch/blob/master/manual/ranch_protocol.md)

Function Index

- *loop/5*
- *start_link/4*

Function Details

loop (*Socket*, *Transport*, *ResponseTTL*, *TCPIdleTime*, *DispatcherTimeout*) → ok

- **DispatcherTimeout:** non_neg_integer()
- **ResponseTTL:** non_neg_integer()
- **Socket:** inet:socket()
- **TCPIdleTime:** non_neg_integer()
- **Transport:** term()

Main handler loop.

start_link (*Ref :: term()*, *Socket :: term()*, *Transport :: term()*, *Opts :: term()*) → Result

- **Pid:** pid()
- **Result:** {ok, Pid}

Starts handler.

dns_udp_handler

Authors Bartosz Polnik

Copyright This software is released under the *MIT License*.

Descritpion This module is responsible for handling udp aspects of dns protocol.

Function Index

- *loop/3*
- *start_link/3*

Function Details

loop (*Socket*, *ResponseTTL*, *DispatcherTimeout) -> no_return()*

- **DispatcherTimeout:** non_neg_integer()
- **ResponseTTL:** non_neg_integer()
- **Socket:** inet:socket()

Loop maintaining state.

start_link (*Port*, *ResponseTTLInSecs*, *DispatcherTimeout*) → {ok, pid()}

- **DispatcherTimeout:** non_neg_integer()
- **Port:** non_neg_integer()
- **ResponseTTLInSecs:** non_neg_integer()

Creates process to handle udp socket.

dns_utils

Authors Bartosz Polnik

Copyright This software is released under the *MIT License*.

Descritpion This module is responsible for creating dns response for given dns request. Currently only supported are queries of type a and class in.

Function Index

- *generate_answer/5*

Function Details

generate_answer (*Packet*, *Dispatcher*, *DispatcherTimeout*, *ResponseTTL*, *Protocol*) → Result

- **Dispatcher:** term()
- **DispatcherTimeout:** non_neg_integer()
- **Packet:** binary()
- **Protocol:** udp | tcp
- **ResponseTTL:** non_neg_integer()
- **Result:** {ok, binary()} | {error, term()}

Generates binary dns response for given binary dns request, non protocol agnostic.

dns_worker

Authors Bartosz Polnik

Copyright This software is released under the *MIT License*.

Descritpion This module implements *worker_plugin_behaviour* to provide functionality of resolution ipv4 addresses for given worker name.

Behaviours *worker_plugin_behaviour*

Function Index

- *cleanup/0*
- *env_dependencies/0*
- *handle/2*
- *init/1*
- *start_listening/0*

Function Details

cleanup() → Result

- **Result:** ok

worker_plugin_behaviour callback cleanup/0

handle(*ProtocolVersion :: term()*, *Request*) → Result

- **Error:** term()
- **Request:** ping | get_version | {update_state, list(), list()} | {get_worker, atom()} | get_nodes
- **Response:** [inet:ip4_address()]
- **Result:** ok | {ok, Response} | {error, Error} | pong | Version
- **Version:** term()

worker_plugin_behaviour callback handle/1. Calling handle(_, ping) returns pong. Calling handle(_, get_version) returns current version of application. Calling handle(_. {update_state, _}) updates plugin state. Calling handle(_, {get_worker, Name}) returns list of ipv4 addresses of workers with specified name.

init(*Args :: term()*) → Result

- **Error:** term()
- **Result:** #dns_worker_state{} | {error, Error}

worker_plugin_behaviour callback init/1.

start_listening() → ok

Starts dns listeners and terminates dns_worker process in case of error.

11.4.11 Fslogic

Documents

fslogic

Authors Michal Wrzeszcz

Copyright This software is released under the *MIT License*.

Descritpion This module implements worker_plugin_behaviour to provide functionality of file system logic.

