edeposit.amqp

Release 1.8.14

Sep 27, 2017

Contents

Welcome. This is documentation for the edeposit's AMQP module. The module is used in edeposit to provide and handle details of lowlevel AMQP communication.

Project can be found at GitHub: https://github.com/edeposit/edeposit.amqp

And installed from PYPI: https://pypi.python.org/pypi/edeposit.amqp

CHAPTER 1

edeposit.amqp package

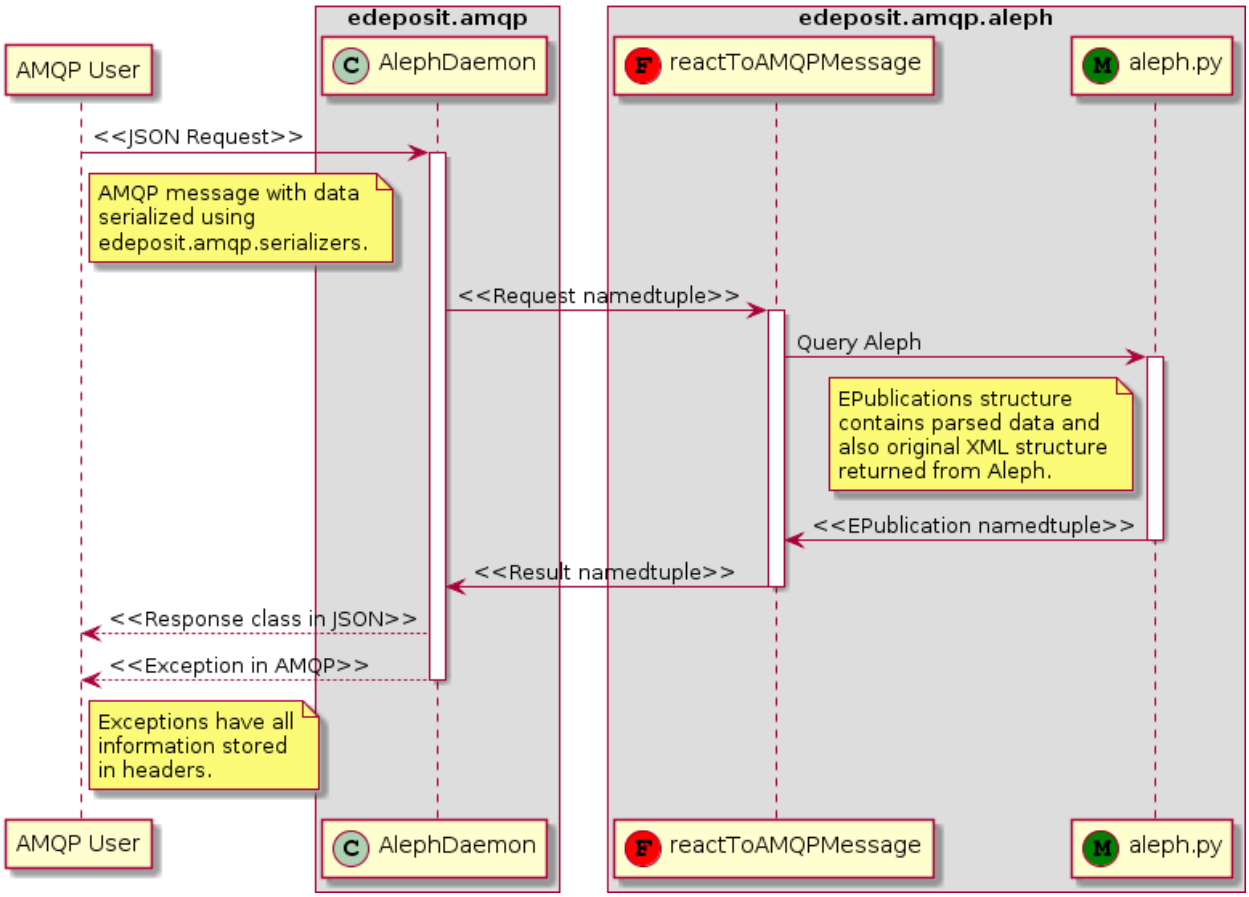
Purpose of this module is to provide class for launching Unix daemons (*daemonwrapper*), generict AMQP communication service based on RabbitMQ's pika library (*pikadaemon*), specific AMQP communication service for edeposit project (*amqpdaemon*) and also AMQP communication classes for sub-modules used in edeposit project:

- `edeposit_amqp_alephdaemon` (AMQP wrapper for the edeposit.amqp.aleph)
- `edeposit_amqp_calibredaemon` (AMQP wrapper for the edeposit.amqp.calibre)
- `edeposit_amqp_ftp_managerd` (user management AMQP wrapper for the edeposit.amqp.ftp)
- `edeposit_amqp_ftp_monitord` (AMQP binding for FTP event monitor - edeposit.amqp.ftp)
- `edeposit_amqp_antivirusd` (AMQP wrapper for the ClamAV daemon edeposit.amqp.antivirus)
- `edeposit_amqp_harvester` (AMQP wrapper for the harvester module edeposit.amqp.harvester)
- `edeposit_amqp_ltpd` (AMQP wrapper for the LTP export module edeposit.amqp.ltp)
- `edeposit_amqp_pdfgend` (AMQP binding for PDF generator edeposit.amqp.pdfgen)
- `edeposit_amqp_downloaderd` (AMQP binding for downloader edeposit.amqp.downloader)
- `edeposit_amqp_storaged` (AMQP binding for storage subsystem edeposit.amqp.storage)
- `edeposit_amqp_marcxml2modsd` (AMQP binding for marcxml2mods subsystem edeposit.amqp.marcxml2mods)
- `aleph_link_exportd` (AMQP binding for aleph_link_export subsystem edeposit.amqp.aleph_link_export)

For example `edeposit_amqp_alephdaemon` script allows you to send simple requests to get data from Aleph (system used in libraries all around the world) and in later versions also requests to put data into Aleph. Details of protocol and communication with Aleph server are handled by edeposit.amqp.aleph module.

