axipy

Версия 5.2

ООО ЭСТИ

20 февраля, 2024

Оглавление

1 О библиотеке

Библиотека **axipy** обеспечивает взаимодействие с ГИС Аксиома для решения геоинформационных задач.

axipy использует библиотеку PySide2 (Qt for Python). Библиотека PySide2 имеет лицензию LGPL. Программу, разработанную с использованием библиотеки PySide2, можно лицензировать на свое усмотрение, включая проприетарные и коммерческие лицензии.

Документация к **axipy** состоит из двух частей:

- руководство разработчика;
- справочник функций.

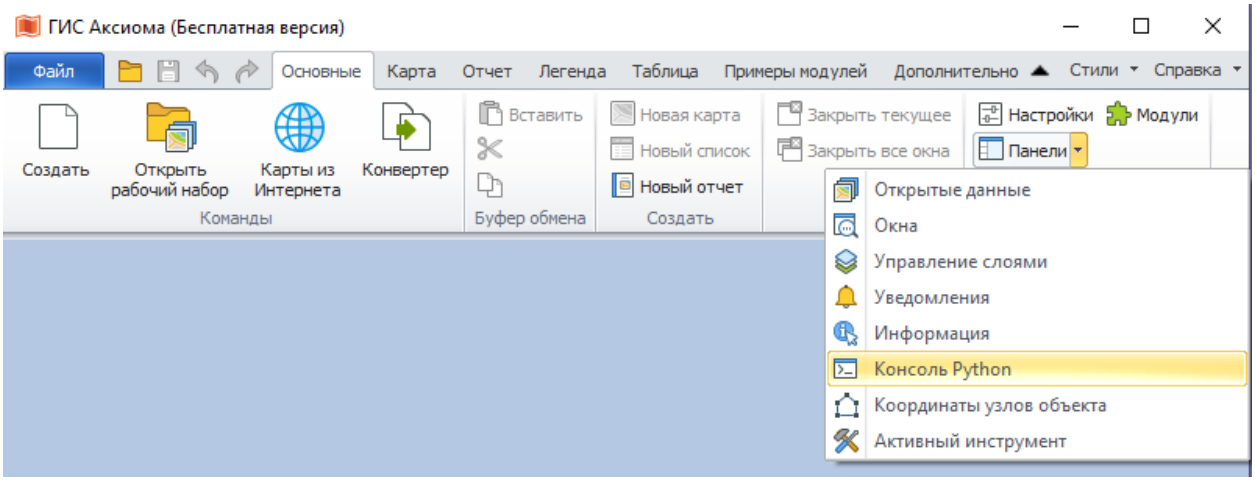
Руководство разработчика логически структурировано. В нем описаны общие принципы решения базовых задач по обработке геопространственных данных. Рекомендуется читать его в прямом порядке от начала до конца, чтобы получить общее представление о возможностях, предоставляемых библиотекой.

Полное описание классов, свойств, методов, исключений, функций и их параметров содержится в справочнике функций.

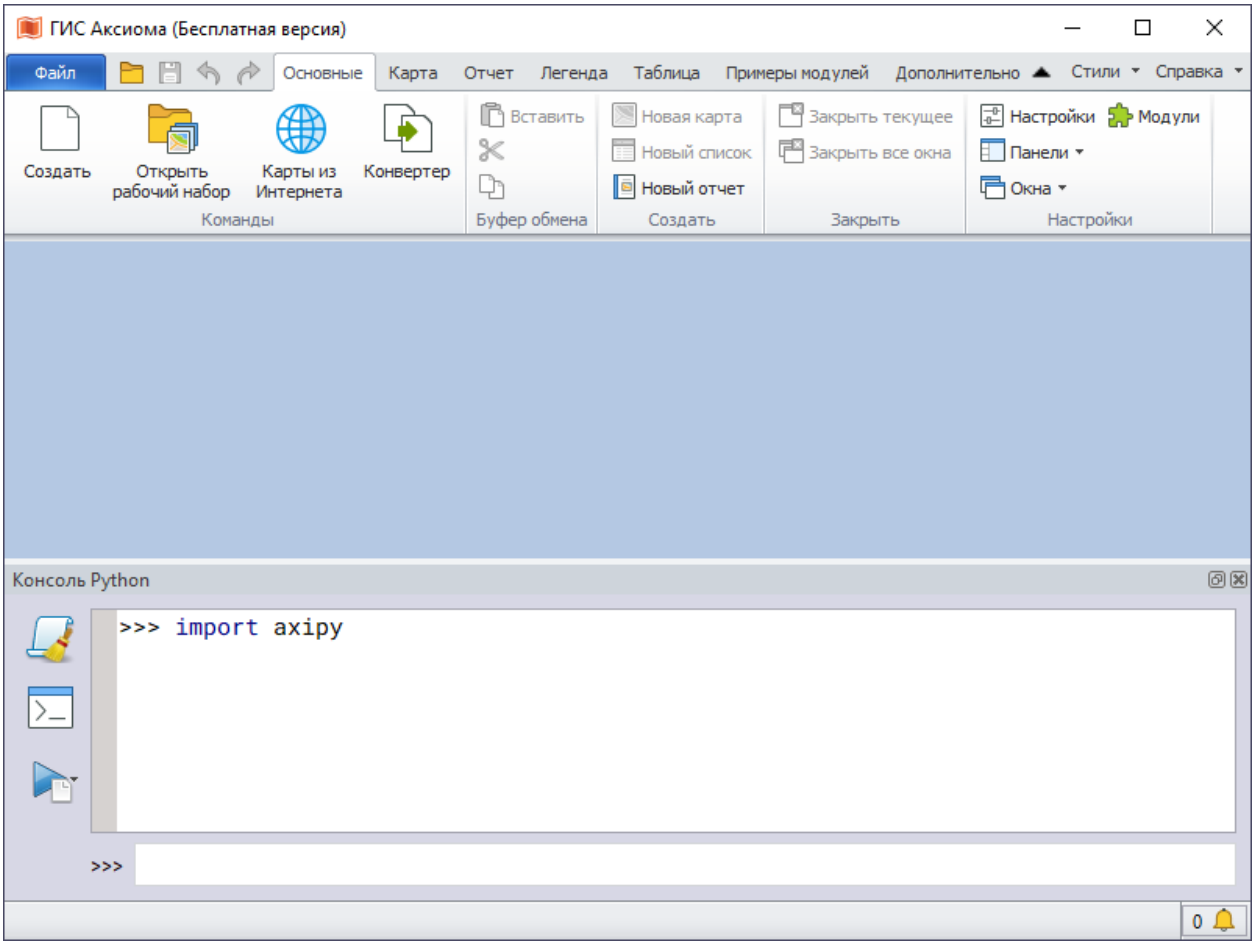
2 Использование в Аксиоме

В Аксиоме есть встроенные средства для использования axipy - консоль Python и редактор кода.

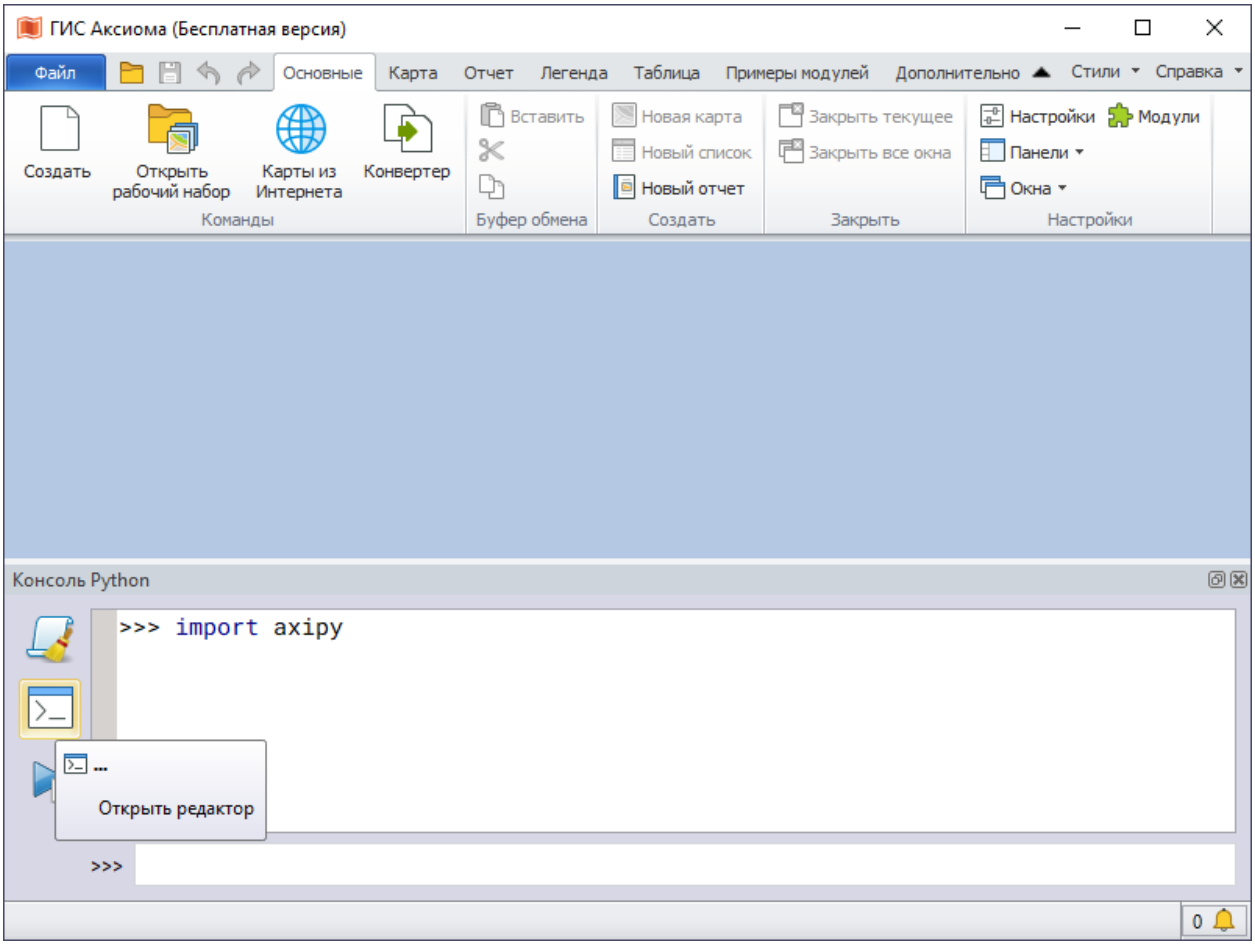
Чтобы открыть консоль Python, на вкладке «Основные» нажмите кнопку «Панели» и в появившемся списке выберите «Консоль Python».



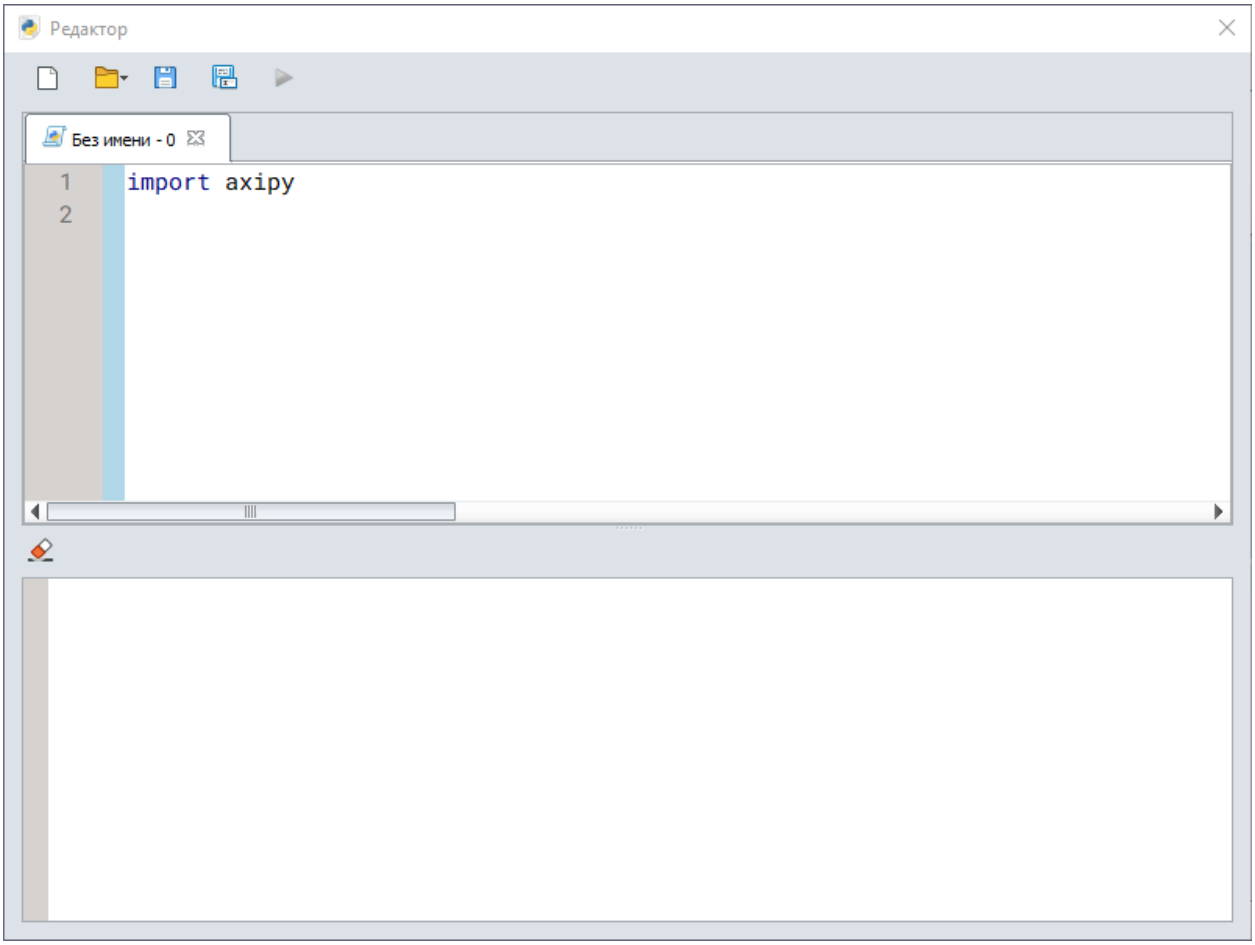
Открытая консоль Python:



Чтобы открыть редактор кода, нажмите кнопку «Открыть редактор» на панели «консоль Python».



Открытый редактор кода:

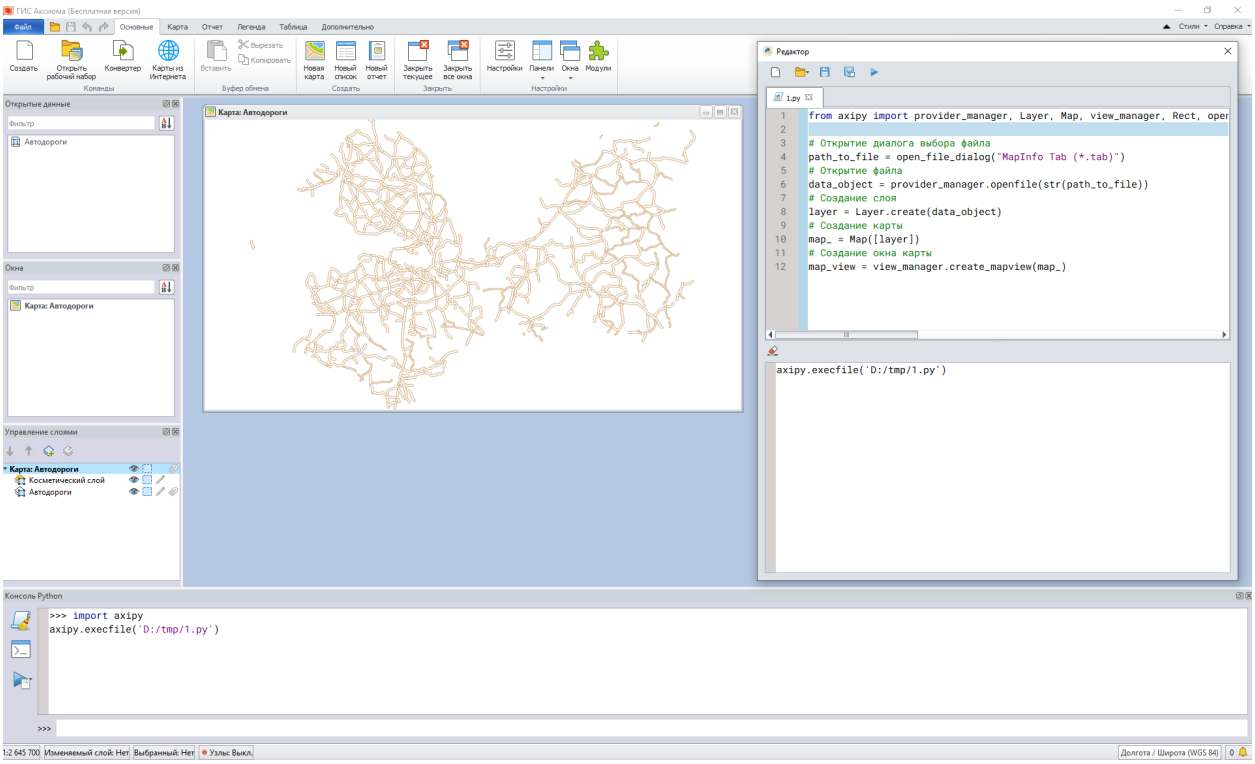


С помощью axipy можно открыть географические данные и показать их на карте.

В примере используются данные с сайта Аксиомы «Обзорная карта Ленинградская область». Скачать данные можно по ссылке https://axioma-gis.ru/download (Раздел «Документация и данные»).

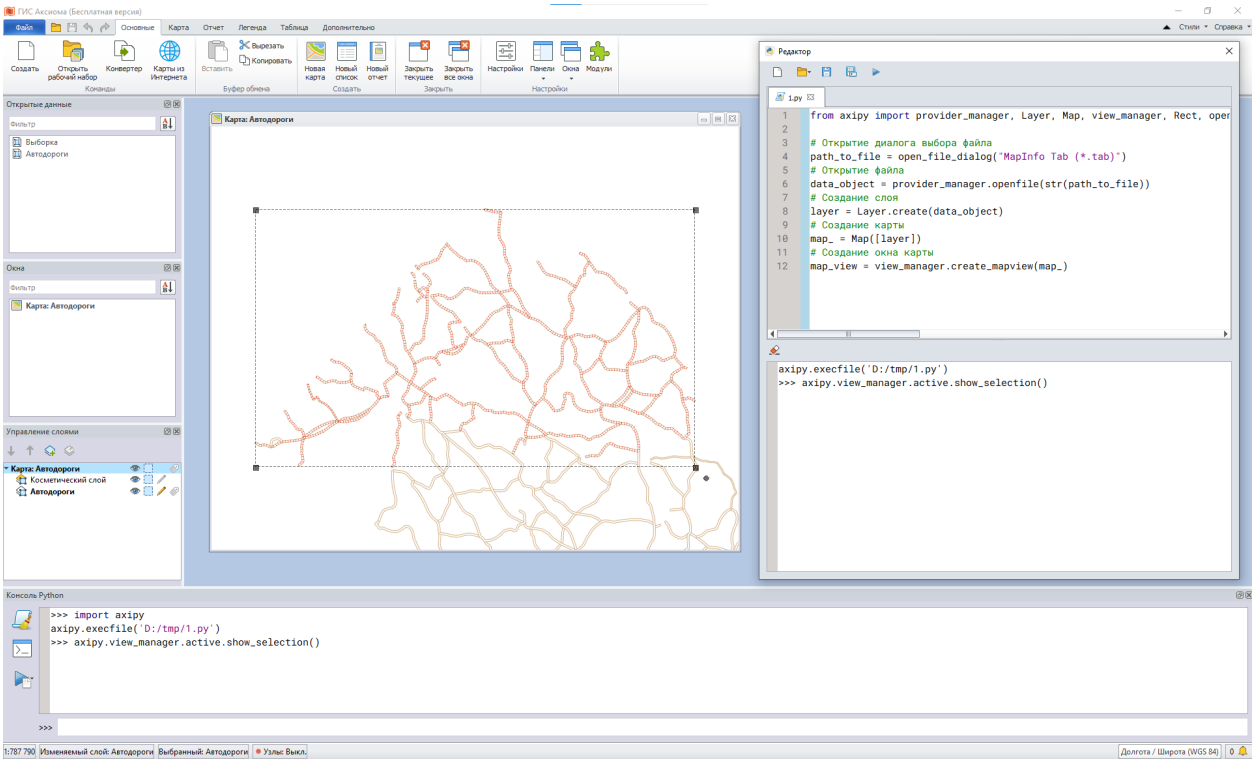
Для выполнения примера, нужно скопировать код ниже и выполнить его в редакторе кода Аксиомы.

```
from axipy import provider_manager, Layer, Map, view_manager, Rect, open_file_dialog

# Открытие диалога выбора файла
path_to_file = open_file_dialog("MapInfo Tab (*.tab)")
# Открытие файла
data_object = provider_manager.openfile(str(path_to_file))
# Создание слоя
layer = Layer.create(data_object)
# Создание карты
map_ = Map([layer])
# Создание окна карты
map_view = view_manager.create_mapview(map_)
```



С открытой картой можно взаимодействовать и из консоли Python, например, приблизить выбранные объекты.

```
axipy.view_manager.active.show_selection()
```



3 Использование среды разработки

При создании плагина или скрипта для ГИС Аксиома рекомендуется использовать интегрированные среды разработки (IDE). Среда разработки предоставляет полезные возможности, такие как: подсветка кода, автодополнение, отладка, система контроля версий и другие.

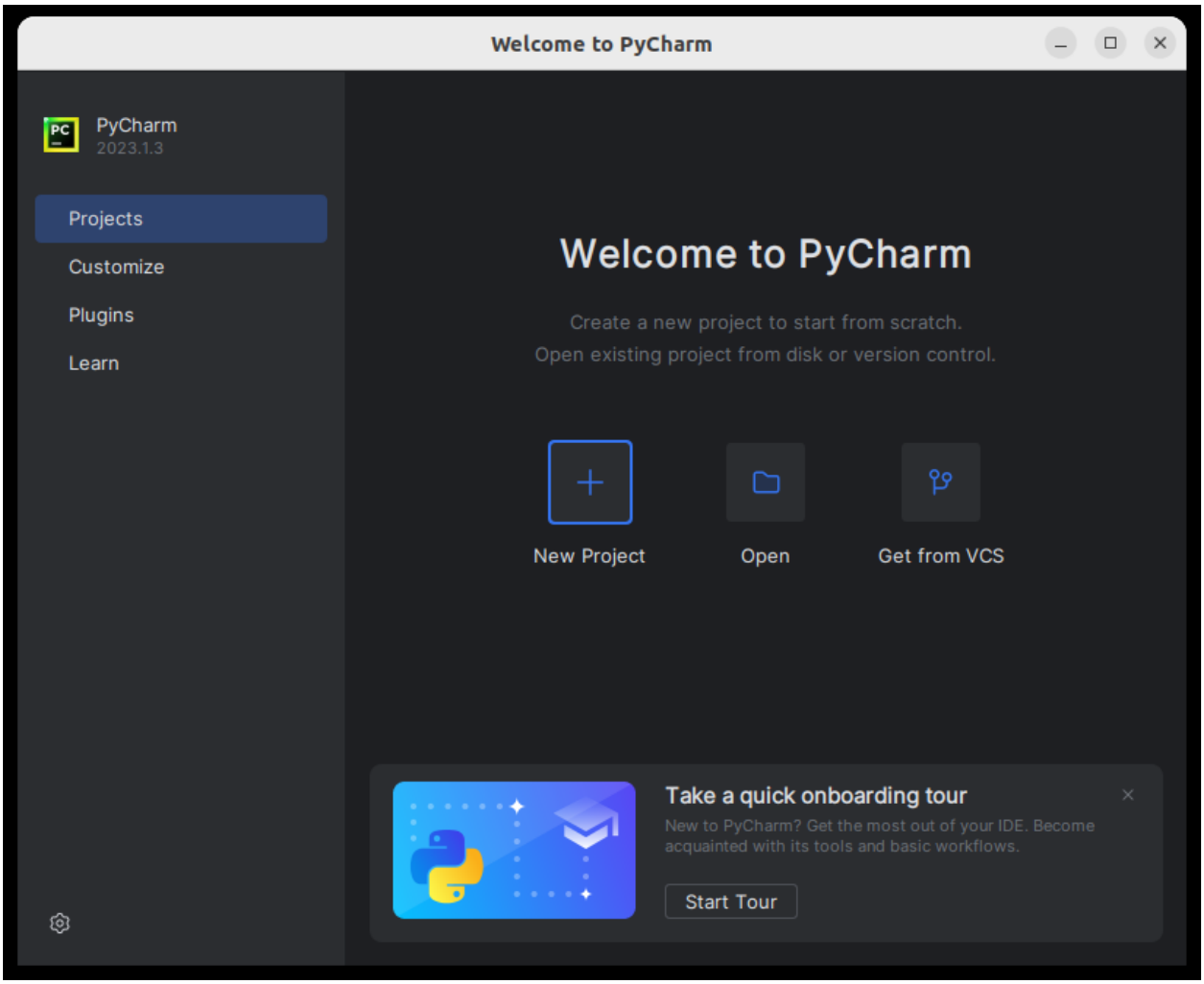
Для использования axipy из среды разработки необходимо вызвать функцию `axipy.init_axioma()`. Чтобы запустить Аксиому из среды разработки под режимом отладки нужно выполнить код:

```
from axipy import init_axioma, mainwindow

# Инициализация ядра
app = init_axioma()
# Создание и отображение главного окна Аксиомы
mainwindow.show()
# Запуск основного цикла приложения
app.exec_()
```

Одной из популярных сред разработки является PyCharm от JetBrains. У PyCharm есть две версии: платная PyCharm Professional и бесплатная PyCharm Community Edition. Скачать бесплатную версию PyCharm Community Edition можно по ссылке: https://www.jetbrains.com/pycharm/download .

3.1 PyCharm Windows

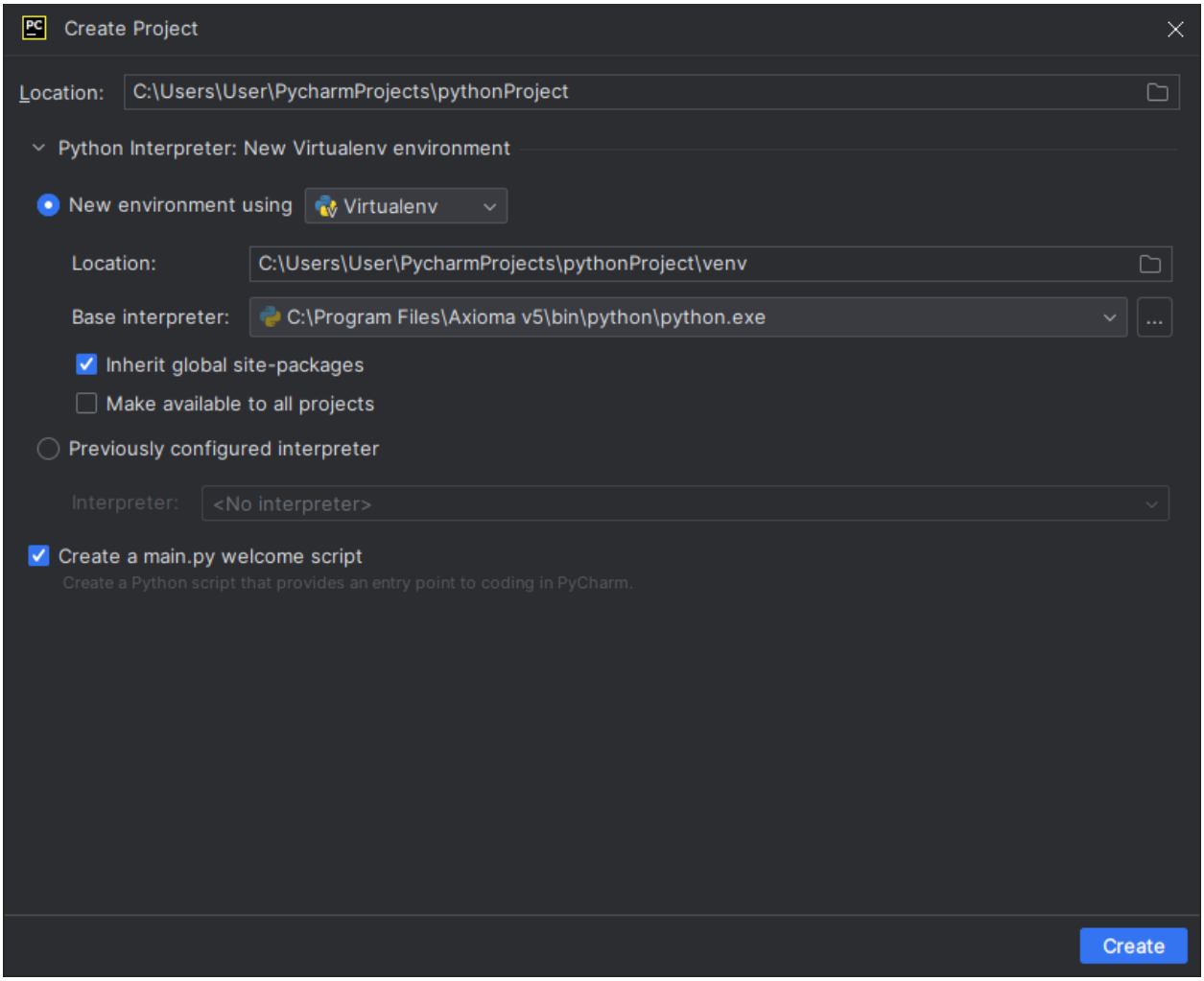
После запуска PyCharm нужно создать новый проект нажав кнопку “New Project” в начальном окне.



В настройках проекта следует в поле Location указать путь к папке проекта. В этой папке будут храниться используемые файлы. Для работы с axipy требуется создать новое виртуальное окружение Python. Для этого нужно:

- Отметить New environment using Virtualenv;
- В поле Location указать путь к создаваемому виртуальному окружению;
- В поле Base interpreter указать путь к Python ГИС Аксиома (C:Program FilesAxioma v5binpythonpython.exe);
- Требуется установить галочку Inherit global site-packages, чтобы Python смог обнаружить axipy;
- Можно отметить Select the Make available to all projects, чтобы использовать виртуальное окружения в других проектах и не настраивать его каждый раз.

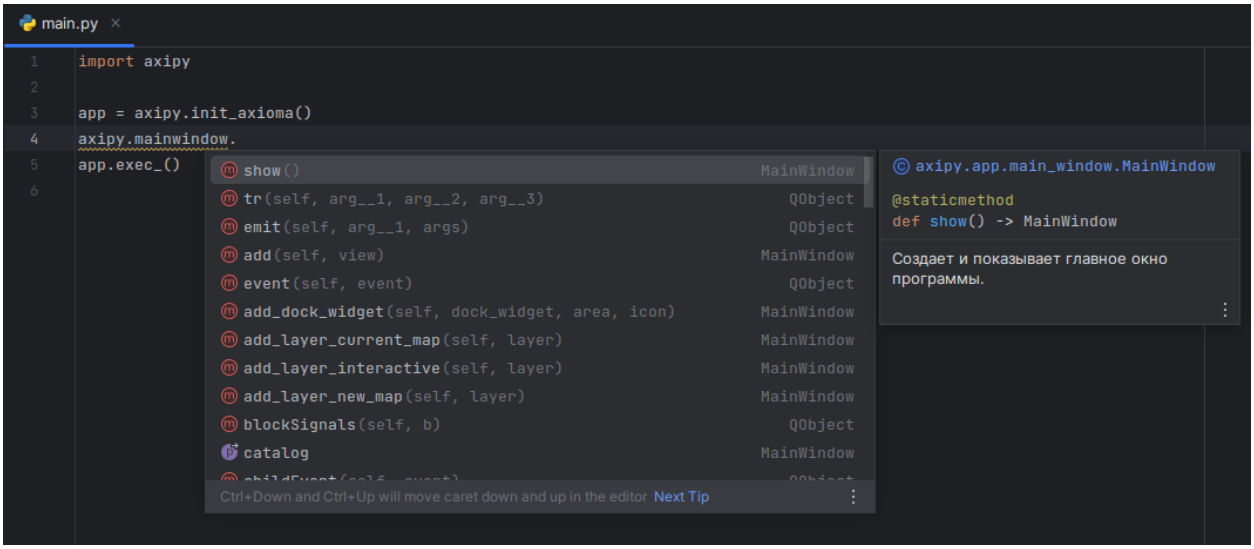
Галочку Create a main.py welcome script можно оставить нажатой, для того чтобы PyCharm создал новый py файл.



После создания проекта, чтобы проверить работу PyCharm с axipy можно выполнить следующий скрипт:

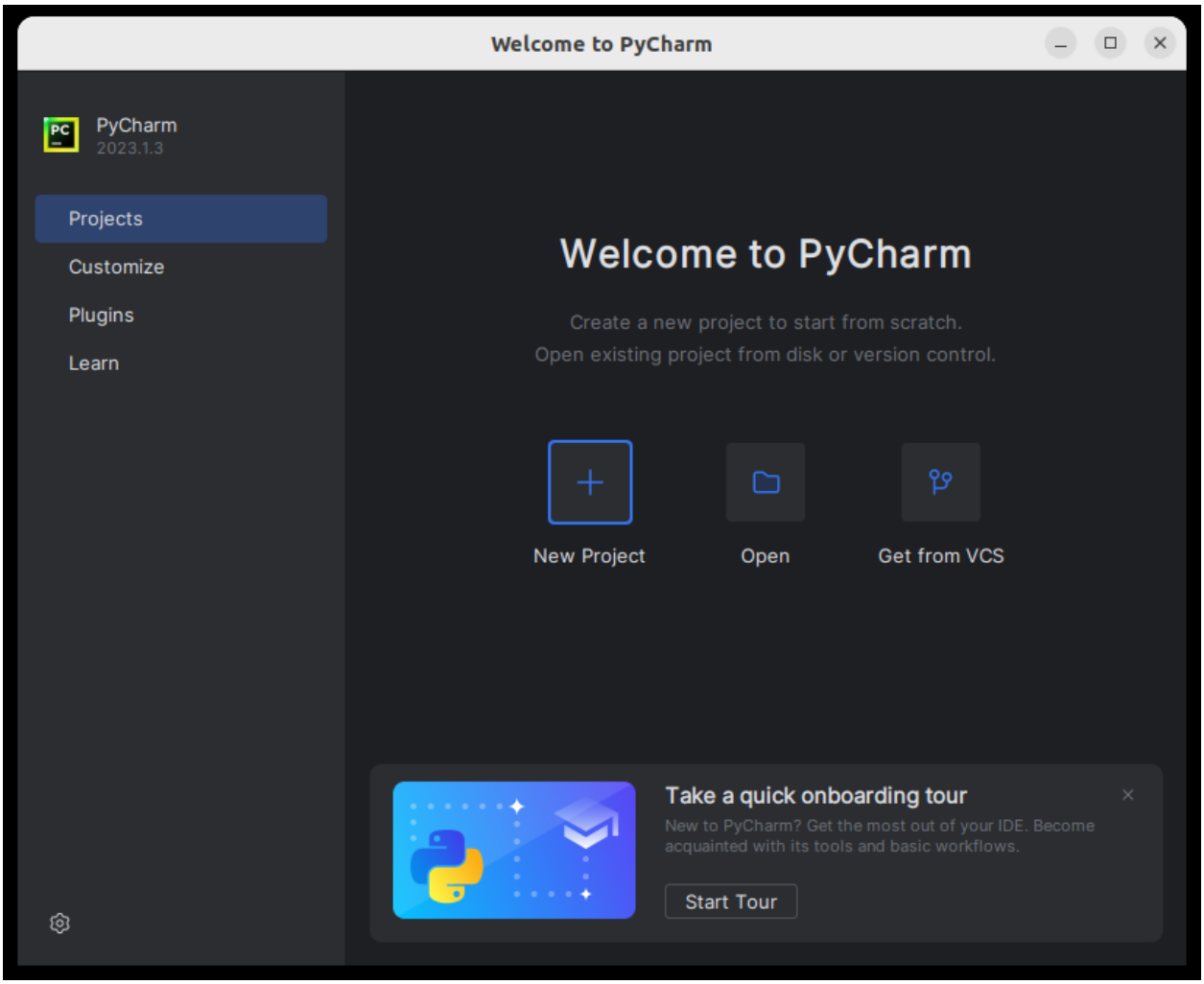
```
from axipy import init_axioma, mainwindow

app = init_axioma()
mainwindow.show()
app.exec_()
```

В процессе ввода кода PyCharm будет отображать подсказки. Чтобы подсказки стали более подробными нажмите Ctrl-Shift-Q для отображения описания.



3.2 PyCharm Linux

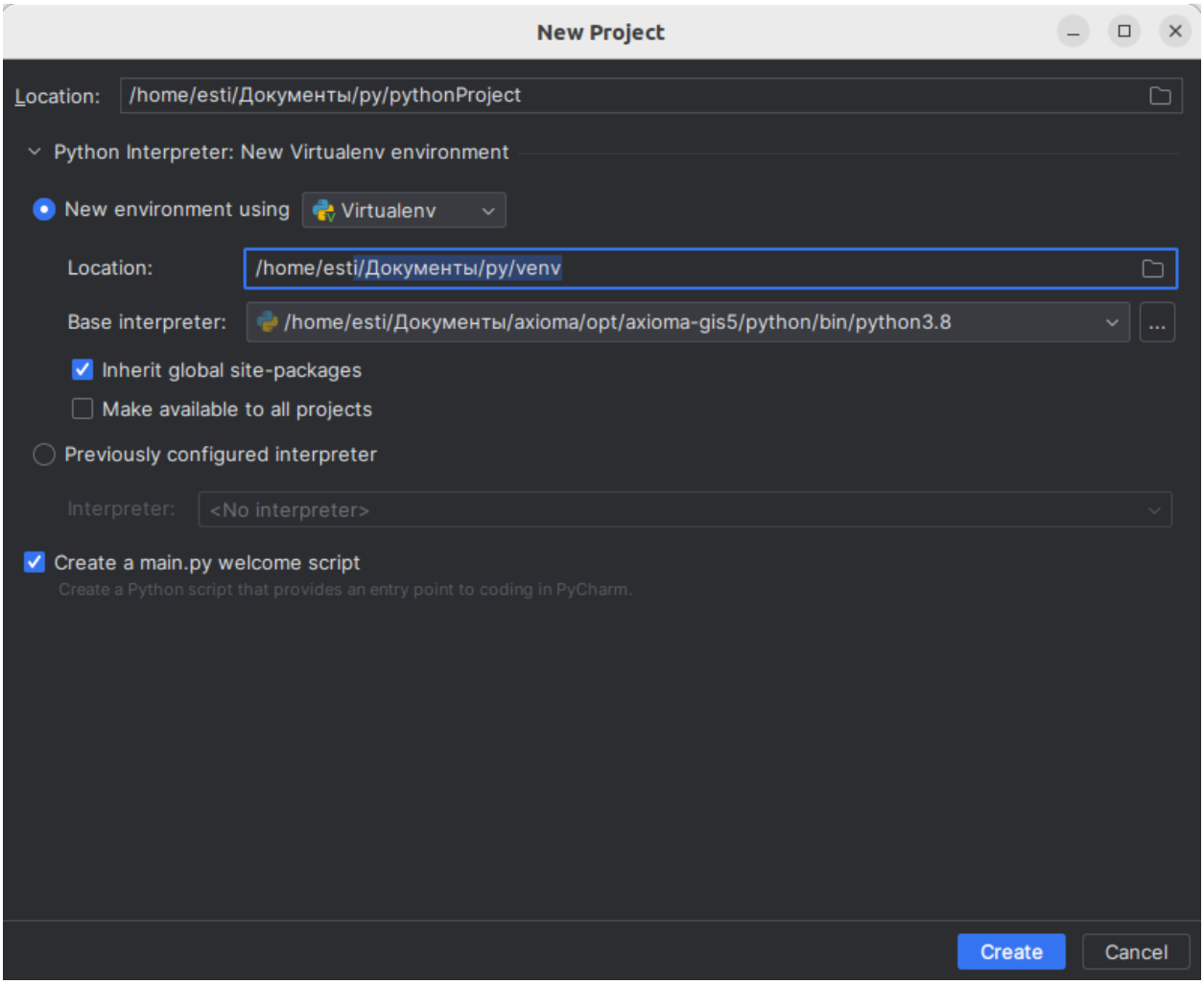
После запуска PyCharm нужно создать новый проект нажав кнопку “New Project” в начальном окне.



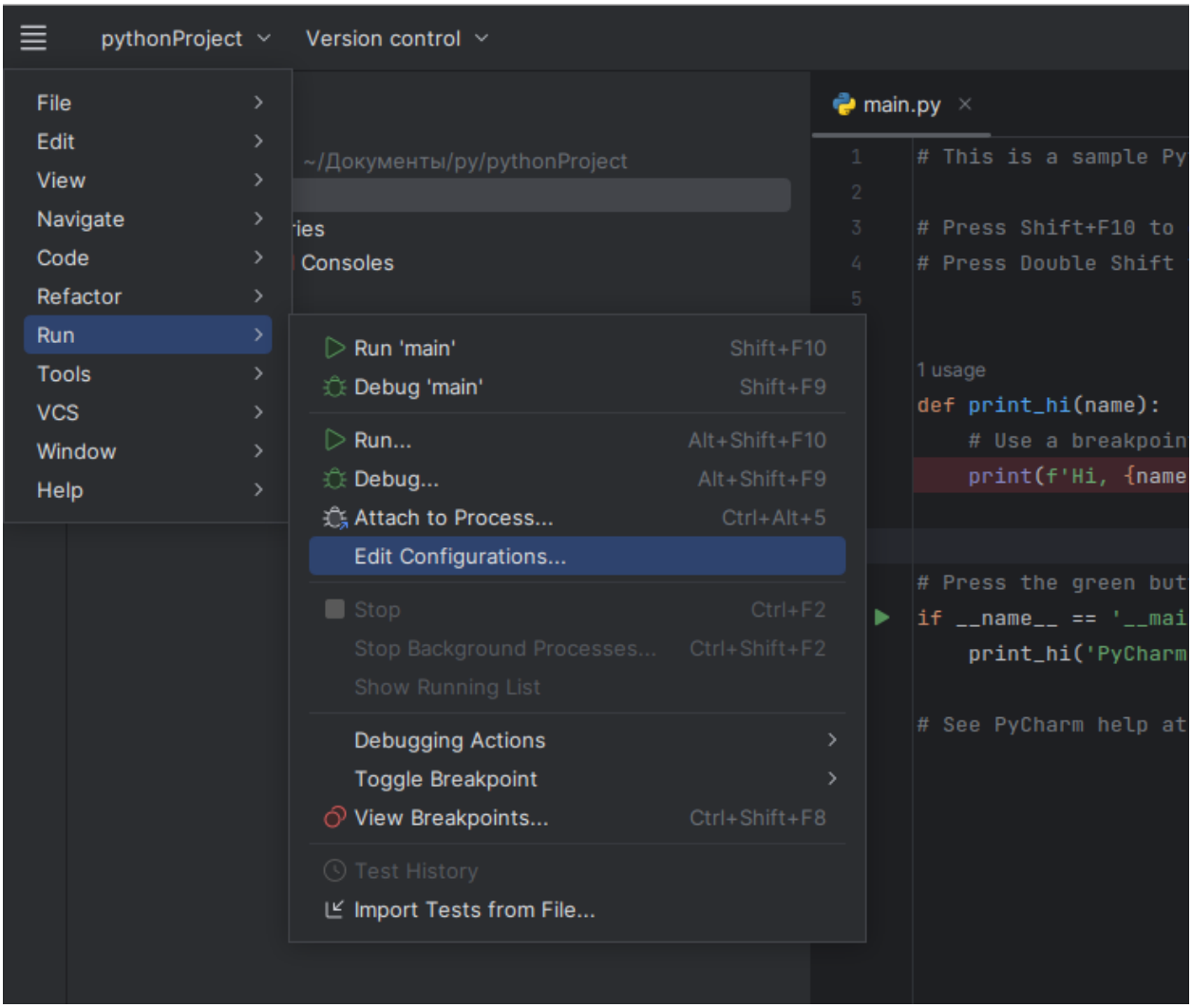
В настройках проекта следует в поле Location указать путь к папке проекта. В этой папке будут храниться используемые файлы. Для работы с axipy требуется создать новое виртуальное окружение Python. Для этого нужно:

- Отметить New environment using Virtualenv;
- В поле Location указать путь к создаваемому виртуальному окружению;
- В поле Base interpreter указать путь к Python ГИС Аксиома. В папке opt/axioma-gis5/python/bin/python3.8 файл python3.8;
- Требуется установить галочку Inherit global site-packages, чтобы Python смог обнаружить axipy;
- Можно отметить Select the Make available to all projects, чтобы использовать виртуальное окружения в других проектах и не настраивать его каждый раз.

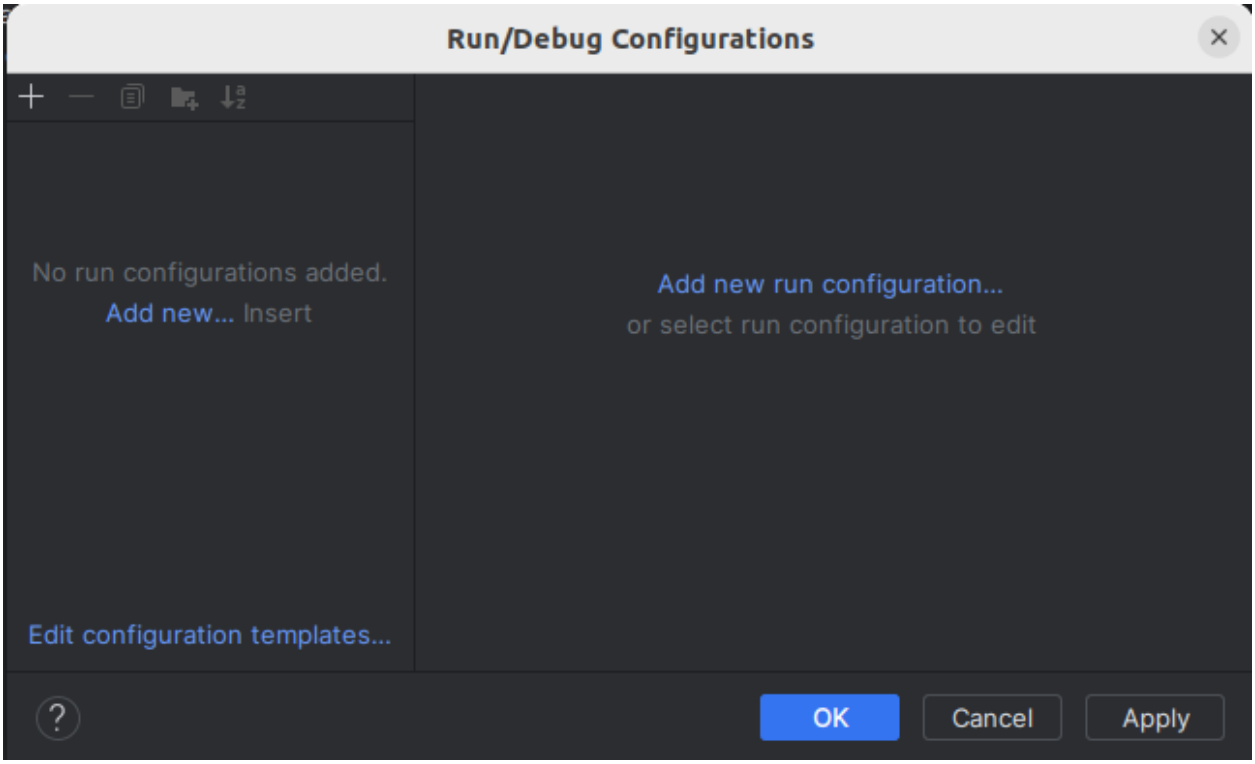
Галочку Create a main.py welcome script можно оставить нажатой, для того чтобы PyCharm создал новый py файл.



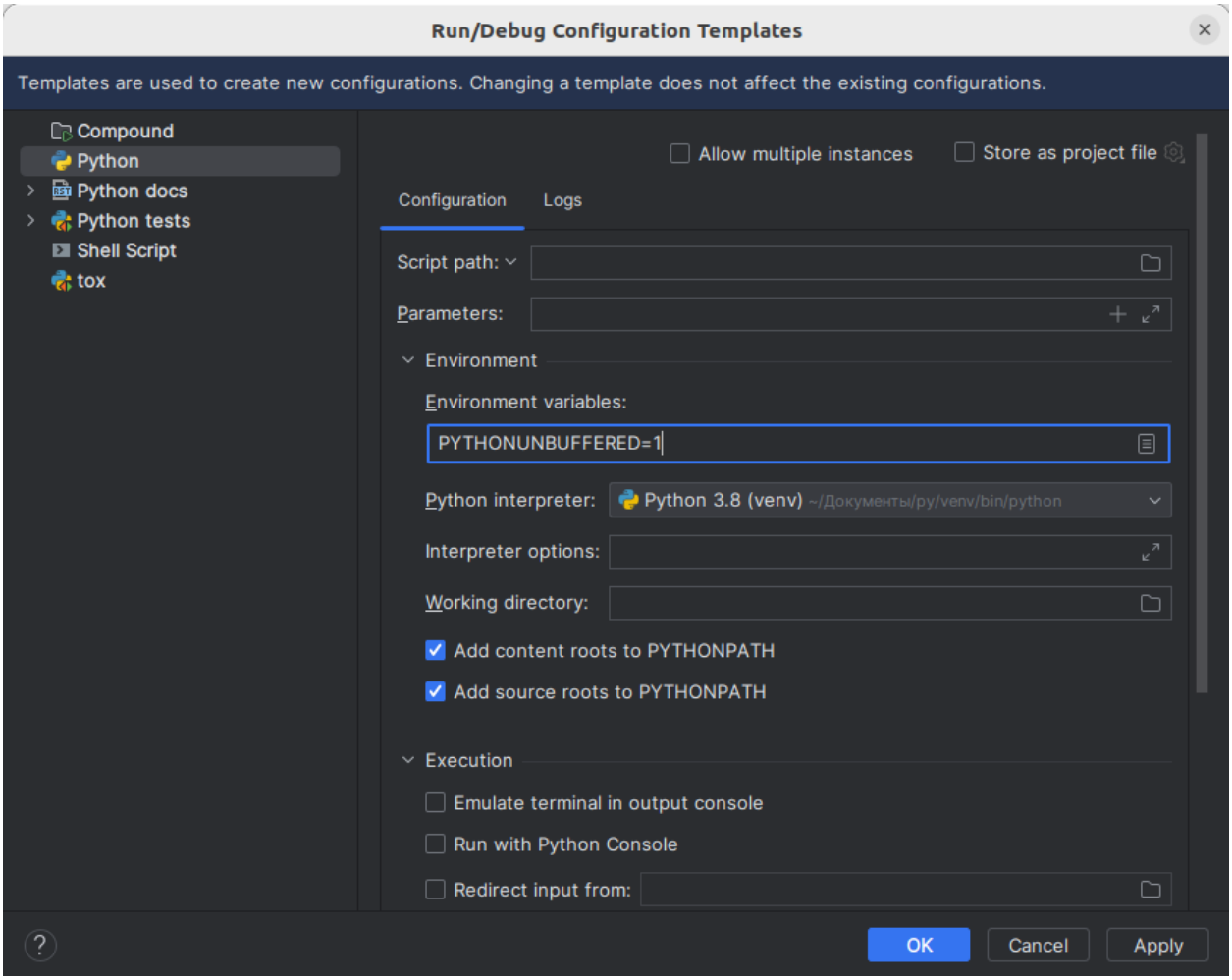
После создания проекта очень важно настроить переменные среды, для того чтобы PyCharm смог обнаружить динамические библиотеки axipy. Для этого нужно дождаться окончательного построения проекта PyCharm, а затем вызвать из меню «Run – Edit Configuration…».



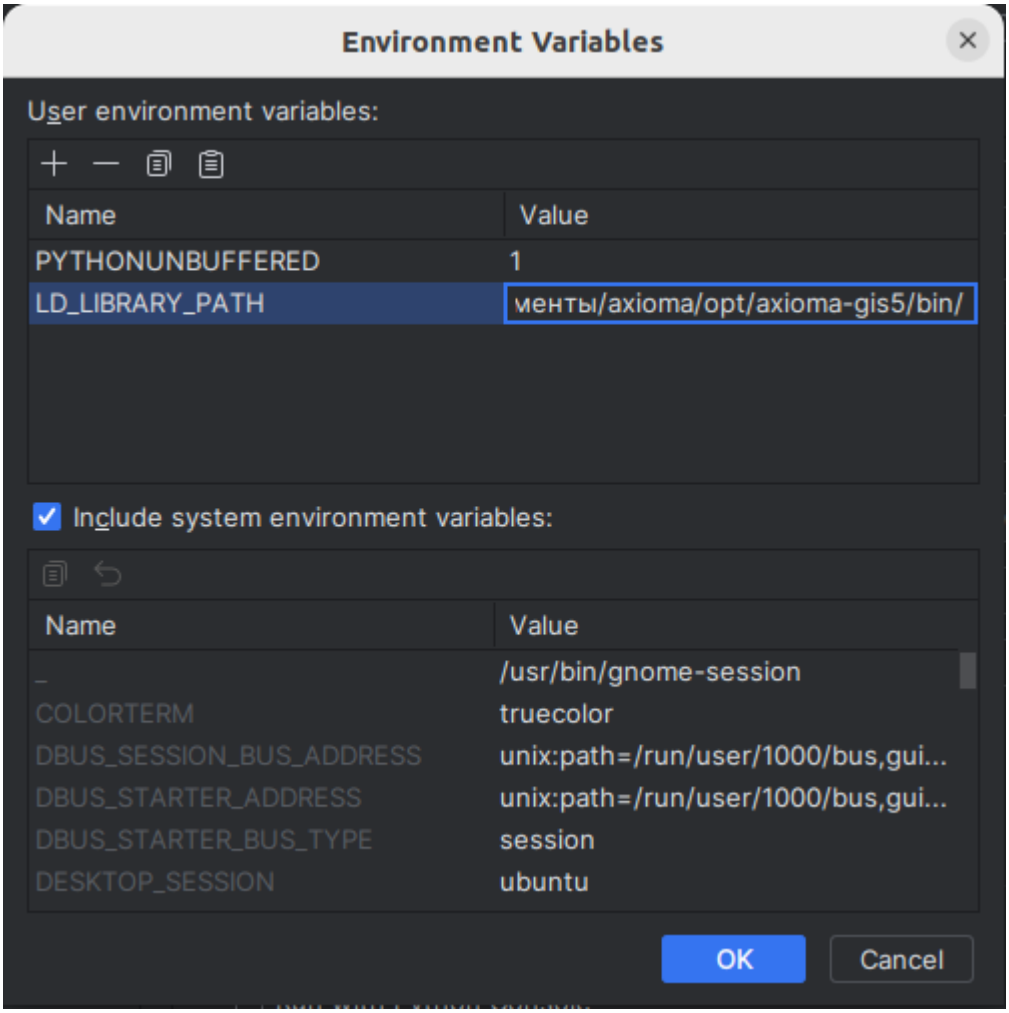
После появления окна настроек конфигураций следует удалить существующие конфигурации в левой части окна (если таковые имеются). А затем нажать Edit configuration templates... в нижней левой части окна.



В появившемся окна нужно добавить переменную окружения LD_LIBRARY_PATH с указанием на папку opt/axioma-gis5/bin/



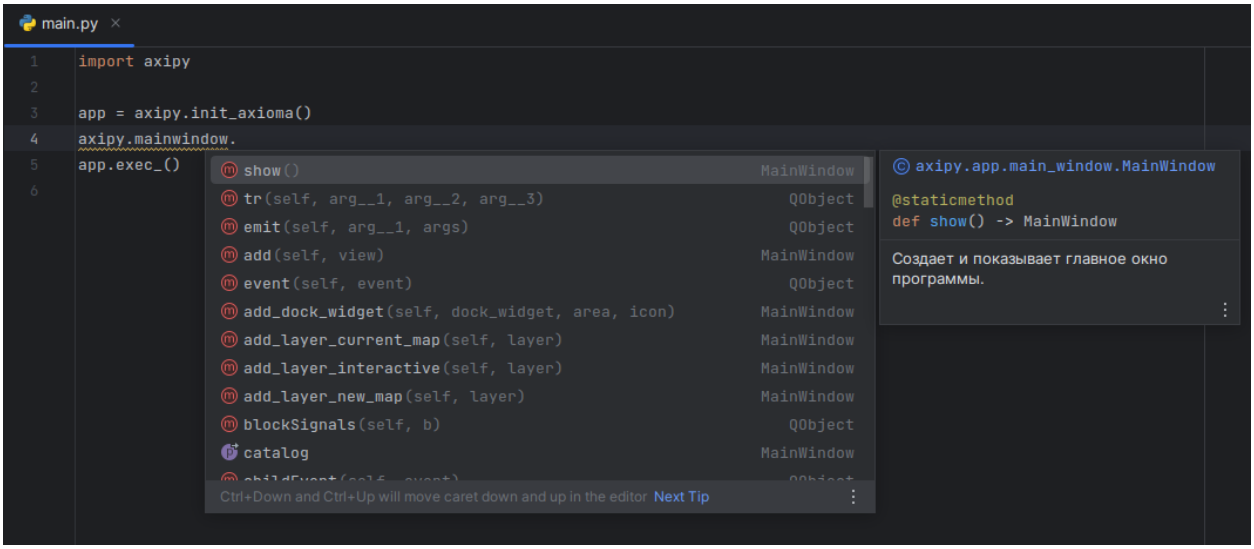
Для этого нужно внести изменение в поле Environment variables нажав на кнопку справа от поля.



Чтобы проверить работу PyCharm с axipy можно выполнить следующий скрипт:

```
from axipy import init_axioma, mainwindow

app = init_axioma()
mainwindow.show()
app.exec_()
```

В процессе ввода кода PyCharm будет отображать подсказки. Чтобы подсказки стали более подробными нажмите Ctrl-Shift-Q для отображения описания.



Другая популярная среда разработки Visual Studio Code (VS Code) от MicroSoft. VS Code бесплатна и скачать ее можно по ссылке: https://code.visualstudio.com .

3.3 VS Code Windows

Настройка

Visual Studio Code - кроссплатформенная среда разработки с бесплатной лицензией. Ее можно скачать с официального сайта: Visual Studio Code.

Перед началом в установленной **Visual Studio Code** необходимо установить расширение Python. Это можно сделать во вкладке Extensions (Ctrl+Shift+X).

Создадим папку с проектом `axioma_hello_world`.

Хорошей практикой при разработке на языке Python является создание виртуального окружения virtualenv. В командной строке (Ctrl+`) Visual Studio Code выполним:

```
"C:\Program Files\Axioma v5\bin\python\python.exe" -m venv .venv --system-site-
↪packages
```

Примечание: Расположение установленной ГИС Аксиома может отличаться.

Важно: Обязательно использовать Python, поставляемый вместе с ГИС Аксиома.

Создастся виртуальное окружение в папке `.venv`. Visual Studio Code сразу найдет эту папку и предложит использовать ее окружение для текущего проекта.

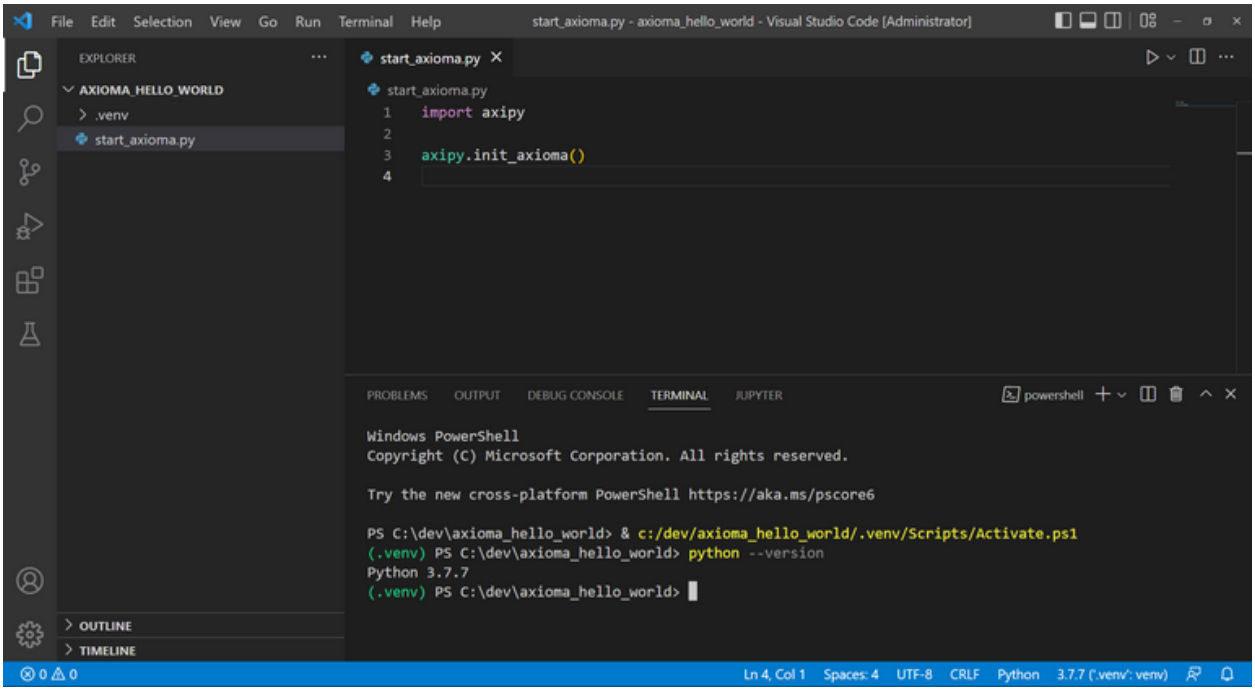
Чтобы убедиться, что окружение работает корректно, выведем версию интерпретатора Python; создадим простой стартовый скрипт и увидим, что ГИС Аксиома запускается из среды разработки:

Список 1: Содержимое файла start_axioma.py

```
from axipy import init_axioma, mainwindow

app = init_axioma(load_python_plugins=True)
mainwindow.show()
app.exec_()
```

Список 2: Версия интерпретатора Python.

```
python --version
```



Создадим простой модуль:

```
axioma_hello_world  # папка с проектом
├── ru_axioma_gis_axipy_example_plugin_minimal  # модуль
│   ├── __init__.py
│   └── manifest.ini
└── start_axioma.py  # стартовый скрипт
```

См.также:

Раздел Модули.

Совет: Готовый скрипт `start_axioma.py` и пример плагина `ru_axioma_gis_axipy_example_plugin_minimal` можно найти в папке с установленной ГИС Аксиома.

Список 3: Содержимое файла manifest.ini

```
name=Пример модуля axipy - Минимальный
description=Модуль для демонстрации возможностей разработки на Python
```

Список 4: Содержимое файла __init__.py

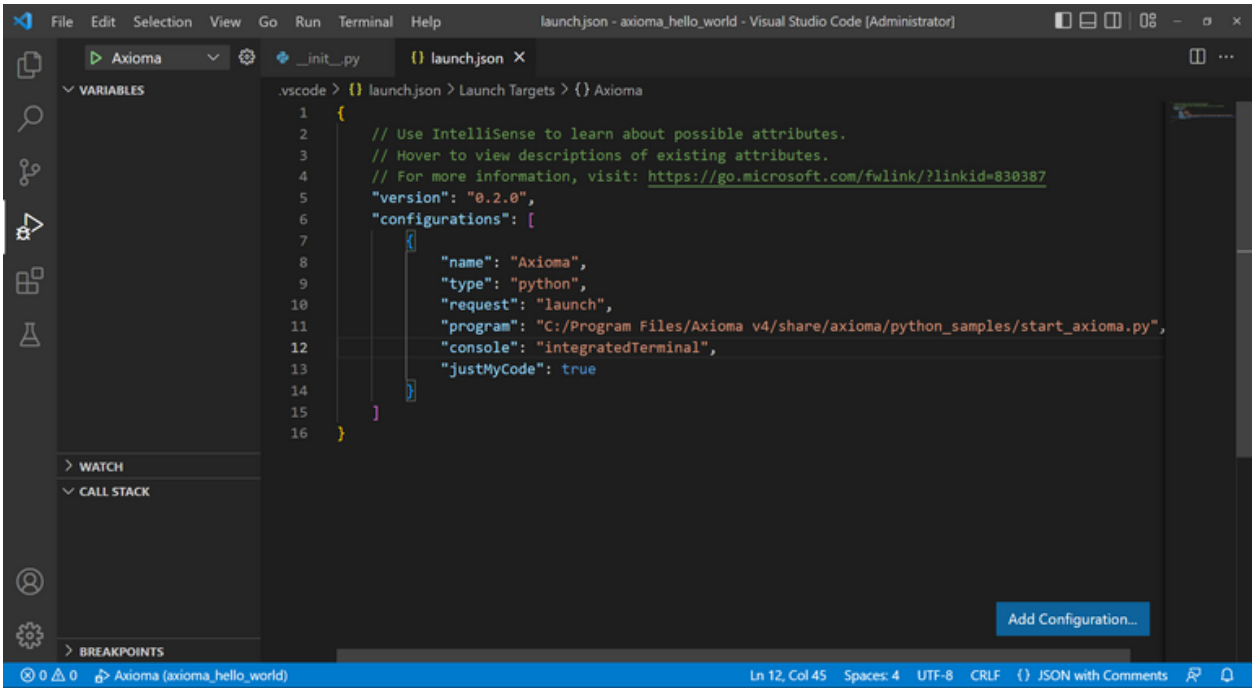
```
from PySide2.QtWidgets import QMessageBox
from axipy import AxiomaPlugin, Position


class Plugin(AxiomaPlugin):
    def load(self):
        self.__action = self.create_action('Пример действия',
                icon=':/icons/share/32px/run3.png', on_click=self.show_message)
        position = Position('Основные', 'Команды')
        position.add(self.__action)
        self.__action.action.setToolTip('Всплывающая подсказка')

    def unload(self):
        self.__action.remove()

    def show_message(self):
        QMessageBox.information(None, 'Сообщение',
                'Пример выполнения действия по нажатию кнопки')
```

Запустим ГИС Аксиома из скрипта `start_axioma.py` и в диалоге Модули -> Настройки добавим путь к нашему проекту с модулем. Найдем его в списке найденных модулей и загрузим (можно сравнить действительное расположение модуля). Так образом можно продолжать разрабатывать этот модуль, выгружая и загружая его вновь после внесения изменений.

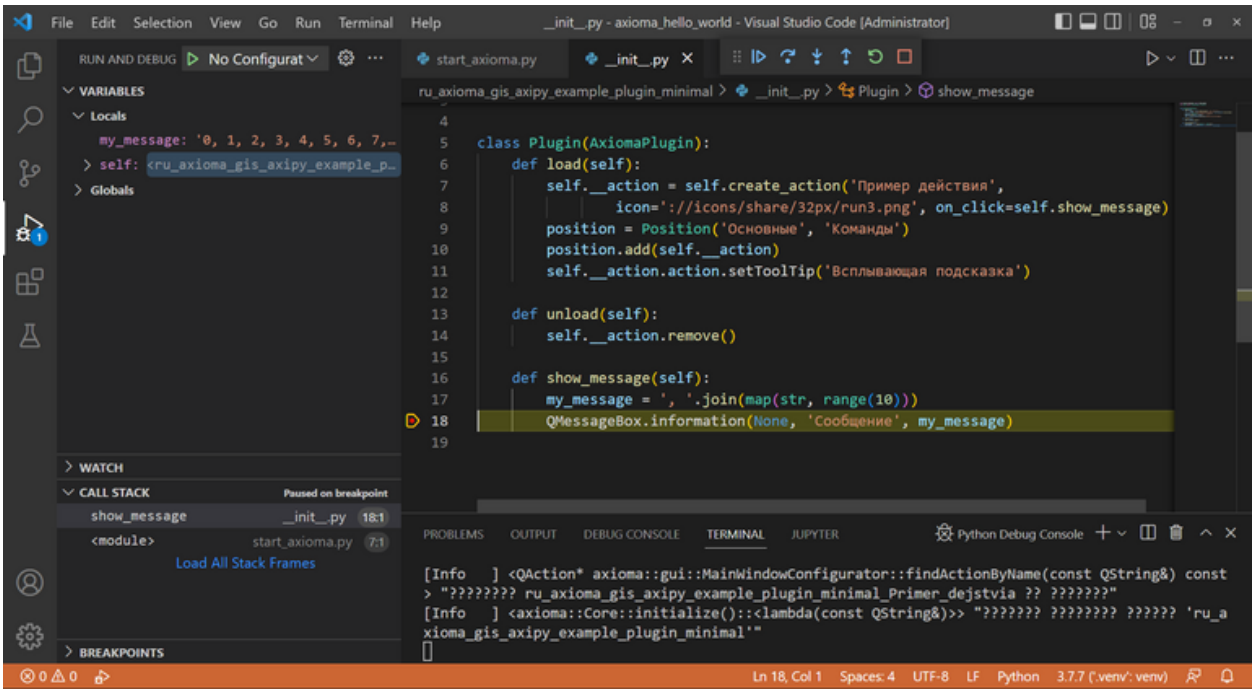
Важно: Чтобы изменения вступили в силу, необходимо выгрузить и повторно загрузить модуль.

Отладка

Проект готов для того, чтобы отлаживать модуль, ставя точки останова в коде. Для удобства можно создать конфигурацию для отладки, которая будет сама запускать стартовый скрипт. Создайте конфигурацию для отладки - Visual Studio Code создаст файл `launch.json`. Стартовый скрипт есть в папке с установленной ГИС Аксиома. Можно указать путь сразу к нему, и убрать из проекта с модулем.



Теперь при запуске этой конфигурации можно отлаживать модуль, ставя точки останова, анализируя стек вызовов и содержимое переменных.



4

Системы Координат

Термины «проекция» и «координатная система» иногда используются один вместо другого, но, на самом деле, понятия, которые они отражают, различны.

Проекция – это уравнения или наборы уравнений, которые содержат математические параметры для карты. Точное число и природа параметров зависят от типа проекции. Проекция – это метод уменьшения искажений карты, вызванных кривизной земной поверхности или, точнее говоря, проекция компенсирует недостатки отображения карты на плоскости в двух измерениях, в то время как координаты существуют в трёх измерениях.

Координатная система – когда параметрам проекции присваиваются определенные значения, они становятся системой координат. Система координат – это набор параметров, описывающих координаты, одна из которых является проекцией.

Системы Координат (СК) представлены типом `CoordSystem`. Объекты типа `CoordSystem` могут быть созданы, используя:

- код EPSG - European Petroleum Survey Group;
- строку MapInfo PRJ;
- строку WKT - Well-known text;
- строку PROJ;
- единиц измерения - для создания СК в план-схеме;

Полный список доступных функций создания систем координат содержится в документации к классу `CoordSystem`.

Примечание: Для выполнения примеров:

```
from axipy import CoordSystem, CoordTransformer
```

Список 1: Например

```
merc = CoordSystem.from_epsg(3395)
print(merc.name)
'''
```

(continues on next page)

(продолжение с предыдущей страницы)

```
>>> Меркатора WGS84
'''
```

Функция `from_string()` позволяет создавать СК из “универсального представления” - строки с префиксом типа, двоеточием и значением. Возможные префиксы: proj, wkt, epsg, prj.

Список 2: Например

```
crs = CoordSystem.from_string('prj:Earth Projection 12, 62, "m", 0')
print(crs.name)
'''
>>> Робинсона NAD27
'''
```

Трансформация координат

Координаты любой точки земной поверхности в разных системах координат будут различаться, переход от одной системы координат к другой осуществляется с помощью специальных формул преобразований и набора параметров,используемых в этих формулах.

Функционал по переходу координат из одной СК в другую выделен в отельный класс `CoordTransformer`. Он позволяет преобразовывать координаты между двумя СК.

Список 3: Например

```
transformer = CoordTransformer('epsg:4326', 'epsg:26953')
coordinate = (55.76, 37.6)
result = transformer.transform(coordinate)
print(f"Point({result.x}, {result.y})")
'''
>>> Point(8513601.095442554, 9873107.576049749)
'''
```

5 Объекты данных

Разные типы данных обобщены одним абстрактным типом `DataObject`. Он образует иерархию объектов данных различного типа: таблица, растр, грид, чертеж, панорама, и так далее.

5.1 Таблицы

Для того, чтобы мы могли работать как с географической, так и с атрибутивной информацией, данные в ГИС Аксиома организованы в виде групп файлов, имеющих общее имя, но разные расширения.

Пользователь оперирует только с одним файлом из этой группы – так называемым «табличным» файлом, которые имеет расширение TAB. Все файлы из группы автоматически создаются, обновляются и поддерживаются самой программой ГИС Аксиома. Таблица `Table` является наследником класса объекта данных `DataObject`.

5.1.1 Открытие таблиц

Работа с источником данных начинается с открытия объекта данных с помощью функции `openfile()` объекта `provider_manager`. Для таблиц возвращаемый объект данных будет типа `Table`.

```
table = provider_manager.openfile('../path/to/datadir/worldcap.tab')
```

Некоторые форматы могут содержать несколько таблиц в одном файле, например GeoPackage. В таком случае нужно указать в параметре `dataobject=`, какую таблицу из файла вы хотите открыть.

```
table = provider_manager.openfile('../path/to/datadir/example.gpkg', dataobject='world
↪')
```

У таблицы можно узнать провайдер, которым она была открыта - свойство `provider`:

```
print(table.provider)
```

```
>>> 'SqliteDataProvider'
```

Источники данных и дополнительные параметры

Источником данных может быть не только файл. Например, это может быть База Данных. Также для открытия или создания некоторых объектов данных может понадобиться указать дополнительные параметры, применимые только для этого типа источника.

Для открытия таких объектов используйте конкретный провайдер данных из коллекции провайдеров `ProviderManager`. Он содержит все методы, параметры и допустимые значения, для работы с конкретным типом объектов данных.

Если по какой-то причине использовать конкретный провайдер данных не удается, то используется функция `open()`, которая принимает словарь со всеми необходимыми параметрами.

Пример открытия таблицы из базы данных PostgreSQL:

Примечание: Исключительно в качестве примера. Намного проще напрямую использовать провайдер данных `PostgreDataProvider`.

```
definition = {
    "src": "<Адрес сервера БД>",
    "port": 5432,
    "db": "<Имя базы данных>",
    "user": "<Имя прользователя>",
    "password": "<Пароль>",
    "dataobject": '"DataAxi"."World"',
    "provider": "PgDataProvider"
}
table = provider_manager.open(definition)
```

Открытие источников с множеством таблиц

Для получения всех доступных таблиц одного источника используется функция `read_contents()`:

```
contents = axipy.provider_manager.read_contents('../path/to/datadir/example.gpkg')
print(contents)
```

```
>>> ['world', 'worldcap']
```

Для источников с единственной таблицей список будет содержать одно имя:

```
contents = axipy.provider_manager.read_contents({'src': '../path/to/datadir/worldcap.
↪tab'})
print(contents)
```

```
>>> ['worldcap']
```

5.1.2 Схема таблицы

Записи таблицы имеют фиксированную структуру, повторяющую столбцы таблицы. Схема представлена типом `Schema`. Свойство `coordsystem` указывает на то, что таблица является пространственной. Атрибуты `Attribute` перечислены в том же порядке, что и столбцы таблицы.

Совет: Схему можно рассматривать как список `list` атрибутов `Attribute`. Манипулировать атрибутами схемы можно также, как элементами списка.

Схему таблицы можно получить, используя свойство `schema`.

Например:

```
schema = table.schema()
```

Схема имеет простую стандартную структуру и с ней просто работать. Например, можно легко вывести все имена атрибутов:

```
schema.attribute_names
# эквивалентно
[attr.name for attr in schema]
```

Атрибуты схемы

Атрибут представлен типом `Attribute`. Его главные параметры - это имя `name` и тип `typedef`. Тип атрибута представляется строкой. В нем может быть указана максимальная длина - для строк и десятичного типа через двоеточие, например, `string:254`. И точность - для десятичного типа через точку, например, `decimal:7.3`.

Доступные типы:

Тип	Описание
string	строка
int	целое число
double	вещественное число с плавающей запятой
decimal	вещественное число с фиксированной запятой
bool	логическое значение
date	дата
time	время
datetime	дата и время

Специальный атрибут Система Координат `coordsystem` содержит значение СК и указывает на то, что таблица пространственная - может содержать геометрию и стиль.

См.также:

Подробнее в главе Системы Координат.

Создание схемы таблицы. Вспомогательные функции

Класс Attribute содержит вспомогательные функции string() и другие для создания атрибутов; для создания схемы таблицы используется конструктор - Schema.

Например, так можно создать схему таблицы:

```
schema = Schema(
    Attribute.string('Столица', 25),
    Attribute.string('Capital', 25),
    Attribute.string('Страна', 30),
    Attribute.string('Country', 30),
    Attribute.decimal('Cap_Pop', 8, 5),
    coordsystem='prj:Earth Projection 12, 62, "m", 0'
)
```

Свойства length и precision позволяют получить длину и точность типа. Список всех свойств описан в справочнике на тип Attribute.

Чтение записей

Объект таблица Table дает возможность работать с записями Feature. Метод items() возвращает итератор (iterator) по записям.

См.также:

Подробнее о записях в главе Записи.

```
features = table.items()
for feature in features:
    # обработка записи
    ...
```

Возвращается именно Итератор. При чтении он движется вперед и в конце становится пустым. Чтобы начать чтение сначала, нужно создать новый итератор.

5.1.3 Создание таблиц

Создавать таблицы несколько сложнее, чем открывать готовые. Для них нужно определить схему, СК, Провайдер и пр. Все эти параметры были рассмотрены выше.

Провайдер может быть задан явно:

```
newtable = provider_manager.tab.create_open('../path/to/datadir/newtable.tab', schema)
```

или найден автоматически из расширения файла:

```
newtable = provider_manager.createfile('../path/to/datadir/newtable.tab', schema)
```

См.также:

tab, createfile().

Добавим в таблицу несколько записей из вселенной Властелин Колец:

```
features_to_insert = [
    Feature({'country': 'Мордор', 'capital': 'Барад-Дур'}),
    Feature(country='Гондор', capital='Минас Тирит'),  # создание с использованием
↪**kwargs
    Feature({'country': 'Рохан'}),  # не обязательно подавать все значения, они будут
↪пустыми
]
newtable.insert(features_to_insert)
```

Иногда может потребоваться массовая вставка записей в таблицу. В случае, если это делать по одной записи, то после каждой вставки будут отрабатываться события на изменение данных, которые в свою очередь используются в инструментах и прочих GUI компонентах. Для того, чтобы это избежать предлагается два метода. Рассмотрим их на примере вставки 1000 записей в таблицу:

1. С предварительных занесением в список `list` и последующей единовременной вставкой:

```
table = provider_manager.openfile('sample.tab')
point = Point(1,1)
pstyle = PointStyle()
features = []
for i in range (1, 1000):
  fpoint = Feature({}, geometry = point, style = pstyle )
  features.append(fpoint)
table.insert(features)
table.commit()
```

Но при использовании данного метода при большом количестве данных есть вероятность перерасхода памяти. Этого можно избежать, если воспользоваться вторым подходом.

2. Использование функции-генератора. При этом читаемые данные сразу же используются при вставке. Этот метод можно использовать, как пример, если необходимо зачитать данные из текстового файла с неподдерживаемым форматом.

```
table = provider_manager.openfile('sample.tab')
point = Point(1,1)
pstyle = PointStyle()

def generate_features(): # функция-генератор
    for i in range(1, 1000):
        yield Feature(geometry = point, style = pstyle)

table.insert(generate_features())
table.commit()
```

Или же это можно записать в другом виде (результат аналогичный):

```
table = provider_manager.openfile('sample.tab')
point = Point(1,1)
pstyle = PointStyle()
generator = ( Feature(geometry = point, style = pstyle) for i in range(1, 1000) )
table.insert(generator)
table.commit()
```

5.1.4 Редактирование таблиц

Один из возможных способов редактирования таблиц - открыть исходную таблицу, создать целевую таблицу с той же или другой структурой. И при чтении записей из исходной таблицы редактировать их и записывать в целевую. В результате получится отредактированная копия.

Например, для таблицы `world` необходимо оставить только колонки “Страна” и “Население”; и оставить только те страны, население которых больше `100 миллионов`.

Вот один из вариантов, как это можно реализовать.

```
def is_over_100million(feature) -> bool:
    return feature['Население'] > 100_000_000


orig_table = provider_manager.openfile('../path/to/datadir/world.tab')
schema = Schema(
    Attribute.string('Страна'),
    Attribute.integer('Население'),
)
copy_table = provider_manager.createfile('../path/to/edited_world.tab', schema)
orig_features = orig_table.items()

filtered_features = filter(is_over_100million, orig_features)
copy_table.insert(filtered_features)
```

При редактировании таблицы так-же присутствует возможность более гибкого управления производимыми в таблице изменениями. Допустимо выполнение отката как назад (отмена последних изменений) `undo()`, так и откат вперед (возврат после выполнения отката назад) `redo()`.

```
table.insert(feature1)
table.insert(feature2)
if table.can_undo:
   table.undo()
```

В рассмотренном примере будет вставлена только feature1.

5.1.5 Запросы

SQL-запросы являются мощным инструментом обработки данных. При выполнении запроса к таблицам образуется отдельный объект данных. При открытии данных они попадают в единый каталог `DataManager`. Запрос выполняется относительно всех таблиц, находящихся в этом каталоге, с помощью метода `query()`.

```
query_text = 'SELECT * FROM world WHERE Население > 100000000'
query_table = provider_manager.query(query_text)
```

Список поддерживаемых функций можно найти в руководстве пользователя ГИС Аксиома и в диалоге построения SQL-запросов самого приложения ГИС Аксиома.

Интерфейс Qt

Более низкоуровневый доступ к данным возможен через стандартный Qt интерфейс PySide2.QtSql и вспомогательный sql. Это позволяет:

- избежать лишних округлений и нормализации значений, так как нет схемы таблицы Schema и атрибутов Attribute;
- не конвертировать значения, так как нет приведения к Feature;
- выделять меньше ресурсов, так как результат читается по одной записи без создания объекта данных Table;
- проще интегрировать с другим Qt кодом.

Примечание: Доступ к колонкам с геометрией и стилем осуществляется через значения geometry_uid и style_uid.

Все открытые таблицы образуют базу данных, которую можно получить функцией get_database().

```
db = axipy.sql.get_database()
query = QSqlQuery(db)
query.exec_('SELECT * FROM world WHERE Население > 100000000')
while query.next():
    print(query.value(0))
```

Обработка ошибок

При выполнении запроса возвращается признак успешности выполнения. Если выполнение оказалось неуспешным, функция PySide2.QtSql.QSqlQuery.lastError() может показать последнюю ошибку. Стандартная обработка ошибок в PySide2.QtSql выглядит следующим образом:

```
q = QSqlQuery(database)
success = q.exec_(sql_text)
if not success:
    # Передаем ошибку выше
    raise RuntimeError(q.lastError().text())
```

5.2 Растры

Растровое изображение – это цифровое представление рисунка, фотографии или иного графического материала в виде набора точек растра.

Класс Raster представляет растровый объект данных.

Примечание: Для выполнения примеров:

```
from axipy import provider_manager, Layer, Map, view_manager, Raster, raster
```

5.2.1 Открытие растров, расположенных в файловой системе

Для открытия растровых файлов используйте функцию `openfile()` объекта `provider_manager`.

```
raster = provider_manager.openfile('path/to/raster.tab')
```

5.2.2 Открытие из СУБД

Так-же поддерживается открытие данных из СУБД PostgreSQL и Oracle. Данный функционал реализован через передачу строки в формате библиотеки GDAL. Пример открытия растра из БД PostgreSQL и показа его на карте:

```
raster = provider_manager.gdal.open("PG:host=server_name dbname='test' user='postgres
↪' password='postgres' schema='public' table=table_name")
raster_layer = Layer.create(raster)
print(raster_layer)
map = Map([ raster_layer ])
mapview = view_manager.create_mapview(map)
```

Пример открытия растра из БД Oracle:

```
raster = provider_manager.gdal.open("GeoRaster:user/password@XE,GDAL_RDT,1")
```

где user/password@XE - строка соединения с БД, GDAL_RDT,1 - источник, который необходимо открыть. Подробнее см gdalinfo.

5.2.3 Открытие данных, расположенных на WEB-ресурсах

Пример открытия тайлового сервера TMS. При этом передается шаблон URL:

```
raster = provider_manager.tms.open('https://maps.axioma-gis.ru/osm/{LEVEL}/{ROW}/{COL}
↪.png')
```

Пример открытия WMTS:

```
raster = provider_manager.wmts.open('https://basemap.at/wmts/1.0.0/WMTSCapabilities.
↪xml', 'geolandbasemap')
```

5.2.4 Регистрация растра

Растры по определению имеют пространственную привязку - координатную систему `coordsystem` и точки привязки `get_gcps()`. Таким образом:

```
Изображение + Пространственная привязка = Растр
```

Для того, чтобы «привязать» растр (задать изображению пространственную привязку), можно воспользоваться функцией `register()` модуля `raster`.

Например так можно привязать растр, используя эквивалентную матрицу преобразования (`QTransform` по умолчанию). Т.е. точка на изображении (x, y) в пикселях будет привязана к точке в пространстве (x, y) в единицах Системы Координат.

```
from PySide2.QtGui import QTransform

matrix = QTransform()
coordsystem = CoordSystem.from_units(Unit.m)
register(imagefile, matrix, coordsystem)
```

Чтобы привязать растр по-другому, можно передать другую матрицу трансформации (аффинное преобразование). Предварительно ее придется узнать или посчитать. Для расчета матрицы преобразования достаточно 3 точек привязки GCP. Обычно по углам изображения. Функция `register()` также может это сделать.

```
gcps = [
    GCP((0, 0), (0, 1000)),
    GCP((100, 0), (1000, 1000)),
    GCP((0, 100), (0, 0)),
]
coordsystem = CoordSystem.from_string('prj:NonEarth 0,7')
register(imagefile, gcps, coordsystem, True)
```

5.2.5 Трансформация растра

Операция трансформации `transform()` растрового файла требуется для устранения или компенсации искажений, возникающих при создании растра. Требуется, когда аффинного преобразования недостаточно.

Список 1: Пример использования

```
coordsystem = CoordSystem.from_epsg(4326)
gcps = [
    GCP((0, 0), (0, 0)),
    GCP((200, 0), (30, 30)),
    GCP((200, 200), (60, 0)),
]
transform(rasterfile, outputfile, gcps, coordsystem)
```

См.также:

Руководство пользователя ГИС Аксиома раздел «Растровые изображения»

6 Провайдеры данных

За каждый тип данных отвечает провайдер данных `axipy.ProviderManager`. Провайдер владеет информацией о том, как представлять конкретный тип объектов данных и как ими манипулировать. Класс `axipy.ProviderManager` содержит все зарегистрированные провайдеры данных.

Каждый провайдер имеет свои особенности и возможности, но общие концепции, такие как открытие и создание, выделены в общий интерфейс `axipy.DataProvider`. В то же время конкретный провайдер может иметь дополнительные функции и параметры, свойственные только ему.

Для получения списка загруженных провайдеров есть функция `axipy.ProviderManager.loaded_providers()`.

Например:

```
provider_manager.loaded_providers()
```

```
{'CsvDataProvider': 'Файловый провайдер: Текст с разделителями',
  'DwgDgnFileProvider': 'Провайдер DWG и DGN',
  'GdalDataProvider': 'Растровый провайдер GDAL',
  'MifMidDataProvider': 'Провайдер данных MIF-MID',
  'OgrDataProvider': 'Векторный провайдер OGR',
  'ShapeDataProvider': 'Shapefile',
  'PgDataProvider': 'PostgreSQL',
  'OracleDataProvider': 'oracle',
  'MsSqlDataProvider': 'MS SQL Server',
  'RestDataProvider': 'ArcGIS REST',
  'SqliteDataProvider': 'Векторный провайдер sqlite',
  'TabDataProvider': 'MapInfo',
  'TmsDataProvider': 'Тайловые сервисы',
  'WfsDataProvider': 'Web Feature Service',
  'WmsDataProvider': 'Web Map Service',
  'WmtsDataProvider': 'Web Map Tile Service',
  'XlsDataProvider': 'Провайдер чтения файлов Excel'}
```

Для открытия разных источников данных может потребоваться разная информация. Например, для подключения к базе данных нужно указать имя пользователя и пароль и прочее. Поэтому для большинства провайдеров данных определены функции задания источников данных `axipy.DataProvider.get_source()` с

дополнительными параметрами. Например, axipy.CsvDataProvider.get_source(), axipy.PostgreDataProvider.get_source() и другие.

Например:

```
source = provider_manager.csv.get_source('path/to/mydata.csv', delimiter=';')
table = source.open()
```

Что эквивалентно:

```
table = provider_manager.csv.open('path/to/mydata.csv', delimiter=';')
```

Или:

```
table = provider_manager.openfile('path/to/mydata.csv', delimiter=';')
```

См.также:

Подробнее в документации axipy.ProviderManager.

6.1 Открытие/Создание

Открытие и создание объектов данных axipy.DataObject выполняется провайдерами данных. Класс axipy.ProviderManager представляет коллекцию зарегистрированных провайдеров данных.

6.1.1 Открытие

Самый простой способ открыть объект данных из файла - использовать функцию axipy.ProviderManager.openfile. Функция сама найдет подходящий провайдер данных для открытия.

Совет: Это предпочтительный способ.

Список 1: Пример открытия

```
# filepath = 'path/to/file.tab'
table = provider_manager.openfile(filepath)
```

Примечание: При открытии данных они попадают в единый каталог axipy.DataManager.

При необходимости указать больше параметров для открытия, или если параметры по умолчанию не подходят, можно явно использовать провайдер данных. Необходимый провайдер данных можно получить из коллекции зарегистрированных провайдеров данных axipy.ProviderManager.

Список 2: Пример открытия конкретным провайдером

```
# csv_filepath = 'path/to/file.csv'
csv_table = provider_manager.csv.open(csv_filepath, delimiter='\t')
```

Самый сложный способ - открытие через словарь `dict`. Здесь нужно заранее знать все параметры, их допустимые значения и идентификатор провайдера данных.

Список 3: Пример открытия из словаря

```
"""
Открывает csv файл через определение ``definition``.
Исключительно в качестве примера. Открывать csv проще провайдером.
"""
# csv_filepath = 'path/to/file.csv'
definition = {
    'src': csv_filepath,
    'delimiter': '\t',
    'charset': 'utf8',
    'hasNamesRow': True,
    'provider': 'CsvDataProvider'
}
csv_table = provider_manager.open(definition)
```

При открытии таблиц из СУБД задаются как параметры подключения, так и требуемая таблица или текст запроса.

Список 4: Пример открытия данный из БД Postgres с указанием имени таблицы.

```
definition = provider_manager.postgre.get_source(
    host='192.168.0.2',
    db_name='test',
    user='test',
    password='pass',
    dataobject='world'
)
table = provider_manager.open(definition)
```

Список 5: Пример открытия данный из БД oracle с указанием текста SQL запроса.

```
definition = provider_manager.oracle.get_source(
    host='192.168.0.2',
    db_name='ORCL',
    user='test',
    password='pass',
    sql='select * from world'
)
table = provider_manager.open(definition)
```

Если источник содержит в себе несколько объектов данных, то этот список можно запрости посредством метода `axipy.ProviderManager.read_contents()`:

```
# Запросим перечень таблиц
definition = {'src':'sample.gpkg'}
```

(continues on next page)

(продолжение с предыдущей страницы)

```
print(provider_manager.read_contents(definition))
>>> ['table1', 'table2', 'table3']
```

Далее подставляем имя нужной таблицы и открываем объект данных.

```
# Откроем нужную таблицу
definition = {'src':'sample.gpkg', 'dataobject':'table1'}
table = provider_manager.open(definition)
```

6.1.2 Создание

Аналогично, самый простой способ создания файлов - с помощью функции `axipy.ProviderManager.createfile`:

Список 6: Пример создания

```
# filepath = 'path/to/file.tab'
schema = Schema(
    Attribute.string('country', 30),
    Attribute.string('capital', 30),
    coordsystem='prj:Earth Projection 1, 104'
)
table = provider_manager.createfile(filepath, schema)
```

Примечание: При создании объекта данных он автоматически открывается.

Для задания дополнительных параметров или использования специализированных возможностей провайдера данных, можно явно вызывать методы конкретного провайдера.

Список 7: Пример создания конкретным провайдером

```
schema = Schema(
    Attribute.string('country', 30),
    Attribute.string('capital', 30),
)
temp_table = provider_manager.shp.open_temporary(schema)
```

Если и этот способ не подходит для решения какой-то задачи, можно создавать объекты данных из словаря. Это самый сложный и непрактичный способ.

6.2 Импорт/Экспорт

6.2.1 Экспорт

Некоторые форматы данных поддерживаются ГИС Аксиома только на импорт и/или экспорт. Не для всех форматом можно создать, открывать и редактировать данные, используя транзакционную модель редактирования, являющуюся основной для ГИС Аксиома. Тем не менее экспорт и создание очень близки по назначению, поэтому они объединены и представлены одним типом `axipy.Destination` - назначение объекта данных.

Так для некоторых типов экспорт `axipy.Destination.export()` является единственной возможностью вывода, в то время как для других - это дополнительная возможность к имеющейся `axipy.Destination.create_open()` в случаях, когда открытие и редактирование не требуется.

Экспортировать можно отдельные записи, таблицы или целые источники данных.

Список 8: Пример экспорта таблицы

```
# Открываем исходную таблицу
table_src = provider_manager.openfile(input_filepath)
# Формируем целевую и производим экспорт
destination = provider_manager.csv.get_destination(output_filepath, Schema())
destination.export_from_table(table_src, copy_schema=True)
```

Если требуется создать таблицу и занести в нее некоторую информацию, но при этом формат данных не поддерживается на изменение, эту проблему можно решить, используя временную таблицу как буфер. Т.е. формируем данные в памяти, а потом производим экспорт этой временной таблицы в нужный нам формат.

Список 9: Пример формирования файла csv на базе временной таблицы

```
# Создаем временную таблицу
definition = {
    'src': '',
    'schema': attr.schema(
        attr.integer('id'),
        attr.string('name')
    )
}
```

(continues on next page)

(продолжение с предыдущей страницы)

```
table_src = provider_manager.create(definition)
# Добавляем в нее записи
table_src.insert(Feature({'id': '1', 'name': 'Name1'}))
table_src.insert(Feature({'id': '2', 'name': 'Name2'}))
# Создаем целевую таблицу и производим экспорт
destination = provider_manager.csv.get_destination('outfile.csv', Schema())
destination.export_from_table(table_src, copy_schema=True)
```

Рассмотрим экспорт таблицы в базу данных на примере.

```
# откроем исходный файл
src = provider_manager.openfile('points.tab')
# Обозначим дополнительные параметры при экспорте. В данном случае это явное␣
↪пересоздание таблицы
# и сохранение результатов в лог файле. Данный параметр можно опустить.
exportParams = ExportParameters()
exportParams.logFile = 'export.log'
exportParams.dropTable = True
# Наименование таблицы в БД
newTableName = 'points_db'
# Определяем источник куда будем производить экспорт
dest = provider_manager.postgre.get_destination(host='db_server_addr',
                                                db_name='test',
                                                dataobject=newTableName,
                                                user='user',
                                                password='pass',
                                                export_params = exportParams,
                                                schema = src.schema)
# Непосредственно производим экспорт
dest.export_from_table(src)
```

Если требуется экспортировать данные из таких источников, как ГИС Панорама или данных AutoCAD, то это можно сделать двумя путяим:

- Вся информация по всем слоям отправляется в один файл
- Производится разбивка по слоям, или же выборочныая выборка нужных слоев.

Рассмотрим первый вариант для фалй AutoCAD.

```
# Откроем исходный файл
definition = {'src':'input_file.dwg'}
data_object = provider_manager.open(definition)
out_folder = '/mnt/hdd/export_folder'
file_name = os.path.join(out_folder, 'output_file.tab')
destination = provider_manager.tab.get_destination(file_name, data_object.schema)
destination.export(data_object.items())
```

По аналогии для данных ГИС Панорама.

```
# Откроем исходный файл
data_object = provider_manager.openfile('Podolsk.map')
out_folder = '/mnt/hdd/export_folder'
file_name = os.path.join(out_folder, 'output_file.tab')
destination = provider_manager.tab.get_destination(file_name, data_object.schema)
destination.export(data_object.items())
```

Рассмотрим второй вариант. Все слои экспортируем как отдельные файлы.

```
# Откроем исходный файл
definition = {'src':'input_file.dwg'}
data_object = provider_manager.open(definition)
out_folder = '/mnt/hdd/export_folder'
# Запросим список слоев и каждый слой экспортируем в отдельный файл
for layer_name in data_object.layers:
    file_name = os.path.join(out_folder, '.'.join([layer_name, 'tab']))
    destination = provider_manager.tab.get_destination(file_name, data_object.schema)
    destination.export(data_object.items(layer_name))
```

6.2.2 Импорт

Источник данных `axipy.Source` - это зеркальный тип назначения объекта данных `axipy.Destination`. Так же как для назначения, некоторые типы данных поддерживают только импорт, и не могут быть напрямую открыты с помощью `axipy.Source.open`. А другие типы поддерживают и открытие и импорт для случаев, когда открытие и редактирование не требуется.

Список 10: Пример экспорта источника

```
source = provider_manager.tab.get_source(input_tab_file)
destination.export_from(source)
```

См.также:

Чтобы узнать, какие типы поддерживают импорт и экспорт, обратитесь к описанию конкретных провайдеров данных `axipy.ProviderManager`.

