

---

# meta-app-script-py-sdk

## Documentation

*Выпуск stable*

ОКТ. 31, 2018



<b>1</b>	<b>Введение</b>	<b>3</b>
1.1	Install . . . . .	3
1.2	Full Examples . . . . .	3
1.3	Usage . . . . .	3
<b>2</b>	<b>Логирование</b>	<b>5</b>
2.1	Logger . . . . .	5
2.2	Bulk Logger . . . . .	6
2.3	Bulk Logger Thread Pool . . . . .	6
<b>3</b>	<b>RPC Meta Services</b>	<b>9</b>
3.1	SettingsService . . . . .	9
3.2	Примеры . . . . .	9
3.3	MediaService . . . . .	9
3.4	DbQueryService . . . . .	11
3.5	SettingsService . . . . .	13
3.6	IssueService . . . . .	13
3.7	UserManagementService . . . . .	13
3.8	StarterService . . . . .	14
3.9	MailService . . . . .	14
<b>4</b>	<b>METAQL</b>	<b>15</b>
4.1	Особенности . . . . .	15
4.2	Функции . . . . .	15
4.3	Примеры . . . . .	16



Contents:



### 1.1 Install

`pip3 install metaappscriptsdk --upgrade --no-cache`

Последний релиз для Python 2 = 0.1.3 Python 3 начинается с > 0.3.0

Получите файл токена разработчика Установите `developer_settings.json` в домашнюю директорию в папку `.rwmeta` Например:

- MacOS: `/Users/arturgspb/.rwmeta/developer_settings.json`
- Windows: `C:\Users\userXXXXXX.rwmeta\developer_settings.json`
- Linux: `~/rwmeta/developer_settings.json`

### 1.2 Full Examples

Полный список примеров

### 1.3 Usage

```
# coding=utf-8
import logging
import starter_api
from metaappscriptsdk import MetaApp

# Инициализация приложения
# конфигурирует логгер и пр.
МЕТА = MetaApp()
```

(continues on next page)

(продолжение с предыдущей страницы)

```
# работает стандартный логгер
logging.info('Hello, from Meta App Script!')
# Можно получить экземпляр логгера с улучшенным интерфейсом для более удобного прикладывания
# ↪ контекста
log = META.log
log.warning('Do warning log', {"count": 1, "mycontextParam": [1, 3, 4]})

# Постановка задач в Запускающ
starter_api.build_submit('YOUR_SERVICE')
# или
META.starter.build_submit('YOUR_SERVICE')
```



Тут мы рассмотрим обычный логгер и Bulk-логгер, который нужен, чтобы не спамить большим кол-вом логов, при обработке больших массивов данных с неизвестным временем на обработку одного элемента

## 2.1 Logger

```
log = META.log

# Объявите глобальный контекст, чтобы не писать это каждый раз
log.set_entity('campaign_id', -1)
# По сути это просто хранилище глобальных переменных контекста
log.set_entity('test', True)
log.warning('Do warning log', {"count": 1, "mycontextParam": [1, 3, 4]})
log.info('Info log')

logging.info('Default logging')

# удалите глобальный контекст, когда он вам больше не нужен
log.remove_entity('test')
log.info('Info log2')
```

Это выведет вам что-то вроде такого:

```
# 00:03:11:WARNING: Do warning log {'count': 1, 'mycontextParam': [1, 3, 4], 'test': True,
→ 'campaign_id': -1}
# 00:03:11:INFO: Info log {'test': True, 'campaign_id': -1}
# 00:03:11:INFO: Default logging {'test': True, 'campaign_id': -1}
# 00:03:11:INFO: Info log2 {'campaign_id': -1}
```

## 2.2 Bulk Logger

Используется для логирования пачек из обрабатываемого списка. Делает запись в лог только если прошло определенное кол-во времени

```
total = 125
# Получаете инстанс bulk-логгера через объект приложения
# Список параметров вам подскажет IDE
bulk_log = META.bulk_log(u'Моя пачка', total, 1)

for idx in xrange(total):
    # Первый вызов всегда try_log_part, чтобы озаглавить начало выполнения цикла
    bulk_log.try_log_part()

    # На ЧАСТЫХ, но возможно БЫСТРЫХ процессах можете использовать параметр with_start_
    ↪message=False
    # Это исключит запись надписи о начале работы над пачкой и, если пачка сделается до
    # мин. время логирования, то записи не произойдет вообще
    # bulk_log.try_log_part(with_start_message=False)

    # далее ваша бизнес-логика
    time.sleep(1)

# finish вызывать необязательно, но часто нужно,
# чтобы точно сказать, что обработка выполнена
bulk_log.finish()
```