Behaviours *worker_plugin_behaviour*

Function Index

Function Details

cleanup() → ok

worker_plugin_behaviour callback cleanup/0

create_dirs(*Count :: integer()*, *CountingBase :: integer()*, *SHInfo :: term()*, *TmpAns :: string()*) -> *string()*

Creates dir at storage for files (if needed). Returns the path that contains created dirs.

get_file(*ProtocolVersion :: term()*, *File :: string()*, *FuseID :: string()*) → Result

- **Result:** term()

Gets file info from DB

get_files_number(*user | group*, *UUID :: uuid() | string()*, *ProtocolVersion :: integer()*) → Result

- **Result:** {ok, Sum} | {error, any()}
- **Sum:** integer()

Returns number of user's or group's files

get_full_file_name(*FileName :: string()*) → Result

- **ErrorDesc:** atom
- **FullFileName:** string()
- **Result:** {ok, FullFileName} | {error, ErrorDesc}

Gets file's full name (user's root is added to name, but only when asking about non-group dir).

get_user_id() → Result

- **ErrorDesc:** atom
- **Result:** {ok, UserID} | {error, ErrorDesc}
- **UserID:** term()

Gets user's id.

get_user_root(*UserDoc :: term()*) → Result

- **ErrorDesc:** atom
- **Result:** {ok, RootDir} | {error, ErrorDesc}
- **RootDir:** string()

Gets user's root directory.

handle(*ProtocolVersion :: term()*, *Request :: term()*) → Result

- **Result:** term()

worker_plugin_behaviour callback handle/1. Processes standard worker requests (e.g. ping) and requests from FUSE.

init(*Args :: term()*) -> *list*()

worker_plugin_behaviour callback init/1

fslogic_storage

Authors Rafal Slota

Copyright This software is released under the *MIT License*.

Descritpion This module exports storage management tools for fslogic

Function Index

- *get_sh_for_fuse/2*
- *insert_storage/2*
- *insert_storage/3*
- *select_storage/2*

Function Details

get_sh_for_fuse(*FuseID :: string()*, *Storage :: #storage_info{}*) → #storage_helper_info{}

Returns #storage_helper_info{} record which describes storage helper that is connected with given storage (described with #storage_info{} record). Each storage can have multiple storage helpers, that varies between FUSE groups, so that different FUSE clients (with different FUSE_ID) could select different storage helper.

insert_storage(*HelperName :: string()*, *HelperArgs :: [string()]*) -> *term*()

Creates new mock-storage info in DB that uses default storage helper with name HelperName and argument list HelperArgs.

insert_storage (*HelperName :: string(), HelperArgs :: [string()], Fuse_groups :: list()) -> term()*

Creates new mock-storage info in DB that uses default storage helper with name HelperName and argument list HelperArgs. TODO: This is mock method and should be replaced by GUI-tool form control_panel module.

select_storage (*FuseID :: string(), StorageList :: [#storage_info{}]*) → #storage_info{}

Chooses and returns one storage_info from given list of #storage_info records. TODO: This method is an mock method that shall be replaced in future. Currently returns random #storage_info{}.

fslogic_utils

Authors Rafal Slota

Copyright This software is released under the *MIT License*.

Descritpion This module exports utility tools for fslogic

Function Index

- *basename/1*
- *create_children_list/1*
- *create_children_list/2*
- *get_parent_and_name_from_path/2*
- *strip_path_leaf/1*
- *time/0*
- *update_meta_attr/3*

Function Details

basename (*Path :: string()) -> string()*

Gives file basename from given path

create_children_list (*Files :: list()*) → Result

- **Result:** term()

Creates list of children logical names on the basis of list with veil_documents that describe children.

create_children_list (*Files :: list(), TmpAns :: list()*) → Result

- **Result:** term()

Creates list of children logical names on the basis of list with veil_documents that describe children.

get_parent_and_name_from_path (*Path :: string()*, *ProtocolVersion :: term()*) → Result

- **Result:** tuple()

Gets parent uuid and file name on the basis of absolute path.

