RFID_module Documentation

Выпуск

Catman

июн. 09, 2017

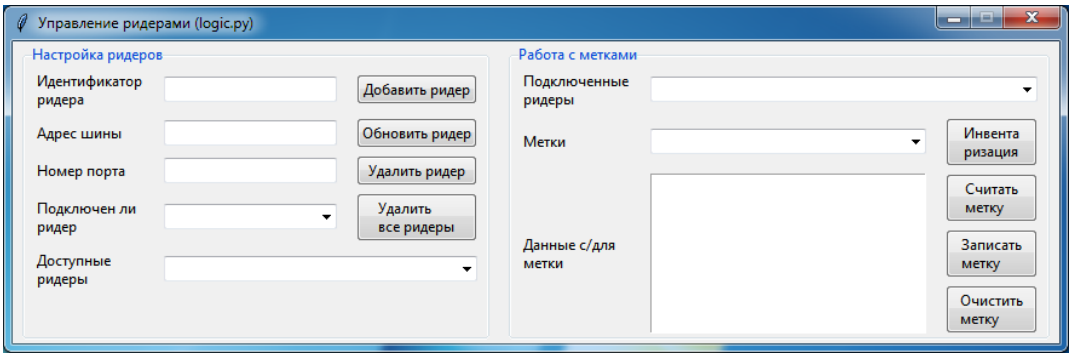
Оглавление

Модуль состоит из двух частей, написанных на языках C++ и Python. Модуль предоставляет возможность работы с RFID-оборудованием (ридером) через код, написанный на языке Python.

Доступ к функциям модуля осуществляется посредством HTTP-запросов по определённому Web API.

Особенности модуля и предоставляемый им функционал:

- управление несколькими ридерами
- хранение настроек ридеров
- соединение с ридером
- инвентаризация (получения серийных номеров меток)
- чтение информации с меток
- запись информации в метки
- работа с ридерами через HTTP-запросы
- Web API, созданный в соответствии с архитектурным стилем RESTful API
- работа с ридерами через Python-модуль (`logic.py`) с использованием тех же принципов, что с Web API
- работа с ридерами через утилиту с графическим интерфейсом (`gui.py`)



- понятный и протестированный код (`unittests`)
- непонятный код в `logic.py`

Глава 1

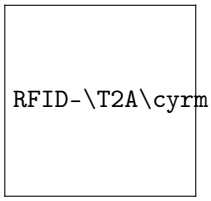
Устройство проекта

Предупреждение: Файлы из FEIG SDK не добавлены в репозиторий, их нужно будет самостоятельно скопировать в нужные директории

- `/lib` — FEIG SDK (заголовочные файлы)
- `/RFID` — папка проекта Visual Studio
  - `dllmain.cpp` — главный файл C++-модуля
  - `RFID.vcxproj` — файл проекта Visual Studio
  - `RFID.sln` — файл проекта Visual Studio
- `/module` — основная папка модуля
  - `RFID.dll` — динамическая библиотека для работы с RFID-оборудованием
  - `reader.py` — обёртка над C++-модулем, работа с ридером
  - `logic.py` — скрипт, содержащий принципы работы с ридерами (доступ осуществляется через объект `Readers`)
  - `gui.py` — графическая утилита для работы с ридерами через `logic.py`
  - `test_Readers.py` — файл с тестами для класса `Readers` из `logic.py`
  - `test_server.py` — файл с тестами для веб-сервера (`server.py`)
  - `FedmIscCoreVC110.dll`, `feisc.dll`, `fefu.dll`, `fecom.dll`, `fetcl.dll` — файлы из FEIG SDK, необходимые для работы `RFID.dll`

Глава 2

Устройство модуля

модуль.png 

Зависимости, сборка проекта

Зависимости

Оборудование, для которого создавалось ПО:

- антенна ID ISC.ANT300/300-A
- ридер ID ISC.MR200 A
- метки по 224 байта

Требования для RFID-оборудования:

- Windows 7 и выше
- наличие COM-порта для подключения ридера

Для сборки C++-модуля:

- Microsoft Visual Studio 2012
- FEIG SDK (ID ISC.SDK.Win V4.7.0)

Для Python-модуля:

- Python 3.6.0 32 bit
- flask == 0.12 (для веб-сервера)

Для сборки документации:

- Python 3.6.0 32 bit
- Sphinx == 1.5.3
- sphinxcontrib-httpdomain == 1.5.0
- sphinx_rtd_theme == 0.2.2

Примечание: Зависимости для Python-модуля и сборки документации могут быть установлены с помощью pip следующим образом:

```
pip install -r requirements.txt
```