Изменение структуры таблицы

Если необходимо изменить структуру таблицы, в частности добавить новое поле, удалить существующее или же его расширить. Эту процедуру можно выполнить через экспорт. Рассмотрим на примере.

Откроем исходную таблицу

```
# открываем исходный файл
src = provider_manager.openfile(src_filepath)
```

Далее, возьмем схему исходной таблицы, сделаем ее копию и произведем все необходимые с ней изменения. Т.е. добавим новое поле в конкретную позицию, удалим ненужное и расширим длину существующего.

```
# Целевая схема как копия исходной
dest_schema = src.schema
# вставка нового поля
dest_schema.insert(2, Attribute.string('Новое поле', 30))
# Удаление поля по имени
attr_to_del = dest_schema.by_name('Capital')
dest_schema.remove(attr_to_del)
# Увеличение длины поля
idx_edit = dest_schema.index_by_name('Страна')
dest_schema[idx_edit] = Attribute.string('Страна', 100)
```

Следующим шагом создаем выходную таблицу на базе полученной схемы и производим непосредственно экспорт.

```
# Создаем таблицу
dest = provider_manager.tab.get_destination(dest_filepath, dest_schema)
# Экспортируем данные
dest.export_from_table(src)
```

Если необходимо создать таблицу в памяти на базе измененной структуры, то эту задачу можно выполнить следующим образом:

```
# Создаем таблицу в памяти на базе измененной схемы
memory_table = provider_manager.create({'schema': dest_schema})
# Производим вставку данных
memory_table.insert(src.items())
```

7 Записи

Запись `Feature`, которая получается при чтении из таблицы, во многом повторяет словарь Python `dict`. В ней содержатся пары (Имя столбца: Значение). Причем значение приводится к типу столбца. Например, если это числовое поле int, а его значение 42, то запись будет содержать число 42, а не строку “42”.

Прочитанная запись никак не ссылается на таблицу и является копией. Присвоенная к переменной запись будет доступна после продвижения итератора или даже после закрытия таблицы. Но поэтому и изменения, внесенные в эту запись-копию, никак не повлияют на значения в таблице.

7.1 Атрибуты

Доступ к атрибутам осуществляется по имени или номеру. Так же как для словаря, если атрибут с заданным именем не существует, то вызывается исключение `KeyError`. Если атрибут с заданным номером не существует, то вызывается исключение `IndexError`.

```
feature = next(table.items())
try:
    value = feature['attr_name']
except KeyError as e:
    print(f'Поймано исключение: {e}')
```

```
>>> Поймано исключение: "Key 'unknown_key' not found"
```

Можно задать значение по умолчанию при чтении атрибута, который может не существовать. Если его не задать, то значение по умолчанию считается равным `None`.

```
value = feature.get('attr_name', 0.0)
```

Проверка существования атрибута с помощью ключевого слова `in` так же, как в словаре:

```
if 'attr_name' in feature:
    ...
```

7.2 Геометрический атрибут

Доступ к специальному атрибуту Геометрия axipy.Geometry производится через свойство axipy.Feature.geometry.

Или можно использовать специальное наименование GEOMETRY_ATTR, представляющее имя геометрического атрибута:

```
geometry = feature.geometry
# эквивалентно
geom = feature[GEOMETRY_ATTR]
```

Проверка существования has_geometry():

```
if feature.has_geometry():
    ...
# эквивалентно
if GEOMETRY_ATTR in feature:
    ...
```

7.3 Стиль для геометрического атрибута

Стиль axipy.Style может содержаться в виде атрибута. Доступ к нему производится по специальному наименованию `STYLE_ATTR или свойствам axipy.Feature.style и has_style().

```
style = feature.style
# эквивалентно
style = feature[STYLE_ATTR]
```

```
feature.has_style()
# эквивалентно
STYLE_ATTR in feature
```

7.4 Идентификаторы записей

Записи имеют свойство axipy.Feature.id. Это значение зависит от типа данных. При этом не гарантируется порядок, начальное значение или отсутствие разрывов между соседними записями. Также идентификатор необязательно является числом.

8 Геометрия

Геометрический объект (геометрия) представляет собой описательную структуру данных, на базе которой формируется графическое представление векторного элемента. Оно может быть частью визуального представления объектов реального мира. В зависимости от целей, геометрия может содержать данные о системе координат, в которой она создана.

8.1 Типы

ГИС Аксиома поддерживает следующие типы геометрических объектов:

Простые типы геометрии:

- Точка
- Полилиния
- Полигон

Коллекции:

- Смешанная коллекция
- Коллекция точек
- Коллекция полилиний
- Коллекция полигонов

Так же возможна работа с типами данных MapInfo:

- Линия
- Прямоугольник
- Скругленный прямоугольник
- Эллипс
- Дуга
- Текст

Рассмотрим подробнее геометрические типы. Переменные из примеров создания объектов будут использованы ниже в примерах более общего характера. Более общие характеристики объектов, а так же операции над ними будут рассмотрены ниже.

8.1.1 Точка

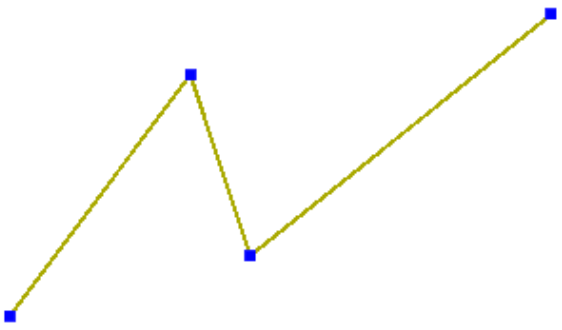
Точка представлена классом axipy.Point. Для нее характерно отсутствие таких параметров, как площади и линейных размеров. Создадим точку с координатами (10, 10) в СК Широта/Долгота. С целью визуального восприятия, результат представим в виде WKT строки (axipy.Geometry.to_wkt()).

```
cs = CoordSystem.from_prj("1, 104")
point = Point(10, 10, cs)
print('Точка ({}, {})'.format(point.x, point.y))
'''
>>> Точка (10, 10)
'''
```

8.1.2 Полилиния

Представлена классом axipy.LineString в виде непрерывной линии, соединяющей последовательность узлов. Для нее так же характерно отсутствие площади, но присутствует длина. Точки полилинии хранятся в виде списка list [axipy.utl.Pnt], то при задании, помимо передачи в конструктор такого списка, так же допустимо указание координат в виде пар координат tuple. Нумерация точек при доступе по индексу начинается с 0. Доступны через свойство axipy.LineString.points Приведем пример: создадим полилинию без СК. В этом же примере заменим 3-ю вершину на другое значение и выведем полученный результат в виде wkt axipy.Geometry.to_wkt():

```
ls = LineString([(1, 1), (4, 5), (5, 2), (10, 6)])
ls.points[2] = (6, 3)
print(ls.to_wkt())
'''
>>> LINESTRING (1 1, 4 5, 6 3, 10 6)
'''
```



Так же присутствует возможность изменения существующих параметров полилинии. Добавим точку в позицию 1 и удалим точку 2:

```
ls.points.insert(1, (3, 6))
ls.points.remove(2)
print(ls.to_wkt())
'''
>>> LINESTRING (1 1, 3 6, 6 3, 10 6)
'''
```

Поддерживается инициализация через существующий итератор. Смоделируем данную ситуацию: создадим итератор на базе точек первой полилинии и на его основе создадим полилинию:

```
itr = (a for a in ls.points)
ls_it = LineString(itr)
```

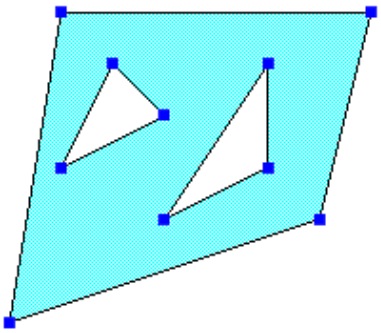
8.1.3 Полигон

Представлен классом `axipy.Polygon`. Полигон представляет собой площадной объект, или другими словами часть плоскости, ограниченная замкнутой полилинией. Кроме внешней границы, полигон может иметь одну или несколько внутренних (дырок). Ему присущи такие свойства, как длина (периметр) и площадь. Точки хранятся по аналогии с полилинией. И инициализация в конструкторе или при добавлении дырки в полигон производится по подобному принципу. Характерной особенностью является тот факт, что все контуры замкнуты и последняя точка совпадает с первой. Это касается как формы полигона, так и его дырок. В качестве примера создадим полигон:

```
polygon = Polygon((1, 1), (2, 7), (8, 7), (7, 3))
print(polygon.to_wkt())
'''
>>> POLYGON ((1 1, 2 7, 8 7, 7 3, 1 1))
'''
```

И добавим в него две дырки:

```
polygon.holes.append((2, 4), (3, 6), (4, 5))
polygon.holes.append((4, 3), (6, 6), (6, 4))
print(polygon.to_wkt())
'''
>>> POLYGON ((1 1, 2 7, 8 7, 7 3, 1 1), (2 4, 3 6, 4 5, 2 4), (4 3, 6 6, 6 4, 4 3))
'''
```



Как уже указывалось выше, работа с точками для полигона производится аналогично

с полилинией. Это же касается и дырок, только доступ производится через свойство axipy.Polygon.holes Обновим значение третьей точки для второй дырки.

```
polygon.holes[1][2] = (6, 3)
print(polygon.to_wkt())
'''
>>> POLYGON ((1 1, 2 7, 8 7, 7 3, 1 1), (2 4, 3 6, 4 5, 2 4), (4 3, 6 6, 6 3, 4 3))
'''
```

8.1.4 Смешанная коллекция

Представлена классом axipy.GeometryCollection. Это нетипизированная коллекция. Может содержать внутри себя геометрии различных типов за исключением коллекций. Попробуем создать коллекцию и добавить в нее последовательно точку, полилинию и полигон. Стоит обратить внимание, что если добавляется последовательность точек, то она рассматривается как полилиния, в тоже время для указания полигона необходимо явно указать принадлежность к классу. Доступ к элементам коллекции производится по индексу, начиная со значения 0.

```
coll = GeometryCollection()
coll.append(1, 2)  # Точка
coll.append((3, 4), (5, 5), (10, 0))  # Полилиния
coll.append(Polygon((3, 4), (5, 5), (10, 0)))  # Полигон
print(coll.to_wkt())
'''
>>> GEOMETRYCOLLECTION (POINT (1 2), LINESTRING (3 4, 5 5, 10 0), POLYGON ((3 4, 5 5,␣
↪10 0, 3 4)))
'''
```

Удалим из коллекции полилинию и полигон, а после этого добавим объекты, созданные в предыдущих примерах. Точку поменяем простой заменой по индексу:

```
coll.remove(2)
coll.remove(1)
coll.append(polygon)
coll.append(ls)
coll[0] = point
print(coll.to_wkt())
'''
>>> GEOMETRYCOLLECTION (POINT (10 10), POLYGON ((1 1, 2 7, 8 7, 7 3, 1 1), (2 4, 3 6,␣
↪4 5, 2 4), (4 3, 6 6, 6 3, 4 3)), LINESTRING (1 1, 3 6, 6 3, 10 6))
'''
```

При замене элемента с заданием координат работают такие же принципы, как и в конструкторе. Обновим полилинию и полигон:

```
coll[2] = [(101, 102), (103, 104), (105, 106)]
coll[1] = Polygon((101, 102), (103, 104), (105, 106))
print(coll.to_wkt())
'''
>>> GEOMETRYCOLLECTION (POINT (10 10), POLYGON ((101 102, 103 104, 105 106, 101 102)),
↪ LINESTRING (101 102, 103 104, 105 106))
'''
```

Поменяем первую (она же последняя) точку полигона:

```
coll[1].points[0] = (0, 0)
print(coll.to_wkt())
'''
>>> GEOMETRYCOLLECTION (POINT (10 10), POLYGON ((0 0, 103 104, 105 106, 0 0)),␣
↪LINESTRING (101 102, 103 104, 105 106))
'''
```

Далее рассмотрим типизированные коллекции. Принципы работы аналогичны нетипизированным, за исключением того, что позволено хранение геометрий только одного типа.

8.1.5 Коллекция точек

Представлена классом `axipy.MultiPoint`. Это типизированная коллекция. Допустимо хранение только точек. Создадим коллекцию точек и добавим в нее 2 элемента разными способами:

```
mpoint = MultiPoint()
mpoint.append(11, 11)
mpoint.append((12, 12))
mpoint.append(point)
print(mpoint.to_wkt())
'''
>>> MULTIPOINT (11 11, 12 12, 10 10)
'''
```

8.1.6 Коллекция полилиний

Представлена классом `axipy.MultiLineString`. Это типизированная коллекция. Допустимо хранение только полилиний.

```
mls = MultiLineString()  # Создадим саму коллекцию.
mls.append((11, 12), (13, 14), (15, 16))  # Добавим как объект
mls.append([(21, 22), (23, 24), (25, 26)])  # Добавим как перечень точек
print(mls.to_wkt())
'''
>>> MULTILINESTRING ((11 12, 13 14, 15 16), (21 22, 23 24, 25 26))
'''
```

8.1.7 Коллекция полигонов

Представлена классом `axipy.MultiPolygon`. Это типизированная коллекция. Допустимо хранение только полигонов. Отдельно стоит отметить, что при добавлении/изменения геометрий в виде перечня точек, в отличие от смешанной коллекции и коллекции полилиний, в данном случае создается полигон. Рассмотрим на примере:

```
mpoly = MultiPolygon()
mpoly.append((1, 2), (3, 4), (5, 6), (7, 8))
mpoly.append(polygon)  # Добавим ранее созданный с дыркой
print(mpoly.to_wkt())
''''
```

(continues on next page)

(продолжение с предыдущей страницы)

```
>>> MULTIPOLYGON (((1 2, 3 4, 5 6, 7 8, 1 2)), ((0 0, 1 10, 14 15, 11 5, 10 2, 0 0),␣
↪(2 2, 44 44, 5 3, 2 2), (2 2, 2 4, 5 3, 2 2)))
'''
```

Далее рассмотрим объекты MapInfo. Одна из особенностей заключается в том, что они нестандартные, и, как следствие, не поддерживают WKT представление.

8.1.8 Линия

Представлена классом `axipy.Line`. Линейный объект. Описывается двумя точками: начальным и конечным узлом. Создадим линию, передав в конструктор пару точек. После этого последовательно поменяем координаты начала и конца линии.

```
line = Line((11, 11), (21, 21))
line.begin = (12, 12)
line.end = (120, 120)
```

8.1.9 Прямоугольник

Представлен классом `axipy.mi.Rectangle`. Площадной объект. Описывается минимальными и максимальными значениями по координатам X и Y. Создадим прямоугольник, задав параметры через конструктор. Затем поменяем предельные значения по X координате. Результат проконтролируем выводом значения `axipy.Geometry.bounds` геометрии.

```
rectangle = Rectangle(0, 0, 40, 20)
rectangle.xmin = 10
rectangle.xmax = 50
print(rectangle.bounds)
'''
>>> (10.0 0.0) (50.0 20.0)
'''
```

8.1.10 Скругленный прямоугольник

Представлен классом `axipy.mi.RoundRectangle`. Так же является площадным объектом. Отличительной особенностью от прямоугольника является наличие скруглений на углах. В остальном данные объекты аналогичны. Во избежании путаницы с параметрами, в конструкторе задается `axipy.utl.Rect` и радиусы скругления:

```
rrectangle = RoundRectangle([0, 0, 40, 20], 0.2, 0.2)
rrectangle.xRadius = 0.3
print(repr(rrectangle))
'''
>>> RoundRectangle xmin=0.0 ymin=0.0 xmax=40.0 ymax=20.0 xraduis=0.3 yraduis=0.2
'''
```

8.1.11 Эллипс

Представлен классом `axipy.mi.Ellipse`. Является площадным объектом. Создадим эллипс, передав в конструктор его Rect. После этого поменяем свойства.

```
ellipse = Ellipse([0, 0, 22, 33])
ellipse.center = (10, 10)  # Переопределим центр
ellipse.majorSemiAxis = 10  # Задание большой полуоси
ellipse.minorSemiAxis = 5  # Задание малой полуоси
print(ellipse.bounds)
'''
>>> (0.0 5.0) (20.0 15.0)
'''
```

8.1.12 Дуга

Представлена классом `axipy.mi.Arc`. Линейный объект. Задается в конструкторе через `axipy.utl.Rect` и, дополнительно, начальный и конечный угол дуги. Создадим объект, затем выведем основные свойства:

```
arc = Arc(Rect(0, 0, 20, 30), 45, 270)
print('center={} start={} end={}'.format(arc.center, arc.startAngle, arc.endAngle))
'''
>>> center=(10.0 15.0) start=45.0 end=270.0
'''
```

8.1.13 Текст

Представлен классом `axipy.mi.Text`. В отличие от описанных выше типов, его геометрические свойства определяются не только параметрами класса, но и параметрами его оформления.

```
rv = view_manager.create_reportview()
rect = Rect(8, 6, 11, 7)
text = Text("Пример", rect, view=rv)
geomItem = GeometryReportItem()
geomItem.geometry = text
geomItem.style = Style.for_geometry(text)
rv.report.items.add(geomItem)
```

Рассмотрим общие свойства геометрии.

8.2 Свойства геометрии

В зависимости от типа объекта, для него могут быть получены свойства. Продемонстрируем использование некоторых из них:

```
polygon = Polygon((0, 0), (0, 10), (10, 10), (10, 0))
print(f'Название: {polygon.name}')
print(f'Площадь: {polygon.get_area()}')
print(f'Периметр: {polygon.get_length()}')
print(f'Ограничивающий прямоугольник: {polygon.bounds}')
'''
>>> Название: Полигон
>>> Площадь: 100.0
>>> Периметр: 40.0
>>> Ограничивающий прямоугольник: (0.0 0.0) (10.0 10.0)
'''
```

8.3 Сериализация

ГИС Аксиома позволяет производить сериализацию геометрию в форматы текстового WKT или бинарного вида WKB. Подробнее

Рассмотрим на примере точечного объекта сначала для случая WKT. Будем использовать метод `axipy.Geometry.from_wkt()` для получения объекта из WKT и метод `axipy.Geometry.to_wkt()` для получения WKT представления полученного объекта:

```
polygon = Geometry.from_wkt('POLYGON ((10 10, 100 100, 100 10, 10 10))')
point = Geometry.from_wkt('SRID=4326;POINT (10 10)')
crs = CoordSystem.from_epsg(4326)
pline = Geometry.from_wkt('LINESTRING (30 10, 10 30, 40 40)', crs)
```

Аналогично проделаем для WKB. При этом используются, соответственно, `axipy.Geometry.from_wkb()` и `axipy.Geometry.to_wkb()`:

```
wkb = b'\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00$@\x00\x00\x00\x00\x00\x00$@'
pnt = Geometry.from_wkb(wkb)
```

8.4 Преобразования

Для перепроецирование в другую СК используется метод `axipy.Geometry.reproject()`. Заметим, что исходный объект должен содержать свою СК. В рамках примера перепроецируем точку из СК Широта/Долгота в проекцию Меркатора:

```
csLL = CoordSystem.from_prj("1, 104")
csMercator = CoordSystem.from_prj("10, 104, 7, 0")
point_ll = Point(10, 10, csLL)
point_merc = point_ll.reproject(csMercator)
print('point result: {}'.format(point_merc.to_wkt()))
'''
```

(continues on next page)

(продолжение с предыдущей страницы)

```
>>> point result: POINT (1113194.90793274 1111475.10285222)
'''
```

Более простые преобразования типа сдвига axipy.Geometry.shift(), масштабирования axipy.Geometry.scale() и поворот axipy.Geometry.scale() игнорируют указанную СК и данные операции производятся в текущих координатах объекта. Рассмотрим на примерах:

```
polygon = Polygon((1, 1), (2, 7), (8, 7), (7, 3))
polygon_shift = polygon.shift(5, 5)
# Сдвинем на величину 5 по X и Y
print('polygon shift: {}'.format(polygon_shift.to_wkt()))
polygon_scale = polygon.scale(5, 5)
# Отмасштабируем координаты относительно центра
print('polygon scale: {}'.format(polygon_scale.to_wkt()))
# Повернем на 90 градусов против часовой стрелки относительно точки (10, 40)
polygon_rotated = polygon.rotate((10, 40), 90)
print('polygon rotate: {}'.format(polygon_rotated.to_wkt()))
'''
>>> polygon shift: POLYGON ((6 6, 7 12, 13 12, 12 8, 6 6))
>>> polygon scale: POLYGON ((-13 -11, -8 19, 22 19, 17 -1, -13 -11))
>>> polygon rotate: POLYGON ((49 31, 43 32, 43 38, 47 37, 49 31))
'''
```

Для более сложных аффинных преобразований можно использовать axipy.Geometry.affine_transform() с указанием матрицы преобразований QTransform

8.5 Пространственные операции

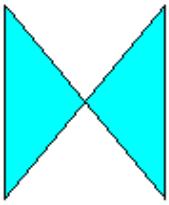
8.5.1 Нормализация объекта

Для проверки валидности геометрии предусмотрено свойство axipy.Geometry.is_valid. Если геометрия не проходит тест на валидность (данное свойство False), то для его нормализации используется метод axipy.Geometry.normalize(). Краткую аннотацию причины почему геометрия недействительна, можно воспользовавшись свойством axipy.Geometry.is_valid_reason.

Создадим заведомо неправильный полигон и попробуем его исправить:

```
poly_bad = Polygon([(1, 1), (6, 7), (6, 1), (1, 7)])
poly_norm = poly_bad.normalize()
print('Validate source: {} ({})'.format(poly_bad.is_valid, poly_bad.is_valid_reason))
print('Validate destination: {}'.format(poly_norm.is_valid))
print('Wkt:{}'.format(poly_norm.to_wkt()))
'''
>>> Validate source: False (Self-intersection[3.5 4])
>>> Validate destination: True
>>> Wkt:MULTIPOLYGON (((3.5 4, 1 1, 1 7, 3.5 4)), ((3.5 4, 6 7, 6 1, 3.5 4)))
'''
```

В результате мы получили коллекцию из двух полигонов.



8.5.2 Клонирование объекта

Для создания копии объекта используется метод axipy.Geometry.clone():

```
point1 = Point(10, 10)
point2 = point1.clone()
point1.x = 12
print('>>>', point1.to_wkt(), point2.to_wkt())
'''
>>> POINT (12 10) POINT (10 10)
'''
```

8.5.3 Логические операции

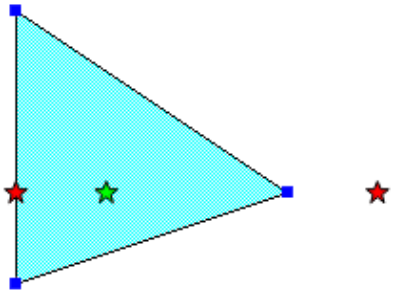
Пространственные отношения, возвращающие логический True или False:

Точное совпадение геометрий производится посредством axipy.Geometry.equals(), если же необходимо произвести приблизительное сравнение, то используем метод axipy.Geometry.almost_equals():

```
polygon1 = Polygon((1, 1), (2, 7), (7, 3))
polygon2 = Polygon((1, 1.1), (2, 7), (7, 3))
print('Точное сравнение:', polygon1.equals(polygon2))
print('Сравнение с точностью 0.2:', polygon1.almost_equals(polygon2, 0.2))
'''
>>> Точное сравнение: False
>>> Сравнение с точностью 0.2: True
'''
```

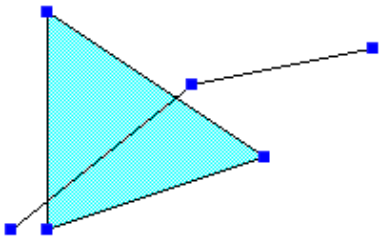
Проверка на попадание axipy.Geometry.contains()

```
poly1 = Polygon((1, 1), (1, 7), (7, 3))
point1 = Point(3, 3)
point2 = Point(9, 3)
point3 = Point(1, 3)
print('Точка внутри:', poly1.contains(point1))
print('Точка снаружи:', poly1.contains(point2))
print('Точка на грани:', poly1.contains(point3))
'''
>>> Точка внутри: True
>>> Точка снаружи: False
>>> Точка на грани: False
'''
```



Проверка на частичное пересечение axipy.Geometry.crosses()

```
poly1 = Polygon((1, 1), (1, 7), (7, 3))
sl = LineString((0, 1), (5, 5), (10, 6))
print(poly1.crosses(sl))
'''
>>> True
'''
```



Проверка на отсутствие соприкосновений axipy.Geometry.disjoint()

```
print(poly1.disjoint(sl))
'''
>>> False
'''
```

Проверка пересечений объектов axipy.Geometry.intersects()

Пересечение геометрий, если результат отличен от анализируемых данных axipy.Geometry.overlaps()

```
poly2 = Polygon((5, 1), (4, 4), (10, 3))
print(poly1.overlaps(poly2))
'''
>>> True
'''
```

Проверка касания axipy.Geometry.touches()

```
point3 = Point(1, 5)
print('Точка на грани:', poly1.touches(point3))
'''
>>> True
'''
```

Функция, обратная contains axipy.Geometry.within()

```
print('Точка внутри:', Point(2, 5).within(poly1))
''''
>>> True
'''
```

Охват одной геометрии другой axipy.Geometry.covers()

8.5.4 Отношения DE-9IM

Функция axipy.Geometry.relate() проверяет все DE-9IM отношения между объектами. Предикаты выше являются их частными случаями.

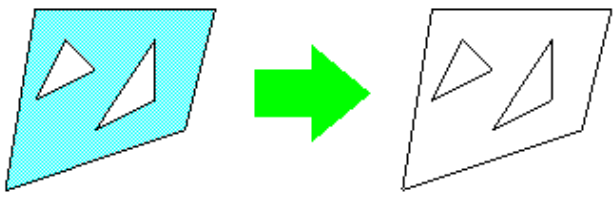
```
print('relate:', poly1.relate(Point(10, 10)))
'''
relate: FF2FF10F2
'''
```

8.5.5 Операции над объектами

В данном разделе рассмотрим операции, результатом выполнения которых будут новые объекты.

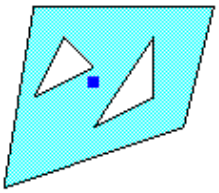
Получение границ геометрии в виде полилинии axipy.Geometry.boundary()

```
poly = Geometry.from_wkt('POLYGON ((1 1, 2 7, 8 7, 7 3, 1 1), (2 4, 3 6, 4 5, 2 4),␣
↪(4 3, 6 6, 6 4, 4 3))')
poly_boundary = poly.boundary()
print(poly_boundary.to_wkt())
'''
>>> MULTILINESTRING ((1 1, 2 7, 8 7, 7 3, 1 1), (2 4, 3 6, 4 5, 2 4), (4 3, 6 6, 6 4,␣
↪4 3))
'''
```



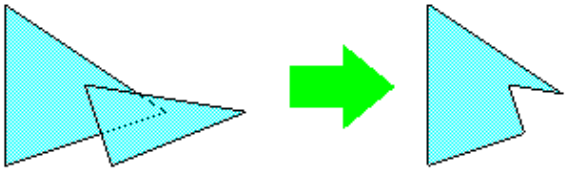
Центроид объекта axipy.Geometry.centroid()

```
centroid = poly.centroid()
print(centroid.to_wkt())
'''
>>> POINT (4 4.5)
'''
```



Вычитание объектов `axipy.Geometry.difference()`

```
poly1 = Polygon((1, 1), (1, 7), (7, 3))
poly2 = Polygon((5, 1), (4, 4), (10, 3))
print(poly1.difference(poly2).to_wkt())
'''
>>> POLYGON ((1 1, 1 7, 6 3.67, 4 4, 4.6 2.2, 1 1))
'''
```



Пересечение объектов `axipy.Geometry.intersection()`

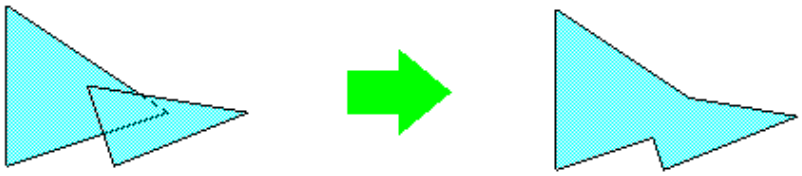
```
print(poly1.intersection(poly2).to_wkt())
'''
>>> POLYGON ((6 3.67, 7 3, 4.6 2.2, 4 4, 6 3.67))
'''
```

Обратное пересечение объектов `axipy.Geometry.symmetric_difference()`

```
print(poly1.symmetric_difference(poly2).to_wkt())
'''
>>> MULTIPOLYGON (((1 1, 1 7, 6 3.67, 4 4, 4.6 2.2, 1 1)), ((4.6 2.2, 7 3, 6 3.67, 10␣
↪3, 5 1, 4.6 2.2)))
'''
```

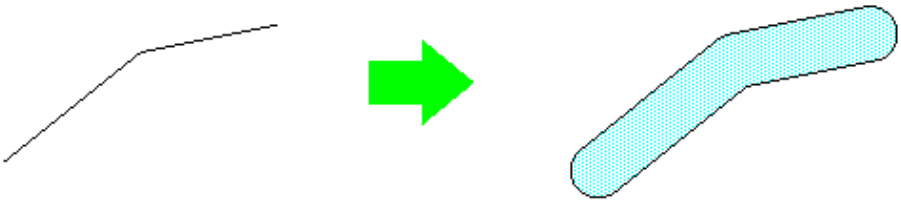
Объединение объектов `axipy.Geometry.union()`

```
print(poly1.union(poly2).to_wkt())
'''
>>> POLYGON ((1 1, 1 7, 6 3.67, 10 3, 5 1, 4.6 2.2, 1 1))
'''
```



Построение буфера `axipy.Geometry.buffer()`

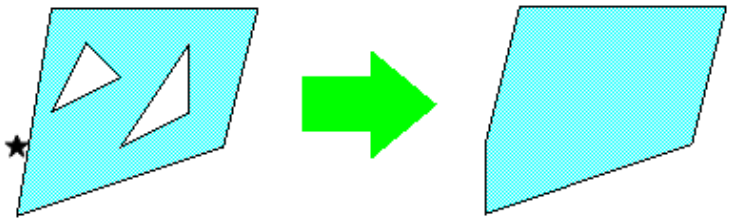
```
sl = LineString((0, 1), (5, 5), (10, 6))
buf = sl.buffer(1)
```



Границы объекта `axipy.Geometry.convex_hull()`

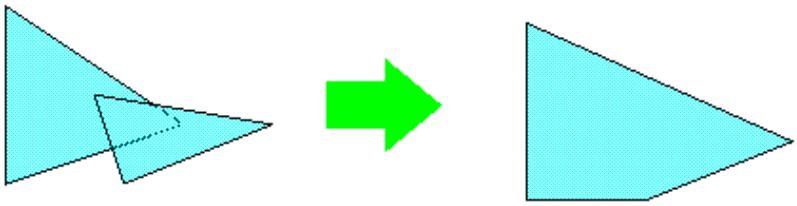
Рассмотрим на примере коллекции:

```
coll = GeometryCollection()
coll.append(polygon)
coll.append(Point(1, 5))
print(coll.convex_hull().to_wkt())
'''
POLYGON ((1 1, 1 5, 2 7, 8 7, 7 3, 1 1))
'''
```



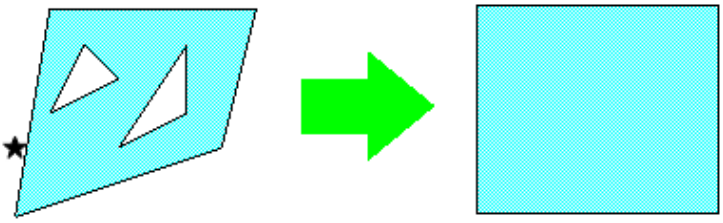
Пример с внутренними углами:

```
print(poly1.union(poly2).convex_hull().to_wkt())
'''
POLYGON ((1 1, 1 7, 10 3, 5 1, 1 1))
'''
```



Прямоугольные границы объекта `axipy.Geometry.envelope()`

```
print(coll.envelope().to_wkt())
'''
POLYGON ((1 1, 8 1, 8 7, 1 7, 1 1))
'''
```



8.5.6 Конвертация объектов

Список 1: Пример конвертации полигона в полилинию.

```
polygon = Polygon((0, 0), (0, 10), (10, 10))
print(polygon.to_linestring().to_wkt())
'''
>>> LINESTRING (0 0, 0 10, 10 10, 0 0)
'''
```

Список 2: Пример конвертации полилинии в полигон.

```
ls = LineString((0, 0), (0, 10), (10, 10))
print(ls.to_polygon().to_wkt())
'''
>>> POLYGON ((0 0, 0 10, 10 10, 0 0))
'''
```

9 Стиль

Стиль представляет собой описательную структуру, которая в свою очередь используется при оформлении геометрического объекта `Geometry` при его отрисовке. Стиль представлен базовым классом `axipy.Style`, а так-же его наследниками.

При работе с табличными данными стиль, при наличии в ней геометрического атрибута, может определяться тремя разными способами:

- Содержаться в специальной колонке таблицы в виде атрибута. В данном случае для геометрии каждой записи таблицы назначается соответствующий ей стиль.
- Определяется для колонки на уровне таблицы. В данном случае геометрия всех записей будет иметь одинаковое оформление.
- В таблице присутствует только геометрия. В данном случае стиль при отрисовке слоя будет браться как значение по умолчанию.

Рассмотрим в дальнейшем первый вариант. Для выполнения последующих примеров создадим таблицу в памяти и зарегистрируем ее в системе:

```
definition = {
    'src': '',
    'schema': Schema(
        Attribute.string('id', 60),
        coordsystem='prj:1, 104'
    )
}
table = provider_manager.create(definition)
```

Далее попробуем различными методами добавить геометрию в эту таблицу. На примере точечного объекта. Создадим точечный объект, и, если стиль оформления не имеет значения, назначим ему наиболее подходящий для данного типа (в нашем случае точки) объекта, просто передав туда нашу геометрию:

```
point = Point(10, 8)
pstyle = Style.for_geometry(point)
print(pstyle.to_mapinfo())
'''
  >>> Symbol (36, 255, 12, "Map Symbols", 0,0)
'''
```

Для иллюстрации результата сформируем на базе созданных объектов геометрии и стиля запись и добавим его в ранее созданную таблицу. Итог покажем на карте:

```
fpoint = Feature(
    geometry=point,
    style=pstyle
)
table.insert([fpoint])
m = Map([table])
view = view_manager.create_mapview(m)
```

В результате получим точку на карте:

Так же доступно создание из строки формата MapBasic. Для этого используется метод `axipy.Style.from_mapinfo()`. Если же для существующего стиля необходимо получить строку MapBasic, то используется метод `axipy.Style.to_mapinfo()`. Создадим для нашей точки стиль на базе представления MapBasic. Строка формирования стиля точки будет выглядеть так:

```
pstyle = Style.from_mapinfo('Symbol (35, 255, 20)')
```

Пример добавления точки, но с использованием стиля, на основании растрового символа:

```
pstyle = PointStyle.create_mi_picture("GLOB1-32.bmp")
print(pstyle)
fpoint = Feature(
    geometry=point,
    style=pstyle
)
'''
>>> Symbol ("GLOB1-32.bmp", 0, 12, 0)
'''
```

Далее вставим в таблицу по аналогии, рассмотренной выше. Результат:

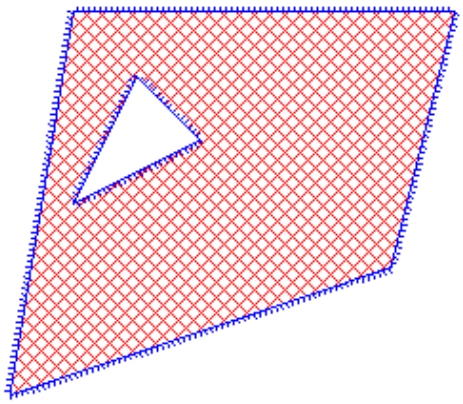
Теперь рассмотрим объект типа полигон.

```
from PySide2.QtCore import Qt

polygon = Polygon((1, 1), (2, 7), (8, 7), (7, 3))
polygon.holes.append((2, 4), (3, 6), (4, 5))
polystyle = PolygonStyle()
polystyle.set_pen(pattern=48, color=Qt.blue)
polystyle.set_brush(pattern=8, color=Qt.red)
print(polystyle)
fpolygon = Feature(
    geometry=polygon,
    style=polystyle
)
```

(continues on next page)

(продолжение с предыдущей страницы)

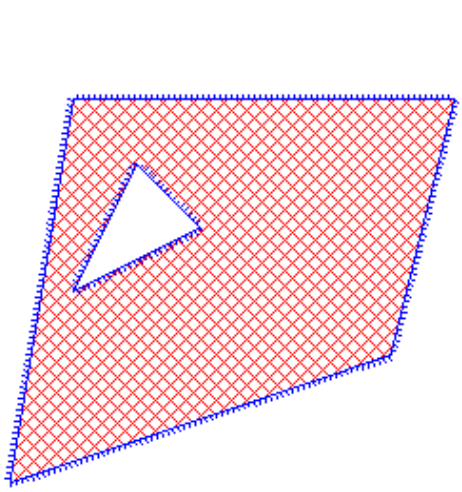
```
table.insert([fpolygon])
'''
>>> Pen (1, 48, 255)  Brush (8, 16711680, 0)
'''
```



Для сложных разнородных объектов применяются стили, которые внутри себя содержат другие стили для каждого типа используемых объектов. Для этого используется класс `axipy.CollectionStyle`. Из ранее созданных полигона и точки сделаем коллекцию посредством объединения. И, одновременно с этим, на базе ранее созданных стилей сделаем сложный стиль:

```
collstyle = CollectionStyle()
collstyle.for_point(pstyle)
collstyle.for_polygon(polystyle)
print(collstyle)
collection = polygon.union(point)
fcollection = Feature(
    geometry=collection,
    style=collstyle
)
table.insert([fcollection])
'''
>>>  Symbol ("GLOB1-32.bmp", 0, 12, 0) Pen (1, 48, 255)  Brush (8, 16711680, 0)
'''
```

В результате получим следующий объект:



10 Окна просмотра

10.1 Слой

Для отображения данных таблицы или растра необходимо создать на основе этого источника данных слой `axipy.Layer`.

```
from axipy.render import Layer

layer = Layer.create(table)
```

В зависимости от типа передаваемого объекта будет создан векторный или растровый слой.

Свойства подписей

Настройки автоматического подписывания определяются классом `Label` свойством `label`.

Список 1: Пример использования

```
world = generate_layer_for_geometry_table()
# Зададим в качестве формулы метки атрибут "Страна" и запретим перекрытие меток друг␣
↪другом:
world.label.text = "Страна"
world.label.placementPolicy = LabelOverlap.DisallowOverlap
# Задание стиля оформления слоя
style_lay = Style.from_mapinfo("Pen (1, 2, 0) Brush (8, 255) Symbol (33,255,14)")
world.overrideStyle = style_lay
# Для сброса переопределения достаточно задать значение None
world.overrideStyle = None
```

Настройки автоматического подписывания `Label` повторяют соответствующие настройки в интерфейсе ГИС Аксиома диалога «Свойства слоя» > «Подписи».

См.также:

Подробнее в разделе «Подписывание» руководства пользователя для ГИС Аксиома.

10.2 Карта

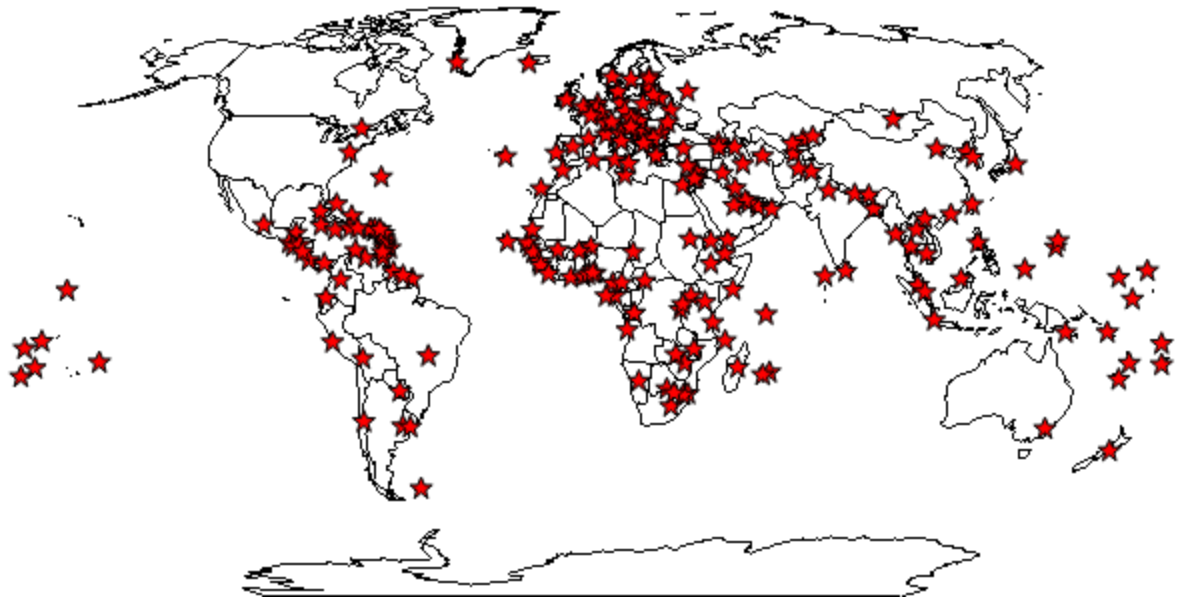
Совокупность слоев образует карту axipy.Map. Порядок отрисовки слоев прямо зависит от их расположения в карте. То есть первый слой будет на самом верху, а последний - в самом низу.

Создадим карту с двумя слоями. Передадим в карту список таблиц - из них автоматически создадутся слои с параметрами по умолчанию.

```
import axipy

world = axipy.provider_manager.openfile('../path/to/datadir/example.gpkg', dataobject=
↪'world')
capital = axipy.provider_manager.openfile('../path/to/datadir/worldcap.tab')
map_ = axipy.Map([capital, world])
```

Карту можно вывести в изображение - PySide2.QtGui.QImage, которое, например, можно в качестве результата сохранить в файл.

```
image = map_.to_image(600, 300)
```



Полученную картинку можно сохранить как растр в файловой системе. Формат файла будет определяться его расширением:

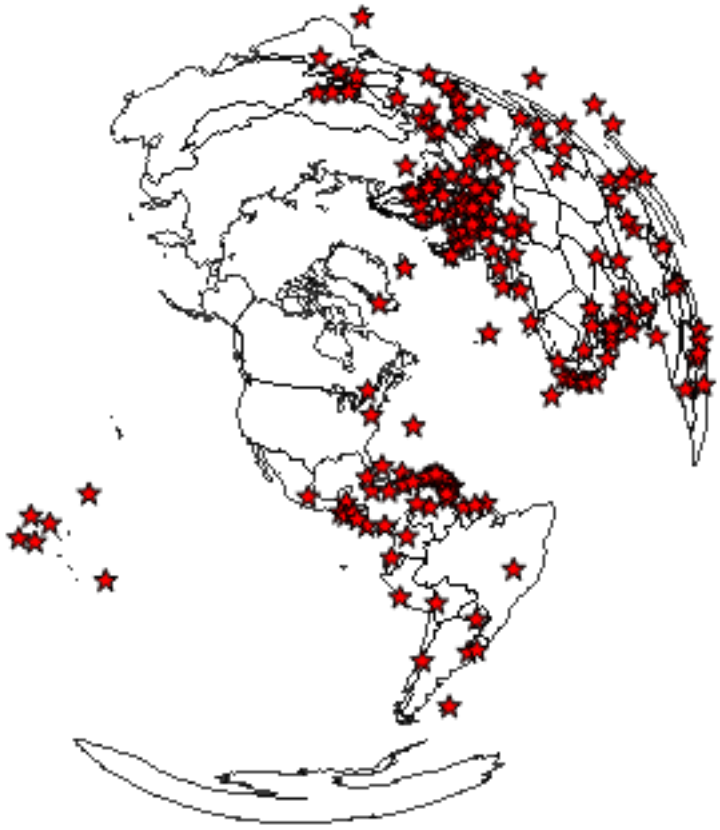
```
print(image.save('../path/to/outdir/map.png'))
```

```
>>> True
```

Мы указали только размеры изображения. Карта вывелась в Системе Координат (СК), наиболее подходящей для отображения слоев в ней. В нашем случае обе таблицы оказались в одной СК, которая и была выбрана для отображения.

Теперь отрисуем эту же карту Азимутальной СК:

```
from axipy.cs import CoordSystem

azimuth = CoordSystem.from_epsg(2163)
image = map_.to_image(600, 300, coordsystem=azimuth)
```



Границы карты по умолчанию определились равными границам СК. Снова нарисуем нашу карту уже в Широте/Долготе в границах, примерно включающих Италию. Ограничивающий прямоугольник указываем в градусах:

```
longlat = CoordSystem.from_epsg(4326)
image = map_.to_image(600, 300, coordsystem=longlat, bbox=(5, 35, 15, 15))
```

Теперь попробуем для слоя capital задать выражение для отображаемых меток axipy.VectorLayer.label, и для обоих слоев переопределить стиль оформления, сделав его однообразным axipy.VectorLayer.overrideStyle. Заметим, что перечень доступных слоев карты доступен через свойство axipy.Map.layers. Т.е. помимо передачи перечня слоев в конструктор карты, также возможно управление этим списком позже.

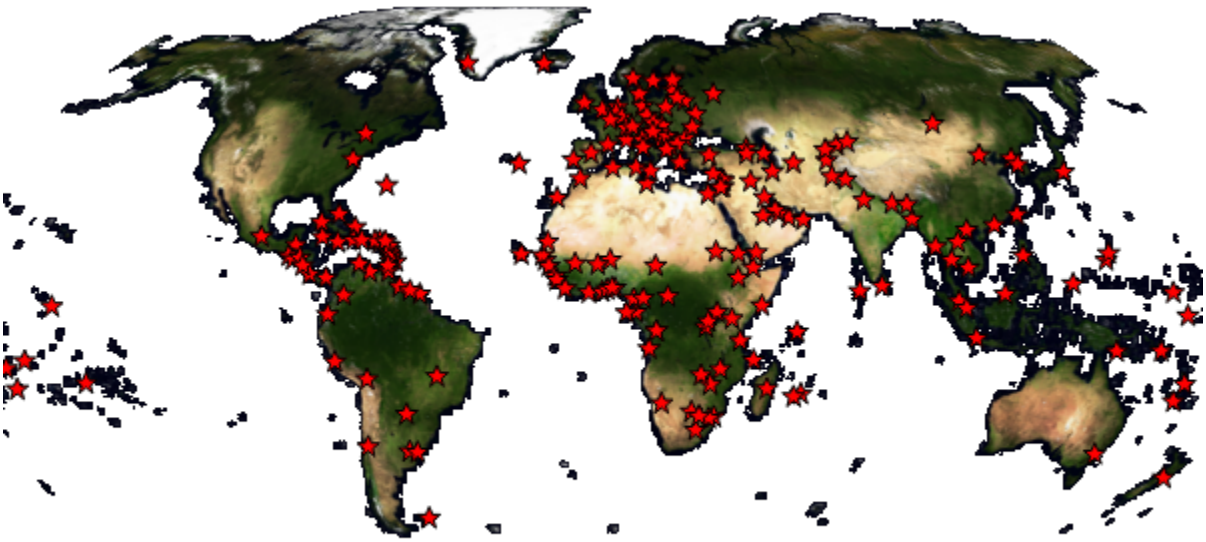
```
from axipy import Label, Style

lay_capital = map_.layers[0]
lay_capital.label.text = 'Столица'
lay_capital.label.placementPolicy = Label.DisallowOverlap
lay_capital.overrideStyle = Style.from_mapinfo('Symbol (34,255,6)')
lay_world = map_.layers['world']
lay_world.overrideStyle = Style.from_mapinfo('Pen (1, 2, 0) Brush (8, 255)')
image = map_.to_image(600, 300, coordsystem=longlat, bbox=(5, 35, 15, 15))
```

В рамках примера по управлению слоями в конце удалим слой со столицами (самый верхний):

```
map_.layers.remove(0)
```

Создадим карту на базе растра как подложки и установим прозрачным цвет фона.

```
from PySide2.QtGui import QColor
from axipy import provider_manager, Layer, Map

raster = provider_manager.openfile("../path/to/datadir/TrueMarble.tab")
rasterLayer = Layer.create(raster)
rasterLayer.transparentColor = QColor("#000014")

mapRaster = Map([capital, rasterLayer])
image = mapRaster.to_image(600, 320)
```



10.2.1 Окно редактирования карты

Окно карты axipy.Map, созданное программно, можно экспортировать в виде растра или же поместить в отчет. Если же стоит задача проведения некоторых манипуляций с картой, таких как редактирование геометрии, то для проведения подобных манипуляций ее необходимо поместить в окно редактирования axipy.MapView. Для создания этого окна необходимо использовать метод axipy.ViewManager.create_mapview(). Данный метод доступен через сервис view_manager. Рассмотрим на примере:

Список 2: Пример использования.

```
# Откроем таблицу, создадим на ее базе слой и добавим в карту
table_world = provider_manager.openfile(filepath)
world = Layer.create(table_world)
map = Map([world])
# Для полученной карты создадим окно просмотра
mapview = view_manager.create_mapview(map)
```

При желании непосредственно в окно карты можно поместить элементы управления. Рассмотрим пример добавления ползунка в левый верхний угол окна карты, который изменяет прозрачность верхнего слоя карты:

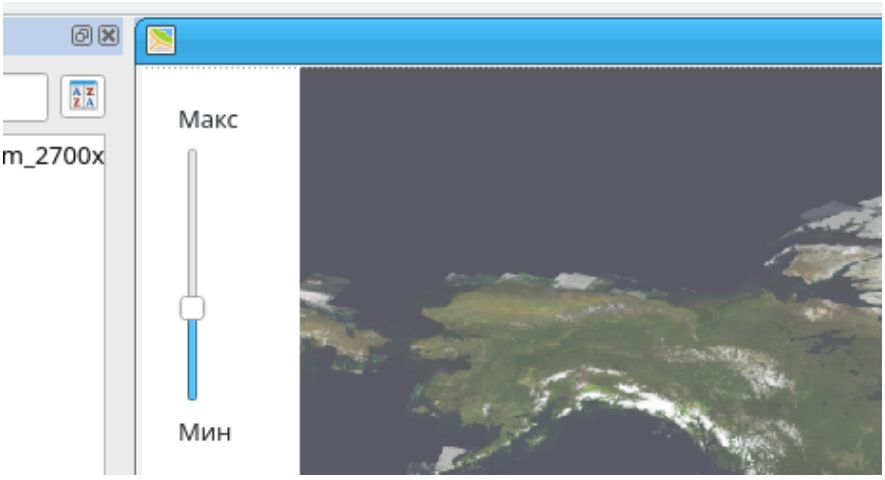
```
from PySide2.QtWidgets import QSlider, QFrame, QLabel, QVBoxLayout
from PySide2.QtCore import Qt

def change_opacity(v):
    mv = view_manager.active
    if mv and len(mv.map.layers):
        mv.map.layers[0].opacity = 100 - v

mv = view_manager.active
if mv is not None:
    frame = QFrame(mv.widget)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    layout = QVBoxLayout()
    slider = QSlider(Qt.Vertical, frame)
    slider.setRange(0, 100)
    slider.valueChanged.connect(change_opacity)
    layout.addWidget(QLabel("Макс", frame))
    layout.addWidget(slider)
    layout.addWidget(QLabel("Мин", frame))
    frame.setGeometry(10, 10, 50, 300)
    frame.setLayout(layout)
    frame.show()
```

Результат будет выглядеть примерно так:



10.3 Тематические слои

Тематическая карта отображает ваши данные в виде условных знаков, выделяя их оттенками, цветами, штриховками, а также представляя их в виде столбчатых и круговых диаграмм.

Для векторных слоев `axipy.VectorLayer` есть возможность формирования и отрисовки тематических слоев. Т.е. применить оформление на базе атрибутивной информации.

Тематические слои добавляются как дочерние к их базовому слою.

```
from axipy import RangeThematicLayer

world = map_.layers[0]
thematic = RangeThematicLayer('Население')
world.thematic.add(thematic)
```

Поддерживаются следующие виды тематических слоев:

- RangeThematicLayer - Интервалы
- PieThematicLayer - Круговые диаграммы
- BarThematicLayer - Столбчатые диаграммы
- SymbolThematicLayer - Знаки

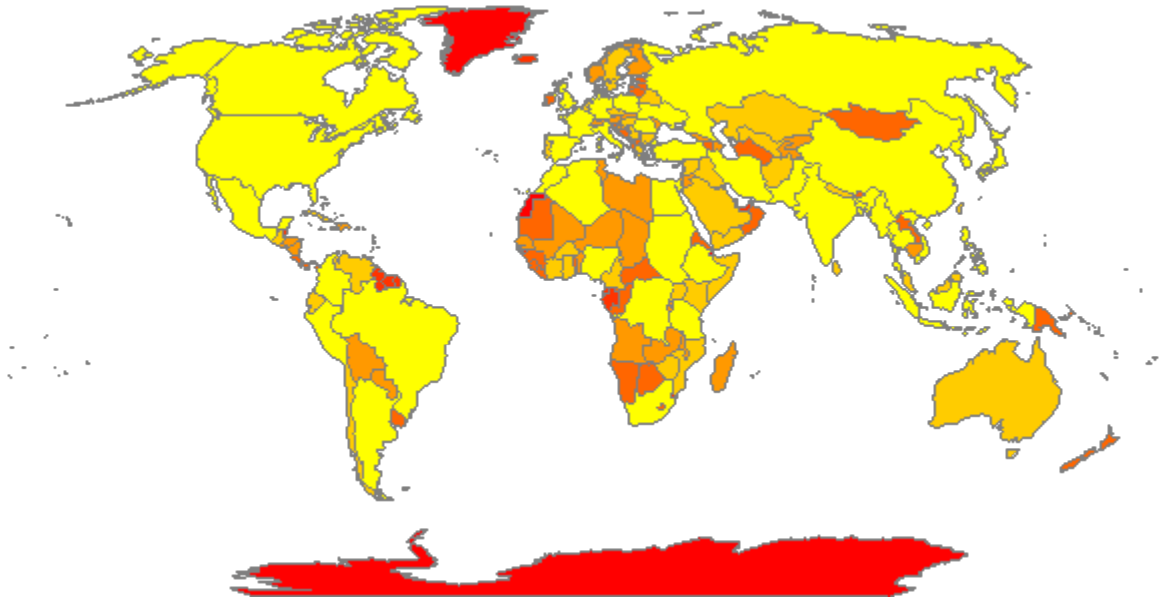
- IndividualThematicLayer - Индивидуальные значения
- DensityThematicLayer - Плотность точек

Для более удобного распределения по цветам используются различного рода алгоритмы. Выбор того или иного алгоритма обусловлен исходными требованиями к составлению тематики. Эти алгоритмы сгруппированы в базовом интерфейсе `axipy.ReallocateThematicColor` и могут быть использованы в наследниках. Поддерживаются следующие виды распределения:

- По двум заданным крайним цветам `axipy.ReallocateThematicColor.assign_two_colors()`.
- По двум заданным крайним цветам и цвету разрыва `axipy.ReallocateThematicColor.assign_two_colors()`.
- По спектру `axipy.ReallocateThematicColor.assign_rainbow()`.
- Градация серого `axipy.ReallocateThematicColor.assign_gray()`.
- Монотонная заливка разной яркости `axipy.ReallocateThematicColor.assign_monotone()`.

Рассмотрим на примере тематики по интервалам. Построим тематику по атрибутивному полю “Население” на 6 интервалов с равномерным распределением по количеству записей. Цвета распределим градиентом от желтого до красного.

```
table_world = provider_manager.openfile('world.tab')
world = Layer.create(table_world)
range1 = RangeThematicLayer("Население")
range1.ranges = 6
range1.splitType = RangeThematicLayer.EQUAL_COUNT
range1.assign_two_colors(Qt.yellow, Qt.red)
world.thematic.add(range1)
```



Поменяем стиль оформления для первого интервала:

```
range1.set_style(0, PolygonStyle(45, Qt.blue))
```

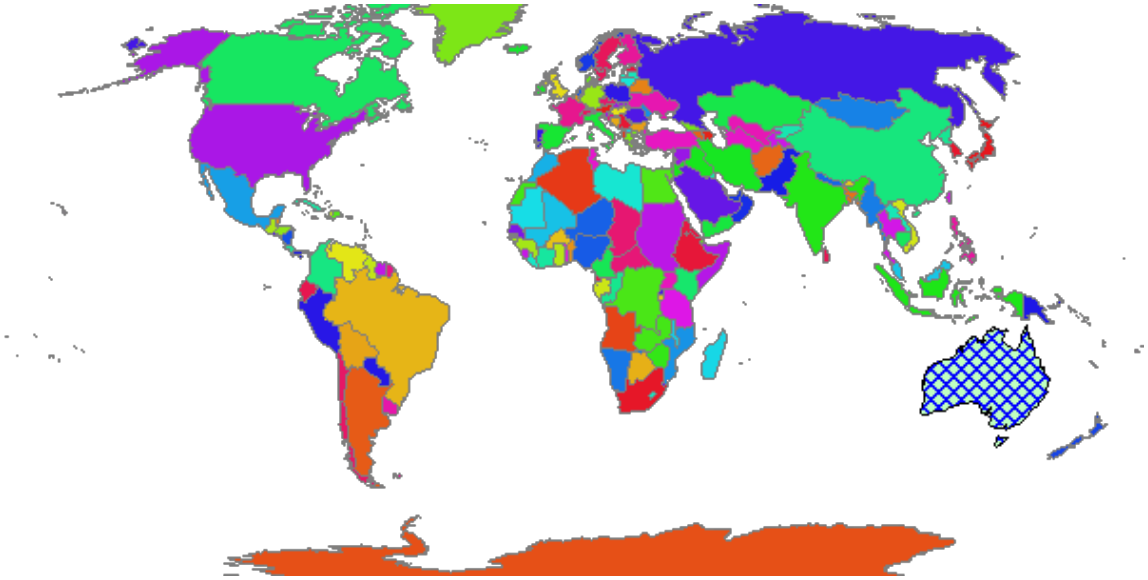
Так же есть возможность ручного переопределения значений разбивки по интервалам. Т.е. задание минимального и максимального значений требуемого интервала. Переопределим верхнее значение для первого интервала:

```
iv = range1.get_interval_value(0)
print('Old values:', iv)
range1.set_interval_value(0, (iv[0], 100000.0))
print('New values:', range1.get_interval_value(0))
```

```
>>> Old values: (0.0, 66687.0)
>>> New values: (0.0, 100000.0)
```

Создадим тематику с отдельными значениями по полю „Страна“. Распределение по цветам случайное:

```
individual = IndividualThematicLayer('Страна')
individual.assign_rainbow()
world.thematic.add(individual)
individual.set_style(0, PolygonStyle(45, Qt.blue))
```



Изменим цвет интервала по индексу 1 на желтый.

```
s = individual.get_style(1)
s.polygon.fill.color = Qt.yellow
individual.set_style(1, s)
```

Необходимую тематику слоя можно получить по ее индексу `axipy.Layer.thematic()`:

```
range1 = world.thematic[0]
```

Если необходимо просмотреть все тематики слоя:

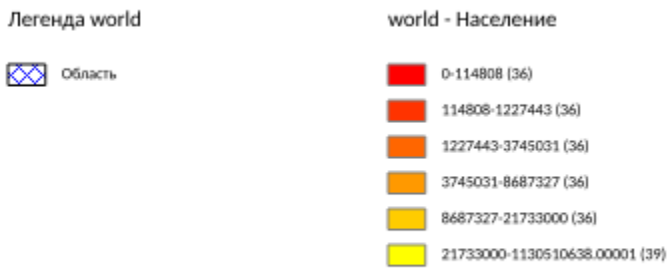
```
for t in world.thematic:
    print('thematic:', t.title)
```

```
>>> thematic: Интервалы
>>> thematic: Значения
```

10.4 Легенда

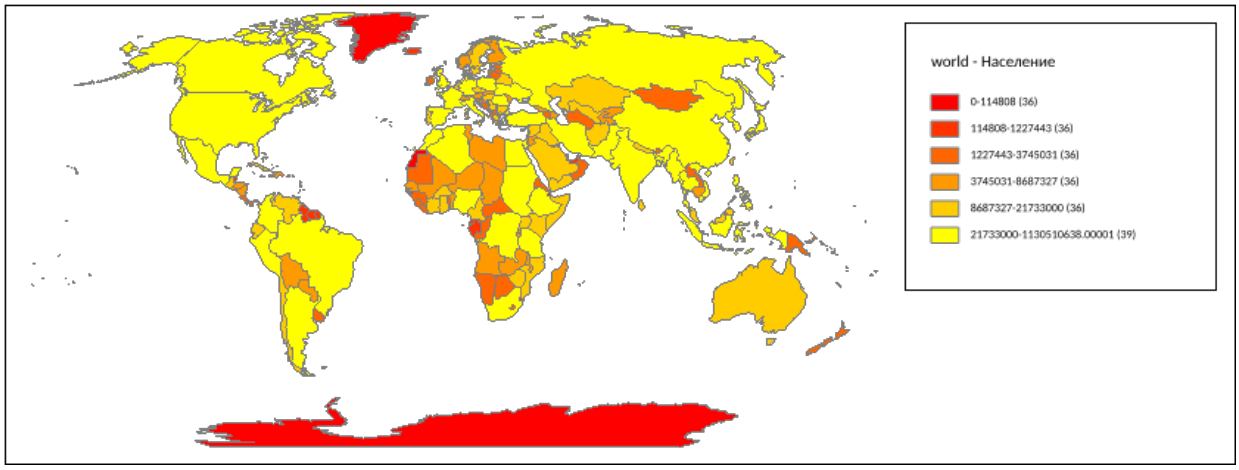
Для вывода условных обозначений используется легенда. Легенда – таблица, содержащая образцы условных обозначений и письменных пояснений к ним. В ГИС Аксиома легенды карт представляются в отдельных окнах. Попробуем отрисовать в растре ранее созданные слой и тематику по интервалам. Заметим, что легенду также можно отрисовать на одном растре вместе с картой.

```
from PySide2.QtCore import Qt
from PySide2.QtGui import QImage, QPainter
from axipy import Legend, Context

legend_world = Legend(lay_world)
legend_world.position = (10, 10)

legend_thematic = Legend(thematic)
legend_thematic.position = (200, 10)

image = QImage(500, 200, QImage.Format_ARGB32_Premultiplied)
image.fill(Qt.white)
painter_legend = QPainter(image)
context_legend = Context(painter_legend)

legend_world.draw(context_legend)
legend_thematic.draw(context_legend)
```



Создадим легенду, на этот раз сразу переведя в `PySide2.QtGui.QImage`, и скомпонуем с тематическим слоем, полученным ранее:

```
from axipy.render import Legend

legend_thematic = Legend(thematic)
legend_thematic.position = (10, 5)
legend_image = legend_thematic.to_image(200, 170)
```



Доступ к элементам легенды производится через свойство `axipy.Legend.items`.

```
for it in legend_thematic.items:
    print(it.title, it.visible, it.style.to_mapinfo())
```

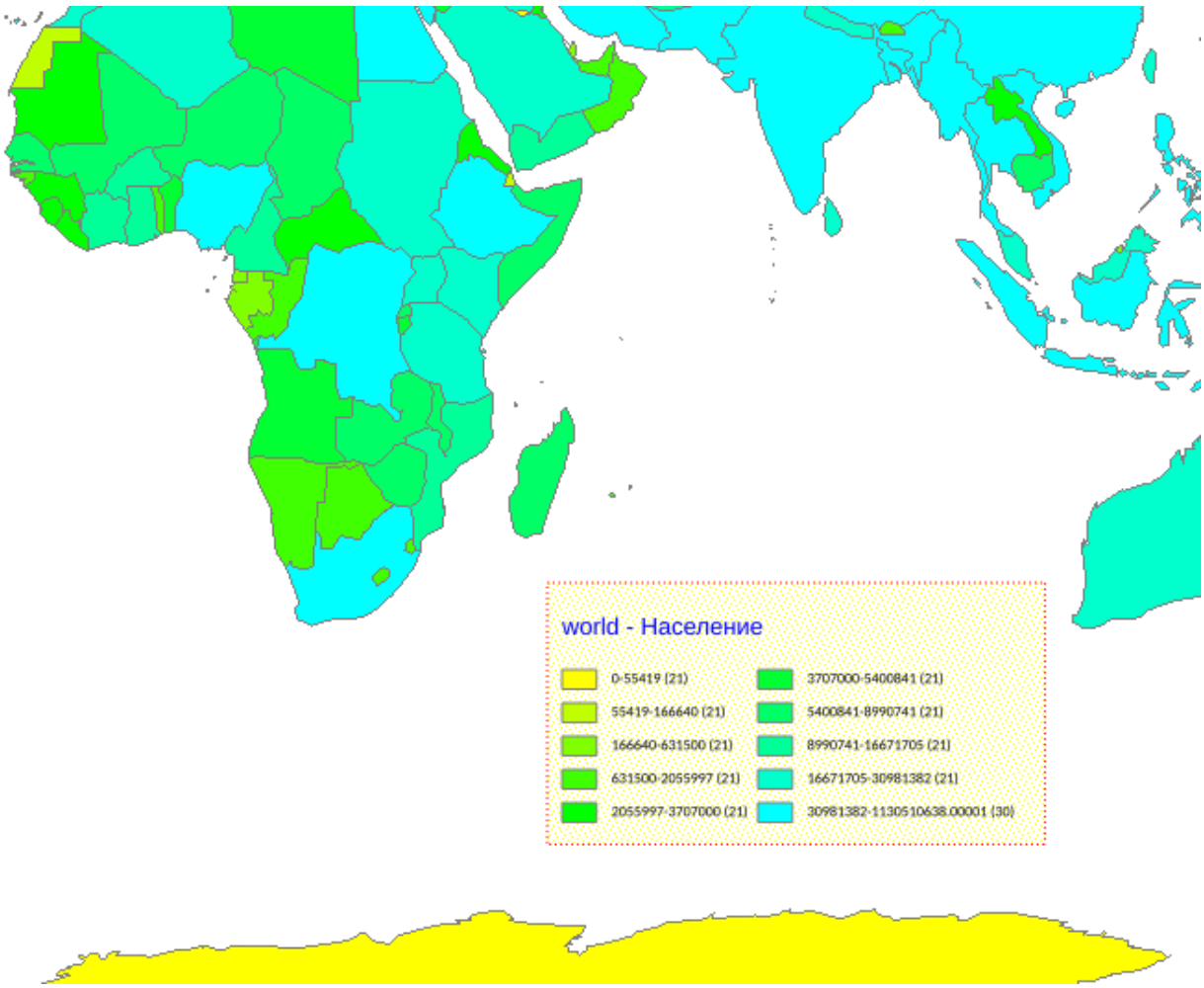
```
>>> 0-55419 True Pen (1, 2, 0) Brush (45, 255)
>>> 166640-631500 True Pen (1, 2, 8421504) Brush (2, 8453888)
>>> 631500-2055997 True Pen (1, 2, 8421504) Brush (2, 4259584)
```

Если требуется изменить какой-то элемент. к примеру сделать его скрытым или изменить описание, то в данном случае необходимо сначала запросить требуемый элемент, изменить его характеристики и обновить на модифицированный. Прямое изменение не поддерживается.

```
item = legend.items[0]
item.title = 'Описание'
legend.items[0] = item
```

Если легенду необходимо как-то выделить на общем фоне или она с ним сливается, то можно указать стиль рамки и заливки заднего фона:

```
legend.border_style = LineStyle(3, Qt.red)
legend.fill_style = PolygonStyle(49, Qt.yellow)
```



10.4.1 Окно легенды для карты

Для просмотра и редактирования легенды axipy.Legend для карты необходимо использовать метод axipy.ViewManager.create_legendview(), передав в него как параметр окно ранее созданной карты:

Список 3: Пример использования.

```
map = Map([world])
mapview = view_manager.create_mapview(map)
legendView = view_manager.create_legendview(mapview)
```

При этом будут помещены все доступные для просмотра легенды слоев карты map_view.

Легенды окна можно посмотреть следующим образом:

Список 4: Пример использования.

```
for legend in legendView.legends:
    print(legend.caption)
'''
```

(continues on next page)

(продолжение с предыдущей страницы)

```
>>> World
'''
```

10.5 Отчет

Для вывода информации на печать предусмотрено создание отчетов. Отчет формируется на базе стандартного подхода работы с принтером в Qt PySide2.QtPrintSupport.QPrinter. Создадим макет отчета, в который поместим геометрический объект и карту. Вывод сделаем в файл формата PDF. Для этого предварительно создадим объект принтера и установим необходимые свойства

```
from PySide2.QtPrintSupport import QPrinter

printer = QPrinter()
printer.setOutputFormat(QPrinter.PdfFormat)
printer.setOutputFileName('../path/to/outdir/report.pdf')
```

Далее, создадим сам отчет и в конструктор передадим созданный ранее принтер.

```
from axipy import Report

report = Report(printer)
```

Создадим геометрический элемент и добавим его в отчет. Координаты в единицах измерения листа принтера.

```
geometryReportItem = GeometryReportItem()
geometryReportItem.geometry = Polygon((10, 10), (10, 100), (100, 100), (10, 10))
geometryReportItem.style = PolygonStyle(45, Qt.red)
report.items.add(geometryReportItem)
```

Аналогично добавим карту.

```
table = provider_manager.openfile('world.tab')
world = Layer.create(table)
map = Map([world])
mapReportItem = MapReportItem(Rect(10, 110, 200, 210), map)
mapReportItem.scale = 200000000
report.items.add(mapReportItem)
```

Контекст для печати по подобию рассмотренному контексту для карты.

```
from PySide2.QtGui import QPainter

painterReport = QPainter(printer)
context = Context(painterReport)
```

Производим печать.

```
report.draw(context)
```

В результате в файловой системе мы получим файл report.pdf, который содержит геометрический элемент и карту.

10.5.1 Окно редактирования отчета

Если необходимости в визуальном редактировании отчета `axipy.Report` нет, а требуется только программно создать макет отчета и распечатать его на принтере или сохранить как PDF, то достаточно создать `axipy.Report` и работать с ним. В противном случае отчет, по аналогии с картой для визуального редактирования его так же помещают в окно редактирования `axipy.ReportView`. В качестве примера создадим окно отчета и поместим в него геометрию и карту. Стоит заметить, что карту так-же можно использовать у уже открытого окна карты `axipy.MapView.map()`. Координаты элементов задаются в единицах измерения отчета `axipy.Report.unit`. В нашем случае это миллиметры.

Список 5: Пример использования.

```
from PySide2.QtCore import Qt
rv = view_manager.create_reportview()

# Добавим полигон
geomItem = GeometryReportItem()
geomItem.geometry = Polygon((10, 10), (10, 100), (100, 100), (10, 10))
geomItem.style = PolygonStyle(45, Qt.red)
rv.report.items.add(geomItem)

# Добавим направляющую
rv.x_guidelines.append(20)

# Текущий масштаб просмотра
rv.view_scale = 33

# Включение режима привязки координат
rv.snap_mode = True

# Размер ячейки сетки
rv.mesh_size = 20

# Откроем и добавим карту
table = provider_manager.openfile(filepath)
world = Layer.create(table)
map = Map([world])
mapItem = MapReportItem(Rect(10, 100, 200, 200), map)
mapItem.scale = 200000000
rv.report.items.add(mapItem)

# Поменяем стиль у первого объекта
rv.report.items[0].style = PolygonStyle(45, Qt.blue)
```

11 Создание кнопок

11.1 Создание кнопки

Расположение кнопки в интерфейсе ГИС Аксиома определяется Вкладкой и Группой. Например, вкладка “Основные” группа “Команды”. В модуле `axipy.menubar` есть необходимые функции для создания кнопок.

```
from axipy import ActionButton, Position, tr

button = ActionButton("Простое действие", on_click=lambda: print("triggered"))
position = Position(tr("Основные"), tr("Команды"))
position.add(button)
```

Детально разберем, что делает этот пример.

Создается кнопка с текстом “Простое действие”, и ,используя параметр `on_click`, привязывается нажатие на кнопку к анонимной лямбде, которая печатает в консоль текст «triggered». Это обработчик нажатия кнопки. Обработчиком может служить любой callable-объект(функтор) без параметров, т.е. функции, лямбды, объекты с методом `__call__`. Также можно задать иконку кнопки параметром `icon=`. Иконкой может быть строка-ссылка на ресурс или объект типа `PySide2.QtGui.QIcon`.

Далее ищется расположение в интерфейсе. Если вкладка или группа с такими именами отсутствуют, то они будут созданы при добавлении кнопки.

В последней строке кнопка добавляется в заданное расположение.

11.2 Доступность кнопки

В приложении ГИС Аксиома многие кнопки и инструменты меняют свойство доступности в зависимости от текущего состояния. Например, если кнопка производит действие над выбранным объектом, то логично сделать эту кнопку доступной только при наличии выбранных объектов. Чтобы проще задавать подобное поведение для своих кнопок, предлагается использовать `axipy.Observer` и место их создания и получения `axipy.ObserverManager`.

Так, чтобы сделать кнопку активной только при наличии выборки, ее можно создать следующим образом, передав параметр `enable_on` функции `axipy.menubar.create_button()`:

```
from axipy import menubar, ObserverManager

button = menubar.create_button(
    "Имя кнопки",
    on_click=lambda: print("on_click"),
    enable_on=ObserverManager.Selection
)
```

Идентификаторы наблюдателей по умолчанию перечислены в `axipy.ObserverManager`.

Для инструментов идентификатор рекомендуется задавать в самом классе инструмента, переопределив параметр `axipy.MapTool.enable_on`.

Возможно добавление своих наблюдателей, в том числе их комбинация с уже существующими `axipy.ObserverManager.create()`.

Чтобы получить текущее значение наблюдателя, используется свойство `axipy.Observer.value`.

12

Создание виджетов

- Программное наполнение диалога
- Использование файла ресурсов *.ui

Для построения пользовательского интерфейса библиотека Qt, поставляемая в рамках ГИС Аксиома, предоставляет большой набор инструментария.

Для примера рассмотрим построение простейшего диалога. Целью данного примера является краткое описание того инструментария, который можно использовать для создания пользовательских форм и диалогов.

Для того, чтобы наш диалог обладал некоторыми специфичными свойствами, унаследуем его от стандартного базового класса PySide2.QtWidgets.QDialog и переопределим его поведение в соответствие с нашими потребностями. Первоначальный код будет выглядеть примерно так:

```
from PySide2.QtWidgets import QDialog
from axipy import view_manager


class MyDialog(QDialog):
    pass


dialog = MyDialog(view_manager.global_parent)
dialog.resize(500, 300)
dialog.exec()
```

Здесь мы создали пользовательский класс диалога и активировали его. Отметим, что свойство view_manager.global_parent, переданное в конструктор рассматривается как объект-владелец данного диалога. Если данное свойство не проставить, то диалог в панели задач будет показан как отдельное приложение.

Тестирование можно производить в рамках самой ГИС Аксиомы. Для этого необходимо открыть редактор. Кнопка запуска находится на панели «Консоль Python». Если она отсутствует в интерфейсе, то для включения необходимо в выпадающем меню кнопки «Панели» включить пункт «Консоль Python». Сохраним данный файл в файловой системе под именем mydialog.py.

Далее, чтобы разместить на данном диалоге элементы управления можно пойти двумя путями:

- Создать эти элементы программно
- Использовать для этого файл ресурсов *.ui

Второй вариант более понятен и удобен, но мы вкратце рассмотрим оба подхода.

12.1 Программное наполнение диалога

Создадим простой диалог, на который программно добавим кнопку и поле ввода. По нажатию на кнопку в консоль будет выведен текст элемента ввода.

```
from PySide2.QtWidgets import QDialog, QPushButton, QLineEdit
from axipy import view_manager


class MyDialog(QDialog):
    def __init__(self, parent):
        super().__init__(parent)
        self.button = QPushButton(self)
        self.button.setText("Кнопка")
        self.button.move(150, 50)
        # Реакция на нажатие кнопки
        self.button.clicked.connect(self.button_clicked)
        self.edit = QLineEdit(self)
        self.edit.setText("Текст")
        self.edit.move(10, 50)

    def button_clicked(self):
        print(self.edit.text())


dialog = MyDialog(view_manager.global_parent)
dialog.resize(500, 300)

dialog.exec()
```

В данном примере элементы располагаются с явным указанием их места. Это не совсем удобно и в зависимости от размеров элементов управления на конкретной системе могут накладываться друг на друга. Или при изменении размеров самой формы могут вообще исчезать. Для решения этой проблемы обычно используются классы динамического размещения элементов на форме. В нашем случае это наследники класса `PySide2.QtWidgets.QLayout`. Модифицируем пример с использованием класса `PySide2.QtWidgets.QGridLayout`.

```
from PySide2.QtCore import Qt
from PySide2.QtWidgets import QDialog, QPushButton, QLineEdit, QGridLayout
from axipy import view_manager


class MyDialog(QDialog):
    def __init__(self, parent):
        super().__init__(parent)
        # создаем layout
        layout = QGridLayout()
        self.button = QPushButton()
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        self.button.setText("Кнопка")
        self.button.clicked.connect(self.button_clicked)
        # Добавляем кнопку
        layout.addWidget(self.button, 0, 0, Qt.AlignTop)
        self.edit = QLineEdit()
        self.edit.setText("Текст")
        # Добавляем элемент ввода
        layout.addWidget(self.edit, 0, 1, Qt.AlignTop)
        # Назначаем layout для диалога
        self.setLayout(layout)

    def button_clicked(self):
        print(self.edit.text())


dialog = MyDialog(view_manager.global_parent)
dialog.resize(500, 300)

dialog.exec()
```

Теперь наши элементы при изменении размеров диалога будут пропорционально менять свои размеры, прижимаясь к верхнему краю.

Если элементов на форме много, то такой способ размещения элементов достаточно трудоемок. Рассмотрим альтернативный способ размещения с использованием дизайнера Qt Designer.

12.2 Использование файла ресурсов *.ui

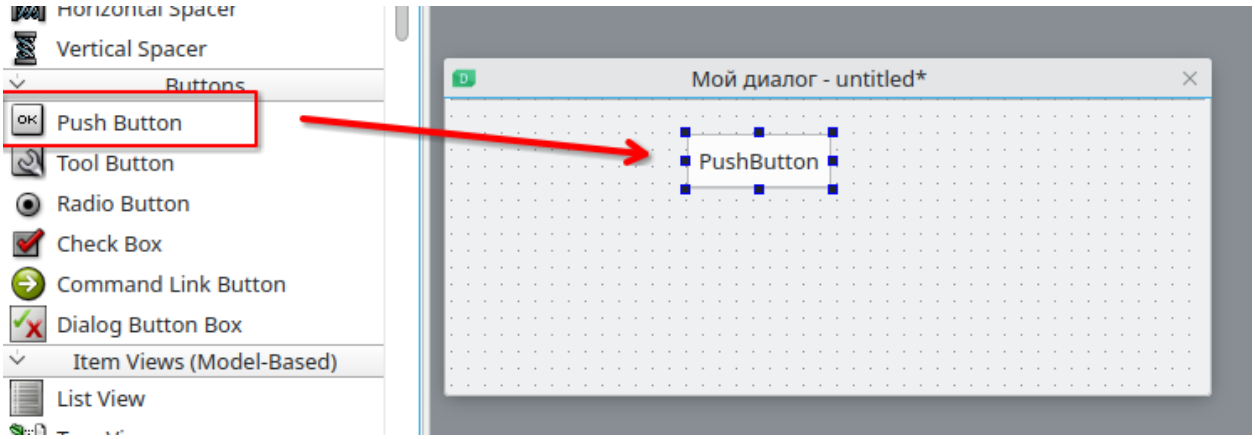
Для формирования UI представления формы нам понадобится инструмент Qt Designer. Он поставляется совместно со средствами разработки Qt, но его также можно установить отдельно как пакет python qt5_applications. Подробнее о инсталляции пакетов см в разделе Без интернета. Ручная установка пакетов.. Т.е. в конечном итоге в зависимости от окружения нам нужно выполнить команду:

```
python3 -m pip install --user qt5_applications
```

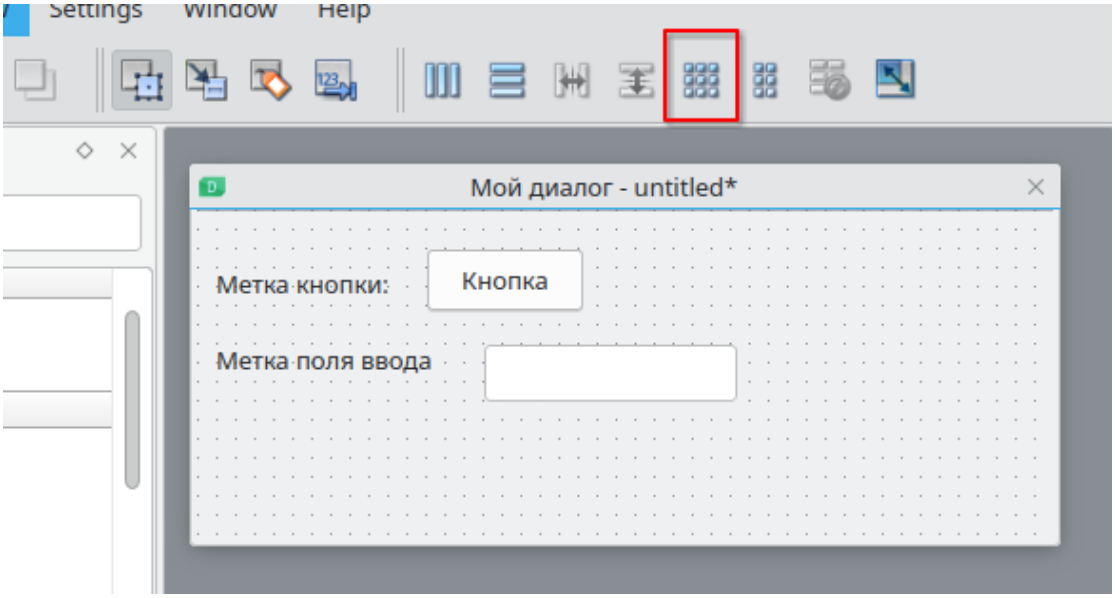
После успешной инсталляции в каталог site-packages из файлового менеджера переходим в каталог site-packages/qt5_applications/Qt/bin и запускаем на выполнение файл designer.

Как альтернативный и более простой вариант, можно воспользоваться возможностью установки и запуска QtDesigner посредством соответвующего модуля. Для этого на вкладке «Основные» необходимо нажать кнопку «Модули». Далее, на вкладке «Дополнительные модули» следует нажать «Загрузить список модулей». Из полученного списка выбираем «Qt Designer» и нажимаем «Установить». На панели «Основные» должна появиться кнопка «QtDesigner».

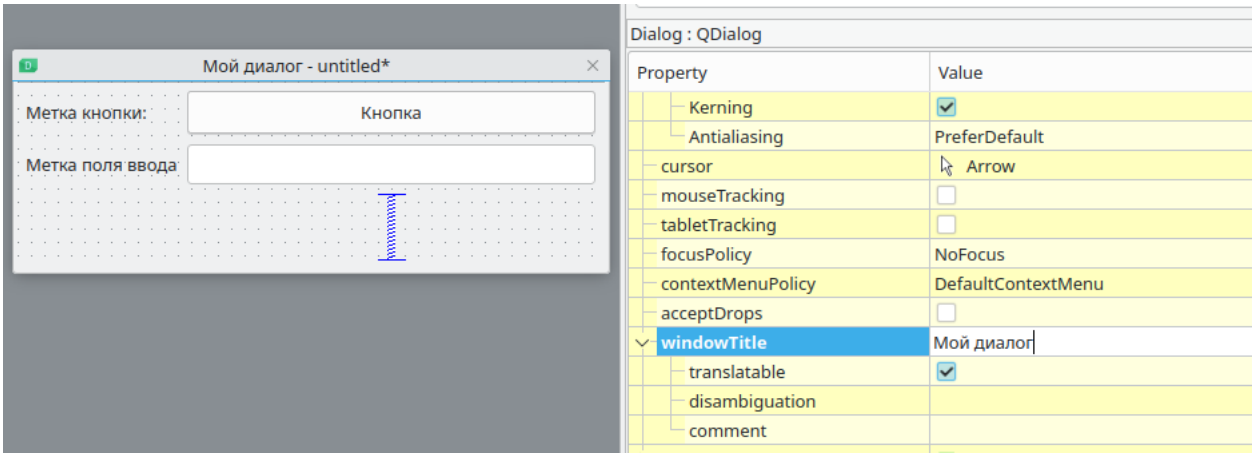
Запускаем инструмент. Выбираем тип диалога или формы. В нашем случае это Dialog without buttons. Далее, помещаем на него посредством перетаскивания мышью кнопку `PySide2.QtWidgets.QPushButton` и элемент ввода `PySide2.QtWidgets.QLineEdit`.



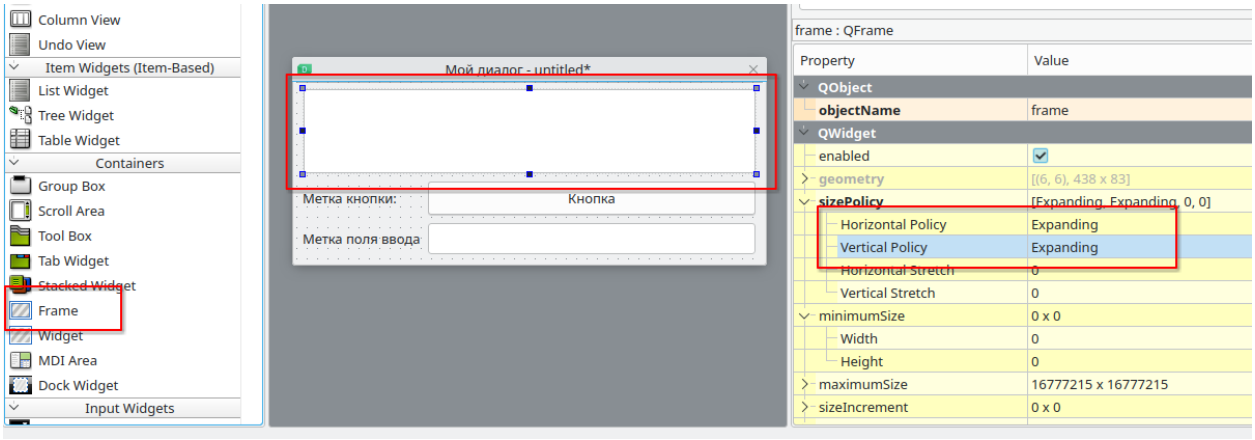
Слева от этих элементов поместить метки, перетащив элемент `PySide2.QtWidgets.QLabel`. Меняем у кнопки ее текст на «Кнопка» (двойным щелчком на элементах) и для поля ввода заносим текст «Текст». Также можно поменять названия элементов на button и edit (чтобы они совпадали с предыдущими примерами). Для этого после выделения соотвествующего элемента, в таблице свойств возле свойства objectName установим значение button. Выберем диалог и установим свойство windowTitle равным «Мой диалог».



Для того, чтобы при изменении размеров формы расположение элементов на ней не уезжало, необходимо применить динамическое выравнивание (Layout). Далее, для нашей формы применим динамическое размещение элементов, выбрав на панели его тип. Для этого выберем мышью основную форму и нажмем кнопку на панели `Lay Out in a Grid`. Элементы займут все пространство и при изменении размеров окна, их размер пропорционально будет изменяться. Чтобы элементы прижать кверху, перетянем по аналогии с кнопкой и полем ввода элемент `Vertical Spacer` и отпустим его над нижней частью (но внутри) окна. После этого наши элементы должны с середины переместиться наверх.



Если мы хотим, чтобы при измении размеров окна изменялись размеры одного из элементов, необходимо сделать следующее: Выделим и удалим ранее добавленный объект Vertical Spacer. После этого находим и перетягиваем объект `PySide2.QtWidgets.QFrame`. Кладем его по аналогии с удаленным Vertical Spacer, но помещаем его сверху на форму над всеми ранее добавленными элементами. Если он занял не все пространство, его необходимо растянуть мышью. После для свойств Horizontal и Verticat Policy устанавливаем значение Expanding.



Сохраняем файл под именем mydialog.ui и кладем его рядом с файлом mydialog.py.

Далее, нам необходимо этот файл загрузить в диалог. Измененный пример с аналогичным функционалом будет выглядеть следующим образом:

```
import os

from PySide2.QtUiTools import QUiLoader
from PySide2.QtWidgets import QDialog
from axipy import view_manager


class MyDialog(QDialog):
    def __init__(self, parent):
        super().__init__(parent)
        # Путь файла в файловой системе
        ui_file = os.path.join(os.path.dirname(__file__), "mydialog.ui")
        # Загружаем файл ui
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        self.ui = QUiLoader().load(ui_file, parent)
        self.ui.button.clicked.connect(self.button_clicked)

    def button_clicked(self):
        print(self.ui.edit.text())

    # Перенаправим метод show на self.ui
    def show(self):
        return self.ui.show()


# Показ диалога как немодальное окно. Для модального используем метод exec
dialog = MyDialog(view_manager.global_parent)
dialog.resize(500, 300)
dialog.show()
```

Стоит заметить, что к элементам теперь необходимо обращаться не через self, а через self.ui. В примерах поставки ГИС Аксиома есть подобное решение с реализацией в виде модуля ru_axioma_gis_axipy_example_dialog.

13

Создание инструментов

13.1 Передача параметров в инструменты

Дополнительные параметры могут быть переданы прямо в конструктор инструмента `axipy.MapTool` при создании кнопки `ToolButton`. Для этого необходимо «обернуть» вызов конструктора в функцию, захватив (capture) все необходимые параметры. Например, сравните:

Список 1: Инструмент без параметров

```
button = ToolButton('Мой инструмент', MyTool)
```

Список 2: Инструмент с захватом параметров

```
button = ToolButton('Мой инструмент', lambda: MyTool(config, params))
```

13.2 Панель активного инструмента

Если при работе с инструментом пользователю регулярно нужно взаимодействовать с различными графическими элементами / настройками то для этого можно воспользоваться готовым решением из `ActiveToolPanel`. Это обертка вокруг стандартной панели `QDockWidget` предоставляющая дополнительные возможности:

1. Можно задать наблюдателя который будет автоматически следить за доступностью панели.
2. Предопределенное место для панели активного инструмента.
3. Готовые компоненты с кнопками управления для упрощения добавления пользовательского графического элемента.

При написании плагинов текущий экземпляр `ActiveToolPanel` можно получить из метода `axipy.AxiomaInterface.active_tool_panel()`. Взаимодействие с панелью активного инструмента идет через обработчик который можно создать через один из методов `make_acceptable()` или `make_custom()`. При работе с панелью активного инструмента через обработчик `AxipyAcceptableActiveToolHandler`

созданный через make_acceptable() то на панели активного инструмента по умолчанию расположены кнопки «Применить» и «Отмена». При нажатии на них будут посланы соответствующие сигналы accepted и rejected. Если нужно настроить кнопки управления по своему усмотрению то можно воспользоваться обработчиком AxipyCustomActiveToolPanelHandler, который создаётся с помощью make_custom().

Список 3: Создание обработчика для взаимодействия с панелью активного инструмента.

```
service = ActiveToolPanel()
# Любой пользовательский графический элемент
widget = QWidget()

# Создаём обработчик для панели активного инструмента через который
# будем управлять панелью.
tool_panel = service.make_acceptable(
    title="Мой инструмент",
    observer_id=DefaultKeys.SelectionEditable,
    widget=widget)

# Подписываемся на сигнал отправляемый после нажатия на кнопку "Применить" в панели
tool_panel.accepted.connect(lambda: print("Применяем изменения"))
```

Чтобы показать панель активного инструмента с переданным ранее графическим элементом нужно вызвать activate(), а для закрытия deactivate().

Список 4: Включение/Выключение панели активного инструмента.

```
class PyTool(MapTool):

    def mousePressEvent(self, event: QMouseEvent) -> Optional[bool]:
        if event.button() == Qt.LeftButton:
            self.tool_panel.activate()
        return self.PassEvent

    def keyPressEvent(self, event: QKeyEvent) -> Optional[bool]:
        if event.key() == Qt.Key_Escape:
            self.tool_panel.deactivate()
            return self.BlockEvent
        return super().keyPressEvent(event
```

См.также:

Создание и добавление кнопок Создание кнопок

В коде выше в методе mousePressEvent() при нажатии левой кнопкой мыши мы показываем панель с активным инструментом, а в методе keyPressEvent() при нажатии на клавишу Esc панель закрываем.

14

Работа с длительными операциями

14.1 Задачи

Скрипты на python выполняются в основном потоке приложения или по другому в потоке интерфейса. Если операция будет выполняться слишком долго, то интерфейс «зависнет» и будет невозможно со стороны пользователя понять это ошибка или программа все-таки продолжает выполнять свою работу. Чтобы этого избежать длительные вычисления следует выносить в фоновые потоки. В потоке интерфейса во время выполнения фоновой задачи есть возможность показывать прогресс операции.

С помощью класса `AxipyAnyCallableTask` можно превратить любую пользовательскую функцию в задачу, либо, что еще проще, воспользоваться методом `axipy.TaskManager.run_and_get()`:

Список 1: Пример использования.

```
def user_heavy_function(arg1: int, arg2: str):
    print(f"Переданные аргументы: {arg1}, {arg2} \n")
    return 1

spec = ProgressSpecification(
    description="Длительная операция",
    flags=ProgressGuiFlags.CANCELABLE,
    with_handler=False)
result = task_manager.run_and_get(spec, user_heavy_function, "Hi, ", "world!")
assert result == 1
```

Если объем кода в одной функции будет сильно увеличиваться или понадобится запускать новые задачи внутри одной базовой, тогда лучше воспользоваться наследованием от базового класса `AxipyTask` и переопределить метод `run`.

14.2 Представление прогресса операции

Для обмена информацией между выполняемой задачей и элементом, отображающим прогресс, используется класс `AxipyProgressHandler`. Типичный вариант использования выглядит следующим образом:

```
def user_heavy_function(ph: AxipyProgressHandler, count: int):
    # Вначале задаём верхнюю планку изменения прогресса
    ph.set_max_progress(count)
    for i in range(0, count):
        if ph.is_canceled():
            break
        # Тут делаем длительные вычисления
        ph.add_progress(1)
    return ph.progress()

spec = ProgressSpecification(
    description="Длительная операция",
    flags=ProgressGuiFlags.CANCELABLE)
times = 6
real_times = task_manager.run_and_get(spec, user_heavy_function, times)
# выводим колличество раз которое отработал цикл внутри
print(real_times)
```

`ProgressSpecification` - это вспомогательная структура данных, которая используется для первичной инициализации элемента, отображающего прогресс. Вместо `is_canceled` можно использовать `raise_if_canceled` если нужно выйти из нескольких вложенных вызовов функций или циклов.

14.3 Выполнение задач и многопоточность

Важно понимать, что задачи будут выполняться не в потоке интерфейса, поэтому внутри этих задач нельзя отображать никакие графические элементы (`QWidget`). Так же нужно внимательно следить за тем какие ресурсы могут использоваться несколькими потоками и при необходимости использовать различные механизмы синхронизации (мьютексы, локи и т.д.). Общее правило при работе с несколькими потоками следующее: старайтесь чтобы каждая задача содержала в себе все необходимые данные для выполнения. Синхронизацию, если она необходима, следует использовать только в момент получения результата. Это упростит код и сведет к минимум количество ошибок.

Переданные на выполнения задачи ставятся в общую очередь, которая распределяется между физическими ядрами процессора в порядке добавления. Поэтому нет гарантии, что переданная задача выполнится мгновенно, а так же то что группа задач будет выполняться именно в порядке добавления. Если задача предполагает долгое ожидание без интенсивных нагрузок на процессор, то лучше воспользоваться стандартными python потоками. Типичные примеры таких задач это загрузка ресурсов с диска или скачивание файлов по сети.

15

Плагины (Модули)

Плагин - это компонент, добавляющий определенный функционал в программу. Это позволяет программе быть расширяемой.

Данный раздел описывает требования и рекомендации по созданию таких компонентов для ГИС Аксиома.

Чтобы создать плагин, руководствуйтесь следующим:

1. Придумать идею - функционал, решающий какую-то задачу.
2. Создать структуру плагина - файлы и папки.
3. Написать код.
4. Протестировать.
5. Опубликовать.

Совет: Изучайте исходный код плагинов встроенных в Аксиому.

15.1 Структура плагина

Плагин для ГИС Аксиома - это специально оформленный модуль Python с дополнительными файлами.

Рассмотрим структуру минимального плагина с обязательными параметрами и более расширенного:

Минимальный:

```
ru_mycompany_minimal_module  # папка плагина
├── __init__.py  # точка входа
└── manifest.ini  # информация о плагине
```

Расширенный:

```
ru_mycompany_extended_module
├── documentation
│   └── index.html
├── business_logic.py
├── i18n
│   ├── translation_en.qm
│   └── translation_en.ts
├── __init__.py
├── manifest.ini
└── ui
    ├── form.ui
    ├── image.png
    └── logo.png
```

15.1.1 Идентификатор плагина

`ru_mycompany_minimal_module` - папка с плагином, она же - уникальный идентификатор плагина. Так же, как и при создании обычных модулей Python, избегайте конфликтов имен. Делайте имя плагина уникальным. Для этого следуйте простому соглашению именования: - используйте имя вашего веб-сайта или электронной почты, разделив на слова в обратном порядке; - используйте только маленькие латинские буквы и символ нижнего подчеркивания "_", т.е. `[a-z0-9_]`.