Это выведет вам что-то вроде такого:

```
# 23:55:31:INFO: Начали цикл: Моя пачка {}
# 23:56:31:INFO: Моя пачка {'counter': 61, 'percentDone': 48, 'maxCount': 125}
# 23:57:31:INFO: Моя пачка {'counter': 121, 'percentDone': 96, 'maxCount': 125}
# 23:57:36:INFO: Закончили цикл: Моя пачка {}
```

## 2.3 Bulk Logger Thread Pool

Используется для логирования пачек из обрабатываемого списка. Делает запись в лог только если прошло определенное кол-во времени

```
# coding=utf-8
import time
from functools import partial
from multiprocessing.pool import ThreadPool

from metaappscriptsdk import MetaApp

META = MetaApp()

def my_thread_fn(bulk_log, job_item):
    bulk_log.try_log_part()
    # Бизнес логика
    # работа с job_item
    time.sleep(1)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
def my_main_fn():
    total = 125 * 2
    thread_cnt = 2

    bulk_log = META.bulk_log(u'Моя пачка', total, 1)
    bulk_log.try_log_part()

    all_data = range(1, total)
    pool = ThreadPool(thread_cnt)

    # Чтобы работать в многопоточном режиме с bulk_log вы
    # должны передать его как аргумент вызываемой функции таким образом
    func = partial(my_thread_fn, bulk_log)
    results = pool.map(func, all_data)

    bulk_log.finish()
    pool.close()
    pool.join()
    #print(results)

my_main_fn()
```

Это выведет вам что-то вроде такого:

```
# 16:25:08:INFO: Начали цикл: Моя пачка {}
# 16:26:08:INFO: Моя пачка {'counter': 122, 'percentDone': 48, 'maxCount': 250}
# 16:27:09:INFO: Моя пачка {'counter': 242, 'percentDone': 96, 'maxCount': 250}
# 16:27:17:INFO: Закончили цикл: Моя пачка {}
```



Это API внутренних функций Меты, все эти функции доступны вам через редактор в Web-интерфейсе

## 3.1 SettingsService

Рассчитан на чтение параметров из мета конфигурации Это удобно когда вы хотите хранить ссылки/токены для внешних апи, какие-то глобальные или частные настройки. При этом вы хотите дать некоторым пользователям возможность это редактировать через интерфейс

## 3.2 Примеры

Список примеров

## 3.3 MediaService

```
import os
from metaappscriptsdk import MetaApp

META = MetaApp()
log = META.log

os.chdir(os.path.dirname(os.path.abspath(__file__)))
__DIR__ = os.getcwd() + "/"

upload_file = open(__DIR__ + '../assets/load_data_sample.tsv', 'rb')

MediaService = META.MediaService
result = MediaService.upload(upload_file, {
```

(continues on next page)

(продолжение с предыдущей страницы)

```

    "isPrivate": True, # Файл будет доступен только для пользователя, работающего с али
    "ttlInSec": 9999, # Обязательно для временных файлов. Кол-во секунд через которые мета-
    ↪ автоматически удалит файл
    "entityId": 2770,
    "objectId": "114aecf5-04f1-44fa-8ad1-842b7f31a2df",
    "info": {"test": True} # Метаданные файла
})

print(u"result = %s" % str(result))
# result = {'id': 'ae2ef57a-c948-4ba4-8b68-6598352a2eb8', 'name': 'load_data_sample.tsv',
    ↪ 'extension': 'tsv', 'mime': 'text', 'url': None, 'creationTime': '2017-11-08T16:57:46Z', 'userId
    ↪ ': 4501, 'fileSize': 256, 'info': {'test': True}, 'private': True, 'downloadUrlPart': '/api/meta/
    ↪ v1/media/d/ae2ef57a-c948-4ba4-8b68-6598352a2eb8'}

# Скачать файл
result = MediaService.download('ae2ef57a-c948-4ba4-8b68-6598352a2eb8', as_stream=False)
print(u"result.content = %s" % str(result.content))

# Информация по файлу
resp = MediaService.info('5665d822-2edb-48b8-85a5-817043900a9a')
print(u"resp = %s" % str(resp))
# resp = {'id': '5665d822-2edb-48b8-85a5-817043900a9a', 'name': 'load_data_sample.tsv', 'extension
    ↪ ': 'tsv', 'mime': 'text', 'url': None, 'creationTime': '2017-11-08T16:45:00Z', 'userId': 4501,
    ↪ 'fileSize': 256, 'info': {'test': True}, 'private': True, 'downloadUrlPart': '/api/meta/v1/media/
    ↪ d/5665d822-2edb-48b8-85a5-817043900a9a'}

