**Работаем с каркасами Micromine с помощью Python**

*Предупреждение! Все форматы хранения данных в Micromine могут быть изменены в любой момент по усмотрению разработчиков компании без каких-либо предупреждений для пользователей.*

**Структура файлов TRIDB**

Сначала поговорим о том, что из себя представляют файлы TRIDB (в Micromine они называются «типами каркасов») и о том, какую структуру они имеют.

Файл TRIDB, по сути, является реляционной базой данных формата [SQLite](https://ru.wikipedia.org/wiki/SQLite), в которой содержится информация о геометрии каркасов (триангуляция), их атрибутах и метаданных. Для чтения и редактирования таких файлов удобно использовать Python и модуль sqlite3, который входит в стандартную библиотеку языка.

База данных TRIDB состоит из десяти таблиц:

* DatabaseInformation;
* DisplayProperties (добавлена в MM2018);
* GeneralInformation;
* Geometry;
* PointAttributeDefinition;
* PointAttributeValue;
* TriangleAttributeDefinition;
* TriangleAttributeValue;
* UserAttributeDefinition;
* UserAttributeValue.

Для пользователей интерес представляют следующие пять таблиц: GeneralInformation, DisplayProperties, Geometry, UserAttributeDefinition и UserAttributeValue.

Таблица GeneralInformation содержит информацию о стандартных атрибутах каждого каркаса (Имя/Name, Заголовок/Title, Код/Code, Цвет/Colour). Кроме этого, здесь же хранится информация об Объеме/Volume, Площади поверхности/SurfaceArea и статусе Проверки/Validated. Значение поля «Проверка» представляет собой целое число, в котором закодирован результат функции Micromine *Проверка каркаса* или *Проверка ЦМП*. Например, числу 2751 соответствует результат проверки «Открыто, верно», 2827 – «Закрыто, верно», 3599 – «ЦМП, верна».

Помимо описанных атрибутов, в таблице GeneralInformation хранятся метаданные:

1) информация о создании каркаса. Дата и время/CreateDateTime в формате [Unix](https://ru.wikipedia.org/wiki/Unix-%D0%B2%D1%80%D0%B5%D0%BC%D1%8F), имя автора/CreateBy и компании/CreateCompany;

2) информация об изменении каркаса. Дата и время/EditDateTime в формате [Unix](https://ru.wikipedia.org/wiki/Unix-%D0%B2%D1%80%D0%B5%D0%BC%D1%8F), имя автора/EditBy и компании/EditCompany;

3) границы каркаса. Минимальные и максимальные координаты (XMinimum, XMaximum, YMinimum, YMaximum, ZMinimum, ZMaximum);

4) примечания/MetaNotes и точность/Accuracy.

Таблица DisplayProperties содержит информацию об используемых штриховках срезов и силуэтов, SliceHatch и SilhouetteHatch соответственно.

В таблице Geometry в бинарном формате ([BLOB](https://ru.wikipedia.org/wiki/BLOB)) хранится информация о геометрии каркаса (треугольники и точки). Она не может быть считана напрямую с помощью Python. Однако, если все-таки необходимо обработать геометрию каркаса в скрипте, можно сначала сделать экспорт каркаса в файлы данных Micromine, воспользовавшись инструментом Micromine *Файл – Экспорт – Каркасы*, а затем считать эти файлы с помощью модуля MMpy. Такой случай будет подробно разобран ниже в Примере 6.

В таблице UserAttributeDefinition перечислены имена пользовательских атрибутов/Name. А в таблице UserAttributeValue атрибуты и их значения/ValueText связаны с каркасами с помощью ключей: TriangulationID (ключи каркасов) и AttributeID (ключи пользовательских атрибутов).

**Примеры скриптов**

Рассмотрим несколько примеров скриптов. Как уже было указано выше, TRIDB файлы – это базы данных SQLite, поэтому работать с ними мы будем с помощью стандартного для Python модуля sqlite3 и SQL-запросов. Наиболее распространёнными задачами являются чтение, редактирование, добавление и удаление информации о каркасах. Скрипт для решения любой из указанных задач будет начинаться одинаково (Пример 1). В комментариях к коду подробно описан процесс работы.

*Пример 1. Начало работы с файлом TRIDB*

1 *# Импорт модуля sqlite3 для работы с файлами TRIDB*

2 **import** sqlite3

3

4 *# Путь к файлу TRIDB*

5 tridb\_path = **r'D:\type\_example\_1.TRIDB'**

6

7 *# Создание подключения к указанному TRIDB файлу*

8 connection = sqlite3.connect(tridb\_path)

9 *# Создание курсора*

10 cursor = connection.cursor()

Для лучшего понимания, что именно происходит в строках 8 и 10 в Примере 1, где создаются подключение и курсор, можно прибегнуть к весьма упрощенной, но наглядной аналогии.