strip_path_leaf (*Path :: string()*) -> *string()*

Strips file name from path

time () -> *Result :: integer()*

Returns time in seconds.

update_meta_attr (*File :: #file{}*, *Attr*, *Value :: term()*) → Result :: #file{}

- **Attr:** atime | mtime | ctime | size | times

Updates file_meta record associated with given #file record. Attr agument decides which field has to be updated with Value. There is one exception to this rule: if Attr == 'times', Value has to be tuple with fallowing format: {ATimeValue, MTimeValue, CTimeValue} or {ATimeValue, MTimeValue}. If there is no #file_meta record associated with given #file, #file_meta will be created and whole function call will be blocking. Otherwise the method call will be asynchronous. Returns given as argument #file record unchanged, unless #file_meta had to be created. In this case returned #file record will have #file.meta_doc field updated and shall be saved to DB after this call.

logical_files_manager

Authors Michal Wrzeszcz

Copyright This software is released under the *MIT License*.

Descritpion This module provides high level file system operations that use logical names of files.

Function Index

- *get_file_user_dependent_name_by_uuid/1*
- *get_share/1*
- *getfileattr/1*
- *getfilelocation/1*
- *ls/3*
- *mkdir/1*
- *mv/2*
- *read/3*
- *remove_share/1*
- *rmdir/1*
- *truncate/2*
- *write/2*
- *write/3*
- *write_from_stream/2*

Function Details

change_file_perm(*FileName :: string()*, *NewPerms :: integer()*) → Result

- **ErrorDetail:** term()
- **ErrorGeneral:** atom()
- **Result:** ok | {ErrorGeneral, ErrorDetail}

Changes file's permissions in db

chown() → {error, not_implemented_yet}

Changes owner of file (in db)

create(*File :: string()*) → Result

- **ErrorDetail:** term()
- **ErrorGeneral:** atom()
- **Result:** ok | {ErrorGeneral, ErrorDetail}

Creates file (uses logical name of file). First it creates file in db and gets information about storage helper and file id at helper. Next it uses storage helper to create file on storage.

create_share(*File :: string()*, *Share_With :: term()*) → Result

- **ErrorDetail:** term()
- **ErrorGeneral:** atom()
- **Result:** {ok, Share_info} | {ErrorGeneral, ErrorDetail}
- **Share_info:** term()

Creates share info for file (file path is an argument).

create_standard_share(*File :: string()*) → Result

- **ErrorDetail:** term()
- **ErrorGeneral:** atom()
- **Result:** {ok, Share_info} | {ErrorGeneral, ErrorDetail}
- **Share_info:** term()

Creates standard share info (share with all) for file (file path is an argument).

delete(*File :: string()*) → Result

- **ErrorDetail:** term()
- **ErrorGeneral:** atom()
- **Result:** ok | {ErrorGeneral, ErrorDetail}

Deletes file (uses logical name of file). First it gets information about storage helper and file id at helper. Next it uses storage helper to delete file from storage. Afterwards it deletes information about file from db.

doUploadTest(*File :: string()*, *WriteFunNum :: integer()*, *Size :: integer()*, *Times :: integer()*) → Result

- **BytesWritten:** integer()
- **Result:** {BytesWritten, WriteTime}
- **WriteTime:** integer()

Tests upload speed

error_to_string(*Error :: term()*) → Result

- **Result:** string()

Translates error to text message.

exists(*File :: string()*) → Result

- **ErrorDetail:** term()
- **ErrorGeneral:** atom()
- **Result:** boolean() | {ErrorGeneral, ErrorDetail}

Checks if file exists.

get_ets_name() → Result

- **Result:** atom()

Generates name of ets table for proc

get_file_by_uuid(*UUID :: string()*) → Result

- **ErrorDetail:** term()
- **ErrorGeneral:** atom()
- **File:** term()
- **Result:** {ok, File} | {ErrorGeneral, ErrorDetail}

Gets file record on the basis of uuid.