Так для плагина mymodule рекомендуемым идентификатором будет: - для веб-сайта axioma-gis.ru - `ru_axioma_gis_mymodule` - для почты andrey@yandex.ru - `ru_yandex_at_andrey_mymodule`

15.1.2 Точка входа

`__init__.py` - точка входа в плагин, так же как и для любого другого модуля Python - является обязательным для системы импорта. Содержит основной код плагина или импортирует другие локальные файлы.

См.также:

Точка входа должна содержать пользовательский класс, наследуемый от класса Plugin.

15.1.3 Информация о плагине

`manifest.ini` - содержит основную информацию, версию, название и прочее. Является ini-файлом с простыми парами ключ=значение.

Примечание: Файл `manifest.ini` должен иметь кодировку UTF-8.

Минимальный пример содержимого:

```
name=Пример плагина
description=Короткий текст с описанием плагина.
```

См.также:

Для более подробного описания формата ini, поддерживаемого ГИС Аксиома, смотрите документацию configparser.

Таблица 1: Таблица значений

Параметр	Обязательно	Описание
name	True	Короткая строка с именем модуля.
description	True	Короткий текст с описанием.
version	False	Строка с версией модуля.
homepage	False	Ссылка на домашнюю страницу модуля.
target	False	Версия Аксиомы, с которой работает модуль; цифры, разделенные точкой.
platforms	False	Список поддерживаемых платформ; через запятую из возможных Windows Linux и Darwin.
icon	False	Имя файла или относительный путь (относительно корневой папки модуля) в формате PNG.

Также могут содержаться другие необязательные параметры:

```
; может содержать коментарии
name=Пример модуля
description=Короткий текст с описанием модуля.
    Может быть многострочным.
; конец обязательных параметров

; необязательные параметры
version=1.0
homepage=https://dev.axioma-gis.ru
platforms=Windows,Darwin
target=3.1.0
author=Developer Name
email=dev@axioma-gis.ru
repository=https://github.com/developer/mymodule
license=New BSD
```

(continues on next page)

(продолжение с предыдущей страницы)

```
; секция локализации
[i18n]
name_en=Plugin example
description_en=Plugin example description.
```

15.1.4 Документация

Документация может быть написана в HTML файлах. ГИС Аксиома откроет документацию в системном веб-браузере. Аксиома ищет документацию в папке с плагином documentation - файлы `index[locale].html`. Пользователь откроет документацию с суффиксом локали, совпадающим с языком системы. При отсутствии совпадения будет открываться файл без суффикса - `index.html`.

15.1.5 Переводы

Можно предусмотреть загрузку плагина на разных языках.

Название и описание самого плагина может быть переведено на другие языки в манифесте в секции `i18n`:

```
; секция локализации
[i18n]
name_en=Plugin example
description_en=Plugin example description.
name_fr=Exemple de plugin
description_fr=Description de l'exemple de plugin.
```

У пользователя отобразится название и описание в случае, если язык системы будет совпадать с суффиксом локали. Иначе отобразится название и описание из основной секции.

Наиболее простой способ создания и сопровождения переводов строк из исходного кода - использование «Qt Linguist».

Примечание: Подробнее о переводе в документации Qt Linguist.

Основные этапы:

1. Отмечаются строки, предназначенные для перевода.
2. Для экспорта строк используется утилита `lupdate`. Она проходит по файлам с исходным кодом и забирает все встречаемые строки, отмеченные для перевода. Результатом является файл с расширением `.ts` - простой структурированный xml файл со строками.
3. `.ts` - файл открывается в «Qt Linguist» и переводится на один или более языков.
4. После завершения перевода отдельных строк файл `.ts` “компилируется” в бинарный файл с расширением `.qm`, который будет загружен Аксиомой.ГИС. Для компиляции используется утилита `lrelease`.

```
lrelease your_plugin.ts
```

5. `.qm` - файлы размещаются в подпапке `i18n` внутри плагина. Они будут загружены вместе с плагином.

15.2 Класс Plugin

Модули пишутся в объектном стиле. Для этого, файл `__init__.py` должен содержать класс, наследуемый от класса `axipy.Plugin`. Тогда ГИС Аксиома при загрузке модуля создаст его экземпляр, а при выгрузке - удалит его.

Вспомогательный класс `axipy.Plugin` содержит свойства и методы, которые нужны для написания плагина. Например: загрузка/сохранение настроек `axipy.Plugin.settings`; получение пути к папке с модулем `axipy.Plugin.plugin_dir`; перевод строк `axipy.Plugin.tr()`; добавление кнопок в интерфейс `axipy.Plugin.create_action` и другие.

Пример модуля `__init__.py`

```
"""
Пример добавления кнопки и подключение действия по нажатию на нее (показ сообщения).
При выгрузке кнопка удаляется из интерфейса.
"""
from axipy import Position, Plugin, Notifications


class ExamplePluginMinimal(Plugin):
    def __init__(self) -> None:
        self._title = self.tr("Минимальный плагин")

        self._action = self.create_action(
            "Пример действия",
            icon=":/icons/share/32px/run3.png",
            on_click=self.show_message,
        )
        position = Position("Примеры модулей", "Минимальный")
        position.add(self._action)
        self._action.action.setToolTip("Всплывающая подсказка")

    def unload(self) -> None:
        self._action.remove()

    def show_message(self) -> None:
        Notifications.push(self._title, "Пример выполнения действия по нажатию кнопки
↪")
```

- При загрузке Аксиома создаст экземпляр модуля и вызовет конструктор класса `axipy.Plugin.__init__()`.
- После создания главного окна, Аксиома вызовет метод `axipy.Plugin.load()`.
- `axipy.Plugin.unload()` - вызывается, когда модуль выгружается.

15.3 Архив

Физически плагин представлен в виде папки с уникальным именем, внутри которой расположены файлы и папки с бизнес-логикой, конфигурациями, документацией, зависимостями, графическими формами и прочим. Для гарантии целостности и удобства распространения готовые плагины помещаются в архив.

Архив использует формат ZIP и имеет следующую структуру:

```
my_plugin_archive_v1.axp
└── ru_axioma_gis_axipy_example_plugin_from_package
    ├── __init__.py
    └── manifest.ini
```

Таким образом архив просто содержит папку с плагином. Имя архива может быть любым и должно заканчиваться на `.axp`, в то время как имя папки с плагином должно быть уникальным.

Для создания архива достаточно запаковать плагин в ZIP любым поддерживаемым архиватором и указать расширение выходного файла как `.axp` вместо стандартного `.zip`.

Архив с плагином распаковывается в пользовательскую директорию при установке. Архив может быть установлен пользователем через интерфейс программы ГИС Аксиома в диалоге «Модули». Плагины, установленные пользователем, могут быть удалены из того же диалога.

Физически плагины устанавливаются в installed_modules в пользовательскую папку. Расположение пользовательской папки зависит от операционной системы:

Операционная система	Пользовательская папка Аксиомы
Windows	`%APPDATA%\ESTI\Axioma.GIS`
Linux	`$HOME/.local/share/ESTI/Axioma.GIS/`
macOS	`$HOME/Library/Application Support/ESTI/Axioma.GIS/`

15.4 Зависимости

Плагины Аксиомы могут использовать сторонние библиотеки Python. Такой плагин во время установки обратится к каталогу пакетов PyPI и установит необходимые зависимости.

В коде плагина пакет импортируется обычным способом:

Список 1: Файл __init__.py

```
import numpy
...
```

Зависимости перечисляются в специальном файле `requirements.txt`. Так плагин с зависимостями может иметь следующую структуру:

```
ru_axioma_gis_axipy_example_plugin_from_package
├── __init__.py
```

(continues on next page)

(продолжение с предыдущей страницы)

```
├── manifest.ini
└── requirements.txt
```

Файл является простым текстовым файлом в кодировке `UTF-8`, в котором построчно перечислены необходимые пакеты. Например:

Список 2: Файл requirements.txt

```
numpy
requests
idna
```

Примечание: В настоящий момент нет возможности указать версию пакета.

15.4.1 Без интернета. Ручная установка пакетов.

Может возникнуть ситуация, когда устанавливаемый плагин имеет внешние зависимости, а доступа к каталогу пакетов PyPI нет. Плагин не сможет успешно установиться.

В этом случае можно скачать все необходимые зависимые пакеты на компьютере, который имеет доступ к интернету и конкретно к каталогу пакетов PyPI, а затем перенести их на целевой компьютер.

Для этого рекомендуется:

1. Установить ГИС Аксиома на компьютер с доступом к сети Интернет (той же версии и платформы).
2. Извлечь из архива с плагином `.axp` файл зависимостей `requirements.txt`.
3. Используя командную строку и интерпретатор Python внутри установленной Аксиомы, выполнить

Список 3: Загрузка необходимых пакетов в папку

```
python -m pip download -r requirements.txt --dest ./module_deps/
```

, где `module_deps` - папка, в которую будут загружены зависимые пакеты.

4. Перенести зависимые пакеты на компьютер без интернета.
5. Аналогично, используя командную строку и интерпретатор Python внутри установленной Аксиомы, выполнить

Список 4: Установка необходимых пакетов из папки

```
python -m pip install -r requirements.txt --user --no-index --find-links ./module_
↪deps/
```

6. Установить архив с плагином `*.axp`.

Определить каталог, куда будут устанавливаться зависимые python пакеты можно следующей командой:

```
python -m site --user-site
```

Результат:

```
C:\Users\user\AppData\Roaming\Python\Python37\site-packages
```

Для ориентировки, примерное расположение в зависимости от системы будет следующее:

Расположение каталога site-packages

- для Windows – «%APPDATA%pythonpython<VERSION>site-packages»
- для Linux – «$HOME/.local/lib/python<VERSION>/site-packages»
- для macOS – «$HOME/Library/Python/<VERSION>/lib/python/site-packages»

Можно использовать для этих целей командный файл.

Пример скрипта для linux (install_requirement), где в качестве параметра можно передать имя файла с набором пакетов. Если параметр опущен, берется файл из текущего каталога requirements.txt:

```
#!/bin/bash

AXIOMA_BASE=/opt/Axioma.GIS

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:${AXIOMA_BASE}/bin

if [ -z "$1" ]; then
 filename="requirements.txt"
else
 filename="$1"
fi

${AXIOMA_BASE}/python/bin/python3 -m pip install --user -r  ${filename}
```

Пример команды для Windows (выполняется в консоли cmd.exe):

```
"C:/Program Files/Axioma v4/bin/python/python.exe" -m pip install --user -r␣
↪requirements.txt
```

Если есть необходимость установить отдельный пакет по имени, то команда будет выглядеть следующим образом:

```
"C:/Program Files/Axioma v4/bin/python/python.exe" -m pip install --user qt5_
↪applications
```

где qt5_applications наименование пакета, который необходимо установить.

15.5 Рекомендации по написанию плагинов

Импорты

При импорте рекомендуется использовать конструкцию, наподобие следующей, где перечислены все необходимые имена:

```
from axipy import (Position, ObserverManager, Notifications, Plugin, Separator,␣
↪ActionButton, ToolButton, Button,
                   Geometry, Style, CoordSystem)
```

Предупреждение: В соответствии с **PEP 8#Imports**, не рекомендуется делать импорт всей библиотеки.

```
from axipy import *
```

При использовании большого количества библиотек, чтобы избежать конфликта имен, рекомендуется использовать конструкцию:

```
import axipy as axp

feature = axp.Feature({'attr_name': 'value'}, geometry=axp.Point(10, 10))
```

16

Создание приложения

В данном разделе рассмотрим создание простейшего приложения, которое можно запускать из оболочки python. При этом пользовательский интерфейс будет строится исходя из предпочтений пользователя. В рамках примера создадим приложение, которое будет открывать файл и показывать его в окне карты. Также вставим стандартное окно управления слоями карты `axipy.LayerControlWidget`.

Примечание: Для использования библиотеки axipy без интерфейса Аксиомы требуется платная лицензия.

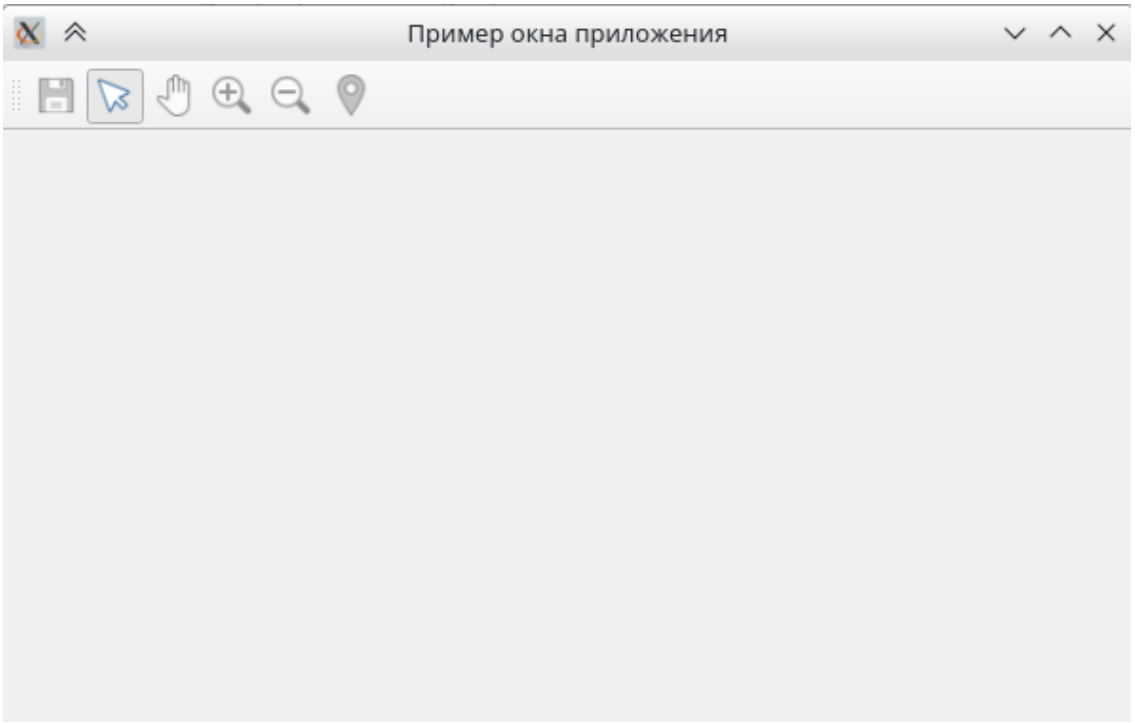
Первым шагом проинициализируем ядро (подробнее см. Initialization, Finalization, and Threads) и определим главное окно нашего будущего приложения. Код будет выглядеть следующим образом:

```
from PySide2.QtWidgets import QMainWindow
from axipy import init_axioma


class MainWindow(QMainWindow):

    def __init__(self):
        super().__init__()
        self.setWindowTitle("Пример окна приложения")


# Инициализация ядра
app = init_axioma()

# Создание и отображение главного окна
main_window = MainWindow()
main_window.resize(1200, 800)
main_window.show()

# Запуск основного цикла программы
app.exec_()
```

После того как мы создали пустое окно добавим панель инструментов со стандартными действиями (полный список действий можно посмотреть `axipy.ActionManager.items()`). Код будет выглядеть следующим образом:

```
from PySide2.QtWidgets import QMainWindow
from axipy import init_axioma, ActionManager


class MainWindow(QMainWindow):

    def __init__(self) -> None:
        super().__init__()
        self.setWindowTitle("Пример окна приложения")
        self._add_buttons()

    def _add_buttons(self) -> None:
        def _add_action(name) -> None:
            # Добавляем системное действие
            act = ActionManager.get(name)
            if act:
                self.toolbar.addAction(act)

        # Создаем панель инструментов
        self.toolbar = self.addToolBar("Панель")
        # Добавляем стандартные кнопки
        _add_action("SaveTables")
        _add_action("Select")
        _add_action("Pan")
        _add_action("ZoomIn")
        _add_action("ZoomOut")
        _add_action("PointDraw")


# Инициализация ядра
app = init_axioma()

# Создание и отображение главного окна
main_window = MainWindow()
main_window.resize(1200, 800)
main_window.show()

# Запуск основного цикла программы
app.exec_()
```

После запуска окно будет выглядеть примерно так:



Далее, нам необходимо добавить пользовательское действие и, как реакция на него, будет открытие и отображение файла в окне карты. Одновременно с этим создается окно управления слоями.

Результирующий код будет выглядеть следующим образом:

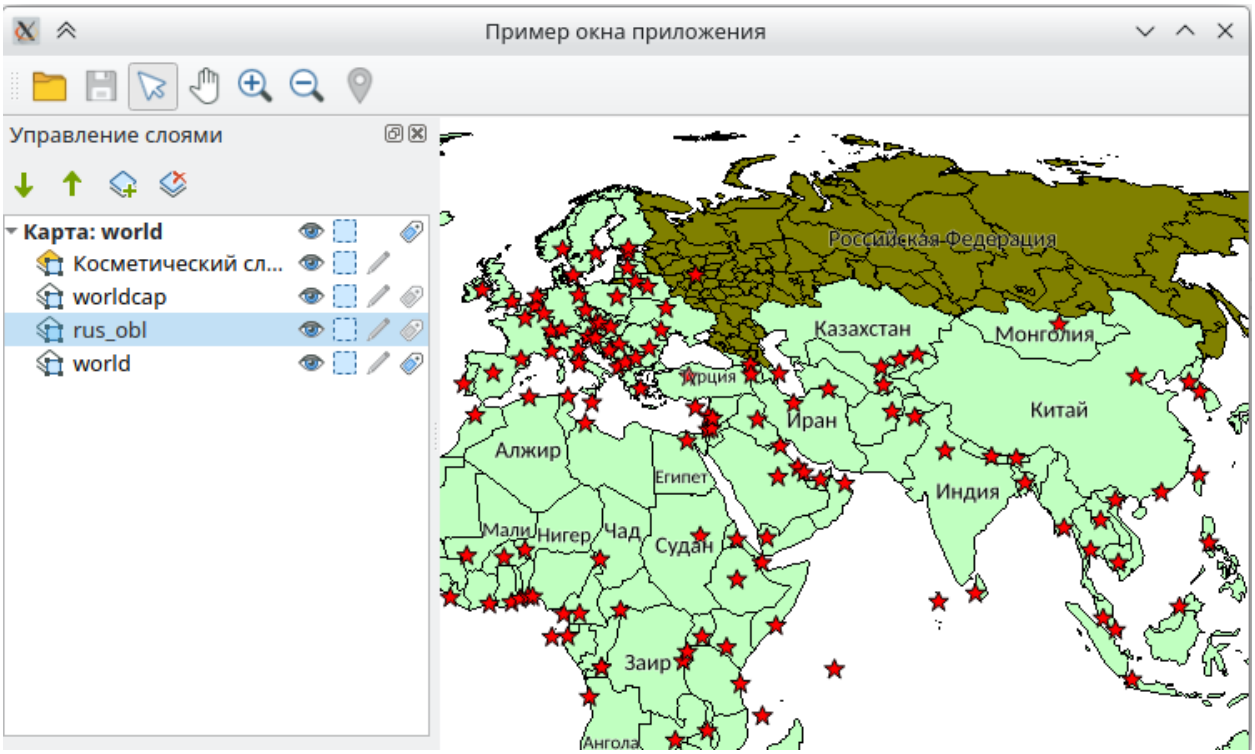
```
from PySide2.QtCore import Qt
from PySide2.QtWidgets import QMainWindow, QFileDialog, QDockWidget, QAction
from axipy import (
    init_axioma, provider_manager, Layer, view_manager, Map, LayerControlWidget,␣
↪ActionManager
)


class MainWindow(QMainWindow):

    def __init__(self) -> None:
        super().__init__()
        self.setWindowTitle("Пример окна приложения")
        self._add_buttons()

    # Открытие файла
    def __open_file(self) -> None:
        file_name, _ = QFileDialog.getOpenFileName(self, "Открыть", filter="MapInfo␣
↪TAB (*.tab)")
        if file_name:
            table = provider_manager.openfile(file_name)
            # создаем слой
            layer = Layer.create(table)
            # если карты нет, то создаем ее
            if not view_manager.active:
                map_ = Map([layer])
                mapview = view_manager.create_mapview(map_)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                self._create_layer_control_dock()
                view_manager.activate(mapview)
                self.setCentralWidget(mapview.widget)
            # если есть, добавляем слой в окно карты
            else:
                view_manager.active.map.layers.append(layer)

    # Добавление по левой стороне окно управления слоями карты
    def _create_layer_control_dock(self) -> None:
        self._layer_control_w = LayerControlWidget()
        dock_layer_control = QDockWidget(self._layer_control_w.widget.windowTitle(),␣
↪self)
        dock_layer_control.setWidget(self._layer_control_w.widget)
        self.addDockWidget(Qt.LeftDockWidgetArea, dock_layer_control)

    # Добавление кнопок на панель инструментов
    def _add_buttons(self) -> None:
        # Создаем панель инструментов
        self.toolbar = self.addToolBar("Панель")

        # Пользовательское действие открытия файла
        self.action_open = QAction(ActionManager.icon_by_name("open"), "Открыть файл")
        self.action_open.triggered.connect(self.__open_file)
        self.toolbar.addAction(self.action_open)

        def _add_action(name) -> None:
            # Добавляем системное действие
            act = ActionManager.get(name)
            if act:
                self.toolbar.addAction(act)

        _add_action("SaveTables")
        _add_action("Select")
        _add_action("Pan")
        _add_action("ZoomIn")
        _add_action("ZoomOut")
        _add_action("PointDraw")


# Инициализация ядра
app = init_axioma()

# Создание и отображение главного окна
main_window = MainWindow()
main_window.resize(1200, 800)
main_window.show()

# Запуск основного цикла программы
app.exec_()
```

Результат работы:



Следующим этапом попробуем усложнить пример и сделать приложение, которое содержит помимо окна карты и окна с деревом слоев и другие. Перечислим их:

- LayerControlWidget - Виджет управления слоями карты.
- ViewManagerWidget - Виджет списка окон.
- DataManagerWidget - Виджет управления списком открытых данных.
- PythonConsoleWidget - Виджет ввода команд python.
- NotificationWidget - Виджет уведомлений.

```
from PySide2.QtCore import Qt, Slot
from PySide2.QtWidgets import QMainWindow, QDockWidget, QStackedWidget

import os
from axipy import (Notifications, LayerControlWidget, DataManagerWidget,␣
↪NotificationWidget, ViewManagerWidget,
                   PythonConsoleWidget, view_manager, init_axioma, provider_manager,␣
↪Layer, Map, View, ActionManager)


class MainWindow(QMainWindow):

    def __init__(self):
        super().__init__()
        self.setWindowTitle("Пример окна приложения")

        self._stacked_widget = QStackedWidget()
        view_manager.active_changed.connect(self.set_stacked_widget)

        self.setCentralWidget(self._stacked_widget)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        self.__initButtons()

    def _add_action(self, id):
        act = ActionManager[id]
        if act:
            self.toolbar.addAction(act)
            self.menu.addAction(act)

    def __initButtons(self):
        self.toolbar = self.addToolBar('Панель')
        self.menu = self.menuBar().addMenu('Меню')
        self._add_action('Select')
        self._add_action('SelectInRect')
        self._add_action('Pan')
        self._add_action('ZoomIn')
        self._add_action('ZoomOut')
        self.toolbar.addSeparator()
        self._add_action('PointDraw')
        self._add_action('RectangleDraw')
        self._add_action('SymbolStyle')
        ActionManager.activate("Pan")

    def _create_maps_dock(self):
        self._layer_control_w = LayerControlWidget()
        self._layer_control_w.mapview_activated.connect(self.set_center_widget)
        dock_layer_control = QDockWidget(self._layer_control_w.widget.windowTitle(),␣
↪main_window)
        dock_layer_control.setWidget(self._layer_control_w.widget)
        self.addDockWidget(Qt.LeftDockWidgetArea, dock_layer_control)

    def _create_data_manager_dock(self):
        data_manager_w = DataManagerWidget()
        dock_catalog = QDockWidget(data_manager_w.widget.windowTitle())
        dock_catalog.setWidget(data_manager_w.widget)

        def notify_selection():
            """
            Отслеживание выборки в списке открытых данных.
            """
            objects = data_manager_w.objects
            if len(objects) > 0:
                names = ', '.join(obj.name for obj in objects)
                Notifications.push("Выбрано", names, Notifications.Warning)

        data_manager_w.selection_changed.connect(notify_selection)
        self.addDockWidget(Qt.LeftDockWidgetArea, dock_catalog)

    def _create_notification_dock(self):
        notification_w = NotificationWidget()
        notification_w.widget.setFixedHeight(150)
        dock_notification = QDockWidget(notification_w.widget.windowTitle())
        dock_notification.setWidget(notification_w.widget)
        self.addDockWidget(Qt.BottomDockWidgetArea, dock_notification)

    def _create_python_dock(self):
        notification_w = PythonConsoleWidget()
```

(continues on next page)

(продолжение с предыдущей страницы)

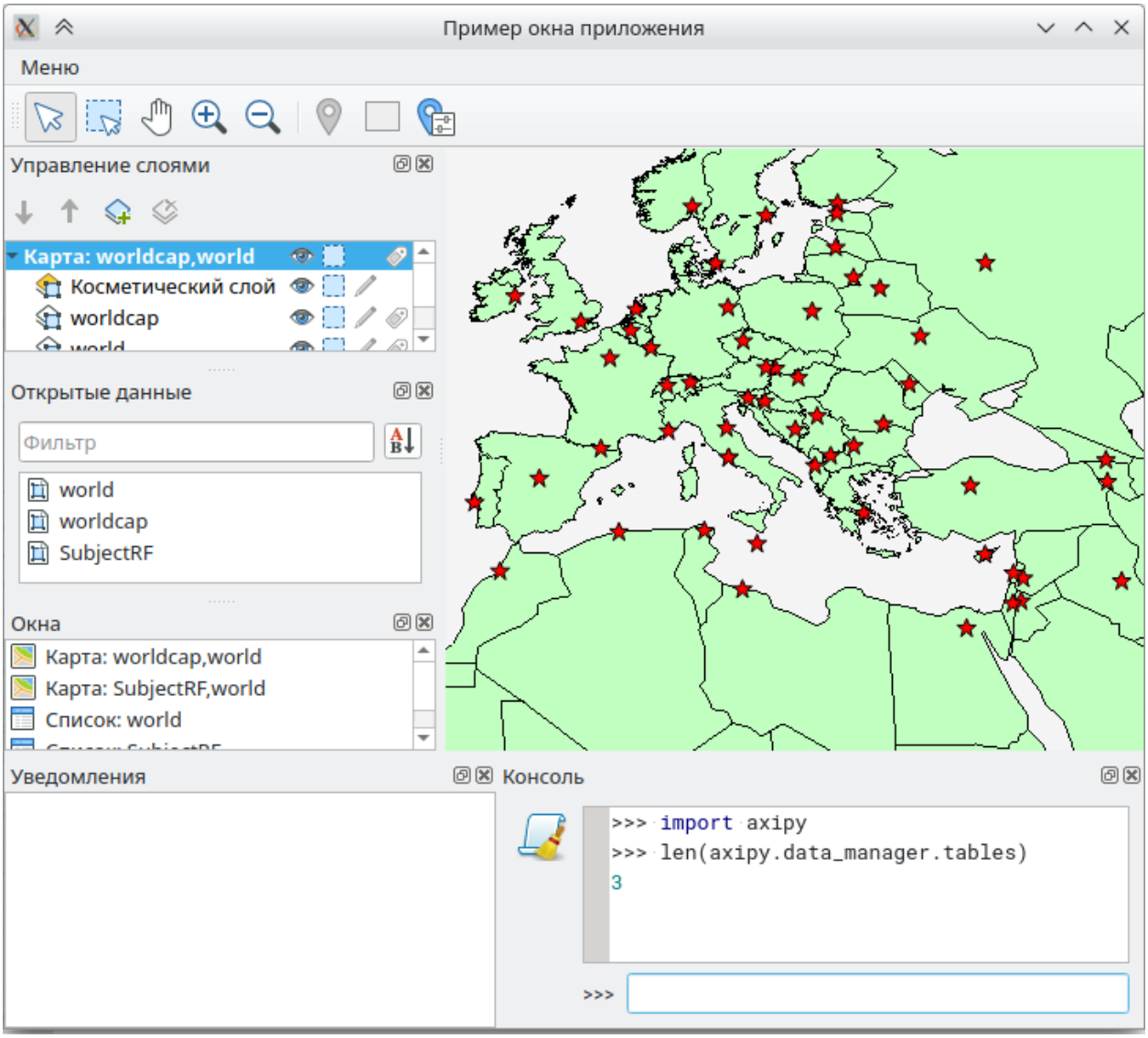
```
        notification_w.widget.setFixedHeight(150)
        dock_notification = QDockWidget(notification_w.widget.windowTitle())
        dock_notification.setWidget(notification_w.widget)
        self.addDockWidget(Qt.BottomDockWidgetArea, dock_notification)

    def _create_view_manager_dock(self):
        self._view_manager_w = ViewManagerWidget()
        self._view_manager_w.view_changed.connect(self.set_center_widget)
        dock_widget = QDockWidget(self._view_manager_w.widget.windowTitle(), main_
↪window)
        dock_widget.setWidget(self._view_manager_w.widget)
        self.addDockWidget(Qt.LeftDockWidgetArea, dock_widget)

    def init_gui(self):
        # Окно управления слоями
        self._create_maps_dock()
        # Открытые источники
        self._create_data_manager_dock()
        # Окно уведомлений
        self._create_notification_dock()
        # Окно консоли python
        self._create_python_dock()
        # Список окон
        self._create_view_manager_dock()

    @Slot(View)
    def set_center_widget(self, inp_view: View):
        """
        Устанавливает активное окно.
        """
        view_manager.activate(inp_view)

    @Slot()
    def set_stacked_widget(self):
        w = view_manager.active.widget
        i = self._stacked_widget.indexOf(w)
        if i == -1:
            self._stacked_widget.insertWidget(0, w)
        else:
            self._stacked_widget.setCurrentIndex(i)


# Инициализция ядра
app = init_axioma()

# Создаем главное окно приложения
main_window = MainWindow()
main_window.init_gui()


# Открываем таблицы
def path_to_file(file_name: str) -> str:
    return os.path.join(os.path.dirname(__file__), "tab", file_name)


table_world = provider_manager.openfile(path_to_file("world.tab"))
```

(continues on next page)

(продолжение с предыдущей страницы)

```
table_world_cap = provider_manager.openfile(path_to_file("worldcap.tab"))
table_rf = provider_manager.openfile(path_to_file("SubjectRF.TAB"))

# Создаем слои
layer_world = Layer.create(table_world)
layer_world_cap = Layer.create(table_world_cap)
layer_rf = Layer.create(table_rf)

# Создаем окно карты
map_world_all = Map([layer_world_cap, layer_world])
mapview = view_manager.create_mapview(map_world_all)

# Еще одна карта
map_world = Map([layer_rf, layer_world])
view_manager.create_mapview(map_world)

# Таблицы просмотра
view_manager.create_tableview(table_world)
view_manager.create_tableview(table_rf)

# Отчет
#view_manager.create_reportview()

# Первое окно как основное
main_window.set_center_widget(mapview)

# Левый верхний угол окна
x, y = 200, 10
# Ширина и высота окна
width, height = 1200, 800
main_window.resize(width, height)
main_window.move(x, y)

# Показываем главное окно
main_window.show()
app.exec_()
```

Результат работы:


Пример окна приложения
Меню
Управление слоями
Карта: worldcap,world
Косметический слой
worldcap
Открытые данные
Фильтр
world
worldcap
SubjectRF
Окна
Карта: worldcap,world
Карта: SubjectRF,world
Список: world
Уведомления
Консоль
>>> import axipy
>>> len(axipy.data_manager.tables)
3
>>>

17

axipy - Основной пакет библиотеки для взаимодействия с ГИС Аксиома.

Основной пакет API для взаимодействия с ГИС Аксиома.

Предоставляет доступ к Аксиоме.ГИС через набор модулей, подмодулей, классов и функций.

17.1 Инициализация Аксиомы

class axipy.**AxiomaInitLogLevel**

Уровень логирования Аксиомы при запуске из python.

Атрибуты:

`maximum`	Максимальный
`high`	Высокий
`medium`	Средний
`minimum`	Минимальный
`disabled`	Логирование отключено

axipy.**init_axioma**(*, log_level: AxiomaInitLogLevel = AxiomaInitLogLevel.medium, language: AxiomaLanguage = AxiomaLanguage.ru, load_python_plugins: bool = False) → QApplication

Инициализирует ядро ГИС Аксиома.

Параметры

- **log_level** – Уровень логирования.
- **language** – Язык Аксиомы.
- **load_python_plugins** – Загружать плагины.

Результат

Приложение Qt5 с очередью событий (event-loop).

Пример:

```
from axipy import init_axioma, mainwindow, AxiomaInitLogLevel

app = init_axioma(log_level=AxiomaInitLogLevel.disabled)
mainwindow.show()
app.exec_()  # запускает обработку очереди событий
```

17.2 Plugin - Плагин ГИС Аксиома

class axipy.**Plugin**

Вспомогательный класс для создания плагинов.

Свойства:

plugin_dir	Возвращает путь к папке плагина.
settings	Настройки плагина.

Методы:

create_action(title, on_click[, icon, ...])	Создает кнопку с действием.
create_tool(title, on_click[, icon, ...])	Создает кнопку с инструментом.
get_plugin_data_dir()	Возвращает каталог, в котором находятся изменяемые данные плагина.
load()	Переопределите этот метод для задания логики загрузки плагина.
tr(text)	Ищет перевод строки.
unload()	Переопределите этот метод для очистки ресурсов при выгрузке плагина.

create_action(title: str, on_click: Callable[[], Any], icon: Union[str, QIcon] = '', enable_on: Optional[Observer] = None, tooltip: Optional[str] = None, doc_file: Optional[str] = None) → ActionButton

Создает кнопку с действием.

Параметры

- **title** – Текст.
- **on_click** – Действие на нажатие.
- **icon** – Иконка. Может быть путем к файлу или адресом ресурса.
- **enable_on** – Идентификатор наблюдателя для определения доступности кнопки.
- **tooltip** – Строка с дополнительной короткой информацией по данному действию.
- **doc_file** – Относительная ссылка на файл документации. Расположение рассматривается по отношению к каталогу documentation.

Результат
Кнопка с действием.

Примечание: То же, что и ActionButton, но дополнительно делает идентификатор кнопки уникальным для данного плагина.

---

create_tool(title: str, on_click: Union[Callable[[], MapTool], MapTool], icon: Union[str, QIcon] = '', enable_on: Optional[Union[str, Observer]] = None, tooltip: Optional[str] = None, doc_file: Optional[str] = None) → ToolButton

Создает кнопку с инструментом.

Параметры

- **title** – Текст.
- **on_click** – Класс инструмента.
- **icon** – Иконка. Может быть путем к файлу или адресом ресурса.
- **enable_on** – Идентификатор наблюдателя для определения доступности кнопки.
- **tooltip** – Строка с дополнительной короткой информацией по данному действию.
- **doc_file** – Относительная ссылка на файл документации. Расположение рассматривается по отношению к каталогу documentation.

Результат
Кнопка с инструментом.

Примечание: То же, что и ToolButton, но дополнительно делает идентификатор кнопки уникальным для данного плагина.

---

get_plugin_data_dir() → Path

Возвращает каталог, в котором находятся изменяемые данные плагина.

load()

Переопределите этот метод для задания логики загрузки плагина.

property plugin_dir: Path

Возвращает путь к папке плагина.

property settings: QSettings

Настройки плагина.

Позволяет сохранять и загружать параметры.

См.также:

Подробнее в документации на класс PySide2.QtCore.QSettings.

tr(text: str) → str

Ищет перевод строки. Производит поиск строки в загруженных файлах перевода.

Параметры
text – Строка для перевода.

Результат
Перевод стоки, если строка найдена. Иначе - сама переданная строка.

Пример:

```
button_name = self.tr("My button")
```

unload()

Переопределите этот метод для очистки ресурсов при выгрузке плагина.

17.3 Настройки ГИС Аксиома

class axipy.**CurrentSettings**

Текущие настройки Аксиомы. Класс является статическим словарем (collections.abc.MutableMapping). Поддерживает обращение по индексу. Для получения настроек по умолчанию, используется класс axipy.DefaultSettings.

Список 1: Пример работы с настройками.

```
from axipy import DefaultSettings, CurrentSettings

# Получение настройки по умолчанию
print(DefaultSettings.ShowSplashScreen)
'''
>>> True
'''
# Назначение текущей настройки
CurrentSettings.ShowSplashScreen = False
# Получение текущей настройки
print(CurrentSettings.ShowSplashScreen)
'''
>>> False
'''
# Сброс настройки обратно к значению по умолчанию
CurrentSettings.ShowSplashScreen = DefaultSettings.ShowSplashScreen
print(CurrentSettings.ShowSplashScreen)
'''
>>> True
'''
```

Классовые методы:

get(key[, default_value])	Возвращает значение по ключу.
items()	Возвращает список кортежей ключ-значение, где ключи - это атрибуты класса, а значения - это значения настроек.
keys()	Возвращает список ключей, где ключи это атрибуты класса.
reset()	Сброс значений всех текущих настроек на значения по умолчанию.
values()	Возвращает список значений, где значения это значения настроек.

Атрибуты:

BrushCatalog	Каталог со стилями заливок
CreateTabAfterOpen	Создавать TAB при открытии
DefaultPathCache	Каталог с кэшированными данными
DistancePrecision	Точность по умолчанию для расстояний и площадей
DrawCoordSysBounds	Отображать границы мира
EditNodeColor	Узлы при редактировании - цвет
EditNodeSize	Узлы при редактировании - размер
EnableSmartTabs	Умное переключение вкладок
Language	Язык Аксиомы
LastNameFilter	Последний использованный фильтр файлов
LastOpenPath	Последний каталог откуда открывались данные
LastPathWorkspace	Последний каталог к рабочему набору
LastSavePath	Последний пусть сохранения
LoadLastWorkspace	Загружать при старте последнее рабочее пространство
MeshSizeLayout	Размер ячейки
MeshSizeLegend	Размер ячейки
NearlyGeometriesTopology	Перемещать узлы соседних объектов при редактировании
NodesUpdateMode	Объединять историю изменения узлов в режиме Форма
PenCatalog	Каталог со стилями линий
PreserveScaleMap	Сохранять масштаб при изменении размеров окна
RenameDataObjectFromTab	Переименовывать открытый объект по имени TAB файла
RulerColorLine	Линейка - цвет линии
SaveAsToOriginalFileFolder	Сохранять копию в каталог с исходным файлом
SelectByInformationTool	Инструмент «Информация» выбирает объект
SensitiveMouse	Чувствительность мыши в пикселях
ShowDegreeTypeNumeric	Отображать градусы в формате Десятичное значение

continues on next page

Таблица 1 – продолжение с предыдущей страницы

ShowDrawingToolTip	Показывать данные при рисовании
ShowMapScaleBar	Показывать масштабную линейку
ShowMeshLayout	Отображать сетку привязки
ShowMeshLegend	Отображать сетку привязки
ShowScrollOnMapView	Показывать полосы прокрутки
ShowSplashScreen	Отображать экран загрузки
SilentCloseWidget	Подтверждать закрытие несохраненных данных
SnapSensitiveRadius	Привязка узлов - размер
SnapToMeshLayout	Привязывать элементы отчета к сетке
SnapToMeshLegend	Привязывать к сетке
SymbolCatalog	Каталог со стилями точек
UseAntialiasing	Использовать сглаживание при отрисовке
UseLastSelectedFilter	Запоминать последний фильтр в диалоге открытия файлов
UseNativeFileDialog	Пользоваться системными файловыми диалогами
UserDataPaths	Список пользовательских каталогов с названиями

BrushCatalog: Path

Каталог со стилями заливок

CreateTabAfterOpen: bool

Создавать TAB при открытии

DefaultPathCache: Path

Каталог с кэшированными данными

DistancePrecision: int

Точность по умолчанию для расстояний и площадей

DrawCoordSysBounds: bool

Отображать границы мира

EditNodeColor: QColor

Узлы при редактировании - цвет

EditNodeSize: int

Узлы при редактировании - размер

EnableSmartTabs: bool

Умное переключение вкладок

Language: AxiomaLanguage

Язык Аксиомы

LastNameFilter: str

Последний использованный фильтр файлов

LastOpenPath: Path

Последний каталог откуда открывались данные

LastPathWorkspace: `Path`
Последний каталог к рабочему набору

LastSavePath: `Path`
Последний пусть сохранения

LoadLastWorkspace: `bool`
Загружать при старте последнее рабочее пространство

MeshSizeLayout: `float`
Размер ячейки

MeshSizeLegend: `float`
Размер ячейки

NearlyGeometriesTopology: `bool`
Перемещать узлы соседних объектов при редактировании

NodesUpdateMode: `bool`
Объединять историю изменения узлов в режиме Форма

PenCatalog: `Path`
Каталог со стилями линий

PreserveScaleMap: `bool`
Сохранять масштаб при изменении размеров окна

RenameDataObjectFromTab: `bool`
Переименовывать открытый объект по имени TAB файла

RulerColorLine: `QColor`
Линейка - цвет линии

SaveAsToOriginalFileFolder: `bool`
Сохранять копию в каталог с исходным файлом

SelectByInformationTool: `bool`
Инструмент «Информация» выбирает объект

SensitiveMouse: `int`
Чувствительность мыши в пикселях

ShowDegreeTypeNumeric: `bool`
Отображать градусы в формате Десятичное значение

ShowDrawingToolTip: `bool`
Показывать данные при рисовании

ShowMapScaleBar: `bool`
Показывать масштабную линейку

ShowMeshLayout: `bool`
Отображать сетку привязки

ShowMeshLegend: `bool`
Отображать сетку привязки

ShowScrollOnMapView: bool
Показывать полосы прокрутки

ShowSplashScreen: bool
Отображать экран загрузки

SilentCloseWidget: bool
Подтверждать закрытие несохраненных данных

SnapSensitiveRadius: int
Привязка узлов - размер

SnapToMeshLayout: bool
Привязывать элементы отчета к сетке

SnapToMeshLegend: bool
Привязывать к сетке

SymbolCatalog: Path
Каталог со стилями точек

UseAntialiasing: bool
Использовать сглаживание при отрисовке

UseLastSelectedFilter: bool
Запоминать последний фильтр в диалоге открытия файлов

UseNativeFileDialog: bool
Пользоваться системными файловыми диалогами

UserDataPaths: Dict[str, Path]
Список пользовательских каталогов с названиями

classmethod get(key: str, default_value: Optional[Any] = None)
Возвращает значение по ключу.

classmethod items() → List[Tuple[str, Any]]
Возвращает список кортежей ключ-значение, где ключи - это атрибуты класса, а значения - это значения настроек.

classmethod keys() → List[str]
Возвращает список ключей, где ключи это атрибуты класса.

static reset()
Сброс значений всех текущих настроек на значения по умолчанию.

classmethod values() → List[Any]
Возвращает список значений, где значения это значения настроек.

class axipy.**DefaultSettings**
Настройки Аксиомы по умолчанию. Класс является статическим словарем, доступным только для чтения (`collections.abc.Mapping`). Методы и атрибуты класса такие же как у класса `axipy.CurrentSettings`, но доступны только для чтения.

class axipy.**AxiomaLanguage**

Язык Аксиомы.

Атрибуты:

en	Английский язык
ru	Русский язык

17.4 Вспомогательные функции

axipy.**open_file_dialog**(filter_arg: Optional[str] = None) → Optional[Path]

Открывает диалог выбора файла. Возвращает путь к выбранному файлу. Если нет главного окна Аксиомы, спрашивает путь к файлу в консоли.

Параметры

filter_arg – Типы файлов. Например: `'MapInfo Tab (*.tab);;Таблицы Excel (*.xls *.xlsx)'`.

axipy.**execfile**(path: Union[str, Path])

Выполняет скрипт на языке python из файла.

Параметры

path – Путь к исполняемому файлу.

axipy.**get_dependencies_folder**() → Path

Возвращает папку, для установки зависимых пакетов.

18

axipy.app - Модуль приложения.

Модуль приложения.

Данный модуль является основным модулем приложения.

18.1 MainWindow - Главное окно

class axipy.**MainWindow**

Главное окно ГИС Аксиома.

Примечание: Используйте готовый объект `axipy.mainwindow`.

Классовые методы:

show()	Создает и показывает главное окно программы.

Свойства:

catalog	Хранилище объектов приложения.
geometry	Положение главного окна.
is_valid	Корректность состояния главного окна.

Методы:

add(view)	Добавляет окно просмотра данных.
add_dock_widget(dock_widget, area[, icon])	Добавляет панель в главное окно приложения.
add_layer_current_map(layer)	Добавляет слой в текущей карте.
add_layer_interactive(layer)	Добавляет слой с запросом на помещение на текущую карту или в новую.
add_layer_new_map(layer)	Открывает слой в новой карте.
qt_object()	Возвращает Qt5 объект окна.
remove_dock_widget(dock)	Удаляет существующую панель у главного окна приложения.
show_html_url(url, caption)	Показывает окно для локального файла html или если это web страница, запускает браузер по ассоциации.

add(view: View) → QMdiSubWindow

Добавляет окно просмотра данных.

Параметры
view – окно просмотра данных.

Примечание: При создании окон просмотра данных axipy.ViewManager.create_mapview() или axipy.ViewManager.create_tableview() они автоматически добавляются в главное окно программы.

---

add_dock_widget(dock_widget: QDockWidget, area: DockWidgetArea, icon: Optional[QIcon] = None) → bool

Добавляет панель в главное окно приложения. При успешном добавлении возвращает True. Если же данная панель уже присутствует, то команда игнорируется и возвращается False. Элементы управления, которые требуется разместить на панели, создаются в дополнительном окне, а уже это окно, в свою очередь, устанавливается для панели (см. пример ниже).

Параметры

- **dock_widget** – Пользовательская созданная панель.
- **area** – Расположение.
- **icon** – Иконка для отображения в списке всех доступных панелей.

Пример:

```
from PySide2.QtWidgets import QDockWidget, QWidget, QPushButton
from PySide2.QtCore import Qt

dock = QDockWidget('Заголовок')
widget = QWidget()
layout = QVBoxLayout()
button = QPushButton("Кнопка")
button.clicked.connect(lambda: print('Реакция на кнопку'))
layout.addWidget(button)
layout.addStretch()
widget.setLayout(layout)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
dock.setWidget(widget)
app.mainwindow.add_dock_widget(dock, Qt.RightDockWidgetArea, QIcon('filename.
↪png'))
```

add_layer_current_map(layer: Layer) → MapView

Добавляет слой в текущей карте.

add_layer_interactive(layer: Layer) → MapView

Добавляет слой с запросом на помещение на текущую карту или в новую.

add_layer_new_map(layer: Layer) → MapView

Открывает слой в новой карте.

property catalog: DataManager

Хранилище объектов приложения.

Это то же хранилище, которое отображается в панели «Открытые данные».

Примечание: При открытии объектов данных `axipy.ProviderManager.openfile()` они автоматически попадают в каталог.

---

property geometry: QRect

Положение главного окна.

property is_valid: bool

Корректность состояния главного окна.

qt_object() → QMainWindow

Возвращает Qt5 объект окна.

remove_dock_widget(dock: QDockWidget) → bool

Удаляет существующую панель у главного окна приложения.

static show() → MainWindow

Создает и показывает главное окно программы.

show_html_url(url: QUrl, caption: Optional[str])

Показывает окно для локального файла html или если это web страница, запускает браузер по ассоциации.

Параметры

- **url** – Ссылка на файл html или адрес страницы.
- **caption** – Заголовок окна.

class axipy.**DockWidgetArea**

Расположение панели

Атрибуты:

Bottom	Снизу
Left	Слева
Right	Справа
Top	Сверху

18.2 Version - Информация о версии

class axipy.**Version**

Информация о версии ГИС Аксиома.

Пример использования:

```
from axipy import Version

# В данном случае, если текущая версия выше 4.3 (например 5.1), то возвращается -
↪1.
print('Версия:', Version.string(), Version.compare(4,3))
```

Классовые методы:

compare(major[, minor, patch])	Сравнивает переданное значение с текущей версией Аксиомы.
number()	Возвращает версию в виде одного числа, в котором каждый сегмент располагается в отдельном байте.
qtFormat()	Возвращает версию в Qt формате.
qt_format()	Возвращает версию в Qt формате.
segments()	Возвращает кортеж чисел - сегменты версии: (major, minor, patch).
string()	Возвращает версию в виде строки.

static compare(major: int, minor: Optional[int] = 0, patch: Optional[int] = 0) → int

Сравнивает переданное значение с текущей версией Аксиомы.

Параметры

- **major** – Основная версия
- **minor** – Минорная версия
- **patch** – Исправления

Возвращает -1, если переданная меньше, 1 если больше и 0 если равны

static number() → int

Возвращает версию в виде одного числа, в котором каждый сегмент располагается в отдельном байте.

static qtFormat() → QVersionNumber

Возвращает версию в Qt формате. :meta private:

static qt_format() → QVersionNumber

Возвращает версию в Qt формате.

static segments() → tuple

Возвращает кортеж чисел - сегменты версии: (major, minor, patch).

static string() → str

Возвращает версию в виде строки.

19

axipy.cs - Модуль систем координат.

Модуль систем координат.

В данном модуле содержатся классы и методы, предназначенные для удобной работы с координатными системами.

19.1 CoordSystem - Система Координат (СК)

class axipy.**CoordSystem**

Система координат (СК). СК описывает каким образом реальные объекты на земной поверхности могут быть представлены в виде двумерной проекции. Выбор СК для представления данных зависит от конкретных исходных условий по представлению исходных данных.

Примечание: Проверка на идентичность параметров двух СК производится простым сравнением.

Примечание: Для получения текстового представления можно воспользоваться функцией `str`.

Поддерживается создание СК посредством следующих вариантов:

- Из строки MapInfo PRJ `from_prj()`
- Из строки PROJ `from_proj()`
- Из строки WKT `from_wkt()`
- Из значения EPSG `from_epsg()`
- План/Схему с указанием единиц измерения и охвата `from_units()`

Список 1: Пример создания СК разного типа.

```
cs_epsg = CoordSystem.from_epsg(4326)
cs_prj = CoordSystem.from_prj('1, 104')
cs_proj = CoordSystem.from_proj('+proj=longlat +ellps=WGS84 +no_defs')
cs_wkt = CoordSystem.from_wkt('GEOGCS["GCS_WGS_1984",DATUM["D_WGS_1984",SPHEROID[
↪"WGS_1984",6378137,298.257223563]],PRIMEM["Greenwich",0],UNIT["Degree",0.
↪0174532925199433295]]')
# Создание из строки с указанием вида формата
crs1 = CoordSystem.from_string('epsg:4326')
crs2 = CoordSystem.from_string('prj:1,104')
```

Список 2: Проверка на идентичность координатных систем производится простым сравнением.

```
cs1 = CoordSystem.from_prj("1, 104")
cs2 = CoordSystem.from_prj("1, 104")
if cs1 == cs2:
    print("Координатные системы эквивалентны.")
'''
>>> Координатные системы эквивалентны.
'''
```

Классовые методы:

current()	Текущая установленная система координат (СК).
from_epsg(code)	Создает координатную систему по коду EPSG.
from_prj(prj)	Создает координатную систему из строки MapBasic.
from_proj(proj)	Создает координатную систему из строки proj.
from_string(string)	Создает систему координат из строки.
from_units(unit[, rect])	Создает декартову систему координат.
from_wkt(wkt)	Создает координатную систему из строки WKT.
set_current(coordsystem)	Устанавливает новую текущую систему координат

Свойства:

epsg	Значение EPSG если существует для данной системы координат, иначе None.
inv_flattening	Полярное сжатие.
lat_lon	Является ли данная СК широтой/долготой.
name	Наименование системы координат.
non_earth	Является ли данная СК декартовой.
prj	Строка prj формата MapBasic или пустая строка, если аналога не найдено.
proj	Строка PROJ или пустая строка, если аналога не найдено.
rect	Максимально допустимый охват.
semi_major	Большая полуось.
semi_minor	Малая полуось.
title	Наименование системы координат.
unit	Единицы измерения.
wkt	Строка WKT или пустая строка, если аналога не найдено.

Методы:

convert_from_degree(value)	Переводит из градусов в единицы измерения системы координат.
convert_to_degree(value)	Переводит из единиц измерения системы координат в градусы.
to_string()	Текстовое представление в виде <тип>:<строка>

convert_from_degree(value: Union[Pnt, Rect, QPointF, List[QPointF], QRectF]) → Union[Pnt, List[Pnt], Rect]

Переводит из градусов в единицы измерения системы координат.

convert_to_degree(value: Union[Pnt, Rect, QPointF, List[QPointF], QRectF]) → Union[Pnt, List[Pnt], Rect]

Переводит из единиц измерения системы координат в градусы.

Список 3: Пример.

```
csMercator = CoordSystem.from_prj("10, 104, 7, 0")
p_out = csMercator.convert_to_degree((1000000, 1000000))
print(p_out)
'''
>>> (8.983152841195214 9.005882635078796)
'''
```

classmethod current() → CoordSystem

Текущая установленная система координат (СК). Данная СК используется как значение по умолчанию, когда она не определена. Например, в диалоге создания новой таблицы.

property epsg: Optional[int]

Значение EPSG если существует для данной системы координат, иначе None.

classmethod from_epsg(code: int) → CoordSystem

Создает координатную систему по коду EPSG.

См.также:

Подробнее см. EPSG

Параметры

code – Стандартное значение EPSG.

classmethod from_prj(prj: str) → CoordSystem

Создает координатную систему из строки MapBasic.

Параметры

prj – Строка MapBasic. Допустима короткая нотация.

Список 4: Пример.

```
csMercator = CoordSystem.from_prj('10, 104, 7, 0')
csLatLon = CoordSystem.from_prj('Earth Projection 1, 104')
csMercator = CoordSystem.from_prj('NonEarth 0, \'m\'')
```

classmethod from_proj(proj: str) → CoordSystem

Создает координатную систему из строки proj.

См.также:

Подробнее см. PROJ

Параметры

proj – Строка proj.

classmethod from_string(string: str) → CoordSystem

Создает систему координат из строки. Строка состоит из двух частей: префикса и строки представления СК. Возможные значения префиксов: «proj», «wkt», «epsg», «prj».

Параметры

string – Строка.

classmethod from_units(unit: LinearUnit, rect: Optional[Union[Rect, QRectF]] = Rect(-10000, -10000, 10000, 10000)) → CoordSystem

Создает декартову систему координат.

Параметры

- **unit** – Единицы измерения системы координат.
- **rect** – Охват системы координат.

Список 5: Пример.

```
ne = CoordSystem.from_units(Unit.km, Rect(-100, -100, 100, 100))
```

classmethod from_wkt(wkt: str) → CoordSystem

Создает координатную систему из строки WKT.

См.также:

Подробнее см.
WKT

Параметры
wkt – Строка WKT.

property inv_flattening: float

Полярное сжатие.

property lat_lon: bool

Является ли данная СК широтой/долготой.

property name: str

Наименование системы координат.

property non_earth: bool

Является ли данная СК декартовой.

property prj: str

Строка prj формата MapBasic или пустая строка, если аналога не найдено.

property proj: str

Строка PROJ или пустая строка, если аналога не найдено.

property rect: Rect

Максимально допустимый охват.

property semi_major: float

Большая полуось.

property semi_minor: float

Малая полуось.

classmethod set_current(coordsystem: CoordSystem)

Устанавливает новую текущую систему координат

Параметры
coordsystem – Новое значение системы координат.

Пример установки нового значения:

```
CoordSystem.set_current(CoordSystem.from_prj("10, 104, 7"))
```

property title: str

Наименование системы координат.

to_string() → str

Текстовое представление в виде <тип>:<строка>

property unit: LinearUnit

Единицы измерения.

property wkt: str

Строка WKT или пустая строка, если аналога не найдено.

19.2 CoordTransformer - Трансформация координат

class axipy.**CoordTransformer**

Класс для преобразования координат из одной СК в другую. При создании объекта трансформации в него передается исходная и целевая СК. После этого данный объект может использоваться для преобразования данных между этими СК.

Параметры

- **cs_from** – Исходная СК.
- **cs_to** – Целевая СК.

Список 6: Пример преобразования точки

```
from axipy import CoordSystem, CoordTransformer, Pnt, Rect

csLL = CoordSystem.from_prj("1, 104")
csMercator = CoordSystem.from_prj("10, 104, 7, 0")
inPoint = Pnt(10, 10)
transformer = CoordTransformer(csLL, csMercator)
outPoint = transformer.transform(inPoint)
print('Result point:', outPoint)
'''
>>> Result point: (1113194.9079327357 1111475.1028522244)
'''
outRect = transformer.transform(Rect(0,0,10,10))
print('Result rect:', outRect)
'''
>>> Result rect: (0.0 0.0) (1113194.9079327357 1111475.1028522244)
'''
```

Конструктор класса:

__init__(cs_from, cs_to)	Создает экземпляр класса.

Классовые методы:

proj_transform_definition(cs_from, cs_to)	Возвращает строку трансформации (pipeline) для преобразования между двумя СК, заданными в формате proj.

Методы:

transform(value)	Преобразовывает точки из исходной СК в целевую СК.

__init__(cs_from: Union[CoordSystem, str], cs_to: Union[CoordSystem, str])

Создает экземпляр класса.

classmethod proj_transform_definition(cs_from: str, cs_to: str) → str

Возвращает строку трансформации (pipeline) для преобразования между двумя СК, заданными в формате proj.

Параметры

- **cs_from** – Строка с определением исходной СК в формате proj
- **cs_to** – Строка с определением исходной СК в формате proj

Результат
Строка с определением трансформации между двумя этими СК в формате proj

Список 7: Пример получения строки трансформации

```
str_from = '+proj=longlat +ellps=WGS84 +no_defs'
str_to = '+proj=merc +ellps=GRS80 +no_defs'
print(CoordTransformer.proj_transform_definition(str_from, str_to))
'''
>>> proj=pipeline step proj=unitconvert xy_in=deg xy_out=rad step proj=merc␣
↪lon_0=0 k=1 x_0=0 y_0=0 ellps=GRS80
'''
```

transform(value: Union[Pnt, List[Pnt], QPointF, QRectF, Rect, List[QPointF]]) → Union[Pnt, Rect, List[Pnt]]

Преобразовывает точки из исходной СК в целевую СК.

Параметры
value – Входное значение. Может быть точкой, массивом точек axipy.utl.Pnt или axipy.utl.Rect.

Результат
Выходное значение. Тип зависит от входного и аналогичен ему.

Исключение
RuntimeError – Ошибка выполнения преобразования.

19.3 Единицы измерения расстояний

class axipy.**LinearUnits**

Единицы измерения расстояний. Класс является статическим словарем, доступным только для чтения (collections.abc.Mapping). Поддерживает обращение по индексу.

Классовые методы:

get(key[, default_value])	Возвращает значение по ключу.
items()	Возвращает список кортежей ключ-значение, где ключи это атрибуты класса, а значения это объекты класса axipy.LinearUnit.
keys()	Возвращает список ключей, где ключи это атрибуты класса.
values()	Возвращает список значений, где значения это объекты класса axipy.LinearUnit.

Атрибуты:

ch	Чейны
cm	Сантиметры
degree	Градусы
ft	Футы
inch	Дюймы
km	Километры
li	Линки
m	Метры
mi	Мили
mm	Миллиметры
nmi	Морские мили
rd	Роды
survey_ft	Топографические футы
yd	Ярды

class axipy.**LinearUnit**

Линейная единица измерения.

Используется для работы с координатами объектов или расстояний.

Примечание: Получить экземпляр можно через класс axipy.LinearUnits по соответствующему атрибуту.

Список 8: Пример создания

```
meters = LinearUnits.m  # LinearUnit
square_kilometers = AreaUnits.sq_km  # AreaUnit
```

Классовые методы:

from_area_unit(area_unit)	Возвращает единицу измерения расстояния, соответствующую единице измерения площадей.

Свойства:

conversion	Коэффициент преобразования в метры.
description	Краткое описание.
localized_name	Локализованное краткое наименование единиц измерения.
name	Краткое наименование единиц измерения.

Методы:

to_unit(unit[, value])	Перевод значения в другие единицы измерения.

property conversion: float

Коэффициент преобразования в метры.

property description: str

Краткое описание.

static from_area_unit(area_unit: AreaUnit)

Возвращает единицу измерения расстояния, соответствующую единице измерения площадей.

property localized_name: str

Локализованное краткое наименование единиц измерения.

property name: str

Краткое наименование единиц измерения.

to_unit(unit: Union[LinearUnit, AreaUnit], value: float = 1) → float

Перевод значения в другие единицы измерения.

Параметры

- **unit** – Единицы измерения, в которые необходимо перевести значение.
- **value** – Значение для перевода.

19.4 Единицы измерения площадей

class axipy.**AreaUnits**

Единицы измерения площадей. Класс является статическим словарем, доступным только для чтения (collections.abc.Mapping). Поддерживает обращение по индексу.

Классовые методы:

get(key[, default_value])	Возвращает значение по ключу.
items()	Возвращает список кортежей ключ-значение, где ключи это атрибуты класса, а значения это объекты класса axipy.AreaUnit.
keys()	Возвращает список ключей, где ключи это атрибуты класса.
values()	Возвращает список значений, где значения это объекты класса axipy.AreaUnit.

Атрибуты:

acre	Акры
hectare	Гектары
perch	Перчи
rood	Руды
sq_ch	Квадратные чейны
sq_cm	Квадратные сантиметры
sq_ft	Квадратные футы
sq_inch	Квадратные дюймы
sq_km	Квадратные километры
sq_li	Квадратные линки
sq_m	Квадратные метры
sq_mi	Квадратные мили
sq_mm	Квадратные миллиметры
sq_nmi	Квадратные морские мили
sq_rd	Квадратные роды
sq_survey_ft	Квадратные топографические футы
sq_yd	Квадратные ярды

class axipy.**AreaUnit**

Единица измерения площадей.

Примечание: Получить экземпляр можно через класс `axipy.AreaUnits` по соответствующему атрибуту.

Список 9: Пример создания

```
meters = LinearUnits.m  # LinearUnit
square_kilometers = AreaUnits.sq_km  # AreaUnit
```

Классовые методы:

`from_linear_unit`(linear_unit)	Возвращает единицу измерения площадей, соответствующую единице измерения расстояний.

Свойства:

`conversion`	Коэффициент преобразования в метры.
`description`	Краткое описание.
`localized_name`	Локализованное краткое наименование единиц измерения.
`name`	Краткое наименование единиц измерения.

Методы:

`to_unit`(unit[, value])	Перевод значения в другие единицы измерения.

property conversion: float

Коэффициент преобразования в метры.

property description: str

Краткое описание.

static from_linear_unit(linear_unit: LinearUnit)

Возвращает единицу измерения площадей, соответствующую единице измерения расстояний.

property localized_name: str

Локализованное краткое наименование единиц измерения.

property name: str

Краткое наименование единиц измерения.

to_unit(unit: Union[LinearUnit, AreaUnit], value: float = 1) → float

Перевод значения в другие единицы измерения.

Параметры

- **unit** – Единицы измерения, в которые необходимо перевести значение.
- **value** – Значение для перевода.

19.5 UnitValue - Значение вместе с единицей измерения

class axipy.**UnitValue**

Базовые классы: object

Контейнер, который хранит значение вместе с его единицей измерения.

Параметры

- **value** – Значение.
- **unit** – Единица измерения, в которой содержится значение. Если значение не указано, принимаются метры LinearUnits.m.

Пример:

```
unit =  UnitValue(2, LinearUnits.km)
print(unit)
>>> 2 km
```

Конструктор класса:

__init__([value, unit])	Создает экземпляр класса.

Свойства:

unit	Единица измерения.
value	Значение.

__init__(value: float = 1, unit: Optional[Unit] = None)
Создает экземпляр класса.

property unit: Unit
Единица измерения.

property value: float
Значение.

20

axipy.concurrent - Модуль для работы с длительными задачами в фоновом потоке.

Модуль для работы с длительными пользовательскими задачами, выполняемыми в фоновом потоке.

20.1 AxipyTask - Пользовательская задача

class axipy.**AxipyTask**

Базовый класс пользовательской задачи. От него можно наследоваться, создавая собственный задачи. Для этого нужно переопределить метод `run()`.

Методы:

`on_finished`(result)	Функция вызывается при завершении пользовательской задачи в потоке интерфейса.
`progress_handler`()	Возвращает обработчик для текущей задачи.
`run`()	Функция, выполняющая код пользовательской задачи.
`set_progress_handler`(ph)	Устанавливает обработчик для текущей задачи.

on_finished(result)

Функция вызывается при завершении пользовательской задачи в потоке интерфейса.

progress_handler() → AxipyProgressHandler

Возвращает обработчик для текущей задачи.

abstract run()

Функция, выполняющая код пользовательской задачи. Переопределяется в дочерних классах.

set_progress_handler(ph: AxipyProgressHandler)

Устанавливает обработчик для текущей задачи. Поддерживаются любые наследники AxipyProgressHandler.

20.2 AxipyAnyCallableTask - Обертка над пользовательской функцией для создания задачи

class axipy.**AxipyAnyCallableTask**

Объекты этого класса оборачивают пользовательские функции, превращая их в задачу, которая будет выполнена в фоновом потоке.

Параметры

- **fn** – Пользовательская функция, которая будет выполнятся. В нее будут переданы сохраненные параметры: список args и словарь kwargs.
- **args** – Список аргументов, передаваемый в функцию при запуске.
- **kwargs** – Словарь, передаваемый в функцию при запуске.

Список 1: Пример использования.

```
def user_heavy_function(arg1: int, arg2: str):
    print(f"Переданные аргументы: {arg1}, {arg2} \n")

task = AxipyAnyCallableTask(user_heavy_function, arg1=1, arg2="Тест")
task.with_handler(False)
task_manager.start_task(task)
```

Методы:

run()	Метод запускает выполнение задачи.
with_handler(value)	По умолчанию в пользовательскую функцию первым аргументом передаётся обработчик для управления задачей, установки прогресса и обработки отмены.

run()

Метод запускает выполнение задачи.

> **Предупреждение:** Вызывается автоматически при выполнении задачи. Вручную вызывать не следует.

with_handler(value: bool)

По умолчанию в пользовательскую функцию первым аргументом передаётся обработчик для управления задачей, установки прогресса и обработки отмены. Однако он не будет передаваться если вызвать эту функцию с False.

20.3 AxipyProgressHandler - Объект для связи с задачей и её управлением

class axipy.**AxipyProgressHandler**

Класс, объекты которого выполняют функцию канала передачи данных между выполняемой задачей и элементом отображающим прогресс.

error

Сигнал об ошибке, содержащий информацию о исключении.

Type
Signal[tuple]

Свойства:

result	Содержит результат выполнения задачи, связанной с обработчиком.

Методы:

add_progress(value)	Добавляет к текущему прогрессу переданное значение.
cancel()	Отменяет задачу, связанную с обработчиком.
is_canceled()	Проверяем не была ли задача отменена.
is_finished()	Проверяем не завершилась ли задача.
is_running()	Возвращает True если задача сейчас выполняется.
prepare_to_write_changes([description])	Делает индикатор выполнения бесконечным, убирает кнопку отмены и добавляет переданное описание.
progress()	Возвращает текущий прогресс выполнения.
raise_if_canceled()	Если задача была отменена выбрасывает исключение.
set_description(description)	Устанавливаем описание для задачи.
set_max_progress(value)	Устанавливает максимальное значение прогресса.
set_progress(value)	Устанавливает текущий прогресс задачи.
set_window_title(title)	Устанавливает заголовок диалога с прогрессом.

Сигналы:

canceled	Уведомляет, что задача была отменена.
description_changed	Уведомляет о изменении описания задачи.
finished	Уведомляет о завершении выполняемой задачи.
progress_changed	Уведомляет о изменении значения прогресса.
started	Уведомляет о старте выполнения задачи.
window_title_changed	Уведомляет о изменении заголовка диалога отображающего прогресс.

add_progress(value: float)

Добавляет к текущему прогрессу переданное значение.

Параметры
value – Значение, которое будет добавлено к прогрессу.

cancel()

Отменяет задачу, связанную с обработчиком.

Примечание: Эта функция посылает только запрос на отмену операции. Реальное прерывание операции возможно только если есть поддержка в пользовательском коде. Например, с помощью функций is_canceled или raise_if_canceled

---

property canceled: Signal

Уведомляет, что задача была отменена. Сигнал испускается когда была вызвана функция cancel.

Тип результата
Signal[]

Предупреждение: Получение этого сигнала не означает, что задача была завершена. Если в пользовательском коде не обрабатывается отмена, то задача будет продолжать выполняться до логического завершения.

property description_changed: Signal

Уведомляет о изменении описания задачи.

Тип результата
Signal[str], где str - текущее описание задачи.

property finished: Signal

Уведомляет о завершении выполняемой задачи.

Тип результата
Signal[]

is_canceled() → bool

Проверяем не была ли задача отменена.

is_finished() → bool

Проверяем не завершилась ли задача.

is_running() → bool

Возвращает True если задача сейчас выполняется.

prepare_to_write_changes(description: str = '')

Делает индикатор выполнения бесконечным, убирает кнопку отмены и добавляет переданное описание. По умолчанию описание содержит запись о том, что производится запись изменений.

Параметры
description – Сообщение которое будет отображаться.

progress() → float

Возвращает текущий прогресс выполнения.

property progress_changed: Signal

Уведомляет о изменении значения прогресса.

Тип результата
Signal[float]

raise_if_canceled()

Если задача была отменена выбрасывает исключение. Удобно при работе с большим количеством вложенных циклов или вызовов функции.

property result

Содержит результат выполнения задачи, связанной с обработчиком. Возвращается None если произошла ошибка, либо задача не предполагает возвращение результата.

Результат
Результат выполнения задачи или None.

set_description(description: str)

Устанавливаем описание для задачи. Эта информация может быть использована элементами отображающими прогресс выполнения.

Параметры
description – Новое описание задачи.

set_max_progress(value: float)

Устанавливает максимальное значение прогресса. Минимальное значение берется за ноль.

Параметры
value – Верхний порог для прогресса операции.

set_progress(value: float)

Устанавливает текущий прогресс задачи.

Параметры
value – Новое значение прогресса.

set_window_title(title: str)

Устанавливает заголовок диалога с прогрессом.

Параметры
title – Новый заголовок.

property started: Signal

Уведомляет о старте выполнения задачи.

Тип результата
Signal[]

property window_title_changed: Signal

Уведомляет о изменении заголовка диалога отображающего прогресс.

Тип результата
Signal[str]

20.4 TaskManager - Сервис для запуска и конфигурирования пользовательских задач

class axipy.**TaskManager**

Сервис, содержащий вспомогательные функции для запуска и конфигурирования пользовательских задач.

Методы:

generate_dialog_for_task(task, spec)	Возвращает диалог, следящий за выполнением задачи.
run_and_get(spec, func, *args, **kwargs)	Превращает пользовательскую функцию в задачу и запускает её с отображением прогресса выполнения.
run_in_gui(func, *args, **kwargs)	Выполняет переданную задачу в потоке интерфейса.
start_task(task)	Добавляет переданную задачу в очередь на выполнение.

generate_dialog_for_task(task: AxipyTask, spec: ProgressSpecification) → QDialog

Возвращает диалог, следящий за выполнением задачи.

Параметры

- **task** – Пользовательская задача.
- **spec** – Описание задачи.

Результат
Диалог, который будет отображать прогресс выполнения задачи.

Список 2: Пример использования.

```
def user_heavy_func(ph: AxipyProgressHandler, arg1: str, arg2: str):
    print(arg1 + arg2)

# Создаём задачу из пользовательской функции и передаём необходимые
# аргументы
task = AxipyAnyCallableTask(user_heavy_func, "Длительная ", "задача")
spec = ProgressSpecification(description="Длительная задача")
dialog = task_manager.generate_dialog_for_task(task, spec)
# Ставим задачу в очередь на выполнение
```

(continues on next page)

(продолжение с предыдущей страницы)

```
task_manager.start_task(task)
# Показываем диалог с прогрессом пока не завершилась задача
dialog.exec_()
```

run_and_get(spec: ProgressSpecification, func: Callable, *args, **kwargs) → Any

Превращает пользовательскую функцию в задачу и запускает её с отображением прогресса выполнения.

Параметры

- **spec** – Описание задачи.
- **func** – Пользовательская функция, которая будет выполняться. В нее передается список args и словарь kwargs.
- **args** – Список аргументов, передаваемый в функцию при запуске.
- **kwargs** – Словарь, передаваемый в функцию при запуске.

Результат

Результат выполнения пользовательской функции или None при ошибке.

Список 3: Пример использования.

```
def user_heavy_function(ph: AxipyProgressHandler, count: int):
    # Вначале задаём верхнюю планку изменения прогресса
    ph.set_max_progress(count)
    for i in range(0, count):
        if ph.is_canceled():
            break
        # Тут делаем длительные вычисления
        ph.add_progress(1)
    return ph.progress()

spec = ProgressSpecification(
    description="Длительная операция",
    flags=ProgressGuiFlags.CANCELABLE)
times = 6
real_times = task_manager.run_and_get(spec, user_heavy_function, times)
# выводим колличество раз которое отработал цикл внутри
print(real_times)
```

run_in_gui(func: Callable, *args, **kwargs) → Any

Выполняет переданную задачу в потоке интерфейса.

Это может быть удобно когда в процессе выполнения длительной фоновой задачи нужно спросить о чем нибудь пользователя отобразив диалог. Так же создавать/взаимодействовать с некоторыми объектами можно только из потока интерфейса.

start_task(task)

Добавляет переданную задачу в очередь на выполнение.

Параметры

task – Пользовательская задача.

Список 4: Пример использования.

```
def user_function(message: str):
    print(message)
task = AxipyAnyCallableTask(user_function, "Hi, world!")
# Т.к. мы не управляем прогрессом, то можно отключить передачу
# обработчика в функцию
task.with_handler(False)
task_manager.start_task(task)
```

20.5 ProgressGuiFlags - Флаги для настройки диалога, отображающего прогресс

class axipy.**ProgressGuiFlags**

Флаги для настройки диалога, отображающего прогресс.

Атрибуты:

CANCELABLE	У диалога с прогрессом будет кнопка отмены.
IDLE	Стандартный диалог с прогрессом.
NO_DELAY	Диалог с прогрессом появляется сразу без задержки.

CANCELABLE

У диалога с прогрессом будет кнопка отмены.

IDLE

Стандартный диалог с прогрессом.

NO_DELAY

Диалог с прогрессом появляется сразу без задержки. По умолчанию это 2 - 3 секунды.

20.6 ProgressSpecification - Параметры для настройки диалога, отображающего прогресс

class axipy.**ProgressSpecification**

Содержит параметры для настройки отображения диалога с прогрессом.

Список 5: Пример использования.

```
flags = ProgressGuiFlags.CANCELABLE | ProgressGuiFlags.NO_DELAY
spec = ProgressSpecification(
    description="Тестовое описание",
    window_title="Заголовок окна",
    flags=flags)
```

window_title

Задаёт заголовок окна для диалога, отображающего прогресс.

Type
str

flags

С помощью флагов можно выбрать тип диалога который будет отображаться пользователю. Флаги можно комбинировать с помощью побитовых операций.

Type
ProgressGuiFlags

description

Описание выполняемой задачи.

Type
str

with_handler

По умолчанию в пользовательскую функцию будет передаваться обработчик прогресса выполнения задачи AxipyProgressHandler. Но иногда это не нужно, тогда можно этот параметр установить в False.

Type
bool

Конструктор класса:

__init__([description, window_title, flags, ...])	Создает экземпляр класса.

__init__(description: str = '', window_title='', flags: ProgressGuiFlags = ProgressGuiFlags.IDLE, with_handler=True)

Создает экземпляр класса.

21

axipy.utl - Вспомогательные классы.

Вспомогательные классы.

21.1 Pnt - Точка

class axipy.**Pnt**

Точка без геопривязки. Может быть использована в качестве параметра геометрии (точки полигона) или при получении параметров, где результат представлен в виде точки (центр карты или элемента отчета).

Список 1: Создание точки.

```
from axipy import Pnt

# Создание точки
print(Pnt(1, 2))
print(Pnt(1.5, 2.5))
'''
>>> (1.0 2.0)
>>> (1.5 2.5)
'''
```

Конструктор класса:

__init__(x, y)	Создает экземпляр класса.

Классовые методы:

eq_approx(point1, point2[, precision])	Сравнивает две точки с заданной точностью.
from_qt(p)	Преобразует из формата Qt.

Свойства:

x	Устанавливает координату X.	или	возвращает
y	Устанавливает координату Y.	или	возвращает

Методы:

to_qt()	Преобразование в формат Qt.

__init__(x: float, y: float)

Создает экземпляр класса.

Конструктор класса.

Параметры

- **x** – X координата.
- **y** – Y координата.

classmethod eq_approx(point1: Pnt, point2: Pnt, precision: float = 1e-12) → bool

Сравнивает две точки с заданной точностью.

Параметры

- **point1** – Первая точка сравнения
- **point2** – Вторая точка сравнения
- **precision** – Точность сравнения

Результат

True если точки равны

classmethod from_qt(p: Union[QPointF, QPoint]) → Optional[Pnt]

Преобразует из формата Qt. Если класс не соответствует, возвращает None.

Параметры

p – Преобразуемая точка.

Список 2: Пример.

```
from PySide2.QtCore import QPoint, QPointF

# Создание точки из формата Qt
qpoint = QPoint(1, 2)
print(Pnt.from_qt(qpoint))
qpointf = QPointF(1.5, 2.5)
print(Pnt.from_qt(qpointf))
'''
>>> (1.0 2.0)
>>> (1.5 2.5)
'''
```

to_qt() → QPointF

Преобразование в формат Qt.

Список 3: Пример.

```
# Представление точки в формате Qt
print(Pnt(1, 2).to_qt())
'''
>>> PySide2.QtCore.QPointF(1.000000, 2.000000)
'''
```

property x: float

Устанавливает или возвращает координату X.

property y: float

Устанавливает или возвращает координату Y.

21.2 Rect - Прямоугольник

class axipy.**Rect**

Прямоугольник, который не обладает геопривязкой. Используется для различного вида запросов.

Список 4: Создание прямоугольника.

```
from axipy import Rect

rect = Rect(0, 0, 10, 5)
print(rect)
'''
>>> (0.0 0.0) (10.0 5.0)
'''
```

Конструктор класса:

__init__(xmin, ymin, xmax, ymax)	Создает экземпляр класса.

Классовые методы:

eq_approx(rect1, rect2[, precision])	Сравнивает два прямоугольника с заданной точностью.
from_qt(r)	Преобразует из формата Qt.

Свойства:

center	Устанавливает или возвращает центр прямоугольника.
height	Высота прямоугольника.
is_empty	Если один или оба размера равны нулю.
is_valid	Является ли прямоугольник правильным.
width	Ширина прямоугольника.
xmax	Устанавливает или возвращает максимальное значение X.
xmin	Устанавливает или возвращает минимальное значение X.
ymax	Устанавливает или возвращает максимальное значение Y.
ymin	Устанавливает или возвращает минимальное значение Y.

Методы:

contains(other)	Содержит ли полностью в своих границах переданный объект.
expanded(dx, dy)	Возвращает прямоугольник, увеличенный на заданные величины.
intersected(other)	Возвращает общий для обоих прямоугольник.
merge(other)	Возвращает прямоугольник, занимаемый обоими прямоугольниками.
normalize()	Исправляет прямоугольник, если его ширина или высота отрицательны.
to_qt()	Преобразование в формат Qt.
translated(dx, dy)	Возвращает прямоугольник, смещенный на заданную величину.

__init__(xmin: float, ymin: float, xmax: float, ymax: float)

Создает экземпляр класса.

property center: Pnt

Устанавливает или возвращает центр прямоугольника.

contains(other: Union[Pnt, QPointF, Rect, QRectF]) → bool

Содержит ли полностью в своих границах переданный объект.

Параметры
other – Переданный объект - точка или прямоугольник.

Список 5: Пример.

```
r = Rect(0, 0, 10, 10)
print("contains (0, 0)", r.contains((0, 0)))
print("contains (15, 10)", r.contains((15, 10)))
print("contains (0, 0) (5, 5)", r.contains(Rect(0, 0, 5, 5)))
print("contains (0, 0) (15, 15)", r.contains(Rect(0, 0, 15, 15)))
'''
```

(continues on next page)

(продолжение с предыдущей страницы)

```
>>> contains (0, 0) True
>>> contains (15, 10) False
>>> contains (0, 0) (5, 5) True
>>> contains (0, 0) (15, 15) False
'''
```

classmethod eq_approx(rect1: Rect, rect2: Rect, precision: float = 1e-12) → bool

Сравнивает два прямоугольника с заданной точностью.

Параметры

- **rect1** – Первый прямоугольник сравнения
- **rect2** – Второй прямоугольник сравнения
- **precision** – Точность сравнения

Результат

True если прямоугольники равны

expanded(dx: float, dy: float) → Rect

Возвращает прямоугольник, увеличенный на заданные величины. Увеличение размеров производится по отношению к центру, который не меняется в результате операции.

Параметры

- **dx** – расширение по X
- **dy** – расширение по Y

Список 6: Пример.

```
r = Rect(0, 0, 10, 10)
print(r.expanded(10, 10))
'''
>>> (-5.0 -5.0) (15.0 15.0)
'''
```

classmethod from_qt(r: Union[QRectF, QRect]) → Optional[Rect]

Преобразует из формата Qt. Если класс не соответствует, возвращает None.

Параметры

r – Преобразуемый прямоугольник.

property height: float

Высота прямоугольника.

intersected(other: Rect) → Rect

Возвращает общий для обоих прямоугольник.

Параметры

other – Прямоугольник, с которым производится операция.

Список 7: Пример.

```
r1 = Rect(0, 0, 5, 10)
r2 = Rect(0, 0, 10, 5)
print(r1.intersected(r2))
```

(continues on next page)

(продолжение с предыдущей страницы)

```
'''
>>> (0.0 0.0) (5.0 5.0)
'''
```

property is_empty: bool

Если один или оба размера равны нулю.

property is_valid: bool

Является ли прямоугольник правильным.

merge(other: Rect) → Rect

Возвращает прямоугольник, занимаемый обоими прямоугольниками.

Параметры

other – Прямоугольник, с которым производится операция.

Список 8: Пример.

```
r1 = Rect(0, 0, 5, 10)
r2 = Rect(0, 0, 10, 5)
print(r1.merge(r2))
'''
>>> (0.0 0.0) (10.0 10.0)
'''
```

normalize()

Исправляет прямоугольник, если его ширина или высота отрицательны.

Список 9: Пример.

```
from PySide2.QtCore import QRect
r = Rect.from_qt(QRect(0, 5, -10, 10))
print(r)
print(r.is_valid)
r.normalize()
print(r)
print(r.is_valid)
'''
>>> (0.0 5.0) (-10.0 15.0)
>>> False
>>> (-10.0 5.0) (0.0 15.0)
>>> True
'''
```

to_qt() → QRectF

Преобразование в формат Qt.

translated(dx: float, dy: float) → Rect

Возвращает прямоугольник, смещенный на заданную величину.

Параметры

- **dx** – смещение по X
- **dy** – смещение по Y

Список 10: Пример.

```
r = Rect(0, 0, 10, 10)
print(r.translated(10, -10))
'''
>>> (10.0 -10.0) (20.0 0.0)
'''
```

property width: float

Ширина прямоугольника.

property xmax: float

Устанавливает или возвращает максимальное значение X.

property xmin: float

Устанавливает или возвращает минимальное значение X.

property ymax: float

Устанавливает или возвращает максимальное значение Y.

property ymin: float

Устанавливает или возвращает минимальное значение Y.

21.3 FloatCoord - Координаты с плавающей точкой.

class axipy.**FloatCoord**

Класс представляет собой координату в формате числа с плавающей точкой (float). Для угловой координаты используется класс `axipy.AngleCoord`.

Конструктор класса:

`__init__`(value)	Создает экземпляр класса.

Свойства:

`value`	Возвращает числовое значение в формате числа с плавающей точкой (`float`).

Методы:

`as_float_round`(precision)	Округляет число до заданной точности
`as_float_round_signific`([digits])	Округляет число с указанием количества значащих цифр.
`as_string`(*[, precision, locale, ...])	Возвращает число в виде строки.

__init__(value: Union[float, int, str])

Создает экземпляр класса.

Создает координату из значения в различных форматах.

Параметры
value – Значение может быть:

- целым числом;
- числом с плавающей точкой;
- строкой, представляющей целое число или число с плавающей точкой;
- строкой, представляющей угловую координату с разделителями, или в формате румбов.

(<dd°mm'ss,zz">, <dd mm ss,zz>, <dd/mm/ss,zz>, <dd-mm-ss,zz>, <dd, mm,ss.zz>, <dd.zz>, <dd,zz> или в румбах WOB dd.zz.)

Исключение
ValueError – если не удалось преобразовать значение в число с плавающей точкой.

as_float_round(precision: int) → float

Округляет число до заданной точности

Параметры
precision – Количество знаков после запятой

Список 11: Пример.

```
from axipy import FloatFormatter

v = 333.99343111113
print(FloatFormatter.float_round(v, 2))
'''
>>> 333.99
'''
```

as_float_round_signific(digits: int = 15) → float

Округляет число с указанием количества значащих цифр.

Параметры
digits – Количество значащих цифр

Список 12: Пример.

```
print(FloatFormatter.float_round_signific(v, 6))
'''
>>> 333.993
'''
```

as_string(*, precision: int = 15, locale: Optional[QLocale] = None, omit_group_separator: bool = True, group_separator: Optional[str] = None, decimal_point: Optional[str] = None, suppress_trailing_zeros: bool = True) → str

Возвращает число в виде строки.

Параметры

- **precision** – Необходимое число знаков после запятой.
- **locale** – Локаль в которой нужно вывести значение. По умолчанию используется текущая локаль: QLocale().

- **omit_group_separator** – Исключить разделитель разрядов.
- **group_separator** – Использовать другой разделитель разрядов.
- **decimal_point** – Использовать другой десятичный разделитель.
- **suppress_trailing_zeros** – Не показывать завершающие нули.

property value: float

Возвращает числовое значение в формате числа с плавающей точкой (float).

21.4 AngleCoord - Угловые координаты.

class axipy.**AngleCoord**

Класс представляет собой угловую координату. Для координаты в формате числа с плавающей точкой (float) используется класс axipy.FloatCoord.

Угловую координату можно создать используя конструктор класса.

Список 13: Создание угловой координаты конструктором класса.

```
from axipy import AngleCoord

# Создание угловой координаты конструктором класса.
angle_coord = AngleCoord('''33°22'28,11972"''')
print(angle_coord)
'''
>>> 33°22'28,11972"
'''
```

Также, угловую координату можно создать из составляющих from_parts().

Конструктор класса:

__init__(value)	Создает экземпляр класса.

Классовые методы:

from_parts(degrees[, minutes, seconds])	Создает угловую координату из составляющих.

Свойства:

degrees	Возвращает градусы.
minutes	Возвращает минуты.
seconds	Возвращает секунды.
value	Возвращает числовое значение в формате числа с плавающей точкой (float).

Методы:

as_rumb([precision, suppress_trailing_zeros])	Получение строкового значения угловой координаты в формате румбов.
as_string([delimiter, precision, ...])	Получение строкового значения угловой координаты.
to_normalized([polar])	Возвращает угловую координату, нормализованную в диапазоне [0; 360) или [-180; 180].

__init__(value: Union[float, int, str])

Создает экземпляр класса.

Создает координату из значения в различных форматах.

Параметры

value – Значение может быть:

- целым числом;
- числом с плавающей точкой;
- строкой, представляющей целое число или число с плавающей точкой;
- строкой, представляющей угловую координату с разделителями, или в формате румбов.

(<dd°mm'ss,zz">, <dd mm ss,zz>, <dd/mm/ss,zz>, <dd-mm-ss,zz>, <dd, mm,ss.zz>, <dd.zz>, <dd,zz> или в румбах ЮВ dd.zz.)

Исключение

ValueError – если не удалось преобразовать значение в число с плавающей точкой.

as_rumb(precision: Optional[int] = None, suppress_trailing_zeros: bool = False) → str

Получение строкового значения угловой координаты в формате румбов.

Параметры

- **precision** – Количество знаков после запятой. Если None, округление не производится.
- **suppress_trailing_zeros** – Признак удаления завершающих нулей.

as_string(delimiter: Optional[str] = None, precision: Optional[int] = None, suppress_trailing_zeros: bool = False) → str

Получение строкового значения угловой координаты.

Параметры

- **delimiter** – Разделитель. Например, '-' или '/'.
- **precision** – Количество знаков после запятой. Если None, округление не производится.
- **suppress_trailing_zeros** – Признак удаления завершающих нулей.

property degrees: int

Возвращает градусы.

classmethod from_parts(degrees: int, minutes: int = 0, seconds: float = 0.0) → AngleCoord

Создает угловую координату из составляющих.

Параметры

- **degrees** – Градусы.
- **minutes** – Минуты.
- **seconds** – Секунды.

property minutes: int

Возвращает минуты.

property seconds: float

Возвращает секунды.

to_normalized(polar=False) → AngleCoord

Возвращает угловую координату, нормализованную в диапазоне [0; 360) или [-180; 180].

Параметры

polar – Если `True`, то для нормализации используется полярная система координат [0; 360), если `False`, то нормализация происходит в диапазоне [-180; 180].

property value: float

Возвращает числовое значение в формате числа с плавающей точкой (`float`).

22

axipy.da - Модуль источников данных.

Модуль источников данных. В данном модуле содержатся классы и методы для работы с источниками данных.

22.1 DataProvider - Провайдеры

22.1.1 ProviderManager - Объект открытия/создания данных

class axipy.**ProviderManager**

Класс открытия/создания объектов данных DataProvider.

Примечание: Используйте готовый экземпляр этого класса axipy.provider_manager.

Примечание: Для удобного задания параметров используйте экземпляры провайдеров: tab, shp, csv, mif, excel, sqlite, postgre, oracle, mssql, ogr, svg, gdal, rest, tms, wms, wmts, dwg, panorama.

Примечание: Открытые данные автоматически попадают в хранилище данных axipy.DataManager.

Пример открытия локальной таблицы:

```
table = provider_manager.openfile('../path/to/datadir/table.tab')
```

Свойства:

csv	Файловый провайдер - Текст с разделителями.
dwg	Провайдер данных AutoCAD.
excel	Провайдер чтения файлов Excel.
gdal	Растровый провайдер GDAL.
mif	Провайдер данных MIF-MID.
mssql	Провайдер для базы данных MSSQLServer.
ogr	Векторный провайдер OGR.
oracle	Провайдер для базы данных Oracle.
panorama	Провайдер данных ГИС Панорама.
postgre	Провайдер для базы данных PostgreSQL.
rest	Провайдер REST.
shp	Векторный провайдер SHP.
sqlite	Векторный провайдер sqlite.
svg	Провайдер для SVG.
tab	Провайдер MapInfo.
tms	Тайловый провайдер.
wms	Web Map Service.
wmts	Web Map Tile Service.

Методы:

create(definition)	Создает и открывает данные из описания.
create_open(definition)	Создает и открывает данные из описания.
createfile(filepath, schema, *args, **kwargs)	Создает таблицу.
loaded_providers()	Возвращает список всех загруженных провайдеров данных.
open(definition)	Открывает данные по описанию.
open_hidden(definition)	Открывает данные по описанию.
openfile(filepath, *args, **kwargs)	Открывает данные из файла.
providers()	Возвращает список всех загруженных провайдеров данных.
query(query_text, *tables)	Выполняет SQL-запрос к перечисленным таблицам.
read_contents(definition)	Читает содержимое источника данных.

create(definition: dict) → DataObject

Создает и открывает данные из описания.

Параметры
definition – Описание объекта данных.

Псевдоним create_open().

create_open(definition: dict) → DataObject

Создает и открывает данные из описания.

Возможные параметры:

- `src` - Строка, определяющая местоположение источника данных. Это может быть либо путь к файлу с расширением TAB, либо пустая строка (для таблицы, размещаемой в памяти).
- `schema` - Схема таблицы. Задается массивом объектов, содержащих атрибуты.
- `hidden` - Если указано True, то созданный объект не будет зарегистрирован в каталоге. См. также `open_hidden()`

Параметры
definition – Описание объекта данных.

Пример:

```
definition = {
    'src': '../path/to/datadir/edit/table.tab',
    'schema': attr.schema(
        attr.string('field1'),
        attr.integer('field2'),
    ),
}
table = provider_manager.create(definition)
```

createfile(filepath: str, schema, *args, **kwargs) → DataObject

Создает таблицу.

`create()` выполняет ту же функцию, но в более обобщенном виде.

Параметры
- **filepath** – Путь к создаваемой таблице.
- **schema** – Схема таблицы.

property csv: CsvDataProvider

Файловый провайдер - Текст с разделителями.

property dwg: DwgDataProvider

Провайдер данных AutoCAD.

property excel: ExcelDataProvider

Провайдер чтения файлов Excel.

property gdal: GdalDataProvider

Растровый провайдер GDAL.

loaded_providers() → dict

Возвращает список всех загруженных провайдеров данных.

Результат
Провайдеры в виде пар `(Идентификатор : Описание)`.

property mif: MifMidDataProvider

Провайдер данных MIF-MID.

property mssql: MsSqlDataProvider

Провайдер для базы данных MSSQLServer.

property ogr: OgrDataProvider

Векторный провайдер OGR.

open(definition: dict) → DataObject

Открывает данные по описанию.

Формат описания объектов данных (набор и тип параметров) индивидуален для каждого провайдера данных, однако многие элементы используются для всех провайдеров данных. В нижеприведенной таблице можно определить какие параметр можно указывать при открытии того или иного источника. Т.е. допустимые параметры для конкретного провайдера указаны в соответствующем методе open для этого провайдера.

Таблица 1: Доступные провайдеры данных и ссылки на дополнительные параметры:

Провайдер	Краткое описание	Ссылка
tab	Провайдер MapInfo	axipy.TabDataProvider.open()
csv	Текст с разделителями	axipy.CsvDataProvider.open()
ogr	Векторный провайдер OGR	axipy.OgrDataProvider.open()
excel	Провайдер чтения файлов Excel	axipy.ExcelDataProvider.open()
shp	Векторный провайдер SHP	axipy.ShapeDataProvider.open()
sqlite	Векторный провайдер sqlite	axipy.SqliteDataProvider.open()
svg	Провайдер для SVG	axipy.SvgDataProvider.open()
gdal	Растровый провайдер GDAL	axipy.GdalDataProvider.open()
postgre	Провайдер для базы данных PostgreSQL	axipy.PostgreDataProvider.open()
oracle	Провайдер для базы данных Oracle	axipy.OracleDataProvider.open()
mssql	Провайдер для базы данных MSSQLServer	axipy.MsSqlDataProvider.open()
rest	Провайдер REST	axipy.RestDataProvider.open()
tms	Тайловый провайдер	axipy.TmsDataProvider.open()
wms	Web Map Service	axipy.WmsDataProvider.open()
wmts	Web Map Tile Service	axipy.WmtsDataProvider.open()
dwg	Провайдер файлов AutoCAD	axipy.DwgDataProvider.open()
panorama	Провайдер файлов	axipy.PanoramaDataProvider.open()

Также существуют параметры, которые допустимы независимо от типа провайдера

Таблица 2: Доступные провайдеры данных и ссылки на дополнительные параметры:

Параметр	Краткое описание
provider	Используемый провайдер. Допустимые значения можно получить `loaded_providers()`. Если не задан, то система пытается его определить самостоятельно.
src	Ссылка на источник. Как правило, это имя файла. Для конкретного провайдера может дублироваться под другим именем.
dataobject	Если источник содержит несколько таблиц, то имя конкретного указывается через данный параметр
alias	Псевдоним для открываемого источника данных. В системе открытыйобъект будет доступен по этому имени

Параметры
definition – Описание объекта данных.

Пример открытия файла (аналогичен `openfile()`):

```
json = {'src':'../path/to/datadir/world.tab'}
table_world = provider_manager.open(json)
```

или, что тоже самое:

```
json = {'filepath':'../path/to/datadir/world.tab'}
table_world = provider_manager.open(json)
```

Пример открытия файла с несколькими таблицами:

```
# Пример открытия GPKG файла::
definition = { 'src': '../path/to/datadir/example.gpkg',
               'dataobject': 'tablename' }
table = provider_manager.open(definition)
```

Пример открытия таблицы базы данных:

```
definition = {"host": "localhost",
              "db": "sample",
              "user": "postgres",
              "password": "postgres",
              "dataobject": "public.world",
```

(continues on next page)

(продолжение с предыдущей страницы)

```
              "provider": "PgDataProvider"}
table = provider_manager.open(definition)
```

open_hidden(definition: dict) → DataObject

Открывает данные по описанию. Аналогична функции `open()` за исключением того, что когда данный объект добавляется в каталог, он не учитывается в общем списке и от него из этого каталога не приходят события.

Примечание: См. также `open()`

Параметры
definition – Описание объекта данных.

Пример:

```
table = axipy.provider_manager.open_hidden({'src':'world.tab'})
print(len(axipy.data_manager), axipy.data_manager.exists(table))
axipy.data_manager.remove(table)
>>> 0 True
```

openfile(filepath: Union[str, Path], *args, **kwargs) → DataObject

Открывает данные из файла.

Параметры

- **filepath** – Путь к открываемому файлу.
- ****kwargs** – Именованные аргументы. Возможные варианты от провайдера. Подробнее см. `open()`

Пример:

```
table = provider_manager.openfile('../path/to/datadir/example.gpkg')
```

property oracle: OracleDataProvider

Провайдер для базы данных Oracle.

property panorama: PanoramaDataProvider

Провайдер данных ГИС Панорама.

property postgre: PostgreDataProvider

Провайдер для базы данных PostgreSQL.

providers() → List[DataProvider]

Возвращает список всех загруженных провайдеров данных.

Результат
Список провайдеров.

query(query_text: str, *tables) → Table

Выполняет SQL-запрос к перечисленным таблицам.