Файл базы данных можно представить как Excel-документ, которой состоит из нескольких листов. Информация на разных листах может быть связана между собой. Для того, чтобы начать работать с Excel-документом, его нужно сначала открыть. Создание подключения (строка 8) аналогично открытию документа. В строке 10 происходит создание курсора. Курсор базы данных в определенном смысле аналогичен курсору мышки. С помощью курсора мышки вы можете выделять ячейки вместе с хранящейся в них информацией, а также копировать, удалять и редактировать данные. То же самое вы можете делать с помощью курсора базы данных.

В Примере 2 показано, как можно получить информацию о каркасах, имеющих объем больше 400 м3, из файла «type\_example\_2.TRIDB». В этом же примере показано, как получить суммарный объем найденных каркасов с помощью функции SUM().

*Пример 2. Чтение информации из файла TRIDB – получение информации о каркасах, имеющих определенный объем*

1 **import** sqlite3

2

3 tridb\_path = **r'D:\type\_example\_2.TRIDB'**

4 connection = sqlite3.connect(tridb\_path)

5 cursor = connection.cursor()

6

7 *# SQL-запрос для получения имен каркасов с объемом более 400 м3*

8 sql\_request = **"SELECT Name, Volume, CreateBy FROM GeneralInformation WHERE Volume > 400"**

9 *# Выполнение SQL-запроса*

10 cursor.execute(sql\_request)

11 *# Вывод результатов на экран*

12 print(**"Имя каркаса, объем и автор: "**, cursor.fetchall())

13

14 *# SQL-запрос для получения суммарного объема*

15 sql\_request = **"SELECT SUM(Volume) FROM GeneralInformation WHERE Volume > 400"**

16 *# Выполнение SQL-запроса*

17 cursor.execute(sql\_request)

18 *# Вывод результата на экран*

19 print(**"Суммарный объем найденных каркасов: "**, cursor.fetchone()[0])

20

21 *# Закрыть соединение*

22 connection.close()

*Результат работы скрипта:*

Имя каркаса, объем и автор: [('каркас\_2', 759.182664791443, 'Ivan'), ('каркас\_3', 566.784729452728, 'Max')]

Суммарный объем найденных каркасов: 1325.967394244171

Метод cursor.fetchall()в строке 12 выдает все записи, которые были получены в результате выполнения SQL-запроса из строки 10. Метод cursor.fetchone()в строке 19 выдает только первую полученную запись для SQL-запроса из строки 15. Поскольку суммарный объем представляет собой одно число (а значит, и одну запись), то используем cursor.fetchone() (хотя ничто не мешает использовать cursor.fetchall(), поэкспериментируйте!). В результате на экран будет выведен список, состоящий из двух найденных записей, и суммарный объем. В конце работы скрипта необходимо закрывать соединение с базой данных с помощью метода close(), чтобы избежать возникновения конфликтов при дальнейшей работе с этой базой данных.

Теперь рассмотрим Пример 3, в котором происходит изменение информации, хранящейся в файле TRIDB: для каркасов, у которых автором числится «Max», изменим имя автора на «Andrey». Заметим, что информация об авторе хранится в метаданных каркаса. Эта информация не может быть изменена стандартными средствами Micromine, но может быть легко отредактирована с помощью скрипта.

*Пример 3. Редактирование информации из файла TRIDB – изменение метаданных каркасов*

1 **import** sqlite3

2

3 tridb\_path = **r'D:\type\_example\_3.TRIDB'**

4 connection = sqlite3.connect(tridb\_path)

5 cursor = connection.cursor()

6

7 *# SQL-запрос для получения имен каркасов и их авторов*

8 sql\_request = **"SELECT Name, CreateBy FROM GeneralInformation"**

9 *# Выполнение SQL-запроса*

10 cursor.execute(sql\_request)

11 *# Вывод результатов на экран*

12 print(**"Имена каркасов и их авторы: "**, cursor.fetchall())

13

14 *# SQL-запрос для изменения имени автора*

15 sql\_request = **"UPDATE GeneralInformation set CreateBy = 'Andrey' where CreateBy = 'Max'"**

16 *# Выполнение SQL-запроса*

17 cursor.execute(sql\_request)

18 *# Подтверждение внесенных изменений*

19 connection.commit()

20

21 *# Повторение первого SQL-запроса для того, чтобы увидеть внесенные изменения*