get_file_full_name_by_uuid(*UUID :: string()*) → Result

- **ErrorDetail:** term()

- **ErrorGeneral:** atom()
- **FullPath:** string()
- **Result:** {ok, FullPath} | {ErrorGeneral, ErrorDetail}

Gets file full name (with root of the user's system) on the basis of uuid.

get_file_name_by_uuid(*UUID :: string()*) → Result

- **ErrorDetail:** term()
- **ErrorGeneral:** atom()
- **Name:** term()
- **Result:** {ok, Name} | {ErrorGeneral, ErrorDetail}

Gets file name on the basis of uuid.

get_file_user_dependent_name_by_uuid(*UUID :: string()*) → Result

- **ErrorDetail:** term()
- **ErrorGeneral:** atom()
- **FullPath:** string()
- **Result:** {ok, FullPath} | {ErrorGeneral, ErrorDetail}

Gets file full name relative to user's dir on the basis of uuid.

get_share(*Key:: {file, File :: uuid()} | {user, User :: uuid()} | {uuid, UUID :: uuid()}*) → Result

- **ErrorDetail:** term()
- **ErrorGeneral:** atom()
- **Result:** {ok, Share_doc} | {ErrorGeneral, ErrorDetail}
- **Share_doc:** term()

Gets info about share from db.

getfileattr(*FileName :: string()*) → Result

- **Attributes:** term()
- **ErrorDetail:** term()
- **ErrorGeneral:** atom()
- **Result:** {ok, Attributes} | {ErrorGeneral, ErrorDetail}

Returns file attributes

getfilelocation(*File :: term()*) → Result

- **ErrorDetail:** term()
- **ErrorGeneral:** atom()
- **Helper:** term()
- **Id:** term()
- **Result:** {ok, {Helper, Id}} | {ErrorGeneral, ErrorDetail}

Gets file location from fslogic or from cache. File can be string (path) or {uuid, UUID}.

ls(*DirName :: string(), ChildrenNum :: integer(), Offset :: integer()*) → Result

- **ErrorDetail:** term()
- **ErrorGeneral:** atom()
- **FilesList:** list()
- **Result:** {ok, FilesList} | {ErrorGeneral, ErrorDetail}

Lists directory (uses data from db)

mkdir (*DirName :: string()*) → Result

- **ErrorDetail:** term()
- **ErrorGeneral:** atom()
- **Result:** ok | {ErrorGeneral, ErrorDetail}

Creates directory (in db)

mv (*From :: string()*, *To :: string()*) → Result

- **ErrorDetail:** term()
- **ErrorGeneral:** atom()
- **Result:** ok | {ErrorGeneral, ErrorDetail}

Moves directory (in db)

read (*File :: term()*, *Offset :: integer()*, *Size :: integer()*) → Result

- **Bytes:** binary()
- **ErrorDetail:** term()
- **ErrorGeneral:** atom()
- **Result:** {ok, Bytes} | {ErrorGeneral, ErrorDetail}

Reads file (uses logical name of file). First it gets information about storage helper and file id at helper. Next it uses storage helper to read data from file. File can be string (path) or {uuid, UUID}.

remove_share (*Key:: {file*, *File :: uuid()}* | *{user*, *User :: uuid()}* | *{uuid*, *UUID :: uuid()}*) → Result

- **ErrorDetail:** term()
- **ErrorGeneral:** atom()
- **Result:** ok | {ErrorGeneral, ErrorDetail}

Removes info about share from db.

rmdir (*DirName :: string()*) → Result

- **ErrorDetail:** term()
- **ErrorGeneral:** atom()
- **Result:** ok | {ErrorGeneral, ErrorDetail}

Deletes directory (in db)

truncate (*File :: string()*, *Size :: integer()*) → Result

- **ErrorDetail:** term()
- **ErrorGeneral:** atom()