Communication with sub-modules

From **user perspective**, communication is very similar to RPC - to each *Request* is returned *Response*.



Note: *Requests* and *Responses* are identified and paired by *UUID*, which is transported in headers of AMQP message.

Request

To send a request, you just need to send serialized structure (`collections.namedtuple()`) to the input queue of the daemon.

For example - for querying Aleph, take one of the Request classes, which are defined in aleph's `__init__.py`, into the RabbitMQ's exchange defined in `settings.RABBITMQ_ALEPH_EXCHANGE`.

Serialization can be done by calling `serialize()` function from edeposit.amqp.serializers.

Example showing how to send data to proper exchange:

```
import uuid

import settings
from alephdaemon import getConnectionParameters
from edeposit.amqp.serializers import serialize

connection = pika.BlockingConnection(alephdaemon.getConnectionParameters())
channel = connection.channel()

UUID = uuid.uuid4()  # this will be used to pair request with response

# put request together
json_data = serialize(
```

```
    aleph.SearchRequest(
        aleph.ISBNQuery("80-251-0225-4")
    )
)

# create properties of message - notice particularly the UUID parameter
properties = pika.BasicProperties(
    content_type="application/json",
    delivery_mode=1,
    headers={"UUID": str(UUID)}
)

# send the message to proper exchange with proper routing key
channel.basic_publish(
    exchange=settings.RABBITMQ_ALEPH_EXCHANGE,
    routing_key=settings.RABBITMQ_ALEPH_INPUT_KEY,
    properties=properties,
    body=json_data
)
```

It looks kinda long, but it is really simple and the most important thing in respect to communication with module is:

```
from edeposit.amqp.serializers import serialize

json_data = serialize(
    aleph.SearchRequest(
        aleph.ISBNQuery("80-251-0225-4")
    )
)
```

Here you say, that you want to perform *SearchRequest* and specifically search for ISBN.

Another important thing is to send and save for later use the *UUID*. You want to do this to be able to pair the response with request.

Warning: Messages received without *UUID* are thrown away without any warning.

Note: Notice also the *routing_key* parameter of `channel.basic_publish()`. It is used to determine into which queue will be message delivered.

Response

Response message is sent into `settings.RABBITMQ_ALEPH_EXCHANGE` with routing key `settings.RABBITMQ_ALEPH_OUTPUT_KEY`.

Format of response is usually one of the **Response* classes from `aleph.__init__.py` serialized to JSON, so you may need to `deserialize()` it. In headers, there should always be the *UUID* parameter, even in case of some unexpected error.

You can detect errors by looking for `exception` key in `parameters.headers` dictionary:

```
for method_frame, properties, body in self.channel.consume(self.queue):
    headers = properties.headers
    if "exception" in headers:
        print "There was an error in processing request ", headers["UUID"]
        print headers["exception_name"] + ": " + headers["exception"]
        break
```

Details of exception are contained in `exception`, `exception_name` and `exception_type` keys of the headers dictionary. First is text of error message, second is the `.__class__.__name__` property of exception and third is just output from `type(exception)`.

Programmers perspective

If you want to add new module, you will have to create your own instance of the *AMQPDaemon* and your module has to have some variant of the `reactToAMQP()` function. See *AMQPDaemon* doc-string for details.

Tips and tricks

Before you start sending the data, it is usually good idea to start the daemon. RabbitMQ will hold the data even when the daemon is not running, but you won't get the data back.

To start the daemon, run:

```
edeposit_amqp_alephdaemon.py start
```

This will start the proper unix daemon listening for the requests at RabbitMQ's message queue defined by *settings.RABBITMQ_ALEPH_INPUT_QUEUE*.

Note: Message queues, exchanges and routing keys have to be defined in RabbitMQ before you start the daemon.

If you don't want to define all details of AMQP communication by yourself, you can just run the *edeposit_amqp_tool.py*, which can build the schema:

```
edeposit_amqp_tool.py --host ftp --create
```