```

Это выведет вам что-то вроде такого:

```

# 16:48:19:INFO: Читаем настройки разработчика из локального файла {'path': '/Users/arturgspb/.
    ↪ rumeta/developer_settings.json'}
# 16:48:19:INFO: Инициализация службы {'debug': True}
# Empty stdin...
# result['rows'][0]['url'] = http://localhost:8080/media/d/c6509ac7-b410-4f77-8f0b-7c1dfd6a871b
# first = {'url': u'http://localhost:8080/media/d/c6509ac7-b410-4f77-8f0b-7c1dfd6a871b', u'id': u
    ↪ 'c6509ac7-b410-4f77-8f0b-7c1dfd6a871b', u'full_path': u'/mnt/static/public/74/reqtxt-2016-09-02_
    ↪ 16-48-19-(4501).txt'}
# result = {
#     "boxed": false,
#     "columns": [
#         {
#             "displayName": "Id",
#             "fullDisplayName": "Id",
#             "isPrimary": true,
#             "isStyled": false,
#             "name": "id",
#             "role": "dimension",
#             "type": "TEXT"
#         },
#         {
#             "displayName": "url",
#             "fullDisplayName": "url",
#             "isStyled": true,
#             "name": "url",
#             "role": "dimension",
#             "type": "TEXT"
#         }
#     ]
# }

```

(continues on next page)

(продолжение с предыдущей страницы)

```

#         "displayName": "downloadUrlPart",
#         "fullDisplayName": "downloadUrlPart",
#         "isStyled": true,
#         "name": "downloadUrlPart",
#         "role": "dimension",
#         "type": "TEXT"
#     },
#     {
#         "displayName": "fullPath",
#         "fullDisplayName": "fullPath",
#         "isStyled": true,
#         "name": "fullPath",
#         "role": "dimension",
#         "type": "TEXT"
#     }
# ],
# "containsLego": false,
# "empty": false,
# "exportable": true,
# "frame": false,
# "hasTemplate": false,
# "legoProperties": null,
# "metaData": {
#     "filtersAvailable": true,
#     "orderByAvailable": false,
#     "pagerAvailable": false,
#     "searchTextAvailable": false
# },
# "name": "",
# "pager": {
#     "limit": 20,
#     "maxPageLimit": 1000,
#     "offset": 0,
#     "total": null
# },
# "rows": [
#     {
#         "full_path": "/mnt/static/public/74/reqtxt-2016-09-02_16-48-19-(4501).txt",
#         "id": "c6509ac7-b410-4f77-8f0b-7c1dfd6a871b",
#         "url": "http://localhost:8080/media/d/c6509ac7-b410-4f77-8f0b-7c1dfd6a871b"
#     }
# ],
# "template": null
# }

```

## 3.4 DbQueryService

Делайте запросы к БД к вашим подключениям

```

db_adplatform = META.db("adplatform")
# Методы query, all, one ОБЯЗАТЕЛЬНО должны возвращать ResultSet (может быть и пустой)
# Т.е. нельзя делать UPDATE, INSET, DELETE, TRUNCATE, исключение - если в PostgreSQL вы делаете
↳ RETURNING

```

(continues on next page)

(продолжение с предыдущей страницы)

```

# Вернет стандартный метоевский data_result, где есть rows, columns, meta_data и np
data_result = db_adplatform.query("SELECT * FROM users LIMIT 10")

# Вернет rows из data result
users = db_adplatform.all("SELECT * FROM users LIMIT 10")

# Вернет первый элемент из rows или None, если нет первого элемента
users = db_adplatform.one("SELECT * FROM users WHERE id=4501 LIMIT 1")

# Метод update используется для запросов, которые НЕ ВОЗВРАЩАЮТ результат в виде ResultSet (в БД)
db_meta_samples = META.db("meta_samples")
dr = db_meta_samples.update("""
    UPDATE counters SET inc = inc + 1 WHERE name = :name
""", {"name": "md_source_update"})
print(u"dr = %s" % pretty_json(dr))

dr = db_meta_samples.batch_update("""
    INSERT INTO test_batch_update VALUES (:id, :mytime::timestamp)
    ON CONFLICT(id) DO UPDATE SET mod_time=NOW()
""", [
    {"id": "py_1", "mytime": "2014-01-01"},
    {"id": "py_2", "mytime": "2014-01-01"},
])
print(u"dr = %s" % pretty_json(dr))

```

Отдельно стоит упомянуть про LoadData Api Этот API позволяет как в BigQuery создавать таблицу у казанной БД и потоково загружать в нее данные из файла формата TSV Это позволяет ускорять вставку данных в таблицу от 2 до 4-5 раз.