22 sql\_request = **"SELECT Name, CreateBy FROM GeneralInformation"**

23 cursor.execute(sql\_request)

24 print(**"Имена каркасов и их авторы: "**, cursor.fetchall())

25

26 connection.close()

*Результат работы скрипта:*

Имена каркасов и их авторы: [('каркас\_1', 'Max'), ('каркас\_2', 'Ivan'), ('каркас\_3', 'Max'), ('каркас\_4', 'Max')]

Имена каркасов и их авторы: [('каркас\_1', 'Andrey'), ('каркас\_2', 'Ivan'), ('каркас\_3', 'Andrey'), ('каркас\_4', 'Andrey')]

В Примере 3 интерес представляют строки 15 и 19. В строке 15 указан новый, по сравнению с предыдущим примером, SQL-запрос, в котором используется функция UPDATE для изменения данных. Метод connection.commit() в строке 19 необходим после того, как мы вносим любые изменения в базу данных (редактируем или удаляем существующие данные, добавляем новые). По сути, этот метод аналогичен нажатию на кнопку «Cохранить» в Excel-документе.

В качестве примера добавления информации в TRIDB файл, рассмотрим задачу создания нового пользовательского атрибута «Тоннаж» и присваивание ему значения для каждого каркаса. Скрипт для решения данной проблемы представлен в Примере 4.

*Пример 4. Добавление информации в файл TRIDB – добавление пользовательского атрибута и задание значений для каждого каркаса*

1 **import** sqlite3

2

3 tridb\_path = **r'D:\type\_example\_4.TRIDB'**

4 connection = sqlite3.connect(tridb\_path)

5 cursor = connection.cursor()

6

7 *# Получение ID и объемов каркасов*

8 cursor.execute(**'SELECT ID, Volume FROM GeneralInformation'**)

9 ids\_volumes = cursor.fetchall()

10 print(**"ID и объем каркасов: "**, ids\_volumes)

11

12 *# Добавление нового пользовательского атрибута "Тоннаж" с ID = 1*

13 cursor.execute(**"INSERT INTO UserAttributeDefinition (ID, Name, Type, Precision) VALUES (1, 'Тоннаж', 0, 0);"**)

14

15 *# Для каждого каркаса рассчитывается и добавляется значение нового атрибута*

16 **for** id, volume **in** ids\_volumes:

17 fact\_volume = volume \* 2.7

18 cursor.execute(**'INSERT INTO UserAttributeValue (TriangulationID, AttributeID, ValueText) VALUES (?, ?, ?)'**, (id, 1, fact\_volume))

19

20 *# Получение ID и тоннажа каркасов*

21 cursor.execute(**'SELECT TriangulationID, ValueText FROM UserAttributeValue'**)

22 print(**"ID и тоннаж каркасов: "**, cursor.fetchall())

23

24 connection.commit()

25 connection.close()

*Результат работы скрипта:*

Имена ID и объем каркасов: [(2, 759.182664791443), (5, 566.784729452728), (6, 374.3882343280012)]

ID и тоннаж каркасов: [(2, '2049.7931949369'), (5, '1530.31876952237'), (6, '1010.8482326856')]

Для добавления записей в таблицы TRIDB используется функция INSERT. Сначала добавляется сам атрибут в строке 13, а затем каждому каркасу присваивается значение этого атрибута в строке 18. Кроме этого, строка 18 интересна еще с точки зрения того, как именно добавлять в SQL-запрос значения, хранящиеся в переменных. Значение атрибута рассчитывается как объем, взятый из метаданных и умноженный на плотность, 2,7 т/м3 (строка 17).

В Примере 5 показан скрипт, с помощью которого из файла TRIDB удаляются все каркасы, которые были созданы пользователем по имени “Andrey”.

*Пример 5. Удаление информации из файла TRIDB – удаление каркасов, созданных определенным автором*

1 **import** sqlite3

2

3 tridb\_path = **r'D:\type\_example\_5.TRIDB'**

4 connection = sqlite3.connect(tridb\_path)

5 cursor = connection.cursor()

6

7 *# Получение имен и ID всех каркасов*

8 cursor.execute(**"SELECT Name, ID FROM GeneralInformation"**)

9 print(**"Имена и ID всех каркасов: "**, cursor.fetchall())

10

11 *# Получение имен и ID каркасов, которые создал пользователь Andrey*

12 cursor.execute(**"SELECT Name, ID FROM GeneralInformation WHERE CreateBy = 'Andrey'"**)

13 names\_and\_ids = cursor.fetchall()

14 print(**"Каркасы, которые создал Андрей: "**, names\_and\_ids)