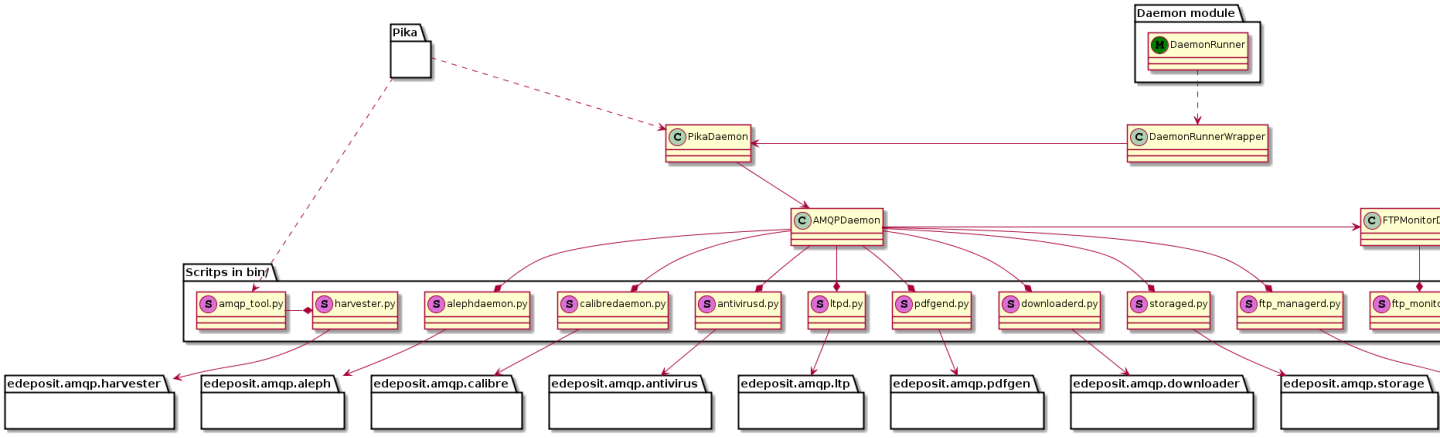
output example:

```
Created exchange 'ftp'.
Creating queues:
  Created durable queue 'daemon'.
  Created durable queue 'plone'.

Routing exchanges using routing key to queues:
  Routing exchange ftp['request'] -> 'daemon'.
  Routing exchange ftp['result'] -> 'plone'.
```

Project structure

Here is pseudo-UML picture documenting relations between module components.



List of submodules

DaemonRunnerWrapper

Module for creating generic, callback based wrappers.

It is little bit easier (at least for me) to use than the original daemon module.

class `edeposit.amqp.daemonwrapper.`**`DaemonRunnerWrapper`**(*pid_filename*)

Bases: `object`

Generic daemon class, which allows you to daemonize your script and react to events in simple callbacks.

Parameters **`pid_filename`** (*`str`*) – name of daemon's PID file, which is stored in `/tmp`. Class automatically adds `.pid` suffix.

`body`()

Here should be your code loop.

Note: Loop is automatically break-ed when daemon receives one of the unix signals. After that, *`onExit()`* is called.

`isRunning`()

Check PID and return true, if it looks like there is already a running instance of daemon.

PID timeout can be set thru `pidfile_timeout` property.

`onExit`()

Called when the daemon received ?SIGTERM? and is shutting down.

Warning: You should probably put something here, by default is there only shutdown message "DaemonRunnerWrapper is shutting down."

`onIsRunning`()

Oposite of *`onStopFail()`* - this callback is called if there is already a running instance of daemon.

`onStopFail`()

Called when it is not possible to stop the daemon.

This kind of event typically occurs if there is no running instance of daemon and script is called with `stop` parameter.

run()
Used to handle some exceptions.

Note: No, this can't be named differently, because it is defined in original DaemonRunner object.

Warning: DO NOT OVERRIDE THIS.

run_daemon()
Used as daemon starter.

Warning: DO NOT OVERRIDE THIS.

PikaDaemon

Generic AMQP blocking communication daemon server.

Usage is simple - just inherit the class and override *PikaDaemon.onMessageReceived()*.

You can send messages back using either *PikaDaemon.sendMessage()* or *PikaDaemon.sendResponse()*. Fist one allows you to send message everywhere, second one send message to the queue defined by constructor.

class edeposit.amqp.pikadaemon.**PikaDaemon**(*connection_param*, *queue*, *output_exchange*, *output_key*)
Bases: *edeposit.amqp.daemonwrapper.DaemonRunnerWrapper*

Pika and Daemon wrapper for handling AMQP connections.

Parameters

- **connection_param** (*pika.ConnectionParameters*) – object setting the connection
- **queue** (*str*) – name of queue where the daemon should listen
- **output_exchange** (*str*) – name of exchange where the daemon should put responses
- **output_key** (*str*) – routing key for output exchange

ack(*ack_delivery_tag*)
Acknowledge, that message was received.

Note: This will in some cases (depends on settings of RabbitMQ) remove the message from the message queue.

body()
This method just handles AMQP connection details and receive loop.

Warning: Don't override this method!

onExit()
Called when daemon is stopped. Basically just AMQP's `.close()` functions to ensure clean exit.

You can override this, but don't forget to call it thru `super()`, or the AMQP communication won't be closed properly!

onMessageReceived(*method_frame*, *properties*, *body*)
Callback which is called every time when message is received.

Warning: You SHOULD override this.

Note: It is expected, that method returns True, if you want to automatically ack the received message, which can be important in some situations, because otherwise the message will be held in message queue until someone will ack it.

You don't have to return True/False - you can ack the message yourself, by calling *ack()*.

Note: Good design choice is to ack the message AFTER you process it, to be sure, that message is processed properly and can be removed from queue.

sendMessage(*exchange*, *routing_key*, *message*, *properties=None*, *UUID=None*)
With this function, you can send message to *exchange*.