Предупреждение: Используйте `axipy.DataManager.query()`.

Параметры

- **query_text** – Текст запроса.
- ***tables** – Список таблиц, к которым выполняется запрос.

Результат
Таблица, если результатом запроса является таблица.

Пример:

```
query_text = "SELECT * FROM world, caps WHERE world.capital = caps.capital"
joined = provider_manager.query(query_text, world, caps)
```

read_contents(definition: Union[dict, str]) → List[str]

Читает содержимое источника данных.

Обычно используется для источников, способных содержать несколько объектов данных.

Параметры
definition – Описание источника данных.

Результат
Имена объектов данных.

Пример:

```
contents = axipy.provider_manager.read_contents('../path/to/datadir/example.
↪gpkg')
print(contents)
>>> ['world', 'worldcap']

world = axipy.provider_manager.openfile('../path/to/datadir/example.gpkg',␣
↪dataobject='world')
```

property rest: RestDataProvider

Провайдер REST.

property shp: ShapeDataProvider

Векторный провайдер SHP.

property sqlite: SqliteDataProvider

Векторный провайдер sqlite.

property svg: SvgDataProvider

Провайдер для SVG.

property tab: TabDataProvider

Провайдер MapInfo.

property tms: TmsDataProvider

Тайловый провайдер.

property wms: WmsDataProvider

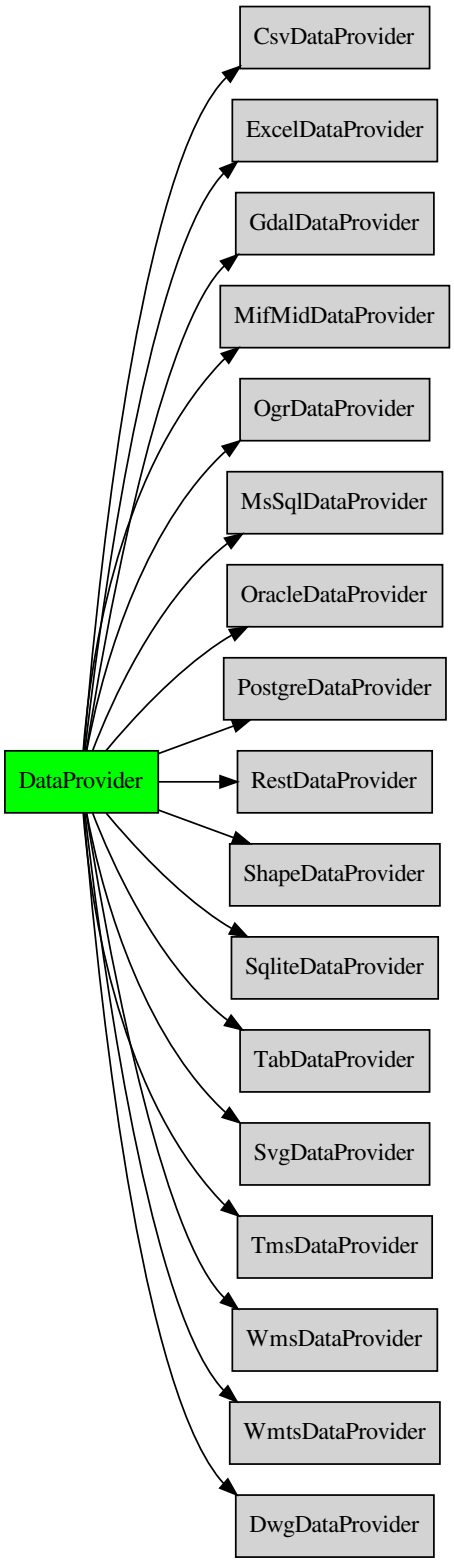
Web Map Service.

property wmts: WmtsDataProvider

Web Map Tile Service.

22.1.2 DataProvider - Провайдер данных

Иерархия классов провайдера данных:



class axipy.**DataProvider**

Абстрактный провайдер данных.

Свойства:

id	Идентификатор провайдера.

Методы:

create_open(*args, **kwargs)	Создает и открывает объект данных.
file_extensions()	Список поддерживаемых расширений файлов.
get_destination()	Создает назначение объекта данных.
get_source()	Создает источник данных.
open(*args, **kwargs)	Открывает объект данных.

create_open(*args, **kwargs)

Создает и открывает объект данных.

Пример:

```
provider.create_open(...)
```

Что эквивалентно:

```
provider.get_destiantion(...).create_open()
```

См.также:

DataProvider.destination().

file_extensions() → List[str]

Список поддерживаемых расширений файлов.

Результат
Пустой список для не файловых провайдеров.

get_destination() → Destination

Создает назначение объекта данных.

Исключение
NotImplementedError – Если провайдер не поддерживает создание назначений.

get_source() → Source

Создает источник данных.

Исключение
NotImplementedError – Если провайдер не поддерживает создание источников.

property id: str

Идентификатор провайдера.

open(*args, **kwargs)

Открывает объект данных.

Пример:

```
provider.open(...)
```

Что эквивалентно:

```
provider.get_source(...).open()
```

См.также:

`DataProvider.source()`.

22.1.3 Source - Источник данных

class axipy.**Source**

Источник данных.

Используется для открытия данных или для указания источника при конвертации.

Пример открытия:

```
table = source.open()
```

Пример конвертации:

```
destination.export_from(source)
```

Примечание: Не все провайдеры поддерживают открытие и конвертацию. См. описание конкретного провайдера данных.

Методы:

open()	Открывает объект данных.

open() → DataObject

Открывает объект данных.

22.1.4 Destination - Назначение объекта данных

class axipy.**Destination**

Назначение объекта данных.

Используется для создания данных или для указания назначения при конвертации.

Пример создания:

```
table = destination.create_open()
```

Пример конвертации:

```
destination.export_from(source)
```

Примечание: Не все провайдеры поддерживают создание и конвертацию. См. описание конкретного провайдера данных.

Методы:

create_open()	Создает и открывает объект данных.
export(features[, func_callback])	Создает объект данных и экспортирует в него записи.
export_from(source[, copy_schema])	Создает объект данных и экспортирует в него записи из источника данных.
export_from_table(table[, copy_schema, ...])	Создает объект данных и экспортирует в него записи из таблицы.

create_open() → DataObject

Создает и открывает объект данных.

export(features: Iterator[Feature], func_callback: Optional[Callable[[Feature, int], Union[None, bool]]] = None)

Создает объект данных и экспортирует в него записи.

Параметры

- **features** – Записи.
- **func_callback** – Функция, которая будет вызываться после экспорта каждой записи. В определении должны быть параметры следующих типов:
  - feature Feature - текущая запись
  - row int - порядковый номер

  Возможно прерывание процесса экспорта, для этого нужно вернуть False в func_callback.

Список 1: Пример экспорта данных

```
# Определяем схему будущей таблицы
schema = Schema(Attribute.string('name', 30), coordsystem="prj:1,104")
# Формируем данные для вставки. В нашем случае одна точка
features = [Feature(name='hello', geometry=Point(10, 10))]
# Имя выходного файла
filepath = './path/to/world_out.tab'
# Создаем таблицу по определенной ранее информации
dest = provider_manager.tab.get_destination(filepath, schema)
# Производим экспорт
dest.export(features)
```

export_from(source: Source, copy_schema: bool = False)

Создает объект данных и экспортирует в него записи из источника данных.

Параметры

- **source** – Источник данных.
- **copy_schema** – Копировать схему источника без изменений.

export_from_table(table: Table, copy_schema: bool = False, func_callback: Optional[Callable[[Feature, int], Union[None, bool]]] = None)

Создает объект данных и экспортирует в него записи из таблицы.

Параметры

- **table** – Таблица.
- **copy_schema** – Копировать схему источника без изменений.
- **func_callback** – Функция, которая будет вызываться после экспорта каждой записи. В определении должны быть параметры следующих типов:
  - feature Feature - текущая запись
  - row int - порядковый номер

  Возможно прерывание процесса экспорта, для этого нужно вернуть False в func_callback.

Список 2: Пример экспорта таблицы с прогрессом

```
# Определяем функцию прогресса
def func(feature: Feature, row: int) -> Union[None, bool]:
    # Экспорт первых 10 записей, затем процесс прерывается
    if row == 10:
        return False  # Прерывание процесса
    print(f'>> feature.id={feature.id} row={row}')

# Открываем исходную таблицу
table_src = provider_manager.openfile(input_filepath)
# Формируем целевую и производим экспорт
destination = provider_manager.tab.get_destination(output_filepath, Schema())
destination.export_from_table(table_src, copy_schema=True, func_callback=func)
```

Список 3: Пример экспорта таблицы в формат CSV

```
# Открываем исходную таблицу
table_src = provider_manager.openfile(input_filepath)
# Формируем целевую и производим экспорт
destination = provider_manager.csv.get_destination(output_filepath, Schema())
destination.export_from_table(table_src, copy_schema=True)
```

22.1.5 ExportParameters - Дополнительные параметры экспорта

class axipy.**ExportParameters**

Дополнительные параметры экспорта в таблицу базы данных.

Атрибуты:

createIndex	Создавать пространственный индекс
dropTable	Предварительно удалять существующую таблицу, если она присутствует в БД
errorFile	Наименование файла, куда будут прописываться команды по вставке записей, не принятых сервером
fixGeometry	Пробовать исправлять инвалидную геометрию
geometryAsText	Геометрию экспортировать как текстовые объекты
geometryColumnName	Наименование геометрической колонки
logFile	Наименование файла, куда будут прописываться успешно выполненные команды
mapCatalog	Регистрация импортируемой таблицы в mapinfo.mapinfo_mapcatalog
renditonColumnName	Наименование колонки с оформлением
srid	Значение SRID

createIndex: bool

Создавать пространственный индекс

dropTable: bool

Предварительно удалять существующую таблицу, если она присутствует в БД

errorFile: str

Наименование файла, куда будут прописываться команды по вставке записей, не принятых сервером

fixGeometry: bool

Пробовать исправлять инвалидную геометрию

geometryAsText: bool

Геометрию экспортировать как текстовые объекты

geometryColumnName: str

Наименование геометрической колонки

logFile: str

Наименование файла, куда будут прописываться успешно выполненные команды

mapCatalog: bool

Регистрация импортируемой таблицы в mapinfo.mapinfo_mapcatalog

renditonColumnName: str

Наименование колонки с оформлением

srid: int

Значение SRID

22.1.6 CsvDataProvider - Текст с разделителями

class axipy.**CsvDataProvider**

Базовые классы: DataProvider

Файловый провайдер: Текст с разделителями.

Примечание: Ссылку на провайдер можно получить через глобальную переменную `axipy.provider_manager.csv`.

Свойства:

id	Идентификатор провайдера.

Методы:

create_open(filepath, schema[, with_header, ...])	Создает и открывает объект данных.
file_extensions()	Список поддерживаемых расширений файлов.
get_destination(filepath, schema[, ...])	Создает назначение объекта данных.
get_source(filepath[, with_header, ...])	Создает источник данных.
open(filepath[, with_header, delimiter, ...])	Открывает объект данных.

create_open(filepath: str, schema: Schema, with_header: bool = True, delimiter: str = ';', encoding: str = 'utf8') → Table

Создает и открывает объект данных.

Параметры

- **filepath** – Путь к файлу.
- **schema** – Схема таблицы.
- **with_header** – Признак того, что в первой строке содержатся имена атрибутов таблицы.
- **delimiter** – Разделитель полей.
- **encoding** – Кодировка.

file_extensions() → List[str]

Список поддерживаемых расширений файлов.

Результат
Пустой список для не файловых провайдеров.

get_destination(filepath: str, schema: Schema, with_header: bool = True, delimiter: str = ';', encoding: str = 'utf8') → Destination

Создает назначение объекта данных.

Параметры

- **filepath** – Путь к файлу.
- **schema** – Схема таблицы.

- **with_header** – Признак того, что в первой строке содержатся имена атрибутов таблицы.
- **delimiter** – Разделитель полей.
- **encoding** – Кодировка.

get_source(filepath: str, with_header: bool = True, delimiter: str = ',', encoding: str = 'utf8', alias: Optional[str] = None) → Source

Создает источник данных.

Параметры

- **filepath** – Путь к файлу.
- **with_header** – Признак того, что в первой строке содержатся имена атрибутов таблицы.
- **delimiter** – Разделитель полей.
- **encoding** – Кодировка.

property id: str

Идентификатор провайдера.

open(filepath: str, with_header: bool = True, delimiter: str = ',', encoding: str = 'utf8', alias: Optional[str] = None) → Table

Открывает объект данных.

Параметры

- **filepath** – Путь к файлу.
- **with_header** – Признак того, что в первой строке содержатся имена атрибутов таблицы.
- **delimiter** – Разделитель полей.
- **encoding** – Кодировка.
- **alias** – Псевдоним для открываемой таблицы.

22.1.7 ExcelDataProvider - Провайдер чтения файлов Excel

class axipy.**ExcelDataProvider**

Базовые классы: DataProvider

Провайдер чтения файлов Excel.

Примечание: Ссылку на провайдер можно получить через глобальную переменную `axipy.provider_manager.excel`.

Свойства:

id	Идентификатор провайдера.

Методы:

create_open(filepath, schema)	Создает и открывает объект данных.
file_extensions()	Список поддерживаемых расширений файлов.
get_destination(filepath, schema)	Создает назначение объекта данных.
get_source(filepath[, page, with_header, ...])	Создает источник данных.
open(filepath[, page, with_header, ...])	Открывает объект данных.

create_open(filepath: str, schema: Schema) → Table

Создает и открывает объект данных.

Параметры

- **filepath** – Путь к файлу.
- **schema** – Схема таблицы.

file_extensions() → List[str]

Список поддерживаемых расширений файлов.

Результат
Пустой список для не файловых провайдеров.

get_destination(filepath: str, schema: Schema) → Destination

Создает назначение объекта данных.

Параметры

- **filepath** – Путь к файлу.
- **schema** – Схема таблицы.

get_source(filepath: str, page: Optional[str] = None, with_header: bool = False, encoding: str = 'utf8', alias: Optional[str] = None) → Source

Создает источник данных.

Параметры

- **filepath** – Путь к файлу.
- **page** – Имя страницы. Если не указана, то берется первая.
- **with_header** – Признак того, что в первой строке содержатся имена атрибутов таблицы.
- **encoding** – Кодировка.

property id: str

Идентификатор провайдера.

open(filepath: str, page: Optional[str] = None, with_header: bool = False, encoding: str = 'utf8', alias: Optional[str] = None) → Table

Открывает объект данных.

Параметры

- **filepath** – Путь к файлу.
- **page** – Имя страницы.
- **with_header** – Признак того, что в первой строке содержатся имена атрибутов таблицы.

- **encoding** – Кодировка.
- **alias** – Псевдоним для открываемой таблицы.

22.1.8 MifMidDataProvider -

class axipy.**MifMidDataProvider**

Базовые классы: DataProvider

Провайдер данных MIF-MID.

Примечание: Поддерживает экспорт только в TAB. См. convert_to_tab().

Примечание: Ссылку на провайдер можно получить через глобальную переменную axipy.provider_manager.mif.

Свойства:

id	Идентификатор провайдера.

Методы:

convert_to_tab(mif_filepath, tab_filepath)	Конвертирует из MIF в TAB.
create_open()	**Внимание:** Не поддерживается.
file_extensions()	Список поддерживаемых расширений файлов.
get_destination(filepath, schema)	Создает назначение объекта данных.
get_source()	**Внимание:** Не поддерживается.
open()	**Внимание:** Не поддерживается.

convert_to_tab(mif_filepath: str, tab_filepath: str)

Конвертирует из MIF в TAB.

Список 4: Пример экспорта

```
# Исходный файл MIF
mif_filepath = './path/to/world.mif'
# Целевой файл TAB
tab_filepath = './path/to/world_out.tab'
# Преобразование MIF в TAB
provider_manager.mif.convert_to_tab(mif_filepath, tab_filepath)
```

Параметры

- **mif_filepath** – Путь к исходному файлу.
- **tab_filepath** – Путь к выходному файлу.

Исключение

Exception – Если при конвертации произошла ошибка.

create_open()

> **Внимание:** Не поддерживается.

Исключение

NotImplementedError –

file_extensions() → List[str]

Список поддерживаемых расширений файлов.

Результат

Пустой список для не файловых провайдеров.

get_destination(filepath: str, schema: Schema) → Destination

Создает назначение объекта данных.

Параметры

- **filepath** – Путь к файлу.
- **schema** – Схема таблицы.

get_source() → Source

> **Внимание:** Не поддерживается.

Исключение

NotImplementedError –

property id: str

Идентификатор провайдера.

open()

> **Внимание:** Не поддерживается.

Исключение
NotImplementedError –

22.1.9 ShapeDataProvider - Векторный провайдер SHP

class axipy.**ShapeDataProvider**

Базовые классы: DataProvider

Векторный провайдер SHP.

Примечание: Ссылку на провайдер можно получить через глобальную переменную `axipy.provider_manager.shp`.

Свойства:

id	Идентификатор провайдера.

Методы:

create_open(filepath, schema[, encoding])	Создает и открывает объект данных.
file_extensions()	Список поддерживаемых расширений файлов.
get_destination(filepath, schema[, encoding])	Создает назначение объекта данных.
get_source(filepath[, encoding, prj, alias])	Создает источник данных.
open(filepath[, encoding, prj, alias])	Открывает объект данных.
open_temporary(schema)	Создает и открывает временную таблицу.

create_open(filepath: str, schema: Schema, encoding: str = 'utf8') → Table

Создает и открывает объект данных.

Параметры

- **filepath** – Путь к файлу.
- **schema** – Схема таблицы.
- **encoding** – Кодировка.

file_extensions() → List[str]

Список поддерживаемых расширений файлов.

Результат
Пустой список для не файловых провайдеров.

get_destination(filepath: str, schema: Schema, encoding: str = 'utf8') → Destination

Создает назначение объекта данных.

Параметры

- **filepath** – Путь к файлу.
- **schema** – Схема таблицы.
- **encoding** – Кодировка.

get_source(filepath: str, encoding: str = 'utf8', prj: Optional[str] = None, alias: Optional[str] = None) → Source

Создает источник данных.

Параметры

- **filepath** – Путь к файлу.
- **encoding** – Кодировка.
- **prj** – Строка Системы Координат.

property id: str

Идентификатор провайдера.

open(filepath: str, encoding: str = 'utf8', prj: Optional[str] = None, alias: Optional[str] = None) → Table

Открывает объект данных.

Пример:

```
shp = provider_manager.shp.open('world.shp', prj='1, 104')
```

Параметры

- **filepath** – Путь к файлу.
- **encoding** – Кодировка.
- **prj** – Строка Системы Координат.
- **alias** – Псевдоним для открываемой таблицы.

open_temporary(schema: Schema) → Table

Создает и открывает временную таблицу.

Параметры

schema – Схема таблицы.

22.1.10 SqliteDataProvider - Векторный провайдер sqlite

class axipy.**SqliteDataProvider**

Базовые классы: DataProvider

Векторный провайдер sqlite.

Примечание: Ссылку на провайдер можно получить через глобальную переменную `axipy.provider_manager.sqlite`.

Свойства:

id	Идентификатор провайдера.

Методы:

create_open()	**Внимание:** Не поддерживается.
file_extensions()	Список поддерживаемых расширений файлов.
get_destination()	**Внимание:** Не поддерживается.
get_source(filepath[, dataobject, sql, prj, ...])	Создает источник данных.
open(filepath[, dataobject, sql, prj, alias])	Открывает объект данных.

create_open()

Внимание: Не поддерживается.

Исключение
NotImplementedError –

file_extensions() → List[str]

Список поддерживаемых расширений файлов.

Результат
Пустой список для не файловых провайдеров.

get_destination()

Внимание: Не поддерживается.

Исключение
NotImplementedError –

get_source(filepath: str, dataobject: Optional[str] = None, sql: Optional[str] = None, prj: Optional[str] = None, alias: Optional[str] = None) → Source

Создает источник данных. В качестве объекта может быть указана либо таблица, либо текст запроса. Если указан sql, то он имеет более высокий

приоритет по отношению к значению dataobject. Если оба параметра опущены, будет возвращен None.

Параметры

- **filepath** – Путь к файлу.
- **dataobject** – Имя таблицы.
- **sql** – SQL-запрос.
- **prj** – Строка Системы Координат.

Пример с таблицей:

```
table = provider_manager.openfile('world.sqlite', dataobject='world')
```

Пример с запросом и переопределенной СК:

```
table = provider_manager.openfile('world.sqlite', sql="select * from world␣
↪where Страна like 'P%'", prj='12, 104, "m", 0')
```

property id: str

Идентификатор провайдера.

open(filepath: str, dataobject: Optional[str] = None, sql: Optional[str] = None, prj: Optional[str] = None, alias: Optional[str] = None) → Table

Открывает объект данных.

В качестве объекта может быть указана либо таблица, либо текст запроса. Если указан sql, то он имеет более высокий приоритет по отношению к значению dataobject. Если оба параметра опущены, будет возвращен None.

Параметры

- **filepath** – Путь к файлу.
- **dataobject** – Имя таблицы.
- **sql** – SQL-запрос.
- **prj** – Строка Системы Координат.
- **alias** – Псевдоним для открываемой таблицы.

22.1.11 TabDataProvider - Провайдер MapInfo

class axipy.**TabDataProvider**

Базовые классы: DataProvider

Провайдер MapInfo.

Примечание: Ссылку на провайдер можно получить через глобальную переменную `axipy.provider_manager.tab`.

Свойства:

id	Идентификатор провайдера.

Методы:

change_coordsystem(filepath, coordsystem)	Изменяет координатную систему в TAB файле без изменения самих данных.
copy_table_files(src_filepath, dest_filepath)	Копирует все связанные файлы с данным файлом в файловой системе под новым именем.
create_open(filepath, schema)	Создает и открывает объект данных.
file_extensions()	Список поддерживаемых расширений файлов.
get_destination(filepath, schema)	Создает назначение объекта данных.
get_source(filepath[, alias])	Создает источник данных.
open(filepath[, alias])	Открывает объект данных.
remove_table_files(filepath)	Удаляет все связанные файлы с данным файлом в файловой системе.
rename_table_files(src_filepath, dest_filepath)	Переименовывает файл и все связанные файлы с ним.

change_coordsystem(filepath: str, coordsystem: CoordSystem)

Изменяет координатную систему в TAB файле без изменения самих данных. Меняется непосредственно сам файл, так что рекомендуется сделать копию.

Параметры

- **filepath** – Путь к файлу TAB (имя файла).
- **coordsystem** – Новое значение СК

Исключение

RuntimeError – При возникновении ошибки

Список 5: Пример использования

```
in_filepath = 'path/to/input_filename.tab'
cs = CoordSystem.from_prj('10, 104, 7, 0')
provider_manager.tab.change_coordsystem(in_filepath, cs)
```

copy_table_files(src_filepath: str, dest_filepath: str)

Копирует все связанные файлы с данным файлом в файловой системе под новым именем.

Параметры

- **src_filepath** – Путь к исходному файлу TAB (имя файла).
- **dest_filepath** – Путь к выходному файлу TAB (имя файла).

Исключение

RuntimeError – При возникновении ошибки

Список 6: Пример использования

```
src_filepath = 'path/to/input_filename.tab'
dest_filepath = 'path/to/output_filename.tab'
provider_manager.tab.copy_table_files(src_filepath, dest_filepath)
```

create_open(filepath: str, schema: Schema) → Destination

Создает и открывает объект данных.

Параметры

- **filepath** – Путь к файлу.
- **schema** – Схема таблицы.

file_extensions() → List[str]

Список поддерживаемых расширений файлов.

Результат

Пустой список для не файловых провайдеров.

get_destination(filepath: str, schema: Schema) → Destination

Создает назначение объекта данных.

Параметры

- **filepath** – Путь к файлу.
- **schema** – Схема таблицы.

get_source(filepath: str, alias: Optional[str] = None) → Source

Создает источник данных.

Параметры

filepath – Путь к файлу.

property id: str

Идентификатор провайдера.

open(filepath: str, alias: Optional[str] = None) → Table

Открывает объект данных.

Параметры

- **filepath** – Путь к файлу.
- **alias** – Псевдоним для открываемой таблицы.

remove_table_files(filepath: str)

Удаляет все связанные файлы с данным файлом в файловой системе.

Параметры

filepath – Путь к файлу TAB (имя файла).

Исключение

RuntimeError – При возникновении ошибки

Список 7: Пример использования

```
filepath = 'path/to/input_filename.tab'
provider_manager.tab.remove_table_files(filepath)
```

rename_table_files(src_filepath: str, dest_filepath: str)

Переименовывает файл и все связанные файлы с ним.

Параметры

- **src_filepath** – Путь к исходному файлу TAB (имя файла).
- **dest_filepath** – Путь к новому имени файла TAB (имя файла). Файл не должен существовать.

Исключение
RuntimeError – При возникновении ошибки

Список 8: Пример использования

```
src_filepath = 'path/to/old_filename.tab'
dest_filepath = 'path/to/new_filename.tab'
provider_manager.tab.rename_table_files(src_filepath, dest_filepath)
```

22.1.12 SvgDataProvider - Провайдер для SVG

class axipy.**SvgDataProvider**

Базовые классы: DataProvider

Провайдер для SVG.

Примечание: Ссылку на провайдер можно получить через глобальную переменную `axipy.provider_manager.excel`.

Свойства:

id	Идентификатор провайдера.

Методы:

create_open()	**Внимание:** Не поддерживается.
file_extensions()	Список поддерживаемых расширений файлов.
get_destination()	**Внимание:** Не поддерживается.
get_source(data[, alias])	Создает источник данных.
open(data[, alias])	Открывает объект данных.

create_open()

Внимание: Не поддерживается.

Исключение
NotImplementedError –

file_extensions() → List[str]

Список поддерживаемых расширений файлов.

Результат

Пустой список для не файловых провайдеров.

get_destination()

> **Внимание:** Не поддерживается.

Исключение

NotImplementedError –

get_source(data: str, alias: Optional[str] = None) → Source

Создает источник данных.

Параметры

data – Имя файла или описание источника данных.

property id: str

Идентификатор провайдера.

open(data: str, alias: Optional[str] = None) → DataObject

Открывает объект данных.

Параметры

- **data** – Имя файла или описание источника данных.
- **alias** – Псевдоним для открываемой таблицы.

22.1.13 PostgreDataProvider - Провайдер для базы данных PostgreSQL

class axipy.**PostgreDataProvider**

Базовые классы: DataProvider

Провайдер для Базы Данных PostgreSQL.

Примечание: Ссылку на провайдер можно получить через глобальную переменную `axipy.provider_manager.postgre`.

Свойства:

id	Идентификатор провайдера.

Методы:

create_open(schema, dataobject, db_name, ...)	Создает и открывает объект данных.
file_extensions()	Список поддерживаемых расширений файлов.
get_destination(schema, dataobject, db_name, ...)	Создает назначение объекта данных.
get_source(host, db_name, user, password[, ...])	Создает описательную структуру для источника данных.
open(host, db_name, user, password[, port, ...])	Открывает объект данных.

create_open(schema: Schema, dataobject: str, db_name: str, host: str, user: str, password: str, port: int = 5432) → Table

Создает и открывает объект данных.

Параметры

- **schema** – Схема таблицы.
- **dataobject** – Имя таблицы.
- **db_name** – Имя базы данных.
- **host** – Адрес сервера.
- **user** – Имя пользователя.
- **password** – Пароль.
- **port** – Порт.

file_extensions() → List[str]

Список поддерживаемых расширений файлов.

Результат

Пустой список для не файловых провайдеров.

get_destination(schema: Schema, dataobject: str, db_name: str, host: str, user: str, password: str, port: int = 5432, export_params: Optional[ExportParameters] = None) → Destination

Создает назначение объекта данных.

Параметры

- **schema** – Схема таблицы.
- **dataobject** – Имя таблицы.
- **db_name** – Имя базы данных.
- **host** – Адрес сервера.
- **user** – Имя пользователя.
- **password** – Пароль.
- **port** – Порт.
- **export_params** – Дополнительные параметры экспорта.

get_source(host: str, db_name: str, user: str, password: str, port: int = 5432, dataobject: Optional[str] = None, sql: Optional[str] = None, prj: Optional[str] = None, alias: Optional[str] = None) → Source

Создает описательную структуру для источника данных. Она в дальнейшем может быть использована при открытии данных `ProviderManager.open()`.

В качестве таблицы можно указать либо ее наименование dataobject либо текст запроса sql.

Параметры

- **host** – Адрес сервера.
- **db_name** – Имя базы данных.
- **user** – Имя пользователя.
- **password** – Пароль.
- **port** – Порт.
- **dataobject** – Имя таблицы.
- **sql** – SQL-запрос. Если указан, то он имеет более высокий приоритет по отношению к значению dataobject.
- **prj** – Строка Системы Координат.

Пример с указанием имени таблицы:

```
definition = provider_manager.postgre.get_source('localhost', 'test',
↪'postgres', 'postgres', dataobject='world')
table = provider_manager.open(definition)
```

Пример с указанием текста запроса:

```
definition = provider_manager.postgre.get_source('localhost', 'test',
↪'postgres', 'postgres', sql="select * from world where Страна like 'P%'")
table = provider_manager.open(definition)
```

property id: str

Идентификатор провайдера.

open(host: str, db_name: str, user: str, password: str, port: int = 5432, dataobject: Optional[str] = None, sql: Optional[str] = None, prj: Optional[str] = None, alias: Optional[str] = None) → Table

Открывает объект данных.

В качестве таблицы можно указать либо ее наименование dataobject либо текст запроса sql.

Параметры

- **host** – Адрес сервера.
- **db_name** – Имя базы данных.
- **user** – Имя пользователя.
- **password** – Пароль.
- **port** – Порт.
- **dataobject** – Имя таблицы.

- **sql** – SQL-запрос. Если указан, то он имеет более высокий приоритет по отношению к значению dataobject.
- **prj** – Строка Системы Координат.
- **alias** – Псевдоним для открываемой таблицы.

22.1.14 OracleDataProvider - Провайдер для базы данных Oracle

class axipy.**OracleDataProvider**

Базовые классы: DataProvider

Провайдер для Базы Данных Oracle.

Примечание: Для подключения к БД Oracle необходимо настроить `Oracle Instant Client`.

См. Руководство по установке и активации.

Примечание: Ссылку на провайдер можно получить через глобальную переменную `axipy.provider_manager.oracle`.

Свойства:

id	Идентификатор провайдера.

Методы:

create_open(schema, dataobject, db_name, ...)	Создает и открывает объект данных.
file_extensions()	Список поддерживаемых расширений файлов.
get_destination(schema, dataobject, db_name, ...)	Создает назначение объекта данных.
get_source(host, db_name, user, password[, ...])	Создает описательную структуру для источника данных.
open(host, db_name, user, password[, port, ...])	Открывает объект данных.

create_open(schema: Schema, dataobject: str, db_name: str, host: str, user: str, password: str, port: int = 1521) → Table

Создает и открывает объект данных.

Параметры

- **schema** – Схема таблицы.
- **dataobject** – Имя таблицы.
- **db_name** – Имя базы данных.
- **host** – Адрес сервера.

- **user** – Имя пользователя.
- **password** – Пароль.
- **port** – Порт.

file_extensions() → List[str]

Список поддерживаемых расширений файлов.

Результат

Пустой список для не файловых провайдеров.

get_destination(schema: Schema, dataobject: str, db_name: str, host: str, user: str, password: str, port: int = 1521, export_params: Optional[ExportParameters] = None) → Destination

Создает назначение объекта данных.

Параметры

- **schema** – Схема таблицы.
- **dataobject** – Имя таблицы.
- **db_name** – Имя базы данных.
- **host** – Адрес сервера.
- **user** – Имя пользователя.
- **password** – Пароль.
- **port** – Порт.
- **export_params** – Дополнительные параметры экспорта.

get_source(host: str, db_name: str, user: str, password: str, port: int = 1521, dataobject: Optional[str] = None, sql: Optional[str] = None, alias: Optional[str] = None) → Source

Создает описательную структуру для источника данных. Она в дальнейшем может быть использована при открытии данных `ProviderManager.open()`.

В качестве таблицы можно указать либо ее наименование dataobject либо текст запроса sql.

Параметры

- **host** – Адрес сервера.
- **db_name** – Имя базы данных.
- **user** – Имя пользователя.
- **password** – Пароль.
- **port** – Порт.
- **dataobject** – Имя таблицы.
- **sql** – SQL-запрос. Если указан, то он имеет более высокий приоритет по отношению к значению dataobject.

Пример с указанием имени таблицы:

```
definition = provider_manager.oracle.get_source('localhost', 'test', 'oracle',
↪ 'oracle', dataobject='world')
table = provider_manager.open(definition)
```

Пример с указанием текста запроса:

```
definition = provider_manager.oracle.get_source('localhost', 'test', 'oracle',
↪ 'oracle', sql="select * from world where Страна like 'P%'")
table = provider_manager.open(definition)
```

property id: str

Идентификатор провайдера.

open(host: str, db_name: str, user: str, password: str, port: int = 1521, dataobject: Optional[str] = None, sql: Optional[str] = None, alias: Optional[str] = None) → Table

Открывает объект данных.

В качестве таблицы можно указать либо ее наименование dataobject либо текст запроса sql.

Параметры

- **host** – Адрес сервера.
- **db_name** – Имя базы данных.
- **user** – Имя пользователя.
- **password** – Пароль.
- **port** – Порт.
- **dataobject** – Имя таблицы.
- **sql** – SQL-запрос. Если указан, то он имеет более высокий приоритет по отношению к значению dataobject.
- **alias** – Псевдоним для открываемой таблицы.

22.1.15 MsSqlDataProvider - Провайдер для базы данных MSSQLServer

class axipy.**MsSqlDataProvider**

Базовые классы: DataProvider

Провайдер для Базы Данных MSSQLServer.

Примечание: Для работы с СУБД Microsoft SQL Server необходимо скачать и установить Microsoft SQL Server Native Client.

См. Руководство по установке и активации.

Примечание: Ссылку на провайдер можно получить через глобальную переменную `axipy.provider_manager.mssql`.

Свойства:

id	Идентификатор провайдера.

Методы:

create_open(*args, **kwargs)	Создает и открывает объект данных.
file_extensions()	Список поддерживаемых расширений файлов.
get_destination()	Создает назначение объекта данных.
get_source(host, db_name, user, password[, ...])	Создает описательную структуру для источника данных.
open(host, db_name, user, password[, port, ...])	Открывает объект данных.

create_open(*args, **kwargs)

Создает и открывает объект данных.

Пример:

```
provider.create_open(...)
```

Что эквивалентно:

```
provider.get_destiantion(...).create_open()
```

См.также:

DataProvider.destination().

file_extensions() → List[str]

Список поддерживаемых расширений файлов.

Результат
Пустой список для не файловых провайдеров.

get_destination() → Destination

Создает назначение объекта данных.

Исключение
NotImplementedError – Если провайдер не поддерживает создание назначений.

get_source(host: str, db_name: str, user: str, password: str, port: int = 1433, dataobject: Optional[str] = None, sql: Optional[str] = None, alias: Optional[str] = None) → Source

Создает описательную структуру для источника данных. Она в дальнейшем может быть использована при открытии данных ProviderManager.open().

Примечание: В качестве таблицы можно указать либо ее наименование dataobject либо текст запроса sql.

Примечание: Ссылку на провайдер можно получить через глобальную переменную axipy.provider_manager.mssql.

Параметры

- **host** – Адрес сервера.

- **db_name** – Имя базы данных.
- **user** – Имя пользователя.
- **password** – Пароль.
- **port** – Порт.
- **dataobject** – Имя таблицы.
- **sql** – SQL-запрос. Если указан, то он имеет более высокий приоритет по отношению к значению dataobject.

Пример с указанием имени таблицы:

```
definition = provider_manager.mssql.get_source('localhost', 'test', 'sa', 'sa
↪', dataobject='world')
table = provider_manager.open(definition)
```

Пример с указанием текста запроса:

```
definition = provider_manager.mssql.get_source('localhost', 'test', 'sa', 'sa
↪', sql="select * from world where Страна like 'P%'")
table = provider_manager.open(definition)
```

property id: str

Идентификатор провайдера.

open(host: str, db_name: str, user: str, password: str, port: int = 1433, dataobject: Optional[str] = None, sql: Optional[str] = None, alias: Optional[str] = None) → Table

Открывает объект данных.

Примечание: В качестве таблицы можно указать либо ее наименование dataobject либо текст запроса sql.

Параметры

- **host** – Адрес сервера.
- **db_name** – Имя базы данных.
- **user** – Имя пользователя.
- **password** – Пароль.
- **port** – Порт.
- **dataobject** – Имя таблицы.
- **sql** – SQL-запрос. Если указан, то он имеет более высокий приоритет по отношению к значению dataobject.
- **alias** – Псевдоним для открываемой таблицы.

22.1.16 TmsDataProvider - Тайловый провайдер

class axipy.**TmsDataProvider**

Базовые классы: DataProvider

Провайдер для тайловых серверов.

Примечание: Ссылку на провайдер можно получить через глобальную переменную `axipy.provider_manager.tms`.

Свойства:

id	Идентификатор провайдера.

Методы:

create_open()	**Внимание:** Не поддерживается.
file_extensions()	Список поддерживаемых расширений файлов.
get_destination()	**Внимание:** Не поддерживается.
get_source(templateUrl[, minLevel, ...])	Создает источник данных.
open(templateUrl[, minLevel, maxLevel, ...])	Открывает объект данных.

create_open()

Внимание: Не поддерживается.

Исключение
NotImplementedError –

file_extensions() → List[str]

Список поддерживаемых расширений файлов.

Результат
Пустой список для не файловых провайдеров.

get_destination()

> **Внимание:** Не поддерживается.

Исключение
NotImplementedError –

get_source(templateUrl: str, minLevel: int = 0, maxLevel: int = 19, size: tuple = (256, 256), type_address: str = 'xyz', watermark: str = '', watermark_style: str = '', prj: Optional[str] = None, live_time: int = 0, alias: Optional[str] = None, maxAttempts: int = 0, noLocalCache: bool = False) → Source

Создает источник данных.

Параметры

- **templateUrl** – Шаблон для запроса данных. Например, https://maps.axioma-gis.ru/osm/{LEVEL}/{ROW}/{COL}.png
- **minLevel** – Минимальный уровень показа
- **maxLevel** – Максимальный уровень показа
- **size** – Размер тайлов
- **type_address** – Тип адресации к тайлам. Поддерживается два значения: xyz и quadkey
- **watermark** – Ссылка на правообладателя
- **watermark_style** – Стиль оформления текста, с которым на карте будут отображаться данные о правообладателе.
- **prj** – Строка с Системой Координат. Если None, то используется значение по умолчанию (CoordSys Earth Projection 10, 157, „m“)
- **live_time** – время жизни тайла в секундах. Если равно 0, то значение не учитывается.
- **maxAttempts** – Максимальное количество попыток запроса. Значение 0 соответствует значению по умолчанию.
- **noLocalCache** – Не использовать локальный кэш

property id: str

Идентификатор провайдера.

open(templateUrl: str, minLevel: int = 0, maxLevel: int = 19, size: tuple = (256, 256), type_address: str = 'xyz', watermark: str = '', watermark_style: str = '', prj: Optional[str] = None, live_time: int = 0, alias: Optional[str] = None, maxAttempts: int = 0, noLocalCache: bool = False) → DataObject

Открывает объект данных.

Параметры

- **templateUrl** – Шаблон для запроса данных. Например, https://maps.axioma-gis.ru/osm/{LEVEL}/{ROW}/{COL}.png
- **minLevel** – Минимальный уровень показа
- **maxLevel** – Максимальный уровень показа

- **size** – Размер тайлов
- **type_address** – Тип адресации к тайлам. Поддерживается два значения: xyz и quadkey
- **watermark** – Ссылка на правообладателя
- **watermark_style** – Стиль оформления текста, с которым на карте будут отображаться данные о правообладателе.
- **prj** – Строка с Системой Координат. Если None, то используется значение по умолчанию (CoordSys Earth Projection 10, 157, „m“)
- **live_time** – время жизни тайла в секундах. Если равно 0, то значение не учитывается.
- **alias** – Псевдоним для открываемого источника данных.
- **maxAttempts** – Максимальное количество попыток запроса. Значение 0 соответствует значению по умолчанию.
- **noLocalCache** – Не использовать локальный кэш

Пример открытия источника:

```
prj_mercator = 'CoordSys Earth Projection 10, 104, "m", 0 Bounds (-20037508.
↪34, -20037508.34) (20037508.34, 20037508.34)'
osm_raster = provider_manager.tms.open('http://maps.axioma-gis.ru/osm/{LEVEL}/
↪{ROW}/{COL}.png', prj=prj_mercator)
osm_layer = Layer.create(osm_raster)
map = Map([ osm_layer ])
view_manager.create_mapview(map)
```

22.1.17 RestDataProvider - Провайдер REST

class axipy.**RestDataProvider**

Базовые классы: DataProvider

Провайдер для ArcGIS REST.

Примечание: Ссылку на провайдер можно получить через глобальную переменную `axipy.provider_manager.rest`.

Свойства:

id	Идентификатор провайдера.

Методы:

create_open()	**Внимание:** Не поддерживается.
file_extensions()	Список поддерживаемых расширений файлов.
get_destination()	**Внимание:** Не поддерживается.
get_source(url[, fmt, imageSR, size, dpi, ...])	Создает источник данных.
open(url[, fmt, imageSR, size, dpi, ...])	Открывает объект данных.

create_open()

> **Внимание:** Не поддерживается.

Исключение
NotImplementedError –

file_extensions() → List[str]

Список поддерживаемых расширений файлов.

Результат
Пустой список для не файловых провайдеров.

get_destination()

> **Внимание:** Не поддерживается.

Исключение
NotImplementedError –

get_source(url: str, fmt: str = 'png32', imageSR: int = 102100, size: str = '1024*1024', dpi: int = 96, transparent: str = 'true', layers: str = '', alias: Optional[str] = None, maxAttempts: int = 0) → Source

Создает источник данных.

Параметры

- **url** – Базовый URL.
- **fmt** – Формат выходного растра.
- **imageSR** – Код EPSG для выходного растра.

- **size** – Размер тайлов.
- **dpi** – DPI.
- **transparent** – Прозрачность выходного растра.
- **layers** – Перечень слоев.
- **maxAttempts** – Максимальное количество попыток запроса. Значение 0 соответствует значению по умолчанию.

property id: str

Идентификатор провайдера.

open(url: str, fmt: str = 'png32', imageSR: int = 102100, size: str = '1024*1024', dpi: int = 96, transparent: str = 'true', layers: str = '', alias: Optional[str] = None, maxAttempts: int = 0) → DataObject

Открывает объект данных.

Параметры

- **url** – Базовый URL.
- **fmt** – Формат выходного растра.
- **imageSR** – Код EPSG для выходного растра.
- **size** – Размер тайлов.
- **dpi** – DPI.
- **transparent** – Прозрачность выходного растра.
- **layers** – Перечень слоев.
- **alias** – Псевдоним для открываемой таблицы.
- **maxAttempts** – Максимальное количество попыток запроса. Значение 0 соответствует значению по умолчанию.

22.1.18 WmsDataProvider - Web Map Service

class axipy.**WmsDataProvider**

Базовые классы: DataProvider

Провайдер для Web Map Service.

Примечание: Ссылку на провайдер можно получить через глобальную переменную `axipy.provider_manager.wms`.

Свойства:

id	Идентификатор провайдера.

Методы:

create_open()	**Внимание:** Не поддерживается.
file_extensions()	Список поддерживаемых расширений файлов.
get_destination()	**Внимание:** Не поддерживается.
get_source(url_capabilities, layers[, ...])	Создает источник данных.
open(url_capabilities, layers[, ...])	Открывает объект данных.

create_open()

> **Внимание:** Не поддерживается.

Исключение
NotImplementedError –

file_extensions() → List[str]

Список поддерживаемых расширений файлов.

Результат
Пустой список для не файловых провайдеров.

get_destination()

> **Внимание:** Не поддерживается.

Исключение
NotImplementedError –

get_source(url_capabilities: str, layers: List[str], image_format: str = 'image/png', prj: Optional[str] = None, style: Optional[str] = None, alias: Optional[str] = None) → Source

Создает источник данных.

Параметры

- **url_capabilities** – URL с метаданными capabilities.
- **layers** – Перечень слоев в виде списка.
- **prj** – Строка Системы Координат

- **image_format** – Формат выходного растра.
- **style** – Наименование стиля оформления.
- **alias** – Псевдоним для открываемого источника данных.

property id: str

Идентификатор провайдера.

open(url_capabilities: str, layers: List[str], image_format: str = 'image/png', prj: Optional[str] = None, style: Optional[str] = None, alias: Optional[str] = None) → DataObject

Открывает объект данных.

Параметры

- **url_capabilities** – URL с метаданными capabilities.
- **layers** – Перечень слоев в виде списка.
- **prj** – Строка Системы Координат
- **image_format** – Формат выходного растра.
- **style** – Наименование стиля оформления.
- **alias** – Псевдоним для открываемого источника данных.

Пример:

```
wms_raster = provider_manager.wms.open('http://www.mapinfo.com/miwms', ['World
↪'], prj='EPSG:4326', style='AreaStyleGreen')
```

22.1.19 WmtsDataProvider - Web Map Tile Service

class axipy.**WmtsDataProvider**

Базовые классы: DataProvider

Провайдер для тайловых серверов.

Примечание: Ссылку на провайдер можно получить через глобальную переменную `axipy.provider_manager.wmts`.

Свойства:

id	Идентификатор провайдера.

Методы:

create_open()	**Внимание:** Не поддерживается.
file_extensions()	Список поддерживаемых расширений файлов.
get_destination()	**Внимание:** Не поддерживается.
get_source(capabilitiesUrl, dataObject[, alias])	Создает источник данных.
open(capabilitiesUrl, dataObject[, alias])	Открывает объект данных.

create_open()

> **Внимание:** Не поддерживается.

Исключение
NotImplementedError –

file_extensions() → List[str]

Список поддерживаемых расширений файлов.

Результат
Пустой список для не файловых провайдеров.

get_destination()

> **Внимание:** Не поддерживается.

Исключение
NotImplementedError –

get_source(capabilitiesUrl: str, dataObject: str, alias: Optional[str] = None) → Source

Создает источник данных.

Параметры

- **capabilitiesUrl** – URL запроса метаданных.
- **dataObject** – Наименование слоя.

property id: str

Идентификатор провайдера.

open(capabilitiesUrl: str, dataObject: str, alias: Optional[str] = None) → DataObject

Открывает объект данных.

Параметры

- **capabilitiesUrl** – URL запроса метаданных.
- **dataObject** – Наименование слоя.
- **alias** – Псевдоним для открываемого источника данных.

22.1.20 GdalDataProvider - Растровый провайдер GDAL

class axipy.**GdalDataProvider**

Базовые классы: DataProvider

Провайдер для растров.

Примечание: Ссылку на провайдер можно получить через глобальную переменную `axipy.provider_manager.gdal`.

Свойства:

id	Идентификатор провайдера.

Методы:

create_open()	**Внимание:** Не поддерживается.
file_extensions()	Список поддерживаемых расширений файлов.
get_destination()	**Внимание:** Не поддерживается.
get_source(data[, alias])	Создает источник данных.
open(data[, alias])	Открывает объект данных.

create_open()

Внимание: Не поддерживается.

Исключение
NotImplementedError –

file_extensions() → List[str]

Список поддерживаемых расширений файлов.

Результат
Пустой список для не файловых провайдеров.

get_destination()

> **Внимание:** Не поддерживается.

Исключение
NotImplementedError –

get_source(data: str, alias: Optional[str] = None) → Source

Создает источник данных.

Параметры
data – Имя файла или описание источника данных.

property id: str

Идентификатор провайдера.

open(data: str, alias: Optional[str] = None) → Table

Открывает объект данных.

Параметры

- **data** – Имя файла или описание источника данных.
- **alias** – Псевдоним для открываемого растра.

22.1.21 OgrDataProvider - Векторный провайдер OGR

class axipy.**OgrDataProvider**

Базовые классы: DataProvider

Провайдер для векторных данных OGR.

Примечание: Ссылку на провайдер можно получить через глобальную переменную `axipy.provider_manager.ogr`.

Свойства:

id	Идентификатор провайдера.

Методы:

create_open()	**Внимание:** Не поддерживается.
file_extensions()	Список поддерживаемых расширений файлов.
get_destination()	**Внимание:** Не поддерживается.
get_source(data, dataobject[, alias, encoding])	Создает источник данных.
open(data, dataobject[, alias, encoding])	Открывает объект данных.

create_open()

> **Внимание:** Не поддерживается.

Исключение
NotImplementedError –

file_extensions() → List[str]

Список поддерживаемых расширений файлов.

Результат
Пустой список для не файловых провайдеров.

get_destination()

> **Внимание:** Не поддерживается.

Исключение
NotImplementedError –

get_source(data: str, dataobject: str, alias: Optional[str] = None, encoding: str = 'utf8') → Source

Создает источник данных.

Параметры

- **data** – Источник данных или имя файла.
- **dataobject** – Наименование таблицы

property id: str

Идентификатор провайдера.

open(data: str, dataobject: str, alias: Optional[str] = None, encoding: str = 'utf8') → DataObject

Открывает объект данных.

Параметры

- **data** – Источник данных или имя файла.
- **dataobject** – Наименование таблицы
- **alias** – Псевдоним для открываемой таблицы.
- **encoding** – Кодировка.

22.1.22 DwgDataProvider - Провайдер для AutoCAD

class axipy.**DwgDataProvider**

Базовые классы: DataProvider

Провайдер для источников формата AutoCAD.

Примечание: Ссылку на провайдер можно получить через глобальную переменную `axipy.provider_manager.dwg`.

Свойства:

id	Идентификатор провайдера.

Методы:

convert_file(src_filepath, dest_filepath[, ...])	Производит конвертацию исходный файл текущего провайдера в другой формат этого же провайдера.
create_open()	**Внимание:** Не поддерживается.
file_extensions()	Список поддерживаемых расширений файлов.
get_destination()	**Внимание:** Не поддерживается.
get_source(data[, alias])	Создает источник данных.
open(data[, alias])	Открывает объект данных.
set_palette(palette)	Устанавливает текущую палитру.

convert_file(src_filepath: str, dest_filepath: str, out_version: DwgFileVersion = DwgFileVersion.AutoCAD_R13, out_format: DwgFileFormat = DwgFileFormat.Dxf)

Производит конвертацию исходный файл текущего провайдера в другой формат этого же провайдера.

Параметры

- **src_filepath** – Путь к исходному файлу (имя файла).
- **dest_filepath** – Путь к выходному файлу (имя файла).
- **out_version** – Версия выходного файла
- **out_format** – Формат выходного файла

```
input_file = 'filename_in.dwg'
output_file = 'filename_out.dxf'
provider_manager.dwg.convert_file(input_file, output_file, out_format =␣
↪DwgFileFormat.Dxf, out_version = DwgFileVersion.AutoCAD_R13)
```

create_open()

Внимание: Не поддерживается.

Исключение
NotImplementedError –

file_extensions() → List[str]

Список поддерживаемых расширений файлов.

Результат
Пустой список для не файловых провайдеров.

get_destination()

> **Внимание:** Не поддерживается.

Исключение
NotImplementedError –

get_source(data: str, alias: Optional[str] = None) → Source
Создает источник данных.

Параметры
data – Имя файла или описание источника данных.

property id: str
Идентификатор провайдера.

open(data: str, alias: Optional[str] = None) → Table
Открывает объект данных.

Параметры

- **data** – Имя файла или описание источника данных.
- **alias** – Псевдоним для открываемого объекта.

set_palette(palette: DwgPalette)
Устанавливает текущую палитру.

Параметры
palette – Индекс палитры.

22.1.23 DwgFileVersion - Версия файла AutoCAD

class axipy.**DwgFileVersion**
Версия файла DWG AutoCAD. Используется при задании параметра в функции DwgDataProvider.convert_file()

Атрибуты:

AutoCAD_R9	AutoCAD Release 9
AutoCAD_R10	AutoCAD Release 10
AutoCAD_R11_12	AutoCAD Release 11-12
AutoCAD_R13	AutoCAD Release 13
AutoCAD_R14	AutoCAD Release 14
AutoCAD_2000	AutoCAD 2000-2002
AutoCAD_2004	AutoCAD 2004-2006
AutoCAD_2007	AutoCAD 2007-2009
AutoCAD_2010	AutoCAD 2010-2012
AutoCAD_2013	AutoCAD 2013-2016

22.1.24 DwgFileFormat - Формат файла AutoCAD

class axipy.**DwgFileFormat**

Формат файла AutoCAD. Используется при задании параметра в функции DwgDataProvider.convert_file()

Атрибуты:

Dwg	Файл DWG
Dxf	Текстовый файл DXF
Dxb	Бинарный файл DXF

22.1.25 DwgPalette - Палитра данных AutoCAD

class axipy.**DwgPalette**

Палитра при работе с файлами AutoCAD. Используется при задании параметра в функции DwgDataProvider.set_palette()

Атрибуты:

Light	Светлая тема
Dark	Темная тема

22.2 DataManager - Каталог данных

class axipy.**DataManager**

Хранилище объектов данных. При открытии таблицы или растра эти объекты автоматически попадают в данный каталог. Для отслеживания изменений в каталоге используются события added и removed.

Если же разработка ведется не в рамках приложения и ядро инициализировано явно посредством axipy.init_axioma(), то объект в каталог надо добавлять явно DataManager.add()

Примечание: Используйте готовый экземпляр этого класса axipy.data_manager.

Список 9: Пример использования.

```
# Отслеживание добавления или удаления в каталоге.
data_manager.added[str].connect(lambda n: print(f'Таблица "{n}" добавлена в␣
↪каталог'))
data_manager.removed[str].connect(lambda n: print(f'Таблица "{n}" удалена из␣
↪каталога'))
# Отслеживание изменения в каталоге.
data_manager.updated.connect(lambda params: print(f"Каталог изменен: {params}"))
# Открываем таблицу
table = provider_manager.openfile(filepath)
# Список объектов каталога
for t in data_manager:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    print(t.name)
# Доступ по имени
'world' in data_manager
try:
    found_table = data_manager['world']
except KeyError:
    pass
# Закрываем таблицу
table.close()
# Убираем отслеживание
data_manager.added[str].disconnect()
data_manager.removed[str].disconnect()
'''
Таблица "world" добавлена в каталог
world
Таблица "world" удалена из каталога
'''
```

Свойства:

added	Сигнал о добавлении объекта.
all_objects	Список всех объектов, включая скрытые.
count	Количество объектов данных.
objects	Список объектов.
removed	Сигнал об удалении объекта.
selection	Таблица выборки, если она существует.
sql_dialect	Тип используемого диалекта по умолчанию для выполнения SQL-предложений.
tables	Список таблиц.
updated	Сигнал об изменении каталога.

Методы:

add(data_object)	Добавляет объект данных в хранилище.
check_query(query_text[, dialect])	Производит проверку SQL-запроса на корректность.
exists(obj)	Проверяет, присутствует ли объект в каталоге.
find(name)	Производит поиск объект данных по имени.
query(query_text[, dialect])	Выполняет SQL-запрос к перечисленным таблицам.
query_hidden(query_text[, dialect])	Выполняет SQL-запрос к таблицам.
remove(data_object)	Удаляет объект данных.
remove_all()	Удаляет все объекты данных.

add(data_object: DataObject)

Добавляет объект данных в хранилище.

Параметры

data_object – Объект данных для добавления.

property added: SignalInstance

Сигнал о добавлении объекта. В качестве параметра передается наименование таблицы.

Тип результата
Signal[str]

property all_objects: List[DataObject]

Список всех объектов, включая скрытые.

check_query(query_text: str, dialect: TypeSqlDialect = TypeSqlDialect.sqlite) → Tuple[bool, str]

Производит проверку SQL-запроса на корректность.

Параметры

- **query_text** – Текст запроса.
- **dialect** – Диалект, который используется при выполнении запроса. Значение по умолчанию установлено как значение свойства sql_dialect.

Результат
Пара значений [Успешность проверки, Сообщение].

Список 10: Пример использования, если работа ведется в рамках Аксиома .

```
filepath = 'path/to/world.tab'
# Открываем таблицу
table = provider_manager.openfile(filepath)
table.name = 'world1'
# Проверка текста запроса
succ, mess = data_manager.check_query(f"select * from world1")
print(f'Check result: {succ}; {mess}')
'''
Check result: True; Запрос составлен верно
'''
```

property count: int

Количество объектов данных.

exists(obj: DataObject) → bool

Проверяет, присутствует ли объект в каталоге. Проверяет так-же и скрытые объекты, которые отсутствуют в общем списке.

Параметры
obj – проверяемый объект данных.

find(name: str) → Optional[DataObject]

Производит поиск объект данных по имени.

Параметры
name – Имя объекта данных.

Результат
Искомый объект данных или None.

property objects: List[DataObject]

Список объектов.

query(query_text: str, dialect: TypeSqlDialect = TypeSqlDialect.sqlite) → Optional[Table]

Выполняет SQL-запрос к перечисленным таблицам.

Параметры

- **query_text** – Текст запроса.
- **dialect** – Диалект, который используется при выполнении запроса. Значение по умолчанию установлено как значение свойства sql_dialect.

Результат

Таблица, если результатом запроса является таблица.

Исключение

RuntimeError – При возникновении ошибки.

Список 11: Пример использования, если работа ведется в рамках Аксиома .

```
filepath = 'path/to/world.tab'
# Открываем таблицу
table = provider_manager.openfile(filepath)
table.name = 'world1'
# Выполняем запрос
qry = data_manager.query('select * from world1 where Страна like "A%"')
# Выполняем тот-же запрос, но с явным указанием диалекта
qry = data_manager.query('select * from world1 where Страна like "A%"',␣
↪TypeSqlDialect.axioma)
```

query_hidden(query_text: str, dialect: TypeSqlDialect = TypeSqlDialect.sqlite) → Optional[Table]

Выполняет SQL-запрос к таблицам. В отличие от query() результирующий объект Table добавляется в каталог как скрытый объект. Он не учитывается в общем списке и от него из этого каталога не приходят события.

Параметры

- **query_text** – Текст запроса.
- **dialect** – Диалект, который используется при выполнении запроса. Значение по умолчанию установлено как sql_dialect.

Результат

Таблица, если результатом запроса является таблица.

remove(data_object: DataObject)

Удаляет объект данных.

Объект данных при этом закрывается.

Параметры

data_object – Объект данных для удаления.

remove_all()

Удаляет все объекты данных.

property removed: SignalInstance

Сигнал об удалении объекта.

Тип результата
Signal[str]

property selection: Optional[SelectionTable]

Таблица выборки, если она существует.

См.также:

`axipy.selection_manager`

property sql_dialect: TypeSqlDialect

Тип используемого диалекта по умолчанию для выполнения SQL-предложений. Если необходимо переопределить, то для конкретного sql предложения необходимо указывать диалект явно `query()`

Результат
Тип диалекта. Возможные значения TypeSqlDialect.axioma или TypeSqlDialect.sqlite.

property tables: List[Table]

Список таблиц.

property updated: SignalInstance

Сигнал об изменении каталога. В качестве параметра передается дополнительная информация в виде dict.

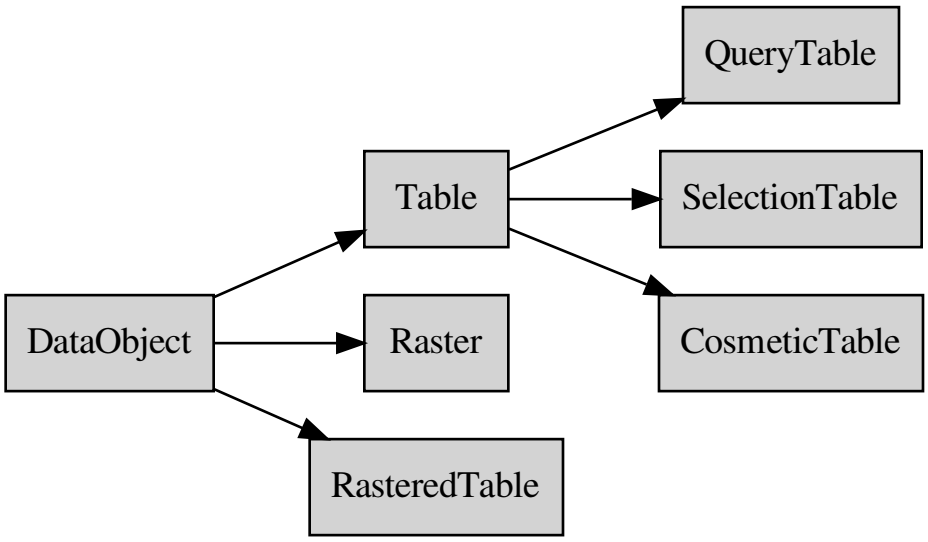
Таблица 3: Доступные параметры

Наименование	Значение	Описание
operation	added	Таблица добавлена в каталог
operation	removed	Таблица удалена из каталога
operation	nameChanged	Изменено наименование таблицы
operation	selectionChanged	Произведены изменения в выборке
name		Наименование таблицы, если оно доступно

Тип результата
Signal[]

22.3 DataObject - Объект данных

Иерархия классов:



class axipy.**DataObject**

Объект данных.

Открываемые объекты из источников данных представляются объектами этого типа. Возможные реализации: таблица, растр, грид, чертеж, панорама, и так далее.

Пример:

```
table = provider_manager.openfile('path/to/file.tab')
...
table.close()  # Закрывает таблицу
```

Для закрытия объекта данных можно использовать менеджер контекста - выражение `with`. В таком случае таблица будет закрыта при выходе из блока. См. `close()`.

Пример:

```
with provider_manager.openfile('path/to/file.tab') as raster:
    ...
    # При выходе из блока растр будет закрыт
```

Свойства:

destroyed	Сигнал оповещения об удалении объекта.
is_spatial	Признак того, что объект данных является пространственным.
name	Название объекта данных.
properties	Дополнительные свойства объекта данных.
provider	Провайдер изначального источника данных.

Методы:

close()	Пытается закрыть таблицу.

close()

Пытается закрыть таблицу.

Исключение
RuntimeError – Ошибка закрытия таблицы.

Примечание: Объект данных не всегда может быть сразу закрыт. Например, для таблиц используется транзакционная модель редактирования и перед закрытием необходимо сохранить или отменить изменения, если они есть. См. Table.is_modified.

---

property destroyed: SignalInstance

Сигнал оповещения об удалении объекта.

property is_spatial: bool

Признак того, что объект данных является пространственным.

property name: str

Название объекта данных.

property properties: dict

Дополнительные свойства объекта данных.

property provider: str

Провайдер изначального источника данных.

22.4 Table - Таблица

class axipy.**Table**

Базовые классы: DataObject

Таблица.

Менеджер контекста сохраняет изменения и закрывает таблицу.

Пример:

```
with provider_manager.openfile('path/to/file.tab') as table:
    ...
    # При выходе из блока таблица будет сохранена и закрыта
```

См.также:

commit(), DataObject.close().

Свойства:

can_redo	Возможен ли откат на один шаг вперед.
can_undo	Возможен ли откат на один шаг назад.
coordsystem	Система координат таблицы.
data_changed	Сигнал об изменении данных таблицы.
destroyed	Сигнал оповещения об удалении объекта.
hotlink	Наименование атрибута таблицы для хранения гиперссылки.
is_editable	Признак того, что таблица является редактируемой.
is_modified	Таблица содержит несохраненные изменения.
is_spatial	Признак того, что объект данных является пространственным.
is_temporary	Признак того, что таблица является временной.
name	Название объекта данных.
properties	Дополнительные свойства объекта данных.
provider	Провайдер изначального источника данных.
schema	Схема таблицы.
schema_changed	Сигнал об изменении схемы таблицы.
supported_operations	Доступные операции.

Методы:

close()	Пытается закрыть таблицу.
commit()	Сохраняет изменения в таблице.
count([bbox])	Возвращает количество записей, удовлетворяющих параметрам.
get_bounds()	Возвращает область, в которую попадают все данные таблицы.
insert(features)	Вставляет записи в таблицу.
items([bbox, ids])	Запрашивает записи, удовлетворяющие параметрам.
itemsByIds(ids)	Запрашивает записи по списку list с идентификаторами записей, либо перечень идентификаторов в виде списка.
itemsInObject(obj)	Запрашивает записи с фильтром по геометрическому объекту.
itemsInRect(bbox)	Запрашивает записи с фильтром по ограничивающему прямоугольнику.
redo([steps])	Производит откат вперед на заданное количество шагов.
remove(ids)	Удаляет записи из таблицы.
rollback()	Отменяет несохраненные изменения в таблице.
undo([steps])	Производит откат вперед на заданное количество шагов.
update(features)	Обновляет записи в таблице.

property can_redo: bool

Возможен ли откат на один шаг вперед.

property can_undo: bool

Возможен ли откат на один шаг назад.

close()

Пытается закрыть таблицу.

Исключение
RuntimeError – Ошибка закрытия таблицы.

Примечание: Объект данных не всегда может быть сразу закрыт. Например, для таблиц используется транзакционная модель редактирования и перед закрытием необходимо сохранить или отменить изменения, если они есть. См. Table.is_modified.

---

commit()

Сохраняет изменения в таблице.

Если таблица не содержит несохраненные изменения, то команда игнорируется.

Исключение
RuntimeError – Невозможно сохранить изменения.

property coordsystem: Optional[CoordSystem]

Система координат таблицы.

count(bbox: Union[Rect, QRectF, tuple] = None) → int

Возвращает количество записей, удовлетворяющих параметрам.

Данный метод является наиболее предпочтительным для оценки количества записей. При этом используется наиболее оптимальный вариант выполнения запроса для каждого конкретного провайдера данных.

Параметры

bbox – Ограничивающий прямоугольник.

Результат

Количество записей.

property data_changed: SignalInstance

Сигнал об изменении данных таблицы. Испускается когда были изменены данные таблицы. В качестве параметра передается дополнительная информация в виде dict.

Таблица 4: Доступные параметры

Наименование	Значение	Описание
operation	insert	Произведена вставка данных
operation	update	Данные были обновлены
operation	remove	Данные были удалены
operation	unknown	Тип операции не определен
ids	list	Перечень затронутых идентификаторов, если он доступен

Список 12: Пример подписки на изменение таблицы.

```
table = provider_manager.openfile(filepath)
table.data_changed.connect(lambda info: print(f'Таблица была изменена {info}
↪'))
```

property destroyed: SignalInstance

Сигнал оповещения об удалении объекта.

get_bounds() → Rect

Возвращает область, в которую попадают все данные таблицы.

property hotlink: str

Наименование атрибута таблицы для хранения гиперссылки.

Таблица 5: Возможны следующие варианты

Значение	Описание
axioma://world.tab	Открывает файл или рабочее пространство в аксиоме
addlayer://world	Добавляет слой world в текущую карту
exec://gimp	Запускает на выполнение программу gimp
https://axioma-gis.ru/	Открывает ссылку в браузере

Если префикс отсутствует, то производится попытка запустить по ассоциации.

См.также:

`axipy.render.VectorLayer.hotlink`.

insert(features: Union[Feature, Iterable[Feature]])

Вставляет записи в таблицу.

Параметры
features – Записи для вставки.

property is_editable: bool

Признак того, что таблица является редактируемой.

property is_modified: bool

Таблица содержит несохраненные изменения.

property is_spatial: bool

Признак того, что объект данных является пространственным.

property is_temporary: bool

Признак того, что таблица является временной. Это в первую очередь касается таблиц, созданных в памяти. А также таблица косметического слоя.

items(bbox: Union[Rect, QRectF, tuple] = None, ids: List[int] = None) → Iterator[Feature]

Запрашивает записи, удовлетворяющие параметрам. В качестве фильтра может быть указан либо ограничивающий прямоугольник, либо перечень идентификаторов в виде списка.

Параметры

- **bbox** – Ограничивающий прямоугольник.
- **ids** – Список идентификаторов.

Результат
Итератор по записям.

itemsByIds(ids: List[int]) → Iterator[Feature]

Запрашивает записи по списку `list` с идентификаторами записей, либо перечень идентификаторов в виде списка. Идентификаторы несохраненных записей имеют отрицательные значения.

Параметры
ids – Список идентификаторов.

Результат
Итератор по записям.

Список 13: Пример

```
table_world = provider_manager.openfile(filepath)
# Пример запроса по списку идентификаторов.
items = table_world.itemsByIds([11, 27, 41, 163, 203])
for f in items:
    print('Feature id={}. Страна={}'.format(f.id, f['Страна']))
# Просмотр идентификаторов всех записей, включая несохраненные
for f in table_world.items():
    print(f.id, f['Страна'])
# Получение несохраненных записей совместно с сохраненными
```

(continues on next page)

(продолжение с предыдущей страницы)

```
items_new = table_world.itemsByIds([-1, -12, 27])
for f in items_new:
    print('Feature id={}. Страна={}'.format(f.id, f['Страна']))
```

itemsInObject(obj: Geometry) → Iterator[Feature]

Запрашивает записи с фильтром по геометрическому объекту.

Параметры
obj – Геометрия. Если для нее не задана СК, используется СК таблицы.

Результат
Итератор по записям.

Список 14: Пример запроса по полигону

```
table_world = provider_manager.openfile(filepath)
v = 2000000
polygon = Polygon((-v, -v), (-v, v), (v, v), (v, -v))
items = table_world.itemsInObject(polygon)
for f in items:
    print('Feature id={}. Страна={}'.format(f.id, f['Страна']))
```

itemsInRect(bbox: Union[Rect, QRectF, tuple]) → Iterator[Feature]

Запрашивает записи с фильтром по ограничивающему прямоугольнику.

Параметры
bbox – Ограничивающий прямоугольник.

Результат
Итератор по записям.

Список 15: Пример запроса (таблица в проекции Робинсона)

```
table_world = provider_manager.openfile(filepath)
v = 2000000
items = table_world.itemsInRect(Rect(-v, -v, v, v))
for f in items:
    print('Feature id={}. Страна={}'.format(f.id, f['Страна']))
```

property name: str

Название объекта данных.

property properties: dict

Дополнительные свойства объекта данных.

property provider: str

Провайдер изначального источника данных.

redo(steps: int = 1)

Производит откат вперед на заданное количество шагов. При этом возвращается состояние до последней отмены.

Параметры
steps – Количество шагов.

remove(ids: Union[int, Iterator[int]])

Удаляет записи из таблицы.

Параметры
ids – Идентификаторы записей для удаления.

rollback()

Отменяет несохраненные изменения в таблице.

Если таблица не содержит несохраненные изменения, то команда игнорируется.

property schema: Schema

Схема таблицы.

property schema_changed: SignalInstance

Сигнал об изменении схемы таблицы. Испускается когда была изменена структура таблицы.

property supported_operations: SupportedOperations

Доступные операции.

Список 16: Пример использования

```
flags = table.supported_operations
assert flags == SupportedOperations.ReadWrite
assert flags & SupportedOperations.Insert
assert SupportedOperations.Write in flags
```

undo(steps: int = 1)

Производит откат вперед на заданное количество шагов. При этом возвращается состояние до последней отмены.

Параметры
steps – Количество шагов.

update(features: Union[Feature, Iterable[Feature]])

Обновляет записи в таблице.

Параметры
features – Записи для обновления.

При обновлении проверяется Feature.id. Если запись с таким идентификатором не найдена, то она пропускается.

Список 17: Пример использования

```
modified_feature = Feature({'attr_name': 'new_value'}, id=1)
table.update(modified_feature)
table.commit()
```

См.также:

Feature.id, commit(), is_modified.

22.5 QueryTable - SQL запрос.

class axipy.**QueryTable**

Базовые классы: Table

Таблица, построенная на основе SQL запроса.

Пример:

```
table = provider_manager.openfile('world.tab')
query = data_manager.query('select * from world')
```

Свойства:

can_redo	Возможен ли откат на один шаг вперед.
can_undo	Возможен ли откат на один шаг назад.
coordsystem	Система координат таблицы.
data_changed	Сигнал об изменении данных таблицы.
destroyed	Сигнал оповещения об удалении объекта.
hotlink	Наименование атрибута таблицы для хранения гиперссылки.
is_editable	Признак того, что таблица является редактируемой.
is_modified	Таблица содержит несохраненные изменения.
is_spatial	Признак того, что объект данных является пространственным.
is_temporary	Признак того, что таблица является временной.
name	Название объекта данных.
properties	Дополнительные свойства объекта данных.
provider	Провайдер изначального источника данных.
schema	Схема таблицы.
schema_changed	Сигнал об изменении схемы таблицы.
sql_text	Текст SQL запроса.
supported_operations	Доступные операции.

Методы:

close()	Пытается закрыть таблицу.
commit()	Сохраняет изменения в таблице.
count([bbox])	Возвращает количество записей, удовлетворяющих параметрам.
get_bounds()	Возвращает область, в которую попадают все данные таблицы.
insert(features)	Вставляет записи в таблицу.
items([bbox, ids])	Запрашивает записи, удовлетворяющие параметрам.
itemsByIds(ids)	Запрашивает записи по списку list с идентификаторами записей, либо перечень идентификаторов в виде списка.
itemsInObject(obj)	Запрашивает записи с фильтром по геометрическому объекту.
itemsInRect(bbox)	Запрашивает записи с фильтром по ограничивающему прямоугольнику.
redo([steps])	Производит откат вперед на заданное количество шагов.
remove(ids)	Удаляет записи из таблицы.
rollback()	Отменяет несохраненные изменения в таблице.
undo([steps])	Производит откат вперед на заданное количество шагов.
update(features)	Обновляет записи в таблице.

property can_redo: bool

Возможен ли откат на один шаг вперед.

property can_undo: bool

Возможен ли откат на один шаг назад.

close()

Пытается закрыть таблицу.

Исключение
RuntimeError – Ошибка закрытия таблицы.

Примечание: Объект данных не всегда может быть сразу закрыт. Например, для таблиц используется транзакционная модель редактирования и перед закрытием необходимо сохранить или отменить изменения, если они есть. См. Table.is_modified.

---

commit()

Сохраняет изменения в таблице.

Если таблица не содержит несохраненные изменения, то команда игнорируется.

Исключение
RuntimeError – Невозможно сохранить изменения.

property coordsystem: Optional[CoordSystem]

Система координат таблицы.

count(bbox: Union[Rect, QRectF, tuple] = None) → int

Возвращает количество записей, удовлетворяющих параметрам.

Данный метод является наиболее предпочтительным для оценки количества записей. При этом используется наиболее оптимальный вариант выполнения запроса для каждого конкретного провайдера данных.

Параметры

bbox – Ограничивающий прямоугольник.

Результат

Количество записей.

property data_changed: SignalInstance

Сигнал об изменении данных таблицы. Испускается когда были изменены данные таблицы. В качестве параметра передается дополнительная информация в виде dict.

Таблица 6: Доступные параметры

Наименование	Значение	Описание
operation	insert	Произведена вставка данных
operation	update	Данные были обновлены
operation	remove	Данные были удалены
operation	unknown	Тип операции не определен
ids	list	Перечень затронутых идентификаторов, если он доступен

Список 18: Пример подписки на изменение таблицы.

```
table = provider_manager.openfile(filepath)
table.data_changed.connect(lambda info: print(f'Таблица была изменена {info}
↪'))
```

property destroyed: SignalInstance

Сигнал оповещения об удалении объекта.

get_bounds() → Rect

Возвращает область, в которую попадают все данные таблицы.

property hotlink: str

Наименование атрибута таблицы для хранения гиперссылки.

Таблица 7: Возможны следующие варианты

Значение	Описание
axioma://world.tab	Открывает файл или рабочее пространство в аксиоме
addlayer://world	Добавляет слой world в текущую карту
exec://gimp	Запускает на выполнение программу gimp
https://axioma-gis.ru/	Открывает ссылку в браузере

Если префикс отсутствует, то производится попытка запустить по ассоциации.

См.также:

`axipy.render.VectorLayer.hotlink`.

insert(features: Union[Feature, Iterable[Feature]])

Вставляет записи в таблицу.

Параметры
features – Записи для вставки.

property is_editable: bool

Признак того, что таблица является редактируемой.

property is_modified: bool

Таблица содержит несохраненные изменения.

property is_spatial: bool

Признак того, что объект данных является пространственным.

property is_temporary: bool

Признак того, что таблица является временной. Это в первую очередь касается таблиц, созданных в памяти. А также таблица косметического слоя.

items(bbox: Union[Rect, QRectF, tuple] = None, ids: List[int] = None) → Iterator[Feature]

Запрашивает записи, удовлетворяющие параметрам. В качестве фильтра может быть указан либо ограничивающий прямоугольник, либо перечень идентификаторов в виде списка.

Параметры

- **bbox** – Ограничивающий прямоугольник.
- **ids** – Список идентификаторов.

Результат
Итератор по записям.

itemsByIds(ids: List[int]) → Iterator[Feature]

Запрашивает записи по списку `list` с идентификаторами записей, либо перечень идентификаторов в виде списка. Идентификаторы несохраненных записей имеют отрицательные значения.

Параметры
ids – Список идентификаторов.

Результат
Итератор по записям.

Список 19: Пример

```
table_world = provider_manager.openfile(filepath)
# Пример запроса по списку идентификаторов.
items = table_world.itemsByIds([11, 27, 41, 163, 203])
for f in items:
    print('Feature id={}. Страна={}'.format(f.id, f['Страна']))
# Просмотр идентификаторов всех записей, включая несохраненные
for f in table_world.items():
    print(f.id, f['Страна'])
# Получение несохраненных записей совместно с сохраненными
```

(continues on next page)

(продолжение с предыдущей страницы)

```
items_new = table_world.itemsByIds([-1, -12, 27])
for f in items_new:
    print('Feature id={}. Страна={}'.format(f.id, f['Страна']))
```

itemsInObject(obj: Geometry) → Iterator[Feature]

Запрашивает записи с фильтром по геометрическому объекту.

Параметры
obj – Геометрия. Если для нее не задана СК, используется СК таблицы.

Результат
Итератор по записям.

Список 20: Пример запроса по полигону

```
table_world = provider_manager.openfile(filepath)
v = 2000000
polygon = Polygon((-v, -v), (-v, v), (v, v), (v, -v))
items = table_world.itemsInObject(polygon)
for f in items:
    print('Feature id={}. Страна={}'.format(f.id, f['Страна']))
```

itemsInRect(bbox: Union[Rect, QRectF, tuple]) → Iterator[Feature]

Запрашивает записи с фильтром по ограничивающему прямоугольнику.

Параметры
bbox – Ограничивающий прямоугольник.

Результат
Итератор по записям.

Список 21: Пример запроса (таблица в проекции Робинсона)

```
table_world = provider_manager.openfile(filepath)
v = 2000000
items = table_world.itemsInRect(Rect(-v, -v, v, v))
for f in items:
    print('Feature id={}. Страна={}'.format(f.id, f['Страна']))
```

property name: str

Название объекта данных.

property properties: dict

Дополнительные свойства объекта данных.

property provider: str

Провайдер изначального источника данных.

redo(steps: int = 1)

Производит откат вперед на заданное количество шагов. При этом возвращается состояние до последней отмены.

Параметры
steps – Количество шагов.

remove(ids: Union[int, Iterator[int]])

Удаляет записи из таблицы.

Параметры
ids – Идентификаторы записей для удаления.

rollback()

Отменяет несохраненные изменения в таблице.

Если таблица не содержит несохраненные изменения, то команда игнорируется.

property schema: Schema

Схема таблицы.

property schema_changed: SignalInstance

Сигнал об изменении схемы таблицы. Испускается когда была изменена структура таблицы.

property sql_text: str

Текст SQL запроса.

property supported_operations: SupportedOperations

Доступные операции.

Список 22: Пример использования

```
flags = table.supported_operations
assert flags == SupportedOperations.ReadWrite
assert flags & SupportedOperations.Insert
assert SupportedOperations.Write in flags
```

undo(steps: int = 1)

Производит откат вперед на заданное количество шагов. При этом возвращается состояние до последней отмены.

Параметры
steps – Количество шагов.

update(features: Union[Feature, Iterable[Feature]])

Обновляет записи в таблице.

Параметры
features – Записи для обновления.

При обновлении проверяется Feature.id. Если запись с таким идентификатором не найдена, то она пропускается.

Список 23: Пример использования

```
modified_feature = Feature({'attr_name': 'new_value'}, id=1)
table.update(modified_feature)
table.commit()
```

См.также:

Feature.id, commit(), is_modified.

22.6 SelectionTable - Таблица с текущей выборкой.

class axipy.**SelectionTable**

Базовые классы: Table

Таблица, построенная на основе текущей выборки. Носит временный характер. Создается, когда в выборку добавляются объекты и удаляется, когда выборка очищается.

Пример доступа к выборке с получением идентификаторов:

```
for f in data_manager.selection.items():
    print('>>>', f.id)
```

Свойства:

base_table	Таблица, на которой основана данная выборка.
can_redo	Возможен ли откат на один шаг вперед.
can_undo	Возможен ли откат на один шаг назад.
coordsystem	Система координат таблицы.
data_changed	Сигнал об изменении данных таблицы.
destroyed	Сигнал оповещения об удалении объекта.
hotlink	Наименование атрибута таблицы для хранения гиперссылки.
is_editable	Признак того, что таблица является редактируемой.
is_modified	Таблица содержит несохраненные изменения.
is_spatial	Признак того, что объект данных является пространственным.
is_temporary	Признак того, что таблица является временной.
name	Название объекта данных.
properties	Дополнительные свойства объекта данных.
provider	Провайдер изначального источника данных.
schema	Схема таблицы.
schema_changed	Сигнал об изменении схемы таблицы.
supported_operations	Доступные операции.

Методы:

close()	Пытается закрыть таблицу.
commit()	Сохраняет изменения в таблице.
count([bbox])	Возвращает количество записей, удовлетворяющих параметрам.
get_bounds()	Возвращает область, в которую попадают все данные таблицы.
insert(features)	Вставляет записи в таблицу.
items([bbox, ids])	Запрашивает записи, удовлетворяющие параметрам.
itemsByIds(ids)	Запрашивает записи по списку list с идентификаторами записей, либо перечень идентификаторов в виде списка.
itemsInObject(obj)	Запрашивает записи с фильтром по геометрическому объекту.
itemsInRect(bbox)	Запрашивает записи с фильтром по ограничивающему прямоугольнику.
redo([steps])	Производит откат вперед на заданное количество шагов.
remove(ids)	Удаляет записи из таблицы.
rollback()	Отменяет несохраненные изменения в таблице.
undo([steps])	Производит откат вперед на заданное количество шагов.
update(features)	Обновляет записи в таблице.

property base_table: Table

Таблица, на которой основана данная выборка.

property can_redo: bool

Возможен ли откат на один шаг вперед.

property can_undo: bool

Возможен ли откат на один шаг назад.

close()

Пытается закрыть таблицу.

Исключение
RuntimeError – Ошибка закрытия таблицы.

Примечание: Объект данных не всегда может быть сразу закрыт. Например, для таблиц используется транзакционная модель редактирования и перед закрытием необходимо сохранить или отменить изменения, если они есть. См. Table.is_modified.

---

commit()

Сохраняет изменения в таблице.

Если таблица не содержит несохраненные изменения, то команда игнорируется.

Исключение
RuntimeError – Невозможно сохранить изменения.

property coordsystem: Optional[CoordSystem]

Система координат таблицы.

count(bbox: Union[Rect, QRectF, tuple] = None) → int

Возвращает количество записей, удовлетворяющих параметрам.

Данный метод является наиболее предпочтительным для оценки количества записей. При этом используется наиболее оптимальный вариант выполнения запроса для каждого конкретного провайдера данных.

Параметры

bbox – Ограничивающий прямоугольник.

Результат

Количество записей.

property data_changed: SignalInstance

Сигнал об изменении данных таблицы. Испускается когда были изменены данные таблицы. В качестве параметра передается дополнительная информация в виде dict.

Таблица 8: Доступные параметры

Наименование	Значение	Описание
operation	insert	Произведена вставка данных
operation	update	Данные были обновлены
operation	remove	Данные были удалены
operation	unknown	Тип операции не определен
ids	list	Перечень затронутых идентификаторов, если он доступен

Список 24: Пример подписки на изменение таблицы.

```
table = provider_manager.openfile(filepath)
table.data_changed.connect(lambda info: print(f'Таблица была изменена {info}
↪'))
```

property destroyed: SignalInstance

Сигнал оповещения об удалении объекта.

get_bounds() → Rect

Возвращает область, в которую попадают все данные таблицы.

property hotlink: str

Наименование атрибута таблицы для хранения гиперссылки.

Таблица 9: Возможны следующие варианты

Значение	Описание
axioma://world.tab	Открывает файл или рабочее пространство в аксиоме
addlayer://world	Добавляет слой world в текущую карту
exec://gimp	Запускает на выполнение программу gimp
https://axioma-gis.ru/	Открывает ссылку в браузере

Если префикс отсутствует, то производится попытка запустить по ассоциации.

См.также:

`axipy.render.VectorLayer.hotlink`.

insert(features: Union[Feature, Iterable[Feature]])

Вставляет записи в таблицу.

Параметры
features – Записи для вставки.

property is_editable: bool

Признак того, что таблица является редактируемой.

property is_modified: bool

Таблица содержит несохраненные изменения.

property is_spatial: bool

Признак того, что объект данных является пространственным.

property is_temporary: bool

Признак того, что таблица является временной. Это в первую очередь касается таблиц, созданных в памяти. А также таблица косметического слоя.

items(bbox: Union[Rect, QRectF, tuple] = None, ids: List[int] = None) → Iterator[Feature]

Запрашивает записи, удовлетворяющие параметрам. В качестве фильтра может быть указан либо ограничивающий прямоугольник, либо перечень идентификаторов в виде списка.

Параметры

- **bbox** – Ограничивающий прямоугольник.
- **ids** – Список идентификаторов.

Результат
Итератор по записям.

itemsByIds(ids: List[int]) → Iterator[Feature]

Запрашивает записи по списку `list` с идентификаторами записей, либо перечень идентификаторов в виде списка. Идентификаторы несохраненных записей имеют отрицательные значения.

Параметры
ids – Список идентификаторов.

Результат
Итератор по записям.

Список 25: Пример

```
table_world = provider_manager.openfile(filepath)
# Пример запроса по списку идентификаторов.
items = table_world.itemsByIds([11, 27, 41, 163, 203])
for f in items:
    print('Feature id={}. Страна={}'.format(f.id, f['Страна']))
# Просмотр идентификаторов всех записей, включая несохраненные
for f in table_world.items():
    print(f.id, f['Страна'])
# Получение несохраненных записей совместно с сохраненными
```

(continues on next page)

(продолжение с предыдущей страницы)

```
items_new = table_world.itemsByIds([-1, -12, 27])
for f in items_new:
    print('Feature id={}. Страна={}'.format(f.id, f['Страна']))
```

itemsInObject(obj: Geometry) → Iterator[Feature]

Запрашивает записи с фильтром по геометрическому объекту.

Параметры
obj – Геометрия. Если для нее не задана СК, используется СК таблицы.

Результат
Итератор по записям.

Список 26: Пример запроса по полигону

```
table_world = provider_manager.openfile(filepath)
v = 2000000
polygon = Polygon((-v, -v), (-v, v), (v, v), (v, -v))
items = table_world.itemsInObject(polygon)
for f in items:
    print('Feature id={}. Страна={}'.format(f.id, f['Страна']))
```

itemsInRect(bbox: Union[Rect, QRectF, tuple]) → Iterator[Feature]

Запрашивает записи с фильтром по ограничивающему прямоугольнику.

Параметры
bbox – Ограничивающий прямоугольник.

Результат
Итератор по записям.

Список 27: Пример запроса (таблица в проекции Робинсона)

```
table_world = provider_manager.openfile(filepath)
v = 2000000
items = table_world.itemsInRect(Rect(-v, -v, v, v))
for f in items:
    print('Feature id={}. Страна={}'.format(f.id, f['Страна']))
```

property name: str

Название объекта данных.

property properties: dict

Дополнительные свойства объекта данных.

property provider: str

Провайдер изначального источника данных.

redo(steps: int = 1)

Производит откат вперед на заданное количество шагов. При этом возвращается состояние до последней отмены.

Параметры
steps – Количество шагов.

remove(ids: Union[int, Iterator[int]])

Удаляет записи из таблицы.

Параметры
ids – Идентификаторы записей для удаления.

rollback()

Отменяет несохраненные изменения в таблице.

Если таблица не содержит несохраненные изменения, то команда игнорируется.

property schema: Schema

Схема таблицы.

property schema_changed: SignalInstance

Сигнал об изменении схемы таблицы. Испускается когда была изменена структура таблицы.

property supported_operations: SupportedOperations

Доступные операции.

Список 28: Пример использования

```
flags = table.supported_operations
assert flags == SupportedOperations.ReadWrite
assert flags & SupportedOperations.Insert
assert SupportedOperations.Write in flags
```

undo(steps: int = 1)

Производит откат вперед на заданное количество шагов. При этом возвращается состояние до последней отмены.

Параметры
steps – Количество шагов.

update(features: Union[Feature, Iterable[Feature]])

Обновляет записи в таблице.

Параметры
features – Записи для обновления.

При обновлении проверяется Feature.id. Если запись с таким идентификатором не найдена, то она пропускается.

Список 29: Пример использования

```
modified_feature = Feature({'attr_name': 'new_value'}, id=1)
table.update(modified_feature)
table.commit()
```

См.также:

Feature.id, commit(), is_modified.

22.7 CosmeticTable - Таблица с данными косметического слоя.

class axipy.**CosmeticTable**

Базовые классы: Table

Объект данных косметического слоя `axipy.render.CosmeticLayer`

Свойства:

can_redo	Возможен ли откат на один шаг вперед.
can_undo	Возможен ли откат на один шаг назад.
coordsystem	Система координат таблицы.
data_changed	Сигнал об изменении данных таблицы.
destroyed	Сигнал оповещения об удалении объекта.
hotlink	Наименование атрибута таблицы для хранения гиперссылки.
is_editable	Признак того, что таблица является редактируемой.
is_modified	Таблица содержит несохраненные изменения.
is_spatial	Признак того, что объект данных является пространственным.
is_temporary	Признак того, что таблица является временной.
name	Название объекта данных.
properties	Дополнительные свойства объекта данных.
provider	Провайдер изначального источника данных.
schema	Схема таблицы.
schema_changed	Сигнал об изменении схемы таблицы.
supported_operations	Доступные операции.

Методы:

close()	Пытается закрыть таблицу.
commit()	Сохраняет изменения в таблице.
count([bbox])	Возвращает количество записей, удовлетворяющих параметрам.
get_bounds()	Возвращает область, в которую попадают все данные таблицы.
insert(features)	Вставляет записи в таблицу.
items([bbox, ids])	Запрашивает записи, удовлетворяющие параметрам.
itemsByIds(ids)	Запрашивает записи по списку list с идентификаторами записей, либо перечень идентификаторов в виде списка.
itemsInObject(obj)	Запрашивает записи с фильтром по геометрическому объекту.
itemsInRect(bbox)	Запрашивает записи с фильтром по ограничивающему прямоугольнику.
redo([steps])	Производит откат вперед на заданное количество шагов.
remove(ids)	Удаляет записи из таблицы.
rollback()	Отменяет несохраненные изменения в таблице.
undo([steps])	Производит откат вперед на заданное количество шагов.
update(features)	Обновляет записи в таблице.

property can_redo: bool

Возможен ли откат на один шаг вперед.

property can_undo: bool

Возможен ли откат на один шаг назад.

close()

Пытается закрыть таблицу.

Исключение
RuntimeError – Ошибка закрытия таблицы.

Примечание: Объект данных не всегда может быть сразу закрыт. Например, для таблиц используется транзакционная модель редактирования и перед закрытием необходимо сохранить или отменить изменения, если они есть. См. Table.is_modified.

---

commit()

Сохраняет изменения в таблице.

Если таблица не содержит несохраненные изменения, то команда игнорируется.

Исключение
RuntimeError – Невозможно сохранить изменения.

property coordsystem: Optional[CoordSystem]

Система координат таблицы.

count(bbox: Union[Rect, QRectF, tuple] = None) → int

Возвращает количество записей, удовлетворяющих параметрам.

Данный метод является наиболее предпочтительным для оценки количества записей. При этом используется наиболее оптимальный вариант выполнения запроса для каждого конкретного провайдера данных.

Параметры

bbox – Ограничивающий прямоугольник.

Результат

Количество записей.

property data_changed: SignalInstance

Сигнал об изменении данных таблицы. Испускается когда были изменены данные таблицы. В качестве параметра передается дополнительная информация в виде dict.

Таблица 10: Доступные параметры

Наименование	Значение	Описание
operation	insert	Произведена вставка данных
operation	update	Данные были обновлены
operation	remove	Данные были удалены
operation	unknown	Тип операции не определен
ids	list	Перечень затронутых идентификаторов, если он доступен

Список 30: Пример подписки на изменение таблицы.

```
table = provider_manager.openfile(filepath)
table.data_changed.connect(lambda info: print(f'Таблица была изменена {info}
↪'))
```

property destroyed: SignalInstance

Сигнал оповещения об удалении объекта.

get_bounds() → Rect

Возвращает область, в которую попадают все данные таблицы.

property hotlink: str

Наименование атрибута таблицы для хранения гиперссылки.

Таблица 11: Возможны следующие варианты

Значение	Описание
axioma://world.tab	Открывает файл или рабочее пространство в аксиоме
addlayer://world	Добавляет слой world в текущую карту
exec://gimp	Запускает на выполнение программу gimp
https://axioma-gis.ru/	Открывает ссылку в браузере

Если префикс отсутствует, то производится попытка запустить по ассоциации.

См.также:

`axipy.render.VectorLayer.hotlink`.

insert(features: Union[Feature, Iterable[Feature]])

Вставляет записи в таблицу.

Параметры
features – Записи для вставки.

property is_editable: bool

Признак того, что таблица является редактируемой.

property is_modified: bool

Таблица содержит несохраненные изменения.

property is_spatial: bool

Признак того, что объект данных является пространственным.

property is_temporary: bool

Признак того, что таблица является временной. Это в первую очередь касается таблиц, созданных в памяти. А также таблица косметического слоя.

items(bbox: Union[Rect, QRectF, tuple] = None, ids: List[int] = None) → Iterator[Feature]

Запрашивает записи, удовлетворяющие параметрам. В качестве фильтра может быть указан либо ограничивающий прямоугольник, либо перечень идентификаторов в виде списка.

Параметры

- **bbox** – Ограничивающий прямоугольник.
- **ids** – Список идентификаторов.

Результат
Итератор по записям.

itemsByIds(ids: List[int]) → Iterator[Feature]

Запрашивает записи по списку `list` с идентификаторами записей, либо перечень идентификаторов в виде списка. Идентификаторы несохраненных записей имеют отрицательные значения.

Параметры
ids – Список идентификаторов.

Результат
Итератор по записям.

Список 31: Пример

```
table_world = provider_manager.openfile(filepath)
# Пример запроса по списку идентификаторов.
items = table_world.itemsByIds([11, 27, 41, 163, 203])
for f in items:
    print('Feature id={}. Страна={}'.format(f.id, f['Страна']))
# Просмотр идентификаторов всех записей, включая несохраненные
for f in table_world.items():
    print(f.id, f['Страна'])
# Получение несохраненных записей совместно с сохраненными
```

(continues on next page)

(продолжение с предыдущей страницы)

```
items_new = table_world.itemsByIds([-1, -12, 27])
for f in items_new:
    print('Feature id={}. Страна={}'.format(f.id, f['Страна']))
```

itemsInObject(obj: Geometry) → Iterator[Feature]

Запрашивает записи с фильтром по геометрическому объекту.

Параметры
obj – Геометрия. Если для нее не задана СК, используется СК таблицы.

Результат
Итератор по записям.

Список 32: Пример запроса по полигону

```
table_world = provider_manager.openfile(filepath)
v = 2000000
polygon = Polygon((-v, -v), (-v, v), (v, v), (v, -v))
items = table_world.itemsInObject(polygon)
for f in items:
    print('Feature id={}. Страна={}'.format(f.id, f['Страна']))
```

itemsInRect(bbox: Union[Rect, QRectF, tuple]) → Iterator[Feature]

Запрашивает записи с фильтром по ограничивающему прямоугольнику.

Параметры
bbox – Ограничивающий прямоугольник.

Результат
Итератор по записям.

Список 33: Пример запроса (таблица в проекции Робинсона)

```
table_world = provider_manager.openfile(filepath)
v = 2000000
items = table_world.itemsInRect(Rect(-v, -v, v, v))
for f in items:
    print('Feature id={}. Страна={}'.format(f.id, f['Страна']))
```

property name: str

Название объекта данных.

property properties: dict

Дополнительные свойства объекта данных.

property provider: str

Провайдер изначального источника данных.

redo(steps: int = 1)

Производит откат вперед на заданное количество шагов. При этом возвращается состояние до последней отмены.

Параметры
steps – Количество шагов.

remove(ids: Union[int, Iterator[int]])

Удаляет записи из таблицы.

Параметры
ids – Идентификаторы записей для удаления.

rollback()

Отменяет несохраненные изменения в таблице.

Если таблица не содержит несохраненные изменения, то команда игнорируется.

property schema: Schema

Схема таблицы.

property schema_changed: SignalInstance

Сигнал об изменении схемы таблицы. Испускается когда была изменена структура таблицы.

property supported_operations: SupportedOperations

Доступные операции.

Список 34: Пример использования

```
flags = table.supported_operations
assert flags == SupportedOperations.ReadWrite
assert flags & SupportedOperations.Insert
assert SupportedOperations.Write in flags
```

undo(steps: int = 1)

Производит откат вперед на заданное количество шагов. При этом возвращается состояние до последней отмены.

Параметры
steps – Количество шагов.

update(features: Union[Feature, Iterable[Feature]])

Обновляет записи в таблице.

Параметры
features – Записи для обновления.

При обновлении проверяется Feature.id. Если запись с таким идентификатором не найдена, то она пропускается.