- **Result:** ok | {ErrorGeneral, ErrorDetail}

Truncates file (uses logical name of file). First it gets information about storage helper and file id at helper. Next it uses storage helper to truncate file on storage.

write (*File :: string()*, *Buf :: binary()*) → Result

- **BytesWritten:** integer()
- **ErrorDetail:** term()
- **ErrorGeneral:** atom()
- **Result:** BytesWritten | {ErrorGeneral, ErrorDetail}

Appends data to the end of file (uses logical name of file). First it gets information about storage helper and file id at helper. Next it uses storage helper to write data to file.

write (*File :: string()*, *Offset :: integer()*, *Buf :: binary()*) → Result

- **BytesWritten:** integer()
- **ErrorDetail:** term()
- **ErrorGeneral:** atom()
- **Result:** BytesWritten | {ErrorGeneral, ErrorDetail}

Writes data to file (uses logical name of file). First it gets information about storage helper and file id at helper. Next it uses storage helper to write data to file.

write_from_stream (*File :: string()*, *Buf :: binary()*) → Result

- **BytesWritten:** integer()
- **ErrorDetail:** term()
- **ErrorGeneral:** atom()
- **Result:** BytesWritten | {ErrorGeneral, ErrorDetail}

Appends data to the end of file (uses logical name of file). First it gets information about storage helper and file id at helper. Next it uses storage helper to write data to file.

storage_files_manager

Authors Michal Wrzeszcz

Copyright This software is released under the *MIT License*.

Descritpion This module provides high level file system operations that operates directly on storage.

Function Index

- *chmod/3*
- *chown/4*
- *create/2*
- *delete/2*
- *delete_dir/2*

- *ls/0*
- *mkdir/2*
- *mv/3*
- *read/4*
- *truncate/3*
- *write/3*
- *write/4*

Function Details

chmod (*Storage_helper_info :: record()*, *Dir :: string()*, *Mode :: integer()*) → Result

- **ErrorDetail:** term()
- **ErrorGeneral:** atom()
- **Result:** ok | {ErrorGeneral, ErrorDetail}

Change file mode at storage

chown (*Storage_helper_info :: record()*, *Dir :: string()*, *User :: string()*, *Group :: string()*) → Result

- **ErrorDetail:** term()
- **ErrorGeneral:** atom()
- **Result:** ok | {ErrorGeneral, ErrorDetail}

Change file's owner (if user or group shouldn't be changed use "" as an argument)

create (*Storage_helper_info :: record()*, *File :: string()*) → Result

- **ErrorDetail:** term()
- **ErrorGeneral:** atom()
- **Result:** ok | {ErrorGeneral, ErrorDetail}

Creates file (operates only on storage). First it checks if file exists. If not, it creates file.

delete (*Storage_helper_info :: record()*, *File :: string()*) → Result

- **ErrorDetail:** term()
- **ErrorGeneral:** atom()
- **Result:** ok | {ErrorGeneral, ErrorDetail}

Deletes file (operates only on storage). First it checks if file exists and is regular file. If everything is ok, it deletes file.

delete_dir (*Storage_helper_info :: record()*, *Dir :: string()*) → Result

- **ErrorDetail:** term()
- **ErrorGeneral:** atom()
- **Result:** ok | {ErrorGeneral, ErrorDetail}

Deletes dir on storage

ls () → {error, not_implemented_yet}

Lists files in directory on storage

mkdir (*Storage_helper_info :: record()*, *Dir :: string()*) → Result

- **ErrorDetail:** term()
- **ErrorGeneral:** atom()
- **Result:** ok | {ErrorGeneral, ErrorDetail}

Creates dir on storage

mv (*Storage_helper_info :: record()*, *From :: string()*, *To :: string()*) → Result

- **ErrorDetail:** term()
- **ErrorGeneral:** atom()
- **Result:** ok | {ErrorGeneral, ErrorDetail}

Moves file on storage

read (*Storage_helper_info :: record()*, *File :: string()*, *Offset :: integer()*, *Size :: integer()*) → Result