Parameters

- **exchange** (*str*) – name of exchange you want to message to be delivered
- **routing_key** (*str*) – which routing key to use in headers of message
- **message** (*str*) – body of message
- **properties** (*dict ,optional*) – properties of message - if not used, or set to `None`, `self.content_type` and `delivery_mode=2` (persistent) is used
- **UUID** (*str, optional*) – UUID of the message. If set, it is included into `properties` of the message.

sendResponse(*message*, *UUID*, *routing_key*)
Send *message* to `self.output_exchange` with routing key `self.output_key`, `self.content_type` in `delivery_mode=2`.

Parameters

- **message** (*str*) – message which will be sent
- **UUID** – unique identification of message
- **routing_key** (*str*) – which routing key to use to send message back

AMQPDaemon

This module provides generic AMQP daemon and builder of common connection informations, which are defined as constants in *edeposit.amqp.settings*.

Daemon is used by `edeposit_amqp_alephdaemon` and `edeposit_amqp_calibredaemon`.

class `edeposit.amqp.amqpdaemon.`**AMQPDaemon**(*con_param*, *queue*, *out_exch*, *out_key*, *react_fn*, *glob*)

Bases: *edeposit.amqp.pikadaemon.PikaDaemon*

Parameters

- **con_param** (*ConnectionParameters*) – see *getConParams()* for details
- **queue** (*str*) – name of the queue
- **out_exch** (*str*) – name of the exchange for outgoing messages
- **out_key** (*str*) – what key will be used to send messages back
- **react_fn** (*fn*) – function, which can react to messages, see Note for details
- **glob** (*dict*) – result of `globals()` call - used in deserializer to automatically build classes, which are not available in this namespace of this package

---

Note: `react_fn` parameter is expected to be function, which gets two parameters - *message* (some form of message, it can be also namedtuple), and *UUID* containing unique identificator of the message.

Example of function used as *react_fn* parameter:

```
def reactToAMQPMessage(message, UUID):
    response = None
    if message == 1:
        return 2
    elif message == "Hello":
        return "Hi"
    elif type(message) == dict:
        return {1: 2}

    raise UserWarning("Unrecognized message")
```

As you can see, protocol is pretty easy. You get *message*, to which you react somehow and return *response*. Thats all.

get_sendback(*uuid*, *key*)

Return function for sending progress messages back to original caller.

Parameters

- **uuid** (*str*) – UUID of the received message.
- **key** (*str*) – Routing key.

Returns Reference to function which takes only one data argument.

Return type fn reference

onMessageReceived(*method_frame*, *properties*, *body*)

React to received message - deserialize it, add it to users reaction function stored in `self.react_fn` and send back result.

If *Exception* is thrown during process, it is sent back instead of message.

Note: In case of *Exception*, response message doesn't have useful *body*, but in headers is stored following (string) parameters:

- `exception`, where the Exception's message is stored
- `exception_type`, where `e.__class__` is stored
- `exception_name`, where `e.__class__.__name__` is stored
- `traceback` where the full traceback is stored (contains line number)

This allows you to react to unexpected cases at the other end of the AMQP communication.

---

parseKey(*method_frame*)

process_exception(*e*, *uuid*, *routing_key*, *body*, *tb=None*)
Callback called when exception was raised.

This method serializes the exception and sends it over AMQP back to caller.

Parameters

- **e** (*obj*) – Instance of the exception.
- **uuid** (*str*) – UUID of the message that caused the exception to raise.
- **routing_key** (*str*) – Which routing key was used.
- **body** (*str*) – Body of the exception - the longer text.
- **tb** (*str, default None*) – Traceback (stacktrace)v of the exception.

edeposit.amqp.amqpdaemon.**getConParams**(*virtualhost*)
Connection object builder.

Parameters **virtualhost** (*str*) – selected virtualhost in rabbitmq

Returns object filled by *constants* from *edeposit.amqp.settings*.

Return type pika.ConnectionParameters

Settings

Module is containing all necessary global variables for package.

Module also has ability to read user-defined data from two paths: $HOME/*SETTINGS_PATH* and /etc/*SETTINGS_PATH*.

Note: If the first path is found, other is ignored.

Example of the configuration file (`$HOME/edeposit/amqp.json`):

```
{
    "RABBITMQ_HOST": "localhost",
    "RABBITMQ_PORT": "2222"
}
```

Attributes

edeposit.amqp.settings.**RABBITMQ_ALEPH_EXCEPTION_KEY = 'exception'**

edeposit.amqp.settings.**RABBITMQ_ALEPH_EXCHANGE = 'search'**

edeposit.amqp.settings.**RABBITMQ_ALEPH_INPUT_KEY = 'request'**

edeposit.amqp.settings.**RABBITMQ_ALEPH_INPUT_QUEUE = 'daemon'**

edeposit.amqp.settings.**RABBITMQ_ALEPH_LINK_EXPORT_INPUT_KEY = 'request'**

edeposit.amqp.settings.**RABBITMQ_ALEPH_LINK_EXPORT_INPUT_QUEUE = 'updater'**
Input Queue

edeposit.amqp.settings.**RABBITMQ_ALEPH_LINK_EXPORT_OUTPUT_KEY = 'result'**

edeposit.amqp.settings.**RABBITMQ_ALEPH_LINK_EXPORT_OUTPUT_QUEUE = 'plone'**
Output Queue