Список 35: Пример использования

```
modified_feature = Feature({'attr_name': 'new_value'}, id=1)
table.update(modified_feature)
table.commit()
```

См.также:

Feature.id, commit(), is_modified.

22.8 SupportedOperations - Доступные операции

class axipy.**SupportedOperations**

Флаги доступных операций.

Реализованы как перечисление enum.IntFlag и поддерживают побитовые операции.

Атрибут	Значение
Empty	0
Insert	1
Update	2
Delete	4
Write	Insert | Update | Delete = 7
Read	8
ReadWrite	Read | Write = 15

Список 36: Пример использования

```
flags = table.supported_operations
assert flags == SupportedOperations.ReadWrite
assert flags & SupportedOperations.Insert
assert SupportedOperations.Write in flags
```

См.также:

enum.IntFlag, axipy.Table.supported_operations

22.9 Feature - Запись в таблице

class axipy.**Feature**

Запись в таблице.

Работа с записью похожа на работу со словарем dict. Но также допускает обращение по индексу.

Список 37: Примеры.

```
feature = Feature({'attr_name': 'value'}, geometry=Point(10, 10),␣
↪style=PointStyle.create_mi_compat(35, 0))
# Количество атрибутов
count = len(feature)
# Запись значения по ключу
feature['attr_name'] = 'new_value'
# Запись значения по индексу
feature[0] = 'another_value'
# Чтение значения по ключу
value = feature['attr_name']
# Чтение значения по индексу
another_value = feature[0]
# Проверка наличия атрибута по ключу
'attr_name' in feature
```

(continues on next page)

(продолжение с предыдущей страницы)

```
# Проверка наличия атрибута по индексу
5 in feature
# Значения атрибутов можно задать словарем или именованными аргументами:
feature2 = Feature({'name1': 'value1', 'name2': 'value2'})
# Это эквивалентно
feature2 = Feature(name1='value1', name2='value2')
# Получение стиля оформления для геометрии
style = feature.style
# Установка нового стиля для геометрии
feature.style = style
# Получение геометрии
point = feature.geometry
# Установка нового значения для геометрии
feature.geometry = Point(20, 20)
# Просмотр всех наименований и значений атрибутов
for key, value in feature.items():
    print('{} = {}'.format(key, value))
'''
>>> attr_name = value
>>> +geometry = Point pos=(10.0 10.0)
>>> +style = PointStyle Symbol (35, 0, 8)
'''
```

Параметры

- **properties** – Значения атрибутов.
- **geometry** – Геометрия.
- **style** – Стиль.
- **id** – Идентификатор.
- ****kwargs** – Значения атрибутов.

Примечание: Для доступа к геометрическому атрибуту и стилю по наименованию можно использовать предопределенные идентификаторы +geometry и +style соответственно:

- GEOMETRY_ATTR=+geometry
- STYLE_ATTR=+style

Конструктор класса:

`__init__`([properties, geometry, style, id])	Создает экземпляр класса.

Свойства:

`geometry`	Геометрия записи.
`id`	Идентификатор записи в таблице.
`style`	Стиль записи.

Методы:

get(key[, default])	Возвращает значение заданного атрибута.
has_geometry()	Проверяет, имеет ли запись атрибут с геометрией.
has_style()	Проверяет, имеет ли запись атрибут со стилем.
items()	Возвращает список пар имя - значение.
keys()	Возвращает список имен атрибутов.
to_geojson()	Представляет запись в виде, похожем на 'GeoJSON'.
values()	Возвращает список значений атрибутов.

__init__(properties: dict = dict(), geometry: Optional[Geometry] = None, style: Optional[Style] = None, id: Optional[int] = None, **kwargs)

Создает экземпляр класса.

property geometry: Optional[Geometry]

Геометрия записи.

См.также:

Feature.has_geometry(), GEOMETRY_ATTR

Результат

Значение геометрического атрибута; или None, если значение пустое или отсутствует.

get(key: str, default: Optional[Any] = None)

Возвращает значение заданного атрибута.

Параметры

- **key** – Имя атрибута.
- **default** – Значение по умолчанию.

Результат

Искомое значение, или значение по умолчанию, если заданный атрибут отсутствует.

has_geometry() → bool

Проверяет, имеет ли запись атрибут с геометрией.

has_style() → bool

Проверяет, имеет ли запись атрибут со стилем.

property id: int

Идентификатор записи в таблице.

Несохраненные записи в таблице будут иметь отрицательное значение.

См.также:

Table.is_modified, Table.commit()

Результат

0 если идентификатор не задан.

items() → List[tuple]
Возвращает список пар имя - значение.

keys() → List[str]
Возвращает список имен атрибутов.

property style: Optional[Style]
Стиль записи.

См.также:

Feature.has_style(), STYLE_ATTR

Результат
Значение атрибута со стилем; или None, если значение пустое или отсутствует.

to_geojson() → dict
Представляет запись в виде, похожем на „GeoJSON“.

values() → List
Возвращает список значений атрибутов.

22.10 Schema - Схема таблицы

class axipy.**Schema**
Базовые классы: list

Схема таблицы. Представляет собой список атрибутов axipy.Attribute. Организован в виде list и свойством coordsystem. При задании coordsystem создается геометрический атрибут.

Параметры

- ***attributes** – Атрибуты.
- **coordsystem** – Система координат для геометрического атрибута. Может быть задана или в виде строки (подробнее axipy.cs.CoordSystem.from_string()) или как объект СК axipy.cs.CoordSystem.

Список 38: Пример создания

```
schema = Schema(
    Attribute.string('one', 25),
    Attribute.integer('two'),
    Attribute.decimal('three'),
    coordsystem='prj:Earth Projection 12, 62, "m", 0'
)
```

Список 39: Пример создания из списка

```
attrs = [Attribute.string('one', 25), Attribute.integer(
    'two'), Attribute.decimal('three')]
# распаковывается список атрибутов
schema = Schema(*attrs, coordsystem='prj:Earth Projection 12, 62, "m", 0')
```

Имеет стандартные функции работы со списком.

Список 40: Пример операций

```
count = len(schema)  # количество атрибутов
attribute = schema[2]  # получение атрибута по индексу
schema[2] = Attribute.string('attr', 30)  # изменяет
del schema[2]  # удаляет
schema.append(Attribute.string('new_attr'))  # добавляет в конец
schema.insert(2, Attribute.integer('num_attr'))  # добавляет по индексу
index = schema.index('new_attr')  # ищет по имени
assert 'new_attr' in schema  # проверяет существование
schema.remove('new_attr')  # удаляет по имени
a = schema.by_name('attr')  # Получение атрибута по его имени
```

Конструктор класса:

__init__(*attributes[, coordsystem])	Создает экземпляр класса.

Свойства:

attribute_names	Возвращает список имен атрибутов.
coordsystem	Система координат.

Методы:

by_name(n)	Возвращает атрибут по его имени.
index_by_name(n)	Возвращает индекс по имени атрибута
insert(index, attr)	Вставляет атрибут.

__init__(*attributes: Attribute, coordsystem: Optional[Union[str, CoordSystem]] = None)

Создает экземпляр класса.

property attribute_names: List[str]

Возвращает список имен атрибутов.

by_name(n: str) → Attribute

Возвращает атрибут по его имени. Если такого имени не существует, возвращает None.

Параметры
n – Наименование атрибута.

property coordsystem: Optional[CoordSystem]

Система координат.

Результат
None, если СК отсутствует.

См.также:

axipy.Table.is_spatial

Список 41: Пример использования

```
if schema.coordsystem is not None:  # проверяет существование
    pass
crs_string = schema.coordsystem  # получает
schema.coordsystem = 'epsg:4326'  # изменяет
schema.coordsystem = None  # удаляет
```

index_by_name(n: str) → int

Возвращает индекс по имени атрибута

Параметры
n – Наименование атрибута.

insert(index: int, attr: Attribute)

Вставляет атрибут.

Параметры

- **index** – Индекс, по которому производится вставка.
- **attr** – Атрибут.

22.11 Attribute - Атрибут схемы таблицы

class axipy.**Attribute**

Атрибут схемы таблицы.

Используется для создания и инспектирования атрибутов и схем axipy.Schema. Для создания атрибутов используйте функции string(), decimal() и другие.

Параметры

- **name** – Название.
- **typedef** – Описание типа.

Список 42: Пример создания

```
string_attr = Attribute.string('attribute_name', 80)
```

Конструктор класса:

__init__(name, typedef)	Создает экземпляр класса.

Классовые методы:

bool(name)	Создает атрибут логического типа.
date(name)	Создает атрибут типа дата.
datetime(name)	Создает атрибут типа дата и время.
decimal(name[, length, precision])	Создает атрибут десятичного типа.
double(name)	Создает атрибут вещественного типа.
float(name)	Создает атрибут вещественного типа.
integer(name)	Создает атрибут целого типа.
string(name[, length])	Создает атрибут строкового типа.
time(name)	Создает атрибут типа время.

Свойства:

length	Длина атрибута.
name	Имя атрибута.
precision	Точность.
type_string	Тип в виде строки без длины и точности.
typedef	Описание типа.

__init__(name: str, typedef: str)

Создает экземпляр класса.

static bool(name: str) → Attribute

Создает атрибут логического типа.

Параметры
name – Имя атрибута.

static date(name: str) → Attribute

Создает атрибут типа дата.

Параметры
name – Имя атрибута.

static datetime(name: str) → Attribute

Создает атрибут типа дата и время.

Параметры
name – Имя атрибута.

static decimal(name: str, length: int = DEFAULT_DECIMAL_LENGTH, precision: int = DEFAULT_DECIMAL_PRECISION) → Attribute

Создает атрибут десятичного типа.

Параметры

- **name** – Имя атрибута.
- **length** – Длина атрибута. Количество символов, включая запятую.
- **precision** – Число знаков после запятой.

static double(name: str) → Attribute

Создает атрибут вещественного типа.

Параметры
name – Имя атрибута.

static float(name: str) → Attribute

Создает атрибут вещественного типа.

То же, что и `double()`

Параметры
name – Имя атрибута.

static integer(name: str) → Attribute

Создает атрибут целого типа.

Параметры
name – Имя атрибута.

property length: int

Длина атрибута.

property name: str

Имя атрибута.

property precision: int

Точность.

static string(name: str, length: int = DEFAULT_STRING_LENGTH) → Attribute

Создает атрибут строкового типа.

Параметры

- **name** – Имя атрибута.
- **length** – Длина атрибута.

static time(name: str) → Attribute

Создает атрибут типа время.

Параметры
name – Имя атрибута.

property type_string: str

Тип в виде строки без длины и точности.

property typedef: str

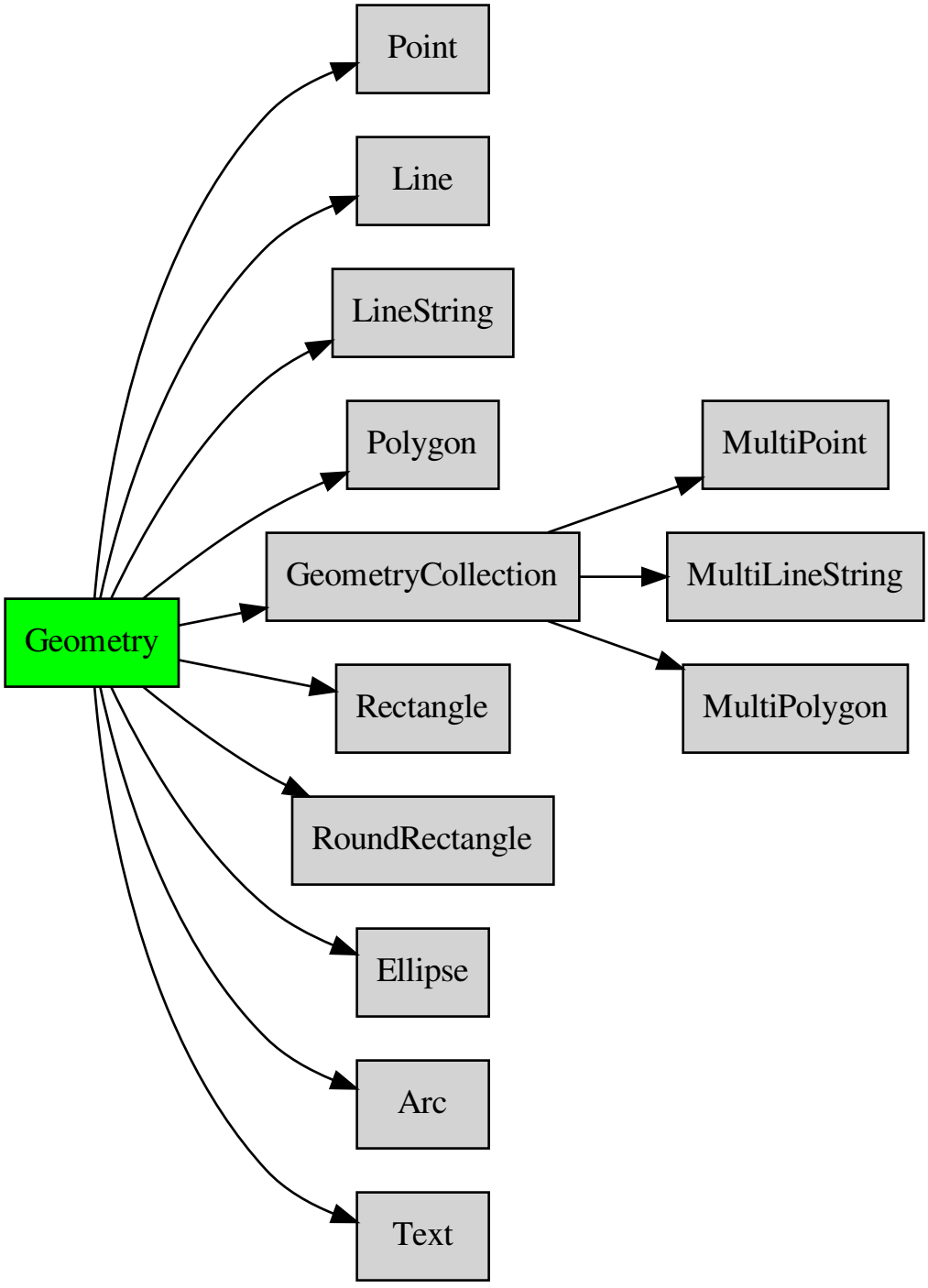
Описание типа.

Строка вида `<тип>[:длина][.точность]`.

22.12 Geometry - Геометрия

22.12.1 Geometry - Геометрия

Иерархия геометрических классов:



class axipy.**Geometry**

Абстрактный класс геометрического объекта (геометрии).

Примечание: Для получения краткого текстового представления геометрии можно воспользоваться функцией `str`. Для более полного - `repr()`.

Классовые методы:

`distance_by_points`(start, end[, cs])	Производит расчет расстояния между двумя точками и азимут от первой до второй точки.
`from_geojson`(json[, cs])	Возвращает геометрию из ее 'GeoJSON' представления.
`from_mif`(mif)	Возвращает геометрию из ее 'MIF' представления.
`from_wkb`(wkb[, coordsystem])	Создает геометрический объект из строки формата WKB.
`from_wkt`(wkt[, coordsystem])	Создает геометрический объект из строки формата WKT.
`point_by_azimuth`(point, azimuth, distance[, cs])	Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

Свойства:

`bounds`	Возвращает минимальный ограничивающий прямоугольник.
`coordsystem`	Система Координат (СК) геометрии.
`is_valid`	Проверяет геометрию на валидность.
`is_valid_reason`	Если геометрия неправильная, возвращает краткую аннотацию причины.
`name`	Возвращает наименование геометрического объекта.
`type`	Возвращает тип геометрического элемента.

Методы:

`affine_transform`(trans)	Трансформирует объект исходя из заданных параметров трансформации.
`almost_equals`(other, tolerance)	Производит примерное сравнения с другой геометрией в пределах заданной точности.
`boundary`()	Возвращает границы геометрии в виде полилинии.
`buffer`(distance[, resolution, capStyle, ...])	Производит построение буфера вокруг объекта.
`centroid`()	Возвращает центроид геометрии.

continues on next page

Таблица 12 – продолжение с предыдущей страницы

clone()	Создает копию объекта.
contains(other)	Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.
convex_hull()	Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.
covers(other)	Возвращает True, если геометрия охватывает геометрию other.
crosses(other)	Возвращает True, если при пересечении геометрий объекты частично пересекаются.
difference(other)	Возвращает область первой геометрии, которая не пересечена второй геометрией.
disjoint(other)	Возвращает True, если геометрии не пересекаются и не соприкасаются.
envelope()	Возвращает полигон, описывающий заданную геометрию.
equals(other)	Производит сравнение с другой геометрией.
get_area([area_unit])	Рассчитывает площадь, если объект площадной.
get_distance(other[, u])	Производит расчет расстояния до объекта other.
get_length([u])	Рассчитывает длину геометрии.
get_perimeter([u])	Рассчитывает периметр геометрии.
intersection(other)	Возвращает область пересечения с другой геометрией.
intersects(other)	Возвращает True, если геометрии пересекаются.
overlaps(other)	Возвращает True, если пересечение геометрий отличается от обеих геометрий.
relate(other)	Проверяет отношения между объектами.
reproject(cs)	Перепроецирует геометрию в другую систему координат.
rotate(point, angle)	Поворот геометрии относительно заданной точки.
scale(kx, ky)	Масштабирует объект по заданным коэффициентам масштабирования.
shift(dx, dy)	Смещает объект на заданную величину.
split_by_polyline(splitter)	Разрезает объект линейным объектом.
symmetric_difference(other)	Возвращает логический XOR областей геометрий (объединение разниц).
to_geojson()	Преобразует геометрию в формат 'GeoJSON'
to_linestring()	Пробует геометрию преобразовать в линейный объект.
to_mif()	Преобразует геометрию в формат MIF.

continues on next page

Таблица 12 – продолжение с предыдущей страницы

to_polygon()	Пробует геометрию преобразовать в площадной объект.
to_wkb()	Возвращает WKB строку для геометрии.
to_wkt()	Возвращает WKT строку для геометрии.
touches(other)	Возвращает True, если геометрии соприкасаются.
union(other)	Возвращает результат объединения двух геометрий.
within(other)	Возвращает True, если геометрия находится полностью внутри геометрии other.

affine_transform(trans: QTransform) → Geometry

Трансформирует объект исходя из заданных параметров трансформации.

См.также:

Для простых операций типа сдвиг, масштабирование и поворот, рекомендуется использовать shift(), scale(), rotate() соответственно.

Параметры
trans – Матрица трансформации.

almost_equals(other: Geometry, tolerance: float) → bool

Производит примерное сравнения с другой геометрией в пределах заданной точности.

Параметры

- **other** – Сравниваемый объект.
- **tolerance** – Точность сравнения.

Результат
Возвращает True, если геометрии равны в пределах заданного отклонения.

boundary() → Geometry

Возвращает границы геометрии в виде полилинии.

property bounds: Rect

Возвращает минимальный ограничивающий прямоугольник.

buffer(distance: float, resolution: int = 16, capStyle: LineCapStyle = LineCapStyle.Round, join_style: LineJoinStyle = LineJoinStyle.Round, mitreLimit: float = 5.0) → Geometry

Производит построение буфера вокруг объекта.

Параметры

- **distance** – Ширина буфера.
- **resolution** – Количество сегментов на квадрант.
- **capStyle** – Стиль окончания.
- **join_style** – Стиль соединения.

- **mitreLimit** – Предел среза.

centroid() → Geometry

Возвращает центроид геометрии.

clone() → Geometry

Создает копию объекта.

contains(other: Geometry) → bool

Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.

convex_hull() → Geometry

Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.

property coordsystem: CoordSystem

Система Координат (СК) геометрии.

covers(other: Geometry) → bool

Возвращает True, если геометрия охватывает геометрию other.

crosses(other: Geometry) → bool

Возвращает True, если при пересечении геометрий объекты частично пересекаются.

difference(other: Geometry) → Geometry

Возвращает область первой геометрии, которая не пересечена второй геометрией.

disjoint(other: Geometry) → bool

Возвращает True, если геометрии не пересекаются и не соприкасаются.

static distance_by_points(start: Union[Pnt, Tuple[float, float]], end: Union[Pnt, Tuple[float, float]], cs: Optional[CoordSystem] = None) → Tuple

Производит расчет расстояния между двумя точками и азимут от первой до второй точки.

См.также:

Обратная функция `point_by_azimuth()`

Параметры

- **start** – Начальная точка. Если задана СК, то координаты в ней.
- **end** – Конечная точка. Если задана СК, то координаты в ней.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращается пара значений (расстояние в метрах, азимут в градусах)

envelope() → Geometry

Возвращает полигон, описывающий заданную геометрию.

equals(other: Geometry) → bool

Производит сравнение с другой геометрией.

Параметры
other – Сравниваемая геометрия.

Результат
Возвращает True, если геометрии равны.

static from_geojson(json: str, cs: Optional[CoordSystem] = None) → Geometry

Возвращает геометрию из ее „GeoJSON“ представления.

Параметры

- **json** – json представление в виде строки.
- **cs** – Система координат.

static from_mif(mif: str) → Geometry

Возвращает геометрию из ее „MIF“ представления.

Параметры
mif – mif представление в виде строки.

static from_wkb(wkb: bytes, coordsystem: Optional[CoordSystem] = None) → Geometry

Создает геометрический объект из строки формата WKB.

Параметры

- **wkb** – Строка WKB
- **coordsystem** – Система координат, которая будет установлена для геометрии.

Список 43: Пример.

```
wkb = b'\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00$@\x00\x00\x00\x00\x00\x00
↪$@'
pnt = Geometry.from_wkb(wkb)
```

static from_wkt(wkt: str, coordsystem: Optional[CoordSystem] = None) → Geometry

Создает геометрический объект из строки формата WKT.

Параметры

- **wkt** – Строка WKT. Допустимо задание строки в формате с указанием SRID (EWKT). В данном случае система координат для создаваемой геометрии будет установлено исходя их этого значения.
- **coordsystem** – Система координат, которая будет установлена для геометрии. Если строка задана в виде EWKT и указано значение SRID, игнорируется.

Список 44: Пример.

```
polygon = Geometry.from_wkt('POLYGON ((10 10, 100 100, 100 10, 10 10))')
point = Geometry.from_wkt('SRID=4326;POINT (10 10)')
crs = CoordSystem.from_epsg(4326)
pline = Geometry.from_wkt('LINESTRING (30 10, 10 30, 40 40)', crs)
```

get_area(area_unit: Optional[AreaUnit] = None) → float

Рассчитывает площадь, если объект площадной. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в квадратных метрах.

Список 45: Пример.

```
# В проекции Меркатора:
csMercatorKm = CoordSystem.from_prj("10, 104, 1, 0")
poly = Polygon([(0, 0), (0, 2), (2, 2), (2, 0)], cs=csMercatorKm)
print('Area Mercator km:', poly.get_area())
print('Area Mercator m:', poly.get_area(Unit.sq_m))
print('Perimeter Mercator km:', poly.get_perimeter())
print('Perimeter Mercator m:', poly.get_perimeter(Unit.m))
# В Широте/Долготе
poly.coordsystem = CoordSystem.from_prj("1, 104")
print('Area LL m:', poly.get_area())
print('Area LL km:', poly.get_area(Unit.sq_km))
print('Perimeter LL m:', poly.get_perimeter())
print('Perimeter LL km:', poly.get_perimeter(Unit.km))
'''
>>> Area Mercator km: 4.017946986632519
>>> Area Mercator m: 4017946.9866325194
>>> Perimeter Mercator km: 8.017972048421552
>>> Perimeter Mercator m: 8017.972048421552
>>> Area LL m: 49452172514.0342
>>> Area LL km: 49452.1725140342
>>> Perimeter LL m: 889423.5067063896
>>> Perimeter LL km: 889.4235067063896
'''
```

Параметры

area_unit – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и она не задана, то производится расчет на плоскости.

get_distance(other: Geometry, u: Optional[LinearUnit] = None) → float

Производит расчет расстояния до объекта other. Результат возвращает в СК текущего объекта.

Параметры

- **other** – Анализируемый объект.
- **u** – Единицы измерения, в которых требуется получить результат.

Список 46: Пример.

```
# Создадим геометрию без СК
first = Point(0, 0)
second = Point(10, 0)
print('В плане:', first.get_distance(second))
# Установим разные СК для точек
first.coordsystem = CoordSystem.from_prj("10, 104, 7, 0")
second.coordsystem = CoordSystem.from_prj("1, 104")
print('На сфере м:', first.get_distance(second))
print('На сфере км:', first.get_distance(second, Unit.km))
```

(continues on next page)

(продолжение с предыдущей страницы)

```
'''
>>> В плане: 10.0
>>> На сфере м: 1111948.7428468117
>>> На сфере км: 1111.9487428468117
'''
```

get_length(u: Optional[LinearUnit] = None) → float

Рассчитывает длину геометрии. Метод применим только для линейных объектов. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Список 47: Пример.

```
# В проекции Меркатора:
csMercatorKm = CoordSystem.from_prj("10, 104, 1, 0")
ls = Line((0, 0), (10, 0), csMercatorKm)
print('Length Mercator km:', ls.get_length())
print('Lenght Mercator m:', ls.get_length(Unit.m))
# В Широте/Долготе
csLL = CoordSystem.from_prj("1, 104")
ls = Line((0, 0), (10, 0), csLL)
print('Length LL m:', ls.get_length())
print('Length LL km:', ls.get_length(Unit.km))
'''
>>> Length Mercator km: 9.988805508567783
>>> Lenght Mercator m: 9988.805508567782
>>> Length LL m: 1111948.7428468117
>>> Length LL km: 1111.9487428468117
'''
```

Параметры

u – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и она не задана, то производится расчет на плоскости.

get_perimeter(u: Optional[LinearUnit] = None) → float

Рассчитывает периметр геометрии. Метод применим только для площадных объектов. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Пример см. `get_area()`

Параметры

u – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и она не задана, то производится расчет на плоскости.

intersection(other: Geometry) → Geometry

Возвращает область пересечения с другой геометрией.

intersects(other: Geometry) → bool

Возвращает True, если геометрии пересекаются.

property is_valid: bool

Проверяет геометрию на валидность.

property is_valid_reason: str

Если геометрия неправильная, возвращает краткую аннотацию причины.

property name: str

Возвращает наименование геометрического объекта.

overlaps(other: Geometry) → bool

Возвращает True, если пересечение геометрий отличается от обеих геометрий.

static point_by_azimuth(point: Union[Pnt, Tuple[float, float]], azimuth: float, distance: float, cs: Optional[CoordSystem] = None) → Pnt

Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

См.также:

Обратная функция `distance_by_points()`

Параметры

- **point** – Точка, относительно которой производится расчет. Если задана СК, то в ней.
- **azimuth** – Азимут в градусах, указывающий направление.
- **distance** – Расстояние по азимуту. Задается в метрах.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращает результирующую точку.

relate(other: Geometry) → str

Проверяет отношения между объектами. Подробнее см. DE-9IM

reproject(cs: CoordSystem) → Geometry

Перепроецирует геометрию в другую систему координат.

Параметры

cs – СК, в которой требуется получить объект.

rotate(point: Union[Pnt, Tuple[float, float]], angle: float) → Geometry

Поворот геометрии относительно заданной точки. Поворот производится против часовой стрелки.

Параметры

- **point** – Точка, вокруг которой необходимо произвести поворот.
- **angle** – Угол поворота в градусах.

scale(kx: float, ky: float) → Geometry

Масштабирует объект по заданным коэффициентам масштабирования. После выполнения операции центр результирующего объекта остается прежним.

Параметры

- **kx** – Масштабирование по координате X.
- **ky** – Масштабирование по координате Y.

shift(dx: float, dy: float) → Geometry

Смещает объект на заданную величину. Значения задаются в текущей проекции.

Параметры

- **dx** – Сдвиг по координате X.
- **dy** – Сдвиг по координате Y.

split_by_polyline(splitter: Geometry) → Optional[Geometry]

Разрезает объект линейным объектом.

Параметры
splitter – Геометрический объект, которым будет производиться разрезание объекта.

Результат
Возвращает полученный результат

```
poly = Polygon((0,10), (10,10), (10, 0))
line = Line((-5,5), (20,5))
r = poly.split_by_polyline(line)
print(r.wkt)
'''
>>> GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5, 5 5, 0 10)), POLYGON␣
↪((10 5, 10 0, 5 5, 10 5))) GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5,␣
↪5 5, 0 10)), POLYGON ((10 5, 10 0, 5 5, 10 5)))
'''
```

symmetric_difference(other: Geometry) → Geometry

Возвращает логический XOR областей геометрий (объединение разниц).

Параметры
other – Геометрия для анализа.

to_geojson() → str

Преобразует геометрию в формат „GeoJSON“

to_linestring() → Optional[Geometry]

Пробует геометрию преобразовать в линейный объект. В случае неудачи возвращает None.

to_mif() → str

Преобразует геометрию в формат MIF.

to_polygon() → Optional[Geometry]

Пробует геометрию преобразовать в площадной объект. В случае неудачи возвращает None.

to_wkb() → bytes

Возвращает WKB строку для геометрии.

to_wkt() → str

Возвращает WKT строку для геометрии.

touches(other: Geometry) → bool

Возвращает True, если геометрии соприкасаются.

property type: Type

Возвращает тип геометрического элемента.

Таблица 13: Возможные значения

Значение	Наименование
Unknown	Не определен
Point	Точка
Line	Линия
LineString	Полилиния
Polygon	Полигон
MultiPoint	Коллекция точек
MultiLineString	Коллекция полилиний
MultiPolygon	Коллекция полигонов
GeometryCollection	Смешанная коллекция
Arc	Дуга
Ellipse	Эллипс
Rectangle	Прямоугольник
RoundedRectangle	Скругленный прямоугольник
Text	Текст

Список 48: Пример.

```
point = Point(10, 10)
if point.type == GeometryType.Point:
    print('Это точка')
```

union(other: Geometry) → Geometry

Возвращает результат объединения двух геометрий.

within(other: Geometry) → bool

Возвращает True, если геометрия находится полностью внутри геометрии other.

22.12.2 Point - Точечный объект

class axipy.**Point**

Базовые классы: Geometry

Геометрический объект типа точка.

Параметры

- **x** – X координата
- **y** – Y координата
- **cs** – Система Координат, в которой создается геометрия.

Список 49: Пример.

```
cs = CoordSystem.from_prj("1, 104")
p = Point(23, 45, cs)  # Создадим точку.
p.x = 55  # Изменим значение координаты X
```

Конструктор класса:

__init__(x, y[, cs])	Создает экземпляр класса.

Классовые методы:

distance_by_points(start, end[, cs])	Производит расчет расстояния между двумя точками и азимут от первой до второй точки.
from_geojson(json[, cs])	Возвращает геометрию из ее 'GeoJSON' представления.
from_mif(mif)	Возвращает геометрию из ее 'MIF' представления.
from_wkb(wkb[, coordsystem])	Создает геометрический объект из строки формата WKB.
from_wkt(wkt[, coordsystem])	Создает геометрический объект из строки формата WKT.
point_by_azimuth(point, azimuth, distance[, cs])	Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

Свойства:

bounds	Возвращает минимальный ограничивающий прямоугольник.
coordsystem	Система Координат (СК) геометрии.
is_valid	Проверяет геометрию на валидность.
is_valid_reason	Если геометрия неправильная, возвращает краткую аннотацию причины.
name	Возвращает наименование геометрического объекта.
type	Возвращает тип геометрического элемента.
x	X Координата.
y	Y Координата.

Методы:

affine_transform(trans)	Трансформирует объект исходя из заданных параметров трансформации.
almost_equals(other, tolerance)	Производит примерное сравнения с другой геометрией в пределах заданной точности.
boundary()	Возвращает границы геометрии в виде полилинии.
buffer(distance[, resolution, capStyle, ...])	Производит построение буфера вокруг объекта.
centroid()	Возвращает центроид геометрии.
clone()	Создает копию объекта.

continues on next page

Таблица 14 – продолжение с предыдущей страницы

contains(other)	Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.
convex_hull()	Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.
covers(other)	Возвращает True, если геометрия охватывает геометрию other.
crosses(other)	Возвращает True, если при пересечении геометрий объекты частично пересекаются.
difference(other)	Возвращает область первой геометрии, которая не пересечена второй геометрией.
disjoint(other)	Возвращает True, если геометрии не пересекаются и не соприкасаются.
envelope()	Возвращает полигон, описывающий заданную геометрию.
equals(other)	Производит сравнение с другой геометрией.
get_area([area_unit])	Рассчитывает площадь, если объект площадной.
get_distance(other[, u])	Производит расчет расстояния до объекта other.
get_length([u])	Рассчитывает длину геометрии.
get_perimeter([u])	Рассчитывает периметр геометрии.
intersection(other)	Возвращает область пересечения с другой геометрией.
intersects(other)	Возвращает True, если геометрии пересекаются.
overlaps(other)	Возвращает True, если пересечение геометрий отличается от обеих геометрий.
relate(other)	Проверяет отношения между объектами.
reproject(cs)	Перепроецирует геометрию в другую систему координат.
rotate(point, angle)	Поворот геометрии относительно заданной точки.
scale(kx, ky)	Масштабирует объект по заданным коэффициентам масштабирования.
shift(dx, dy)	Смещает объект на заданную величину.
split_by_polyline(splitter)	Разрезает объект линейным объектом.
symmetric_difference(other)	Возвращает логический XOR областей геометрий (объединение разниц).
to_geojson()	Преобразует геометрию в формат 'GeoJSON'
to_linestring()	Пробует геометрию преобразовать в линейный объект.
to_mif()	Преобразует геометрию в формат MIF.

continues on next page

Таблица 14 – продолжение с предыдущей страницы

to_polygon()	Пробует геометрию преобразовать в площадной объект.
to_wkb()	Возвращает WKB строку для геометрии.
to_wkt()	Возвращает WKT строку для геометрии.
touches(other)	Возвращает True, если геометрии соприкасаются.
union(other)	Возвращает результат объединения двух геометрий.
within(other)	Возвращает True, если геометрия находится полностью внутри геометрии other.

__init__(x: float, y: float, cs: Optional[CoordSystem] = None)

Создает экземпляр класса.

affine_transform(trans: QTransform) → Geometry

Трансформирует объект исходя из заданных параметров трансформации.

См.также:

Для простых операций типа сдвиг, масштабирование и поворот, рекомендуется использовать shift(), scale(), rotate() соответственно.

Параметры
trans – Матрица трансформации.

almost_equals(other: Geometry, tolerance: float) → bool

Производит примерное сравнения с другой геометрией в пределах заданной точности.

Параметры

- **other** – Сравниваемый объект.
- **tolerance** – Точность сравнения.

Результат
Возвращает True, если геометрии равны в пределах заданного отклонения.

boundary() → Geometry

Возвращает границы геометрии в виде полилинии.

property bounds: Rect

Возвращает минимальный ограничивающий прямоугольник.

buffer(distance: float, resolution: int = 16, capStyle: LineCapStyle = LineCapStyle.Round, join_style: LineJoinStyle = LineJoinStyle.Round, mitreLimit: float = 5.0) → Geometry

Производит построение буфера вокруг объекта.

Параметры

- **distance** – Ширина буфера.
- **resolution** – Количество сегментов на квадрант.

- **capStyle** – Стиль окончания.
- **join_style** – Стиль соединения.
- **mitreLimit** – Предел среза.

centroid() → Geometry

Возвращает центроид геометрии.

clone() → Geometry

Создает копию объекта.

contains(other: Geometry) → bool

Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.

convex_hull() → Geometry

Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.

property coordsystem: CoordSystem

Система Координат (СК) геометрии.

covers(other: Geometry) → bool

Возвращает True, если геометрия охватывает геометрию other.

crosses(other: Geometry) → bool

Возвращает True, если при пересечении геометрий объекты частично пересекаются.

difference(other: Geometry) → Geometry

Возвращает область первой геометрии, которая не пересечена второй геометрией.

disjoint(other: Geometry) → bool

Возвращает True, если геометрии не пересекаются и не соприкасаются.

static distance_by_points(start: Union[Pnt, Tuple[float, float]], end: Union[Pnt, Tuple[float, float]], cs: Optional[CoordSystem] = None) → Tuple

Производит расчет расстояния между двумя точками и азимут от первой до второй точки.

См.также:

Обратная функция `point_by_azimuth()`

Параметры

- **start** – Начальная точка. Если задана СК, то координаты в ней.
- **end** – Конечная точка. Если задана СК, то координаты в ней.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращается пара значений (расстояние в метрах, азимут в градусах)

envelope() → Geometry

Возвращает полигон, описывающий заданную геометрию.

equals(other: Geometry) → bool

Производит сравнение с другой геометрией.

Параметры
other – Сравниваемая геометрия.

Результат
Возвращает True, если геометрии равны.

static from_geojson(json: str, cs: Optional[CoordSystem] = None) → Geometry

Возвращает геометрию из ее „GeoJSON“ представления.

Параметры

- **json** – json представление в виде строки.
- **cs** – Система координат.

static from_mif(mif: str) → Geometry

Возвращает геометрию из ее „MIF“ представления.

Параметры
mif – mif представление в виде строки.

static from_wkb(wkb: bytes, coordsystem: Optional[CoordSystem] = None) → Geometry

Создает геометрический объект из строки формата WKB.

Параметры

- **wkb** – Строка WKB
- **coordsystem** – Система координат, которая будет установлена для геометрии.

Список 50: Пример.

```
wkb = b'\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00$@\x00\x00\x00\x00\x00\x00
 ↪$@'
pnt = Geometry.from_wkb(wkb)
```

static from_wkt(wkt: str, coordsystem: Optional[CoordSystem] = None) → Geometry

Создает геометрический объект из строки формата WKT.

Параметры

- **wkt** – Строка WKT. Допустимо задание строки в формате с указанием SRID (EWKT). В данном случае система координат для создаваемой геометрии будет установлено исходя их этого значения.
- **coordsystem** – Система координат, которая будет установлена для геометрии. Если строка задана в виде EWKT и указано значение SRID, игнорируется.

Список 51: Пример.

```
polygon = Geometry.from_wkt('POLYGON ((10 10, 100 100, 100 10, 10 10))')
point = Geometry.from_wkt('SRID=4326;POINT (10 10)')
crs = CoordSystem.from_epsg(4326)
pline = Geometry.from_wkt('LINESTRING (30 10, 10 30, 40 40)', crs)
```

get_area(area_unit: Optional[AreaUnit] = None) → float

Рассчитывает площадь, если объект площадной. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в квадратных метрах.

Список 52: Пример.

```
# В проекции Меркатора:
csMercatorKm = CoordSystem.from_prj("10, 104, 1, 0")
poly = Polygon([(0, 0), (0, 2), (2, 2), (2, 0)], cs=csMercatorKm)
print('Area Mercator km:', poly.get_area())
print('Area Mercator m:', poly.get_area(Unit.sq_m))
print('Perimeter Mercator km:', poly.get_perimeter())
print('Perimeter Mercator m:', poly.get_perimeter(Unit.m))
# В Широте/Долготе
poly.coordsystem = CoordSystem.from_prj("1, 104")
print('Area LL m:', poly.get_area())
print('Area LL km:', poly.get_area(Unit.sq_km))
print('Perimeter LL m:', poly.get_perimeter())
print('Perimeter LL km:', poly.get_perimeter(Unit.km))
'''
>>> Area Mercator km: 4.017946986632519
>>> Area Mercator m: 4017946.9866325194
>>> Perimeter Mercator km: 8.017972048421552
>>> Perimeter Mercator m: 8017.972048421552
>>> Area LL m: 49452172514.0342
>>> Area LL km: 49452.1725140342
>>> Perimeter LL m: 889423.5067063896
>>> Perimeter LL km: 889.4235067063896
'''
```

Параметры

area_unit – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и она не задана, то производится расчет на плоскости.

get_distance(other: Geometry, u: Optional[LinearUnit] = None) → float

Производит расчет расстояния до объекта other. Результат возвращает в СК текущего объекта.

Параметры

- **other** – Анализируемый объект.
- **u** – Единицы измерения, в которых требуется получить результат.

Список 53: Пример.

```
# Создадим геометрию без СК
first = Point(0, 0)
second = Point(10, 0)
print('В плане:', first.get_distance(second))
# Установим разные СК для точек
first.coordsystem = CoordSystem.from_prj("10, 104, 7, 0")
second.coordsystem = CoordSystem.from_prj("1, 104")
print('На сфере м:', first.get_distance(second))
print('На сфере км:', first.get_distance(second, Unit.km))
'''
>>> В плане: 10.0
>>> На сфере м: 1111948.7428468117
>>> На сфере км: 1111.9487428468117
'''
```

get_length(u: Optional[LinearUnit] = None) → float

Рассчитывает длину геометрии. Метод применим только для линейных объектов. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Список 54: Пример.

```
# В проекции Меркатора:
csMercatorKm = CoordSystem.from_prj("10, 104, 1, 0")
ls = Line((0, 0), (10, 0), csMercatorKm)
print('Length Mercator km:', ls.get_length())
print('Lenght Mercator m:', ls.get_length(Unit.m))
# В Широте/Долготе
csLL = CoordSystem.from_prj("1, 104")
ls = Line((0, 0), (10, 0), csLL)
print('Length LL m:', ls.get_length())
print('Length LL km:', ls.get_length(Unit.km))
'''
>>> Length Mercator km: 9.988805508567783
>>> Lenght Mercator m: 9988.805508567782
>>> Length LL m: 1111948.7428468117
>>> Length LL km: 1111.9487428468117
'''
```

Параметры

u – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и она не задана, то производится расчет на плоскости.

get_perimeter(u: Optional[LinearUnit] = None) → float

Рассчитывает периметр геометрии. Метод применим только для площадных объектов. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Пример см. `get_area()`

Параметры

u – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и

она не задана, то производится расчет на плоскости.

intersection(other: Geometry) → Geometry

Возвращает область пересечения с другой геометрией.

intersects(other: Geometry) → bool

Возвращает True, если геометрии пересекаются.

property is_valid: bool

Проверяет геометрию на валидность.

property is_valid_reason: str

Если геометрия неправильная, возвращает краткую аннотацию причины.

property name: str

Возвращает наименование геометрического объекта.

overlaps(other: Geometry) → bool

Возвращает True, если пересечение геометрий отличается от обеих геометрий.

static point_by_azimuth(point: Union[Pnt, Tuple[float, float]], azimuth: float, distance: float, cs: Optional[CoordSystem] = None) → Pnt

Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

См.также:

Обратная функция `distance_by_points()`

Параметры

- **point** – Точка, относительно которой производится расчет. Если задана СК, то в ней.
- **azimuth** – Азимут в градусах, указывающий направление.
- **distance** – Расстояние по азимуту. Задается в метрах.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращает результирующую точку.

relate(other: Geometry) → str

Проверяет отношения между объектами. Подробнее см. DE-9IM

reproject(cs: CoordSystem) → Geometry

Перепроецирует геометрию в другую систему координат.

Параметры

cs – СК, в которой требуется получить объект.

rotate(point: Union[Pnt, Tuple[float, float]], angle: float) → Geometry

Поворот геометрии относительно заданной точки. Поворот производится против часовой стрелки.

Параметры

- **point** – Точка, вокруг которой необходимо произвести поворот.
- **angle** – Угол поворота в градусах.

scale(kx: float, ky: float) → Geometry

Масштабирует объект по заданным коэффициентам масштабирования. После выполнения операции центр результирующего объекта остается прежним.

Параметры

- **kx** – Масштабирование по координате X.
- **ky** – Масштабирование по координате Y.

shift(dx: float, dy: float) → Geometry

Смещает объект на заданную величину. Значения задаются в текущей проекции.

Параметры

- **dx** – Сдвиг по координате X.
- **dy** – Сдвиг по координате Y.

split_by_polyline(splitter: Geometry) → Optional[Geometry]

Разрезает объект линейным объектом.

Параметры
splitter – Геометрический объект, которым будет производиться разрезание объекта.

Результат
Возвращает полученный результат

```
poly = Polygon((0,10), (10,10), (10, 0))
line = Line((-5,5), (20,5))
r = poly.split_by_polyline(line)
print(r.wkt)
'''
>>> GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5, 5 5, 0 10)), POLYGON␣
↪((10 5, 10 0, 5 5, 10 5))) GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5,␣
↪5 5, 0 10)), POLYGON ((10 5, 10 0, 5 5, 10 5)))
'''
```

symmetric_difference(other: Geometry) → Geometry

Возвращает логический XOR областей геометрий (объединение разниц).

Параметры
other – Геометрия для анализа.

to_geojson() → str

Преобразует геометрию в формат „GeoJSON“

to_linestring() → Optional[Geometry]

Пробует геометрию преобразовать в линейный объект. В случае неудачи возвращает None.

to_mif() → str

Преобразует геометрию в формат MIF.

to_polygon() → Optional[Geometry]

Пробует геометрию преобразовать в площадной объект. В случае неудачи возвращает None.

to_wkb() → bytes

Возвращает WKB строку для геометрии.

to_wkt() → str

Возвращает WKT строку для геометрии.

touches(other: Geometry) → bool

Возвращает True, если геометрии соприкасаются.

property type: Type

Возвращает тип геометрического элемента.

Таблица 15: Возможные значения

Значение	Наименование
Unknown	Не определен
Point	Точка
Line	Линия
LineString	Полилиния
Polygon	Полигон
MultiPoint	Коллекция точек
MultiLineString	Коллекция полилиний
MultiPolygon	Коллекция полигонов
GeometryCollection	Смешанная коллекция
Arc	Дуга
Ellipse	Эллипс
Rectangle	Прямоугольник
RoundedRectangle	Скругленный прямоугольник
Text	Текст

Список 55: Пример.

```
point = Point(10, 10)
if point.type == GeometryType.Point:
    print('Это точка')
```

union(other: Geometry) → Geometry

Возвращает результат объединения двух геометрий.

within(other: Geometry) → bool

Возвращает True, если геометрия находится полностью внутри геометрии other.

property x: float

X Координата.

property y: float

Y Координата.

22.12.3 Line - Линия

class axipy.**Line**

Базовые классы: Geometry

Геометрический объект типа линия.

Параметры

- **begin** – Начальная точка линии.
- **end** – Конечная точка линии.
- **cs** – Система Координат, в которой создается геометрия.

Список 56: Пример.

```
cs = CoordSystem.from_prj("1, 104")
line = Line(
    Pnt(22, 44), (100, 101), cs
)  # Создадим линию с различным подходом при указании начальной о конечной точки
line.begin = (88, 99)  # Заменим координаты начальной точки.
line.end = Pnt(120, 120)
```

Конструктор класса:

__init__(begin, end[, cs])	Создает экземпляр класса.

Классовые методы:

distance_by_points(start, end[, cs])	Производит расчет расстояния между двумя точками и азимут от первой до второй точки.
from_geojson(json[, cs])	Возвращает геометрию из ее 'GeoJSON' представления.
from_mif(mif)	Возвращает геометрию из ее 'MIF' представления.
from_wkb(wkb[, coordsystem])	Создает геометрический объект из строки формата WKB.
from_wkt(wkt[, coordsystem])	Создает геометрический объект из строки формата WKT.
point_by_azimuth(point, azimuth, distance[, cs])	Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

Свойства:

begin	Начальная точка линии.
bounds	Возвращает минимальный ограничивающий прямоугольник.
coordsystem	Система Координат (СК) геометрии.
end	Конечная точка линии.
is_valid	Проверяет геометрию на валидность.
is_valid_reason	Если геометрия неправильная, возвращает краткую аннотацию причины.
name	Возвращает наименование геометрического объекта.
type	Возвращает тип геометрического элемента.

Методы:

affine_transform(trans)	Трансформирует объект исходя из заданных параметров трансформации.
almost_equals(other, tolerance)	Производит примерное сравнения с другой геометрией в пределах заданной точности.
boundary()	Возвращает границы геометрии в виде полилинии.
buffer(distance[, resolution, capStyle, ...])	Производит построение буфера вокруг объекта.
centroid()	Возвращает центроид геометрии.
clone()	Создает копию объекта.
contains(other)	Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.
convex_hull()	Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.
covers(other)	Возвращает True, если геометрия охватывает геометрию other.
crosses(other)	Возвращает True, если при пересечении геометрий объекты частично пересекаются.
difference(other)	Возвращает область первой геометрии, которая не пересечена второй геометрией.
disjoint(other)	Возвращает True, если геометрии не пересекаются и не соприкасаются.
envelope()	Возвращает полигон, описывающий заданную геометрию.
equals(other)	Производит сравнение с другой геометрией.
get_area([area_unit])	Рассчитывает площадь, если объект площадной.
get_distance(other[, u])	Производит расчет расстояния до объекта other.

continues on next page

Таблица 16 – продолжение с предыдущей страницы

get_length([u])	Рассчитывает длину геометрии.
get_perimeter([u])	Рассчитывает периметр геометрии.
intersection(other)	Возвращает область пересечения с другой геометрией.
intersects(other)	Возвращает True, если геометрии пересекаются.
overlaps(other)	Возвращает True, если пересечение геометрий отличается от обеих геометрий.
relate(other)	Проверяет отношения между объектами.
reproject(cs)	Перепроецирует геометрию в другую систему координат.
rotate(point, angle)	Поворот геометрии относительно заданной точки.
scale(kx, ky)	Масштабирует объект по заданным коэффициентам масштабирования.
shift(dx, dy)	Смещает объект на заданную величину.
split_by_polyline(splitter)	Разрезает объект линейным объектом.
symmetric_difference(other)	Возвращает логический XOR областей геометрий (объединение разниц).
to_geojson()	Преобразует геометрию в формат 'GeoJSON'
to_linestring()	Пробует геометрию преобразовать в линейный объект.
to_mif()	Преобразует геометрию в формат MIF.
to_polygon()	Пробует геометрию преобразовать в площадной объект.
to_wkb()	Возвращает WKB строку для геометрии.
to_wkt()	Возвращает WKT строку для геометрии.
touches(other)	Возвращает True, если геометрии соприкасаются.
union(other)	Возвращает результат объединения двух геометрий.
within(other)	Возвращает True, если геометрия находится полностью внутри геометрии other.

__init__(begin: Pnt, end: Pnt, cs: Optional[CoordSystem] = None)

Создает экземпляр класса.

affine_transform(trans: QTransform) → Geometry

Трансформирует объект исходя из заданных параметров трансформации.

См.также:

Для простых операций типа сдвиг, масштабирование и поворот, рекомендуется использовать shift(), scale(), rotate() соответственно.

Параметры

trans – Матрица трансформации.

almost_equals(other: Geometry, tolerance: float) → bool

Производит примерное сравнения с другой геометрией в пределах заданной точности.

Параметры

- **other** – Сравниваемый объект.
- **tolerance** – Точность сравнения.

Результат

Возвращает True, если геометрии равны в пределах заданного отклонения.

property begin: Pnt

Начальная точка линии. Точки допустимо создавать как экземпляр Pnt либо в виде пары float значений tuple

boundary() → Geometry

Возвращает границы геометрии в виде полилинии.

property bounds: Rect

Возвращает минимальный ограничивающий прямоугольник.

buffer(distance: float, resolution: int = 16, capStyle: LineCapStyle = LineCapStyle.Round, join_style: LineJoinStyle = LineJoinStyle.Round, mitreLimit: float = 5.0) → Geometry

Производит построение буфера вокруг объекта.

Параметры

- **distance** – Ширина буфера.
- **resolution** – Количество сегментов на квадрант.
- **capStyle** – Стиль окончания.
- **join_style** – Стиль соединения.
- **mitreLimit** – Предел среза.

centroid() → Geometry

Возвращает центроид геометрии.

clone() → Geometry

Создает копию объекта.

contains(other: Geometry) → bool

Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.

convex_hull() → Geometry

Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.

property coordsystem: CoordSystem

Система Координат (СК) геометрии.

covers(other: Geometry) → bool

Возвращает True, если геометрия охватывает геометрию other.

crosses(other: Geometry) → bool

Возвращает True, если при пересечении геометрий объекты частично пересекаются.

difference(other: Geometry) → Geometry

Возвращает область первой геометрии, которая не пересечена второй геометрией.

disjoint(other: Geometry) → bool

Возвращает True, если геометрии не пересекаются и не соприкасаются.

static distance_by_points(start: Union[Pnt, Tuple[float, float]], end: Union[Pnt, Tuple[float, float]], cs: Optional[CoordSystem] = None) → Tuple

Производит расчет расстояния между двумя точками и азимут от первой до второй точки.

См.также:

Обратная функция `point_by_azimuth()`

Параметры

- **start** – Начальная точка. Если задана СК, то координаты в ней.
- **end** – Конечная точка. Если задана СК, то координаты в ней.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращается пара значений (расстояние в метрах, азимут в градусах)

property end: Pnt

Конечная точка линии.

envelope() → Geometry

Возвращает полигон, описывающий заданную геометрию.

equals(other: Geometry) → bool

Производит сравнение с другой геометрией.

Параметры

other – Сравниваемая геометрия.

Результат

Возвращает True, если геометрии равны.

static from_geojson(json: str, cs: Optional[CoordSystem] = None) → Geometry

Возвращает геометрию из ее „GeoJSON“ представления.

Параметры

- **json** – json представление в виде строки.
- **cs** – Система координат.

static from_mif(mif: str) → Geometry

Возвращает геометрию из ее „MIF“ представления.

Параметры
mif – mif представление в виде строки.

static from_wkb(wkb: bytes, coordsystem: Optional[CoordSystem] = None) → Geometry

Создает геометрический объект из строки формата WKB.

Параметры

- **wkb** – Строка WKB
- **coordsystem** – Система координат, которая будет установлена для геометрии.

Список 57: Пример.

```
wkb = b'\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00$@\x00\x00\x00\x00\x00\x00
↪$@'
pnt = Geometry.from_wkb(wkb)
```

static from_wkt(wkt: str, coordsystem: Optional[CoordSystem] = None) → Geometry

Создает геометрический объект из строки формата WKT.

Параметры

- **wkt** – Строка WKT. Допустимо задание строки в формате с указанием SRID (EWKT). В данном случае система координат для создаваемой геометрии будет установлено исходя их этого значения.
- **coordsystem** – Система координат, которая будет установлена для геометрии. Если строка задана в виде EWKT и указано значение SRID, игнорируется.

Список 58: Пример.

```
polygon = Geometry.from_wkt('POLYGON ((10 10, 100 100, 100 10, 10 10))')
point = Geometry.from_wkt('SRID=4326;POINT (10 10)')
crs = CoordSystem.from_epsg(4326)
pline = Geometry.from_wkt('LINESTRING (30 10, 10 30, 40 40)', crs)
```

get_area(area_unit: Optional[AreaUnit] = None) → float

Рассчитывает площадь, если объект площадной. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в квадратных метрах.

Список 59: Пример.

```
# В проекции Меркатора:
csMercatorKm = CoordSystem.from_prj("10, 104, 1, 0")
poly = Polygon([(0, 0), (0, 2), (2, 2), (2, 0)], cs=csMercatorKm)
print('Area Mercator km:', poly.get_area())
print('Area Mercator m:', poly.get_area(Unit.sq_m))
print('Perimeter Mercator km:', poly.get_perimeter())
print('Perimeter Mercator m:', poly.get_perimeter(Unit.m))
```

(continues on next page)

(продолжение с предыдущей страницы)

```
# В Широте/Долготе
poly.coordsystem = CoordSystem.from_prj("1, 104")
print('Area LL m:', poly.get_area())
print('Area LL km:', poly.get_area(Unit.sq_km))
print('Perimeter LL m:', poly.get_perimeter())
print('Perimeter LL km:', poly.get_perimeter(Unit.km))
'''
>>> Area Mercator km: 4.017946986632519
>>> Area Mercator m: 4017946.9866325194
>>> Perimeter Mercator km: 8.017972048421552
>>> Perimeter Mercator m: 8017.972048421552
>>> Area LL m: 49452172514.0342
>>> Area LL km: 49452.1725140342
>>> Perimeter LL m: 889423.5067063896
>>> Perimeter LL km: 889.4235067063896
'''
```

Параметры

area_unit – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и она не задана, то производится расчет на плоскости.

get_distance(other: Geometry, u: Optional[LinearUnit] = None) → float

Производит расчет расстояния до объекта other. Результат возвращает в СК текущего объекта.

Параметры

- **other** – Анализируемый объект.
- **u** – Единицы измерения, в которых требуется получить результат.

Список 60: Пример.

```
# Создадим геометрию без СК
first = Point(0, 0)
second = Point(10, 0)
print('В плане:', first.get_distance(second))
# Установим разные СК для точек
first.coordsystem = CoordSystem.from_prj("10, 104, 7, 0")
second.coordsystem = CoordSystem.from_prj("1, 104")
print('На сфере м:', first.get_distance(second))
print('На сфере км:', first.get_distance(second, Unit.km))
'''
>>> В плане: 10.0
>>> На сфере м: 1111948.7428468117
>>> На сфере км: 1111.9487428468117
'''
```

get_length(u: Optional[LinearUnit] = None) → float

Рассчитывает длину геометрии. Метод применим только для линейных объектов. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Список 61: Пример.

```
# В проекции Меркатора:
csMercatorKm = CoordSystem.from_prj("10, 104, 1, 0")
ls = Line((0, 0), (10, 0), csMercatorKm)
print('Length Mercator km:', ls.get_length())
print('Lenght Mercator m:', ls.get_length(Unit.m))
# В Широте/Долготе
csLL = CoordSystem.from_prj("1, 104")
ls = Line((0, 0), (10, 0), csLL)
print('Length LL m:', ls.get_length())
print('Length LL km:', ls.get_length(Unit.km))
'''
>>> Length Mercator km: 9.988805508567783
>>> Lenght Mercator m: 9988.805508567782
>>> Length LL m: 1111948.7428468117
>>> Length LL km: 1111.9487428468117
'''
```

Параметры

u – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и она не задана, то производится расчет на плоскости.

get_perimeter(u: Optional[LinearUnit] = None) → float

Рассчитывает периметр геометрии. Метод применим только для площадных объектов. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Пример см. `get_area()`

Параметры

u – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и она не задана, то производится расчет на плоскости.

intersection(other: Geometry) → Geometry

Возвращает область пересечения с другой геометрией.

intersects(other: Geometry) → bool

Возвращает True, если геометрии пересекаются.

property is_valid: bool

Проверяет геометрию на валидность.

property is_valid_reason: str

Если геометрия неправильная, возвращает краткую аннотацию причины.

property name: str

Возвращает наименование геометрического объекта.

overlaps(other: Geometry) → bool

Возвращает True, если пересечение геометрий отличается от обеих геометрий.

static point_by_azimuth(point: Union[Pnt, Tuple[float, float]], azimuth: float, distance: float, cs: Optional[CoordSystem] = None) → Pnt

Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

См.также:

Обратная функция `distance_by_points()`

Параметры

- **point** – Точка, относительно которой производится расчет. Если задана СК, то в ней.
- **azimuth** – Азимут в градусах, указывающий направление.
- **distance** – Расстояние по азимуту. Задается в метрах.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат
Возвращает результирующую точку.

relate(other: Geometry) → str

Проверяет отношения между объектами. Подробнее см. DE-9IM

reproject(cs: CoordSystem) → Geometry

Перепроецирует геометрию в другую систему координат.

Параметры
cs – СК, в которой требуется получить объект.

rotate(point: Union[Pnt, Tuple[float, float]], angle: float) → Geometry

Поворот геометрии относительно заданной точки. Поворот производится против часовой стрелки.

Параметры

- **point** – Точка, вокруг которой необходимо произвести поворот.
- **angle** – Угол поворота в градусах.

scale(kx: float, ky: float) → Geometry

Масштабирует объект по заданным коэффициентам масштабирования. После выполнения операции центр результирующего объекта остается прежним.

Параметры

- **kx** – Масштабирование по координате X.
- **ky** – Масштабирование по координате Y.

shift(dx: float, dy: float) → Geometry

Смещает объект на заданную величину. Значения задаются в текущей проекции.

Параметры

- **dx** – Сдвиг по координате X.
- **dy** – Сдвиг по координате Y.

split_by_polyline(splitter: Geometry) → Optional[Geometry]

Разрезает объект линейным объектом.

Параметры
splitter – Геометрический объект, которым будет производиться разрезание объекта.

Результат
Возвращает полученный результат

```
poly = Polygon((0,10), (10,10), (10, 0))
line = Line((-5,5), (20,5))
r = poly.split_by_polyline(line)
print(r.wkt)
'''
>>> GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5, 5 5, 0 10)), POLYGON␣
↪((10 5, 10 0, 5 5, 10 5))) GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5,␣
↪5 5, 0 10)), POLYGON ((10 5, 10 0, 5 5, 10 5)))
'''
```

symmetric_difference(other: Geometry) → Geometry

Возвращает логический XOR областей геометрий (объединение разниц).

Параметры
other – Геометрия для анализа.

to_geojson() → str

Преобразует геометрию в формат „GeoJSON“

to_linestring() → Optional[Geometry]

Пробует геометрию преобразовать в линейный объект. В случае неудачи возвращает None.

to_mif() → str

Преобразует геометрию в формат MIF.

to_polygon() → Optional[Geometry]

Пробует геометрию преобразовать в площадной объект. В случае неудачи возвращает None.

to_wkb() → bytes

Возвращает WKB строку для геометрии.

to_wkt() → str

Возвращает WKT строку для геометрии.

touches(other: Geometry) → bool

Возвращает True, если геометрии соприкасаются.

property type: Type

Возвращает тип геометрического элемента.

Таблица 17: Возможные значения

Значение	Наименование
Unknown	Не определен
Point	Точка
Line	Линия
LineString	Полилиния
Polygon	Полигон
MultiPoint	Коллекция точек
MultiLineString	Коллекция полилиний
MultiPolygon	Коллекция полигонов
GeometryCollection	Смешанная коллекция
Arc	Дуга
Ellipse	Эллипс
Rectangle	Прямоугольник
RoundedRectangle	Скругленный прямоугольник
Text	Текст

Список 62: Пример.

```
point = Point(10, 10)
if point.type == GeometryType.Point:
    print('Это точка')
```

union(other: Geometry) → Geometry

Возвращает результат объединения двух геометрий.

within(other: Geometry) → bool

Возвращает True, если геометрия находится полностью внутри геометрии other.

22.12.4 LineString - Полилиния

class axipy.**LineString**

Базовые классы: Geometry

Геометрический объект типа полилиния.

Параметры

- **points** – Список точек. Может задаваться следующим образом:
  - В виде списка list из пар tuple.
  - В виде перечня точек. В данном случае, если необходимо задать СК, то требуется явно указать наименование параметра.
  - В виде итератора по элементам, состоящих из пар tuple.
- **cs** – Система Координат, в которой создается геометрия.

Список 63: Пример.

```
csLL = CoordSystem.from_prj("1, 104")
ls = LineString([(1, 2), Pnt(3, 4), Pnt(5, 6), (7, 8)])  # Создадим полилинию без␣
↪СК
```

(continues on next page)

(продолжение с предыдущей страницы)

```
"""
Обновим точку с индексом 1. Допустимо только обновление точки целиком.
Изменение координат по одиночке не поддерживается.
"""
ls.points[1] = (33, 44)
ls.points.append((9, 10))  # Добавим точку в конец
ls.points.remove(2)  # Удалим вторую точку
ls.points.insert(3, (11, 12))  # Добавим точку на позицию 3
for p in ls.points:  # Просмотр всех точек
    print("point:", p)
ls2 = LineString((1, 2), (3, 4), (5, 6), (7, 8), cs=csLL)  # Создание полинии,␣
↪передав перечень точек.
itr = (a for a in ls.points)  # Создадим итератор на базе точек первой полилинии
ls3 = LineString(itr)
```

Конструктор класса:

__init__(*points[, cs])	Создает экземпляр класса.

Классовые методы:

distance_by_points(start, end[, cs])	Производит расчет расстояния между двумя точками и азимут от первой до второй точки.
from_geojson(json[, cs])	Возвращает геометрию из ее 'GeoJSON' представления.
from_mif(mif)	Возвращает геометрию из ее 'MIF' представления.
from_wkb(wkb[, coordsystem])	Создает геометрический объект из строки формата WKB.
from_wkt(wkt[, coordsystem])	Создает геометрический объект из строки формата WKT.
point_by_azimuth(point, azimuth, distance[, cs])	Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

Свойства:

bounds	Возвращает минимальный ограничивающий прямоугольник.
coordsystem	Система Координат (СК) геометрии.
is_valid	Проверяет геометрию на валидность.
is_valid_reason	Если геометрия неправильная, возвращает краткую аннотацию причины.
name	Возвращает наименование геометрического объекта.
points	Точки полилинии.
type	Возвращает тип геометрического элемента.

Методы:

affine_transform(trans)	Трансформирует объект исходя из заданных параметров трансформации.
almost_equals(other, tolerance)	Производит примерное сравнения с другой геометрией в пределах заданной точности.
boundary()	Возвращает границы геометрии в виде полилинии.
buffer(distance[, resolution, capStyle, ...])	Производит построение буфера вокруг объекта.
centroid()	Возвращает центроид геометрии.
clone()	Создает копию объекта.
contains(other)	Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.
convex_hull()	Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.
covers(other)	Возвращает True, если геометрия охватывает геометрию other.
crosses(other)	Возвращает True, если при пересечении геометрий объекты частично пересекаются.
difference(other)	Возвращает область первой геометрии, которая не пересечена второй геометрией.
disjoint(other)	Возвращает True, если геометрии не пересекаются и не соприкасаются.
envelope()	Возвращает полигон, описывающий заданную геометрию.
equals(other)	Производит сравнение с другой геометрией.
get_area([area_unit])	Рассчитывает площадь, если объект площадной.
get_distance(other[, u])	Производит расчет расстояния до объекта other.
get_length([u])	Рассчитывает длину геометрии.
get_line_direction()	Направление линии.
get_perimeter([u])	Рассчитывает периметр геометрии.
intersection(other)	Возвращает область пересечения с другой геометрией.
intersects(other)	Возвращает True, если геометрии пересекаются.
overlaps(other)	Возвращает True, если пересечение геометрий отличается от обеих геометрий.
relate(other)	Проверяет отношения между объектами.
reproject(cs)	Перепроецирует геометрию в другую систему координат.

continues on next page

Таблица 18 – продолжение с предыдущей страницы

rotate(point, angle)	Поворот геометрии относительно заданной точки.
scale(kx, ky)	Масштабирует объект по заданным коэффициентам масштабирования.
set_line_direction(direction)	Задание направления линии.
shift(dx, dy)	Смещает объект на заданную величину.
split_by_polyline(splitter)	Разрезает объект линейным объектом.
symmetric_difference(other)	Возвращает логический XOR областей геометрий (объединение разниц).
to_geojson()	Преобразует геометрию в формат 'GeoJSON'
to_linestring()	Пробует геометрию преобразовать в линейный объект.
to_mif()	Преобразует геометрию в формат MIF.
to_polygon()	Пробует геометрию преобразовать в площадной объект.
to_wkb()	Возвращает WKB строку для геометрии.
to_wkt()	Возвращает WKT строку для геометрии.
touches(other)	Возвращает True, если геометрии соприкасаются.
union(other)	Возвращает результат объединения двух геометрий.
within(other)	Возвращает True, если геометрия находится полностью внутри геометрии other.

__init__(*points: Union[Pnt, Tuple[float, float], Iterable[Pnt], Iterable[Tuple[float, float]]], cs: Optional[CoordSystem] = None)

Создает экземпляр класса.

affine_transform(trans: QTransform) → Geometry

Трансформирует объект исходя из заданных параметров трансформации.

См.также:

Для простых операций типа сдвиг, масштабирование и поворот, рекомендуется использовать shift(), scale(), rotate() соответственно.

Параметры

trans – Матрица трансформации.

almost_equals(other: Geometry, tolerance: float) → bool

Производит примерное сравнения с другой геометрией в пределах заданной точности.

Параметры

- **other** – Сравниваемый объект.
- **tolerance** – Точность сравнения.

Результат

Возвращает True, если геометрии равны в пределах заданного отклонения.

boundary() → Geometry

Возвращает границы геометрии в виде полилинии.

property bounds: Rect

Возвращает минимальный ограничивающий прямоугольник.

buffer(distance: float, resolution: int = 16, capStyle: LineCapStyle = LineCapStyle.Round, join_style: LineJoinStyle = LineJoinStyle.Round, mitreLimit: float = 5.0) → Geometry

Производит построение буфера вокруг объекта.

Параметры

- **distance** – Ширина буфера.
- **resolution** – Количество сегментов на квадрант.
- **capStyle** – Стиль окончания.
- **join_style** – Стиль соединения.
- **mitreLimit** – Предел среза.

centroid() → Geometry

Возвращает центроид геометрии.

clone() → Geometry

Создает копию объекта.

contains(other: Geometry) → bool

Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.

convex_hull() → Geometry

Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.

property coordsystem: CoordSystem

Система Координат (СК) геометрии.

covers(other: Geometry) → bool

Возвращает True, если геометрия охватывает геометрию other.

crosses(other: Geometry) → bool

Возвращает True, если при пересечении геометрий объекты частично пересекаются.

difference(other: Geometry) → Geometry

Возвращает область первой геометрии, которая не пересечена второй геометрией.

disjoint(other: Geometry) → bool

Возвращает True, если геометрии не пересекаются и не соприкасаются.

static distance_by_points(start: Union[Pnt, Tuple[float, float]], end: Union[Pnt, Tuple[float, float]], cs: Optional[CoordSystem] = None) → Tuple

Производит расчет расстояния между двумя точками и азимут от первой до второй точки.

См.также:

Обратная функция `point_by_azimuth()`

Параметры

- **start** – Начальная точка. Если задана СК, то координаты в ней.
- **end** – Конечная точка. Если задана СК, то координаты в ней.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращается пара значений (расстояние в метрах, азимут в градусах)

envelope() → Geometry

Возвращает полигон, описывающий заданную геометрию.

equals(other: Geometry) → bool

Производит сравнение с другой геометрией.

Параметры

other – Сравниваемая геометрия.

Результат

Возвращает True, если геометрии равны.

static from_geojson(json: str, cs: Optional[CoordSystem] = None) → Geometry

Возвращает геометрию из ее „GeoJSON“ представления.

Параметры

- **json** – json представление в виде строки.
- **cs** – Система координат.

static from_mif(mif: str) → Geometry

Возвращает геометрию из ее „MIF“ представления.

Параметры

mif – mif представление в виде строки.

static from_wkb(wkb: bytes, coordsystem: Optional[CoordSystem] = None) → Geometry

Создает геометрический объект из строки формата WKB.

Параметры

- **wkb** – Строка WKB
- **coordsystem** – Система координат, которая будет установлена для геометрии.

Список 64: Пример.

```
wkb = b'\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00$@\x00\x00\x00\x00\x00\x00
↪$@'
pnt = Geometry.from_wkb(wkb)
```

static from_wkt(wkt: str, coordsystem: Optional[CoordSystem] = None) → Geometry

Создает геометрический объект из строки формата WKT.

Параметры

- **wkt** – Строка WKT. Допустимо задание строки в формате с указанием SRID (EWKT). В данном случае система координат для создаваемой геометрии будет установлено исходя их этого значения.
- **coordsystem** – Система координат, которая будет установлена для геометрии. Если строка задана в виде EWKT и указано значение SRID, игнорируется.

Список 65: Пример.

```
polygon = Geometry.from_wkt('POLYGON ((10 10, 100 100, 100 10, 10 10))')
point = Geometry.from_wkt('SRID=4326;POINT (10 10)')
crs = CoordSystem.from_epsg(4326)
pline = Geometry.from_wkt('LINESTRING (30 10, 10 30, 40 40)', crs)
```

get_area(area_unit: Optional[AreaUnit] = None) → float

Рассчитывает площадь, если объект площадной. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в квадратных метрах.

Список 66: Пример.

```
# В проекции Меркатора:
csMercatorKm = CoordSystem.from_prj("10, 104, 1, 0")
poly = Polygon([(0, 0), (0, 2), (2, 2), (2, 0)], cs=csMercatorKm)
print('Area Mercator km:', poly.get_area())
print('Area Mercator m:', poly.get_area(Unit.sq_m))
print('Perimeter Mercator km:', poly.get_perimeter())
print('Perimeter Mercator m:', poly.get_perimeter(Unit.m))
# В Широте/Долготе
poly.coordsystem = CoordSystem.from_prj("1, 104")
print('Area LL m:', poly.get_area())
print('Area LL km:', poly.get_area(Unit.sq_km))
print('Perimeter LL m:', poly.get_perimeter())
print('Perimeter LL km:', poly.get_perimeter(Unit.km))
'''
>>> Area Mercator km: 4.017946986632519
>>> Area Mercator m: 4017946.9866325194
>>> Perimeter Mercator km: 8.017972048421552
>>> Perimeter Mercator m: 8017.972048421552
>>> Area LL m: 49452172514.0342
>>> Area LL km: 49452.1725140342
>>> Perimeter LL m: 889423.5067063896
>>> Perimeter LL km: 889.4235067063896
'''
```

Параметры

area_unit – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и она не задана, то производится расчет на плоскости.

get_distance(other: Geometry, u: Optional[LinearUnit] = None) → float

Производит расчет расстояния до объекта other. Результат возвращает в СК текущего объекта.

Параметры

- **other** – Анализируемый объект.
- **u** – Единицы измерения, в которых требуется получить результат.

Список 67: Пример.

```
# Создадим геометрию без СК
first = Point(0, 0)
second = Point(10, 0)
print('В плане:', first.get_distance(second))
# Установим разные СК для точек
first.coordsystem = CoordSystem.from_prj("10, 104, 7, 0")
second.coordsystem = CoordSystem.from_prj("1, 104")
print('На сфере м:', first.get_distance(second))
print('На сфере км:', first.get_distance(second, Unit.km))
'''
>>> В плане: 10.0
>>> На сфере м: 1111948.7428468117
>>> На сфере км: 1111.9487428468117
'''
```

get_length(u: Optional[LinearUnit] = None) → float

Рассчитывает длину геометрии. Метод применим только для линейных объектов. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Список 68: Пример.

```
# В проекции Меркатора:
csMercatorKm = CoordSystem.from_prj("10, 104, 1, 0")
ls = Line((0, 0), (10, 0), csMercatorKm)
print('Length Mercator km:', ls.get_length())
print('Lenght Mercator m:', ls.get_length(Unit.m))
# В Широте/Долготе
csLL = CoordSystem.from_prj("1, 104")
ls = Line((0, 0), (10, 0), csLL)
print('Length LL m:', ls.get_length())
print('Length LL km:', ls.get_length(Unit.km))
'''
>>> Length Mercator km: 9.988805508567783
>>> Lenght Mercator m: 9988.805508567782
>>> Length LL m: 1111948.7428468117
>>> Length LL km: 1111.9487428468117
'''
```

Параметры

u – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и она не задана, то производится расчет на плоскости.

get_line_direction() → LineDirection

Направление линии.

get_perimeter(u: Optional[LinearUnit] = None) → float

Рассчитывает периметр геометрии. Метод применим только для площадных объектов. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Пример см. `get_area()`

Параметры

u – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и она не задана, то производится расчет на плоскости.

intersection(other: Geometry) → Geometry

Возвращает область пересечения с другой геометрией.

intersects(other: Geometry) → bool

Возвращает True, если геометрии пересекаются.

property is_valid: bool

Проверяет геометрию на валидность.

property is_valid_reason: str

Если геометрия неправильная, возвращает краткую аннотацию причины.

property name: str

Возвращает наименование геометрического объекта.

overlaps(other: Geometry) → bool

Возвращает True, если пересечение геометрий отличается от обеих геометрий.

static point_by_azimuth(point: Union[Pnt, Tuple[float, float]], azimuth: float, distance: float, cs: Optional[CoordSystem] = None) → Pnt

Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

См.также:

Обратная функция `distance_by_points()`

Параметры

- **point** – Точка, относительно которой производится расчет. Если задана СК, то в ней.
- **azimuth** – Азимут в градусах, указывающий направление.
- **distance** – Расстояние по азимуту. Задается в метрах.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращает результирующую точку.

property points: List[Pnt]

Точки полилинии. Реализован как список python `list` точек `Pnt`. Также поддерживаются список пар `tuple`.

Примечание: При обновлении значения точки допустимо только изменение ее заменой.

relate(other: Geometry) → str

Проверяет отношения между объектами. Подробнее см. DE-9IM

reproject(cs: CoordSystem) → Geometry

Перепроецирует геометрию в другую систему координат.

Параметры
cs – СК, в которой требуется получить объект.

rotate(point: Union[Pnt, Tuple[float, float]], angle: float) → Geometry

Поворот геометрии относительно заданной точки. Поворот производится против часовой стрелки.

Параметры

- **point** – Точка, вокруг которой необходимо произвести поворот.
- **angle** – Угол поворота в градусах.

scale(kx: float, ky: float) → Geometry

Масштабирует объект по заданным коэффициентам масштабирования. После выполнения операции центр результирующего объекта остается прежним.

Параметры

- **kx** – Масштабирование по координате X.
- **ky** – Масштабирование по координате Y.

set_line_direction(direction: LineDirection)

Задание направления линии.

Параметры
direction – Желаемое значение

shift(dx: float, dy: float) → Geometry

Смещает объект на заданную величину. Значения задаются в текущей проекции.

Параметры

- **dx** – Сдвиг по координате X.
- **dy** – Сдвиг по координате Y.

split_by_polyline(splitter: Geometry) → Optional[Geometry]

Разрезает объект линейным объектом.

Параметры
splitter – Геометрический объект, которым будет производиться разрезание объекта.

Результат
Возвращает полученный результат

```
poly = Polygon((0,10), (10,10), (10, 0))
line = Line((-5,5), (20,5))
r = poly.split_by_polyline(line)
print(r.wkt)
'''
>>> GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5, 5 5, 0 10)), POLYGON␣
↪((10 5, 10 0, 5 5, 10 5))) GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5,␣
↪5 5, 0 10)), POLYGON ((10 5, 10 0, 5 5, 10 5)))
'''
```

symmetric_difference(other: Geometry) → Geometry

Возвращает логический XOR областей геометрий (объединение разниц).

Параметры
other – Геометрия для анализа.

to_geojson() → str

Преобразует геометрию в формат „GeoJSON“

to_linestring() → Optional[Geometry]

Пробует геометрию преобразовать в линейный объект. В случае неудачи возвращает None.

to_mif() → str

Преобразует геометрию в формат MIF.

to_polygon() → Optional[Geometry]

Пробует геометрию преобразовать в площадной объект. В случае неудачи возвращает None.

to_wkb() → bytes

Возвращает WKB строку для геометрии.

to_wkt() → str

Возвращает WKT строку для геометрии.

touches(other: Geometry) → bool

Возвращает True, если геометрии соприкасаются.

property type: Type

Возвращает тип геометрического элемента.

Таблица 19: Возможные значения

Значение	Наименование
Unknown	Не определен
Point	Точка
Line	Линия
LineString	Полилиния
Polygon	Полигон
MultiPoint	Коллекция точек
MultiLineString	Коллекция полилиний
MultiPolygon	Коллекция полигонов
GeometryCollection	Смешанная коллекция
Arc	Дуга
Ellipse	Эллипс
Rectangle	Прямоугольник
RoundedRectangle	Скругленный прямоугольник
Text	Текст

Список 69: Пример.

```
point = Point(10, 10)
if point.type == GeometryType.Point:
    print('Это точка')
```

union(other: Geometry) → Geometry

Возвращает результат объединения двух геометрий.

within(other: Geometry) → bool

Возвращает True, если геометрия находится полностью внутри геометрии other.

22.12.5 Polygon - Полигон

class axipy.**Polygon**

Базовые классы: Geometry

Геометрический объект типа полигон. Представляет собой часть плоскости, ограниченной замкнутой полилинией. Кроме внешней границы, полигон может иметь одну или несколько внутренних (дырок).

Параметры

- **points** – Список точек внешнего контура. Может задаваться следующим образом:
  - В виде списка list из пар tuple.
  - В виде перечня точек. В данном случае, если необходимо задать СК, то требуется явно указать наименование параметра.
  - В виде итератора по элементам, состоящих из пар tuple.
- **cs** – Система Координат, в которой создается геометрия.

Список 70: Пример.

```
poly = Polygon([(0, 0), Pnt(1, 10), Pnt(10, 11), (10, 2)])  # Создадим объект
poly.points[2] = (14, 15)  # Поменяем вторую точку
poly.points.insert(3, (11, 5))  # Добавим точку
for p in poly.points:  # Просмотр точек полигона
    print("point:", p)
poly.points.remove(2)  # Удалим вторую точку
```

Конструктор класса:

__init__(*points[, cs])	Создает экземпляр класса.

Классовые методы:

distance_by_points(start, end[, cs])	Производит расчет расстояния между двумя точками и азимут от первой до второй точки.
from_geojson(json[, cs])	Возвращает геометрию из ее 'GeoJSON' представления.
from_mif(mif)	Возвращает геометрию из ее 'MIF' представления.
from_rect(rect[, cs])	Создает полигон на базе прямоугольника.
from_wkb(wkb[, coordsystem])	Создает геометрический объект из строки формата WKB.
from_wkt(wkt[, coordsystem])	Создает геометрический объект из строки формата WKT.
point_by_azimuth(point, azimuth, distance[, cs])	Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

Свойства:

bounds	Возвращает минимальный ограничивающий прямоугольник.
coordsystem	Система Координат (СК) геометрии.
holes	Дырки полигона.
is_valid	Проверяет геометрию на валидность.
is_valid_reason	Если геометрия неправильная, возвращает краткую аннотацию причины.
name	Возвращает наименование геометрического объекта.
points	Точки полигона.
type	Возвращает тип геометрического элемента.

Методы:

affine_transform(trans)	Трансформирует объект исходя из заданных параметров трансформации.
almost_equals(other, tolerance)	Производит примерное сравнения с другой геометрией в пределах заданной точности.
boundary()	Возвращает границы геометрии в виде полилинии.
buffer(distance[, resolution, capStyle, ...])	Производит построение буфера вокруг объекта.
centroid()	Возвращает центроид геометрии.
clone()	Создает копию объекта.
contains(other)	Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.
convex_hull()	Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.
covers(other)	Возвращает True, если геометрия охватывает геометрию other.
crosses(other)	Возвращает True, если при пересечении геометрий объекты частично пересекаются.
difference(other)	Возвращает область первой геометрии, которая не пересечена второй геометрией.
disjoint(other)	Возвращает True, если геометрии не пересекаются и не соприкасаются.
envelope()	Возвращает полигон, описывающий заданную геометрию.
equals(other)	Производит сравнение с другой геометрией.
get_area([area_unit])	Рассчитывает площадь, если объект площадной.
get_distance(other[, u])	Производит расчет расстояния до объекта other.
get_length([u])	Рассчитывает длину геометрии.
get_line_direction()	Направление линии.
get_perimeter([u])	Рассчитывает периметр геометрии.
intersection(other)	Возвращает область пересечения с другой геометрией.
intersects(other)	Возвращает True, если геометрии пересекаются.
overlaps(other)	Возвращает True, если пересечение геометрий отличается от обеих геометрий.
relate(other)	Проверяет отношения между объектами.
reproject(cs)	Перепроецирует геометрию в другую систему координат.
rotate(point, angle)	Поворот геометрии относительно заданной точки.

continues on next page

Таблица 20 – продолжение с предыдущей страницы

scale(kx, ky)	Масштабирует объект по заданным коэффициентам масштабирования.
set_line_direction(direction)	Задание направления линии.
shift(dx, dy)	Смещает объект на заданную величину.
split_by_polyline(splitter)	Разрезает объект линейным объектом.
symmetric_difference(other)	Возвращает логический XOR областей геометрий (объединение разниц).
to_geojson()	Преобразует геометрию в формат 'GeoJSON'
to_linestring()	Пробует геометрию преобразовать в линейный объект.
to_mif()	Преобразует геометрию в формат MIF.
to_polygon()	Пробует геометрию преобразовать в площадной объект.
to_wkb()	Возвращает WKB строку для геометрии.
to_wkt()	Возвращает WKT строку для геометрии.
touches(other)	Возвращает True, если геометрии соприкасаются.
union(other)	Возвращает результат объединения двух геометрий.
within(other)	Возвращает True, если геометрия находится полностью внутри геометрии other.

__init__(*points: Union[Pnt, Tuple[float, float], Iterable[Pnt], Iterable[Tuple[float, float]]], cs: Optional[CoordSystem] = None)

Создает экземпляр класса.

affine_transform(trans: QTransform) → Geometry

Трансформирует объект исходя из заданных параметров трансформации.

См.также:

Для простых операций типа сдвиг, масштабирование и поворот, рекомендуется использовать shift(), scale(), rotate() соответственно.

Параметры

trans – Матрица трансформации.

almost_equals(other: Geometry, tolerance: float) → bool

Производит примерное сравнения с другой геометрией в пределах заданной точности.

Параметры

- **other** – Сравниваемый объект.
- **tolerance** – Точность сравнения.

Результат

Возвращает True, если геометрии равны в пределах заданного отклонения.

boundary() → Geometry

Возвращает границы геометрии в виде полилинии.

property bounds: Rect

Возвращает минимальный ограничивающий прямоугольник.

buffer(distance: float, resolution: int = 16, capStyle: LineCapStyle = LineCapStyle.Round, join_style: LineJoinStyle = LineJoinStyle.Round, mitreLimit: float = 5.0) → Geometry

Производит построение буфера вокруг объекта.

Параметры

- **distance** – Ширина буфера.
- **resolution** – Количество сегментов на квадрант.
- **capStyle** – Стиль окончания.
- **join_style** – Стиль соединения.
- **mitreLimit** – Предел среза.

centroid() → Geometry

Возвращает центроид геометрии.

clone() → Geometry

Создает копию объекта.

contains(other: Geometry) → bool

Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.

convex_hull() → Geometry

Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.

property coordsystem: CoordSystem

Система Координат (СК) геометрии.

covers(other: Geometry) → bool

Возвращает True, если геометрия охватывает геометрию other.

crosses(other: Geometry) → bool

Возвращает True, если при пересечении геометрий объекты частично пересекаются.

difference(other: Geometry) → Geometry

Возвращает область первой геометрии, которая не пересечена второй геометрией.

disjoint(other: Geometry) → bool

Возвращает True, если геометрии не пересекаются и не соприкасаются.

static distance_by_points(start: Union[Pnt, Tuple[float, float]], end: Union[Pnt, Tuple[float, float]], cs: Optional[CoordSystem] = None) → Tuple

Производит расчет расстояния между двумя точками и азимут от первой до второй точки.

См.также:

Обратная функция `point_by_azimuth()`

Параметры

- **start** – Начальная точка. Если задана СК, то координаты в ней.
- **end** – Конечная точка. Если задана СК, то координаты в ней.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращается пара значений (расстояние в метрах, азимут в градусах)

envelope() → Geometry

Возвращает полигон, описывающий заданную геометрию.

equals(other: Geometry) → bool

Производит сравнение с другой геометрией.

Параметры

other – Сравниваемая геометрия.

Результат

Возвращает True, если геометрии равны.

static from_geojson(json: str, cs: Optional[CoordSystem] = None) → Geometry

Возвращает геометрию из ее „GeoJSON“ представления.

Параметры

- **json** – json представление в виде строки.
- **cs** – Система координат.

static from_mif(mif: str) → Geometry

Возвращает геометрию из ее „MIF“ представления.

Параметры

mif – mif представление в виде строки.

static from_rect(rect: Rect, cs: Optional[CoordSystem] = None) → Polygon

Создает полигон на базе прямоугольника.

Параметры

- **rect** – Прямоугольник, на основе которого формируются координаты.
- **cs** – Система Координат, в которой создается геометрия.

static from_wkb(wkb: bytes, coordsystem: Optional[CoordSystem] = None) → Geometry

Создает геометрический объект из строки формата WKB.

Параметры

- **wkb** – Строка WKB
- **coordsystem** – Система координат, которая будет установлена для геометрии.

Список 71: Пример.

```
wkb = b'\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00$@\x00\x00\x00\x00\x00\x00
↪$@'
pnt = Geometry.from_wkb(wkb)
```

static from_wkt(wkt: str, coordsystem: Optional[CoordSystem] = None) → Geometry

Создает геометрический объект из строки формата WKT.

Параметры

- **wkt** – Строка WKT. Допустимо задание строки в формате с указанием SRID (EWKT). В данном случае система координат для создаваемой геометрии будет установлено исходя их этого значения.
- **coordsystem** – Система координат, которая будет установлена для геометрии. Если строка задана в виде EWKT и указано значение SRID, игнорируется.

Список 72: Пример.

```
polygon = Geometry.from_wkt('POLYGON ((10 10, 100 100, 100 10, 10 10))')
point = Geometry.from_wkt('SRID=4326;POINT (10 10)')
crs = CoordSystem.from_epsg(4326)
pline = Geometry.from_wkt('LINESTRING (30 10, 10 30, 40 40)', crs)
```

get_area(area_unit: Optional[AreaUnit] = None) → float

Рассчитывает площадь, если объект площадной. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в квадратных метрах.

Список 73: Пример.

```
# В проекции Меркатора:
csMercatorKm = CoordSystem.from_prj("10, 104, 1, 0")
poly = Polygon([(0, 0), (0, 2), (2, 2), (2, 0)], cs=csMercatorKm)
print('Area Mercator km:', poly.get_area())
print('Area Mercator m:', poly.get_area(Unit.sq_m))
print('Perimeter Mercator km:', poly.get_perimeter())
print('Perimeter Mercator m:', poly.get_perimeter(Unit.m))
# В Широте/Долготе
poly.coordsystem = CoordSystem.from_prj("1, 104")
print('Area LL m:', poly.get_area())
print('Area LL km:', poly.get_area(Unit.sq_km))
print('Perimeter LL m:', poly.get_perimeter())
print('Perimeter LL km:', poly.get_perimeter(Unit.km))
'''
>>> Area Mercator km: 4.017946986632519
>>> Area Mercator m: 4017946.9866325194
>>> Perimeter Mercator km: 8.017972048421552
>>> Perimeter Mercator m: 8017.972048421552
>>> Area LL m: 49452172514.0342
>>> Area LL km: 49452.1725140342
>>> Perimeter LL m: 889423.5067063896
>>> Perimeter LL km: 889.4235067063896
'''
```

Параметры
area_unit – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и она не задана, то производится расчет на плоскости.

get_distance(other: Geometry, u: Optional[LinearUnit] = None) → float

Производит расчет расстояния до объекта other. Результат возвращает в СК текущего объекта.

Параметры

- **other** – Анализируемый объект.
- **u** – Единицы измерения, в которых требуется получить результат.

Список 74: Пример.

```
# Создадим геометрию без СК
first = Point(0, 0)
second = Point(10, 0)
print('В плане:', first.get_distance(second))
# Установим разные СК для точек
first.coordsystem = CoordSystem.from_prj("10, 104, 7, 0")
second.coordsystem = CoordSystem.from_prj("1, 104")
print('На сфере м:', first.get_distance(second))
print('На сфере км:', first.get_distance(second, Unit.km))
'''
>>> В плане: 10.0
>>> На сфере м: 1111948.7428468117
>>> На сфере км: 1111.9487428468117
'''
```

get_length(u: Optional[LinearUnit] = None) → float

Рассчитывает длину геометрии. Метод применим только для линейных объектов. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Список 75: Пример.

```
# В проекции Меркатора:
csMercatorKm = CoordSystem.from_prj("10, 104, 1, 0")
ls = Line((0, 0), (10, 0), csMercatorKm)
print('Length Mercator km:', ls.get_length())
print('Lenght Mercator m:', ls.get_length(Unit.m))
# В Широте/Долготе
csLL = CoordSystem.from_prj("1, 104")
ls = Line((0, 0), (10, 0), csLL)
print('Length LL m:', ls.get_length())
print('Length LL km:', ls.get_length(Unit.km))
'''
>>> Length Mercator km: 9.988805508567783
>>> Lenght Mercator m: 9988.805508567782
>>> Length LL m: 1111948.7428468117
>>> Length LL km: 1111.9487428468117
'''
```

Параметры
u – Единица измерения, в которой необходимо получить результат.

Если не задана, то используется единица измерения для СК. Если и она не задана, то производится расчет на плоскости.

get_line_direction() → LineDirection

Направление линии.

get_perimeter(u: Optional[LinearUnit] = None) → float

Рассчитывает периметр геометрии. Метод применим только для площадных объектов. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Пример см. `get_area()`

Параметры

u – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и она не задана, то производится расчет на плоскости.

property holes: List[Pnt]

Дырки полигона. Реализован в виде списка `list`.

Список 76: Пример.

```
poly = Polygon((0, 0), (1, 10), (10, 1))
poly.holes.append([(2, 2), (2, 4), (5, 3)])  # Добавим дырку
for p in poly.holes[0]:  # Просмотр точек дырки полигона
    print("Point of hole:", p)
print('Вторая точка первой дырки:', poly.holes[0][1])
poly.holes[0][1] = (33, 44)  # Обновим значение этой точки
# Изменим направление для полигона
poly.set_line_direction(LineDirection.CounterClockwise)
# То же, но для дырки
poly.holes[0].set_line_direction(LineDirection.CounterClockwise)
```

intersection(other: Geometry) → Geometry

Возвращает область пересечения с другой геометрией.

intersects(other: Geometry) → bool

Возвращает True, если геометрии пересекаются.

property is_valid: bool

Проверяет геометрию на валидность.

property is_valid_reason: str

Если геометрия неправильная, возвращает краткую аннотацию причины.

property name: str

Возвращает наименование геометрического объекта.

overlaps(other: Geometry) → bool

Возвращает True, если пересечение геометрий отличается от обеих геометрий.

static point_by_azimuth(point: Union[Pnt, Tuple[float, float]], azimuth: float, distance: float, cs: Optional[CoordSystem] = None) → Pnt

Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

См.также:

Обратная функция `distance_by_points()`

Параметры

- **point** – Точка, относительно которой производится расчет. Если задана СК, то в ней.
- **azimuth** – Азимут в градусах, указывающий направление.
- **distance** – Расстояние по азимуту. Задается в метрах.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращает результирующую точку.

property points: List[Pnt]

Точки полигона. Реализован как список python `list` точек `Pnt`. В каждом контуре, количество точек на 1 больше, чем количество узлов, так как контур - это замкнутая линия, и первая и последняя точка совпадают.

relate(other: Geometry) → str

Проверяет отношения между объектами. Подробнее см. DE-9IM

reproject(cs: CoordSystem) → Geometry

Перепроецирует геометрию в другую систему координат.

Параметры

cs – СК, в которой требуется получить объект.

rotate(point: Union[Pnt, Tuple[float, float]], angle: float) → Geometry

Поворот геометрии относительно заданной точки. Поворот производится против часовой стрелки.

Параметры

- **point** – Точка, вокруг которой необходимо произвести поворот.
- **angle** – Угол поворота в градусах.

scale(kx: float, ky: float) → Geometry

Масштабирует объект по заданным коэффициентам масштабирования. После выполнения операции центр результирующего объекта остается прежним.

Параметры

- **kx** – Масштабирование по координате X.
- **ky** – Масштабирование по координате Y.

set_line_direction(direction: LineDirection)

Задание направления линии.

Параметры

direction – Желаемое значение

shift(dx: float, dy: float) → Geometry

Смещает объект на заданную величину. Значения задаются в текущей проекции.

Параметры

- **dx** – Сдвиг по координате X.
- **dy** – Сдвиг по координате Y.

split_by_polyline(splitter: Geometry) → Optional[Geometry]

Разрезает объект линейным объектом.

Параметры
splitter – Геометрический объект, которым будет производиться разрезание объекта.

Результат
Возвращает полученный результат

```
poly = Polygon((0,10), (10,10), (10, 0))
line = Line((-5,5), (20,5))
r = poly.split_by_polyline(line)
print(r.wkt)
'''
>>> GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5, 5 5, 0 10)), POLYGON␣
↪((10 5, 10 0, 5 5, 10 5))) GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5,␣
↪5 5, 0 10)), POLYGON ((10 5, 10 0, 5 5, 10 5)))
'''
```

symmetric_difference(other: Geometry) → Geometry

Возвращает логический XOR областей геометрий (объединение разниц).

Параметры
other – Геометрия для анализа.

to_geojson() → str

Преобразует геометрию в формат „GeoJSON“

to_linestring() → Optional[Geometry]

Пробует геометрию преобразовать в линейный объект. В случае неудачи возвращает None.

to_mif() → str

Преобразует геометрию в формат MIF.

to_polygon() → Optional[Geometry]

Пробует геометрию преобразовать в площадной объект. В случае неудачи возвращает None.

to_wkb() → bytes

Возвращает WKB строку для геометрии.

to_wkt() → str

Возвращает WKT строку для геометрии.

touches(other: Geometry) → bool

Возвращает True, если геометрии соприкасаются.

property type: Type

Возвращает тип геометрического элемента.

Таблица 21: Возможные значения

Значение	Наименование
Unknown	Не определен
Point	Точка
Line	Линия
LineString	Полилиния
Polygon	Полигон
MultiPoint	Коллекция точек
MultiLineString	Коллекция полилиний
MultiPolygon	Коллекция полигонов
GeometryCollection	Смешанная коллекция
Arc	Дуга
Ellipse	Эллипс
Rectangle	Прямоугольник
RoundedRectangle	Скругленный прямоугольник
Text	Текст

Список 77: Пример.

```
point = Point(10, 10)
if point.type == GeometryType.Point:
    print('Это точка')
```

union(other: Geometry) → Geometry

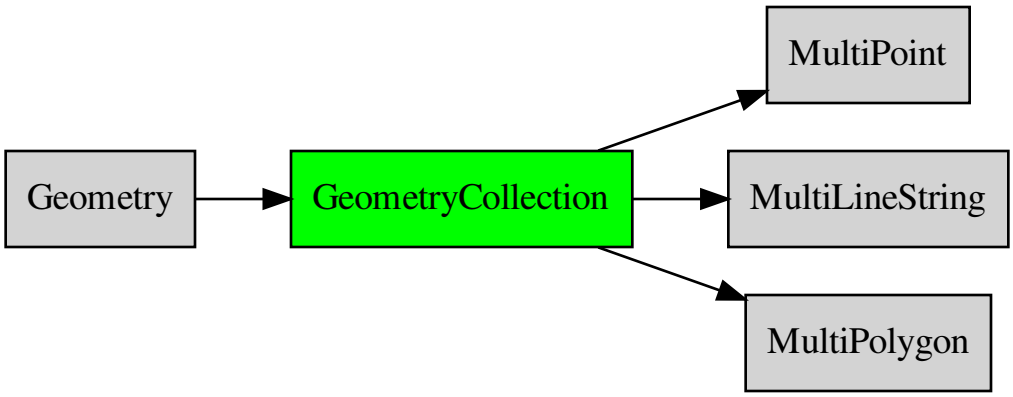
Возвращает результат объединения двух геометрий.

within(other: Geometry) → bool

Возвращает True, если геометрия находится полностью внутри геометрии other.