- **Bytes:** binary()
- **ErrorDetail:** term()
- **ErrorGeneral:** atom()
- **Result:** {ok, Bytes} | {ErrorGeneral, ErrorDetail}

Reads file (operates only on storage). First it checks file attributes (file type and file size). If everything is ok, it reads data from file.

truncate (*Storage_helper_info :: record()*, *File :: string()*, *Size :: integer()*) → Result

- **ErrorDetail:** term()
- **ErrorGeneral:** atom()
- **Result:** ok | {ErrorGeneral, ErrorDetail}

Truncates file (operates only on storage). First it checks if file exists and is regular file. If everything is ok, it truncates file.

write (*Storage_helper_info :: record()*, *File :: string()*, *Buf :: binary()*) → Result

- **BytesWritten:** integer()
- **ErrorDetail:** term()
- **ErrorGeneral:** atom()
- **Result:** BytesWritten | {ErrorGeneral, ErrorDetail}

Appends data to the end of file (operates only on storage). First it checks file attributes (file type and file size). If everything is ok, it reads data from file.

write (*Storage_helper_info :: record()*, *File :: string()*, *Offset :: integer()*, *Buf :: binary()*) → Result

- **BytesWritten:** integer()
- **ErrorDetail:** term()
- **ErrorGeneral:** atom()
- **Result:** BytesWritten | {ErrorGeneral, ErrorDetail}

Writes data to file (operates only on storage). First it checks file attributes (file type and file size). If everything is ok, it reads data from file.

veilhelpers

Authors Rafal Slota

Copyright This software is released under the *MIT License*.

Descritpion This module shoul be used as an proxy to veilhelpers_nif module. This module controls way of accessing veilhelpers_nif methods.

Function Index

- *exec/2*
- *exec/3*

Function Details

exec (*Method :: atom()*, *SHInfo :: #storage_helper_info{}*[, *Arg :: term()*]) → {error, Reason :: term()} | Response

- **Response:** term()

Executes apply(veilhelper_nif, Method, Args) through slave node. Before executing, fields from struct SHInfo are preappend to Args list. You can also skip SHInfo argument in order to pass exact Args into target Method.

veilhelpers_nif

Authors Rafal Slota

Copyright This software is released under the *MIT License*.

Descritpion This module wraps storage helper's methods using NIF driver

Function Index

- *access/4*
- *chmod/4*
- *chown/5*
- *chown_name/5*
- *fsync/5*
- *get_flag/3*
- *getattr/3*
- *is_dir/3*
- *is_reg/3*
- *mkdir/4*

- *mknod/5*
- *open/4*
- *read/5*
- *read/6*
- *release/4*
- *rename/4*
- *rmdir/3*
- *start/1*
- *statfs/3*
- *truncate/4*
- *unlink/3*
- *write/5*
- *write/6*

Function Details

access (*_sh_name :: string(), _sh_args :: [string()], _path :: string(), _mask :: integer()*) → ErrorCode :: integer() | {error, 'NIF_not_loaded'}

First 2 arguments of this method should come from #storage_helper_info{} record. Those two arguments decide which Storage Helper shall be used for this operation. ErrorCode return value shall be 0 if operation was succesfull, otherwise negated POSIX error code will be returned. access/4 checks if the calling process has specified by _mask premissions to file with given _path. Most storage helpers will always return 0 (success), therefore this method can be used only to check if calling process does NOT have permissions to the file.

chmod (*_sh_name :: string(), _sh_args :: [string()], _path :: string(), _mode :: integer()*) → ErrorCode :: integer() | {error, 'NIF_not_loaded'}

First 2 arguments of this method should come from #storage_helper_info{} record. Those two arguments decide which Storage Helper shall be used for this operation. ErrorCode return value shall be 0 if operation was succesfull, otherwise negated POSIX error code will be returned. chmod/4 changes file's _mode.