edeposit.amqp.settings.**RABBITMQ_ALEPH_LINK_EXPORT_VIRTUALHOST = 'aleph'**
Virtualhost

edeposit.amqp.settings.**RABBITMQ_ALEPH_OUTPUT_KEY = 'result'**

edeposit.amqp.settings.**RABBITMQ_ALEPH_OUTPUT_QUEUE = 'plone'**

edeposit.amqp.settings.**RABBITMQ_ALEPH_VIRTUALHOST = 'aleph'**

edeposit.amqp.settings.**RABBITMQ_ANTIVIRUS_INPUT_KEY = 'request'**

edeposit.amqp.settings.**RABBITMQ_ANTIVIRUS_INPUT_QUEUE = 'daemon'**
Input Queue for AV AMQP daemon

edeposit.amqp.settings.**RABBITMQ_ANTIVIRUS_OUTPUT_KEY = 'result'**

edeposit.amqp.settings.**RABBITMQ_ANTIVIRUS_OUTPUT_QUEUE = 'plone'**
Queue to put responses

edeposit.amqp.settings.**RABBITMQ_ANTIVIRUS_VIRTUALHOST = 'antivirus'**
Virtualhost for AV module

edeposit.amqp.settings.**RABBITMQ_CALIBRE_EXCHANGE = 'convert'**

edeposit.amqp.settings.**RABBITMQ_CALIBRE_INPUT_KEY = 'request'**

edeposit.amqp.settings.**RABBITMQ_CALIBRE_INPUT_QUEUE = 'daemon'**

edeposit.amqp.settings.**RABBITMQ_CALIBRE_OUTPUT_KEY = 'result'**

edeposit.amqp.settings.**RABBITMQ_CALIBRE_OUTPUT_QUEUE = 'plone'**

edeposit.amqp.settings.**RABBITMQ_CALIBRE_VIRTUALHOST = 'calibre'**

edeposit.amqp.settings.**RABBITMQ_DOWNER_INPUT_KEY = 'request'**

edeposit.amqp.settings.**RABBITMQ_DOWNER_INPUT_QUEUE = 'daemon'**
Input Queue for downloader

edeposit.amqp.settings.**RABBITMQ_DOWNER_OUTPUT_KEY = 'result'**

edeposit.amqp.settings.**RABBITMQ_DOWNER_OUTPUT_QUEUE = 'plone'**
Queue to put responses

edeposit.amqp.settings.**RABBITMQ_DOWNER_VIRTUALHOST = 'downloader'**
Virtualhost for downloader

edeposit.amqp.settings.**RABBITMQ_FTP_INPUT_KEY = 'request'**

edeposit.amqp.settings.**RABBITMQ_FTP_INPUT_QUEUE = 'daemon'**
Input Queue for FTP AMQP daemon

edeposit.amqp.settings.**RABBITMQ_FTP_OUTPUT_KEY = 'result'**

edeposit.amqp.settings.**RABBITMQ_FTP_OUTPUT_QUEUE = 'plone'**
Queue to put responses from daemon

edeposit.amqp.settings.**RABBITMQ_FTP_VIRTUALHOST = 'ftp'**
Virtualhost for FTP module

edeposit.amqp.settings.**RABBITMQ_HARVESTER_INPUT_KEY = 'request'**

edeposit.amqp.settings.**RABBITMQ_HARVESTER_INPUT_QUEUE = 'daemon'**
Input Queue for harvester

edeposit.amqp.settings.**RABBITMQ_HARVESTER_OUTPUT_KEY = 'result'**

edeposit.amqp.settings.**RABBITMQ_HARVESTER_OUTPUT_QUEUE = 'plone'**
Queue to put responses

edeposit.amqp.settings.**RABBITMQ_HARVESTER_VIRTUALHOST = 'harvester'**
Virtualhost for harvester

edeposit.amqp.settings.**RABBITMQ_HOST = '127.0.0.1'**

edeposit.amqp.settings.**RABBITMQ_LTP_INPUT_KEY = 'request'**

edeposit.amqp.settings.**RABBITMQ_LTP_INPUT_QUEUE = 'daemon'**
Input Queue for ltp

edeposit.amqp.settings.**RABBITMQ_LTP_OUTPUT_KEY = 'result'**

edeposit.amqp.settings.**RABBITMQ_LTP_OUTPUT_QUEUE = 'plone'**
Queue to put responses

edeposit.amqp.settings.**RABBITMQ_LTP_VIRTUALHOST = 'ltp'**
Virtualhost for ltp

edeposit.amqp.settings.**RABBITMQ_MX2MODS_INPUT_KEY = 'request'**

edeposit.amqp.settings.**RABBITMQ_MX2MODS_INPUT_QUEUE = 'daemon'**
Input Queue for marcxml2mods

edeposit.amqp.settings.**RABBITMQ_MX2MODS_OUTPUT_KEY = 'result'**

edeposit.amqp.settings.**RABBITMQ_MX2MODS_OUTPUT_QUEUE = 'plone'**
Queue to put responses

edeposit.amqp.settings.**RABBITMQ_MX2MODS_VIRTUALHOST = 'marcxml2mods'**
Virtualhost for marcxml2mods

edeposit.amqp.settings.**RABBITMQ_PDFGEN_INPUT_KEY = 'request'**

edeposit.amqp.settings.**RABBITMQ_PDFGEN_INPUT_QUEUE = 'daemon'**
Input Queue for pdfgen