15 *# Используя полученные ID каркасов, удаляем соответствующие записи в базе данных*

16 **for** name, id **in** names\_and\_ids:

17 cmd =**"""**

18  **DELETE FROM UserAttributeValue WHERE TriangulationId = {0};**

19  **DELETE FROM TriangleAttributeValue WHERE TriangulationId = {0};**

20  **DELETE FROM PointAttributeValue WHERE TriangulationId = {0};**

21  **DELETE FROM Geometry WHERE ID = {0};**

22  **DELETE FROM GeneralInformation WHERE ID = {0};**

23  **"""**.format(id)

24 cursor.executescript(cmd)

25 connection.commit()

26

27 *# Повторение первого SQL-запроса для того, чтобы увидеть изменения*

28 cursor.execute(**"SELECT Name, ID FROM GeneralInformation"**)

29 print(**"Имена и ID всех оставшихся каркасов: "**, cursor.fetchall())

30

31 connection.close()

*Результат работы скрипта:*

Имена и ID всех каркасов: [('каркас\_1', 1), ('каркас\_2', 2), ('каркас\_3', 5), ('каркас\_4', 6)]

Каркасы, которые создал Андрей: [('каркас\_1', 1), ('каркас\_3', 5), ('каркас\_4', 6)]

Имена и ID всех оставшихся каркасов: [('каркас\_2', 2)]

Для того чтобы удалить какие-либо каркасы из файла TRIDB, необходимо знать их идентификационные номера (ID). В данном примере нужно узнать ID каркасов, которые были созданы определенным автором. Эту информацию можно получить из таблицы GeneralInformation (строки 12 и 13). Затем с помощью цикла происходит удаление информации по выбранному каркасу из всех таблиц файла TRIDB (строки 16-25). Заметим, что был использован метод executescript(), который позволяет выполнить несколько SQL-запросов за один раз. Кроме этого, в данном примере применяется другой способ использования переменных в SQL-запросе – форматирование строки с помощью метода format()[[1]](#footnote-1).

В результате работы скрипта были удалены три каркаса, созданные пользователем по имени Andrey. Справедливости ради нужно отметить, что успешно выполненная проверка по одной таблице (строки 28 и 29) не гарантирует того, что информация по каркасам была удалена из каждой таблицы базы данных. В качестве самостоятельной работы выведите информацию о имеющихся каркасах в других таблицах. С помощью Micromine (функция «Управление каркасами») вы также можете убедиться, что нужные каркасы были удалены.

В качестве заключительного примера рассмотрим, как можно выгрузить геометрию каркаса с помощью скрипта и затем визуализировать его с помощью библиотеки matplotlib.

*Пример 6. Экспорт геометрии каркаса и его визуализация c помощью matplotlib*

1 *# Импорт библиотек для визуализации экспортированного каркаса*

2 **import** matplotlib.pyplot **as** plt

3 **from** mpl\_toolkits.mplot3d **import** axes3d, Axes3D

4

5 *# Функция для вызова диалогового окна Файл - Экспорт - Каркасы*

6 *# Параметры функции: тип и имя каркаса, пути к новым файлам с точками и треугольниками*

7 **def** wf\_geo\_to\_mm\_file(wf\_type, wf\_name, points\_path, triangles\_path):

8 WireframesExport\_FormSet1 = MMpy.FormSet(**"WIREFRAMES\_EXPORT"**, **"16.1.1251.2"**)

9 WireframesExport\_FormSet1.set\_field(**"POINTSFILE\_TYPE"**, **"2"**)

10 WireframesExport\_FormSet1.set\_field(**"POINTSFILE"**, points\_path)

11 WireframesExport\_FormSet1.set\_field(**"TRIANGLESFILE\_TYPE"**, **"2"**)

12 WireframesExport\_FormSet1.set\_field(**"TRIANGLESFILE"**, triangles\_path)

13 WireframesExport\_FormSet1.set\_field(**"EXPORT\_TYPE"**, **"5"**)

14 WireframesExport\_FormSet1.set\_field(**"FILENAME"**, **""**)

15 WireframesExport\_FormSet1.set\_field(**"LAYER\_BOOL"**, **"0"**)

16 WireframesExport\_FormSet1.set\_field(**"WF\_NAME"**, wf\_name)

17 WireframesExport\_FormSet1.set\_field(**"WFTYPE"**, wf\_type)

18 WireframesExport\_FormSet1.set\_field(**"SET\_BOOL"**, **"0"**)

19 WireframesExport\_FormSet1.set\_field(**"SINGLE\_BOOL"**, **"1"**)

20 WireframesExport\_FormSet1.run()

21

22 *# Функция для чтения экспортированных файлов STR*

23 *# Данные записываются в список data*

24 **def** mm\_read(path):