Примечание: Зависимости в Windows 7 и Visual Studio 2012 исходят из того, что FEIG SDK (ID ISC.SDK.Win V4.7.0) была собрана с инструментами сборки версии VC110 (Visual Studio 2012)

Подготовка рабочего окружения, сборка проекта

1. Клонировать проект с GitHub'а
2. Скопировать в папку `/lib` заголовочные файлы FEIG SDK с сохранением первоначальной иерархии папок и файлов
3. Скопировать в папку `/module` библиотеки `FedmIscCoreVC110.dll`, `feisc.dll`, `fefu.dll`, `fecom.dll`, `fetcl.dll` из FEIG SDK
4. Установить зависимости для Python-модуля
5. Собрать C++-модуль
   (a) В Visual Studio открыть проект, выбрав файл `/RFID/RFID.vcxproj`
   (b) Собрать **32-битную Release-версию**
   (c) Итоговый файл `RFID.dll` окажется в папке `/module`
6. Теперь доступны два выбора для работы с ридерами:
   - Работать через `logic.py`
   - Запустить сервер (`server.py`) и работать через WebAPI

Примечание: Важно, чтобы разрядность, под которую собран RFID.dll, и разрядность интерпретатора Python были одинаковыми — **x32**. Однако, возможно, всё будет работать, если и то, и другое будет 64-разрядными.

Сборка документации

1. Перейти в папку `/docs`
2. В консоли ввести следующую команду:

```
make html
```

3. Собранная документация окажется в папке `/docs/_build/html`

Web API - server.py

Примечание: Работа с ридерами через logic.py

Есть возможность работать с ридерами через RFID-модуль не только с помощью Web API, но и через скрипт `logic.py`, импортировав его как простой Python-модуль. При этом вся работа осуществляется через объект `Readers`. Сохраняются все те же принципы работы, что и через WebAPI: та же структура запросов и ответов. **При вызове методов обязательно нужно указывать названия параметров.**

Принципы работа с API

API создавалось согласно архитектурному стилю RESTful API.

Запрос	GET	POST	PUT	DELETE
`/`	Страница с описанием API	—	—	—
`/readers/`	Возвращает список настроек для ридеров	Добавляет настройки для ридера	Заменяет все настройки ридеров переданными	Удаляет все настройки ридеров
`/readers/<reader_id>/`	Возвращает настройки ридера	—	Обновляет настройки ридера	Удаляет ридер
`/readers/<reader_id>/tags/inventory/`	Возвращает идентификаторы меток	—	—	—
`/readers/<reader_id>/tags/`	Возвращает информацию с меток	—	Записывает информацию в метки	Очищает информацию с меток

Формат запроса

Запросы, предназначенные для определённых ридеров, должны иметь в себе идентификатор ридера.

При выполнении запроса, требующего некоторые данные, эти данные нужно отправлять в формате JSON.

Формат ответа

В качестве ответа на запросы выступают данные в формате JSON.

Общий формат ответа:

```
{
    "response": "какие-то данные"
}
```

В качестве значения по ключу `"response"` может выступать любые типы данных JSON.

При наличии ошибки (в общем случае):

```
{
    "error": {
        "error_code": 0,
        "error_msg": "Описание ошибки"
    }
}
```

Примечание: Может быть и такое, что в ответе будут присутствовать оба ключа.

В зависимости от вызываемого метода формат ответа может отличаться от представленных выше. Поэтому также необходимо изучить информацию по каждому из используемых запросов.

Начало работы с API

1. Перед началом работы с ридерами нужно получить данные о уже существующих ридерах: так как настройки сохраняются после каждой манипуляции с ними, то может оказаться так, что RFID-модуль будет стартовать с полученными ранее данными.

   GET /readers/ — получение данных о ридерах

2. Подготовить ридеры к работе

   - Добавить новый ридер: *POST /readers/*
   - Обновить настройки ридера: *PUT /readers/<reader_id>/*
   - Если необходимо заменить все настройками ридеров своими: *PUT /readers/*

3. Изменение состояния ридера

   Подключение/отключение ридера может производиться с помощью методов *POST /readers/*, *PUT /readers/<reader_id>/*, *PUT /readers/* при обязательном наличии ключа `"state": true` в теле запроса

   Примечание: После завершения работы с ридером, его необходимо отключить (но это не точно)

   Предупреждение: Важное замечание: невозможно изменить настройки ридера или удалить его, пока он подключен

4. Работа с метками осуществляется с помощью методов:

   - *GET /readers/<reader_id>/tags/inventory/*
   - *GET /readers/<reader_id>/tags/*
   - *PUT /readers/<reader_id>/tags/*
   - *DELETE /readers/<reader_id>/tags/*

Предупреждение: При вызове методов не забывайте указывать слэш (“/”) в конце запроса

Система ошибок

Ниже представлен список ошибок, которые могут возникнуть в ходе работы программы:

Номер	Описание
<0	[Ошибки на уровне ридера (FEIG SDK)]
0	Некорректный запрос
1	Некорректный набор параметров
2	Некорректное значение идентификатора ридера
3	Некорректное значение параметра шины адреса
4	Некорректное значение параметра номера порта
5	Некорректное значение параметра состояния ридера
6	Некорректное значение идентификатора метки
7	Некорректное значение данных для записи в метку
8	Ридер с данным идентификатором существует
9	Ридера с данным идентификатором не существует
10	Операция не может быть совершена, так как ридер подключён
11	Операция не может быть совершена, так как ридер отключён
12	Операция не может совершена, так как один или больше ридеров подключены
13	Список ридеров уже пуст
100	Вызван метод без указания названий параметров *(внутренняя ошибка)*

Формат данных на ячейке

Для обмена данными меток между клиентом и RFID-модулем используется формат строки. То есть для записи в определённую метку информации необходимо отправить строку, при считывании данных с метки также вернётся строка. На формат строки накладывается два ограничения:

- длина строки не должна превышать максимальную ёмкость метки, иначе данные будут записаны не полностью
- в передаваемой строке не должно быть символов не из кодировки CP866.

Данные в строке могут быть представлены JSON-форматом. Пример подготовки данных на языке Python:

```
import json
# исходные данные
data = {
    'pallete_id': 123,
    'tag_location': 4,
    'detail_type': 5,
    'state': 3,
    'params': {
        'diameter': 1.323,
        'thickness': 1.1,
        'radius': 0.5,
        'tolerance': 0.002
    }
}
# второй вариант представления данных - более экономный
data = [123, 4, 5, 3, [1.323, 1.1, 0.5, 0.002]]
```

```
# представление данных в формате json
# параметр separators используется для удаления незначащих пробелов, что поможет сэкономить место
json_data = json.dumps(data, separators=(',', ':'))

TAG_SIZE = 224  # объём метки
# проверим, сможем ли записать данные на метку
if len(json_data == TAG_SIZE):
    # формирование запроса на запись данных в метку, совершение запроса
    pass

# пример разбора данных с метки, представленных в JSON-формате
obtained_data = json.loads(obtained_data)
```

Разное

`GET /docs`
: Документация на RFID-модуль. Если документация не собрана, то будет выведо соответствующее сообщение

Управление ридерами

`GET /readers/`
: Возвращает список настроек и состояний ридеров

Пример запроса:

```
GET /readers/ HTTP/1.1
```

Примеры ответа:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "response": {
        "2": {
            "bus_addr": 1,
            "port_number": 1,
            "state": false
        },
        "3": {
            "bus_addr": 50,
            "port_number": 1,
            "state": true
        }
    }
}
```

Если нет ни одного ридера:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
```

```
    "response": {}
}
```

Status Codes

- 200 OK – нет ошибок

`POST /readers/`

Добавляет настройки для ридера. При наличии в data необязательного ключа state==True произойдёт подключениие ридера

Пример запроса:

```
POST /readers/ HTTP/1.1
Content-Type: application/json

{
    "reader_id": "1",
    "bus_addr": 1,
    "port_number": 1
}
```

Примеры ответа:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "response": 0
}
```

Если отправлено параметров меньше, чем необходимо, или названия параметров некорректны:

```
HTTP/1.0 400 BAD REQUEST
Content-Type: application/json

{
  "error": {
    "error_code": 1,
    "error_msg": "Некорректный набор параметров"
  }
}
```

```
HTTP/1.0 400 BAD REQUEST
Content-Type: application/json

{
    "error": {
        "error_code": 8,
        "error_msg": "Ридер с данным идентификатором существует"
    }
}
```

Status Codes

- 200 OK – нет ошибок
- 400 Bad Request – ошибка в запросе, ошибки в названиях полей, передан не json

Возможные ошибки 1, 2, 3, 4, 8

`PUT /readers/`

Заменяет все настройки ридеров переданными

Пример запроса:

```
POST /readers/ HTTP/1.1
Content-Type: application/json

{
    "1": {
        "bus_addr": 1,
        "port_number": 1
    },
    "42": {
        "bus_addr": 255,
        "port_number": 0,
        "state": true
}
```

Примеры ответа:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "response": {
        "1": 0
    },
    "error": {
        "42": {
            "error_code": 10,
            "error_msg": "Операция не может быть совершена, так как ридер подключён"
        }
    }
}
```