edeposit.amqp.settings.**RABBITMQ_PDFGEN_OUTPUT_KEY = 'result'**

edeposit.amqp.settings.**RABBITMQ_PDFGEN_OUTPUT_QUEUE = 'plone'**
Queue to put responses

edeposit.amqp.settings.**RABBITMQ_PDFGEN_VIRTUALHOST = 'pdfgen'**
Virtualhost for pdfgen

edeposit.amqp.settings.**RABBITMQ_PORT = '5672'**

edeposit.amqp.settings.**RABBITMQ_STORAGE_INPUT_KEY = 'request'**

edeposit.amqp.settings.**RABBITMQ_STORAGE_INPUT_QUEUE = 'daemon'**
Input Queue for storage

edeposit.amqp.settings.**RABBITMQ_STORAGE_OUTPUT_KEY = 'result'**

edeposit.amqp.settings.**RABBITMQ_STORAGE_OUTPUT_QUEUE = 'plone'**
Queue to put responses

edeposit.amqp.settings.**RABBITMQ_STORAGE_VIRTUALHOST = 'storage'**
Virtualhost for storage

edeposit.amqp.settings.**RABBITMQ_USER_NAME = 'guest'**

edeposit.amqp.settings.**RABBITMQ_USER_PASSWORD = 'guest'**

edeposit.amqp.settings.**SETTINGS_PATH = '/edeposit/amqp.json'**
Path which is appended to default search paths (`$HOME` and `/etc`).

Note: It has to start with `/`. Variable is **appended** to the default search paths, so this doesn't mean, that the path is absolute!

---

edeposit.amqp.settings.**get_all_constants**()
Get list of all uppercase, non-private globals (doesn't start with _).

Returns Uppercase names defined in *globals()* (variables from this module).

Return type list

edeposit.amqp.settings.**get_amqp_settings**()
Return all settings in dict in following format:

```
{
    "submodule_name": {
        "vhost": VIRTUALHOST,
        "exchange": EXCHANGE,
        "queues": {
            QUEUE_NAME: ROUTING_KEY,
            QUEUE_NAME: ROUTING_KEY
        },
        "in_key": INPUT_KEY,
        "out_key": OUTPUT_KEY
    },
    ...
}
```

edeposit.amqp.settings.**substitute_globals**(*config_dict*)
Set global variables to values defined in *config_dict*.

Parameters **config_dict** (*dict*) – dictionary with data, which are used to set *globals*.

Note: *config_dict* have to be dictionary, or it is ignored. Also all variables, that are not already in globals, or are not types defined in `_ALLOWED` (str, int, float) or starts with _ are silently ignored.

List of scripts

Following scrips are installed to user's `/bin` directory and therefore can be called by simply typing their name to the shell:

edeposit_amqp_alephdaemon.py script

Help

```
$ ./edeposit_amqp_alephdaemon.py -husage: edeposit_amqp_alephdaemon.py start/stop/
↪restart [-f] FN

Aleph communicator. This daemon provides AMQP API for aleph module.

positional arguments:
  start/stop/restart  Start/stop/restart the daemon.

optional arguments:
  -h, --help          show this help message and exit
  -f, --foreground    Run at foreground, not as daemon. If not set, script is
                      will run at background as unix daemon.
```

Example usage:

```
./edeposit_amqp_alephdaemon.py start
started with pid 4595
```

or:

```
$ ./edeposit_amqp_alephdaemon.py start --foreground
```

In this case, the script runs as normal program, and it is not daemonized.

Stopping:

```
$ ./edeposit_amqp_alephdaemon.py stop
No handlers could be found for logger "pika.adapters.base_connection"
```

Don't be concerned by warnings when stopping the daemon, it is just something that out communication library does.

edeposit_amqp_calibredaemon.py script

Help

```
$ ./edeposit_amqp_calibredaemon.py -h
usage: edeposit_amqp_calibredaemon.py start/stop/restart [-f] FN

Calibre daemon, providing AMQP interface for edeposit.amqp.calibre.

positional arguments:
  start/stop/restart  Start/stop/restart the daemon.

optional arguments:
  -h, --help          show this help message and exit
  -f, --foreground    Run at foreground, not as daemon. If not set, script is
                      will run at background as unix daemon.
```

Example usage:

```
$ ./edeposit_amqp_calibredaemon.py start
started with pid 10005
```

or:

```
$ ./edeposit_amqp_calibredaemon.py start --foreground
```

In this case, the script runs as normal program, and it is not daemonized.

Stopping:

```
$ ./edeposit_amqp_calibredaemon.py stop
No handlers could be found for logger "pika.adapters.base_connection"
```

Don't be concerned by warnings when stopping the daemon, it is just something that out communication library does.

edeposit_amqp_ftp_monitord.py script

Help

```
$ ./edeposit_amqp_ftp_managerd.py -h
usage: edeposit_amqp_ftp_managerd.py start/stop/restart [-f] FN

ProFTPD user manager. This script allows you to add/ remove/change users in
ProFTPD server over AMQP API.

positional arguments:
  start/stop/restart  Start/stop/restart the daemon.

optional arguments:
  -h, --help          show this help message and exit
  -f, --foreground    Run at foreground, not as daemon. If not set, script is
                      will run at background as unix daemon
```

Example usage:

```
./edeposit_amqp_ftp_managerd.py start
started with pid 5469
```

or:

```
$ ./edeposit_amqp_ftp_managerd.py start --foreground
```

In this case, the script runs as normal program, and it is not daemonized.