chown (*_sh_name :: string(), _sh_args :: [string()], _path :: string(), _uid :: integer(), _gid :: integer()*) → ErrorCode :: integer() | {error, 'NIF_not_loaded'}

First 2 arguments of this method should come from #storage_helper_info{} record. Those two arguments decide which Storage Helper shall be used for this operation. ErrorCode return value shall be 0 if operation was succesfull, otherwise negated POSIX error code will be returned. chown/5 changes file's uid and gid

chown_name (*_sh_name :: string(), _sh_args :: [string()], _path :: string(), _uname :: string(), _gname :: string()*) → ErrorCode :: integer() | {error, 'NIF_not_loaded'}

First 2 arguments of this method should come from #storage_helper_info{} record. Those two arguments decide which Storage Helper shall be used for this operation. ErrorCode return value shall be 0 if operation was succesfull, otherwise negated POSIX error code will be returned. chown/5 changes file's uid and gid

fsync (*_sh_name :: string(), _sh_args :: [string()], _path :: string(), _isdatasync :: integer(), _fi :: #st_fuse_file_info{}*) → ErrorCode :: integer() | {error, 'NIF_not_loaded'}

First 2 arguments of this method should come from #storage_helper_info{} record. Those two arguments decide which Storage Helper shall be used for this operation. ErrorCode return value shall be 0 if operation was succesfull, otherwise negated POSIX error code will be returned.

getattr (*_sh_name :: string(), _sh_args :: [string()], _path :: string()*) → {ErrorCode :: integer(), Stats :: #st_stat{}} | {error, 'NIF_not_loaded'}

First 2 arguments of this method should come from #storage_helper_info{} record. Those two arguments decide which Storage Helper shall be used for this operation. ErrorCode return value shall be 0 if operation was succesfull, otherwise negated POSIX error code will be returned. gettattr/3 returns #st_stat{} record for given file _path. Note that if ErrorCode does not equal 0, fields of #st_stat{} are undefined and shall be ignored.

mkdir (*_sh_name :: string(), _sh_args :: [string()], _path :: string(), _mode :: integer()*) → ErrorCode :: integer() | {error, 'NIF_not_loaded'}

First 2 arguments of this method should come from #storage_helper_info{} record. Those two arguments decide which Storage Helper shall be used for this operation. ErrorCode return value shall be 0 if operation was succesfull, otherwise negated POSIX error code will be returned. mkdir creates directory with given _path and _mode (permissions).

mknod (*_sh_name :: string(), _sh_args :: [string()], _path :: string(), _mode :: integer(), _rdev :: integer()*) → ErrorCode :: integer() | {error, 'NIF_not_loaded'}

First 2 arguments of this method should come from #storage_helper_info{} record. Those two arguments decide which Storage Helper shall be used for this operation. ErrorCode return value shall be 0 if operation was succesfull, otherwise negated POSIX error code will be returned. mknod/5 can and shall be used in order to create file (not directory). _mode and _rdev arguments are the same as in mknod syscall.

open (*_sh_name :: string(), _sh_args :: [string()], _path :: string(), _fi :: #st_fuse_file_info{}*) → {ErrorCode :: integer(), FFI :: #st_fuse_file_info{}} | {error, 'NIF_not_loaded'}

First 2 arguments of this method should come from #storage_helper_info{} record. Those two arguments decide which Storage Helper shall be used for this operation. ErrorCode return value shall be 0 if operation was succesfull, otherwise negated POSIX error code will be returned. open/4 opens file. _fi aregument is an #st_fuse_file_info record that shall contain open flags. Same record will be returnd with 'fd' field set (file descriptor), therefore record returned by 'open' shall be passed to next read/write/release calls.

read (*_sh_name :: string(), _sh_args :: [string()], _path :: string(), _size :: integer(), _offset :: integer(), _fi :: #st_fuse_file_info{}*) → {ErrorCode :: integer(), Data :: binary()} | {error, 'NIF_not_loaded'}