25 mm\_file = MMpy.File()

26 mm\_file.open(path)

27 data = []

28 **for** i\_record **in** range(mm\_file.records\_count):

29 data.append([])

30 **for** i\_column **in** range(1, 4):

31 data[i\_record].append(mm\_file.get\_num\_field\_value(i\_column, i\_record + 1))

32 mm\_file.close()

33 **return** data

34

35 *# Путь к файлу TRIDB и имя каркаса*

36 wf\_type = **r'D:\type\_example\_6.tridb'**

37 wf\_name = **'каркас\_2'**

38 *# Пути для сохранения файлов точек и треугольников*

39 points\_path = **r'D:\points.STR'**

40 triangles\_path = **r'D:\tris.STR'**

41

42 *# Вызываем функцию для получения файлов с геометрией*

43 wf\_geo\_to\_mm\_file(wf\_type, wf\_name, points\_path, triangles\_path)

44

45 *# Записываем информацию из файлов точек и треугольников в списки points и tri соответственно*

46 points = mm\_read(points\_path)

47 tri = mm\_read(triangles\_path)

48

49 *# Списки координат*

50 x, y, z = list(zip(\*points))

51 *# Создание визуализации*

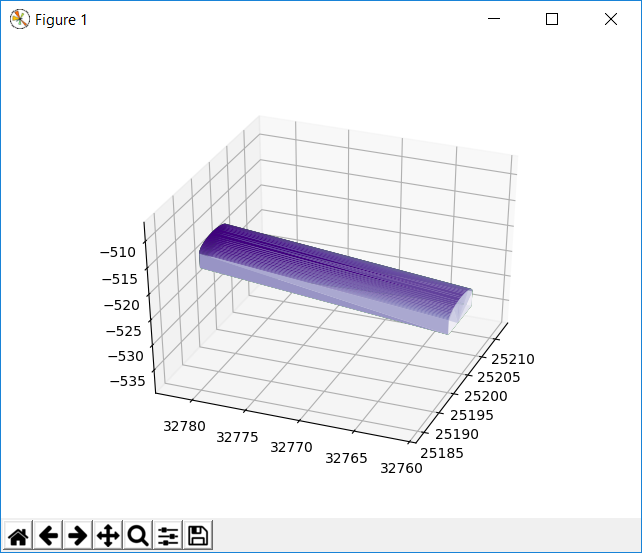
52 fig = plt.figure()

53 ax = fig.add\_subplot(1, 1, 1, projection=**'3d'**)

54 ax.plot\_trisurf(x, y, z, triangles=tri, cmap=plt.cm.Purples)

55 plt.show()

*Результат работы скрипта:*



Для получения геометрии каркаса нецелесообразно использовать библиотеку sqlite3, поскольку триангуляция хранится в бинарном формате BLOB. Для чтения геометрии используется функция Micromine «Экспорт каркасов» (строки 7-20). Полученные файлы точек и треугольников триангуляции можно прочитать с помощью встроенного в Micromine API, модуль MMpy (строки 24-33). Визуализация каркаса в строках 51-55 происходит с помощью метода plot\_trisurf()из библиотеки matplotlib.

Все использованные в данной инструкции файлы вы можете найти здесь.

Полезные ссылки:

* [Официальная документация по модулю sqlite3](https://docs.python.org/3.5/library/sqlite3.html)
* [Официальная документация по API для написания скриптов в Micromine](http://webhelp.micromine.com/mm/16.1/russian/Micromine.htm#../Subsystems/mmmacro/Content/IDH_SCRIPT_API.htm%3FTocPath%3D%25D0%259D%25D0%25B0%25D0%25BF%25D0%25B8%25D1%2581%25D0%25B0%25D0%25BD%25D0%25B8%25D0%25B5%2520%25D1%2581%25D0%25BA%25D1%2580%25D0%25B8%25D0%25BF%25D1%2582%25D0%25BE%25D0%25B2%2520%25D0%25B8%252)
* [Уроки по matplotlib](https://matplotlib.org/tutorials/index.html)
* [Удобная программа для просмотра и изменения баз данных SQLite с графическим интерфейсом - DB Browser for SQLite](http://sqlitebrowser.org/)

1. В общем случае данный метод является небезопасным (см. [здесь](https://stackoverflow.com/questions/13613037/is-this-python-code-vulnerable-to-sql-injection-sqlite3)). [↑](#footnote-ref-1)