**ВАЖНО!** Данные всегда добавляются в указанную таблицу и никакой очистки старых данных нет - вы должны почистить таблицу сами, если вам это нужно

```

import os
from metaappscriptsdk import MetaApp

META = MetaApp()

os.chdir(os.path.dirname(__file__))
__DIR__ = os.getcwd() + "/"

upload_file = open(__DIR__ + 'assets/load_data_sample.tsv', 'rb')

configuration = {
    "load": {
        "destinationTable": {
            "schema": "public",
            "table": "xxx_ya_stat"
        },
        "schema": {
            "fields": [
                {"name": "Date", "type": "DATE"},
                {"name": "Clicks", "type": "LONG"},
                {"name": "Cost", "type": "DECIMAL"},
                {"name": "AdNetworkType", "type": "TEXT"},
            ]
        }
    }
}

```

(continues on next page)



(продолжение с предыдущей страницы)

```

    ]
    }
}

db = META.db("meta_samples")
db.upload_data(upload_file, configuration)

```

## 3.5 SettingsService

Получайте настройки из стандартного источника настроек

```

settings = META.SettingsService

# Вернуть только данные
rwapp_conf = settings.data_get("rwapp")

# Полная информация о данных + данные
full_rwapp_conf = settings.data_get("rwapp", data_only=False)

onec_url = settings.config_param("rwapp", "app.onec.url")

```

## 3.6 IssueService

Управляйте тикетами через стандартные методы

```

from metaappscriptsdk import MetaApp

META = MetaApp()

IssueService = META.IssueService

test_issue_id = 12067
IssueService.add_issue_msg(test_issue_id, "robo test")
IssueService.done_issue(test_issue_id)

```

## 3.7 UserManagementService

Управляйте пользователями

```

from metaappscriptsdk import MetaApp

META = MetaApp()

UserManagementService = META.UserManagementService
resp = UserManagementService.send_recovery_notice("arturgspb", "meta")
print(u"resp = %s" % str(resp))
# resp = {'error': None, 'error_details': None, 'success_details': 'Вам отправлено уведомление о
↳ сбросе пароля на email art@realweb.ru. Следуйте инструкциям из письма.'}

```

(continues on next page)

(продолжение с предыдущей страницы)

```
resp = UserManagementService.send_recovery_notice("unknown_login_123123123", "meta")
print(u"resp = %s" % str(resp))
# resp = {'error': 'user_not_found', 'error_details': 'Пользователь с таким логином не найден',
↪ 'success_details': None}
```

## 3.8 StarterService

Для работы с апи запускатора

## 3.9 MailService

Для работы с почтовым клиентом

```
from metaappscriptsdk import MetaApp

META = MetaApp()

# Рекомендуется выдумывать unique_id для КАЖДОГО письма, чтобы избежать спама при ошибках или
↪ повторных запусках ваших скриптов
gen_id = "HJljkasdlkjasd"
META.MailService.submit_mail("meta@realweb.ru", "art@realweb.ru", "TTT", "ttt pong", unique_id="my_
↪ mail_category__" + gen_id)

# Без уникализации письма. Не рекомендуется, так как если ваш код будет багать и бесконечно
↪ добавлять письма - то, вы можете заспамить адресатов
META.MailService.submit_mail("meta@realweb.ru", "art@realweb.ru", "TTT", "ttt pong")
```

Основан на синтаксисе Oracle METAQL - Это защищенный SQL для запросов к данным меты.

## 4.1 Особенности

- Нельзя называть поля или алясы зарезарвированными именами: date
- Добавлен оператор ILIKE для регистронезависимного сравнения строк
- Пока нет преобразований типов
- Поддерживаются только SELECT запросы
- JOIN пока не поддерживаются

## 4.2 Функции

Работают функции агрегации:

- MIN, MAX, SUM, AGV
- COUNT(\*), COUNT(DISTINCT поле)

Функции преобразований:

- ROUND
- CONCAT
- NULLIF
- COALESCE

## 4.3 Примеры

Полный список metaql примеров

Список отчетов

```
import os
from metaappscriptsdk import MetaApp

META = MetaApp()
log = META.log

os.chdir(os.path.dirname(os.path.abspath(__file__)))
__DIR__ = os.getcwd() + "/"

q = """
SELECT
    engine as platform,
    campaign_remote_id,
    SUM(impressions) as impressions,
    SUM(clicks) as clicks,
    ROUND(SUM(cost), 3) as cost
FROM adplatform.campaign_stats_report
WHERE stat_date BETWEEN '2017-02-01' AND '2017-03-31'
AND engine = 'banner'
GROUP BY platform, campaign_remote_id
ORDER BY platform, campaign_remote_id
"""

configuration = {
    "download": {
        # "skipHeaders": True,
        "dbQuery": {
            "command": q,
        }
    }
}

metaql = META.MetaqlService
resp = metaql.download_data(configuration, output_file=__DIR__ + 'assets/stat.tsv')
log.info("end")
```