22.12.6 GeometryCollection - Коллекция геометрий

Иерархия геометрических классов:



class axipy.**GeometryCollection**

Базовые классы: Geometry

Коллекция разнотипных геометрических объектов. Допустимо хранение геометрических объектов различного типа, за исключением коллекций. Доступ к элементам производится по аналогии работы со списком list.

Параметры
cs – Система Координат, в которой создается геометрия.

Для получения размера коллекции используйте функцию len:

```
cnt = len(coll)
```

Доступ к элементам производится по индексу. Нумерация начинается с 0.

В качестве примера получим первый элемент:

```
coll[1]
```

Обновление геометрии в коллекции так же производится по ее индексу. Также допустимо изменение некоторых свойств геометрии в зависимости от ее типа.

```
coll.append((3, 4), (5, 5), (10, 0))  # Полилиния
coll.append(Polygon([(3, 4), (5, 5), (10, 0)]))  # Полигон

# Примеры установки элемента с индексом 1
coll[1] = (2, 2)  # Как точки с координатами (2,2)
coll[1] = [(101, 102), (103, 104), (105, 106)]  # Как полилинии
coll[1] = Polygon((101, 102), (103, 104), (105, 106))  # Как полигона

# Доступ к элементам коллекции::
```

(continues on next page)

(продолжение с предыдущей страницы)

```
for g in coll:
    print('Геометрия:', g)
```

Классовые методы:

distance_by_points(start, end[, cs])	Производит расчет расстояния между двумя точками и азимут от первой до второй точки.
from_geojson(json[, cs])	Возвращает геометрию из ее 'GeoJSON' представления.
from_mif(mif)	Возвращает геометрию из ее 'MIF' представления.
from_wkb(wkb[, coordsystem])	Создает геометрический объект из строки формата WKB.
from_wkt(wkt[, coordsystem])	Создает геометрический объект из строки формата WKT.
point_by_azimuth(point, azimuth, distance[, cs])	Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

Свойства:

bounds	Возвращает минимальный ограничивающий прямоугольник.
coordsystem	Система Координат (СК) геометрии.
is_valid	Проверяет геометрию на валидность.
is_valid_reason	Если геометрия неправильная, возвращает краткую аннотацию причины.
name	Возвращает наименование геометрического объекта.
type	Возвращает тип геометрического элемента.

Методы:

affine_transform(trans)	Трансформирует объект исходя из заданных параметров трансформации.
almost_equals(other, tolerance)	Производит примерное сравнения с другой геометрией в пределах заданной точности.
append(*value)	Добавление геометрии в коллекцию.
boundary()	Возвращает границы геометрии в виде полилинии.
buffer(distance[, resolution, capStyle, ...])	Производит построение буфера вокруг объекта.
centroid()	Возвращает центроид геометрии.
clone()	Создает копию объекта.

continues on next page

Таблица 22 – продолжение с предыдущей страницы

contains(other)	Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.
convex_hull()	Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.
covers(other)	Возвращает True, если геометрия охватывает геометрию other.
crosses(other)	Возвращает True, если при пересечении геометрий объекты частично пересекаются.
difference(other)	Возвращает область первой геометрии, которая не пересечена второй геометрией.
disjoint(other)	Возвращает True, если геометрии не пересекаются и не соприкасаются.
envelope()	Возвращает полигон, описывающий заданную геометрию.
equals(other)	Производит сравнение с другой геометрией.
get_area([area_unit])	Рассчитывает площадь, если объект площадной.
get_distance(other[, u])	Производит расчет расстояния до объекта other.
get_length([u])	Рассчитывает длину геометрии.
get_perimeter([u])	Рассчитывает периметр геометрии.
intersection(other)	Возвращает область пересечения с другой геометрией.
intersects(other)	Возвращает True, если геометрии пересекаются.
overlaps(other)	Возвращает True, если пересечение геометрий отличается от обеих геометрий.
relate(other)	Проверяет отношения между объектами.
remove(idx)	Удаление геометрии из коллекции.
reproject(cs)	Перепроецирует геометрию в другую систему координат.
rotate(point, angle)	Поворот геометрии относительно заданной точки.
scale(kx, ky)	Масштабирует объект по заданным коэффициентам масштабирования.
shift(dx, dy)	Смещает объект на заданную величину.
split_by_polyline(splitter)	Разрезает объект линейным объектом.
symmetric_difference(other)	Возвращает логический XOR областей геометрий (объединение разниц).
to_geojson()	Преобразует геометрию в формат 'GeoJSON'
to_linestring()	Пробует геометрию преобразовать в линейный объект.
to_mif()	Преобразует геометрию в формат MIF.

continues on next page

Таблица 22 – продолжение с предыдущей страницы

to_polygon()	Пробует геометрию преобразовать в площадной объект.
to_wkb()	Возвращает WKB строку для геометрии.
to_wkt()	Возвращает WKT строку для геометрии.
touches(other)	Возвращает True, если геометрии соприкасаются.
try_to_simplified()	Создает копию коллекции при этом пробует упростить ее тип.
union(other)	Возвращает результат объединения двух геометрий.
within(other)	Возвращает True, если геометрия находится полностью внутри геометрии other.

affine_transform(trans: QTransform) → Geometry

Трансформирует объект исходя из заданных параметров трансформации.

См.также:

Для простых операций типа сдвиг, масштабирование и поворот, рекомендуется использовать shift(), scale(), rotate() соответственно.

Параметры
trans – Матрица трансформации.

almost_equals(other: Geometry, tolerance: float) → bool

Производит примерное сравнения с другой геометрией в пределах заданной точности.

Параметры

- **other** – Сравниваемый объект.
- **tolerance** – Точность сравнения.

Результат
Возвращает True, если геометрии равны в пределах заданного отклонения.

append(*value: Union[Geometry, Pnt, Tuple[float, float]])

Добавление геометрии в коллекцию. Задание параметров аналогично указанию их при создании объектов конкретного типа. Если опустить указание класса, то для одной точки или пары значений float будет создан точечный объект Point, если точек больше, - объект класса LineString.

Список 78: Пример.

```
# Создадим коллекцию и добавим в нее несколько объектов различного типа.
coll = GeometryCollection()
coll.append((1, 2))  # Точка
coll.append(1, 2)  # Точка
coll.append(
    [(3, 4), (5, 5), (10, 0)]
)  # Полилиния в виде :class:`list`. Можно это сделать через конструктор␣
↪:class:`LinearString`.
```

boundary() → Geometry

Возвращает границы геометрии в виде полилинии.

property bounds: Rect

Возвращает минимальный ограничивающий прямоугольник.

buffer(distance: float, resolution: int = 16, capStyle: LineCapStyle = LineCapStyle.Round, join_style: LineJoinStyle = LineJoinStyle.Round, mitreLimit: float = 5.0) → Geometry

Производит построение буфера вокруг объекта.

Параметры

- **distance** – Ширина буфера.
- **resolution** – Количество сегментов на квадрант.
- **capStyle** – Стиль окончания.
- **join_style** – Стиль соединения.
- **mitreLimit** – Предел среза.

centroid() → Geometry

Возвращает центроид геометрии.

clone() → Geometry

Создает копию объекта.

contains(other: Geometry) → bool

Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.

convex_hull() → Geometry

Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.

property coordsystem: CoordSystem

Система Координат (СК) геометрии.

covers(other: Geometry) → bool

Возвращает True, если геометрия охватывает геометрию other.

crosses(other: Geometry) → bool

Возвращает True, если при пересечении геометрий объекты частично пересекаются.

difference(other: Geometry) → Geometry

Возвращает область первой геометрии, которая не пересечена второй геометрией.

disjoint(other: Geometry) → bool

Возвращает True, если геометрии не пересекаются и не соприкасаются.

static distance_by_points(start: Union[Pnt, Tuple[float, float]], end: Union[Pnt, Tuple[float, float]], cs: Optional[CoordSystem] = None) → Tuple

Производит расчет расстояния между двумя точками и азимут от первой до второй точки.

См.также:

Обратная функция `point_by_azimuth()`

Параметры

- **start** – Начальная точка. Если задана СК, то координаты в ней.
- **end** – Конечная точка. Если задана СК, то координаты в ней.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращается пара значений (расстояние в метрах, азимут в градусах)

envelope() → Geometry

Возвращает полигон, описывающий заданную геометрию.

equals(other: Geometry) → bool

Производит сравнение с другой геометрией.

Параметры

other – Сравниваемая геометрия.

Результат

Возвращает True, если геометрии равны.

static from_geojson(json: str, cs: Optional[CoordSystem] = None) → Geometry

Возвращает геометрию из ее „GeoJSON“ представления.

Параметры

- **json** – json представление в виде строки.
- **cs** – Система координат.

static from_mif(mif: str) → Geometry

Возвращает геометрию из ее „MIF“ представления.

Параметры

mif – mif представление в виде строки.

static from_wkb(wkb: bytes, coordsystem: Optional[CoordSystem] = None) → Geometry

Создает геометрический объект из строки формата WKB.

Параметры

- **wkb** – Строка WKB
- **coordsystem** – Система координат, которая будет установлена для геометрии.

Список 79: Пример.

```
wkb = b'\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00$@\x00\x00\x00\x00\x00\x00
↪$@'
pnt = Geometry.from_wkb(wkb)
```

static from_wkt(wkt: str, coordsystem: Optional[CoordSystem] = None) → Geometry

Создает геометрический объект из строки формата WKT.

Параметры

- **wkt** – Строка WKT. Допустимо задание строки в формате с указанием SRID (EWKT). В данном случае система координат для создаваемой геометрии будет установлено исходя их этого значения.
- **coordsystem** – Система координат, которая будет установлена для геометрии. Если строка задана в виде EWKT и указано значение SRID, игнорируется.

Список 80: Пример.

```
polygon = Geometry.from_wkt('POLYGON ((10 10, 100 100, 100 10, 10 10))')
point = Geometry.from_wkt('SRID=4326;POINT (10 10)')
crs = CoordSystem.from_epsg(4326)
pline = Geometry.from_wkt('LINESTRING (30 10, 10 30, 40 40)', crs)
```

get_area(area_unit: Optional[AreaUnit] = None) → float

Рассчитывает площадь, если объект площадной. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в квадратных метрах.

Список 81: Пример.

```
# В проекции Меркатора:
csMercatorKm = CoordSystem.from_prj("10, 104, 1, 0")
poly = Polygon([(0, 0), (0, 2), (2, 2), (2, 0)], cs=csMercatorKm)
print('Area Mercator km:', poly.get_area())
print('Area Mercator m:', poly.get_area(Unit.sq_m))
print('Perimeter Mercator km:', poly.get_perimeter())
print('Perimeter Mercator m:', poly.get_perimeter(Unit.m))
# В Широте/Долготе
poly.coordsystem = CoordSystem.from_prj("1, 104")
print('Area LL m:', poly.get_area())
print('Area LL km:', poly.get_area(Unit.sq_km))
print('Perimeter LL m:', poly.get_perimeter())
print('Perimeter LL km:', poly.get_perimeter(Unit.km))
'''
>>> Area Mercator km: 4.017946986632519
>>> Area Mercator m: 4017946.9866325194
>>> Perimeter Mercator km: 8.017972048421552
>>> Perimeter Mercator m: 8017.972048421552
>>> Area LL m: 49452172514.0342
>>> Area LL km: 49452.1725140342
>>> Perimeter LL m: 889423.5067063896
>>> Perimeter LL km: 889.4235067063896
'''
```

Параметры

area_unit – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и она не задана, то производится расчет на плоскости.

get_distance(other: Geometry, u: Optional[LinearUnit] = None) → float

Производит расчет расстояния до объекта other. Результат возвращает в СК текущего объекта.

Параметры

- **other** – Анализируемый объект.
- **u** – Единицы измерения, в которых требуется получить результат.

Список 82: Пример.

```
# Создадим геометрию без СК
first = Point(0, 0)
second = Point(10, 0)
print('В плане:', first.get_distance(second))
# Установим разные СК для точек
first.coordsystem = CoordSystem.from_prj("10, 104, 7, 0")
second.coordsystem = CoordSystem.from_prj("1, 104")
print('На сфере м:', first.get_distance(second))
print('На сфере км:', first.get_distance(second, Unit.km))
'''
>>> В плане: 10.0
>>> На сфере м: 1111948.7428468117
>>> На сфере км: 1111.9487428468117
'''
```

get_length(u: Optional[LinearUnit] = None) → float

Рассчитывает длину геометрии. Метод применим только для линейных объектов. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Список 83: Пример.

```
# В проекции Меркатора:
csMercatorKm = CoordSystem.from_prj("10, 104, 1, 0")
ls = Line((0, 0), (10, 0), csMercatorKm)
print('Length Mercator km:', ls.get_length())
print('Lenght Mercator m:', ls.get_length(Unit.m))
# В Широте/Долготе
csLL = CoordSystem.from_prj("1, 104")
ls = Line((0, 0), (10, 0), csLL)
print('Length LL m:', ls.get_length())
print('Length LL km:', ls.get_length(Unit.km))
'''
>>> Length Mercator km: 9.988805508567783
>>> Lenght Mercator m: 9988.805508567782
>>> Length LL m: 1111948.7428468117
>>> Length LL km: 1111.9487428468117
'''
```

Параметры

u – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и она не задана, то производится расчет на плоскости.

get_perimeter(u: Optional[LinearUnit] = None) → float

Рассчитывает периметр геометрии. Метод применим только для площадных

объектов. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Пример см. `get_area()`

Параметры

u – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и она не задана, то производится расчет на плоскости.

intersection(other: Geometry) → Geometry

Возвращает область пересечения с другой геометрией.

intersects(other: Geometry) → bool

Возвращает True, если геометрии пересекаются.

property is_valid: bool

Проверяет геометрию на валидность.

property is_valid_reason: str

Если геометрия неправильная, возвращает краткую аннотацию причины.

property name: str

Возвращает наименование геометрического объекта.

overlaps(other: Geometry) → bool

Возвращает True, если пересечение геометрий отличается от обеих геометрий.

static point_by_azimuth(point: Union[Pnt, Tuple[float, float]], azimuth: float, distance: float, cs: Optional[CoordSystem] = None) → Pnt

Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

См.также:

Обратная функция `distance_by_points()`

Параметры

- **point** – Точка, относительно которой производится расчет. Если задана СК, то в ней.
- **azimuth** – Азимут в градусах, указывающий направление.
- **distance** – Расстояние по азимуту. Задается в метрах.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращает результирующую точку.

relate(other: Geometry) → str

Проверяет отношения между объектами. Подробнее см. DE-9IM

remove(idx: int)

Удаление геометрии из коллекции.

Параметры

idx – Индекс геометрии в коллекции.

reproject(cs: CoordSystem) → Geometry

Перепроецирует геометрию в другую систему координат.

Параметры
cs – СК, в которой требуется получить объект.

rotate(point: Union[Pnt, Tuple[float, float]], angle: float) → Geometry

Поворот геометрии относительно заданной точки. Поворот производится против часовой стрелки.

Параметры

- **point** – Точка, вокруг которой необходимо произвести поворот.
- **angle** – Угол поворота в градусах.

scale(kx: float, ky: float) → Geometry

Масштабирует объект по заданным коэффициентам масштабирования. После выполнения операции центр результирующего объекта остается прежним.

Параметры

- **kx** – Масштабирование по координате X.
- **ky** – Масштабирование по координате Y.

shift(dx: float, dy: float) → Geometry

Смещает объект на заданную величину. Значения задаются в текущей проекции.

Параметры

- **dx** – Сдвиг по координате X.
- **dy** – Сдвиг по координате Y.

split_by_polyline(splitter: Geometry) → Optional[Geometry]

Разрезает объект линейным объектом.

Параметры
splitter – Геометрический объект, которым будет производиться разрезание объекта.

Результат
Возвращает полученный результат

```
poly = Polygon((0,10), (10,10), (10, 0))
line = Line((-5,5), (20,5))
r = poly.split_by_polyline(line)
print(r.wkt)
'''
>>> GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5, 5 5, 0 10)), POLYGON␣
↪((10 5, 10 0, 5 5, 10 5))) GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5,␣
↪5 5, 0 10)), POLYGON ((10 5, 10 0, 5 5, 10 5)))
'''
```

symmetric_difference(other: Geometry) → Geometry

Возвращает логический XOR областей геометрий (объединение разниц).

Параметры
other – Геометрия для анализа.

to_geojson() → str

Преобразует геометрию в формат „GeoJSON“

to_linestring() → Optional[Geometry]

Пробует геометрию преобразовать в линейный объект. В случае неудачи возвращает None.

to_mif() → str

Преобразует геометрию в формат MIF.

to_polygon() → Optional[Geometry]

Пробует геометрию преобразовать в площадной объект. В случае неудачи возвращает None.

to_wkb() → bytes

Возвращает WKB строку для геометрии.

to_wkt() → str

Возвращает WKT строку для геометрии.

touches(other: Geometry) → bool

Возвращает True, если геометрии соприкасаются.

try_to_simplified() → Union[MultiPoint, MultiLineString, MultiPolygon, GeometryCollection]

Создает копию коллекции при этом пробует упростить ее тип. К примеру, если в коллекции только полигоны, то вернет объект типа `MultiPolygon`. Если исходная коллекция разнородна, возвращается копия исходной.

property type: Type

Возвращает тип геометрического элемента.

Таблица 23: Возможные значения

Значение	Наименование
Unknown	Не определен
Point	Точка
Line	Линия
LineString	Полилиния
Polygon	Полигон
MultiPoint	Коллекция точек
MultiLineString	Коллекция полилиний
MultiPolygon	Коллекция полигонов
GeometryCollection	Смешанная коллекция
Arc	Дуга
Ellipse	Эллипс
Rectangle	Прямоугольник
RoundedRectangle	Скругленный прямоугольник
Text	Текст

Список 84: Пример.

```
point = Point(10, 10)
if point.type == GeometryType.Point:
    print('Это точка')
```

union(other: Geometry) → Geometry

Возвращает результат объединения двух геометрий.

within(other: Geometry) → bool

Возвращает True, если геометрия находится полностью внутри геометрии other.

22.12.7 MultiPoint - Коллекция точек

class axipy.**MultiPoint**

Базовые классы: GeometryCollection

Коллекция точечных объектов. Может содержать только объекты типа точка.

Параметры

cs – Система Координат, в которой создается геометрия.

Список 85: Пример.

```
mpoint = MultiPoint()  # Создаем коллекцию.
# Добавим точку разными способами.
p = Point(23, 34)
mpoint.append(p)
mpoint.append((12, 12))
mpoint.append(Pnt(10, 10))
mpoint[0] = (66, 66)  # Заменяем первый объект (индекс 0)
mpoint[0].x = 77  # Заменим только координату x для первого объекта в коллекции
mpoint.remove(1)  # Удаляем второй объект
```

Классовые методы:

distance_by_points(start, end[, cs])	Производит расчет расстояния между двумя точками и азимут от первой до второй точки.
from_geojson(json[, cs])	Возвращает геометрию из ее 'GeoJSON' представления.
from_mif(mif)	Возвращает геометрию из ее 'MIF' представления.
from_wkb(wkb[, coordsystem])	Создает геометрический объект из строки формата WKB.
from_wkt(wkt[, coordsystem])	Создает геометрический объект из строки формата WKT.
point_by_azimuth(point, azimuth, distance[, cs])	Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

Свойства:

bounds	Возвращает минимальный ограничивающий прямоугольник.
coordsystem	Система Координат (СК) геометрии.
is_valid	Проверяет геометрию на валидность.
is_valid_reason	Если геометрия неправильная, возвращает краткую аннотацию причины.
name	Возвращает наименование геометрического объекта.
type	Возвращает тип геометрического элемента.

Методы:

affine_transform(trans)	Трансформирует объект исходя из заданных параметров трансформации.
almost_equals(other, tolerance)	Производит примерное сравнения с другой геометрией в пределах заданной точности.
append(*value)	Добавление геометрии в коллекцию.
boundary()	Возвращает границы геометрии в виде полилинии.
buffer(distance[, resolution, capStyle, ...])	Производит построение буфера вокруг объекта.
centroid()	Возвращает центроид геометрии.
clone()	Создает копию объекта.
contains(other)	Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.
convex_hull()	Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.
covers(other)	Возвращает True, если геометрия охватывает геометрию other.
crosses(other)	Возвращает True, если при пересечении геометрий объекты частично пересекаются.
difference(other)	Возвращает область первой геометрии, которая не пересечена второй геометрией.
disjoint(other)	Возвращает True, если геометрии не пересекаются и не соприкасаются.
envelope()	Возвращает полигон, описывающий заданную геометрию.
equals(other)	Производит сравнение с другой геометрией.
get_area([area_unit])	Рассчитывает площадь, если объект площадной.
get_distance(other[, u])	Производит расчет расстояния до объекта other.
get_length([u])	Рассчитывает длину геометрии.

continues on next page

Таблица 24 – продолжение с предыдущей страницы

get_perimeter([u])	Рассчитывает периметр геометрии.
intersection(other)	Возвращает область пересечения с другой геометрией.
intersects(other)	Возвращает True, если геометрии пересекаются.
overlaps(other)	Возвращает True, если пересечение геометрий отличается от обеих геометрий.
relate(other)	Проверяет отношения между объектами.
remove(idx)	Удаление геометрии из коллекции.
reproject(cs)	Перепроецирует геометрию в другую систему координат.
rotate(point, angle)	Поворот геометрии относительно заданной точки.
scale(kx, ky)	Масштабирует объект по заданным коэффициентам масштабирования.
shift(dx, dy)	Смещает объект на заданную величину.
split_by_polyline(splitter)	Разрезает объект линейным объектом.
symmetric_difference(other)	Возвращает логический XOR областей геометрий (объединение разниц).
to_geojson()	Преобразует геометрию в формат 'GeoJSON'
to_linestring()	Пробует геометрию преобразовать в линейный объект.
to_mif()	Преобразует геометрию в формат MIF.
to_polygon()	Пробует геометрию преобразовать в площадной объект.
to_wkb()	Возвращает WKB строку для геометрии.
to_wkt()	Возвращает WKT строку для геометрии.
touches(other)	Возвращает True, если геометрии соприкасаются.
try_to_simplified()	Создает копию коллекции при этом пробует упростить ее тип.
union(other)	Возвращает результат объединения двух геометрий.
within(other)	Возвращает True, если геометрия находится полностью внутри геометрии other.

affine_transform(trans: QTransform) → Geometry

Трансформирует объект исходя из заданных параметров трансформации.

См.также:

Для простых операций типа сдвиг, масштабирование и поворот, рекомендуется использовать shift(), scale(), rotate() соответственно.

Параметры

trans – Матрица трансформации.

almost_equals(other: Geometry, tolerance: float) → bool

Производит примерное сравнения с другой геометрией в пределах заданной точности.

Параметры

- **other** – Сравниваемый объект.
- **tolerance** – Точность сравнения.

Результат

Возвращает True, если геометрии равны в пределах заданного отклонения.

append(*value: Union[Geometry, Pnt, Tuple[float, float]])

Добавление геометрии в коллекцию. Задание параметров аналогично указанию их при создании объектов конкретного типа. Если опустить указание класса, то для одной точки или пары значений float будет создан точечный объект `Point`, если точек больше, - объект класса `LineString`.

Список 86: Пример.

```
# Создадим коллекцию и добавим в нее несколько объектов различного типа.
coll = GeometryCollection()
coll.append((1, 2))  # Точка
coll.append(1, 2)  # Точка
coll.append(
    [(3, 4), (5, 5), (10, 0)]
)  # Полилиния в виде :class:`list`. Можно это сделать через конструктор␣
↪:class:`LinearString`.
```

boundary() → Geometry

Возвращает границы геометрии в виде полилинии.

property bounds: Rect

Возвращает минимальный ограничивающий прямоугольник.

buffer(distance: float, resolution: int = 16, capStyle: LineCapStyle = LineCapStyle.Round, join_style: LineJoinStyle = LineJoinStyle.Round, mitreLimit: float = 5.0) → Geometry

Производит построение буфера вокруг объекта.

Параметры

- **distance** – Ширина буфера.
- **resolution** – Количество сегментов на квадрант.
- **capStyle** – Стиль окончания.
- **join_style** – Стиль соединения.
- **mitreLimit** – Предел среза.

centroid() → Geometry

Возвращает центроид геометрии.

clone() → Geometry

Создает копию объекта.

contains(other: Geometry) → bool

Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.

convex_hull() → Geometry

Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.

property coordsystem: CoordSystem

Система Координат (СК) геометрии.

covers(other: Geometry) → bool

Возвращает True, если геометрия охватывает геометрию other.

crosses(other: Geometry) → bool

Возвращает True, если при пересечении геометрий объекты частично пересекаются.

difference(other: Geometry) → Geometry

Возвращает область первой геометрии, которая не пересечена второй геометрией.

disjoint(other: Geometry) → bool

Возвращает True, если геометрии не пересекаются и не соприкасаются.

static distance_by_points(start: Union[Pnt, Tuple[float, float]], end: Union[Pnt, Tuple[float, float]], cs: Optional[CoordSystem] = None) → Tuple

Производит расчет расстояния между двумя точками и азимут от первой до второй точки.

См.также:

Обратная функция `point_by_azimuth()`

Параметры

- **start** – Начальная точка. Если задана СК, то координаты в ней.
- **end** – Конечная точка. Если задана СК, то координаты в ней.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращается пара значений (расстояние в метрах, азимут в градусах)

envelope() → Geometry

Возвращает полигон, описывающий заданную геометрию.

equals(other: Geometry) → bool

Производит сравнение с другой геометрией.

Параметры

other – Сравниваемая геометрия.

Результат

Возвращает True, если геометрии равны.

static from_geojson(json: str, cs: Optional[CoordSystem] = None) → Geometry

Возвращает геометрию из ее „GeoJSON“ представления.

Параметры

- **json** – json представление в виде строки.
- **cs** – Система координат.

static from_mif(mif: str) → Geometry

Возвращает геометрию из ее „MIF“ представления.

Параметры

mif – mif представление в виде строки.

static from_wkb(wkb: bytes, coordsystem: Optional[CoordSystem] = None) → Geometry

Создает геометрический объект из строки формата WKB.

Параметры

- **wkb** – Строка WKB
- **coordsystem** – Система координат, которая будет установлена для геометрии.

Список 87: Пример.

```
wkb = b'\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00$@\x00\x00\x00\x00\x00\x00
↪$@'
pnt = Geometry.from_wkb(wkb)
```

static from_wkt(wkt: str, coordsystem: Optional[CoordSystem] = None) → Geometry

Создает геометрический объект из строки формата WKT.

Параметры

- **wkt** – Строка WKT. Допустимо задание строки в формате с указанием SRID (EWKT). В данном случае система координат для создаваемой геометрии будет установлено исходя их этого значения.
- **coordsystem** – Система координат, которая будет установлена для геометрии. Если строка задана в виде EWKT и указано значение SRID, игнорируется.

Список 88: Пример.

```
polygon = Geometry.from_wkt('POLYGON ((10 10, 100 100, 100 10, 10 10))')
point = Geometry.from_wkt('SRID=4326;POINT (10 10)')
crs = CoordSystem.from_epsg(4326)
pline = Geometry.from_wkt('LINESTRING (30 10, 10 30, 40 40)', crs)
```

get_area(area_unit: Optional[AreaUnit] = None) → float

Рассчитывает площадь, если объект площадной. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в квадратных метрах.

Список 89: Пример.

```
# В проекции Меркатора:
csMercatorKm = CoordSystem.from_prj("10, 104, 1, 0")
poly = Polygon([(0, 0), (0, 2), (2, 2), (2, 0)], cs=csMercatorKm)
print('Area Mercator km:', poly.get_area())
print('Area Mercator m:', poly.get_area(Unit.sq_m))
print('Perimeter Mercator km:', poly.get_perimeter())
print('Perimeter Mercator m:', poly.get_perimeter(Unit.m))
# В Широте/Долготе
poly.coordsystem = CoordSystem.from_prj("1, 104")
print('Area LL m:', poly.get_area())
print('Area LL km:', poly.get_area(Unit.sq_km))
print('Perimeter LL m:', poly.get_perimeter())
print('Perimeter LL km:', poly.get_perimeter(Unit.km))
'''
>>> Area Mercator km: 4.017946986632519
>>> Area Mercator m: 4017946.9866325194
>>> Perimeter Mercator km: 8.017972048421552
>>> Perimeter Mercator m: 8017.972048421552
>>> Area LL m: 49452172514.0342
>>> Area LL km: 49452.1725140342
>>> Perimeter LL m: 889423.5067063896
>>> Perimeter LL km: 889.4235067063896
'''
```

Параметры

area_unit – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и она не задана, то производится расчет на плоскости.

get_distance(other: Geometry, u: Optional[LinearUnit] = None) → float

Производит расчет расстояния до объекта other. Результат возвращает в СК текущего объекта.

Параметры

- **other** – Анализируемый объект.
- **u** – Единицы измерения, в которых требуется получить результат.

Список 90: Пример.

```
# Создадим геометрию без СК
first = Point(0, 0)
second = Point(10, 0)
print('В плане:', first.get_distance(second))
# Установим разные СК для точек
first.coordsystem = CoordSystem.from_prj("10, 104, 7, 0")
second.coordsystem = CoordSystem.from_prj("1, 104")
print('На сфере м:', first.get_distance(second))
print('На сфере км:', first.get_distance(second, Unit.km))
'''
>>> В плане: 10.0
>>> На сфере м: 1111948.7428468117
>>> На сфере км: 1111.9487428468117
'''
```

get_length(u: Optional[LinearUnit] = None) → float

Рассчитывает длину геометрии. Метод применим только для линейных объектов. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Список 91: Пример.

```
# В проекции Меркатора:
csMercatorKm = CoordSystem.from_prj("10, 104, 1, 0")
ls = Line((0, 0), (10, 0), csMercatorKm)
print('Length Mercator km:', ls.get_length())
print('Lenght Mercator m:', ls.get_length(Unit.m))
# В Широте/Долготе
csLL = CoordSystem.from_prj("1, 104")
ls = Line((0, 0), (10, 0), csLL)
print('Length LL m:', ls.get_length())
print('Length LL km:', ls.get_length(Unit.km))
'''
>>> Length Mercator km: 9.988805508567783
>>> Lenght Mercator m: 9988.805508567782
>>> Length LL m: 1111948.7428468117
>>> Length LL km: 1111.9487428468117
'''
```

Параметры

u – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и она не задана, то производится расчет на плоскости.

get_perimeter(u: Optional[LinearUnit] = None) → float

Рассчитывает периметр геометрии. Метод применим только для площадных объектов. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Пример см. `get_area()`

Параметры

u – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и она не задана, то производится расчет на плоскости.

intersection(other: Geometry) → Geometry

Возвращает область пересечения с другой геометрией.

intersects(other: Geometry) → bool

Возвращает True, если геометрии пересекаются.

property is_valid: bool

Проверяет геометрию на валидность.

property is_valid_reason: str

Если геометрия неправильная, возвращает краткую аннотацию причины.

property name: str

Возвращает наименование геометрического объекта.

overlaps(other: Geometry) → bool

Возвращает True, если пересечение геометрий отличается от обеих геометрий.

static point_by_azimuth(point: Union[Pnt, Tuple[float, float]], azimuth: float, distance: float, cs: Optional[CoordSystem] = None) → Pnt

Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

См.также:

Обратная функция `distance_by_points()`

Параметры

- **point** – Точка, относительно которой производится расчет. Если задана СК, то в ней.
- **azimuth** – Азимут в градусах, указывающий направление.
- **distance** – Расстояние по азимуту. Задается в метрах.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращает результирующую точку.

relate(other: Geometry) → str

Проверяет отношения между объектами. Подробнее см. DE-9IM

remove(idx: int)

Удаление геометрии из коллекции.

Параметры

idx – Индекс геометрии в коллекции.

reproject(cs: CoordSystem) → Geometry

Перепроецирует геометрию в другую систему координат.

Параметры

cs – СК, в которой требуется получить объект.

rotate(point: Union[Pnt, Tuple[float, float]], angle: float) → Geometry

Поворот геометрии относительно заданной точки. Поворот производится против часовой стрелки.

Параметры

- **point** – Точка, вокруг которой необходимо произвести поворот.
- **angle** – Угол поворота в градусах.

scale(kx: float, ky: float) → Geometry

Масштабирует объект по заданным коэффициентам масштабирования. После выполнения операции центр результирующего объекта остается прежним.

Параметры

- **kx** – Масштабирование по координате X.
- **ky** – Масштабирование по координате Y.

shift(dx: float, dy: float) → Geometry

Смещает объект на заданную величину. Значения задаются в текущей проекции.

Параметры

- **dx** – Сдвиг по координате X.
- **dy** – Сдвиг по координате Y.

split_by_polyline(splitter: Geometry) → Optional[Geometry]

Разрезает объект линейным объектом.

Параметры
splitter – Геометрический объект, которым будет производиться разрезание объекта.

Результат
Возвращает полученный результат

```
poly = Polygon((0,10), (10,10), (10, 0))
line = Line((-5,5), (20,5))
r = poly.split_by_polyline(line)
print(r.wkt)
'''
>>> GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5, 5 5, 0 10)), POLYGON␣
↪((10 5, 10 0, 5 5, 10 5))) GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5,␣
↪5 5, 0 10)), POLYGON ((10 5, 10 0, 5 5, 10 5)))
'''
```

symmetric_difference(other: Geometry) → Geometry

Возвращает логический XOR областей геометрий (объединение разниц).

Параметры
other – Геометрия для анализа.

to_geojson() → str

Преобразует геометрию в формат „GeoJSON“

to_linestring() → Optional[Geometry]

Пробует геометрию преобразовать в линейный объект. В случае неудачи возвращает None.

to_mif() → str

Преобразует геометрию в формат MIF.

to_polygon() → Optional[Geometry]

Пробует геометрию преобразовать в площадной объект. В случае неудачи возвращает None.

to_wkb() → bytes

Возвращает WKB строку для геометрии.

to_wkt() → str

Возвращает WKT строку для геометрии.

touches(other: Geometry) → bool

Возвращает True, если геометрии соприкасаются.

try_to_simplified() → Union[MultiPoint, MultiLineString, MultiPolygon, GeometryCollection]

Создает копию коллекции при этом пробует упростить ее тип. К примеру, если в коллекции только полигоны, то вернет объект типа `MultiPolygon`. Если исходная коллекция разнородна, возвращается копия исходной.

property type: Type

Возвращает тип геометрического элемента.

Таблица 25: Возможные значения

Значение	Наименование
Unknown	Не определен
Point	Точка
Line	Линия
LineString	Полилиния
Polygon	Полигон
MultiPoint	Коллекция точек
MultiLineString	Коллекция полилиний
MultiPolygon	Коллекция полигонов
GeometryCollection	Смешанная коллекция
Arc	Дуга
Ellipse	Эллипс
Rectangle	Прямоугольник
RoundedRectangle	Скругленный прямоугольник
Text	Текст

Список 92: Пример.

```
point = Point(10, 10)
if point.type == GeometryType.Point:
    print('Это точка')
```

union(other: Geometry) → Geometry

Возвращает результат объединения двух геометрий.

within(other: Geometry) → bool

Возвращает True, если геометрия находится полностью внутри геометрии other.

22.12.8 MultiLineString - Коллекция полилиний

class axipy.**MultiLineString**

Базовые классы: GeometryCollection

Коллекция полилиний. Может содержать только объекты типа полилиния.

Параметры

cs – Система Координат, в которой создается геометрия.

Список 93: Пример.

```
msl = MultiLineString()  # Создадим саму коллекцию.
ls = LineString([(1, 2), (3, 4), (5, 6), (7, 8)])
```

(continues on next page)

(продолжение с предыдущей страницы)

```
msl.append(ls)  # Добавим как объект по ссылке
msl.append(LineString([(11, 12), (13, 14), (15, 16)]))  # Добавим как объект
msl.append([(21, 22), (23, 24), (25, 26)])  # Добавим как перечень точек как
↪:class:`list`
msl[2].points[1] = (101, 102)  # Обновим значение точки 3 полилинии по индексу 2.
msl.remove(0)  # Удалим первый объект из коллекции.
msl[1].points.remove(2)  # Удалим точку с индексом 1 из полилинии 2
msl[0] = [(101, 102), (103, 104), (105, 106), (107, 108)]  # Обновим первую
↪геометрию
```

Классовые методы:

distance_by_points(start, end[, cs])	Производит расчет расстояния между двумя точками и азимут от первой до второй точки.
from_geojson(json[, cs])	Возвращает геометрию из ее 'GeoJSON' представления.
from_mif(mif)	Возвращает геометрию из ее 'MIF' представления.
from_wkb(wkb[, coordsystem])	Создает геометрический объект из строки формата WKB.
from_wkt(wkt[, coordsystem])	Создает геометрический объект из строки формата WKT.
point_by_azimuth(point, azimuth, distance[, cs])	Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

Свойства:

bounds	Возвращает минимальный ограничивающий прямоугольник.
coordsystem	Система Координат (СК) геометрии.
is_valid	Проверяет геометрию на валидность.
is_valid_reason	Если геометрия неправильная, возвращает краткую аннотацию причины.
name	Возвращает наименование геометрического объекта.
type	Возвращает тип геометрического элемента.

Методы:

affine_transform(trans)	Трансформирует объект исходя из заданных параметров трансформации.
almost_equals(other, tolerance)	Производит примерное сравнения с другой геометрией в пределах заданной точности.
append(*value)	Добавление геометрии в коллекцию.

continues on next page

Таблица 26 – продолжение с предыдущей страницы

boundary()	Возвращает границы геометрии в виде полилинии.
buffer(distance[, resolution, capStyle, ...])	Производит построение буфера вокруг объекта.
centroid()	Возвращает центроид геометрии.
clone()	Создает копию объекта.
contains(other)	Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.
convex_hull()	Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.
covers(other)	Возвращает True, если геометрия охватывает геометрию other.
crosses(other)	Возвращает True, если при пересечении геометрий объекты частично пересекаются.
difference(other)	Возвращает область первой геометрии, которая не пересечена второй геометрией.
disjoint(other)	Возвращает True, если геометрии не пересекаются и не соприкасаются.
envelope()	Возвращает полигон, описывающий заданную геометрию.
equals(other)	Производит сравнение с другой геометрией.
get_area([area_unit])	Рассчитывает площадь, если объект площадной.
get_distance(other[, u])	Производит расчет расстояния до объекта other.
get_length([u])	Рассчитывает длину геометрии.
get_perimeter([u])	Рассчитывает периметр геометрии.
intersection(other)	Возвращает область пересечения с другой геометрией.
intersects(other)	Возвращает True, если геометрии пересекаются.
overlaps(other)	Возвращает True, если пересечение геометрий отличается от обеих геометрий.
relate(other)	Проверяет отношения между объектами.
remove(idx)	Удаление геометрии из коллекции.
reproject(cs)	Перепроецирует геометрию в другую систему координат.
rotate(point, angle)	Поворот геометрии относительно заданной точки.
scale(kx, ky)	Масштабирует объект по заданным коэффициентам масштабирования.
shift(dx, dy)	Смещает объект на заданную величину.
split_by_polyline(splitter)	Разрезает объект линейным объектом.

continues on next page

Таблица 26 – продолжение с предыдущей страницы

symmetric_difference(other)	Возвращает логический XOR областей геометрий (объединение разниц).
to_geojson()	Преобразует геометрию в формат 'GeoJSON'
to_linestring()	Пробует геометрию преобразовать в линейный объект.
to_mif()	Преобразует геометрию в формат MIF.
to_polygon()	Пробует геометрию преобразовать в площадной объект.
to_wkb()	Возвращает WKB строку для геометрии.
to_wkt()	Возвращает WKT строку для геометрии.
touches(other)	Возвращает True, если геометрии соприкасаются.
try_to_simplified()	Создает копию коллекции при этом пробует упростить ее тип.
union(other)	Возвращает результат объединения двух геометрий.
within(other)	Возвращает True, если геометрия находится полностью внутри геометрии other.

affine_transform(trans: QTransform) → Geometry

Трансформирует объект исходя из заданных параметров трансформации.

См.также:

Для простых операций типа сдвиг, масштабирование и поворот, рекомендуется использовать shift(), scale(), rotate() соответственно.

Параметры
trans – Матрица трансформации.

almost_equals(other: Geometry, tolerance: float) → bool

Производит примерное сравнения с другой геометрией в пределах заданной точности.

Параметры

- **other** – Сравниваемый объект.
- **tolerance** – Точность сравнения.

Результат
Возвращает True, если геометрии равны в пределах заданного отклонения.

append(*value: Union[Geometry, Pnt, Tuple[float, float]])

Добавление геометрии в коллекцию. Задание параметров аналогично указанию их при создании объектов конкретного типа. Если опустить указание класса, то для одной точки или пары значений float будет создан точечный объект Point, если точек больше, - объект класса LineString.

Список 94: Пример.

```
# Создадим коллекцию и добавим в нее несколько объектов различного типа.
coll = GeometryCollection()
coll.append((1, 2))  # Точка
coll.append(1, 2)  # Точка
coll.append(
    [(3, 4), (5, 5), (10, 0)]
)  # Полилиния в виде :class:`list`. Можно это сделать через конструктор␣
↪:class:`LinearString`.
```

boundary() → Geometry

Возвращает границы геометрии в виде полилинии.

property bounds: Rect

Возвращает минимальный ограничивающий прямоугольник.

buffer(distance: float, resolution: int = 16, capStyle: LineCapStyle = LineCapStyle.Round, join_style: LineJoinStyle = LineJoinStyle.Round, mitreLimit: float = 5.0) → Geometry

Производит построение буфера вокруг объекта.

Параметры

- **distance** – Ширина буфера.
- **resolution** – Количество сегментов на квадрант.
- **capStyle** – Стиль окончания.
- **join_style** – Стиль соединения.
- **mitreLimit** – Предел среза.

centroid() → Geometry

Возвращает центроид геометрии.

clone() → Geometry

Создает копию объекта.

contains(other: Geometry) → bool

Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.

convex_hull() → Geometry

Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.

property coordsystem: CoordSystem

Система Координат (СК) геометрии.

covers(other: Geometry) → bool

Возвращает True, если геометрия охватывает геометрию other.

crosses(other: Geometry) → bool

Возвращает True, если при пересечении геометрий объекты частично пересекаются.

difference(other: Geometry) → Geometry

Возвращает область первой геометрии, которая не пересечена второй геометрией.

disjoint(other: Geometry) → bool

Возвращает True, если геометрии не пересекаются и не соприкасаются.

static distance_by_points(start: Union[Pnt, Tuple[float, float]], end: Union[Pnt, Tuple[float, float]], cs: Optional[CoordSystem] = None) → Tuple

Производит расчет расстояния между двумя точками и азимут от первой до второй точки.

См.также:

Обратная функция `point_by_azimuth()`

Параметры

- **start** – Начальная точка. Если задана СК, то координаты в ней.
- **end** – Конечная точка. Если задана СК, то координаты в ней.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращается пара значений (расстояние в метрах, азимут в градусах)

envelope() → Geometry

Возвращает полигон, описывающий заданную геометрию.

equals(other: Geometry) → bool

Производит сравнение с другой геометрией.

Параметры

other – Сравниваемая геометрия.

Результат

Возвращает True, если геометрии равны.

static from_geojson(json: str, cs: Optional[CoordSystem] = None) → Geometry

Возвращает геометрию из ее „GeoJSON“ представления.

Параметры

- **json** – json представление в виде строки.
- **cs** – Система координат.

static from_mif(mif: str) → Geometry

Возвращает геометрию из ее „MIF“ представления.

Параметры

mif – mif представление в виде строки.

static from_wkb(wkb: bytes, coordsystem: Optional[CoordSystem] = None) → Geometry

Создает геометрический объект из строки формата WKB.

Параметры

- **wkb** – Строка WKB
- **coordsystem** – Система координат, которая будет установлена для геометрии.

Список 95: Пример.

```
wkb = b'\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00$@\x00\x00\x00\x00\x00\x00
↪$@'
pnt = Geometry.from_wkb(wkb)
```

static from_wkt(wkt: str, coordsystem: Optional[CoordSystem] = None) → Geometry

Создает геометрический объект из строки формата WKT.

Параметры

- **wkt** – Строка WKT. Допустимо задание строки в формате с указанием SRID (EWKT). В данном случае система координат для создаваемой геометрии будет установлено исходя их этого значения.
- **coordsystem** – Система координат, которая будет установлена для геометрии. Если строка задана в виде EWKT и указано значение SRID, игнорируется.

Список 96: Пример.

```
polygon = Geometry.from_wkt('POLYGON ((10 10, 100 100, 100 10, 10 10))')
point = Geometry.from_wkt('SRID=4326;POINT (10 10)')
crs = CoordSystem.from_epsg(4326)
pline = Geometry.from_wkt('LINESTRING (30 10, 10 30, 40 40)', crs)
```

get_area(area_unit: Optional[AreaUnit] = None) → float

Рассчитывает площадь, если объект площадной. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в квадратных метрах.

Список 97: Пример.

```
# В проекции Меркатора:
csMercatorKm = CoordSystem.from_prj("10, 104, 1, 0")
poly = Polygon([(0, 0), (0, 2), (2, 2), (2, 0)], cs=csMercatorKm)
print('Area Mercator km:', poly.get_area())
print('Area Mercator m:', poly.get_area(Unit.sq_m))
print('Perimeter Mercator km:', poly.get_perimeter())
print('Perimeter Mercator m:', poly.get_perimeter(Unit.m))
# В Широте/Долготе
poly.coordsystem = CoordSystem.from_prj("1, 104")
print('Area LL m:', poly.get_area())
print('Area LL km:', poly.get_area(Unit.sq_km))
print('Perimeter LL m:', poly.get_perimeter())
print('Perimeter LL km:', poly.get_perimeter(Unit.km))
'''
>>> Area Mercator km: 4.017946986632519
>>> Area Mercator m: 4017946.9866325194
>>> Perimeter Mercator km: 8.017972048421552
```

(continues on next page)

(продолжение с предыдущей страницы)

```
>>> Perimeter Mercator m: 8017.972048421552
>>> Area LL m: 49452172514.0342
>>> Area LL km: 49452.1725140342
>>> Perimeter LL m: 889423.5067063896
>>> Perimeter LL km: 889.4235067063896
'''
```

Параметры

area_unit – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и она не задана, то производится расчет на плоскости.

get_distance(other: Geometry, u: Optional[LinearUnit] = None) → float

Производит расчет расстояния до объекта other. Результат возвращает в СК текущего объекта.

Параметры

- **other** – Анализируемый объект.
- **u** – Единицы измерения, в которых требуется получить результат.

Список 98: Пример.

```
# Создадим геометрию без СК
first = Point(0, 0)
second = Point(10, 0)
print('В плане:', first.get_distance(second))
# Установим разные СК для точек
first.coordsystem = CoordSystem.from_prj("10, 104, 7, 0")
second.coordsystem = CoordSystem.from_prj("1, 104")
print('На сфере м:', first.get_distance(second))
print('На сфере км:', first.get_distance(second, Unit.km))
'''
>>> В плане: 10.0
>>> На сфере м: 1111948.7428468117
>>> На сфере км: 1111.9487428468117
'''
```

get_length(u: Optional[LinearUnit] = None) → float

Рассчитывает длину геометрии. Метод применим только для линейных объектов. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Список 99: Пример.

```
# В проекции Меркатора:
csMercatorKm = CoordSystem.from_prj("10, 104, 1, 0")
ls = Line((0, 0), (10, 0), csMercatorKm)
print('Length Mercator km:', ls.get_length())
print('Lenght Mercator m:', ls.get_length(Unit.m))
# В Широте/Долготе
csLL = CoordSystem.from_prj("1, 104")
ls = Line((0, 0), (10, 0), csLL)
print('Length LL m:', ls.get_length())
print('Length LL km:', ls.get_length(Unit.km))
```

(continues on next page)

(продолжение с предыдущей страницы)

```
'''
>>> Length Mercator km: 9.988805508567783
>>> Lenght Mercator m: 9988.805508567782
>>> Length LL m: 1111948.7428468117
>>> Length LL km: 1111.9487428468117
'''
```

Параметры

u – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и она не задана, то производится расчет на плоскости.

get_perimeter(u: Optional[LinearUnit] = None) → float

Рассчитывает периметр геометрии. Метод применим только для площадных объектов. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Пример см. `get_area()`

Параметры

u – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и она не задана, то производится расчет на плоскости.

intersection(other: Geometry) → Geometry

Возвращает область пересечения с другой геометрией.

intersects(other: Geometry) → bool

Возвращает True, если геометрии пересекаются.

property is_valid: bool

Проверяет геометрию на валидность.

property is_valid_reason: str

Если геометрия неправильная, возвращает краткую аннотацию причины.

property name: str

Возвращает наименование геометрического объекта.

overlaps(other: Geometry) → bool

Возвращает True, если пересечение геометрий отличается от обеих геометрий.

static point_by_azimuth(point: Union[Pnt, Tuple[float, float]], azimuth: float, distance: float, cs: Optional[CoordSystem] = None) → Pnt

Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

См.также:

Обратная функция `distance_by_points()`

Параметры

- **point** – Точка, относительно которой производится расчет. Если задана СК, то в ней.
- **azimuth** – Азимут в градусах, указывающий направление.

- **distance** – Расстояние по азимуту. Задается в метрах.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат
Возвращает результирующую точку.

relate(other: Geometry) → str

Проверяет отношения между объектами. Подробнее см. DE-9IM

remove(idx: int)

Удаление геометрии из коллекции.

Параметры
idx – Индекс геометрии в коллекции.

reproject(cs: CoordSystem) → Geometry

Перепроецирует геометрию в другую систему координат.

Параметры
cs – СК, в которой требуется получить объект.

rotate(point: Union[Pnt, Tuple[float, float]], angle: float) → Geometry

Поворот геометрии относительно заданной точки. Поворот производится против часовой стрелки.

Параметры

- **point** – Точка, вокруг которой необходимо произвести поворот.
- **angle** – Угол поворота в градусах.

scale(kx: float, ky: float) → Geometry

Масштабирует объект по заданным коэффициентам масштабирования. После выполнения операции центр результирующего объекта остается прежним.

Параметры

- **kx** – Масштабирование по координате X.
- **ky** – Масштабирование по координате Y.

shift(dx: float, dy: float) → Geometry

Смещает объект на заданную величину. Значения задаются в текущей проекции.

Параметры

- **dx** – Сдвиг по координате X.
- **dy** – Сдвиг по координате Y.

split_by_polyline(splitter: Geometry) → Optional[Geometry]

Разрезает объект линейным объектом.

Параметры
splitter – Геометрический объект, которым будет производиться разрезание объекта.

Результат
Возвращает полученный результат

```
poly = Polygon((0,10), (10,10), (10, 0))
line = Line((-5,5), (20,5))
r = poly.split_by_polyline(line)
print(r.wkt)
'''
>>> GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5, 5 5, 0 10)), POLYGON␣
↪((10 5, 10 0, 5 5, 10 5))) GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5,␣
↪5 5, 0 10)), POLYGON ((10 5, 10 0, 5 5, 10 5)))
'''
```

symmetric_difference(other: Geometry) → Geometry

Возвращает логический XOR областей геометрий (объединение разниц).

Параметры
other – Геометрия для анализа.

to_geojson() → str

Преобразует геометрию в формат „GeoJSON“

to_linestring() → Optional[Geometry]

Пробует геометрию преобразовать в линейный объект. В случае неудачи возвращает None.

to_mif() → str

Преобразует геометрию в формат MIF.

to_polygon() → Optional[Geometry]

Пробует геометрию преобразовать в площадной объект. В случае неудачи возвращает None.

to_wkb() → bytes

Возвращает WKB строку для геометрии.

to_wkt() → str

Возвращает WKT строку для геометрии.

touches(other: Geometry) → bool

Возвращает True, если геометрии соприкасаются.

try_to_simplified() → Union[MultiPoint, MultiLineString, MultiPolygon, GeometryCollection]

Создает копию коллекции при этом пробует упростить ее тип. К примеру, если в коллекции только полигоны, то вернет объект типа `MultiPolygon`. Если исходная коллекция разнородна, возвращается копия исходной.

property type: Type

Возвращает тип геометрического элемента.

Таблица 27: Возможные значения

Значение	Наименование
Unknown	Не определен
Point	Точка
Line	Линия
LineString	Полилиния
Polygon	Полигон
MultiPoint	Коллекция точек
MultiLineString	Коллекция полилиний
MultiPolygon	Коллекция полигонов
GeometryCollection	Смешанная коллекция
Arc	Дуга
Ellipse	Эллипс
Rectangle	Прямоугольник
RoundedRectangle	Скругленный прямоугольник
Text	Текст

Список 100: Пример.

```
point = Point(10, 10)
if point.type == GeometryType.Point:
    print('Это точка')
```

union(other: Geometry) → Geometry

Возвращает результат объединения двух геометрий.

within(other: Geometry) → bool

Возвращает True, если геометрия находится полностью внутри геометрии other.

22.12.9 MultiPolygon - Коллекция полигонов

class axipy.**MultiPolygon**

Базовые классы: GeometryCollection

Коллекция полигонов. Может содержать только объекты типа полигон.

Параметры

cs – Система Координат, в которой создается геометрия.

Список 101: Пример.

```
poly = Polygon((0, 0), (1, 10), (10, 1))
poly.holes.append([(2, 2), (2, 4), (5, 3)])  # Добавим дырку
mpoly = MultiPolygon()  # Создадим саму коллекцию.
mpoly.append([(1, 2), (3, 4), (5, 6), (7, 8)])  # Добавим полигон в виде списка точек
mpoly.append(poly)  # Добавим ранее созданный с дыркой
mpoly[1].holes[0][1] = (99, 99)  # Изменение второй точки дырки
mpoly[1].points[0] = (0, 0)  # Заменим первую (она же последняя) точку полигона
poly2 = Polygon([(11, 12), (13, 14), (15, 16), (17, 18)])
mpoly[0] = poly2  # Полностью заменим первый полигон
```

Классовые методы:

distance_by_points(start, end[, cs])	Производит расчет расстояния между двумя точками и азимут от первой до второй точки.
from_geojson(json[, cs])	Возвращает геометрию из ее 'GeoJSON' представления.
from_mif(mif)	Возвращает геометрию из ее 'MIF' представления.
from_wkb(wkb[, coordsystem])	Создает геометрический объект из строки формата WKB.
from_wkt(wkt[, coordsystem])	Создает геометрический объект из строки формата WKT.
point_by_azimuth(point, azimuth, distance[, cs])	Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

Свойства:

bounds	Возвращает минимальный ограничивающий прямоугольник.
coordsystem	Система Координат (СК) геометрии.
is_valid	Проверяет геометрию на валидность.
is_valid_reason	Если геометрия неправильная, возвращает краткую аннотацию причины.
name	Возвращает наименование геометрического объекта.
type	Возвращает тип геометрического элемента.

Методы:

affine_transform(trans)	Трансформирует объект исходя из заданных параметров трансформации.
almost_equals(other, tolerance)	Производит примерное сравнения с другой геометрией в пределах заданной точности.
append(*value)	Добавление геометрии в коллекцию.
boundary()	Возвращает границы геометрии в виде полилинии.
buffer(distance[, resolution, capStyle, ...])	Производит построение буфера вокруг объекта.
centroid()	Возвращает центроид геометрии.
clone()	Создает копию объекта.
contains(other)	Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.
convex_hull()	Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.

continues on next page

Таблица 28 – продолжение с предыдущей страницы

covers(other)	Возвращает True, если геометрия охватывает геометрию other.
crosses(other)	Возвращает True, если при пересечении геометрий объекты частично пересекаются.
difference(other)	Возвращает область первой геометрии, которая не пересечена второй геометрией.
disjoint(other)	Возвращает True, если геометрии не пересекаются и не соприкасаются.
envelope()	Возвращает полигон, описывающий заданную геометрию.
equals(other)	Производит сравнение с другой геометрией.
get_area([area_unit])	Рассчитывает площадь, если объект площадной.
get_distance(other[, u])	Производит расчет расстояния до объекта other.
get_length([u])	Рассчитывает длину геометрии.
get_perimeter([u])	Рассчитывает периметр геометрии.
intersection(other)	Возвращает область пересечения с другой геометрией.
intersects(other)	Возвращает True, если геометрии пересекаются.
overlaps(other)	Возвращает True, если пересечение геометрий отличается от обеих геометрий.
relate(other)	Проверяет отношения между объектами.
remove(idx)	Удаление геометрии из коллекции.
reproject(cs)	Перепроецирует геометрию в другую систему координат.
rotate(point, angle)	Поворот геометрии относительно заданной точки.
scale(kx, ky)	Масштабирует объект по заданным коэффициентам масштабирования.
shift(dx, dy)	Смещает объект на заданную величину.
split_by_polyline(splitter)	Разрезает объект линейным объектом.
symmetric_difference(other)	Возвращает логический XOR областей геометрий (объединение разниц).
to_geojson()	Преобразует геометрию в формат 'GeoJSON'
to_linestring()	Пробует геометрию преобразовать в линейный объект.
to_mif()	Преобразует геометрию в формат MIF.
to_polygon()	Пробует геометрию преобразовать в площадной объект.
to_wkb()	Возвращает WKB строку для геометрии.
to_wkt()	Возвращает WKT строку для геометрии.

continues on next page

Таблица 28 – продолжение с предыдущей страницы

`touches`(other)	Возвращает True, если геометрии соприкасаются.
`try_to_simplified`()	Создает копию коллекции при этом пробует упростить ее тип.
`union`(other)	Возвращает результат объединения двух геометрий.
`within`(other)	Возвращает True, если геометрия находится полностью внутри геометрии other.

affine_transform(trans: QTransform) → Geometry

Трансформирует объект исходя из заданных параметров трансформации.

См.также:

Для простых операций типа сдвиг, масштабирование и поворот, рекомендуется использовать `shift()`, `scale()`, `rotate()` соответственно.

Параметры
trans – Матрица трансформации.

almost_equals(other: Geometry, tolerance: float) → bool

Производит примерное сравнения с другой геометрией в пределах заданной точности.

Параметры

- **other** – Сравниваемый объект.
- **tolerance** – Точность сравнения.

Результат
Возвращает True, если геометрии равны в пределах заданного отклонения.

append(*value: Union[Geometry, Pnt, Tuple[float, float]])

Добавление геометрии в коллекцию. Задание параметров аналогично указанию их при создании объектов конкретного типа. Если опустить указание класса, то для одной точки или пары значений float будет создан точечный объект `Point`, если точек больше, - объект класса `LineString`.

Список 102: Пример.

```
# Создадим коллекцию и добавим в нее несколько объектов различного типа.
coll = GeometryCollection()
coll.append((1, 2))  # Точка
coll.append(1, 2)  # Точка
coll.append(
    [(3, 4), (5, 5), (10, 0)]
)  # Полилиния в виде :class:`list`. Можно это сделать через конструктор␣
↪:class:`LinearString`.
```

boundary() → Geometry

Возвращает границы геометрии в виде полилинии.

property bounds: Rect

Возвращает минимальный ограничивающий прямоугольник.

buffer(distance: float, resolution: int = 16, capStyle: LineCapStyle = LineCapStyle.Round, join_style: LineJoinStyle = LineJoinStyle.Round, mitreLimit: float = 5.0) → Geometry

Производит построение буфера вокруг объекта.

Параметры

- **distance** – Ширина буфера.
- **resolution** – Количество сегментов на квадрант.
- **capStyle** – Стиль окончания.
- **join_style** – Стиль соединения.
- **mitreLimit** – Предел среза.

centroid() → Geometry

Возвращает центроид геометрии.

clone() → Geometry

Создает копию объекта.

contains(other: Geometry) → bool

Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.

convex_hull() → Geometry

Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.

property coordsystem: CoordSystem

Система Координат (СК) геометрии.

covers(other: Geometry) → bool

Возвращает True, если геометрия охватывает геометрию other.

crosses(other: Geometry) → bool

Возвращает True, если при пересечении геометрий объекты частично пересекаются.

difference(other: Geometry) → Geometry

Возвращает область первой геометрии, которая не пересечена второй геометрией.

disjoint(other: Geometry) → bool

Возвращает True, если геометрии не пересекаются и не соприкасаются.

static distance_by_points(start: Union[Pnt, Tuple[float, float]], end: Union[Pnt, Tuple[float, float]], cs: Optional[CoordSystem] = None) → Tuple

Производит расчет расстояния между двумя точками и азимут от первой до второй точки.

См.также:

Обратная функция `point_by_azimuth()`

Параметры

- **start** – Начальная точка. Если задана СК, то координаты в ней.
- **end** – Конечная точка. Если задана СК, то координаты в ней.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат
Возвращается пара значений (расстояние в метрах, азимут в градусах)

envelope() → Geometry

Возвращает полигон, описывающий заданную геометрию.

equals(other: Geometry) → bool

Производит сравнение с другой геометрией.

Параметры
other – Сравниваемая геометрия.

Результат
Возвращает True, если геометрии равны.

static from_geojson(json: str, cs: Optional[CoordSystem] = None) → Geometry

Возвращает геометрию из ее „GeoJSON“ представления.

Параметры

- **json** – json представление в виде строки.
- **cs** – Система координат.

static from_mif(mif: str) → Geometry

Возвращает геометрию из ее „MIF“ представления.

Параметры
mif – mif представление в виде строки.

static from_wkb(wkb: bytes, coordsystem: Optional[CoordSystem] = None) → Geometry

Создает геометрический объект из строки формата WKB.

Параметры

- **wkb** – Строка WKB
- **coordsystem** – Система координат, которая будет установлена для геометрии.

Список 103: Пример.

```
wkb = b'\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00$@\x00\x00\x00\x00\x00\x00
↪$@'
pnt = Geometry.from_wkb(wkb)
```

static from_wkt(wkt: str, coordsystem: Optional[CoordSystem] = None) → Geometry

Создает геометрический объект из строки формата WKT.

Параметры

- **wkt** – Строка WKT. Допустимо задание строки в формате с указанием SRID (EWKT). В данном случае система координат для создаваемой геометрии будет установлено исходя их этого значения.
- **coordsystem** – Система координат, которая будет установлена для геометрии. Если строка задана в виде EWKT и указано значение SRID, игнорируется.

Список 104: Пример.

```
polygon = Geometry.from_wkt('POLYGON ((10 10, 100 100, 100 10, 10 10))')
point = Geometry.from_wkt('SRID=4326;POINT (10 10)')
crs = CoordSystem.from_epsg(4326)
pline = Geometry.from_wkt('LINESTRING (30 10, 10 30, 40 40)', crs)
```

get_area(area_unit: Optional[AreaUnit] = None) → float

Рассчитывает площадь, если объект площадной. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в квадратных метрах.

Список 105: Пример.

```
# В проекции Меркатора:
csMercatorKm = CoordSystem.from_prj("10, 104, 1, 0")
poly = Polygon([(0, 0), (0, 2), (2, 2), (2, 0)], cs=csMercatorKm)
print('Area Mercator km:', poly.get_area())
print('Area Mercator m:', poly.get_area(Unit.sq_m))
print('Perimeter Mercator km:', poly.get_perimeter())
print('Perimeter Mercator m:', poly.get_perimeter(Unit.m))
# В Широте/Долготе
poly.coordsystem = CoordSystem.from_prj("1, 104")
print('Area LL m:', poly.get_area())
print('Area LL km:', poly.get_area(Unit.sq_km))
print('Perimeter LL m:', poly.get_perimeter())
print('Perimeter LL km:', poly.get_perimeter(Unit.km))
'''
>>> Area Mercator km: 4.017946986632519
>>> Area Mercator m: 4017946.9866325194
>>> Perimeter Mercator km: 8.017972048421552
>>> Perimeter Mercator m: 8017.972048421552
>>> Area LL m: 49452172514.0342
>>> Area LL km: 49452.1725140342
>>> Perimeter LL m: 889423.5067063896
>>> Perimeter LL km: 889.4235067063896
'''
```

Параметры

area_unit – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и она не задана, то производится расчет на плоскости.

get_distance(other: Geometry, u: Optional[LinearUnit] = None) → float

Производит расчет расстояния до объекта other. Результат возвращает в СК текущего объекта.

Параметры

- **other** – Анализируемый объект.
- **u** – Единицы измерения, в которых требуется получить результат.

Список 106: Пример.

```
# Создадим геометрию без СК
first = Point(0, 0)
second = Point(10, 0)
print('В плане:', first.get_distance(second))
# Установим разные СК для точек
first.coordsystem = CoordSystem.from_prj("10, 104, 7, 0")
second.coordsystem = CoordSystem.from_prj("1, 104")
print('На сфере м:', first.get_distance(second))
print('На сфере км:', first.get_distance(second, Unit.km))
'''
>>> В плане: 10.0
>>> На сфере м: 1111948.7428468117
>>> На сфере км: 1111.9487428468117
'''
```

get_length(u: Optional[LinearUnit] = None) → float

Рассчитывает длину геометрии. Метод применим только для линейных объектов. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Список 107: Пример.

```
# В проекции Меркатора:
csMercatorKm = CoordSystem.from_prj("10, 104, 1, 0")
ls = Line((0, 0), (10, 0), csMercatorKm)
print('Length Mercator km:', ls.get_length())
print('Lenght Mercator m:', ls.get_length(Unit.m))
# В Широте/Долготе
csLL = CoordSystem.from_prj("1, 104")
ls = Line((0, 0), (10, 0), csLL)
print('Length LL m:', ls.get_length())
print('Length LL km:', ls.get_length(Unit.km))
'''
>>> Length Mercator km: 9.988805508567783
>>> Lenght Mercator m: 9988.805508567782
>>> Length LL m: 1111948.7428468117
>>> Length LL km: 1111.9487428468117
'''
```

Параметры

u – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и она не задана, то производится расчет на плоскости.

get_perimeter(u: Optional[LinearUnit] = None) → float

Рассчитывает периметр геометрии. Метод применим только для площадных объектов. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Пример см. `get_area()`

Параметры
u – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и она не задана, то производится расчет на плоскости.

intersection(other: Geometry) → Geometry

Возвращает область пересечения с другой геометрией.

intersects(other: Geometry) → bool

Возвращает True, если геометрии пересекаются.

property is_valid: bool

Проверяет геометрию на валидность.

property is_valid_reason: str

Если геометрия неправильная, возвращает краткую аннотацию причины.

property name: str

Возвращает наименование геометрического объекта.

overlaps(other: Geometry) → bool

Возвращает True, если пересечение геометрий отличается от обеих геометрий.

static point_by_azimuth(point: Union[Pnt, Tuple[float, float]], azimuth: float, distance: float, cs: Optional[CoordSystem] = None) → Pnt

Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

См.также:

Обратная функция `distance_by_points()`

Параметры

- **point** – Точка, относительно которой производится расчет. Если задана СК, то в ней.
- **azimuth** – Азимут в градусах, указывающий направление.
- **distance** – Расстояние по азимуту. Задается в метрах.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат
Возвращает результирующую точку.

relate(other: Geometry) → str

Проверяет отношения между объектами. Подробнее см. DE-9IM

remove(idx: int)

Удаление геометрии из коллекции.

Параметры
idx – Индекс геометрии в коллекции.

reproject(cs: CoordSystem) → Geometry

Перепроецирует геометрию в другую систему координат.

Параметры
cs – СК, в которой требуется получить объект.

rotate(point: Union[Pnt, Tuple[float, float]], angle: float) → Geometry

Поворот геометрии относительно заданной точки. Поворот производится против часовой стрелки.

Параметры

- **point** – Точка, вокруг которой необходимо произвести поворот.
- **angle** – Угол поворота в градусах.

scale(kx: float, ky: float) → Geometry

Масштабирует объект по заданным коэффициентам масштабирования. После выполнения операции центр результирующего объекта остается прежним.

Параметры

- **kx** – Масштабирование по координате X.
- **ky** – Масштабирование по координате Y.

shift(dx: float, dy: float) → Geometry

Смещает объект на заданную величину. Значения задаются в текущей проекции.

Параметры

- **dx** – Сдвиг по координате X.
- **dy** – Сдвиг по координате Y.

split_by_polyline(splitter: Geometry) → Optional[Geometry]

Разрезает объект линейным объектом.

Параметры
splitter – Геометрический объект, которым будет производиться разрезание объекта.

Результат
Возвращает полученный результат

```
poly = Polygon((0,10), (10,10), (10, 0))
line = Line((-5,5), (20,5))
r = poly.split_by_polyline(line)
print(r.wkt)
'''
>>> GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5, 5 5, 0 10)), POLYGON␣
↪((10 5, 10 0, 5 5, 10 5))) GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5,␣
↪5 5, 0 10)), POLYGON ((10 5, 10 0, 5 5, 10 5)))
'''
```

symmetric_difference(other: Geometry) → Geometry

Возвращает логический XOR областей геометрий (объединение разниц).

Параметры
other – Геометрия для анализа.

to_geojson() → str

Преобразует геометрию в формат „GeoJSON“

to_linestring() → Optional[Geometry]

Пробует геометрию преобразовать в линейный объект. В случае неудачи возвращает None.

to_mif() → str

Преобразует геометрию в формат MIF.

to_polygon() → Optional[Geometry]

Пробует геометрию преобразовать в площадной объект. В случае неудачи возвращает None.

to_wkb() → bytes

Возвращает WKB строку для геометрии.

to_wkt() → str

Возвращает WKT строку для геометрии.

touches(other: Geometry) → bool

Возвращает True, если геометрии соприкасаются.

try_to_simplified() → Union[MultiPoint, MultiLineString, MultiPolygon, GeometryCollection]

Создает копию коллекции при этом пробует упростить ее тип. К примеру, если в коллекции только полигоны, то вернет объект типа `MultiPolygon`. Если исходная коллекция разнородна, возвращается копия исходной.

property type: Type

Возвращает тип геометрического элемента.

Таблица 29: Возможные значения

Значение	Наименование
Unknown	Не определен
Point	Точка
Line	Линия
LineString	Полилиния
Polygon	Полигон
MultiPoint	Коллекция точек
MultiLineString	Коллекция полилиний
MultiPolygon	Коллекция полигонов
GeometryCollection	Смешанная коллекция
Arc	Дуга
Ellipse	Эллипс
Rectangle	Прямоугольник
RoundedRectangle	Скругленный прямоугольник
Text	Текст

Список 108: Пример.

```
point = Point(10, 10)
if point.type == GeometryType.Point:
    print('Это точка')
```

union(other: Geometry) → Geometry

Возвращает результат объединения двух геометрий.

within(other: Geometry) → bool

Возвращает True, если геометрия находится полностью внутри геометрии other.

22.12.10 Rectangle - Прямоугольник

class axipy.**Rectangle**

Базовые классы: Geometry

Геометрический объект типа прямоугольник.

Параметры

- **par** – Прямоугольник класса Rect или перечень координат через запятую (xmin, ymin, xmax, ymax) или списком list.
- **cs** – Система Координат, в которой создается геометрия.

Список 109: Пример.

```
r1 = Rectangle(0, 0, 40, 20)  # Создадим объект через перечень координат.
r2 = Rectangle(Rect(0, 0, 40, 20))  # Создадим объект передав объект␣
↪:class:`Rect`.
r3 = Rectangle([0, 0, 40, 20])  # Создадим объект передав списком :class:`list`.
```

Конструктор класса:

__init__(*par[, cs])	Создает экземпляр класса.

Классовые методы:

distance_by_points(start, end[, cs])	Производит расчет расстояния между двумя точками и азимут от первой до второй точки.
from_geojson(json[, cs])	Возвращает геометрию из ее 'GeoJSON' представления.
from_mif(mif)	Возвращает геометрию из ее 'MIF' представления.
from_wkb(wkb[, coordsystem])	Создает геометрический объект из строки формата WKB.
from_wkt(wkt[, coordsystem])	Создает геометрический объект из строки формата WKT.
point_by_azimuth(point, azimuth, distance[, cs])	Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

Свойства:

bounds	Возвращает минимальный ограничивающий прямоугольник.
coordsystem	Система Координат (СК) геометрии.
is_valid	Проверяет геометрию на валидность.
is_valid_reason	Если геометрия неправильная, возвращает краткую аннотацию причины.
name	Возвращает наименование геометрического объекта.
type	Возвращает тип геометрического элемента.
xmax	Максимальное значение X.
xmin	Минимальное значение X.
ymax	Максимальное значение Y.
ymin	Минимальное значение Y.

Методы:

affine_transform(trans)	Трансформирует объект исходя из заданных параметров трансформации.
almost_equals(other, tolerance)	Производит примерное сравнения с другой геометрией в пределах заданной точности.
boundary()	Возвращает границы геометрии в виде полилинии.
buffer(distance[, resolution, capStyle, ...])	Производит построение буфера вокруг объекта.
centroid()	Возвращает центроид геометрии.
clone()	Создает копию объекта.
contains(other)	Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.
convex_hull()	Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.
covers(other)	Возвращает True, если геометрия охватывает геометрию other.
crosses(other)	Возвращает True, если при пересечении геометрий объекты частично пересекаются.
difference(other)	Возвращает область первой геометрии, которая не пересечена второй геометрией.
disjoint(other)	Возвращает True, если геометрии не пересекаются и не соприкасаются.
envelope()	Возвращает полигон, описывающий заданную геометрию.
equals(other)	Производит сравнение с другой геометрией.
get_area([area_unit])	Рассчитывает площадь, если объект площадной.

continues on next page

Таблица 30 – продолжение с предыдущей страницы

get_distance(other[, u])	Производит расчет расстояния до объекта other.
get_length([u])	Рассчитывает длину геометрии.
get_perimeter([u])	Рассчитывает периметр геометрии.
intersection(other)	Возвращает область пересечения с другой геометрией.
intersects(other)	Возвращает True, если геометрии пересекаются.
overlaps(other)	Возвращает True, если пересечение геометрий отличается от обеих геометрий.
relate(other)	Проверяет отношения между объектами.
reproject(cs)	Перепроецирует геометрию в другую систему координат.
rotate(point, angle)	Поворот геометрии относительно заданной точки.
scale(kx, ky)	Масштабирует объект по заданным коэффициентам масштабирования.
shift(dx, dy)	Смещает объект на заданную величину.
split_by_polyline(splitter)	Разрезает объект линейным объектом.
symmetric_difference(other)	Возвращает логический XOR областей геометрий (объединение разниц).
to_geojson()	Преобразует геометрию в формат 'GeoJSON'
to_linestring()	Пробует геометрию преобразовать в линейный объект.
to_mif()	Преобразует геометрию в формат MIF.
to_polygon()	Пробует геометрию преобразовать в площадной объект.
to_wkb()	Возвращает WKB строку для геометрии.
to_wkt()	Возвращает WKT строку для геометрии.
touches(other)	Возвращает True, если геометрии соприкасаются.
union(other)	Возвращает результат объединения двух геометрий.
within(other)	Возвращает True, если геометрия находится полностью внутри геометрии other.

__init__(*par: Union[Rect, float], cs: Optional[CoordSystem] = None)

Создает экземпляр класса.

affine_transform(trans: QTransform) → Geometry

Трансформирует объект исходя из заданных параметров трансформации.

См.также:

Для простых операций типа сдвиг, масштабирование и поворот, рекомендуется использовать shift(), scale(), rotate() соответственно.

Параметры

trans – Матрица трансформации.

almost_equals(other: Geometry, tolerance: float) → bool

Производит примерное сравнения с другой геометрией в пределах заданной точности.

Параметры

- **other** – Сравниваемый объект.
- **tolerance** – Точность сравнения.

Результат

Возвращает True, если геометрии равны в пределах заданного отклонения.

boundary() → Geometry

Возвращает границы геометрии в виде полилинии.

property bounds: Rect

Возвращает минимальный ограничивающий прямоугольник.

buffer(distance: float, resolution: int = 16, capStyle: LineCapStyle = LineCapStyle.Round, join_style: LineJoinStyle = LineJoinStyle.Round, mitreLimit: float = 5.0) → Geometry

Производит построение буфера вокруг объекта.

Параметры

- **distance** – Ширина буфера.
- **resolution** – Количество сегментов на квадрант.
- **capStyle** – Стиль окончания.
- **join_style** – Стиль соединения.
- **mitreLimit** – Предел среза.

centroid() → Geometry

Возвращает центроид геометрии.

clone() → Geometry

Создает копию объекта.

contains(other: Geometry) → bool

Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.

convex_hull() → Geometry

Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.

property coordsystem: CoordSystem

Система Координат (СК) геометрии.

covers(other: Geometry) → bool

Возвращает True, если геометрия охватывает геометрию other.

crosses(other: Geometry) → bool

Возвращает True, если при пересечении геометрий объекты частично пересекаются.

difference(other: Geometry) → Geometry

Возвращает область первой геометрии, которая не пересечена второй геометрией.

disjoint(other: Geometry) → bool

Возвращает True, если геометрии не пересекаются и не соприкасаются.

static distance_by_points(start: Union[Pnt, Tuple[float, float]], end: Union[Pnt, Tuple[float, float]], cs: Optional[CoordSystem] = None) → Tuple

Производит расчет расстояния между двумя точками и азимут от первой до второй точки.

См.также:

Обратная функция `point_by_azimuth()`

Параметры

- **start** – Начальная точка. Если задана СК, то координаты в ней.
- **end** – Конечная точка. Если задана СК, то координаты в ней.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращается пара значений (расстояние в метрах, азимут в градусах)

envelope() → Geometry

Возвращает полигон, описывающий заданную геометрию.

equals(other: Geometry) → bool

Производит сравнение с другой геометрией.

Параметры

other – Сравниваемая геометрия.

Результат

Возвращает True, если геометрии равны.

static from_geojson(json: str, cs: Optional[CoordSystem] = None) → Geometry

Возвращает геометрию из ее „GeoJSON“ представления.

Параметры

- **json** – json представление в виде строки.
- **cs** – Система координат.

static from_mif(mif: str) → Geometry

Возвращает геометрию из ее „MIF“ представления.

Параметры

mif – mif представление в виде строки.

static from_wkb(wkb: bytes, coordsystem: Optional[CoordSystem] = None) → Geometry

Создает геометрический объект из строки формата WKB.

Параметры

- **wkb** – Строка WKB
- **coordsystem** – Система координат, которая будет установлена для геометрии.

Список 110: Пример.

```
wkb = b'\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00$@\x00\x00\x00\x00\x00\x00
↪$@'
pnt = Geometry.from_wkb(wkb)
```

static from_wkt(wkt: str, coordsystem: Optional[CoordSystem] = None) → Geometry

Создает геометрический объект из строки формата WKT.

Параметры

- **wkt** – Строка WKT. Допустимо задание строки в формате с указанием SRID (EWKT). В данном случае система координат для создаваемой геометрии будет установлено исходя их этого значения.
- **coordsystem** – Система координат, которая будет установлена для геометрии. Если строка задана в виде EWKT и указано значение SRID, игнорируется.

Список 111: Пример.

```
polygon = Geometry.from_wkt('POLYGON ((10 10, 100 100, 100 10, 10 10))')
point = Geometry.from_wkt('SRID=4326;POINT (10 10)')
crs = CoordSystem.from_epsg(4326)
pline = Geometry.from_wkt('LINESTRING (30 10, 10 30, 40 40)', crs)
```

get_area(area_unit: Optional[AreaUnit] = None) → float

Рассчитывает площадь, если объект площадной. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в квадратных метрах.

Список 112: Пример.

```
# В проекции Меркатора:
csMercatorKm = CoordSystem.from_prj("10, 104, 1, 0")
poly = Polygon([(0, 0), (0, 2), (2, 2), (2, 0)], cs=csMercatorKm)
print('Area Mercator km:', poly.get_area())
print('Area Mercator m:', poly.get_area(Unit.sq_m))
print('Perimeter Mercator km:', poly.get_perimeter())
print('Perimeter Mercator m:', poly.get_perimeter(Unit.m))
# В Широте/Долготе
poly.coordsystem = CoordSystem.from_prj("1, 104")
print('Area LL m:', poly.get_area())
print('Area LL km:', poly.get_area(Unit.sq_km))
print('Perimeter LL m:', poly.get_perimeter())
print('Perimeter LL km:', poly.get_perimeter(Unit.km))
'''
>>> Area Mercator km: 4.017946986632519
>>> Area Mercator m: 4017946.9866325194
>>> Perimeter Mercator km: 8.017972048421552
```

(continues on next page)

(продолжение с предыдущей страницы)

```
>>> Perimeter Mercator m: 8017.972048421552
>>> Area LL m: 49452172514.0342
>>> Area LL km: 49452.1725140342
>>> Perimeter LL m: 889423.5067063896
>>> Perimeter LL km: 889.4235067063896
'''
```

Параметры

area_unit – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и она не задана, то производится расчет на плоскости.

get_distance(other: Geometry, u: Optional[LinearUnit] = None) → float

Производит расчет расстояния до объекта other. Результат возвращает в СК текущего объекта.

Параметры

- **other** – Анализируемый объект.
- **u** – Единицы измерения, в которых требуется получить результат.

Список 113: Пример.

```
# Создадим геометрию без СК
first = Point(0, 0)
second = Point(10, 0)
print('В плане:', first.get_distance(second))
# Установим разные СК для точек
first.coordsystem = CoordSystem.from_prj("10, 104, 7, 0")
second.coordsystem = CoordSystem.from_prj("1, 104")
print('На сфере м:', first.get_distance(second))
print('На сфере км:', first.get_distance(second, Unit.km))
'''
>>> В плане: 10.0
>>> На сфере м: 1111948.7428468117
>>> На сфере км: 1111.9487428468117
'''
```

get_length(u: Optional[LinearUnit] = None) → float

Рассчитывает длину геометрии. Метод применим только для линейных объектов. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Список 114: Пример.

```
# В проекции Меркатора:
csMercatorKm = CoordSystem.from_prj("10, 104, 1, 0")
ls = Line((0, 0), (10, 0), csMercatorKm)
print('Length Mercator km:', ls.get_length())
print('Lenght Mercator m:', ls.get_length(Unit.m))
# В Широте/Долготе
csLL = CoordSystem.from_prj("1, 104")
ls = Line((0, 0), (10, 0), csLL)
print('Length LL m:', ls.get_length())
print('Length LL km:', ls.get_length(Unit.km))
```

(continues on next page)

(продолжение с предыдущей страницы)

```
'''
>>> Length Mercator km: 9.988805508567783
>>> Lenght Mercator m: 9988.805508567782
>>> Length LL m: 1111948.7428468117
>>> Length LL km: 1111.9487428468117
'''
```

Параметры

u – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и она не задана, то производится расчет на плоскости.

get_perimeter(u: Optional[LinearUnit] = None) → float

Рассчитывает периметр геометрии. Метод применим только для площадных объектов. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Пример см. `get_area()`

Параметры

u – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и она не задана, то производится расчет на плоскости.

intersection(other: Geometry) → Geometry

Возвращает область пересечения с другой геометрией.

intersects(other: Geometry) → bool

Возвращает True, если геометрии пересекаются.

property is_valid: bool

Проверяет геометрию на валидность.

property is_valid_reason: str

Если геометрия неправильная, возвращает краткую аннотацию причины.

property name: str

Возвращает наименование геометрического объекта.

overlaps(other: Geometry) → bool

Возвращает True, если пересечение геометрий отличается от обеих геометрий.

static point_by_azimuth(point: Union[Pnt, Tuple[float, float]], azimuth: float, distance: float, cs: Optional[CoordSystem] = None) → Pnt

Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

См.также:

Обратная функция `distance_by_points()`

Параметры

- **point** – Точка, относительно которой производится расчет. Если задана СК, то в ней.
- **azimuth** – Азимут в градусах, указывающий направление.

- **distance** – Расстояние по азимуту. Задается в метрах.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат
Возвращает результирующую точку.

relate(other: Geometry) → str

Проверяет отношения между объектами. Подробнее см. DE-9IM

reproject(cs: CoordSystem) → Geometry

Перепроецирует геометрию в другую систему координат.

Параметры
cs – СК, в которой требуется получить объект.

rotate(point: Union[Pnt, Tuple[float, float]], angle: float) → Geometry

Поворот геометрии относительно заданной точки. Поворот производится против часовой стрелки.

Параметры

- **point** – Точка, вокруг которой необходимо произвести поворот.
- **angle** – Угол поворота в градусах.

scale(kx: float, ky: float) → Geometry

Масштабирует объект по заданным коэффициентам масштабирования. После выполнения операции центр результирующего объекта остается прежним.

Параметры

- **kx** – Масштабирование по координате X.
- **ky** – Масштабирование по координате Y.

shift(dx: float, dy: float) → Geometry

Смещает объект на заданную величину. Значения задаются в текущей проекции.

Параметры

- **dx** – Сдвиг по координате X.
- **dy** – Сдвиг по координате Y.

split_by_polyline(splitter: Geometry) → Optional[Geometry]

Разрезает объект линейным объектом.

Параметры
splitter – Геометрический объект, которым будет производиться разрезание объекта.

Результат
Возвращает полученный результат

```
poly = Polygon((0,10), (10,10), (10, 0))
line = Line((-5,5), (20,5))
r = poly.split_by_polyline(line)
print(r.wkt)
'''
```

(continues on next page)

(продолжение с предыдущей страницы)

```
>>> GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5, 5 5, 0 10)), POLYGON␣
↪((10 5, 10 0, 5 5, 10 5))) GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5,␣
↪5 5, 0 10)), POLYGON ((10 5, 10 0, 5 5, 10 5)))
'''
```

symmetric_difference(other: Geometry) → Geometry

Возвращает логический XOR областей геометрий (объединение разниц).

Параметры
other – Геометрия для анализа.

to_geojson() → str

Преобразует геометрию в формат „GeoJSON“

to_linestring() → Optional[Geometry]

Пробует геометрию преобразовать в линейный объект. В случае неудачи возвращает None.

to_mif() → str

Преобразует геометрию в формат MIF.

to_polygon() → Optional[Geometry]

Пробует геометрию преобразовать в площадной объект. В случае неудачи возвращает None.

to_wkb() → bytes

Возвращает WKB строку для геометрии.

to_wkt() → str

Возвращает WKT строку для геометрии.

touches(other: Geometry) → bool

Возвращает True, если геометрии соприкасаются.

property type: Type

Возвращает тип геометрического элемента.

Таблица 31: Возможные значения

Значение	Наименование
Unknown	Не определен
Point	Точка
Line	Линия
LineString	Полилиния
Polygon	Полигон
MultiPoint	Коллекция точек
MultiLineString	Коллекция полилиний
MultiPolygon	Коллекция полигонов
GeometryCollection	Смешанная коллекция
Arc	Дуга
Ellipse	Эллипс
Rectangle	Прямоугольник
RoundedRectangle	Скругленный прямоугольник
Text	Текст

Список 115: Пример.

```
point = Point(10, 10)
if point.type == GeometryType.Point:
    print('Это точка')
```

union(other: Geometry) → Geometry

Возвращает результат объединения двух геометрий.

within(other: Geometry) → bool

Возвращает True, если геометрия находится полностью внутри геометрии other.

property xmax: float

Максимальное значение X.

property xmin: float

Минимальное значение X.

property ymax: float

Максимальное значение Y.

property ymin: float

Минимальное значение Y.

22.12.11 RoundRectangle - Скругленный прямоугольник

class axipy.**RoundRectangle**

Базовые классы: Rectangle

Геометрический объект типа скругленный прямоугольник.

Параметры

- **rect** – Прямоугольник класса Rect или как list.
- **xRad** – Скругление по X.
- **yRad** – Скругление по Y.
- **cs** – Система Координат, в которой создается геометрия.

Список 116: Пример.

```
r1 = RoundRectangle(Rect(0, 0, 22, 33), 0.1, 0.1)
r2 = RoundRectangle([0, 0, 22, 33], 0.1, 0.1)
```

Конструктор класса:

__init__(rect, xRad, yRad[, cs])	Создает экземпляр класса.

Классовые методы:

distance_by_points(start, end[, cs])	Производит расчет расстояния между двумя точками и азимут от первой до второй точки.
from_geojson(json[, cs])	Возвращает геометрию из ее 'GeoJSON' представления.
from_mif(mif)	Возвращает геометрию из ее 'MIF' представления.
from_wkb(wkb[, coordsystem])	Создает геометрический объект из строки формата WKB.
from_wkt(wkt[, coordsystem])	Создает геометрический объект из строки формата WKT.
point_by_azimuth(point, azimuth, distance[, cs])	Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

Свойства:

bounds	Возвращает минимальный ограничивающий прямоугольник.
coordsystem	Система Координат (СК) геометрии.
is_valid	Проверяет геометрию на валидность.
is_valid_reason	Если геометрия неправильная, возвращает краткую аннотацию причины.
name	Возвращает наименование геометрического объекта.
type	Возвращает тип геометрического элемента.
xRadius	Скругление углов по координате X.
xmax	Максимальное значение X.
xmin	Минимальное значение X.
yRadius	Скругление углов по координате Y.
ymax	Максимальное значение Y.
ymin	Минимальное значение Y.

Методы:

affine_transform(trans)	Трансформирует объект исходя из заданных параметров трансформации.
almost_equals(other, tolerance)	Производит примерное сравнения с другой геометрией в пределах заданной точности.
boundary()	Возвращает границы геометрии в виде полилинии.
buffer(distance[, resolution, capStyle, ...])	Производит построение буфера вокруг объекта.
centroid()	Возвращает центроид геометрии.
clone()	Создает копию объекта.

continues on next page

Таблица 32 – продолжение с предыдущей страницы

contains(other)	Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.
convex_hull()	Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.
covers(other)	Возвращает True, если геометрия охватывает геометрию other.
crosses(other)	Возвращает True, если при пересечении геометрий объекты частично пересекаются.
difference(other)	Возвращает область первой геометрии, которая не пересечена второй геометрией.
disjoint(other)	Возвращает True, если геометрии не пересекаются и не соприкасаются.
envelope()	Возвращает полигон, описывающий заданную геометрию.
equals(other)	Производит сравнение с другой геометрией.
get_area([area_unit])	Рассчитывает площадь, если объект площадной.
get_distance(other[, u])	Производит расчет расстояния до объекта other.
get_length([u])	Рассчитывает длину геометрии.
get_perimeter([u])	Рассчитывает периметр геометрии.
intersection(other)	Возвращает область пересечения с другой геометрией.
intersects(other)	Возвращает True, если геометрии пересекаются.
overlaps(other)	Возвращает True, если пересечение геометрий отличается от обеих геометрий.
relate(other)	Проверяет отношения между объектами.
reproject(cs)	Перепроецирует геометрию в другую систему координат.
rotate(point, angle)	Поворот геометрии относительно заданной точки.
scale(kx, ky)	Масштабирует объект по заданным коэффициентам масштабирования.
shift(dx, dy)	Смещает объект на заданную величину.
split_by_polyline(splitter)	Разрезает объект линейным объектом.
symmetric_difference(other)	Возвращает логический XOR областей геометрий (объединение разниц).
to_geojson()	Преобразует геометрию в формат 'GeoJSON'
to_linestring()	Пробует геометрию преобразовать в линейный объект.
to_mif()	Преобразует геометрию в формат MIF.

continues on next page

Таблица 32 – продолжение с предыдущей страницы

to_polygon()	Пробует геометрию преобразовать в площадной объект.
to_wkb()	Возвращает WKB строку для геометрии.
to_wkt()	Возвращает WKT строку для геометрии.
touches(other)	Возвращает True, если геометрии соприкасаются.
union(other)	Возвращает результат объединения двух геометрий.
within(other)	Возвращает True, если геометрия находится полностью внутри геометрии other.

__init__(rect: Union[Rect, list], xRad: float, yRad: float, cs: Optional[CoordSystem] = None)

Создает экземпляр класса.

affine_transform(trans: QTransform) → Geometry

Трансформирует объект исходя из заданных параметров трансформации.

См.также:

Для простых операций типа сдвиг, масштабирование и поворот, рекомендуется использовать shift(), scale(), rotate() соответственно.

Параметры
trans – Матрица трансформации.

almost_equals(other: Geometry, tolerance: float) → bool

Производит примерное сравнения с другой геометрией в пределах заданной точности.

Параметры

- **other** – Сравниваемый объект.
- **tolerance** – Точность сравнения.

Результат
Возвращает True, если геометрии равны в пределах заданного отклонения.

boundary() → Geometry

Возвращает границы геометрии в виде полилинии.

property bounds: Rect

Возвращает минимальный ограничивающий прямоугольник.

buffer(distance: float, resolution: int = 16, capStyle: LineCapStyle = LineCapStyle.Round, join_style: LineJoinStyle = LineJoinStyle.Round, mitreLimit: float = 5.0) → Geometry

Производит построение буфера вокруг объекта.

Параметры

- **distance** – Ширина буфера.
- **resolution** – Количество сегментов на квадрант.

- **capStyle** – Стиль окончания.
- **join_style** – Стиль соединения.
- **mitreLimit** – Предел среза.

centroid() → Geometry

Возвращает центроид геометрии.

clone() → Geometry

Создает копию объекта.

contains(other: Geometry) → bool

Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.

convex_hull() → Geometry

Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.

property coordsystem: CoordSystem

Система Координат (СК) геометрии.

covers(other: Geometry) → bool

Возвращает True, если геометрия охватывает геометрию other.

crosses(other: Geometry) → bool

Возвращает True, если при пересечении геометрий объекты частично пересекаются.

difference(other: Geometry) → Geometry

Возвращает область первой геометрии, которая не пересечена второй геометрией.

disjoint(other: Geometry) → bool

Возвращает True, если геометрии не пересекаются и не соприкасаются.

static distance_by_points(start: Union[Pnt, Tuple[float, float]], end: Union[Pnt, Tuple[float, float]], cs: Optional[CoordSystem] = None) → Tuple

Производит расчет расстояния между двумя точками и азимут от первой до второй точки.

См.также:

Обратная функция `point_by_azimuth()`

Параметры

- **start** – Начальная точка. Если задана СК, то координаты в ней.
- **end** – Конечная точка. Если задана СК, то координаты в ней.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращается пара значений (расстояние в метрах, азимут в градусах)

envelope() → Geometry

Возвращает полигон, описывающий заданную геометрию.

equals(other: Geometry) → bool

Производит сравнение с другой геометрией.

Параметры
other – Сравниваемая геометрия.

Результат
Возвращает True, если геометрии равны.

static from_geojson(json: str, cs: Optional[CoordSystem] = None) → Geometry

Возвращает геометрию из ее „GeoJSON“ представления.

Параметры

- **json** – json представление в виде строки.
- **cs** – Система координат.

static from_mif(mif: str) → Geometry

Возвращает геометрию из ее „MIF“ представления.

Параметры
mif – mif представление в виде строки.

static from_wkb(wkb: bytes, coordsystem: Optional[CoordSystem] = None) → Geometry

Создает геометрический объект из строки формата WKB.

Параметры

- **wkb** – Строка WKB
- **coordsystem** – Система координат, которая будет установлена для геометрии.

Список 117: Пример.

```
wkb = b'\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00$@\x00\x00\x00\x00\x00\x00
 ↪$@'
pnt = Geometry.from_wkb(wkb)
```

static from_wkt(wkt: str, coordsystem: Optional[CoordSystem] = None) → Geometry

Создает геометрический объект из строки формата WKT.

Параметры

- **wkt** – Строка WKT. Допустимо задание строки в формате с указанием SRID (EWKT). В данном случае система координат для создаваемой геометрии будет установлено исходя их этого значения.
- **coordsystem** – Система координат, которая будет установлена для геометрии. Если строка задана в виде EWKT и указано значение SRID, игнорируется.

Список 118: Пример.

```
polygon = Geometry.from_wkt('POLYGON ((10 10, 100 100, 100 10, 10 10))')
point = Geometry.from_wkt('SRID=4326;POINT (10 10)')
crs = CoordSystem.from_epsg(4326)
pline = Geometry.from_wkt('LINESTRING (30 10, 10 30, 40 40)', crs)
```

get_area(area_unit: Optional[AreaUnit] = None) → float

Рассчитывает площадь, если объект площадной. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в квадратных метрах.

Список 119: Пример.

```
# В проекции Меркатора:
csMercatorKm = CoordSystem.from_prj("10, 104, 1, 0")
poly = Polygon([(0, 0), (0, 2), (2, 2), (2, 0)], cs=csMercatorKm)
print('Area Mercator km:', poly.get_area())
print('Area Mercator m:', poly.get_area(Unit.sq_m))
print('Perimeter Mercator km:', poly.get_perimeter())
print('Perimeter Mercator m:', poly.get_perimeter(Unit.m))
# В Широте/Долготе
poly.coordsystem = CoordSystem.from_prj("1, 104")
print('Area LL m:', poly.get_area())
print('Area LL km:', poly.get_area(Unit.sq_km))
print('Perimeter LL m:', poly.get_perimeter())
print('Perimeter LL km:', poly.get_perimeter(Unit.km))
'''
>>> Area Mercator km: 4.017946986632519
>>> Area Mercator m: 4017946.9866325194
>>> Perimeter Mercator km: 8.017972048421552
>>> Perimeter Mercator m: 8017.972048421552
>>> Area LL m: 49452172514.0342
>>> Area LL km: 49452.1725140342
>>> Perimeter LL m: 889423.5067063896
>>> Perimeter LL km: 889.4235067063896
'''
```

Параметры

area_unit – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и она не задана, то производится расчет на плоскости.

get_distance(other: Geometry, u: Optional[LinearUnit] = None) → float

Производит расчет расстояния до объекта other. Результат возвращает в СК текущего объекта.

Параметры

- **other** – Анализируемый объект.
- **u** – Единицы измерения, в которых требуется получить результат.