First 2 arguments of this method should come from #storage_helper_info{} record. Those two arguments decide which Storage Helper shall be used for this operation. ErrorCode return value equals to bytes read count if operation was succesfull, otherwise negated POSIX error code will be returned. read/6 reads _size bytes (starting with _offset) from given file. If the _fi arguemnt is given with valid file descriptor ('fd' field) the 'fd' will be used to access file. Otherwise read/6 will open file for you.

release (*_sh_name :: string(), _sh_args :: [string()], _path :: string(), _fi :: #st_fuse_file_info{}*) → ErrorCode :: integer() | {error, 'NIF_not_loaded'}

First 2 arguments of this method should come from #storage_helper_info{} record. Those two arguments decide which Storage Helper shall be used for this operation. ErrorCode re-

turn value shall be 0 if operation was succesfull, otherwise negated POSIX error code will be returned. release/4 closes file that was previously opened with open/4.

rename (*_sh_name :: string(), _sh_args :: [string()], _from :: string(), _to :: string()*) → ErrorCode :: integer() | {error, 'NIF_not_loaded'}

First 2 arguments of this method should come from #storage_helper_info{} record. Those two arguments decide which Storage Helper shall be used for this operation. ErrorCode return value shall be 0 if operation was succesfull, otherwise negated POSIX error code will be returned. rename/4 shall be used to rename/move file from _from path to _to path.

rmdir (*_sh_name :: string(), _sh_args :: [string()], _path :: string()*) → ErrorCode :: integer() | {error, 'NIF_not_loaded'}

First 2 arguments of this method should come from #storage_helper_info{} record. Those two arguments decide which Storage Helper shall be used for this operation. ErrorCode return value shall be 0 if operation was succesfull, otherwise negated POSIX error code will be returned. rmdir removes directory with given _path.

start (*Prefix :: string()*) → ok | {error, Reason :: term()}

This method loads NIF library into erlang VM. This should be used once before using any other method in this module.

statfs (*_sh_name :: string(), _sh_args :: [string()], _path :: string()*) → {ErrorCode :: integer(), #st_statvfs{}} | {error, 'NIF_not_loaded'}

First 2 arguments of this method should come from #storage_helper_info{} record. Those two arguments decide which Storage Helper shall be used for this operation. ErrorCode return value shall be 0 if operation was succesfull, otherwise negated POSIX error code will be returned. statfs/3 returns #st_statvfs record for given _path. See statfs syscall for more details.

truncate (*_sh_name :: string(), _sh_args :: [string()], _path :: string(), _size :: integer()*) → ErrorCode :: integer() | {error, 'NIF_not_loaded'}

First 2 arguments of this method should come from #storage_helper_info{} record. Those two arguments decide which Storage Helper shall be used for this operation. ErrorCode return value shall be 0 if operation was succesfull, otherwise negated POSIX error code will be returned. truncate/4 changes file size to _size.

unlink (*_sh_name :: string(), _sh_args :: [string()], _path :: string()*) → ErrorCode :: integer() | {error, 'NIF_not_loaded'}

First 2 arguments of this method should come from #storage_helper_info{} record. Those two arguments decide which Storage Helper shall be used for this operation. ErrorCode return value shall be 0 if operation was succesfull, otherwise negated POSIX error code will be returned. unlink/3 removes given file (not directory).

write (*_sh_name :: string(), _sh_args :: [string()], _path :: string(), _buf :: binary(), _offset :: integer(), _fi :: #st_fuse_file_info{}*) → ErrorCode :: integer() | {error, 'NIF_not_loaded'}

First 2 arguments of this method should come from #storage_helper_info{} record. Those two arguments decide which Storage Helper shall be used for this operation. ErrorCode return value equals to bytes writen count if operation was succesfull, otherwise negated POSIX error code will be returned. write/6 writes _buf binary data to given file starting with _offset. _fi argument has the same meaning as in read/6.

Part III

MIT License