Stopping:

```
$ ./edeposit_amqp_ftp_managerd.py stop
No handlers could be found for logger "pika.adapters.base_connection"
```

Don't be concerned by warnings when stopping the daemon, it is just something that out communication library does.

edeposit_amqp_ftp_monitord.py script

Help

```
./edeposit_amqp_ftp_monitord.py -h
usage: edeposit_amqp_ftp_monitord.py start/stop/restart [-f] FN

ProFTPD log monitor. This script reacts to preprogrammed events from FTP
server.

positional arguments:
  start/stop/restart    Start/stop/restart the daemon.

optional arguments:
  -h, --help            show this help message and exit
  -f, --foreground      Run at foreground, not as daemon. If not set, script
                        is will run at background as unix daemon.
  -n FILENAME, --filename FILENAME
                        Path to the log file (usually
                        /var/log/proftpd/extended.log).
```

Example usage:

```
$ ./edeposit_amqp_ftp_monitord.py start --filename /var/log/proftpd/extended.log
Monitoring file '/var/log/proftpd/extended.log'.
started with pid 1633
```

or:

```
$ ./edeposit_amqp_ftp_monitord.py start --filename /var/log/proftpd/extended.log --
↪foreground
Monitoring file '/var/log/proftpd/extended.log'.
```

In this case, the script runs as normal program, and it is not daemonized.

Stopping:

```
$ ./edeposit_amqp_ftp_monitord.py stop
WARNING:pika.adapters.base_connection:Unknown state on disconnect: 0
WARNING:pika.adapters.base_connection:Unknown state on disconnect: 0
```

Don't be concerned by warnings when stopping the daemon, it is just something that out communication library does.

edeposit_amqp_antivirusd.py script

Help

```
$ ./edeposit_amqp_antivirusd.py -h
usage: edeposit_amqp_antivirusd.py start/stop/restart [-f] FN

AMQP daemon for ClamAV antivirus.

positional arguments:
  start/stop/restart  Start/stop/restart the daemon.

optional arguments:
  -h, --help          show this help message and exit
  -f, --foreground    Run at foreground, not as daemon. If not set, script is
                      will run at background as unix daemon.
```

Example usage:

```
./edeposit_amqp_antivirusd.py start
started with pid 7195
```

or:

```
$ ./edeposit_amqp_antivirusd.py start --foreground
```

In this case, the script runs as normal program, and it is not daemonized.

Stopping:

```
$ ./edeposit_amqp_antivirusd.py stop
No handlers could be found for logger "pika.adapters.base_connection"
```

Don't be concerned by warnings when stopping the daemon, it is just something that out communication library does.

edeposit_amqp_harvester.py script

Help

```
./edeposit_amqp_harvester.py
usage: edeposit_amqp_harvester.py [-h] [-u] [-r]

This script is used to send data from edeposit.amqp.harvester to AMQP queue.

optional arguments:
  -h, --help      show this help message and exit
  -u, --unittest  Perform unittest.
  -r, --harvest   Harvest all data and send them to harvester queue.
```

Example usage:

```
$ ./edeposit_amqp_harvester.py start --harvest
```

Which will harvest all possible sources of publications and send them to AMQP, or:

```
$ ./edeposit_amqp_harvester.py --unittest
```

Which will perform unittests and send results over AQMP to proper exchange.

edeposit_amqp_ltpd.py script

Help

```
$ ./edeposit_amqp_ltpd.py -h
usage: edeposit_amqp_ltpd.py start/stop/restart [-f/--foreground]

AMQP daemon for LTP exporter.

positional arguments:
  start/stop/restart  Start/stop/restart the daemon.

optional arguments:
```

```
  -h, --help           show this help message and exit
  -f, --foreground     Run at foreground, not as daemon. If not set, script is
                       will run at background as unix daemon.
```

Example usage:

```
./edeposit_amqp_ltpd.py start
started with pid 7195
```

or:

```
$ ./edeposit_amqp_ltpd.py start --foreground
```

In this case, the script runs as normal program, and it is not daemonized.

Stopping:

```
$ ./edeposit_amqp_ltpd.py stop
No handlers could be found for logger "pika.adapters.base_connection"
```

Don't be concerned by warnings when stopping the daemon, it is just something that out communication library does.

edeposit_amqp_pdfgend.py script

Help

```
$ ./edeposit_amqp_pdfgend.py -h
usage: edeposit_amqp_pdfgend.py start/stop/restart [-f/--foreground]

AMQP binding for PDF generator.

positional arguments:
  start/stop/restart  Start/stop/restart the daemon.

optional arguments:
  -h, --help           show this help message and exit
  -f, --foreground     Run at foreground, not as daemon. If not set, script is
                       will run at background as unix daemon.
```

Example usage:

```
./edeposit_amqp_pdfgend.py start
started with pid 7195
```

or:

```
$ ./edeposit_amqp_pdfgend.py start --foreground
```

In this case, the script runs as normal program, and it is not daemonized.