Список 120: Пример.

```
# Создадим геометрию без СК
first = Point(0, 0)
second = Point(10, 0)
print('В плане:', first.get_distance(second))
# Установим разные СК для точек
first.coordsystem = CoordSystem.from_prj("10, 104, 7, 0")
second.coordsystem = CoordSystem.from_prj("1, 104")
print('На сфере м:', first.get_distance(second))
print('На сфере км:', first.get_distance(second, Unit.km))
'''
>>> В плане: 10.0
>>> На сфере м: 1111948.7428468117
>>> На сфере км: 1111.9487428468117
'''
```

get_length(u: Optional[LinearUnit] = None) → float

Рассчитывает длину геометрии. Метод применим только для линейных объектов. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Список 121: Пример.

```
# В проекции Меркатора:
csMercatorKm = CoordSystem.from_prj("10, 104, 1, 0")
ls = Line((0, 0), (10, 0), csMercatorKm)
print('Length Mercator km:', ls.get_length())
print('Lenght Mercator m:', ls.get_length(Unit.m))
# В Широте/Долготе
csLL = CoordSystem.from_prj("1, 104")
ls = Line((0, 0), (10, 0), csLL)
print('Length LL m:', ls.get_length())
print('Length LL km:', ls.get_length(Unit.km))
'''
>>> Length Mercator km: 9.988805508567783
>>> Lenght Mercator m: 9988.805508567782
>>> Length LL m: 1111948.7428468117
>>> Length LL km: 1111.9487428468117
'''
```

Параметры

u – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и она не задана, то производится расчет на плоскости.

get_perimeter(u: Optional[LinearUnit] = None) → float

Рассчитывает периметр геометрии. Метод применим только для площадных объектов. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Пример см. `get_area()`

Параметры

u – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и

она не задана, то производится расчет на плоскости.

intersection(other: Geometry) → Geometry

Возвращает область пересечения с другой геометрией.

intersects(other: Geometry) → bool

Возвращает True, если геометрии пересекаются.

property is_valid: bool

Проверяет геометрию на валидность.

property is_valid_reason: str

Если геометрия неправильная, возвращает краткую аннотацию причины.

property name: str

Возвращает наименование геометрического объекта.

overlaps(other: Geometry) → bool

Возвращает True, если пересечение геометрий отличается от обеих геометрий.

static point_by_azimuth(point: Union[Pnt, Tuple[float, float]], azimuth: float, distance: float, cs: Optional[CoordSystem] = None) → Pnt

Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

См.также:

Обратная функция `distance_by_points()`

Параметры

- **point** – Точка, относительно которой производится расчет. Если задана СК, то в ней.
- **azimuth** – Азимут в градусах, указывающий направление.
- **distance** – Расстояние по азимуту. Задается в метрах.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращает результирующую точку.

relate(other: Geometry) → str

Проверяет отношения между объектами. Подробнее см. DE-9IM

reproject(cs: CoordSystem) → Geometry

Перепроецирует геометрию в другую систему координат.

Параметры

cs – СК, в которой требуется получить объект.

rotate(point: Union[Pnt, Tuple[float, float]], angle: float) → Geometry

Поворот геометрии относительно заданной точки. Поворот производится против часовой стрелки.

Параметры

- **point** – Точка, вокруг которой необходимо произвести поворот.
- **angle** – Угол поворота в градусах.

scale(kx: float, ky: float) → Geometry

Масштабирует объект по заданным коэффициентам масштабирования. После выполнения операции центр результирующего объекта остается прежним.

Параметры

- **kx** – Масштабирование по координате X.
- **ky** – Масштабирование по координате Y.

shift(dx: float, dy: float) → Geometry

Смещает объект на заданную величину. Значения задаются в текущей проекции.

Параметры

- **dx** – Сдвиг по координате X.
- **dy** – Сдвиг по координате Y.

split_by_polyline(splitter: Geometry) → Optional[Geometry]

Разрезает объект линейным объектом.

Параметры
splitter – Геометрический объект, которым будет производиться разрезание объекта.

Результат
Возвращает полученный результат

```
poly = Polygon((0,10), (10,10), (10, 0))
line = Line((-5,5), (20,5))
r = poly.split_by_polyline(line)
print(r.wkt)
'''
>>> GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5, 5 5, 0 10)), POLYGON␣
↪((10 5, 10 0, 5 5, 10 5))) GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5,␣
↪5 5, 0 10)), POLYGON ((10 5, 10 0, 5 5, 10 5)))
'''
```

symmetric_difference(other: Geometry) → Geometry

Возвращает логический XOR областей геометрий (объединение разниц).

Параметры
other – Геометрия для анализа.

to_geojson() → str

Преобразует геометрию в формат „GeoJSON“

to_linestring() → Optional[Geometry]

Пробует геометрию преобразовать в линейный объект. В случае неудачи возвращает None.

to_mif() → str

Преобразует геометрию в формат MIF.

to_polygon() → Optional[Geometry]

Пробует геометрию преобразовать в площадной объект. В случае неудачи возвращает None.

to_wkb() → bytes
Возвращает WKB строку для геометрии.

to_wkt() → str
Возвращает WKT строку для геометрии.

touches(other: Geometry) → bool
Возвращает True, если геометрии соприкасаются.

property type: Type
Возвращает тип геометрического элемента.

Таблица 33: Возможные значения

Значение	Наименование
Unknown	Не определен
Point	Точка
Line	Линия
LineString	Полилиния
Polygon	Полигон
MultiPoint	Коллекция точек
MultiLineString	Коллекция полилиний
MultiPolygon	Коллекция полигонов
GeometryCollection	Смешанная коллекция
Arc	Дуга
Ellipse	Эллипс
Rectangle	Прямоугольник
RoundedRectangle	Скругленный прямоугольник
Text	Текст

Список 122: Пример.

```
point = Point(10, 10)
if point.type == GeometryType.Point:
    print('Это точка')
```

union(other: Geometry) → Geometry
Возвращает результат объединения двух геометрий.

within(other: Geometry) → bool
Возвращает True, если геометрия находится полностью внутри геометрии other.

property xRadius: float
Скругление углов по координате X.

property xmax: float
Максимальное значение X.

property xmin: float
Минимальное значение X.

property yRadius: float
Скругление углов по координате Y.

property ymax: float
Максимальное значение Y.

property ymin: float
Минимальное значение Y.

22.12.12 Ellipse - Эллипс

class axipy.**Ellipse**
Базовые классы: Geometry

Геометрический объект типа эллипс.

Параметры

- **rect** – Прямоугольник класса Rect или как list.
- **cs** – Система Координат, в которой создается геометрия.

Список 123: Пример.

```
e1 = Ellipse(Rect(0, 0, 22, 33))
e2 = Ellipse([0, 0, 22, 33])
e1.center = (10, 10)  # Переопределим центр
e1.majorSemiAxis = 10  # Задание большой полуоси
e1.minorSemiAxis = 5  # Задание малой полуоси
```

Конструктор класса:

__init__(rect[, cs])	Создает экземпляр класса.

Классовые методы:

distance_by_points(start, end[, cs])	Производит расчет расстояния между двумя точками и азимут от первой до второй точки.
from_geojson(json[, cs])	Возвращает геометрию из ее 'GeoJSON' представления.
from_mif(mif)	Возвращает геометрию из ее 'MIF' представления.
from_wkb(wkb[, coordsystem])	Создает геометрический объект из строки формата WKB.
from_wkt(wkt[, coordsystem])	Создает геометрический объект из строки формата WKT.
point_by_azimuth(point, azimuth, distance[, cs])	Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

Свойства:

bounds	Возвращает минимальный ограничивающий прямоугольник.
center	Центр эллипса.
coordsystem	Система Координат (СК) геометрии.
is_valid	Проверяет геометрию на валидность.
is_valid_reason	Если геометрия неправильная, возвращает краткую аннотацию причины.
majorSemiAxis	Радиус большой полуоси эллипса.
minorSemiAxis	Радиус малой полуоси эллипса.
name	Возвращает наименование геометрического объекта.
type	Возвращает тип геометрического элемента.

Методы:

affine_transform(trans)	Трансформирует объект исходя из заданных параметров трансформации.
almost_equals(other, tolerance)	Производит примерное сравнения с другой геометрией в пределах заданной точности.
boundary()	Возвращает границы геометрии в виде полилинии.
buffer(distance[, resolution, capStyle, ...])	Производит построение буфера вокруг объекта.
centroid()	Возвращает центроид геометрии.
clone()	Создает копию объекта.
contains(other)	Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.
convex_hull()	Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.
covers(other)	Возвращает True, если геометрия охватывает геометрию other.
crosses(other)	Возвращает True, если при пересечении геометрий объекты частично пересекаются.
difference(other)	Возвращает область первой геометрии, которая не пересечена второй геометрией.
disjoint(other)	Возвращает True, если геометрии не пересекаются и не соприкасаются.
envelope()	Возвращает полигон, описывающий заданную геометрию.
equals(other)	Производит сравнение с другой геометрией.
get_area([area_unit])	Рассчитывает площадь, если объект площадной.

continues on next page

Таблица 34 – продолжение с предыдущей страницы

get_distance(other[, u])	Производит расчет расстояния до объекта other.
get_length([u])	Рассчитывает длину геометрии.
get_perimeter([u])	Рассчитывает периметр геометрии.
intersection(other)	Возвращает область пересечения с другой геометрией.
intersects(other)	Возвращает True, если геометрии пересекаются.
overlaps(other)	Возвращает True, если пересечение геометрий отличается от обеих геометрий.
relate(other)	Проверяет отношения между объектами.
reproject(cs)	Перепроецирует геометрию в другую систему координат.
rotate(point, angle)	Поворот геометрии относительно заданной точки.
scale(kx, ky)	Масштабирует объект по заданным коэффициентам масштабирования.
shift(dx, dy)	Смещает объект на заданную величину.
split_by_polyline(splitter)	Разрезает объект линейным объектом.
symmetric_difference(other)	Возвращает логический XOR областей геометрий (объединение разниц).
to_geojson()	Преобразует геометрию в формат 'GeoJSON'
to_linestring()	Пробует геометрию преобразовать в линейный объект.
to_mif()	Преобразует геометрию в формат MIF.
to_polygon()	Пробует геометрию преобразовать в площадной объект.
to_wkb()	Возвращает WKB строку для геометрии.
to_wkt()	Возвращает WKT строку для геометрии.
touches(other)	Возвращает True, если геометрии соприкасаются.
union(other)	Возвращает результат объединения двух геометрий.
within(other)	Возвращает True, если геометрия находится полностью внутри геометрии other.

__init__(rect: Union[Rect, list], cs: Optional[CoordSystem] = None)

Создает экземпляр класса.

affine_transform(trans: QTransform) → Geometry

Трансформирует объект исходя из заданных параметров трансформации.

См.также:

Для простых операций типа сдвиг, масштабирование и поворот, рекомендуется использовать shift(), scale(), rotate() соответственно.

Параметры

trans – Матрица трансформации.

almost_equals(other: Geometry, tolerance: float) → bool

Производит примерное сравнения с другой геометрией в пределах заданной точности.

Параметры

- **other** – Сравниваемый объект.
- **tolerance** – Точность сравнения.

Результат

Возвращает True, если геометрии равны в пределах заданного отклонения.

boundary() → Geometry

Возвращает границы геометрии в виде полилинии.

property bounds: Rect

Возвращает минимальный ограничивающий прямоугольник.

buffer(distance: float, resolution: int = 16, capStyle: LineCapStyle = LineCapStyle.Round, join_style: LineJoinStyle = LineJoinStyle.Round, mitreLimit: float = 5.0) → Geometry

Производит построение буфера вокруг объекта.

Параметры

- **distance** – Ширина буфера.
- **resolution** – Количество сегментов на квадрант.
- **capStyle** – Стиль окончания.
- **join_style** – Стиль соединения.
- **mitreLimit** – Предел среза.

property center: Pnt

Центр эллипса.

centroid() → Geometry

Возвращает центроид геометрии.

clone() → Geometry

Создает копию объекта.

contains(other: Geometry) → bool

Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.

convex_hull() → Geometry

Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.

property coordsystem: CoordSystem

Система Координат (СК) геометрии.

covers(other: Geometry) → bool

Возвращает True, если геометрия охватывает геометрию other.

crosses(other: Geometry) → bool

Возвращает True, если при пересечении геометрий объекты частично пересекаются.

difference(other: Geometry) → Geometry

Возвращает область первой геометрии, которая не пересечена второй геометрией.

disjoint(other: Geometry) → bool

Возвращает True, если геометрии не пересекаются и не соприкасаются.

static distance_by_points(start: Union[Pnt, Tuple[float, float]], end: Union[Pnt, Tuple[float, float]], cs: Optional[CoordSystem] = None) → Tuple

Производит расчет расстояния между двумя точками и азимут от первой до второй точки.

См.также:

Обратная функция `point_by_azimuth()`

Параметры

- **start** – Начальная точка. Если задана СК, то координаты в ней.
- **end** – Конечная точка. Если задана СК, то координаты в ней.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращается пара значений (расстояние в метрах, азимут в градусах)

envelope() → Geometry

Возвращает полигон, описывающий заданную геометрию.

equals(other: Geometry) → bool

Производит сравнение с другой геометрией.

Параметры

other – Сравниваемая геометрия.

Результат

Возвращает True, если геометрии равны.

static from_geojson(json: str, cs: Optional[CoordSystem] = None) → Geometry

Возвращает геометрию из ее „GeoJSON“ представления.

Параметры

- **json** – json представление в виде строки.
- **cs** – Система координат.

static from_mif(mif: str) → Geometry

Возвращает геометрию из ее „MIF“ представления.

Параметры

mif – mif представление в виде строки.

static from_wkb(wkb: bytes, coordsystem: Optional[CoordSystem] = None) → Geometry

Создает геометрический объект из строки формата WKB.

Параметры

- **wkb** – Строка WKB
- **coordsystem** – Система координат, которая будет установлена для геометрии.

Список 124: Пример.

```
wkb = b'\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00$@\x00\x00\x00\x00\x00\x00
↪$@'
pnt = Geometry.from_wkb(wkb)
```

static from_wkt(wkt: str, coordsystem: Optional[CoordSystem] = None) → Geometry

Создает геометрический объект из строки формата WKT.

Параметры

- **wkt** – Строка WKT. Допустимо задание строки в формате с указанием SRID (EWKT). В данном случае система координат для создаваемой геометрии будет установлено исходя их этого значения.
- **coordsystem** – Система координат, которая будет установлена для геометрии. Если строка задана в виде EWKT и указано значение SRID, игнорируется.

Список 125: Пример.

```
polygon = Geometry.from_wkt('POLYGON ((10 10, 100 100, 100 10, 10 10))')
point = Geometry.from_wkt('SRID=4326;POINT (10 10)')
crs = CoordSystem.from_epsg(4326)
pline = Geometry.from_wkt('LINESTRING (30 10, 10 30, 40 40)', crs)
```

get_area(area_unit: Optional[AreaUnit] = None) → float

Рассчитывает площадь, если объект площадной. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в квадратных метрах.

Список 126: Пример.

```
# В проекции Меркатора:
csMercatorKm = CoordSystem.from_prj("10, 104, 1, 0")
poly = Polygon([(0, 0), (0, 2), (2, 2), (2, 0)], cs=csMercatorKm)
print('Area Mercator km:', poly.get_area())
print('Area Mercator m:', poly.get_area(Unit.sq_m))
print('Perimeter Mercator km:', poly.get_perimeter())
print('Perimeter Mercator m:', poly.get_perimeter(Unit.m))
# В Широте/Долготе
poly.coordsystem = CoordSystem.from_prj("1, 104")
print('Area LL m:', poly.get_area())
print('Area LL km:', poly.get_area(Unit.sq_km))
print('Perimeter LL m:', poly.get_perimeter())
print('Perimeter LL km:', poly.get_perimeter(Unit.km))
```

(continues on next page)

(продолжение с предыдущей страницы)

```
'''
>>> Area Mercator km: 4.017946986632519
>>> Area Mercator m: 4017946.9866325194
>>> Perimeter Mercator km: 8.017972048421552
>>> Perimeter Mercator m: 8017.972048421552
>>> Area LL m: 49452172514.0342
>>> Area LL km: 49452.1725140342
>>> Perimeter LL m: 889423.5067063896
>>> Perimeter LL km: 889.4235067063896
'''
```

Параметры

area_unit – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и она не задана, то производится расчет на плоскости.

get_distance(other: Geometry, u: Optional[LinearUnit] = None) → float

Производит расчет расстояния до объекта other. Результат возвращает в СК текущего объекта.

Параметры

- **other** – Анализируемый объект.
- **u** – Единицы измерения, в которых требуется получить результат.

Список 127: Пример.

```
# Создадим геометрию без СК
first = Point(0, 0)
second = Point(10, 0)
print('В плане:', first.get_distance(second))
# Установим разные СК для точек
first.coordsystem = CoordSystem.from_prj("10, 104, 7, 0")
second.coordsystem = CoordSystem.from_prj("1, 104")
print('На сфере м:', first.get_distance(second))
print('На сфере км:', first.get_distance(second, Unit.km))
'''
>>> В плане: 10.0
>>> На сфере м: 1111948.7428468117
>>> На сфере км: 1111.9487428468117
'''
```

get_length(u: Optional[LinearUnit] = None) → float

Рассчитывает длину геометрии. Метод применим только для линейных объектов. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Список 128: Пример.

```
# В проекции Меркатора:
csMercatorKm = CoordSystem.from_prj("10, 104, 1, 0")
ls = Line((0, 0), (10, 0), csMercatorKm)
print('Length Mercator km:', ls.get_length())
print('Lenght Mercator m:', ls.get_length(Unit.m))
# В Широте/Долготе
```

(continues on next page)

(продолжение с предыдущей страницы)

```
csLL = CoordSystem.from_prj("1, 104")
ls = Line((0, 0), (10, 0), csLL)
print('Length LL m:', ls.get_length())
print('Length LL km:', ls.get_length(Unit.km))
'''
>>> Length Mercator km: 9.988805508567783
>>> Lenght Mercator m: 9988.805508567782
>>> Length LL m: 1111948.7428468117
>>> Length LL km: 1111.9487428468117
'''
```

Параметры

u – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и она не задана, то производится расчет на плоскости.

get_perimeter(u: Optional[LinearUnit] = None) → float

Рассчитывает периметр геометрии. Метод применим только для площадных объектов. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Пример см. `get_area()`

Параметры

u – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и она не задана, то производится расчет на плоскости.

intersection(other: Geometry) → Geometry

Возвращает область пересечения с другой геометрией.

intersects(other: Geometry) → bool

Возвращает True, если геометрии пересекаются.

property is_valid: bool

Проверяет геометрию на валидность.

property is_valid_reason: str

Если геометрия неправильная, возвращает краткую аннотацию причины.

property majorSemiAxis: float

Радиус большой полуоси эллипса.

property minorSemiAxis: float

Радиус малой полуоси эллипса.

property name: str

Возвращает наименование геометрического объекта.

overlaps(other: Geometry) → bool

Возвращает True, если пересечение геометрий отличается от обеих геометрий.

static point_by_azimuth(point: Union[Pnt, Tuple[float, float]], azimuth: float, distance: float, cs: Optional[CoordSystem] = None) → Pnt

Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

См.также:

Обратная функция `distance_by_points()`

Параметры

- **point** – Точка, относительно которой производится расчет. Если задана СК, то в ней.
- **azimuth** – Азимут в градусах, указывающий направление.
- **distance** – Расстояние по азимуту. Задается в метрах.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат
Возвращает результирующую точку.

relate(other: Geometry) → str

Проверяет отношения между объектами. Подробнее см. DE-9IM

reproject(cs: CoordSystem) → Geometry

Перепроецирует геометрию в другую систему координат.

Параметры
cs – СК, в которой требуется получить объект.

rotate(point: Union[Pnt, Tuple[float, float]], angle: float) → Geometry

Поворот геометрии относительно заданной точки. Поворот производится против часовой стрелки.

Параметры

- **point** – Точка, вокруг которой необходимо произвести поворот.
- **angle** – Угол поворота в градусах.

scale(kx: float, ky: float) → Geometry

Масштабирует объект по заданным коэффициентам масштабирования. После выполнения операции центр результирующего объекта остается прежним.

Параметры

- **kx** – Масштабирование по координате X.
- **ky** – Масштабирование по координате Y.

shift(dx: float, dy: float) → Geometry

Смещает объект на заданную величину. Значения задаются в текущей проекции.

Параметры

- **dx** – Сдвиг по координате X.
- **dy** – Сдвиг по координате Y.

split_by_polyline(splitter: Geometry) → Optional[Geometry]

Разрезает объект линейным объектом.

Параметры
splitter – Геометрический объект, которым будет производиться разрезание объекта.

Результат
Возвращает полученный результат

```
poly = Polygon((0,10), (10,10), (10, 0))
line = Line((-5,5), (20,5))
r = poly.split_by_polyline(line)
print(r.wkt)
'''
>>> GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5, 5 5, 0 10)), POLYGON␣
↪((10 5, 10 0, 5 5, 10 5))) GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5,␣
↪5 5, 0 10)), POLYGON ((10 5, 10 0, 5 5, 10 5)))
'''
```

symmetric_difference(other: Geometry) → Geometry

Возвращает логический XOR областей геометрий (объединение разниц).

Параметры
other – Геометрия для анализа.

to_geojson() → str

Преобразует геометрию в формат „GeoJSON“

to_linestring() → Optional[Geometry]

Пробует геометрию преобразовать в линейный объект. В случае неудачи возвращает None.

to_mif() → str

Преобразует геометрию в формат MIF.

to_polygon() → Optional[Geometry]

Пробует геометрию преобразовать в площадной объект. В случае неудачи возвращает None.

to_wkb() → bytes

Возвращает WKB строку для геометрии.

to_wkt() → str

Возвращает WKT строку для геометрии.

touches(other: Geometry) → bool

Возвращает True, если геометрии соприкасаются.

property type: Type

Возвращает тип геометрического элемента.

Таблица 35: Возможные значения

Значение	Наименование
Unknown	Не определен
Point	Точка
Line	Линия
LineString	Полилиния
Polygon	Полигон
MultiPoint	Коллекция точек
MultiLineString	Коллекция полилиний
MultiPolygon	Коллекция полигонов
GeometryCollection	Смешанная коллекция
Arc	Дуга
Ellipse	Эллипс
Rectangle	Прямоугольник
RoundedRectangle	Скругленный прямоугольник
Text	Текст

Список 129: Пример.

```
point = Point(10, 10)
if point.type == GeometryType.Point:
    print('Это точка')
```

union(other: Geometry) → Geometry

Возвращает результат объединения двух геометрий.

within(other: Geometry) → bool

Возвращает True, если геометрия находится полностью внутри геометрии other.

22.12.13 Arc - Дуга

class axipy.**Arc**

Базовые классы: Geometry

Геометрический объект типа дуга.

Параметры

- **rect** – Прямоугольник класса Rect или как list.
- **startAngle** – Начальный угол дуги.
- **endAngle** – Конечный угол дуги.
- **cs** – Система Координат, в которой создается геометрия.

Список 130: Пример.

```
a1 = Arc(Rect(0, 0, 22, 33), 0, 90)
a2 = Arc([0, 0, 22, 33], 0, 90)
```

Конструктор класса:

__init__(rect, startAngle, endAngle[, cs])	Создает экземпляр класса.

Классовые методы:

distance_by_points(start, end[, cs])	Производит расчет расстояния между двумя точками и азимут от первой до второй точки.
from_geojson(json[, cs])	Возвращает геометрию из ее 'GeoJSON' представления.
from_mif(mif)	Возвращает геометрию из ее 'MIF' представления.
from_wkb(wkb[, coordsystem])	Создает геометрический объект из строки формата WKB.
from_wkt(wkt[, coordsystem])	Создает геометрический объект из строки формата WKT.
point_by_azimuth(point, azimuth, distance[, cs])	Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

Свойства:

bounds	Возвращает минимальный ограничивающий прямоугольник.
center	Центр дуги.
coordsystem	Система Координат (СК) геометрии.
endAngle	Конечный угол дуги.
is_valid	Проверяет геометрию на валидность.
is_valid_reason	Если геометрия неправильная, возвращает краткую аннотацию причины.
name	Возвращает наименование геометрического объекта.
startAngle	Начальный угол дуги.
type	Возвращает тип геометрического элемента.
xRadius	Радиус большой полуоси прямоугольника, в который вписана дуга.
yRadius	Радиус малой полуоси прямоугольника, в который вписана дуга.

Методы:

affine_transform(trans)	Трансформирует объект исходя из заданных параметров трансформации.
almost_equals(other, tolerance)	Производит примерное сравнения с другой геометрией в пределах заданной точности.
boundary()	Возвращает границы геометрии в виде полилинии.
buffer(distance[, resolution, capStyle, ...])	Производит построение буфера вокруг объекта.

continues on next page

Таблица 36 – продолжение с предыдущей страницы

centroid()	Возвращает центроид геометрии.
clone()	Создает копию объекта.
contains(other)	Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.
convex_hull()	Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.
covers(other)	Возвращает True, если геометрия охватывает геометрию other.
crosses(other)	Возвращает True, если при пересечении геометрий объекты частично пересекаются.
difference(other)	Возвращает область первой геометрии, которая не пересечена второй геометрией.
disjoint(other)	Возвращает True, если геометрии не пересекаются и не соприкасаются.
envelope()	Возвращает полигон, описывающий заданную геометрию.
equals(other)	Производит сравнение с другой геометрией.
get_area([area_unit])	Рассчитывает площадь, если объект площадной.
get_distance(other[, u])	Производит расчет расстояния до объекта other.
get_length([u])	Рассчитывает длину геометрии.
get_perimeter([u])	Рассчитывает периметр геометрии.
intersection(other)	Возвращает область пересечения с другой геометрией.
intersects(other)	Возвращает True, если геометрии пересекаются.
overlaps(other)	Возвращает True, если пересечение геометрий отличается от обеих геометрий.
relate(other)	Проверяет отношения между объектами.
reproject(cs)	Перепроецирует геометрию в другую систему координат.
rotate(point, angle)	Поворот геометрии относительно заданной точки.
scale(kx, ky)	Масштабирует объект по заданным коэффициентам масштабирования.
shift(dx, dy)	Смещает объект на заданную величину.
split_by_polyline(splitter)	Разрезает объект линейным объектом.
symmetric_difference(other)	Возвращает логический XOR областей геометрий (объединение разниц).
to_geojson()	Преобразует геометрию в формат 'GeoJSON'
to_linestring()	Пробует геометрию преобразовать в линейный объект.

continues on next page

Таблица 36 – продолжение с предыдущей страницы

to_mif()	Преобразует геометрию в формат MIF.
to_polygon()	Пробует геометрию преобразовать в площадной объект.
to_wkb()	Возвращает WKB строку для геометрии.
to_wkt()	Возвращает WKT строку для геометрии.
touches(other)	Возвращает True, если геометрии соприкасаются.
union(other)	Возвращает результат объединения двух геометрий.
within(other)	Возвращает True, если геометрия находится полностью внутри геометрии other.

__init__(rect: Union[Rect, list], startAngle: float, endAngle: float, cs: Optional[CoordSystem] = None)

Создает экземпляр класса.

affine_transform(trans: QTransform) → Geometry

Трансформирует объект исходя из заданных параметров трансформации.

См.также:

Для простых операций типа сдвиг, масштабирование и поворот, рекомендуется использовать shift(), scale(), rotate() соответственно.

Параметры
trans – Матрица трансформации.

almost_equals(other: Geometry, tolerance: float) → bool

Производит примерное сравнения с другой геометрией в пределах заданной точности.

Параметры

- **other** – Сравниваемый объект.
- **tolerance** – Точность сравнения.

Результат
Возвращает True, если геометрии равны в пределах заданного отклонения.

boundary() → Geometry

Возвращает границы геометрии в виде полилинии.

property bounds: Rect

Возвращает минимальный ограничивающий прямоугольник.

buffer(distance: float, resolution: int = 16, capStyle: LineCapStyle = LineCapStyle.Round, join_style: LineJoinStyle = LineJoinStyle.Round, mitreLimit: float = 5.0) → Geometry

Производит построение буфера вокруг объекта.

Параметры

- **distance** – Ширина буфера.

- **resolution** – Количество сегментов на квадрант.
- **capStyle** – Стиль окончания.
- **join_style** – Стиль соединения.
- **mitreLimit** – Предел среза.

property center: Pnt

Центр дуги.

centroid() → Geometry

Возвращает центроид геометрии.

clone() → Geometry

Создает копию объекта.

contains(other: Geometry) → bool

Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.

convex_hull() → Geometry

Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.

property coordsystem: CoordSystem

Система Координат (СК) геометрии.

covers(other: Geometry) → bool

Возвращает True, если геометрия охватывает геометрию other.

crosses(other: Geometry) → bool

Возвращает True, если при пересечении геометрий объекты частично пересекаются.

difference(other: Geometry) → Geometry

Возвращает область первой геометрии, которая не пересечена второй геометрией.

disjoint(other: Geometry) → bool

Возвращает True, если геометрии не пересекаются и не соприкасаются.

static distance_by_points(start: Union[Pnt, Tuple[float, float]], end: Union[Pnt, Tuple[float, float]], cs: Optional[CoordSystem] = None) → Tuple

Производит расчет расстояния между двумя точками и азимут от первой до второй точки.

См.также:

Обратная функция `point_by_azimuth()`

Параметры

- **start** – Начальная точка. Если задана СК, то координаты в ней.
- **end** – Конечная точка. Если задана СК, то координаты в ней.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат
Возвращается пара значений (расстояние в метрах, азимут в градусах)

property endAngle: float
Конечный угол дуги.

envelope() → Geometry
Возвращает полигон, описывающий заданную геометрию.

equals(other: Geometry) → bool
Производит сравнение с другой геометрией.

Параметры
other – Сравниваемая геометрия.

Результат
Возвращает True, если геометрии равны.

static from_geojson(json: str, cs: Optional[CoordSystem] = None) → Geometry
Возвращает геометрию из ее „GeoJSON“ представления.

Параметры

- **json** – json представление в виде строки.
- **cs** – Система координат.

static from_mif(mif: str) → Geometry
Возвращает геометрию из ее „MIF“ представления.

Параметры
mif – mif представление в виде строки.

static from_wkb(wkb: bytes, coordsystem: Optional[CoordSystem] = None) → Geometry
Создает геометрический объект из строки формата WKB.

Параметры

- **wkb** – Строка WKB
- **coordsystem** – Система координат, которая будет установлена для геометрии.

Список 131: Пример.

```
wkb = b'\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00$@\x00\x00\x00\x00\x00\x00
↪$@'
pnt = Geometry.from_wkb(wkb)
```

static from_wkt(wkt: str, coordsystem: Optional[CoordSystem] = None) → Geometry
Создает геометрический объект из строки формата WKT.

Параметры

- **wkt** – Строка WKT. Допустимо задание строки в формате с указанием SRID (EWKT). В данном случае система координат для создаваемой геометрии будет установлено исходя их этого значения.

- **coordsystem** – Система координат, которая будет установлена для геометрии. Если строка задана в виде EWKT и указано значение SRID, игнорируется.

Список 132: Пример.

```
polygon = Geometry.from_wkt('POLYGON ((10 10, 100 100, 100 10, 10 10))')
point = Geometry.from_wkt('SRID=4326;POINT (10 10)')
crs = CoordSystem.from_epsg(4326)
pline = Geometry.from_wkt('LINESTRING (30 10, 10 30, 40 40)', crs)
```

get_area(area_unit: Optional[AreaUnit] = None) → float

Рассчитывает площадь, если объект площадной. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в квадратных метрах.

Список 133: Пример.

```
# В проекции Меркатора:
csMercatorKm = CoordSystem.from_prj("10, 104, 1, 0")
poly = Polygon([(0, 0), (0, 2), (2, 2), (2, 0)], cs=csMercatorKm)
print('Area Mercator km:', poly.get_area())
print('Area Mercator m:', poly.get_area(Unit.sq_m))
print('Perimeter Mercator km:', poly.get_perimeter())
print('Perimeter Mercator m:', poly.get_perimeter(Unit.m))
# В Широте/Долготе
poly.coordsystem = CoordSystem.from_prj("1, 104")
print('Area LL m:', poly.get_area())
print('Area LL km:', poly.get_area(Unit.sq_km))
print('Perimeter LL m:', poly.get_perimeter())
print('Perimeter LL km:', poly.get_perimeter(Unit.km))
'''
>>> Area Mercator km: 4.017946986632519
>>> Area Mercator m: 4017946.9866325194
>>> Perimeter Mercator km: 8.017972048421552
>>> Perimeter Mercator m: 8017.972048421552
>>> Area LL m: 49452172514.0342
>>> Area LL km: 49452.1725140342
>>> Perimeter LL m: 889423.5067063896
>>> Perimeter LL km: 889.4235067063896
'''
```

Параметры

area_unit – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и она не задана, то производится расчет на плоскости.

get_distance(other: Geometry, u: Optional[LinearUnit] = None) → float

Производит расчет расстояния до объекта other. Результат возвращает в СК текущего объекта.

Параметры

- **other** – Анализируемый объект.
- **u** – Единицы измерения, в которых требуется получить результат.

Список 134: Пример.

```
# Создадим геометрию без СК
first = Point(0, 0)
second = Point(10, 0)
print('В плане:', first.get_distance(second))
# Установим разные СК для точек
first.coordsystem = CoordSystem.from_prj("10, 104, 7, 0")
second.coordsystem = CoordSystem.from_prj("1, 104")
print('На сфере м:', first.get_distance(second))
print('На сфере км:', first.get_distance(second, Unit.km))
'''
>>> В плане: 10.0
>>> На сфере м: 1111948.7428468117
>>> На сфере км: 1111.9487428468117
'''
```

get_length(u: Optional[LinearUnit] = None) → float

Рассчитывает длину геометрии. Метод применим только для линейных объектов. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Список 135: Пример.

```
# В проекции Меркатора:
csMercatorKm = CoordSystem.from_prj("10, 104, 1, 0")
ls = Line((0, 0), (10, 0), csMercatorKm)
print('Length Mercator km:', ls.get_length())
print('Lenght Mercator m:', ls.get_length(Unit.m))
# В Широте/Долготе
csLL = CoordSystem.from_prj("1, 104")
ls = Line((0, 0), (10, 0), csLL)
print('Length LL m:', ls.get_length())
print('Length LL km:', ls.get_length(Unit.km))
'''
>>> Length Mercator km: 9.988805508567783
>>> Lenght Mercator m: 9988.805508567782
>>> Length LL m: 1111948.7428468117
>>> Length LL km: 1111.9487428468117
'''
```

Параметры

u – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и она не задана, то производится расчет на плоскости.

get_perimeter(u: Optional[LinearUnit] = None) → float

Рассчитывает периметр геометрии. Метод применим только для площадных объектов. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Пример см. `get_area()`

Параметры

u – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и

она не задана, то производится расчет на плоскости.

intersection(other: Geometry) → Geometry

Возвращает область пересечения с другой геометрией.

intersects(other: Geometry) → bool

Возвращает True, если геометрии пересекаются.

property is_valid: bool

Проверяет геометрию на валидность.

property is_valid_reason: str

Если геометрия неправильная, возвращает краткую аннотацию причины.

property name: str

Возвращает наименование геометрического объекта.

overlaps(other: Geometry) → bool

Возвращает True, если пересечение геометрий отличается от обеих геометрий.

static point_by_azimuth(point: Union[Pnt, Tuple[float, float]], azimuth: float, distance: float, cs: Optional[CoordSystem] = None) → Pnt

Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

См.также:

Обратная функция `distance_by_points()`

Параметры

- **point** – Точка, относительно которой производится расчет. Если задана СК, то в ней.
- **azimuth** – Азимут в градусах, указывающий направление.
- **distance** – Расстояние по азимуту. Задается в метрах.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращает результирующую точку.

relate(other: Geometry) → str

Проверяет отношения между объектами. Подробнее см. DE-9IM

reproject(cs: CoordSystem) → Geometry

Перепроецирует геометрию в другую систему координат.

Параметры

cs – СК, в которой требуется получить объект.

rotate(point: Union[Pnt, Tuple[float, float]], angle: float) → Geometry

Поворот геометрии относительно заданной точки. Поворот производится против часовой стрелки.

Параметры

- **point** – Точка, вокруг которой необходимо произвести поворот.
- **angle** – Угол поворота в градусах.

scale(kx: float, ky: float) → Geometry

Масштабирует объект по заданным коэффициентам масштабирования. После выполнения операции центр результирующего объекта остается прежним.

Параметры

- **kx** – Масштабирование по координате X.
- **ky** – Масштабирование по координате Y.

shift(dx: float, dy: float) → Geometry

Смещает объект на заданную величину. Значения задаются в текущей проекции.

Параметры

- **dx** – Сдвиг по координате X.
- **dy** – Сдвиг по координате Y.

split_by_polyline(splitter: Geometry) → Optional[Geometry]

Разрезает объект линейным объектом.

Параметры
splitter – Геометрический объект, которым будет производиться разрезание объекта.

Результат
Возвращает полученный результат

```
poly = Polygon((0,10), (10,10), (10, 0))
line = Line((-5,5), (20,5))
r = poly.split_by_polyline(line)
print(r.wkt)
'''
>>> GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5, 5 5, 0 10)), POLYGON␣
↪((10 5, 10 0, 5 5, 10 5))) GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5,␣
↪5 5, 0 10)), POLYGON ((10 5, 10 0, 5 5, 10 5)))
'''
```

property startAngle: float

Начальный угол дуги.

symmetric_difference(other: Geometry) → Geometry

Возвращает логический XOR областей геометрий (объединение разниц).

Параметры
other – Геометрия для анализа.

to_geojson() → str

Преобразует геометрию в формат „GeoJSON“

to_linestring() → Optional[Geometry]

Пробует геометрию преобразовать в линейный объект. В случае неудачи возвращает None.

to_mif() → str

Преобразует геометрию в формат MIF.

to_polygon() → Optional[Geometry]

Пробует геометрию преобразовать в площадной объект. В случае неудачи возвращает None.

to_wkb() → bytes

Возвращает WKB строку для геометрии.

to_wkt() → str

Возвращает WKT строку для геометрии.

touches(other: Geometry) → bool

Возвращает True, если геометрии соприкасаются.

property type: Type

Возвращает тип геометрического элемента.

Таблица 37: Возможные значения

Значение	Наименование
Unknown	Не определен
Point	Точка
Line	Линия
LineString	Полилиния
Polygon	Полигон
MultiPoint	Коллекция точек
MultiLineString	Коллекция полилиний
MultiPolygon	Коллекция полигонов
GeometryCollection	Смешанная коллекция
Arc	Дуга
Ellipse	Эллипс
Rectangle	Прямоугольник
RoundedRectangle	Скругленный прямоугольник
Text	Текст

Список 136: Пример.

```
point = Point(10, 10)
if point.type == GeometryType.Point:
    print('Это точка')
```

union(other: Geometry) → Geometry

Возвращает результат объединения двух геометрий.

within(other: Geometry) → bool

Возвращает True, если геометрия находится полностью внутри геометрии other.

property xRadius: float

Радиус большой полуоси прямоугольника, в который вписана дуга.

property yRadius: float

Радиус малой полуоси прямоугольника, в который вписана дуга.

22.12.14 Text - Текст

class axipy.**Text**

Базовые классы: Geometry

Геометрический объект типа текст.

Предупреждение: Геометрия текста и, в отличие от остальных типов объектов, определяется кроме геометрического представления так же и стилем его оформления axipy.TextStyle. Так же стоит заметить, что параметры геометрии текста зависит от того, куда (с карту или отчет) будет добавлен созданный текст. Поэтому созданный объект для карты должен быть добавлен в карту, а для отчета - в отчет. Подробнее см. create_by_style()

Примечание: Для создания текстового объекта с указанием его размера в поинтах можно использовать метод create_by_style()

Параметры

- **text** – Строка с текстом. Многострочный текст можно задать, вставив символ «\n» в место переноса.
- **rect** – Прямоугольник, в который будет вписан текст. Прямоугольник задается в координатах отчета. Верхней левой точкой является startPoint. В этот прямоугольник будет произведена попытка размещения текста.
- **angle** – Угол поворота текстового объекта. Задается значением в градусах против ЧС по отношению к горизонтали.
- **view** – Окно карты или отчета.
- **cs** – Система Координат, в которой создается геометрия.

Список 137: Пример создания текста и вставки его в отчет.

```
report_view = view_manager.create_reportview()
r = Rect(8, 6, 11, 7)
text = Text("Пример\nтекста", rect=r, angle=20, view=report_view)
geomItem = GeometryReportItem()
geomItem.geometry = text
geomItem.style = Style.for_geometry(text)
report_view.report.items.add(geomItem)
```

Конструктор класса:

__init__(text, rect[, view, angle, cs])	Создает экземпляр класса.

Классовые методы:

create_by_style(text, point, style, view[, ...])	Создает текстовый объект по точке привязки и стиля для карты или отчета.
distance_by_points(start, end[, cs])	Производит расчет расстояния между двумя точками и азимут от первой до второй точки.
from_geojson(json[, cs])	Возвращает геометрию из ее 'GeoJSON' представления.
from_mif(mif)	Возвращает геометрию из ее 'MIF' представления.
from_wkb(wkb[, coordsystem])	Создает геометрический объект из строки формата WKB.
from_wkt(wkt[, coordsystem])	Создает геометрический объект из строки формата WKT.
point_by_azimuth(point, azimuth, distance[, cs])	Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

Свойства:

angle	Угол поворота текста, отсчитываемый от горизонтали против часовой стрелки
bounds	Возвращает минимальный ограничивающий прямоугольник.
coordsystem	Система Координат (СК) геометрии.
endPoint	Координаты точки выноски (указки).
height	"
is_valid	Проверяет геометрию на валидность.
is_valid_reason	Если геометрия неправильная, возвращает краткую аннотацию причины.
name	Возвращает наименование геометрического объекта.
rect_as_polygon	Представление ограничивающего прямоугольника в виде полигона.
startPoint	Координаты точки привязки.
text	Текст.
type	Возвращает тип геометрического элемента.
width	"

Методы:

affine_transform(trans)	Трансформирует объект исходя из заданных параметров трансформации.

continues on next page

Таблица 38 – продолжение с предыдущей страницы

almost_equals(other, tolerance)	Производит примерное сравнения с другой геометрией в пределах заданной точности.
boundary()	Возвращает границы геометрии в виде полилинии.
buffer(distance[, resolution, capStyle, ...])	Производит построение буфера вокруг объекта.
centroid()	Возвращает центроид геометрии.
clone()	Создает копию объекта.
contains(other)	Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.
convex_hull()	Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.
covers(other)	Возвращает True, если геометрия охватывает геометрию other.
crosses(other)	Возвращает True, если при пересечении геометрий объекты частично пересекаются.
difference(other)	Возвращает область первой геометрии, которая не пересечена второй геометрией.
disjoint(other)	Возвращает True, если геометрии не пересекаются и не соприкасаются.
envelope()	Возвращает полигон, описывающий заданную геометрию.
equals(other)	Производит сравнение с другой геометрией.
get_area([area_unit])	Рассчитывает площадь, если объект площадной.
get_distance(other[, u])	Производит расчет расстояния до объекта other.
get_length([u])	Рассчитывает длину геометрии.
get_perimeter([u])	Рассчитывает периметр геометрии.
intersection(other)	Возвращает область пересечения с другой геометрией.
intersects(other)	Возвращает True, если геометрии пересекаются.
overlaps(other)	Возвращает True, если пересечение геометрий отличается от обеих геометрий.
relate(other)	Проверяет отношения между объектами.
reproject(cs)	Перепроецирует геометрию в другую систему координат.
rotate(point, angle)	Поворот геометрии относительно заданной точки.
scale(kx, ky)	Масштабирует объект по заданным коэффициентам масштабирования.
shift(dx, dy)	Смещает объект на заданную величину.

continues on next page

Таблица 38 – продолжение с предыдущей страницы

size_for_view(style, view)	Размер шрифта в пунктах для конкретного окна.
split_by_polyline(splitter)	Разрезает объект линейным объектом.
symmetric_difference(other)	Возвращает логический XOR областей геометрий (объединение разниц).
to_geojson()	Преобразует геометрию в формат 'GeoJSON'
to_linestring()	Пробует геометрию преобразовать в линейный объект.
to_mif()	Преобразует геометрию в формат MIF.
to_polygon()	Пробует геометрию преобразовать в площадной объект.
to_wkb()	Возвращает WKB строку для геометрии.
to_wkt()	Возвращает WKT строку для геометрии.
touches(other)	Возвращает True, если геометрии соприкасаются.
union(other)	Возвращает результат объединения двух геометрий.
within(other)	Возвращает True, если геометрия находится полностью внутри геометрии other.

__init__(text: str, rect: Union[Rect, QRectF], view: Optional[Union[MapView, ReportView]] = None, angle: float = 0, cs: Optional[CoordSystem] = None)

Создает экземпляр класса.

affine_transform(trans: QTransform) → Geometry

Трансформирует объект исходя из заданных параметров трансформации.

См.также:

Для простых операций типа сдвиг, масштабирование и поворот, рекомендуется использовать shift(), scale(), rotate() соответственно.

Параметры

trans – Матрица трансформации.

almost_equals(other: Geometry, tolerance: float) → bool

Производит примерное сравнения с другой геометрией в пределах заданной точности.

Параметры

- **other** – Сравниваемый объект.
- **tolerance** – Точность сравнения.

Результат

Возвращает True, если геометрии равны в пределах заданного отклонения.

property angle: float

Угол поворота текста, отсчитываемый от горизонтали против часовой стрелки

boundary() → Geometry

Возвращает границы геометрии в виде полилинии.

property bounds: Rect

Возвращает минимальный ограничивающий прямоугольник.

buffer(distance: float, resolution: int = 16, capStyle: LineCapStyle = LineCapStyle.Round, join_style: LineJoinStyle = LineJoinStyle.Round, mitreLimit: float = 5.0) → Geometry

Производит построение буфера вокруг объекта.

Параметры

- **distance** – Ширина буфера.
- **resolution** – Количество сегментов на квадрант.
- **capStyle** – Стиль окончания.
- **join_style** – Стиль соединения.
- **mitreLimit** – Предел среза.

centroid() → Geometry

Возвращает центроид геометрии.

clone() → Geometry

Создает копию объекта.

contains(other: Geometry) → bool

Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.

convex_hull() → Geometry

Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.

property coordsystem: CoordSystem

Система Координат (СК) геометрии.

covers(other: Geometry) → bool

Возвращает True, если геометрия охватывает геометрию other.

classmethod create_by_style(text: str, point: Union[Pnt, QPointF], style: Style, view: Union[MapView, ReportView], angle: float = 0, cs: Optional[CoordSystem] = None)

Создает текстовый объект по точке привязки и стиля для карты или отчета. Результирующий объект будет корректен только для текущего состояния окна карты или отчета, куда он должен быть добавлен. Это связано с тем, что расчет размера производится на основании текущего установленного масштаба. Поэтому при изменении масштаба перед добавлением необходимо заново создать текстовый объект.

Параметры

- **text** – Текст.
- **point** – Точка привязки `startPoint`. Этой точкой служит левая верхняя точка.
- **style** – Стиль оформления текста.

- **view** – Окно карты или отчета.
- **angle** – Угол поворота текстового объекта. Задается значением в градусах против ЧС по отношению к горизонтали.
- **cs** – Система Координат, в которой создается геометрия.

Список 138: Пример изменения параметров текстовых объектов для окна карты.

```
import axipy
from PySide2.QtCore import Qt

mapview = axipy.view_manager.active
if len(axipy.data_manager) and mapview:
    table = axipy.data_manager.tables[0]
    # Новый стиль для текстовых объектов
    new_style = axipy.TextStyle("Droid Sans", 20)
    new_style.color = Qt.red
    new_style.bold = True
    for f in table.items():
        # Для всех текстовых объектов применим данный стиль
        if isinstance(f.geometry, axipy.Text):
            g = f.geometry
            text = axipy.Text.create_by_style(g.text, g.startPoint, style=new_
↪style, view=mapview, angle=g.angle)
            text.endPoint = g.endPoint
            new_style.callout = f.style.callout
            new_style.callout_style = f.style.callout_style
            f.style = new_style
            f.geometry = text
            table.update([f])
    # Сохраним изменения
    table.commit()
```

Список 139: Пример создания текста для отчета.

```
report_view = view_manager.create_reportview()
style_txt = Style.from_mapinfo('Font ("Times New Roman", 0, 23, 16711680)')
text = Text.create_by_style("Пример\nтекста2", (10, 10), style=style_txt,␣
↪view=report_view, angle=20)
# Поменяем угол
text.angle = -20
# Изменим начальную точку
text.startPoint = (11, 11)
```

crosses(other: Geometry) → bool

Возвращает True, если при пересечении геометрий объекты частично пересекаются.

difference(other: Geometry) → Geometry

Возвращает область первой геометрии, которая не пересечена второй геометрией.

disjoint(other: Geometry) → bool

Возвращает True, если геометрии не пересекаются и не соприкасаются.

static distance_by_points(start: Union[Pnt, Tuple[float, float]], end: Union[Pnt, Tuple[float, float]], cs: Optional[CoordSystem] = None) → Tuple

Производит расчет расстояния между двумя точками и азимут от первой до второй точки.

См.также:

Обратная функция `point_by_azimuth()`

Параметры

- **start** – Начальная точка. Если задана СК, то координаты в ней.
- **end** – Конечная точка. Если задана СК, то координаты в ней.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращается пара значений (расстояние в метрах, азимут в градусах)

property endPoint: Pnt

Координаты точки выноски (указки).

envelope() → Geometry

Возвращает полигон, описывающий заданную геометрию.

equals(other: Geometry) → bool

Производит сравнение с другой геометрией.

Параметры

other – Сравниваемая геометрия.

Результат

Возвращает True, если геометрии равны.

static from_geojson(json: str, cs: Optional[CoordSystem] = None) → Geometry

Возвращает геометрию из ее „GeoJSON“ представления.

Параметры

- **json** – json представление в виде строки.
- **cs** – Система координат.

static from_mif(mif: str) → Geometry

Возвращает геометрию из ее „MIF“ представления.

Параметры

mif – mif представление в виде строки.

static from_wkb(wkb: bytes, coordsystem: Optional[CoordSystem] = None) → Geometry

Создает геометрический объект из строки формата WKB.

Параметры

- **wkb** – Строка WKB
- **coordsystem** – Система координат, которая будет установлена для геометрии.

Список 140: Пример.

```
wkb = b'\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00$@\x00\x00\x00\x00\x00\x00
↪$@'
pnt = Geometry.from_wkb(wkb)
```

static from_wkt(wkt: str, coordsystem: Optional[CoordSystem] = None) → Geometry

Создает геометрический объект из строки формата WKT.

Параметры

- **wkt** – Строка WKT. Допустимо задание строки в формате с указанием SRID (EWKT). В данном случае система координат для создаваемой геометрии будет установлено исходя их этого значения.
- **coordsystem** – Система координат, которая будет установлена для геометрии. Если строка задана в виде EWKT и указано значение SRID, игнорируется.

Список 141: Пример.

```
polygon = Geometry.from_wkt('POLYGON ((10 10, 100 100, 100 10, 10 10))')
point = Geometry.from_wkt('SRID=4326;POINT (10 10)')
crs = CoordSystem.from_epsg(4326)
pline = Geometry.from_wkt('LINESTRING (30 10, 10 30, 40 40)', crs)
```

get_area(area_unit: Optional[AreaUnit] = None) → float

Рассчитывает площадь, если объект площадной. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в квадратных метрах.

Список 142: Пример.

```
# В проекции Меркатора:
csMercatorKm = CoordSystem.from_prj("10, 104, 1, 0")
poly = Polygon([(0, 0), (0, 2), (2, 2), (2, 0)], cs=csMercatorKm)
print('Area Mercator km:', poly.get_area())
print('Area Mercator m:', poly.get_area(Unit.sq_m))
print('Perimeter Mercator km:', poly.get_perimeter())
print('Perimeter Mercator m:', poly.get_perimeter(Unit.m))
# В Широте/Долготе
poly.coordsystem = CoordSystem.from_prj("1, 104")
print('Area LL m:', poly.get_area())
print('Area LL km:', poly.get_area(Unit.sq_km))
print('Perimeter LL m:', poly.get_perimeter())
print('Perimeter LL km:', poly.get_perimeter(Unit.km))
'''
>>> Area Mercator km: 4.017946986632519
>>> Area Mercator m: 4017946.9866325194
>>> Perimeter Mercator km: 8.017972048421552
>>> Perimeter Mercator m: 8017.972048421552
>>> Area LL m: 49452172514.0342
>>> Area LL km: 49452.1725140342
>>> Perimeter LL m: 889423.5067063896
>>> Perimeter LL km: 889.4235067063896
'''
```

Параметры
area_unit – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и она не задана, то производится расчет на плоскости.

get_distance(other: Geometry, u: Optional[LinearUnit] = None) → float

Производит расчет расстояния до объекта other. Результат возвращает в СК текущего объекта.

Параметры

- **other** – Анализируемый объект.
- **u** – Единицы измерения, в которых требуется получить результат.

Список 143: Пример.

```
# Создадим геометрию без СК
first = Point(0, 0)
second = Point(10, 0)
print('В плане:', first.get_distance(second))
# Установим разные СК для точек
first.coordsystem = CoordSystem.from_prj("10, 104, 7, 0")
second.coordsystem = CoordSystem.from_prj("1, 104")
print('На сфере м:', first.get_distance(second))
print('На сфере км:', first.get_distance(second, Unit.km))
'''
>>> В плане: 10.0
>>> На сфере м: 1111948.7428468117
>>> На сфере км: 1111.9487428468117
'''
```

get_length(u: Optional[LinearUnit] = None) → float

Рассчитывает длину геометрии. Метод применим только для линейных объектов. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Список 144: Пример.

```
# В проекции Меркатора:
csMercatorKm = CoordSystem.from_prj("10, 104, 1, 0")
ls = Line((0, 0), (10, 0), csMercatorKm)
print('Length Mercator km:', ls.get_length())
print('Lenght Mercator m:', ls.get_length(Unit.m))
# В Широте/Долготе
csLL = CoordSystem.from_prj("1, 104")
ls = Line((0, 0), (10, 0), csLL)
print('Length LL m:', ls.get_length())
print('Length LL km:', ls.get_length(Unit.km))
'''
>>> Length Mercator km: 9.988805508567783
>>> Lenght Mercator m: 9988.805508567782
>>> Length LL m: 1111948.7428468117
>>> Length LL km: 1111.9487428468117
'''
```

Параметры
u – Единица измерения, в которой необходимо получить результат.

Если не задана, то используется единица измерения для СК. Если и она не задана, то производится расчет на плоскости.

get_perimeter(u: Optional[LinearUnit] = None) → float

Рассчитывает периметр геометрии. Метод применим только для площадных объектов. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Пример см. `get_area()`

Параметры

u – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и она не задана, то производится расчет на плоскости.

property height

» Высота прямоугольника с текстом.

intersection(other: Geometry) → Geometry

Возвращает область пересечения с другой геометрией.

intersects(other: Geometry) → bool

Возвращает True, если геометрии пересекаются.

property is_valid: bool

Проверяет геометрию на валидность.

property is_valid_reason: str

Если геометрия неправильная, возвращает краткую аннотацию причины.

property name: str

Возвращает наименование геометрического объекта.

overlaps(other: Geometry) → bool

Возвращает True, если пересечение геометрий отличается от обеих геометрий.

static point_by_azimuth(point: Union[Pnt, Tuple[float, float]], azimuth: float, distance: float, cs: Optional[CoordSystem] = None) → Pnt

Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

См.также:

Обратная функция `distance_by_points()`

Параметры

- **point** – Точка, относительно которой производится расчет. Если задана СК, то в ней.
- **azimuth** – Азимут в градусах, указывающий направление.
- **distance** – Расстояние по азимуту. Задается в метрах.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращает результирующую точку.

property rect_as_polygon: Polygon

Представление ограничивающего прямоугольника в виде полигона.

relate(other: Geometry) → str

Проверяет отношения между объектами. Подробнее см. DE-9IM

reproject(cs: CoordSystem) → Geometry

Перепроецирует геометрию в другую систему координат.

Параметры
cs – СК, в которой требуется получить объект.

rotate(point: Union[Pnt, Tuple[float, float]], angle: float) → Geometry

Поворот геометрии относительно заданной точки. Поворот производится против часовой стрелки.

Параметры

- **point** – Точка, вокруг которой необходимо произвести поворот.
- **angle** – Угол поворота в градусах.

scale(kx: float, ky: float) → Geometry

Масштабирует объект по заданным коэффициентам масштабирования. После выполнения операции центр результирующего объекта остается прежним.

Параметры

- **kx** – Масштабирование по координате X.
- **ky** – Масштабирование по координате Y.

shift(dx: float, dy: float) → Geometry

Смещает объект на заданную величину. Значения задаются в текущей проекции.

Параметры

- **dx** – Сдвиг по координате X.
- **dy** – Сдвиг по координате Y.

size_for_view(style, view: Union[MapView, ReportView]) → float

Размер шрифта в пунктах для конкретного окна. Значение меняется в зависимости от текущего масштаба.

Параметры

- **style** – Стиль оформления текста.
- **view** – Окно карты или отчета.

split_by_polyline(splitter: Geometry) → Optional[Geometry]

Разрезает объект линейным объектом.

Параметры
splitter – Геометрический объект, которым будет производиться разрезание объекта.

Результат
Возвращает полученный результат

```
poly = Polygon((0,10), (10,10), (10, 0))
line = Line((-5,5), (20,5))
r = poly.split_by_polyline(line)
print(r.wkt)
'''
>>> GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5, 5 5, 0 10)), POLYGON␣
↪((10 5, 10 0, 5 5, 10 5))) GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5,␣
↪5 5, 0 10)), POLYGON ((10 5, 10 0, 5 5, 10 5)))
'''
```

property startPoint: Pnt

Координаты точки привязки.

symmetric_difference(other: Geometry) → Geometry

Возвращает логический XOR областей геометрий (объединение разниц).

Параметры
other – Геометрия для анализа.

property text: str

Текст.

to_geojson() → str

Преобразует геометрию в формат „GeoJSON“

to_linestring() → Optional[Geometry]

Пробует геометрию преобразовать в линейный объект. В случае неудачи возвращает None.

to_mif() → str

Преобразует геометрию в формат MIF.

to_polygon() → Optional[Geometry]

Пробует геометрию преобразовать в площадной объект. В случае неудачи возвращает None.

to_wkb() → bytes

Возвращает WKB строку для геометрии.

to_wkt() → str

Возвращает WKT строку для геометрии.

touches(other: Geometry) → bool

Возвращает True, если геометрии соприкасаются.

property type: Type

Возвращает тип геометрического элемента.

Таблица 39: Возможные значения

Значение	Наименование
Unknown	Не определен
Point	Точка
Line	Линия
LineString	Полилиния
Polygon	Полигон
MultiPoint	Коллекция точек
MultiLineString	Коллекция полилиний
MultiPolygon	Коллекция полигонов
GeometryCollection	Смешанная коллекция
Arc	Дуга
Ellipse	Эллипс
Rectangle	Прямоугольник
RoundedRectangle	Скругленный прямоугольник
Text	Текст

Список 145: Пример.

```
point = Point(10, 10)
if point.type == GeometryType.Point:
    print('Это точка')
```

union(other: Geometry) → Geometry

Возвращает результат объединения двух геометрий.

property width

» Ширина прямоугольника с текстом.

within(other: Geometry) → bool

Возвращает True, если геометрия находится полностью внутри геометрии other.

22.12.15 LineDirection - Направление линии

class axipy.**LineDirection**

Базовые классы: int, Enum

Направление линии.

Атрибуты:

Clockwise	По часовой стрелке
CounterClockwise	Против часовой стрелки

Clockwise

По часовой стрелке

CounterClockwise

Против часовой стрелки

22.12.16 LineCapStyle - Стиль окончания линии

class axipy.**LineCapStyle**

Стиль окончания линии.

Атрибуты:

Flat	Конец линии срезан по ее окончанию
Round	Конец линии закруглен
Square	Конец линии срезан посередине между ее окончанием и полученным результатом

Flat

Конец линии срезан по ее окончанию

Round

Конец линии закруглен

Square

Конец линии срезан посередине между ее окончанием и полученным результатом

22.12.17 LineJoinStyle - Стиль соединения линий

class axipy.**LineJoinStyle**

Стиль соединения линий.

Атрибуты:

Bevel	Соединение - точка пересечения края результата
Mitre	Соединение линий плоское
Round	Соединение линий скруглено

Bevel

Соединение - точка пересечения края результата

Mitre

Соединение линий плоское

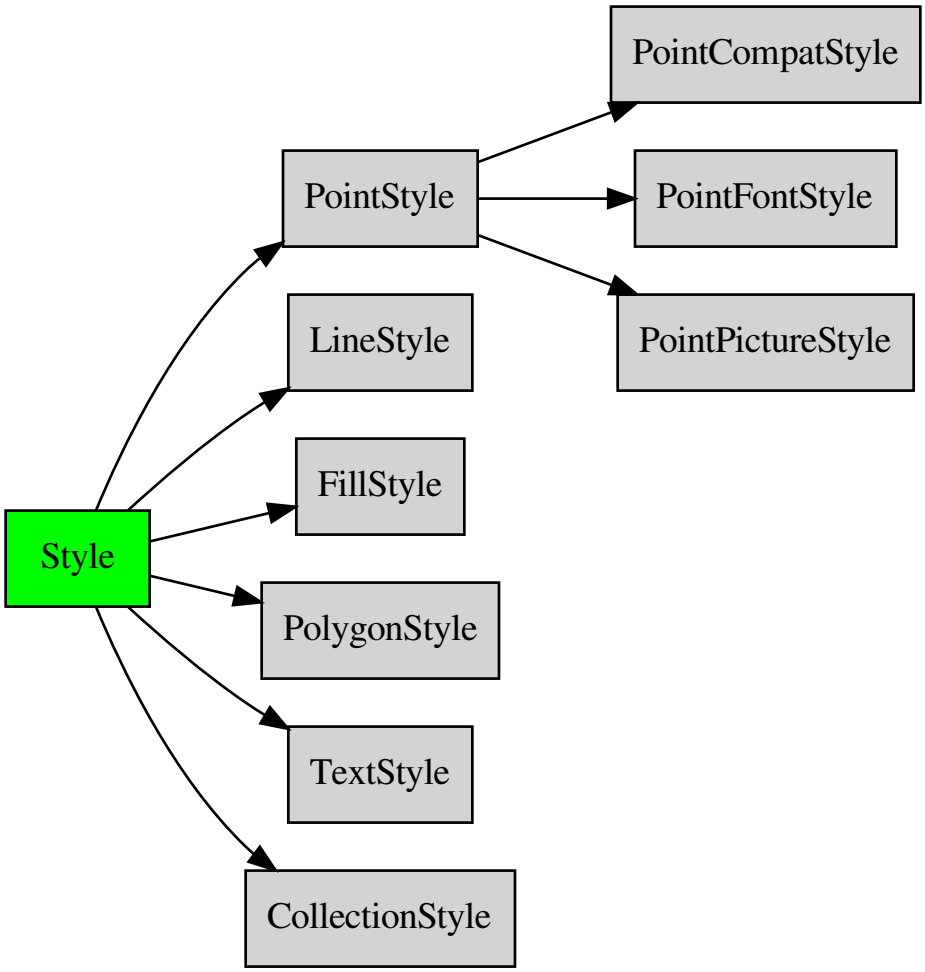
Round

Соединение линий скруглено

22.13 Style - Стиль

22.13.1 Style - Стиль

Иерархия классов стилей геометрических объектов:



class axipy.**Style**

Абстрактный класс стиля оформления геометрического объекта. Определяет как будет отрисован геометрический объект.

Примечание: Для получения текстового представления стиля можно воспользоваться функцией `str`.

Классовые методы:

for_geometry(geom)	Возвращает стиль по умолчанию для переданного объекта.
from_mapinfo(mapbasic_string)	Получает стиль из строки формата MapBasic.

Методы:

clone()	Создаёт копию объекта стиля
draw(geometry, painter)	Рисует геометрический объект с текущим стилем в произвольном контексте вывода.
to_mapinfo()	Возвращает строковое представление в формате MapBasic.

clone() → Style

Создаёт копию объекта стиля

draw(geometry: Geometry, painter: QPainter)

Рисует геометрический объект с текущим стилем в произвольном контексте вывода. Это может быть востребовано при желании отрисовать геометрию со стилем на форме или диалоге.

Параметры

- **geometry** – Геометрия. Должна соответствовать стилю. Т.е. если объект полигон, а стиль для рисования точечных объектов, то ничего нарисовано не будет.
- **painter** – Контекст вывода.

Список 146: Пример отрисовки в растре и сохранение результата в файле.

```
image = QImage(100, 100, QImage.Format_ARGB32_Premultiplied)
image.fill(Qt.white)
painter = QPainter(image)
point = Point(50, 50)
style = PointStyle.create_mi_font(42, Qt.red, 24)
style.draw(point, painter)
image.save(filename)
```

classmethod for_geometry(geom: Geometry) → Style

Возвращает стиль по умолчанию для переданного объекта.

Параметры

geom – Геометрический объект, для которого необходимо получить соответствующий ему стиль.

classmethod from_mapinfo(mapbasic_string: str) → Optional[Style]

Получает стиль из строки формата MapBasic.

Параметры

mapbasic_string – Строка в формате MapBasic.

```
style = Style.from_mapinfo("Pen (1, 2, 0) Brush (8, 255)")
```

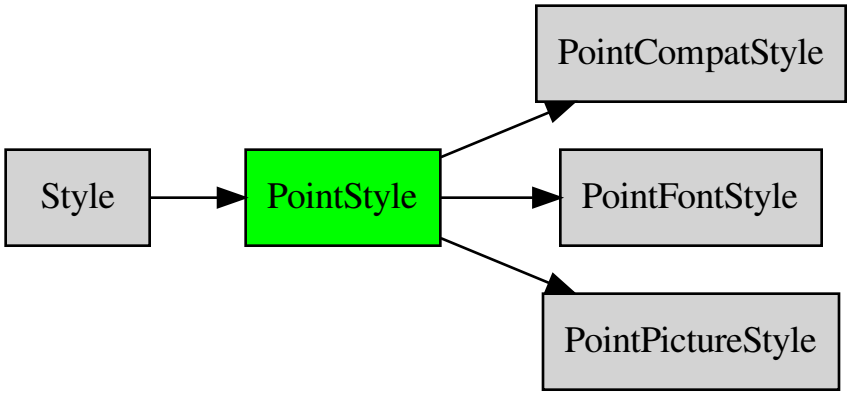
to_mapinfo() → str

Возвращает строковое представление в формате MapBasic.

```
print(style.to_mapinfo())
'''
>>> Pen (1, 2, 0) Brush (8, 255)
'''
```

22.13.2 PointStyle - Стиль точек

Иерархия классов стилей геометрических объектов:



class axipy.**PointStyle**

Базовые классы: Style

Стиль оформления точечных объектов.

По умолчанию создается стиль на базе шрифта True Type, а параметры аналогичны значениям по умолчанию в методе create_mi_font().

Поддерживается 3 вида оформления:

- Совместимое с MapInfo версии 3. Для создания такого стиля необходимо использовать create_mi_compat().
- На базе шрифтов True Type. Задано по умолчанию. Стиль создается посредством create_mi_font().
- На базе растрового файла. Стиль можно создать с помощью create_mi_picture().

Список 147: Пример.

```
style_1 = PointStyle.create_mi_compat(color=Qt.blue)
style_2 = PointStyle.create_mi_font(42, Qt.red, 24)
style_3 = PointStyle.create_mi_picture('AMBU-64.bmp')
```

Классовые методы:

create_mi_compat([symbol, color, pointSize])	Создание стиля в виде совместимого с MapInfo 3 PointCompatStyle.
create_mi_font([symbol, color, size, ...])	Создание стиля на базе шрифта True Type PointFontStyle.
create_mi_picture(filename[, color, size, ...])	Создание стиля с ссылкой на растровый файл PointPictureStyle.
for_geometry(geom)	Возвращает стиль по умолчанию для переданного объекта.
from_mapinfo(mapbasic_string)	Получает стиль из строки формата MapBasic.

Свойства:

color	Цвет символа.

Методы:

clone()	Создаёт копию объекта стиля
draw(geometry, painter)	Рисует геометрический объект с текущим стилем в произвольном контексте вывода.
to_mapinfo()	Возвращает строковое представление в формате MapBasic.

clone() → Style

Создаёт копию объекта стиля

property color: QColor

Цвет символа.

static create_mi_compat(symbol: int = 35, color: QColor = Qt.red, pointSize: int = 8) → PointCompatStyle

Создание стиля в виде совместимого с MapInfo 3 PointCompatStyle.

Параметры

- **symbol** – Номер символа, который будет отображен при отрисовке. Для создания невидимого символа используйте значение 31. Стандартный набор условных знаков включает символы от 31 до 67.
- **color** – Цвет символа
- **pointSize** – Целое число, размер символа в пунктах от 1 до 48.

static create_mi_font(symbol: int = 36, color: QColor = Qt.red, size: int = 8, fontname: str = 'Axioma MI MapSymbols', fontstyle: int = 0, rotation: float = 0.0) → PointFontStyle

Создание стиля на базе шрифта True Type PointFontStyle.

Параметры

- **symbol** – Целое, имеющее значение 31 или больше, определяющее, какой используется символ из шрифтов TrueType. Для создания невидимого символа используйте значение 31.
- **color** – Цвет символа
- **size** – Целое число, размер символа в пунктах от 1 до 48;
- **fontname** – Строка с именем шрифта TrueType (например, значение по умолчание „Axioma MI MapSymbols“)
- **fontstyle** – Стиль дополнительного оформления, например, курсивный текст. Возможные параметры см. в таблице ниже. Для указания нескольких параметров их суммируют между собой.
- **rotation** – Угол поворота символа в градусах.

static create_mi_picture(filename: str, color: QColor = Qt.black, size: int = 12, customstyle: int = 0) → PointPictureStyle

Создание стиля с ссылкой на растровый файл PointPictureStyle.

Параметры

- **filename** – Наименование растрового файла. Строка до 31 символа длиной. Данный файл должен находится в каталоге CustSymb с ресурсами. Например, “Arrow.BMP”.
- **color** – Цвет символа.
- **size** – Размер символа в в пунктах от 1 до 48.
- **customstyle** – Задание дополнительных параметров стиля оформления.

draw(geometry: Geometry, painter: QPainter)

Рисует геометрический объект с текущим стилем в произвольном контексте вывода. Это может быть востребовано при желании отрисовать геометрию со стилем на форме или диалоге.

Параметры

- **geometry** – Геометрия. Должна соответствовать стилю. Т.е. если объект полигон, а стиль для рисования точечных объектов, то ничего нарисовано не будет.
- **painter** – Контекст вывода.

Список 148: Пример отрисовки в растре и сохранение результата в файле.

```
image = QImage(100, 100, QImage.Format_ARGB32_Premultiplied)
image.fill(Qt.white)
painter = QPainter(image)
point = Point(50, 50)
style = PointStyle.create_mi_font(42, Qt.red, 24)
style.draw(point, painter)
image.save(filename)
```

classmethod for_geometry(geom: Geometry) → Style

Возвращает стиль по умолчанию для переданного объекта.

Параметры
geom – Геометрический объект, для которого необходимо получить соответствующий ему стиль.

classmethod from_mapinfo(mapbasic_string: str) → Optional[Style]

Получает стиль из строки формата MapBasic.

Параметры
mapbasic_string – Строка в формате MapBasic.

```
style = Style.from_mapinfo("Pen (1, 2, 0) Brush (8, 255)")
```

to_mapinfo() → str

Возвращает строковое представление в формате MapBasic.

```
print(style.to_mapinfo())
'''
>>> Pen (1, 2, 0) Brush (8, 255)
'''
```

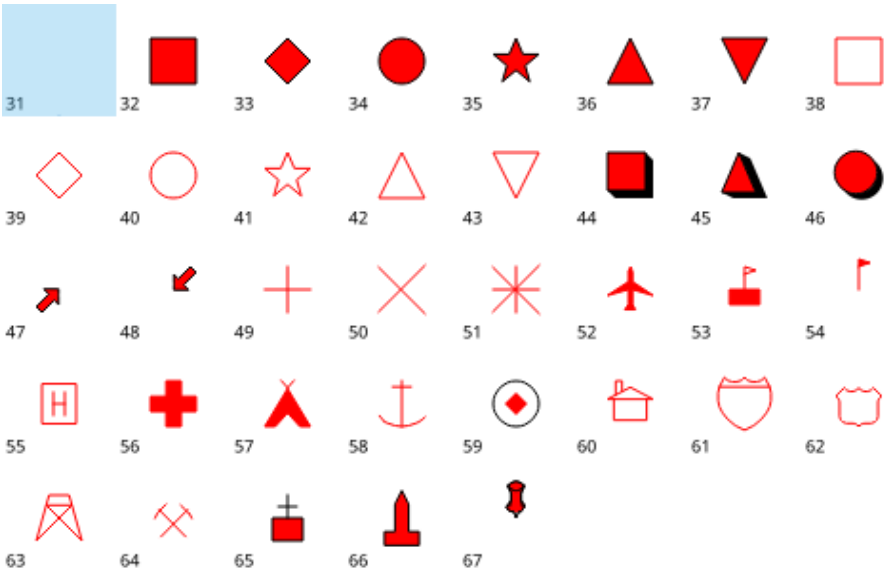
22.13.3 PointCompatStyle - Стиль, совместимый с MapInfo 3

class axipy.**PointCompatStyle**

Базовые классы: PointStyle

Класс стиля, совместимого с MapInfo 3.

В системе доступны следующие стили:



Список 149: Пример.

```
style = PointCompatStyle()
style.color = Qt.blue
style.symbol = 43
```

Конструктор класса:

__init__([symbol, color, pointSize])	Создает экземпляр класса.

Классовые методы:

create_mi_compat([symbol, color, pointSize])	Создание стиля в виде совместимого с MapInfo 3 PointCompatStyle.
create_mi_font([symbol, color, size, ...])	Создание стиля на базе шрифта True Type PointFontStyle.
create_mi_picture(filename[, color, size, ...])	Создание стиля с ссылкой на растровый файл PointPictureStyle.
for_geometry(geom)	Возвращает стиль по умолчанию для переданного объекта.
from_mapinfo(mapbasic_string)	Получает стиль из строки формата MapBasic.

Свойства:

color	Цвет символа.
size	Размер символа в пунктах.
symbol	Номер символа.

Методы:

clone()	Создаёт копию объекта стиля
draw(geometry, painter)	Рисует геометрический объект с текущим стилем в произвольном контексте вывода.
to_mapinfo()	Возвращает строковое представление в формате MapBasic.

__init__(symbol: int = 35, color: QColor = Qt.red, pointSize: int = 8)

Создает экземпляр класса.

Параметры

- **symbol** – Номер символа, который будет отображен при отрисовке. Для создания невидимого символа используйте значение 31. Стандартный набор условных знаков включает символы от 31 до 67.
- **color** – Цвет символа
- **pointSize** – Целое число, размер символа в пунктах от 1 до 48.

clone() → Style

Создаёт копию объекта стиля

property color: QColor

Цвет символа.

static create_mi_compat(symbol: int = 35, color: QColor = Qt.red, pointSize: int = 8) → PointCompatStyle

Создание стиля в виде совместимого с MapInfo 3 PointCompatStyle.

Параметры

- **symbol** – Номер символа, который будет отображен при отрисовке. Для создания невидимого символа используйте значение 31. Стандартный набор условных знаков включает символы от 31 до 67.
- **color** – Цвет символа
- **pointSize** – Целое число, размер символа в пунктах от 1 до 48.

static create_mi_font(symbol: int = 36, color: QColor = Qt.red, size: int = 8, fontname: str = 'Axioma MI MapSymbols', fontstyle: int = 0, rotation: float = 0.0) → PointFontStyle

Создание стиля на базе шрифта True Type `PointFontStyle`.

Параметры

- **symbol** – Целое, имеющее значение 31 или больше, определяющее, какой используется символ из шрифтов TrueType. Для создания невидимого символа используйте значение 31.
- **color** – Цвет символа
- **size** – Целое число, размер символа в пунктах от 1 до 48;
- **fontname** – Строка с именем шрифта TrueType (например, значение по умолчание „Axioma MI MapSymbols“)
- **fontstyle** – Стиль дополнительного оформления, например, курсивный текст. Возможные параметры см. в таблице ниже. Для указания нескольких параметров их суммируют между собой.
- **rotation** – Угол поворота символа в градусах.

static create_mi_picture(filename: str, color: QColor = Qt.black, size: int = 12, customstyle: int = 0) → PointPictureStyle

Создание стиля с ссылкой на растровый файл `PointPictureStyle`.

Параметры

- **filename** – Наименование растрового файла. Строка до 31 символа длиной. Данный файл должен находится в каталоге CustSymb с ресурсами. Например, “Arrow.BMP”.
- **color** – Цвет символа.
- **size** – Размер символа в в пунктах от 1 до 48.
- **customstyle** – Задание дополнительных параметров стиля оформления.

draw(geometry: Geometry, painter: QPainter)

Рисует геометрический объект с текущим стилем в произвольном контексте вывода. Это может быть востребовано при желании отрисовать геометрию со стилем на форме или диалоге.

Параметры

- **geometry** – Геометрия. Должна соответствовать стилю. Т.е. если объект полигон, а стиль для рисования точечных объектов, то ничего нарисовано не будет.
- **painter** – Контекст вывода.

Список 150: Пример отрисовки в растре и сохранение результата в файле.

```
image = QImage(100, 100, QImage.Format_ARGB32_Premultiplied)
image.fill(Qt.white)
painter = QPainter(image)
point = Point(50, 50)
style = PointStyle.create_mi_font(42, Qt.red, 24)
style.draw(point, painter)
image.save(filename)
```

classmethod for_geometry(geom: Geometry) → Style

Возвращает стиль по умолчанию для переданного объекта.

Параметры

geom – Геометрический объект, для которого необходимо получить соответствующий ему стиль.

classmethod from_mapinfo(mapbasic_string: str) → Optional[Style]

Получает стиль из строки формата MapBasic.

Параметры

mapbasic_string – Строка в формате MapBasic.

```
style = Style.from_mapinfo("Pen (1, 2, 0) Brush (8, 255)")
```

property size: int

Размер символа в пунктах.

property symbol: int

Номер символа.

to_mapinfo() → str

Возвращает строковое представление в формате MapBasic.

```
print(style.to_mapinfo())
'''
>>> Pen (1, 2, 0) Brush (8, 255)
'''
```

22.13.4 PointFontStyle - Стиль на базе шрифта True Type

class axipy.**PointFontStyle**

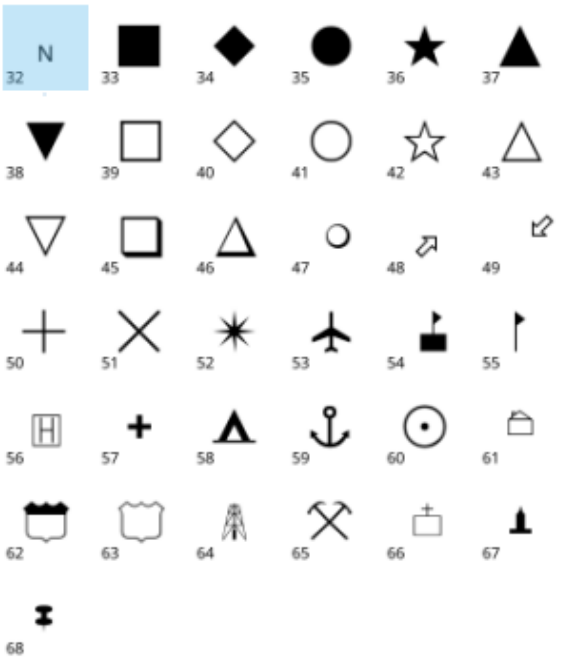
Базовые классы: PointStyle

Стиль на базе шрифта True Type.

Таблица 40: Возможные значения параметра fontstyle:

Значение	Наименование
0	Обычный текст
1	Жирный текст
16	Черная кайма вокруг символа
32	Тень
256	Белая кайма вокруг символа

В системе доступны следующие стили:



Список 151: Пример.

```
fs = PointFontStyle()
fs.font_name = 'Axioma MI Oil&Gas'
fs.has_shadow = True
fs.rotation = 30
fs.bold = True
fs.size = 44
fs.symbol = 45
fs.black_border = True
fs.white_border = True
```

Конструктор класса:

__init__([symbol, color, size, fontname, ...])	Создает экземпляр класса.

Классовые методы:

create_mi_compat([symbol, color, pointSize])	Создание стиля в виде совместимого с MapInfo 3 PointCompatStyle.
create_mi_font([symbol, color, size, ...])	Создание стиля на базе шрифта True Type PointFontStyle.
create_mi_picture(filename[, color, size, ...])	Создание стиля с ссылкой на растровый файл PointPictureStyle.
for_geometry(geom)	Возвращает стиль по умолчанию для переданного объекта.
from_mapinfo(mapbasic_string)	Получает стиль из строки формата MapBasic.

Свойства:

black_border	Темная окантовка.
bold	Жирный шрифт..
color	Цвет символа.
font_name	Наименование шрифта.
has_shadow	Признак тени.
rotation	Угол поворота.
size	Размер символа в пунктах.
symbol	Номер символа.
white_border	Светлая окантовка.

Методы:

clone()	Создаёт копию объекта стиля
draw(geometry, painter)	Рисует геометрический объект с текущим стилем в произвольном контексте вывода.
to_mapinfo()	Возвращает строковое представление в формате MapBasic.

__init__(symbol: int = 36, color: QColor = Qt.red, size: int = 8, fontname: str = 'Axioma MI MapSymbols', fontstyle: int = 0, rotation: float = 0.0)

Создает экземпляр класса.

Параметры

- **symbol** – Целое, имеющее значение 31 или больше, определяющее, какой используется символ из шрифтов TrueType. Для создания невидимого символа используйте значение 31.
- **color** – Цвет символа
- **size** – Целое число, размер символа в пунктах от 1 до 48;
- **fontname** – Строка с именем шрифта TrueType (например, значение по умолчание „Axioma MI MapSymbols“)
- **fontstyle** – Стиль дополнительного оформления, например, курсивный текст. Возможные параметры см. в таблице ниже. Для указания нескольких параметров их суммируют между собой.
- **rotation** – Угол поворота символа в градусах.

property black_border: bool

Темная окантовка.

property bold: bool

Жирный шрифт..

clone() → Style

Создаёт копию объекта стиля

property color: QColor

Цвет символа.

static create_mi_compat(symbol: int = 35, color: QColor = Qt.red, pointSize: int = 8) → PointCompatStyle

Создание стиля в виде совместимого с MapInfo 3 `PointCompatStyle`.

Параметры

- **symbol** – Номер символа, который будет отображен при отрисовке. Для создания невидимого символа используйте значение 31. Стандартный набор условных знаков включает символы от 31 до 67.
- **color** – Цвет символа
- **pointSize** – Целое число, размер символа в пунктах от 1 до 48.

static create_mi_font(symbol: int = 36, color: QColor = Qt.red, size: int = 8, fontname: str = 'Axioma MI MapSymbols', fontstyle: int = 0, rotation: float = 0.0) → PointFontStyle

Создание стиля на базе шрифта True Type `PointFontStyle`.

Параметры

- **symbol** – Целое, имеющее значение 31 или больше, определяющее, какой используется символ из шрифтов TrueType. Для создания невидимого символа используйте значение 31.
- **color** – Цвет символа
- **size** – Целое число, размер символа в пунктах от 1 до 48;
- **fontname** – Строка с именем шрифта TrueType (например, значение по умолчание „Axioma MI MapSymbols“)
- **fontstyle** – Стиль дополнительного оформления, например, курсивный текст. Возможные параметры см. в таблице ниже. Для указания нескольких параметров их суммируют между собой.
- **rotation** – Угол поворота символа в градусах.

static create_mi_picture(filename: str, color: QColor = Qt.black, size: int = 12, customstyle: int = 0) → PointPictureStyle

Создание стиля с ссылкой на растровый файл `PointPictureStyle`.

Параметры

- **filename** – Наименование растрового файла. Строка до 31 символа длиной. Данный файл должен находится в каталоге CustSymb с ресурсами. Например, “Arrow.BMP”.
- **color** – Цвет символа.
- **size** – Размер символа в в пунктах от 1 до 48.
- **customstyle** – Задание дополнительных параметров стиля оформления.

draw(geometry: Geometry, painter: QPainter)

Рисует геометрический объект с текущим стилем в произвольном контексте вывода. Это может быть востребовано при желании отрисовать геометрию со стилем на форме или диалоге.

Параметры

- **geometry** – Геометрия. Должна соответствовать стилю. Т.е. если объект полигон, а стиль для рисования точечных объектов, то ничего нарисовано не будет.
- **painter** – Контекст вывода.

Список 152: Пример отрисовки в растре и сохранение результата в файле.

```
image = QImage(100, 100, QImage.Format_ARGB32_Premultiplied)
image.fill(Qt.white)
painter = QPainter(image)
point = Point(50, 50)
style = PointStyle.create_mi_font(42, Qt.red, 24)
style.draw(point, painter)
image.save(filename)
```

property font_name: str

Наименование шрифта. Например, Adobe Helvetica

classmethod for_geometry(geom: Geometry) → Style

Возвращает стиль по умолчанию для переданного объекта.

Параметры
geom – Геометрический объект, для которого необходимо получить соответствующий ему стиль.

classmethod from_mapinfo(mapbasic_string: str) → Optional[Style]

Получает стиль из строки формата MapBasic.

Параметры
mapbasic_string – Строка в формате MapBasic.

```
style = Style.from_mapinfo("Pen (1, 2, 0) Brush (8, 255)")
```

property has_shadow: bool

Признак тени.

property rotation: float

Угол поворота.

property size: int

Размер символа в пунктах.

property symbol: int

Номер символа.

to_mapinfo() → str

Возвращает строковое представление в формате MapBasic.

```
print(style.to_mapinfo())
'''
>>> Pen (1, 2, 0) Brush (8, 255)
'''
```

property white_border: bool

Светлая окантовка.

22.13.5 PointPictureStyle - Стиль со ссылкой на растровый файл

class axipy.**PointPictureStyle**

Базовые классы: PointStyle

Стиль со ссылкой на растровый файл.

Параметры

- **filename** – Наименование растрового файла. Строка до 31 символа длиной. Данный файл должен находится в каталоге CustSymb с ресурсами. Например, “Arrow.BMP”.
- **color** – Цвет символа.
- **size** – Размер символа в в пунктах от 1 до 48.
- **customstyle** – Задание дополнительных параметров стиля оформления.

Таблица 41: Возможные значения параметра customstyle:

Значение	Наименование
0	**Флажки Фон и Покрасить одним цветом не установлены. Символ показывается стандартно.** Все белые точки изображения становятся прозрачными и под ними видны объекты Карты.
1	Установлен флажок Фон; все белые точки изображения становятся непрозрачными.
2	Установлен флажок Покрасить одним цветом все не белые точки изображения красятся в цвет символа.
3	Установлены флажки Фон и Покрасить одним цветом.

Список 153: Пример.

```
fs = PointPictureStyle('AMBU1-32.bmp')
fs.color = Qt.red
fs.apply_color = True
fs.actual_size = True
fs.show_background = True
```

Конструктор класса:

__init__(filename[, color, size, customstyle])	Создает экземпляр класса.

Классовые методы:

create_mi_compat([symbol, color, pointSize])	Создание стиля в виде совместимого с MapInfo 3 PointCompatStyle.
create_mi_font([symbol, color, size, ...])	Создание стиля на базе шрифта True Type PointFontStyle.
create_mi_picture(filename[, color, size, ...])	Создание стиля с ссылкой на растровый файл PointPictureStyle.
for_geometry(geom)	Возвращает стиль по умолчанию для переданного объекта.
from_mapinfo(mapbasic_string)	Получает стиль из строки формата MapBasic.

Свойства:

actual_size	Реальный размер.
apply_color	Применить цвет.
color	Цвет символа.
filename	Наименование файла растра.
show_background	Непрозрачный фон.
size	Размер символа в пунктах.

Методы:

clone()	Создаёт копию объекта стиля
draw(geometry, painter)	Рисует геометрический объект с текущим стилем в произвольном контексте вывода.
to_mapinfo()	Возвращает строковое представление в формате MapBasic.

__init__(filename: str, color: QColor = Qt.black, size: int = 12, customstyle: int = 0)

Создает экземпляр класса.

property actual_size: bool

Реальный размер.

property apply_color: bool

Применить цвет.

clone() → Style

Создаёт копию объекта стиля

property color: QColor

Цвет символа.

static create_mi_compat(symbol: int = 35, color: QColor = Qt.red, pointSize: int = 8) → PointCompatStyle

Создание стиля в виде совместимого с MapInfo 3 PointCompatStyle.

Параметры

- **symbol** – Номер символа, который будет отображен при отрисовке. Для создания невидимого символа используйте значение 31. Стандартный набор условных знаков включает символы от 31 до 67.

- **color** – Цвет символа
- **pointSize** – Целое число, размер символа в пунктах от 1 до 48.

static create_mi_font(symbol: int = 36, color: QColor = Qt.red, size: int = 8, fontname: str = 'Axioma MI MapSymbols', fontstyle: int = 0, rotation: float = 0.0) → PointFontStyle

Создание стиля на базе шрифта True Type `PointFontStyle`.

Параметры

- **symbol** – Целое, имеющее значение 31 или больше, определяющее, какой используется символ из шрифтов TrueType. Для создания невидимого символа используйте значение 31.
- **color** – Цвет символа
- **size** – Целое число, размер символа в пунктах от 1 до 48;
- **fontname** – Строка с именем шрифта TrueType (например, значение по умолчание „Axioma MI MapSymbols“)
- **fontstyle** – Стиль дополнительного оформления, например, курсивный текст. Возможные параметры см. в таблице ниже. Для указания нескольких параметров их суммируют между собой.
- **rotation** – Угол поворота символа в градусах.

static create_mi_picture(filename: str, color: QColor = Qt.black, size: int = 12, customstyle: int = 0) → PointPictureStyle

Создание стиля с ссылкой на растровый файл `PointPictureStyle`.

Параметры

- **filename** – Наименование растрового файла. Строка до 31 символа длиной. Данный файл должен находится в каталоге CustSymb с ресурсами. Например, “Arrow.BMP”.
- **color** – Цвет символа.
- **size** – Размер символа в в пунктах от 1 до 48.
- **customstyle** – Задание дополнительных параметров стиля оформления.

draw(geometry: Geometry, painter: QPainter)

Рисует геометрический объект с текущим стилем в произвольном контексте вывода. Это может быть востребовано при желании отрисовать геометрию со стилем на форме или диалоге.

Параметры

- **geometry** – Геометрия. Должна соответствовать стилю. Т.е. если объект полигон, а стиль для рисования точечных объектов, то ничего нарисовано не будет.
- **painter** – Контекст вывода.

Список 154: Пример отрисовки в растре и сохранение результата в файле.

```
image = QImage(100, 100, QImage.Format_ARGB32_Premultiplied)
image.fill(Qt.white)
painter = QPainter(image)
point = Point(50, 50)
style = PointStyle.create_mi_font(42, Qt.red, 24)
style.draw(point, painter)
image.save(filename)
```

property filename: str

Наименование файла растра.

classmethod for_geometry(geom: Geometry) → Style

Возвращает стиль по умолчанию для переданного объекта.

Параметры

geom – Геометрический объект, для которого необходимо получить соответствующий ему стиль.

classmethod from_mapinfo(mapbasic_string: str) → Optional[Style]

Получает стиль из строки формата MapBasic.

Параметры

mapbasic_string – Строка в формате MapBasic.

```
style = Style.from_mapinfo("Pen (1, 2, 0) Brush (8, 255)")
```

property show_background: bool

Непрозрачный фон.

property size: int

Размер символа в пунктах.

to_mapinfo() → str

Возвращает строковое представление в формате MapBasic.

```
print(style.to_mapinfo())
'''
>>> Pen (1, 2, 0) Brush (8, 255)
'''
```

22.13.6 LineStyle - Стиль линий

class axipy.**LineStyle**

Базовые классы: Style

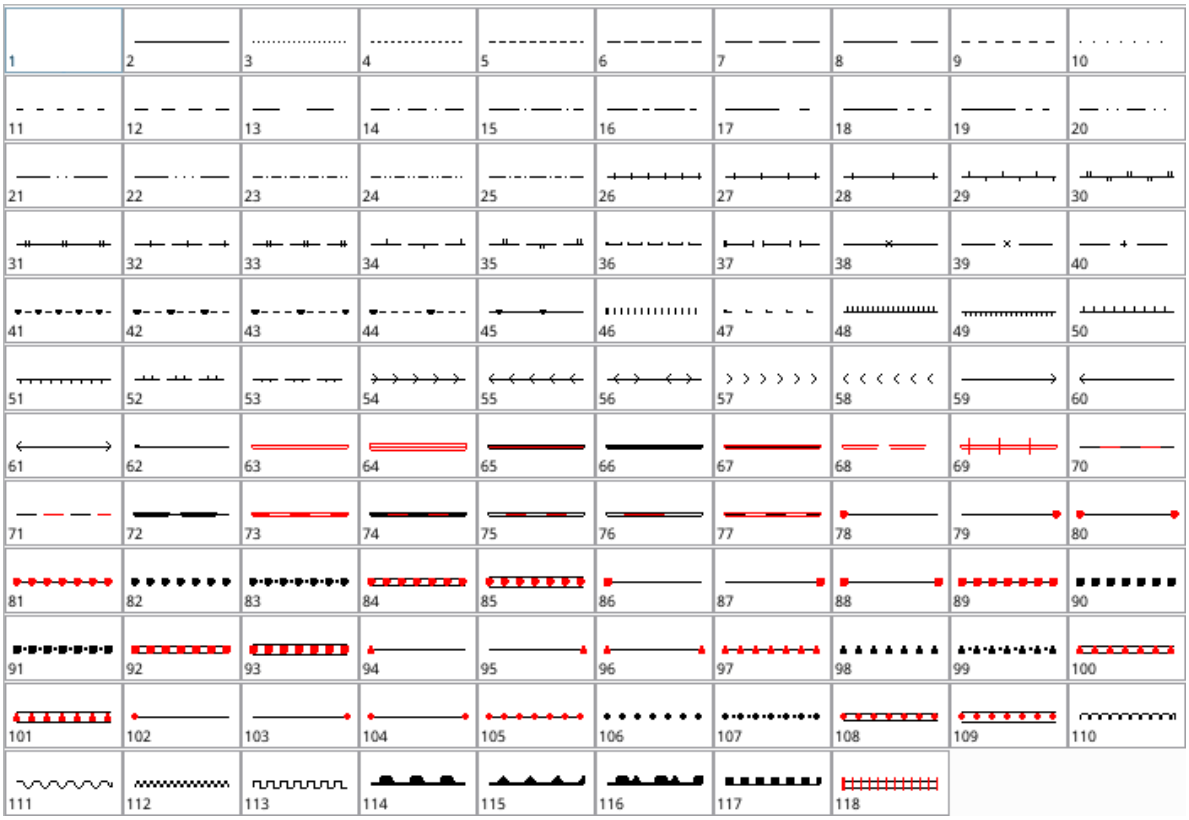
Стиль линейного объекта, совместимый с MapInfo.

Параметры

- **pattern** – Тип линии. Типы линий обозначаются кодами от 1 до 118. Тип 1 представляет собой невидимую линию.
- **color** – Цвет линии

- **width** – Толщина линии. Задается числом от 0 до 7, при этом линия нулевой ширины невидима на экране. 11-2047 - это значения, которые могут быть преобразованы в пункты: ширина линии = (число пунктов * 10) + 10 Значение 0 допустимо только для типа линии 1 или невидимых линий.

В системе доступны следующие стили линии:



Список 155: Пример.

```
style = LineStyle(3, Qt.red)
```

Конструктор класса:

__init__([pattern, color, width])	Создает экземпляр класса.

Классовые методы:

for_geometry(geom)	Возвращает стиль по умолчанию для переданного объекта.
from_mapinfo(mapbasic_string)	Получает стиль из строки формата MapBasic.

Свойства:

color	Цвет линии.
pattern	Номер стиля линии.
width	Толщина линии.

Методы:

clone()	Создаёт копию объекта стиля
draw(geometry, painter)	Рисует геометрический объект с текущим стилем в произвольном контексте вывода.
to_mapinfo()	Возвращает строковое представление в формате MapBasic.

__init__(pattern: int = 2, color: QColor = Qt.black, width: int = 1)

Создает экземпляр класса.

clone() → Style

Создаёт копию объекта стиля

property color: QColor

Цвет линии.

draw(geometry: Geometry, painter: QPainter)

Рисует геометрический объект с текущим стилем в произвольном контексте вывода. Это может быть востребовано при желании отрисовать геометрию со стилем на форме или диалоге.

Параметры

- **geometry** – Геометрия. Должна соответствовать стилю. Т.е. если объект полигон, а стиль для рисования точечных объектов, то ничего нарисовано не будет.
- **painter** – Контекст вывода.

Список 156: Пример отрисовки в растре и сохранение результата в файле.

```
image = QImage(100, 100, QImage.Format_ARGB32_Premultiplied)
image.fill(Qt.white)
painter = QPainter(image)
point = Point(50, 50)
style = PointStyle.create_mi_font(42, Qt.red, 24)
style.draw(point, painter)
image.save(filename)
```

classmethod for_geometry(geom: Geometry) → Style

Возвращает стиль по умолчанию для переданного объекта.

Параметры

geom – Геометрический объект, для которого необходимо получить соответствующий ему стиль.

classmethod from_mapinfo(mapbasic_string: str) → Optional[Style]

Получает стиль из строки формата MapBasic.

Параметры

mapbasic_string – Строка в формате MapBasic.

```
style = Style.from_mapinfo("Pen (1, 2, 0) Brush (8, 255)")
```

property pattern: int

Номер стиля линии.

to_mapinfo() → str

Возвращает строковое представление в формате MapBasic.

```
print(style.to_mapinfo())
'''
>>> Pen (1, 2, 0) Brush (8, 255)
'''
```

property width: int

Толщина линии.