Status Codes

- 200 OK – настройки удалены
- 400 Bad Request – ошибка в запросе, ошибки в названиях полей, передан не json

Возможные ошибки <0, 2, 3, 4, 5, 8, 9, 10

`DELETE /readers/`

Удаляет все настройки ридеров

Пример запроса:

```
DELETE /readers/ HTTP/1.1
```

Пример ответа:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
```

```
    "response": 0
}
```

Status Codes

- 200 OK – настройки удалены

Возможные ошибки 12, 13

Управление отдельным ридером

`GET /readers/<reader_id>/`

Возвращает настройки и состояние ридера

Пример запроса:

```
GET /readers/1/ HTTP/1.1
```

Пример ответа:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "response": {
        "1": {
            "bus_addr": 255,
            "port_number": 1,
            "state": false
        }
    }
}
```

Status Codes

- 200 OK – возвращены настройки
- 404 Not Found – ридер не найден

Возможные ошибки 9

`PUT /readers/<reader_id>/`

Обновляет настройки и состояние ридера

Примеры запроса:

Изменение параметров ридера:

```
PUT /readers/1/ HTTP/1.1
Content-Type: application/json

{
    "bus_addr": 255,
    "port_number": 1,
}
```

Изменение идентификатора ридера:

```
PUT /readers/1/ HTTP/1.1
Content-Type: application/json

{
    "reader_id": "cat"
}
```

Подключение ридера:

```
PUT /readers/meow/ HTTP/1.1
Content-Type: application/json

{
    "state": true
}
```

Пример ответа:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "response": 0
}
```

Status Codes

- 200 OK – настройки обновлены
- 400 Bad Request – ошибка в запросе, ошибки в названиях полей, передан не json
- 404 Not Found – ридер не найден

Возможные ошибки <0, 2, 3, 4, 5, 8, 9, 10

`DELETE /readers/<reader_id>/`

Удаляет ридер

Пример запроса:

```
DELETE /readers/1/ HTTP/1.1
```

Пример ответа:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "response": 0
}
```

Status Codes

- 200 OK – настройки удалены
- 404 Not Found – ридер не найден

Возможные ошибки 9, 10

Работа с метками

`GET /readers/<reader_id>/tags/inventory/`
Возвращает идентификаторы меток

Пример запроса:

```
GET /readers/1/tags/inventory/ HTTP/1.1
```

Пример ответа:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "response": ["meow", "woof"]
}
```

Status Codes

- 200 OK – идентификаторы возвращены
- 404 Not Found – ридер не найден

Возможные ошибки <0, 9, 11

`GET /readers/<reader_id>/tags/`
Возвращает информацию с меток

Пример запроса:

```
GET /readers/1/tags/ HTTP/1.1
Content-Type: application/json

[
    "meow",
    "woof"
]
```

Если переданный массив будет пустым, произойдёт считывание с меток, находящихся в зоне действия антенны:

```
GET /readers/1/tags/ HTTP/1.1
```

Пример ответа:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "response": {
        "meow": "Vasya",
        "woof": "Kys-kys-kys"
    }
}
```

Status Codes

- 200 OK – информация возвращена

- 400 Bad Request – ошибка в запросе, передан не json
- 404 Not Found – ридер не найден

Возможные ошибки <0, 9, 11

`PUT /readers/<reader_id>/tags/`

Записывает информацию в метки

Пример запроса:

```
PUT /readers/1/tags/ HTTP/1.1
Content-Type: application/json

{
    "meow": "Vasya",
    "woof": "Kys-kys-kys"
}
```

Пример ответа:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "response": 0
}
```

Status Codes

- 200 OK – информация записана
- 400 Bad Request – ошибка в запросе, ошибки в названиях полей, передан не json
- 404 Not Found – ридер не найден

Возможные ошибки <0, 9, 11

`DELETE /readers/<reader_id>/tags/`

Очищает информацию с меток

Пример запроса:

```
DELETE /readers/1/tags/ HTTP/1.1
Content-Type: application/json

[
    "meow",
    "woof"
]
```

Пример ответа:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "response": 0
}
```

Status Codes

- 200 OK – информация удалена
- 400 Bad Request – ошибка в запросе, ошибки в названиях полей, передан не json
- 404 Not Found – ридер не найден

Возможные ошибки <0, 9, 11

HTTP Routing Table

/docs

/readers