Stopping:

```
$ ./edeposit_amqp_pdfgend.py stop
No handlers could be found for logger "pika.adapters.base_connection"
```

Don't be concerned by warnings when stopping the daemon, it is just something that out communication library does.

edeposit_amqp_downloaderd.py script

Help

```
$ ./edeposit_amqp_downloaderd.py -h
usage: edeposit_amqp_downloaderd.py start/stop/restart [-f/--foreground]

AMQP binding for downloader.

positional arguments:
  start/stop/restart  Start/stop/restart the daemon.

optional arguments:
  -h, --help          show this help message and exit
  -f, --foreground    Run at foreground, not as daemon. If not set, script is
                      will run at background as unix daemon.
```

Example usage:

```
./edeposit_amqp_downloaderd.py start
started with pid 7195
```

or:

```
$ ./edeposit_amqp_downloaderd.py start --foreground
```

In this case, the script runs as normal program, and it is not daemonized.

Stopping:

```
$ ./edeposit_amqp_downloaderd.py stop
No handlers could be found for logger "pika.adapters.base_connection"
```

Don't be concerned by warnings when stopping the daemon, it is just something that out communication library does.

edeposit_amqp_storaged.py script

..storaged

members

undoc-members

show-inheritance

Help

```
$ ./edeposit_amqp_storaged.py start/stop/restart [-f/--foreground]

AMQP binding for storage.

positional arguments:
  start/stop/restart  Start/stop/restart the daemon.

optional arguments:
  -h, --help          show this help message and exit
```

```
  -f, --foreground    Run at foreground, not as daemon. If not set, script is
                      will run at background as unix daemon.
```

Example usage:

```
./edeposit_amqp_storaged.py start
started with pid 7195
```

or:

```
$ ./edeposit_amqp_storaged.py start --foreground
```

In this case, the script runs as normal program, and it is not daemonized.

Stopping:

```
$ ./edeposit_amqp_storaged.py stop
No handlers could be found for logger "pika.adapters.base_connection"
```

Don't be concerned by warnings when stopping the daemon, it is just something that out communication library does.

edeposit_amqp_marcxml2modsd.py script

..marcxml2modsd

members

undoc-members

show-inheritance

Help

```
$ ./edeposit_amqp_marcxml2modsd.py marcxml2modsd.py start/stop/restart [-f/--
↪foreground]

AMQP binding for marcxml2mods.

positional arguments:
  start/stop/restart  Start/stop/restart the daemon.

optional arguments:
  -h, --help          show this help message and exit
  -f, --foreground    Run at foreground, not as daemon. If not set, script is
                      will run at background as unix daemon.
```

Example usage:

```
./edeposit_amqp_marcxml2modsd.py start
started with pid 7195
```

or:

```
$ ./edeposit_amqp_marcxml2modsd.py start --foreground
```

In this case, the script runs as normal program, and it is not daemonized.

Stopping:

```
$ ./edeposit_amqp_marcxml2modsd.py stop
No handlers could be found for logger "pika.adapters.base_connection"
```

Don't be concerned by warnings when stopping the daemon, it is just something that out communication library does.

edeposit_amqp_aleph_link_exportd.py script

..aleph_link_exportd

members

undoc-members

show-inheritance

Help

```
$ ./edeposit_amqp_aleph_link_exportd.py start/stop/restart [-f/--foreground]

AMQP binding for aleph_link_export.

positional arguments:
  start/stop/restart  Start/stop/restart the daemon.

optional arguments:
  -h, --help          show this help message and exit
  -f, --foreground    Run at foreground, not as daemon. If not set, script is
                      will run at background as unix daemon.
```

Example usage:

```
./edeposit_amqp_aleph_link_exportd.py start
started with pid 7195
```

or:

```
$ ./edeposit_amqp_aleph_link_exportd.py start --foreground
```

In this case, the script runs as normal program, and it is not daemonized.

Stopping:

```
$ ./edeposit_amqp_aleph_link_exportd.py stop
No handlers could be found for logger "pika.adapters.base_connection"
```

Don't be concerned by warnings when stopping the daemon, it is just something that out communication library does.

edeposit_amqp_tool.py script

AMQP tool used for debugging and automatic RabbitMQ schema making.

edeposit_amqp_tool.**create_blocking_connection**(**args*, ***kwargs*)

Return properly created blocking connection.

Parameters **host** (*str*) – Host as it is defined in *get_amqp_settings()*.

Uses *edeposit.amqp.amqpdaemon.getConParams()*.

edeposit_amqp_tool.**create_schema**(*host*)
Create exchanges, queues and route them.

Parameters **host** (*str*) – One of the possible hosts.

edeposit_amqp_tool.**get_list_of_hosts**()

Returns List of strings with names of possible hosts.

Return type list

edeposit_amqp_tool.**main**()

edeposit_amqp_tool.**receive**(*host*, *timeout*)
Print all messages in queue.

Parameters

- **host** (*str*) – Specified –host.
- **timeout** (*int*) – How log should script wait for message.

edeposit_amqp_tool.**send_message**(*host*, *data*, *timeout=None*, *properties=None*)
Send message to given *host*.

Parameters

- **host** (*str*) – Specified host: aleph/ftp/whatever available host.
- **data** (*str*) – JSON data.
- **timeout** (*int, default None*) – How much time wait for connection.

edeposit_amqp_tool.**test_virtualhost**(*fn*)

Indices and tables

- genindex
- modindex
- search

Python Module Index

e

Index

S

T