22.13.7 FillStyle - Стиль заливки полигона

class axipy.**FillStyle**

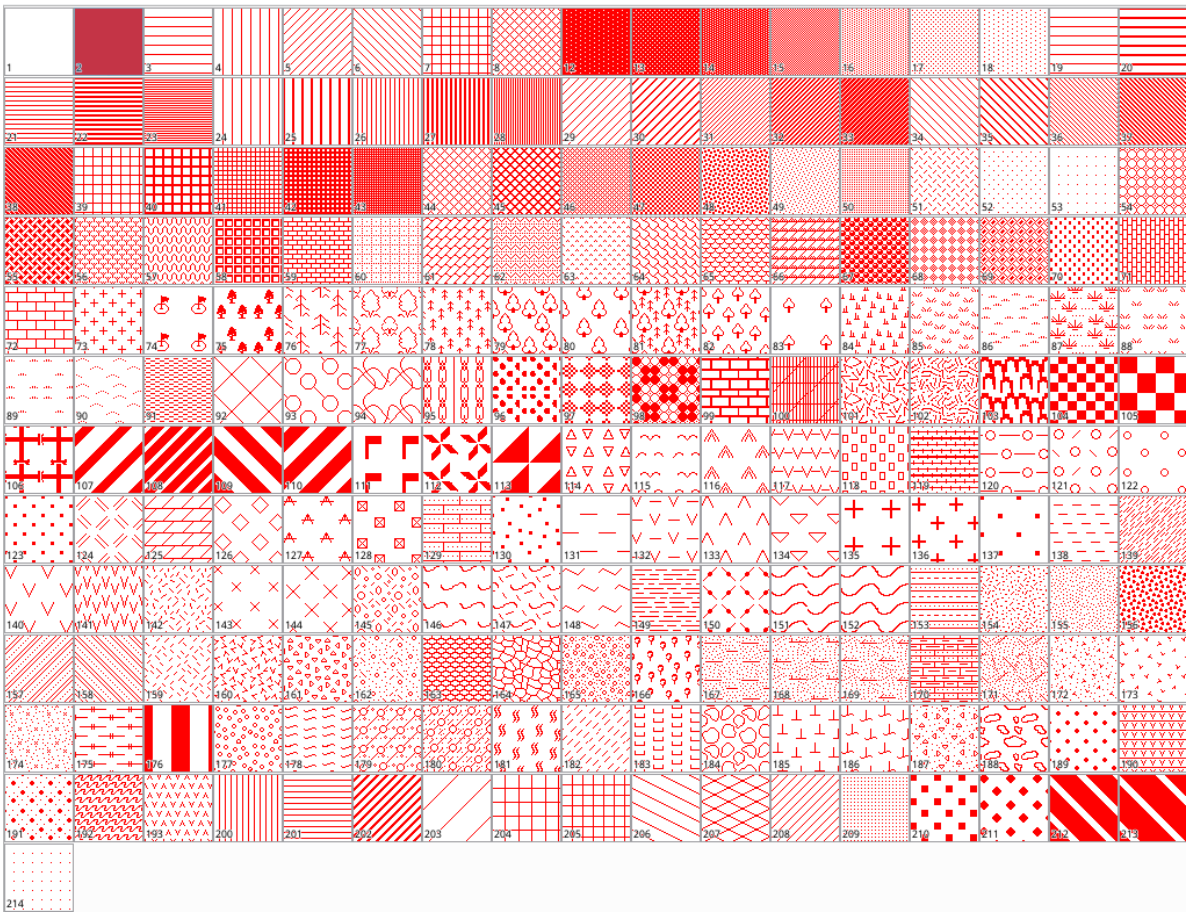
Базовые классы: Style

Стиль заливки полигонов PolygonStyle.

Параметры

- **pattern** – Номер стиля заливки.
- **color** – Цвет основной заливки.

В системе доступны следующие стили заливки:



Конструктор класса:

`__init__`([pattern, color])	Создает экземпляр класса.

Классовые методы:

`for_geometry`(geom)	Возвращает стиль по умолчанию для переданного объекта.
`from_mapinfo`(mapbasic_string)	Получает стиль из строки формата MapBasic.

Свойства:

`bg_color`	Цвет фона.
`color`	Цвет линии.
`pattern`	Номер стиля заливки.

Методы:

clone()	Создаёт копию объекта стиля
draw(geometry, painter)	Рисует геометрический объект с текущим стилем в произвольном контексте вывода.
to_mapinfo()	Возвращает строковое представление в формате MapBasic.

__init__(pattern: int = 1, color: QColor = Qt.transparent)

Создает экземпляр класса.

property bg_color: QColor

Цвет фона.

clone() → Style

Создаёт копию объекта стиля

property color: QColor

Цвет линии.

draw(geometry: Geometry, painter: QPainter)

Рисует геометрический объект с текущим стилем в произвольном контексте вывода. Это может быть востребовано при желании отрисовать геометрию со стилем на форме или диалоге.

Параметры

- **geometry** – Геометрия. Должна соответствовать стилю. Т.е. если объект полигон, а стиль для рисования точечных объектов, то ничего нарисовано не будет.
- **painter** – Контекст вывода.

Список 157: Пример отрисовки в растре и сохранение результата в файле.

```
image = QImage(100, 100, QImage.Format_ARGB32_Premultiplied)
image.fill(Qt.white)
painter = QPainter(image)
point = Point(50, 50)
style = PointStyle.create_mi_font(42, Qt.red, 24)
style.draw(point, painter)
image.save(filename)
```

classmethod for_geometry(geom: Geometry) → Style

Возвращает стиль по умолчанию для переданного объекта.

Параметры

geom – Геометрический объект, для которого необходимо получить соответствующий ему стиль.

classmethod from_mapinfo(mapbasic_string: str) → Optional[Style]

Получает стиль из строки формата MapBasic.

Параметры

mapbasic_string – Строка в формате MapBasic.

```
style = Style.from_mapinfo("Pen (1, 2, 0) Brush (8, 255)")
```

property pattern: int

Номер стиля заливки.

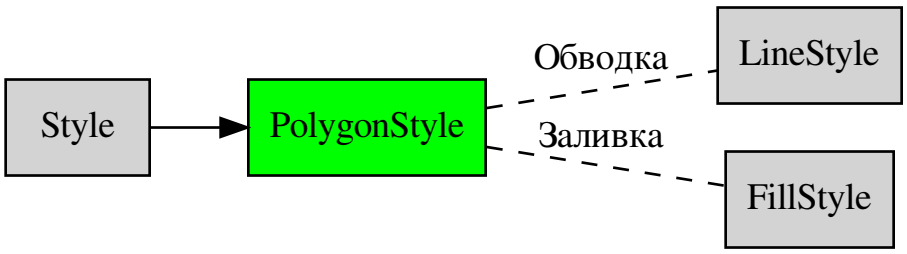
to_mapinfo() → str

Возвращает строковое представление в формате MapBasic.

```
print(style.to_mapinfo())
'''
>>> Pen (1, 2, 0) Brush (8, 255)
'''
```

22.13.8 PolygonStyle - Стиль полигонов

Зависимости стиля площадного объекта:



class axipy.**PolygonStyle**

Базовые классы: Style

Стиль площадного объекта. По умолчанию создается прозрачный стиль FillStyle с черной окантовкой LineStyle.

Список 158: Пример.

```
# Создадим стиль по умолчанию
plstyle = PolygonStyle()
print(plstyle.to_mapinfo())
# Назначим цвет заливки и фона заливки
plstyle.set_brush(color=Qt.green, bgColor=Qt.blue)
print(plstyle.to_mapinfo())
# Установим обводку
plstyle.set_pen(color=Qt.black)
print(plstyle.to_mapinfo())
# Установим параметры заливки через свойства
plstyle.fill.pattern = 5
plstyle.fill.color = Qt.blue
# Установим параметры обводки через свойства
```

(continues on next page)

(продолжение с предыдущей страницы)

```
plstyle.border.width = 4
plstyle.border.color = Qt.blue
'''
>>> Brush (1, 16777215)
>>> Brush (1, 65280, 255)
>>> Pen (1, 2, 0)  Brush (1, 65280, 255)
'''
```

Параметры

- **pattern** – Номер стиля заливки.
- **color** – Цвет основной заливки.
- **pattern_pen** – Цвет обводки. По умолчанию обводка отсутствует.

Конструктор класса:

__init__([pattern, color, pattern_pen])	Создает экземпляр класса.

Классовые методы:

for_geometry(geom)	Возвращает стиль по умолчанию для переданного объекта.
from_mapinfo(mapbasic_string)	Получает стиль из строки формата MapBasic.

Свойства:

border	Стиль окантовки.
fill	Стиль заливки.

Методы:

clone()	Создаёт копию объекта стиля
draw(geometry, painter)	Рисует геометрический объект с текущим стилем в произвольном контексте вывода.
set_brush([pattern, color, bgColor])	Задание стиля заливки площадного объекта.
set_pen([pattern, color, width])	Задание стиля обводки.
to_mapinfo()	Возвращает строковое представление в формате MapBasic.

__init__(pattern: int = 1, color: QColor = Qt.white, pattern_pen: int = 1)

Создает экземпляр класса.

property border: LineStyle

Стиль окантовки. Может отсутствовать. Для переопределения можно также использовать set_pen().

clone() → Style

Создаёт копию объекта стиля

draw(geometry: Geometry, painter: QPainter)

Рисует геометрический объект с текущим стилем в произвольном контексте вывода. Это может быть востребовано при желании отрисовать геометрию со стилем на форме или диалоге.

Параметры

- **geometry** – Геометрия. Должна соответствовать стилю. Т.е. если объект полигон, а стиль для рисования точечных объектов, то ничего нарисовано не будет.
- **painter** – Контекст вывода.

Список 159: Пример отрисовки в растре и сохранение результата в файле.

```
image = QImage(100, 100, QImage.Format_ARGB32_Premultiplied)
image.fill(Qt.white)
painter = QPainter(image)
point = Point(50, 50)
style = PointStyle.create_mi_font(42, Qt.red, 24)
style.draw(point, painter)
image.save(filename)
```

property fill: FillStyle

Стиль заливки.

classmethod for_geometry(geom: Geometry) → Style

Возвращает стиль по умолчанию для переданного объекта.

Параметры
geom – Геометрический объект, для которого необходимо получить соответствующий ему стиль.

classmethod from_mapinfo(mapbasic_string: str) → Optional[Style]

Получает стиль из строки формата MapBasic.

Параметры
mapbasic_string – Строка в формате MapBasic.

```
style = Style.from_mapinfo("Pen (1, 2, 0) Brush (8, 255)")
```

set_brush(pattern: int = 1, color: QColor = Qt.white, bgColor: QColor = Qt.transparent)

Задание стиля заливки площадного объекта.

Параметры

- **pattern** – Номер стиля заливки. Шаблон задается числом от 1 до 71, при этом в шаблоне с номером 1 оба цвета отсутствуют, а в шаблоне 2 отсутствует цвет фона. Шаблоны с кодами 9-11 зарезервированы для внутренних целей.
- **color** – Цвет основной заливки.
- **bgColor** – Цвет заднего фона, если заливка неполная.

Доступные стили заливки см. FillStyle:

set_pen(pattern: int = 2, color: QColor = Qt.black, width: int = 1)

Задание стиля обводки. Параметры аналогичны при задании стиля линии LineStyle()

Параметры

- **pattern** – Номер стиля линии.
- **color** – Цвет линии
- **width** – Толщина линии.

to_mapinfo() → str

Возвращает строковое представление в формате MapBasic.

```
print(style.to_mapinfo())
'''
>>> Pen (1, 2, 0) Brush (8, 255)
'''
```

22.13.9 TextStyle - Стиль текста

class axipy.**TextStyle**

Базовые классы: Style

Стиль текстового объекта.

Таблица 42: Возможные значения параметра style

Значение	Наименование
0	Обычный
1	Жирный
2	Курсив
4	Подчеркнутый
16	Контур (только для Macintosh)
32	Тень
256	Кайма
512	Капитель
1024	Разрядка

Список 160: Пример.

```
style = TextStyle("Droid Sans", 24)
style.fontname = 'Adobe Helvetica'
style.color = Qt.red
style.bold = True
style.italic = True
style.shadow = True
style.spacing = True
style.capital = True
style.alignment = TextAlignment.Center
print(style.size)
style.size = 22
```

(continues on next page)

(продолжение с предыдущей страницы)

```
style.callout = TextCallout.Arrow
arrow_style = LineStyle(2, Qt.red)
style.callout_style = arrow_style
print(style.to_mapinfo())
'''
>>> Font ("Adobe Helvetica", 1571, 22, 16711680) Label Line Arrow Line (1, 2,␣
↪16711680)
'''
```

Примечание: При назначении стиля для текста необходимо помнить, что его параметры и параметры геометрии `axipy.mi.Text` взаимозависимы.

Конструктор класса:

`__init__`(fontname, size[, style, forecolor, ...])	Создает экземпляр класса.

Классовые методы:

`for_geometry`(geom)	Возвращает стиль по умолчанию для переданного объекта.
`from_mapinfo`(mapbasic_string)	Получает стиль из строки формата MapBasic.

Свойства:

`alignment`	Выравнивание текста.
`bg_color`	Цвет фона текста
`bg_type`	Тип отрисовки фона текста.
`bold`	Жирный текст.
`callout`	Тип выноски.
`callout_style`	Стиль выноски.
`color`	Цвет текста
`fontname`	Наименование шрифта
`italic`	Курсив текста.
`shadow`	Тень.
`size`	Базовый размер шрифта в пунктах.
`spacing`	Разрядка

Методы:

`clone`()	Создаёт копию объекта стиля
`draw`(geometry, painter)	Рисует геометрический объект с текущим стилем в произвольном контексте вывода.
`to_mapinfo`()	Возвращает строковое представление в формате MapBasic.

__init__(fontname: str, size: int, style: int = 0, forecolor: QColor = Qt.black, backcolor: QColor = Qt.transparent)

Создает экземпляр класса.

Параметры

- **fontname** – Наименование шрифта.
- **size** – Размер шрифта в пунктах. Может принимать значение 0 для подписей в окне карты, так как они являются атрибутами карты и их размер определяется динамически.
- **style** – Дополнительные параметры стиля. Подробнее см. в таблице ниже. Стоит заметить, что если оставить значение, равным 0, то необходимые свойства можно установить позже через соответствующие свойства.
- **forecolor** – Цвет шрифта.
- **backcolor** – Цвет заднего фона, если он задан.

property alignment: TextAlignment

Выравнивание текста.

property bg_color: QColor

Цвет фона текста

property bg_type: TextBackgroundType

Тип отрисовки фона текста.

property bold: bool

Жирный текст.

property callout: TextCallout

Тип выноски.

property callout_style: LineStyle

Стиль выноски.

clone() → Style

Создаёт копию объекта стиля

property color: QColor

Цвет текста

draw(geometry: Geometry, painter: QPainter)

Рисует геометрический объект с текущим стилем в произвольном контексте вывода. Это может быть востребовано при желании отрисовать геометрию со стилем на форме или диалоге.

Параметры

- **geometry** – Геометрия. Должна соответствовать стилю. Т.е. если объект полигон, а стиль для рисования точечных объектов, то ничего нарисовано не будет.
- **painter** – Контекст вывода.

Список 161: Пример отрисовки в растре и сохранение результата в файле.

```
image = QImage(100, 100, QImage.Format_ARGB32_Premultiplied)
image.fill(Qt.white)
painter = QPainter(image)
point = Point(50, 50)
style = PointStyle.create_mi_font(42, Qt.red, 24)
style.draw(point, painter)
image.save(filename)
```

property fontname: str

Наименование шрифта

classmethod for_geometry(geom: Geometry) → Style

Возвращает стиль по умолчанию для переданного объекта.

Параметры

geom – Геометрический объект, для которого необходимо получить соответствующий ему стиль.

classmethod from_mapinfo(mapbasic_string: str) → Optional[Style]

Получает стиль из строки формата MapBasic.

Параметры

mapbasic_string – Строка в формате MapBasic.

```
style = Style.from_mapinfo("Pen (1, 2, 0) Brush (8, 255)")
```

property italic: bool

Курсив текста.

property shadow: bool

Тень.

property size: float

Базовый размер шрифта в пунктах. Следует отметить, что точный размер шрифта высчитывается исходя из контекста рисования (карта или отчет).

property spacing: bool

Разрядка

to_mapinfo() → str

Возвращает строковое представление в формате MapBasic.

```
print(style.to_mapinfo())
'''
>>> Pen (1, 2, 0) Brush (8, 255)
'''
```

22.13.10 TextCallout - Тип выноски

class axipy.**TextCallout**

Тип выноски.

Таблица 43: Значения

Значение	Наименование
NoCallout	Не отображать
Line	Линия
Arrow	Стрелка

22.13.11 TextAlignment - Выравнивание текста

class axipy.**TextAlignment**

Выравнивание текста.

Таблица 44: Значения

Значение	Наименование
Left	По левому краю
Center	По центру
Right	По правому краю

22.13.12 TextBackgroundType - Тип отрисовки фона

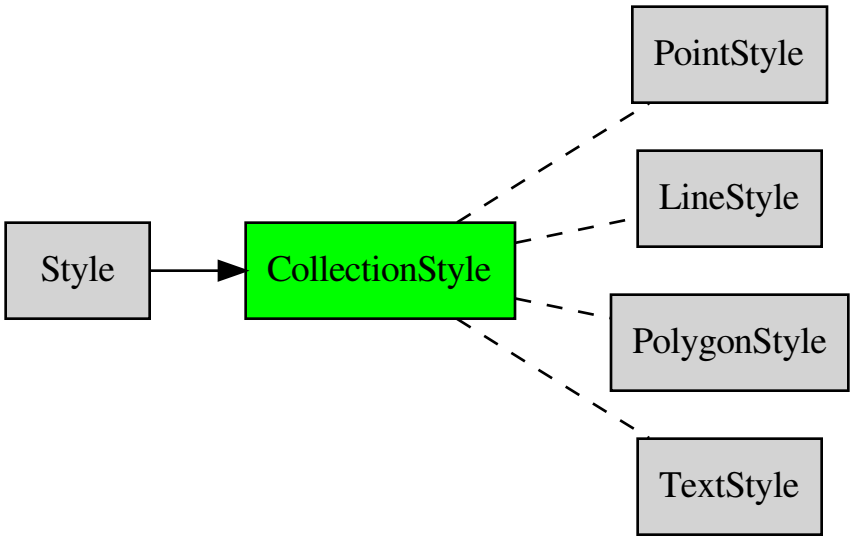
class axipy.**TextBackgroundType**

Тип отрисовки фона текстового объекта.

Таблица 45: Значения

Значение	Наименование
NoBackground	Фон отсутствует
Outline	Обводка по контору букв текста
Frame	Рамка вокруг текста

22.13.13 CollectionStyle - Стиль коллекций



class axipy.**CollectionStyle**

Базовые классы: `Style`

Смешанный стиль для разнородного типа объектов.

Данный стиль представляет собой контейнер стилей. может применяться в купе с геометрическим объектом типа разнородная коллекция `axipy.GeometryCollection`. Для задания или переопределения стилей простейших объектов, необходимо вызывать соответствующие методы для необходимых типов объектов.

Примечание: Объекты стилей, полученные через методы `line()`, `polygon()` и т.д. будут удалены сразу же после удаления объекта стиля коллекции. Если их нужно сохранить, воспользуйтесь операцией `clone()`.

Классовые методы:

`for_geometry`(geom)	Возвращает стиль по умолчанию для переданного объекта.
`from_mapinfo`(mapbasic_string)	Получает стиль из строки формата MapBasic.

Свойства:

line	Стиль для линейных объектов LineStyle.
point	Стиль для точечных объектов PointStyle.
polygon	Стиль для полигональных объектов PolygonStyle.
text	Стиль для текстовых объектов TextStyle.

Методы:

clone()	Создаёт копию объекта стиля
draw(geometry, painter)	Рисует геометрический объект с текущим стилем в произвольном контексте вывода.
find_style(geom)	Возвращает стиль, подходящий для переданной геометрии.
for_line(style)	Задание стиля для линейных объектов LineStyle.
for_point(style)	Задание стиля для точечных объектов PointStyle.
for_polygon(style)	Задание стиля для полигональных объектов PolygonStyle.
for_text(style)	Задание стиля для текстовых объектов TextStyle.
to_mapinfo()	Возвращает строковое представление в формате MapBasic.

clone() → Style

Создаёт копию объекта стиля

draw(geometry: Geometry, painter: QPainter)

Рисует геометрический объект с текущим стилем в произвольном контексте вывода. Это может быть востребовано при желании отрисовать геометрию со стилем на форме или диалоге.

Параметры

- **geometry** – Геометрия. Должна соответствовать стилю. Т.е. если объект полигон, а стиль для рисования точечных объектов, то ничего нарисовано не будет.
- **painter** – Контекст вывода.

Список 162: Пример отрисовки в растре и сохранение результата в файле.

```
image = QImage(100, 100, QImage.Format_ARGB32_Premultiplied)
image.fill(Qt.white)
painter = QPainter(image)
point = Point(50, 50)
style = PointStyle.create_mi_font(42, Qt.red, 24)
style.draw(point, painter)
image.save(filename)
```

find_style(geom: Geometry) → Style

Возвращает стиль, подходящий для переданной геометрии.

classmethod for_geometry(geom: Geometry) → Style

Возвращает стиль по умолчанию для переданного объекта.

Параметры

geom – Геометрический объект, для которого необходимо получить соответствующий ему стиль.

for_line(style: LineStyle)

Задание стиля для линейных объектов LineStyle.

for_point(style: PointStyle)

Задание стиля для точечных объектов PointStyle.

for_polygon(style: PolygonStyle)

Задание стиля для полигональных объектов PolygonStyle.

for_text(style: TextStyle)

Задание стиля для текстовых объектов TextStyle.

classmethod from_mapinfo(mapbasic_string: str) → Optional[Style]

Получает стиль из строки формата MapBasic.

Параметры

mapbasic_string – Строка в формате MapBasic.

```
style = Style.from_mapinfo("Pen (1, 2, 0) Brush (8, 255)")
```

property line: Optional[LineStyle]

Стиль для линейных объектов LineStyle.

property point: Optional[PointStyle]

Стиль для точечных объектов PointStyle.

property polygon: Optional[PolygonStyle]

Стиль для полигональных объектов PolygonStyle.

property text: Optional[TextStyle]

Стиль для текстовых объектов TextStyle.

to_mapinfo() → str

Возвращает строковое представление в формате MapBasic.

```
print(style.to_mapinfo())
'''
>>> Pen (1, 2, 0) Brush (8, 255)
'''
```

22.14 TypeSqlDialect - Диалект при выполнении запросов

class axipy.**TypeSqlDialect**

Используемый диалект при выполнении SQL запросов. Есть отличия как в скорости, так и в функционале. выбор зависит от конкретной задачи. Значение, установленное по умолчанию можно получить посредством свойства `data_manager.sql_dialect`

Таблица 46: Значения

Значение	Наименование
axioma	Использовать собственную реализацию инструмента выполнения запросов
sqlite	Использовать при выполнении реализацию, являющийся надстройкой над sqlite

22.15 Raster - Растр

class axipy.**Raster**

Базовые классы: DataObject

Растровый объект.

Свойства:

coordsystem	Система координат растра.
destroyed	Сигнал оповещения об удалении объекта.
device_to_scene_transform	Матрица преобразования из точек на изображении (пиксели) в точки на карте.
is_spatial	Признак того, что объект данных является пространственным.
name	Название объекта данных.
properties	Дополнительные свойства объекта данных.
provider	Провайдер изначального источника данных.
scene_to_device_transform	Матрица преобразования из точек на карте в точки на изображении (пиксели).
size	Размер растра в пикселях.

Методы:

close()	Пытается закрыть таблицу.
get_gcps()	Возвращает точки привязки.

close()

Пытается закрыть таблицу.

Исключение
RuntimeError – Ошибка закрытия таблицы.

Примечание: Объект данных не всегда может быть сразу закрыт. Например, для таблиц используется транзакционная модель редактирования и перед закрытием необходимо сохранить или отменить изменения, если они есть. См. Table.is_modified.

---

property coordsystem: Optional[CoordSystem]

Система координат растра.

property destroyed: SignalInstance

Сигнал оповещения об удалении объекта.

property device_to_scene_transform: QTransform

Матрица преобразования из точек на изображении (пиксели) в точки на карте.

get_gcps() → List[GCP]

Возвращает точки привязки.

property is_spatial: bool

Признак того, что объект данных является пространственным.

property name: str

Название объекта данных.

property properties: dict

Дополнительные свойства объекта данных.

property provider: str

Провайдер изначального источника данных.

property scene_to_device_transform: QTransform

Матрица преобразования из точек на карте в точки на изображении (пиксели).

property size: QSize

Размер растра в пикселях.

22.16 raster - Операции с растром

Модуль операций с растрами.

class axipy.**GCP**

Точка привязки (Ground Control Point).

Атрибуты:

device	Точка на изображении (пиксели).
label	Идентификатор.
scene	Точка на карте.

device: Union[Tuple[float, float], QPointF]
Точка на изображении (пиксели).

label: str
Идентификатор.

scene: Union[Tuple[float, float], QPointF]
Точка на карте.

class axipy.**Algorithm**
Алгоритм трансформации.

Атрибуты:

POLYNOM1	Аффинитет
POLYNOM2	Полиномиальный второго порядка
POLYNOM3	Полиномиальный третьего порядка
SPLINE	Сплайновый

class axipy.**Resample**
Метод интерполяции.

Атрибуты:

Average	Средний
Bilinear	Билинейный
Cubic	Кубический
CubicSpline	Кубический сплайн
Lanczos	Ланцоша
Max	Максимальный
Med	Медианный
Min	Минимальный
Mode	Самый встречающийся
NearestNeighbour	Ближайший
Q1	Первый квартиль
Q3	Третий квартиль
Sum	Сумма

class axipy.**Format**
Формат изображения.

Атрибуты:

BMP
GTiff
JPEG
PNG

class axipy.**Compression**

Сжатие.

Примечание: Сжатие можно использовать только для файлов формата GeoTIFF.

Атрибуты:

DEFLATE
LZW
NONE
PACKBITS

axipy.**register**(filepath: str, bindings: Union[List[GCP], QTransform], coordsystem: CoordSystem, override: bool = False)

Регистрирует растр.

Добавляет изображению пространственную привязку.

Параметры

- **filepath** – Файл с изображением.
- **bindings** – Привязка в виде точек или матрицы преобразования.
- **coordsystem** – Координатная система.
- **override** – При регистрации формируется файл привязки TAB. Если он существует, то данный параметр управляет его перезаписью.

Список 163: Пример использования

```
from PySide2.QtGui import QTransform

matrix = QTransform()
coordsystem = CoordSystem.from_units(Unit.m)
register(imagefile, matrix, coordsystem)
```

axipy.**transform**(inputfile: str, outputfile: str, points: List[GCP], coordsystem: CoordSystem, algorithm: Algorithm = Algorithm.SPLINE, resample: Resample = Resample.NearestNeighbour, output_format: Format = Format.GTiff, compression: Compression = Compression.NONE)

Трансформирует растр.

Растру, имеющему пространственную привязку, задает новую привязку. На выходе получается новый растр.

Параметры

- **inputfile** – Входной файл с растром.
- **outputfile** – Выходной файл с растром.
- **points** – Точки привязки.

- **coordsystem** – Координатная система.
- **algorithm** – Алгоритм трансформации.
- **resample** – Метод интерполяции.
- **output_format** – Выходной формат.
- **compression** – Метод сжатия.

Список 164: Пример использования

```
coordsystem = CoordSystem.from_epsg(4326)
gcps = [
    GCP((0, 0), (0, 0)),
    GCP((200, 0), (30, 30)),
    GCP((200, 200), (60, 0)),
]
transform(rasterfile, outputfile, gcps, coordsystem)
```

22.17 ObserverManager - Менеджер наблюдателей

class axipy.**ObserverManager**

Наблюдатели за состоянием. Класс является статическим словарем, доступным только для чтения (collections.abc.Mapping). Поддерживает обращение по индексу.

Классовые методы:

create(name, init_value)	Создает наблюдатель.
get(key[, default_value])	Возвращает значение по ключу, где ключ это имя наблюдателя, а значение это объект класса axipy.Observer.
items()	Возвращает список кортежей ключ-значение, где ключи это имена наблюдателей, а значения это объекты класса axipy.Observer.
keys()	Возвращает список ключей, где ключи это имена наблюдателей.
remove(name)	Удаляет наблюдатель по имени.
values()	Возвращает список значений, где значения это объекты класса axipy.Observer.

Атрибуты:

ActiveMapView	Есть активное окно карты
ActiveTableView	Есть активное окно таблицы
ActiveView	Есть активное окно
Editable	Активная карта имеет редактируемый слой
HasTables	Открыта хотя бы одна таблица
Selection	Есть выборка
SelectionEditable	Карта имеет редактируемый слой и есть выделенные объекты на одном из слоев карты
SelectionEditableIsSame	Карта имеет редактируемый слой и выборку на этом слое

ActiveMapView: Observer

Есть активное окно карты

ActiveTableView: Observer

Есть активное окно таблицы

ActiveView: Observer

Есть активное окно

Editable: Observer

Активная карта имеет редактируемый слой

HasTables: Observer

Открыта хотя бы одна таблица

Selection: Observer

Есть выборка

SelectionEditable: Observer

Карта имеет редактируемый слой и есть выделенные объекты на одном из слоев карты

SelectionEditableIsSame: Observer

Карта имеет редактируемый слой и выборку на этом слое

classmethod create(name: str, init_value: Any) → Observer

Создает наблюдатель.

Параметры

- **name** – Имя наблюдателя.
- **init_value** – Начальное значение наблюдателя.

Результат

Наблюдатель.

classmethod get(key: str, default_value: Optional[Any] = None) → Optional[Observer]

Возвращает значение по ключу, где ключ это имя наблюдателя, а значение это объект класса axipy.Observer. Если ключа нет, возвращается значение по умолчанию.

classmethod items() → List[Tuple[str, Observer]]

Возвращает список кортежей ключ-значение, где ключи это имена наблюдателей, а значения это объекты класса axipy.Observer.

classmethod keys() → List[str]

Возвращает список ключей, где ключи это имена наблюдателей.

classmethod remove(name: str)

Удаляет наблюдатель по имени.

Параметры
name – Имя наблюдателя.

classmethod values() → List[Observer]

Возвращает список значений, где значения это объекты класса axipy.Observer.

22.18 Observer - Наблюдатель

class axipy.**Observer**

Наблюдатель

Свойства:

name	Возвращает имя наблюдателя.
value	Устанавливает или возвращает значение наблюдателя.

Сигналы:

changed	Сигнал об изменении значения.

property changed: Signal

Сигнал об изменении значения.

Тип результата
Signal[Any]

```
from axipy import ObserverManager

def print_func(value):
    print(value)

ObserverManager.Selection.changed.connect(print_func)
```

property name: str

Возвращает имя наблюдателя.

property value: Any

Устанавливает или возвращает значение наблюдателя. При изменении значения испускается сигнал changed.

22.19 TabFile - Файл TAB

class axipy.**TabFile**

Класс поддержки файла TAB формата MapInfo.

Методы:

generate_tab(data_object, out_file[, override])	Генерирует файл TAB для переданного открытого объекта, если такую возможность поддерживает провайдер данных.
suggest_tab_name(data_object)	Сервисная функция.

generate_tab(data_object: DataObject, out_file: str, override: bool = True) → bool

Генерирует файл TAB для переданного открытого объекта, если такую возможность поддерживает провайдер данных.

Параметры

- **data_object** – открытый объект данных, для которого необходимо создать файл TAB.
- **out_file** – Имя файла с расширением tab. Как вариант, можно использовать результат suggest_tab_name().
- **override** – Перезаписывать файл. Если установлено False и файл существует, будет выброшено исключение FileExistsError

Результат

Возвращает True, если успешно.

Создание TAB файла для открытой таблицы или растра:

```
filepath = 'world.tif'
out_file_name = 'world.tab'
filepath = generate_geometry_table()
tab = TabFile()
tab.generate_tab(table, out_file_name)
```

Создание TAB файла для открытого источника тайлового сервиса:

```
prj_mercator = 'Earth Projection 10, 104, "m", 0 Bounds (-20037508.34, -
↪20037508.34) (20037508.34, 20037508.34)'
osm_raster = provider_manager.tms.open('http://maps.axioma-gis.ru/osm/{LEVEL}/
↪{ROW}/{COL}.png', prj=prj_mercator)
tab = TabFile()
out_file_name = tab.suggest_tab_name(osm_raster)
tab.generate_tab(osm_raster, out_file_name)
```

suggest_tab_name(data_object: DataObject)

Сервисная функция. Предлагает наименование TAB файла для объекта данных. Результат можно использовать в методе generate_tab() в качестве имени выходного файла.

22.20 RasteredTable - Источники ГИС Панорама и AutoCAD.

class axipy.**RasteredTable**

Базовые классы: DataObject

Данные из таких источников, как ГИС Панорама и AutoCAD.

Свойства:

coordsystem	Система координат.
destroyed	Сигнал оповещения об удалении объекта.
is_spatial	Признак того, что объект данных является пространственным.
layers	Список доступных для запроса данных слоев.
name	Название объекта данных.
properties	Дополнительные свойства объекта данных.
provider	Провайдер изначального источника данных.
schema	Схема таблицы.

Методы:

close()	Пытается закрыть таблицу.
items([layer_name])	Запрашивает записи из источника.

close()

Пытается закрыть таблицу.

Исключение
RuntimeError – Ошибка закрытия таблицы.

Примечание: Объект данных не всегда может быть сразу закрыт. Например, для таблиц используется транзакционная модель редактирования и перед закрытием необходимо сохранить или отменить изменения, если они есть. См. Table.is_modified.

property coordsystem: Optional[CoordSystem]

Система координат.

property destroyed: SignalInstance

Сигнал оповещения об удалении объекта.

property is_spatial: bool

Признак того, что объект данных является пространственным.

items(layer_name: Optional[str] = None) → Iterator[Feature]

Запрашивает записи из источника.

Параметры

- **слоя** (layer_name Наименование) –
- **выборка.** (по ктоорому будет произведена) –
- **пустое** (Если значение) –
- **слоям.** (выдаются данные по всем) –

Результат
Итератор по записям.

property layers
Список доступных для запроса данных слоев.

property name: str
Название объекта данных.

property properties: dict
Дополнительные свойства объекта данных.

property provider: str
Провайдер изначального источника данных.

property schema: Schema
Схема таблицы.

23

axipy.render - Модуль отрисовки.

Модуль отрисовки.

Данный модуль содержит инструменты, предназначенные для отрисовки геопространственных и прочих данных.

23.1 Map - Карта

class axipy.**Map**

Класс карты. Рассматривается как группа слоев, объединенная в единую сущность. Вне зависимости от СК входящих в карту слоев, карта отображает все слои в одной СК. Найти наиболее подходящую для этого можно с помощью `get_best_coordsystem()` или же установить другую.

Единицы измерения расстояний `distanceUnit` и площадей `areaUnit` берутся из настроек по умолчанию.

Параметры

layers – Список слоев, с которым будет создана карта.

Исключение

ValueError – Если один и тот же слой был передан несколько раз.

Список 1: Пример.

```
table_world = provider_manager.openfile(filepath)
world = Layer.create(table_world)
map = Map([world])
print('СК:', map.get_best_coordsystem().prj)
print('Охват:', map.get_best_rect())
print('Единицы измерения расстояний:', map.distanceUnit.description)
map.distanceUnit = Unit.mi
print('Единицы измерения расстояний (изменено):', map.distanceUnit.description)
'''
>>> СК: Earth Projection 12, 62, "m", 0
>>> Охват: (-16194966.287183324 -8621185.324024437) (16789976.633236416 8326222.
↪646170927)
>>> Единицы измерения расстояний: километры
```

(continues on next page)

(продолжение с предыдущей страницы)

```
>>> Единицы измерения расстояний (изменено): мили
'''
```

Свойства:

areaUnit	Единицы измерения площадей карты.
cosmetic	Косметический слой карты.
custom_labels	Пользовательские метки
distanceUnit	Единицы измерения расстояний на карте.
editable_layer	Редактируемый слой для текущей карты.
layers	Список слоев и групп слоев.

Методы:

draw(context)	Рисует карту в контексте.
get_best_coordsystem()	Определяет координатную системы карты, наиболее подходящую исходя из содержимого перечня слоев.
get_best_rect([coordsystem])	Определяет ограничивающий прямоугольник карты.
to_image(width, height[, coordsystem, bbox])	Рисует карту в изображение.

Сигналы:

need_redraw	Сигнал о необходимости перерисовки карты.

property areaUnit: AreaUnit

Единицы измерения площадей карты.

property cosmetic: CosmeticLayer

Косметический слой карты.

property custom_labels: CustomLabels

Пользовательские метки

Список 2: Пример.

```
table_world = provider_manager.openfile(filepath)
world_layer = Layer.create(table_world)
map_ = Map([world_layer])
# Определим свойства
p = CustomLabelProperties()
# Переназначим выражение
p.expression = 'Новое выражение'
# Угол поворота
p.angle = 30
# Выноска будет в виде стрелки
p.endType = CustomLabelEndType.Arrow
```

(continues on next page)

(продолжение с предыдущей страницы)

```
# Положение выноски
p.position = Point(100, 200)
# Установим свойства
map_.custom_labels.set(map_.layers[0], 1, p)
# Запрос свойств
props = map_.custom_labels.get(map_.layers[0], 1)
```

property distanceUnit: LinearUnit

Единицы измерения расстояний на карте.

draw(context: Context)

Рисует карту в контексте.

Параметры
context – Контекст рисования.

Список 3: Пример.

```
# Пример получения карты как растра
map = Map([world])
image = QImage(1600, 800, QImage.Format_ARGB32_Premultiplied)
image.fill(Qt.white)
painter = QPainter(image)
context = Context(painter)
map.draw(context)
```

property editable_layer: VectorLayer

Редактируемый слой для текущей карты.

Исключение
ValueError – При попытке установить слой, не принадлежащий этой карте.

get_best_coordsystem() → CoordSystem

Определяет координатную системы карты, наиболее подходящую исходя из содержимого перечня слоев.

get_best_rect(coordsystem: Optional[CoordSystem] = None) → Rect

Определяет ограничивающий прямоугольник карты.

Параметры
coordsystem – Координатная система, в которой необходимо получить результат. Если отсутствует, будет выдан результат для наиболее подходящей координатной системы.

property layers: ListLayers

Список слоев и групп слоев.

Примечание: Не содержит косметический слой cosmetic.

Список 4: Примеры доступа.

```
# Создадим карту с тремя слоями
map = Map([world, worldcap, russia])
```

(continues on next page)

(продолжение с предыдущей страницы)

```
print(len(map.layers))
'''
>>> 3
'''
print(map.layers[0].title)
'''
>>> world
'''
# Группировка первый двух слоев с именем "Мир"
map.layers.group([0, 1], 'Мир')
# Перечень элементов
for l in map.layers:
    if isinstance(l, Layer):
        print('Слой:', l.title)
    elif isinstance(l, ListLayers):
        print('Группа:', l.title)
'''
>>> Группа: Мир
>>> Слой: russia
'''
# Изменение позиции
map.layers.move(0, 1)
# Управление видимостью группы слоев
map.layers[1].visible = False
# Разгруппировка ранее созданной группы
map.layers.ungroup(1)
# Удаление слоя из карты
map.layers.remove(1)
# Добавление пустой группы
map.layers.add_group('Новая группа')
```

property need_redraw: Signal

Сигнал о необходимости перерисовки карты. Возникает при изменении контента одного или нескольких слоев карты. Это может быть обусловлено изменением данных таблиц.

Тип результата
Signal[]

Список 5: Пример.

```
# Смотрим активное окно.
if isinstance(view_manager.active, MapView):
    # Если это карта, подключимся к событию обновления окна этой карты.
    map_view = view_manager.active
    map_view.map.need_redraw.connect(lambda: print('Update map'))
```

to_image(width: int, height: int, coordsystem: Optional[CoordSystem] = None, bbox: Optional[Rect] = None) → QImage

Рисует карту в изображение.

Параметры

- **width** – Ширина выходного изображения.
- **height** – Высота выходного изображения.
- **coordsystem** – Координатная система. Если не задана, берется

наиболее подходящая.

- **bbox** – Ограничивающий прямоугольник. Если не задан, берется у карты.

Результат
Изображение.

23.2 ListLayers - Список слоев карты

class axipy.**ListLayers**

Группа слоев. Может включать в себя как слои `axipy.Layer` так и группы слоев `axipy.ListLayers`. Пример использования см `axipy.Map.layers`

Свойства:

`count`	Количество слоев и групп слоев.
`title`	Наименование группы.
`visible`	Управляет видимостью группы.

Методы:

`add_group`(name)	Создает пустую группу.
`append`(layer)	Добавляет слой в карту.
`at`(index)	Возвращает слой или группы слоев по их индексу.
`group`(indexes, name)	Группировка слоев и групп в соответствие со списком их индексов.
`move`(from_index, to_index)	Перемещает слой или вложенную группу слоев в списке слоев по его индексу.
`remove`(index)	Удаляет слой по индексу.
`ungroup`(index)	Разгруппировка группы слоев по его индексу.

add_group(name: str)

Создает пустую группу.

Параметры
name – Наименование создаваемой группы.

append(layer: Layer)

Добавляет слой в карту. Добавление группы слоев не поддерживается и производится путем группировки существующих элементов посредством метода `group()`.

Параметры
layer – Добавляемый слой.

Исключение
ValueError – Если слой уже содержится в карте.

at(index: int) → Union[Layer, ListLayers]

Возвращает слой или группы слоев по их индексу.

Параметры
index – Индекс слоя или группы в списке.

Например:

```
layers.at(2)
layers[2]
```

property count: int

Количество слоев и групп слоев. Так же допустимо использование функции `len()`

group(indexes: List[int], name: str)

Группировка слоев и групп в соответствие со списком их индексов. При этом создается новая группа и все элементы (слои и группы слоев) помещаются внутрь этой группы.

Параметры

- **indexes** – Список индексов элементов, которые необходимо объединить.
- **name** – Наименование создаваемой группы.

move(from_index: int, to_index: int)

Перемещает слой или вложенную группу слоев в списке слоев по его индексу.

Параметры

- **from_index** – Индекс слоя для перемещения.
- **to_index** – Целевой индекс.

remove(index: int)

Удаляет слой по индексу.

Параметры
index – Индекс удаляемого слоя.

property title: str

Наименование группы.

ungroup(index: int)

Разгруппировка группы слоев по его индексу. при этом все внутренние элементы переносятся на верхний уровень данного списка. Если по индексу располагается не группа, то будет выброшено исключение.

Параметры
index – Индекс группы слоев.

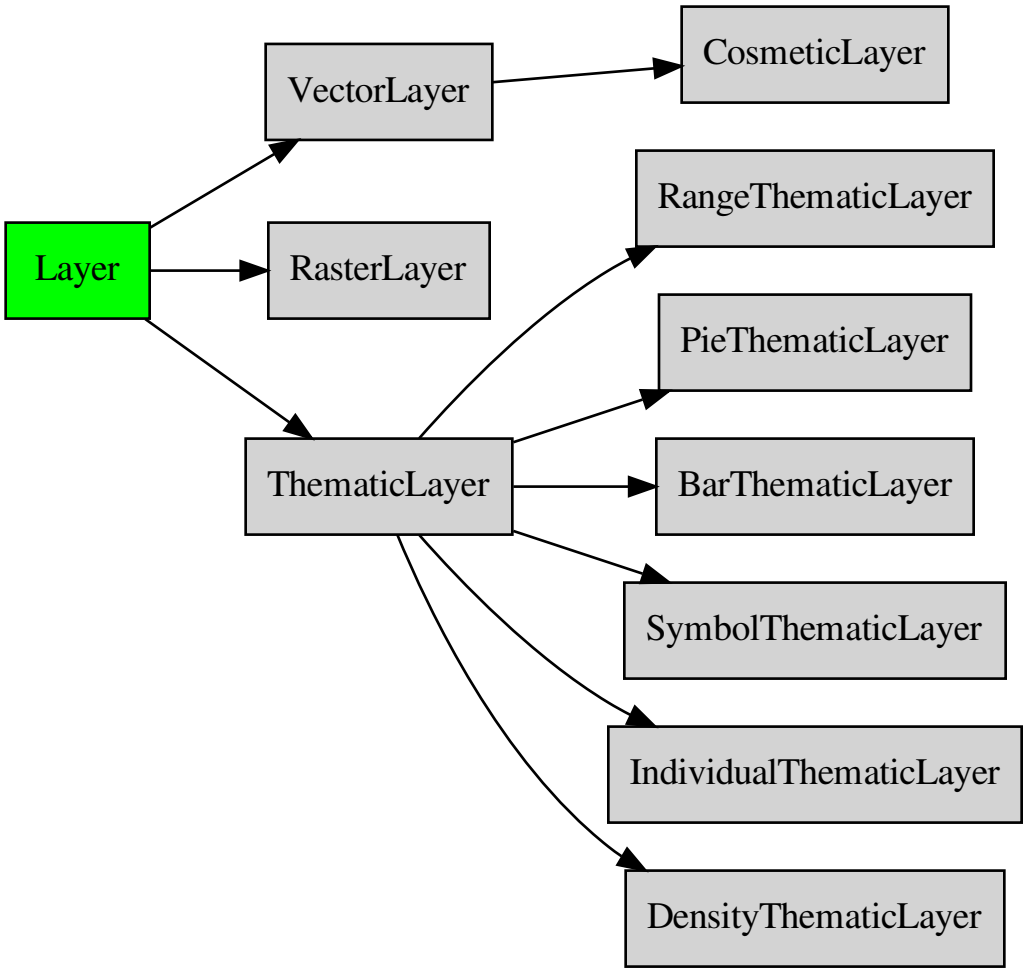
property visible

Управляет видимостью группы.

23.3 Слой

23.3.1 Layer - Слой

Иерархия классов слоев карты:



class axipy.**Layer**

Абстрактный базовый класс для слоя карты.

Для создания нового экземпляра для векторного или растрового источника данных необходимо использовать метод `Layer.create()`. Для тематических слоев - использовать соответствующие им конструкторы.

Классовые методы:

create(dataObject)	Создает слой на базе открытой таблицы или растра.

Свойства:

coordsystem	Координатная система, в которой находятся данные, отображаемые слоем.
data_object	Источник данных для слоя.
is_valid	Проверка на валидность объекта.
max_zoom	Максимальная ширина окна, при котором слой отображается на карте.
min_zoom	Минимальная ширина окна, при котором слой отображается на карте.
opacity	Прозрачность слоя в составе карты.
selectable	Управляет доступностью для выбора объектов слоя, если это поддерживается.
title	Наименование слоя.
visible	Управляет видимостью слоя.
zoom_restrict	Будет ли использоваться ограничение по отображению.

Методы:

get_bounds()	Возвращает область, в которую попадают все данные, которые могут быть отображены на слое.

Сигналы:

data_changed	Сигнал об изменении контента слоя.
need_redraw	Сигнал о необходимости перерисовать слой.

property coordsystem: CoordSystem

Координатная система, в которой находятся данные, отображаемые слоем.

classmethod create(dataObject: DataObject) → Layer

Создает слой на базе открытой таблицы или растра.

Параметры

dataObject – Таблица или растр. В зависимости от переданного объекта будет создан VectorLayer или RasterLayer.

Список 6: Пример создания слоя на базе файла.

```
# Векторный слой
table = provider_manager.openfile(filepath)
vector_layer = Layer.create(table)
# Подпишемся на обновление контента слоя
vector_layer.need_redraw.connect(lambda: print('Update layer'))
```

property data_changed: Signal

Сигнал об изменении контента слоя.

Тип результата
Signal[]

property data_object: DataObject

Источник данных для слоя.

get_bounds() → Rect

Возвращает область, в которую попадают все данные, которые могут быть отображены на слое.

property is_valid: bool

Проверка на валидность объекта. Слой мог быть удален, как пример, в связи с закрытием таблицы

property max_zoom: float

Максимальная ширина окна, при котором слой отображается на карте. Учитывается только при установленном zoom_restrict=True

property min_zoom: float

Минимальная ширина окна, при котором слой отображается на карте. Учитывается только при установленном zoom_restrict=True

property need_redraw: Signal

Сигнал о необходимости перерисовать слой.

Тип результата
Signal[]

property opacity: int

Прозрачность слоя в составе карты. Доступные значения от 0 до 100.

property selectable

Управляет доступностью для выбора объектов слоя, если это поддерживается.

property title: str

Наименование слоя.

property visible

Управляет видимостью слоя.

Выключение видимости верхнего слоя для активной карты:

```
if view_manager.active is not None:
    view_manager.active.map.layers[0].visible = False
```

property zoom_restrict: bool

Будет ли использоваться ограничение по отображению. Если установлено True, то для ограничения отображения слоя в зависимости от масштаба используются значения свойств zoom_min и zoom_max

23.3.2 RasterLayer - Растровый слой

class axipy.**RasterLayer**

Базовые классы: Layer

Класс, который должен использоваться в качестве базового класса для тех слоев, в которых используются свойства отрисовки растрового изображения.

Примечание: Создание слоя производится посредством метода вызова Layer.create()

Список 7: Примеры создания растрового слоя.

```
raster = provider_manager.openfile(filename)
raster_layer = Layer.create(raster)
raster_layer.transparentColor = QColor('#000014')
```

Классовые методы:

create(dataObject)	Создает слой на базе открытой таблицы или растра.

Свойства:

brightness	Яркость.
contrast	Контраст.
coordsystem	Координатная система, в которой находятся данные, отображаемые слоем.
data_object	Источник данных для слоя.
grayscale	Является ли данное изображение черно-белым.
is_valid	Проверка на валидность объекта.
max_zoom	Максимальная ширина окна, при котором слой отображается на карте.
min_zoom	Минимальная ширина окна, при котором слой отображается на карте.
opacity	Прозрачность слоя в составе карты.
selectable	Управляет доступностью для выбора объектов слоя, если это поддерживается.
title	Наименование слоя.
transparentColor	Цвет растра, который обрабатывается как прозрачный.
visible	Управляет видимостью слоя.
zoom_restrict	Будет ли использоваться ограничение по отображению.

Методы:

get_bounds()	Возвращает область, в которую попадают все данные, которые могут быть отображены на слое.

Сигналы:

data_changed	Сигнал об изменении контента слоя.
need_redraw	Сигнал о необходимости перерисовать слой.

property brightness: int

Яркость. Значение может быть в пределах от -100 до 100.

property contrast: int

Контраст. Значение может быть в пределах от -100 до 100.

property coordsystem: CoordSystem

Координатная система, в которой находятся данные, отображаемые слоем.

classmethod create(dataObject: DataObject) → Layer

Создает слой на базе открытой таблицы или растра.

Параметры

dataObject – Таблица или растр. В зависимости от переданного объекта будет создан VectorLayer или RasterLayer.

Список 8: Пример создания слоя на базе файла.

```
# Векторный слой
table = provider_manager.openfile(filepath)
vector_layer = Layer.create(table)
# Подпишемся на обновление контента слоя
vector_layer.need_redraw.connect(lambda: print('Update layer'))
```

property data_changed: Signal

Сигнал об изменении контента слоя.

Тип результата

Signal[]

property data_object: DataObject

Источник данных для слоя.

get_bounds() → Rect

Возвращает область, в которую попадают все данные, которые могут быть отображены на слое.

property grayscale: bool

Является ли данное изображение черно-белым.

property is_valid: bool

Проверка на валидность объекта. Слой мог быть удален, как пример, в связи с закрытием таблицы

property max_zoom: float

Максимальная ширина окна, при котором слой отображается на карте. Учитывается только при установленном zoom_restrict=True

property min_zoom: float

Минимальная ширина окна, при котором слой отображается на карте. Учитывается только при установленном zoom_restrict=True

property need_redraw: Signal

Сигнал о необходимости перерисовать слой.

Тип результата
Signal[]

property opacity: int

Прозрачность слоя в составе карты. Доступные значения от 0 до 100.

property selectable

Управляет доступностью для выбора объектов слоя, если это поддерживается.

property title: str

Наименование слоя.

property transparentColor: QColor

Цвет растра, который обрабатывается как прозрачный.

property visible

Управляет видимостью слоя.

Выключение видимости верхнего слоя для активной карты:

```
if view_manager.active is not None:
    view_manager.active.map.layers[0].visible = False
```

property zoom_restrict: bool

Будет ли использоваться ограничение по отображению. Если установлено True, то для ограничения отображения слоя в зависимости от масштаба используются значения свойств zoom_min и zoom_max

23.3.3 VectorLayer - Векторный слой

class axipy.**VectorLayer**

Базовые классы: Layer

Слой, основанный на базе векторных данных.

Примечание: Создание слоя производится посредством метода вызова Layer.create()

Список 9: Примеры работы со свойствами слоя.

```
# Зададим в качестве формулы метки атрибут "Страна" и запретим перекрытие меток␣
↪друг другом:
world.label.text = "Страна"
```

(continues on next page)

(продолжение с предыдущей страницы)

```
world.label.placementPolicy = LabelOverlap.DisallowOverlap
# Задание стиля оформления слоя
style_lay = Style.from_mapinfo("Pen (1, 2, 0) Brush (8, 255) Symbol (33,255,14)")
world.overrideStyle = style_lay
# Для сброса переопределения достаточно задать значение None
world.overrideStyle = None
```

Классовые методы:

create(dataObject)	Создает слой на базе открытой таблицы или растра.

Свойства:

coordsystem	Координатная система, в которой находятся данные, отображаемые слоем.
data_object	Источник данных для слоя.
hotlink	Наименование атрибута таблицы для хранения гиперссылки.
is_valid	Проверка на валидность объекта.
label	Метки слоя.
linesDirectionVisibile	Показ направлений линий.
max_zoom	Максимальная ширина окна, при котором слой отображается на карте.
min_zoom	Минимальная ширина окна, при котором слой отображается на карте.
nodesVisible	Показ узлов линий и полигонов.
opacity	Прозрачность слоя в составе карты.
overrideStyle	Переопределяемый стиль слоя.
selectable	Управляет доступностью для выбора объектов слоя, если это поддерживается.
showCentroid	Показ центроидов на слое.
thematic	Перечень тематик для данного слоя.
title	Наименование слоя.
visible	Управляет видимостью слоя.
zoom_restrict	Будет ли использоваться ограничение по отображению.

Методы:

get_bounds()	Возвращает область, в которую попадают все данные, которые могут быть отображены на слое.

Сигналы:

data_changed	Сигнал об изменении контента слоя.
need_redraw	Сигнал о необходимости перерисовать слой.

property coordsystem: CoordSystem

Координатная система, в которой находятся данные, отображаемые слоем.

classmethod create(dataObject: DataObject) → Layer

Создает слой на базе открытой таблицы или растра.

Параметры

dataObject – Таблица или растр. В зависимости от переданного объекта будет создан VectorLayer или RasterLayer.

Список 10: Пример создания слоя на базе файла.

```
# Векторный слой
table = provider_manager.openfile(filepath)
vector_layer = Layer.create(table)
# Подпишемся на обновление контента слоя
vector_layer.need_redraw.connect(lambda: print('Update layer'))
```

property data_changed: Signal

Сигнал об изменении контента слоя.

Тип результата

Signal[]

property data_object: DataObject

Источник данных для слоя.

get_bounds() → Rect

Возвращает область, в которую попадают все данные, которые могут быть отображены на слое.

property hotlink: str

Наименование атрибута таблицы для хранения гиперссылки.

Таблица 1: Возможны следующие варианты

Значение	Описание
axioma://world.tab	Открывает файл или рабочее пространство в аксиоме
addlayer://world	Добавляет слой world в текущую карту
exec://gimp	Запускает на выполнение программу gimp
https://axioma-gis.ru/	Открывает ссылку в браузере

Если префикс отсутствует, то производится попытка запустить по ассоциации.

property is_valid: bool

Проверка на валидность объекта. Слой мог быть удален, как пример, в связи с закрытием таблицы

property label: Label

Метки слоя. В качестве формулы может использоваться или наименование поля таблицы или выражение.

property linesDirectionVisibile: bool

Показ направлений линий.

property max_zoom: float

Максимальная ширина окна, при котором слой отображается на карте. Учитывается только при установленном zoom_restrict=True

property min_zoom: float

Минимальная ширина окна, при котором слой отображается на карте. Учитывается только при установленном zoom_restrict=True

property need_redraw: Signal

Сигнал о необходимости перерисовать слой.

Тип результата
Signal[]

property nodesVisible: bool

Показ узлов линий и полигонов.

property opacity: int

Прозрачность слоя в составе карты. Доступные значения от 0 до 100.

property overrideStyle: Style

Переопределяемый стиль слоя. Если задан как None (по умолчанию), объекты будут отображены на основании оформления источника данных.

property selectable

Управляет доступностью для выбора объектов слоя, если это поддерживается.

property showCentroid: bool

Показ центроидов на слое.

property thematic: ListThematic

Перечень тематик для данного слоя. Работа с тематическими слоями похожа на работу со списком list.

Список 11: Пример.

```
# Создадим тематический слой
range1 = RangeThematicLayer("Население")
# Добавим в основной слой
world.thematic.append(range1)
# Получим добавленный тематический слой
range1 = world.thematic[0]
# Просмотр всех тематик слоя
for t in world.thematic:
    print('thematic:', t.title)
```

property title: str

Наименование слоя.

property visible

Управляет видимостью слоя.

Выключение видимости верхнего слоя для активной карты:

```
if view_manager.active is not None:
    view_manager.active.map.layers[0].visible = False
```

property zoom_restrict: bool

Будет ли использоваться ограничение по отображению. Если установлено True, то для ограничения отображения слоя в зависимости от масштаба используются значения свойств zoom_min и zoom_max

23.3.4 CosmeticLayer - Косметический слой

class axipy.**CosmeticLayer**

Базовые классы: VectorLayer

Косметический слой.

Классовые методы:

create(dataObject)	Создает слой на базе открытой таблицы или растра.

Свойства:

coordsystem	Координатная система, в которой находятся данные, отображаемые слоем.
data_object	Источник данных для слоя.
hotlink	Наименование атрибута таблицы для хранения гиперссылки.
is_valid	Проверка на валидность объекта.
label	Метки слоя.
linesDirectionVisibile	Показ направлений линий.
max_zoom	Максимальная ширина окна, при котором слой отображается на карте.
min_zoom	Минимальная ширина окна, при котором слой отображается на карте.
nodesVisible	Показ узлов линий и полигонов.
opacity	Прозрачность слоя в составе карты.
overrideStyle	Переопределяемый стиль слоя.
selectable	Управляет доступностью для выбора объектов слоя, если это поддерживается.
showCentroid	Показ центроидов на слое.
thematic	Перечень тематик для данного слоя.
title	Наименование слоя.
visible	Управляет видимостью слоя.
zoom_restrict	Будет ли использоваться ограничение по отображению.

Методы:

get_bounds()	Возвращает область, в которую попадают все данные, которые могут быть отображены на слое.

Сигналы:

data_changed	Сигнал об изменении контента слоя.
need_redraw	Сигнал о необходимости перерисовать слой.

property coordsystem: CoordSystem

Координатная система, в которой находятся данные, отображаемые слоем.

classmethod create(dataObject: DataObject) → Layer

Создает слой на базе открытой таблицы или растра.

Параметры

dataObject – Таблица или растр. В зависимости от переданного объекта будет создан VectorLayer или RasterLayer.

Список 12: Пример создания слоя на базе файла.

```
# Векторный слой
table = provider_manager.openfile(filepath)
vector_layer = Layer.create(table)
# Подпишемся на обновление контента слоя
vector_layer.need_redraw.connect(lambda: print('Update layer'))
```

property data_changed: Signal

Сигнал об изменении контента слоя.

Тип результата

Signal[]

property data_object: DataObject

Источник данных для слоя.

get_bounds() → Rect

Возвращает область, в которую попадают все данные, которые могут быть отображены на слое.

property hotlink: str

Наименование атрибута таблицы для хранения гиперссылки.

Таблица 2: Возможны следующие варианты

Значение	Описание
axioma://world.tab	Открывает файл или рабочее пространство в аксиоме
addlayer://world	Добавляет слой world в текущую карту
exec://gimp	Запускает на выполнение программу gimp
https://axioma-gis.ru/	Открывает ссылку в браузере

Если префикс отсутствует, то производится попытка запустить по ассоциации.

property is_valid: bool

Проверка на валидность объекта. Слой мог быть удален, как пример, в связи с закрытием таблицы

property label: Label

Метки слоя. В качестве формулы может использоваться или наименование поля таблицы или выражение.

property linesDirectionVisibile: bool

Показ направлений линий.

property max_zoom: float

Максимальная ширина окна, при котором слой отображается на карте. Учитывается только при установленном zoom_restrict=True

property min_zoom: float

Минимальная ширина окна, при котором слой отображается на карте. Учитывается только при установленном zoom_restrict=True

property need_redraw: Signal

Сигнал о необходимости перерисовать слой.

Тип результата
Signal[]

property nodesVisible: bool

Показ узлов линий и полигонов.

property opacity: int

Прозрачность слоя в составе карты. Доступные значения от 0 до 100.

property overrideStyle: Style

Переопределяемый стиль слоя. Если задан как None (по умолчанию), объекты будут отображены на основании оформления источника данных.

property selectable

Управляет доступностью для выбора объектов слоя, если это поддерживается.

property showCentroid: bool

Показ центроидов на слое.

property thematic: ListThematic

Перечень тематик для данного слоя. Работа с тематическими слоями похожа на работу со списком list.

Список 13: Пример.

```
# Создадим тематический слой
range1 = RangeThematicLayer("Население")
# Добавим в основной слой
world.thematic.append(range1)
# Получим добавленный тематический слой
range1 = world.thematic[0]
# Просмотр всех тематик слоя
for t in world.thematic:
    print('thematic:', t.title)
```

property title: str

Наименование слоя.

property visible

Управляет видимостью слоя.

Выключение видимости верхнего слоя для активной карты:

```
if view_manager.active is not None:
    view_manager.active.map.layers[0].visible = False
```

property zoom_restrict: bool

Будет ли использоваться ограничение по отображению. Если установлено True, то для ограничения отображения слоя в зависимости от масштаба используются значения свойств zoom_min и zoom_max

23.3.5 ListThematic - Перечень тематик для векторного слоя

class axipy.**ListThematic**

Список тематических слоев (тематик) карты.

Свойства:

count	Количество тематик слоя.

Методы:

append(lay)	Добавить тематику.
at(idx)	Получение тематики по ее индексу.
move(fromIdx, toIdx)	Поменять тематики местами.
remove(idx)	Удалить тематику.

append(lay: ThematicLayer)

Добавить тематику.

Параметры
lay – Добавляемый тематический слой.

at(idx: int) → ThematicLayer

Получение тематики по ее индексу.

Параметры
idx – Индекс запрашиваемой тематики.

property count: int

Количество тематик слоя.

move(fromIdx: int, toIdx: int)

Поменять тематики местами.

Параметры

- **fromIdx** – Текущий индекс.
- **toIdx** – Новое положение.

remove(idx: int)

Удалить тематику.

Параметры
idx – Индекс удаляемого слоя.

23.3.6 Метки

Label - Метка для векторного слоя

Модуль отрисовки.

Данный модуль содержит инструменты, предназначенные для отрисовки геопространственных и прочих данных.

class axipy.render.**Label**

Метки слоя. Доступны через свойство векторного слоя `VectorLayer.label`.

Список 14: Пример использования.

```
# Открываем таблицу
table = provider_manager.openfile(filepath)
# Создаем слой
layer = Layer.create(table)
# Формула метки
layer.label.text = 'Страна'
# Видимость
layer.label.visible = True
# Если метки перекрывают друг друга, ищем другое положение
layer.label.placementPolicy = LabelOverlap.OtherPosition
# Цвет шрифта
layer.label.color = Qt.blue
# устанавливаем прорачность
layer.label.opacity = 50
# Показываем в пределах (0...3000км)
layer.label.rangeEnabled = True
layer.label.rangeMax = 3000000
# Положение подписей для точечных объектов
p_layout = layer.label.pointLayout
p_layout.position = LabelLayoutPosition.BottomRight
p_layout.visible = True
p_layout.offset = QSize(3, 3)
layer.label.pointLayout = p_layout
# Положение подписей для линейных объектов
l_layout = layer.label.lineLayout
l_layout.position = LabelLayoutPosition.Bottom
layer.label.lineLayout = l_layout
# Горизонтальное выравнивание подписей для линий
layer.label.horizontalAlign = LabelHorizontalAlign.Center
# Игнорируем дубликаты
layer.label.supressDuplicates = True
# Свес линии
layer.label.overhang = 67
```

Свойства:

areaInterior	Режим подписей для областей.
areaLayout	Положение подписей для областей.
areaPosition	Режим подписей для областей.
backgroundColor	Цвет фона
backgroundSize	Толщина фона в пунктах.
backgroundType	Фон подписи.
color	Цвет шрифта меток.
font	Шрифт.
horizontalAlign	Горизонтальное выравнивание подписей.
lineKeepDirection	Направление текста строится вдоль направления линии.
lineLayout	Положение подписей для линий.
linePosition	Режим подписей для линий.
opacity	Прозрачность (0..100).
overhang	Максимальный свес для линии (в %).
placementPolicy	Принцип наложения меток на слой карты.
pointLayout	Положение подписей для точек.
rangeEnabled	Показывать в пределах.
rangeMax	Максимальный предел показа с метрах при включенном свойстве rangeEnabled.
rangeMin	Минимальный предел показа с метрах при включенном свойстве rangeEnabled.
shadow	Тень.
spacing	Разрядка.
supressDuplicates	Запретить повтор подписей.
text	Наименование атрибута таблицы либо выражение для метки, которое может основываться на одном или нескольких атрибутах.
useClip	Использовать динамические подписи.
visible	Управляет видимостью меток.

property areaInterior: LabelAreaInterior

Режим подписей для областей. По умолчанию LabelAreaInterior.Centroid.

property areaLayout: LabelLayout

Положение подписей для областей.

property areaPosition: LabelAreaPosition

Режим подписей для областей. По умолчанию LabelAreaPosition.Horizontal.

property backgroundColor: QColor

Цвет фона

property backgroundSize: int

Толщина фона в пунктах.

property backgroundType: LabelBackgroundType

Фон подписи. По умолчанию отсутствует

property color: QColor

Цвет шрифта меток.

property font: QFont

Шрифт.

property horizontalAlign: LabelHorizontalAlign

Горизонтальное выравнивание подписей. По умолчанию LabelHorizontalAlign.Flat

property lineKeepDirection: bool

Направление текста строится вдоль направления линии. По умолчанию False.

property lineLayout: LabelLayout

Положение подписей для линий.

property linePosition: LabelLinePosition

Режим подписей для линий. По умолчанию LabelLinePosition.FollowPath.

property opacity: int

Прозрачность (0..100). По умолчанию 100 (Непрозрачно).

property overhang: int

Максимальный свес для линии (в %).

property placementPolicy: LabelOverlap

Принцип наложения меток на слой карты. По умолчанию LabelOverlap.AllowOverlap

property pointLayout: LabelLayout

Положение подписей для точек.

property rangeEnabled: bool

Показывать в пределах. Если True, используются свойства `rangeMin` и `rangeMax`. По умолчанию False.

property rangeMax: float

Максимальный предел показа с метрах при включенном свойстве `rangeEnabled`.

property rangeMin: float

Минимальный предел показа с метрах при включенном свойстве `rangeEnabled`.

property shadow: bool

Тень. По умолчанию False

property spacing: bool

Разрядка. По умолчанию False

property supressDuplicates: bool

Запретить повтор подписей. Подписи с одинаковым текстом на этом слое будут отображаться один раз. По умолчанию False.

property text: str

Наименование атрибута таблицы либо выражение для метки, которое может основываться на одном или нескольких атрибутах.

property useClip: bool

Использовать динамические подписи. По умолчанию False.

property visible: bool

Управляет видимостью меток.

Свойства меток

class axipy.render.**LabelOverlap**

Стратегия при наложении подписей `Label.placementPolicy`.

Атрибуты:

`AllowOverlap`	Допускать перекрытие меток (по умолчанию).
`DisallowOverlap`	Не допускать перекрытие меток.
`OtherPosition`	Пробовать найти для метки новую позицию.

class axipy.render.**LabelBackgroundType**

Фон подписи `Label.backgroundType`.

Атрибуты:

`Empty`	Отсутствует
`Frame`	Фломастер
`Outline`	Кайма

class axipy.render.**LabelLinePosition**

Режим подписей для линий `Label.linePosition`.

Атрибуты:

`FollowPath`	Вдоль линии
`Horizontal`	Горизонтально
`Parallel`	Вдоль сегмента

class axipy.render.**LabelAreaPosition**

Режим подписей для областей `Label.areaPosition`.

Атрибуты:

`Automatic`	Автоматически
`Centroid`	У центроида
`Horizontal`	Горизонтально
`Vertical`	Вертикально

class axipy.render.**LabelAreaInterior**

Выбор участка для областей `Label.areaInterior`.

Атрибуты:

Centroid	Ближайший к центроиду
Max	Наибольший

class axipy.render.**LabelHorizontalAlign**

Горизонтальное выравнивание подписи Label.horizontalAlign.

Атрибуты:

Begin	Подпись располагается в начале линии
Center	Середина подписи совпадает с центром линии
Centroid	Подпись располагается относительно центроида
End	Подпись располагается в конце линии
Flat	По центру пологого участка (можно задать угол на вкладке Дополнительно)

LabelLayout - Положение подписей

class axipy.**LabelLayout**

Положение подписей Label.

Свойства:

offset	Смещение.
position	Расположение подписи.
visible	Видимость подписи для данного типа геометрии.

property offset: QSize

Смещение. Интервал значений (0..100)

property position: LabelLayoutPosition

Расположение подписи.

property visible: bool

Видимость подписи для данного типа геометрии. По умолчанию True

class axipy.**LabelLayoutPosition**

Относительное положение подписи LabelLayout.position.

Атрибуты:

Bottom	Снизу
BottomLeft	Снизу слева
BottomRight	Снизу справа
Center	По центру
Left	Слева
Right	Справа
Top	Сверху
TopLeft	Сверху слева
TopRight	Сверху справа

CustomLabelProperties - Свойства выносной метки

class axipy.**CustomLabelProperties**

Свойства выносной метки. Используется при задании параметров положения метки карты `CustomLabels.set()` или получения их `CustomLabels.get()`.

Свойства:

`angle`	Угол поворота текста в градусах
`endType`	Тип выноски.
`expression`	Текст.
`position`	Переопределенное положение метки.

property angle: float

Угол поворота текста в градусах

property endType: CustomLabelEndType

Тип выноски. По умолчанию CustomLabelEndType.EndNone

property expression: str

Текст. Если не задано, используется базовая формула для слоя.

property position: Point

Переопределенное положение метки. Если используется по умолчанию, возвращается None. Если при задании не указана система координат, используется СК слоя, которому принадлежит метка.

class axipy.**CustomLabelEndType**

Тип выноски для метки `CustomLabelProperties.endType`.

Атрибуты:

`Arrow`	Стрелка
`EndNone`	Не отображать (по умолчанию)
`Line`	Линия

23.4 Legend - Легенда слоя

class axipy.**Legend**

Легенда слоя. Позволяет получить информацию об условных обозначениях на слое. Созданная легенда в дальнейшем может быть помещена на лист отчета `axipy.render.Report` как `axipy.render.LegendReportItem` или же расположена в отдельном окне легенд слоев для карты `axipy.render.Map`.

Параметры
lay – Слой, для которого создается легенда.

Список 15: Пример создания легенды.

```
range1 = RangeThematicLayer("Население")
world.thematic.add(range1)
# Легенда для тематического слоя
```

(continues on next page)

(продолжение с предыдущей страницы)

```
legend = Legend(range1)
legend.columns = 2
# Зададим стиль заднего фона и окантовки
legend.border_style = LineStyle(3, Qt.red)
legend.fill_style = PolygonStyle(49, Qt.yellow)
# Изменим описание для первого элемента
item = legend.items[0]
item.title = 'Описание'
legend.items[0] = item
# Заголовок легенды
legend.caption = 'Легенда для слоя'
# Стиль заголовка
legend.style_caption = Style.from_mapinfo("Font (\"Arial\", 0, 9, 255)")
# Просмотр всех стилей легенды
for it in legend.items:
    print(it.title, it.visible, it.style.to_mapinfo())
# Изменение позиции элемента легенды
legend.items.move(0, 1)
# Отрисуем легенду в контексте вместе с картой
image = QImage(800, 600, QImage.Format_ARGB32_Premultiplied)
image.fill(Qt.white)
painter = QPainter(image)
context = Context(painter)
legend.position = (100, 50)
legend.draw(context)
'''
>>> Описание True Pen (1, 2, 8421504) Brush (2, 16776960)
>>> 55419-166640 True Pen (1, 2, 8421504) Brush (2, 12582656)
>>> 166640-631500 True Pen (1, 2, 8421504) Brush (2, 8453888)
...
'''
```

Свойства:

border_style	Стиль используемой окантовки.
caption	Заголовок легенды.
columns	Количество колонок в легенде.
fill_style	Стиль заливки заднего фона.
items	Перечень стилей легенды.
position	Положение легенды в контексте рисования.
style_caption	Стиль заголовка легенды.
style_subcaption	Стиль подзаголовка легенды.
style_text	Стиль текстовых подписей.
subcaption	Подзаголовок легенды.

Методы:

draw(context)	Рисует легенду в контексте.
refresh()	Обновляет стили из источника.
to_image(width, height)	Возвращает легенду в виде растра.

property border_style: Style

Стиль используемой окантовки. Отображается если has_border установлено в

True.

property caption: str

Заголовок легенды. Стиль заголовка задается свойством `style_caption`

property columns: int

Количество колонок в легенде. По умолчанию 1.

draw(context: Context)

Рисует легенду в контексте.

Легенду также можно отрисовать совместно с картой в одном контексте (см. `Map.draw()`).

Параметры

context – Контекст рисования.

property fill_style: Style

Стиль заливки заднего фона.

property items: ListLegendItems

Перечень стилей легенды. Реализован в виде списка. Для изменения какого-либо параметра необходимо сначала получить элемент, затем поменять требуемое свойство, а затем измененный элемент переназначить.

property position: Pnt

Положение легенды в контексте рисования.

refresh()

Обновляет стили из источника.

property style_caption: Style

Стиль заголовка легенды.

property style_subcaption: Style

Стиль подзаголовка легенды.

property style_text: Style

Стиль текстовых подписей.

property subcaption: str

Подзаголовок легенды. Стиль заголовка задается свойством `style_subcaption`

to_image(width: int, height: int) → QImage

Возвращает легенду в виде растра.

Параметры

- **width** – Ширина выходного растра.
- **height** – Высота выходного растра.

23.5 LegendItem - Элемент легенды

class axipy.**LegendItem**

Элемент легенды.

Свойства:

style	Стиль оформления элемента легенды.
title	Описание элемента легенды.
visible	Видимость элемента легенды.

property style: Style

Стиль оформления элемента легенды.

property title: str

Описание элемента легенды.

property visible: bool

Видимость элемента легенды.

23.6 ListLegendItems - Список элементов легенды

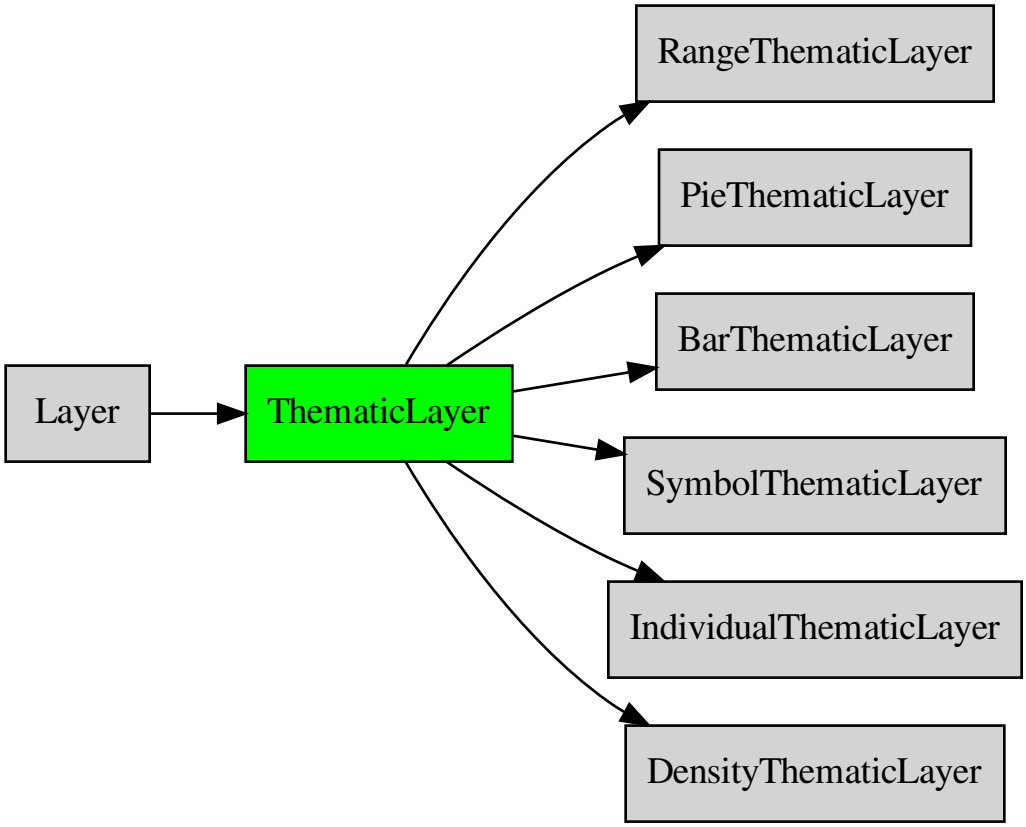
class axipy.**ListLegendItems**

Элементы легенды.

23.7 Тематика

23.7.1 ThematicLayer - Тематика

Иерархия классов тематик:



class axipy.**ThematicLayer**

Базовые классы: Layer

Абстрактный класс слоя с тематическим оформлением векторного слоя карты на базе атрибутивной информации.

23.7.2 ReallocateThematicColor - Распределение цветов

class axipy.**ReallocateThematicColor**

Базовые классы: object

Поддержка различного рода алгоритмов распределения оформления.

Методы:

assign_gray([minV, maxV])	Распределение в виде градации серого.
assign_monotone(color[, minv, maxv])	Монотонная заливка разной яркости (оттенки красного, синего и т.п.).
assign_rainbow([sequential, saturation, value])	Распределение цветов по спектру.
assign_three_colors(colorMin, colorMax, ...)	Цвет, распределенный между тремя заданными цветами (с разрывом).
assign_two_colors(colorMin, colorMax[, useHSV])	Равномерно распределяет оформление по заданным крайним цветам.

assign_gray(minV: int = 20, maxV: int = 80)

Распределение в виде градации серого. Значение задается в интервале (0..100) от черного до белого.

Параметры

- **minV** – Минимальное значение.
- **maxV** – Максимальное значение.

assign_monotone(color: QColor, minv: int = 20, maxv: int = 80)

Монотонная заливка разной яркости (оттенки красного, синего и т.п.). Цветовая схема HSL. Максимальное и минимальное значения задаются в интервале (0..100).

Параметры

- **color** – Базовый цвет.
- **minV** – Минимальное значение.
- **maxV** – Максимальное значение.

assign_rainbow(sequential: bool = True, saturation: float = 90, value: float = 90)

Распределение цветов по спектру. Цветовая схема HSV.

Параметры

- **sequential** – Если True, то последовательное распределение цветов. В противном случае распределение случайно.
- **saturation** – Яркость. Задается в интервале (0..100)
- **value** – Насыщенность. Задается в интервале (0..100)

assign_three_colors(colorMin: QColor, colorMax: QColor, colorBreak: QColor, br: int, useHSV: bool = True)

Цвет, распределенный между тремя заданными цветами (с разрывом).

Параметры

- **colorMin** – Цвет нижнего диапазона.

- **colorMax** – Цвет верхнего диапазона.
- **colorBreak** – Цвет на уровне разрыва.
- **br** – Индекс интервала, на на котором используется цвет разрыва.
- **useHSV** – Если True, то будет использоваться схема HSV. В противном случае - RGB.

assign_two_colors(colorMin: QColor, colorMax: QColor, useHSV: bool = False)

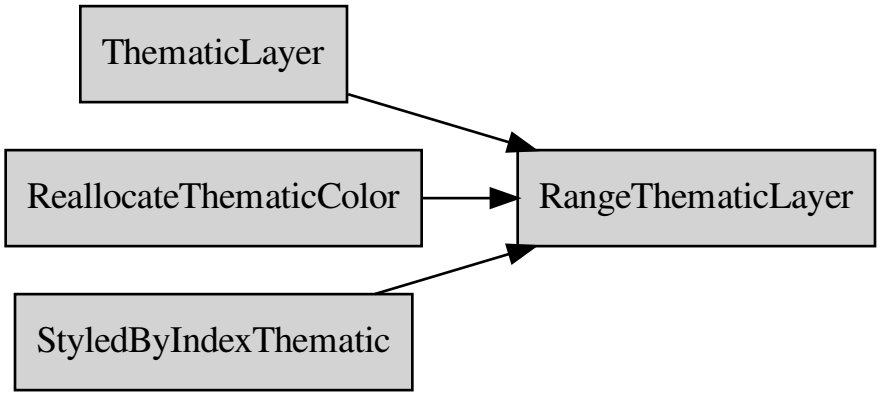
Равномерно распределяет оформление по заданным крайним цветам.

Параметры

- **colorMin** – Цвет нижнего диапазона.
- **colorMax** – Цвет верхнего диапазона.
- **useHSV** – Если True, то будет использоваться схема HSV. В противном случае - RGB.

23.7.3 RangeThematicLayer - Интервалы

Иерархия классов:



class axipy.**RangeThematicLayer**

Базовые классы: ThematicLayer, StyledByIndexThematic, ReallocateThematicColor

Тематическое оформление слоя с распределением значений по интервалам. Для распределения цветов по заданным интервалам могут быть использованы функции assign_* класса ReallocateThematicColor в зависимости от требуемых целей.

Параметры

expression – Наименование атрибута таблицы или выражение.

Список 16: Пример создания тематики по интервалам.

```
# Пример создания тематики с последующим добавлением ее к базовому слою `world`
range1 = RangeThematicLayer("Население")
range1.ranges = 6
range1.splitType = RangeThematicLayer.EQUAL_COUNT
range1.assign_two_colors(Qt.red, Qt.cyan)
world.thematic.add(range1)
# Пример запроса с последующей заменой::
v = world.thematic[0].get_interval_value(2)  # Запрос
v = (999, v[1])  # Заменим минимальное значение для интервала с индексом 2
world.thematic[0].set_interval_value(2, v)  # Замена
# Различные виды распределения интервалов тематик по цветам
range1.assign_two_colors(Qt.red, Qt.yellow)
range1.assign_three_colors(Qt.yellow, Qt.cyan, Qt.green, 4)
range1.assign_rainbow()
range1.assign_gray(80, 100)
```

Классовые методы:

create(dataObject)	Создает слой на базе открытой таблицы или растра.

Свойства:

coordsystem	Координатная система, в которой находятся данные, отображаемые слоем.
data_object	Источник данных для слоя.
is_valid	Проверка на валидность объекта.
max_zoom	Максимальная ширина окна, при котором слой отображается на карте.
min_zoom	Минимальная ширина окна, при котором слой отображается на карте.
opacity	Прозрачность слоя в составе карты.
ranges	Количество интервалов.
selectable	Управляет доступностью для выбора объектов слоя, если это поддерживается.
splitType	Тип распределения значений по интервалам.
title	Наименование слоя.
visible	Управляет видимостью слоя.
zoom_restrict	Будет ли использоваться ограничение по отображению.

Методы:

assign_gray([minV, maxV])	Распределение в виде градации серого.
assign_monotone(color[, minv, maxv])	Монотонная заливка разной яркости (оттенки красного, синего и т.п.).
assign_rainbow([sequential, saturation, value])	Распределение цветов по спектру.
assign_three_colors(colorMin, colorMax, ...)	Цвет, распределенный между тремя заданными цветами (с разрывом).
assign_two_colors(colorMin, colorMax[, useHSV])	Равномерно распределяет оформление по заданным крайним цветам.
get_bounds()	Возвращает область, в которую попадают все данные, которые могут быть отображены на слое.
get_interval_value(idx)	Возвращает предельные значения для указанного интервала в виде пары значений.
get_style(idx)	Стиль для указанного выражения.
set_interval_value(idx, v)	Заменяет предельные значения интервала.
set_style(idx, style)	Установка стиля оформления для выражения по его индексу в списке выражений.

Сигналы:

data_changed	Сигнал об изменении контента слоя.
need_redraw	Сигнал о необходимости перерисовать слой.

assign_gray(minV: int = 20, maxV: int = 80)

Распределение в виде градации серого. Значение задается в интервале (0..100) от черного до белого.

Параметры

- **minV** – Минимальное значение.
- **maxV** – Максимальное значение.

assign_monotone(color: QColor, minv: int = 20, maxv: int = 80)

Монотонная заливка разной яркости (оттенки красного, синего и т.п.). Цветовая схема HSL. Максимальное и минимальное значения задаются в интервале (0..100).

Параметры

- **color** – Базовый цвет.
- **minV** – Минимальное значение.
- **maxV** – Максимальное значение.

assign_rainbow(sequential: bool = True, saturation: float = 90, value: float = 90)

Распределение цветов по спектру. Цветовая схема HSV.

Параметры

- **sequential** – Если True, то последовательное распределение цветов. В противном случае распределение случайно.
- **saturation** – Яркость. Задается в интервале (0..100)
- **value** – Насыщенность. Задается в интервале (0..100)

assign_three_colors(colorMin: QColor, colorMax: QColor, colorBreak: QColor, br: int, useHSV: bool = True)

Цвет, распределенный между тремя заданными цветами (с разрывом).

Параметры

- **colorMin** – Цвет нижнего диапазона.
- **colorMax** – Цвет верхнего диапазона.
- **colorBreak** – Цвет на уровне разрыва.
- **br** – Индекс интервала, на на котором используется цвет разрыва.
- **useHSV** – Если True, то будет использоваться схема HSV. В противном случае - RGB.

assign_two_colors(colorMin: QColor, colorMax: QColor, useHSV: bool = False)

Равномерно распределяет оформление по заданным крайним цветам.

Параметры

- **colorMin** – Цвет нижнего диапазона.
- **colorMax** – Цвет верхнего диапазона.
- **useHSV** – Если True, то будет использоваться схема HSV. В противном случае - RGB.

property coordsystem: CoordSystem

Координатная система, в которой находятся данные, отображаемые слоем.

classmethod create(dataObject: DataObject) → Layer

Создает слой на базе открытой таблицы или растра.

Параметры

dataObject – Таблица или растр. В зависимости от переданного объекта будет создан VectorLayer или RasterLayer.

Список 17: Пример создания слоя на базе файла.

```
# Векторный слой
table = provider_manager.openfile(filepath)
vector_layer = Layer.create(table)
# Подпишемся на обновление контента слоя
vector_layer.need_redraw.connect(lambda: print('Update layer'))
```

property data_changed: Signal

Сигнал об изменении контента слоя.

Тип результата

Signal[]

property data_object: DataObject

Источник данных для слоя.

get_bounds() → Rect

Возвращает область, в которую попадают все данные, которые могут быть отображены на слое.

get_interval_value(idx: int) → Tuple[float, float]

Возвращает предельные значения для указанного интервала в виде пары значений.

Параметры
idx – Индекс диапазона.

get_style(idx: int) → Style

Стиль для указанного выражения.

Параметры
idx – Порядковый номер выражения.

property is_valid: bool

Проверка на валидность объекта. Слой мог быть удален, как пример, в связи с закрытием таблицы

property max_zoom: float

Максимальная ширина окна, при котором слой отображается на карте. Учитывается только при установленном zoom_restrict=True

property min_zoom: float

Минимальная ширина окна, при котором слой отображается на карте. Учитывается только при установленном zoom_restrict=True

property need_redraw: Signal

Сигнал о необходимости перерисовать слой.

Тип результата
Signal[]

property opacity: int

Прозрачность слоя в составе карты. Доступные значения от 0 до 100.

property ranges: int

Количество интервалов.

property selectable

Управляет доступностью для выбора объектов слоя, если это поддерживается.

set_interval_value(idx: int, v: Tuple[float, float])

Заменяет предельные значения интервала.

Параметры

- **idx** – Индекс диапазона.
- **v** – Значение в виде пары.

set_style(idx: int, style: Style)

Установка стиля оформления для выражения по его индексу в списке выражений.

Параметры

- **idx** – Индекс.

- **style** – Назначаемый стиль.

Список 18: Пример установки стиля для значения с индексом 2 первого тематического слоя.

```
style_new = Style.from_mapinfo("Brush (2, 255, 0)")
world.thematic[0].set_style(2, style_new)
```

property splitType: int

Тип распределения значений по интервалам.

Таблица 3: Допустимые значения:

Константа	Значение	Описание
EQUAL_INTERVAL	1	Распределение исходя из равномерности интервалов (по умолчанию).
EQUAL_COUNT	2	Распределение исходя их равного количества объектов в каждом интервале.
MANUAL	3	Ручное распределение значений путем задания пределов вручную.

property title: str

Наименование слоя.

property visible

Управляет видимостью слоя.

Выключение видимости верхнего слоя для активной карты:

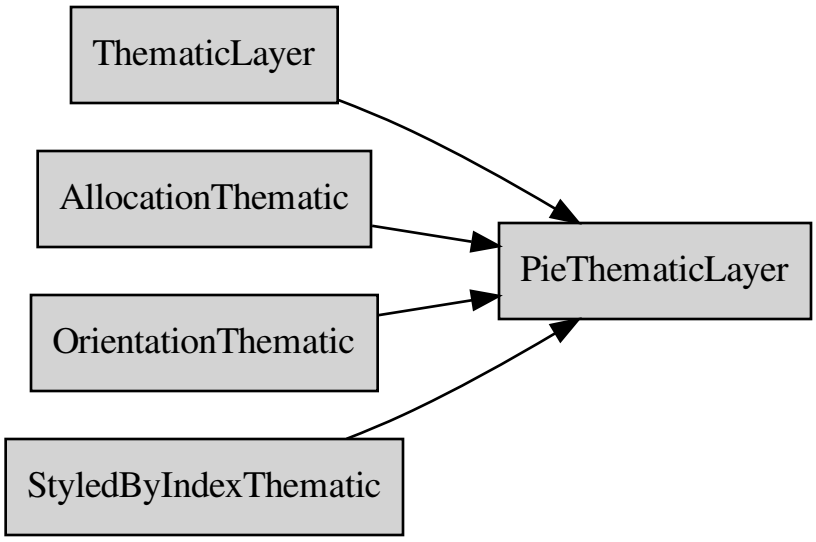
```
if view_manager.active is not None:
    view_manager.active.map.layers[0].visible = False
```

property zoom_restrict: bool

Будет ли использоваться ограничение по отображению. Если установлено True, то для ограничения отображения слоя в зависимости от масштаба используются значения свойств zoom_min и zoom_max

23.7.4 PieThematicLayer - Круговые диаграммы

Иерархия классов:



class axipy.**PieThematicLayer**

Базовые классы: ThematicLayer, AllocationThematic, OrientationThematic, StyledByIndexThematic

Тематика в виде круговых диаграмм.

Параметры

expressions – Наименования атрибутов или выражений в виде списка list.

Список 19: Создание тематики с последующим добавлением ее к базовому слою.

```
pie = PieThematicLayer(["Население", "Мужское", "Женское"])
pie.allocationType = PieThematicLayer.SQRT
style_lay_pie = Style.from_mapinfo("Brush (8, 65535, 0)")
# Заменим стиль
pie.set_style(0, style_lay_pie)
# Добавляем к основному слою
world.thematic.add(pie)
```

Классовые методы:

create(dataObject)	Создает слой на базе открытой таблицы или растра.

Свойства:

allocationType	Тип распределения значений.
coordsystem	Координатная система, в которой находятся данные, отображаемые слоем.
data_object	Источник данных для слоя.
is_valid	Проверка на валидность объекта.
max_zoom	Максимальная ширина окна, при котором слой отображается на карте.
min_zoom	Минимальная ширина окна, при котором слой отображается на карте.
opacity	Прозрачность слоя в составе карты.
orientationType	Ориентация относительно центроида.
selectable	Управляет доступностью для выбора объектов слоя, если это поддерживается.
startAngle	Начальный угол отсчета диаграммы.
title	Наименование слоя.
visible	Управляет видимостью слоя.
zoom_restrict	Будет ли использоваться ограничение по отображению.

Методы:

get_bounds()	Возвращает область, в которую попадают все данные, которые могут быть отображены на слое.
get_style(idx)	Стиль для указанного выражения.
set_style(idx, style)	Установка стиля оформления для выражения по его индексу в списке выражений.

Сигналы:

data_changed	Сигнал об изменении контента слоя.
need_redraw	Сигнал о необходимости перерисовать слой.

property allocationType: int

Тип распределения значений.

Таблица 4: Допустимые значения.

Константа	Значение	Описание
LINEAR	1	Линейное (по умолчанию)
SQRT	2	Квадратичное
LOG10	3	Логарифмическое

property coordsystem: CoordSystem

Координатная система, в которой находятся данные, отображаемые слоем.

classmethod create(dataObject: DataObject) → Layer

Создает слой на базе открытой таблицы или растра.

Параметры
dataObject – Таблица или растр. В зависимости от переданного объекта будет создан VectorLayer или RasterLayer.

Список 20: Пример создания слоя на базе файла.

```
# Векторный слой
table = provider_manager.openfile(filepath)
vector_layer = Layer.create(table)
# Подпишемся на обновление контента слоя
vector_layer.need_redraw.connect(lambda: print('Update layer'))
```

property data_changed: Signal

Сигнал об изменении контента слоя.

Тип результата
Signal[]

property data_object: DataObject

Источник данных для слоя.

get_bounds() → Rect

Возвращает область, в которую попадают все данные, которые могут быть отображены на слое.

get_style(idx: int) → Style

Стиль для указанного выражения.

Параметры
idx – Порядковый номер выражения.

property is_valid: bool

Проверка на валидность объекта. Слой мог быть удален, как пример, в связи с закрытием таблицы

property max_zoom: float

Максимальная ширина окна, при котором слой отображается на карте. Учитывается только при установленном zoom_restrict=True

property min_zoom: float

Минимальная ширина окна, при котором слой отображается на карте. Учитывается только при установленном zoom_restrict=True

property need_redraw: Signal

Сигнал о необходимости перерисовать слой.

Тип результата
Signal[]

property opacity: int

Прозрачность слоя в составе карты. Доступные значения от 0 до 100.

property orientationType: int

Ориентация относительно центроида.

Таблица 5: Допустимые значения.

Константа	Значение	Описание
CENTER	0	Диаграмма рисуется по центру (по умолчанию)
LEFT_UP	1	Диаграмма выравнивается по левому верхнему краю
UP	2	Диаграмма выравнивается по верхнему краю
RIGHT_UP	3	Диаграмма выравнивается по верхнему правому краю
RIGHT	4	Диаграмма выравнивается по правому краю
RIGHT_DOWN	5	Диаграмма выравнивается по нижнему правому краю
DOWN	6	Диаграмма выравнивается по нижнему краю
LEFT_DOWN	7	Диаграмма выравнивается по нижнему левому краю
LEFT	8	Диаграмма выравнивается по левому краю

property selectable

Управляет доступностью для выбора объектов слоя, если это поддерживается.

set_style(idx: int, style: Style)

Установка стиля оформления для выражения по его индексу в списке выражений.

Параметры

- **idx** – Индекс.
- **style** – Назначаемый стиль.

Список 21: Пример установки стиля для значения с индексом 2 первого тематического слоя.

```
style_new = Style.from_mapinfo("Brush (2, 255, 0)")
world.thematic[0].set_style(2, style_new)
```

property startAngle: bool

Начальный угол отсчета диаграммы.

property title: str

Наименование слоя.

property visible

Управляет видимостью слоя.

Выключение видимости верхнего слоя для активной карты:

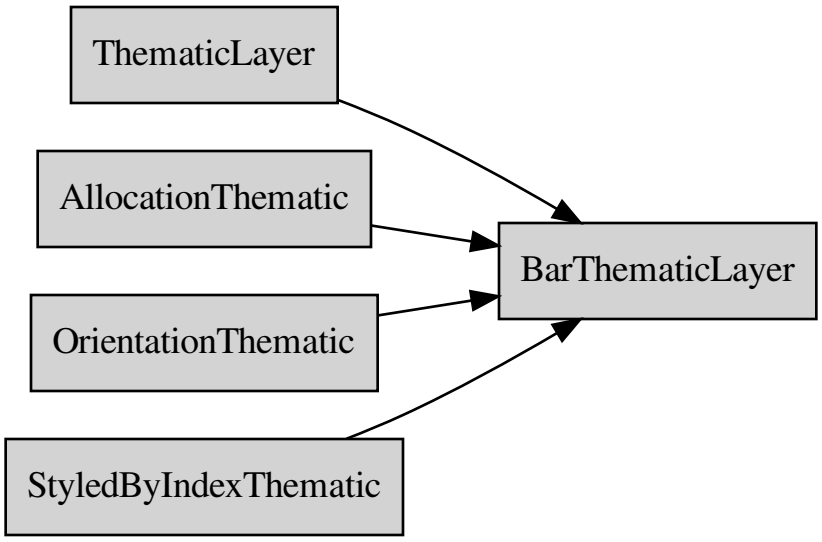
```
if view_manager.active is not None:
    view_manager.active.map.layers[0].visible = False
```

property zoom_restrict: bool

Будет ли использоваться ограничение по отображению. Если установлено True, то для ограничения отображения слоя в зависимости от масштаба используются значения свойств zoom_min и zoom_max

23.7.5 BarThematicLayer - Столбчатые диаграммы

Иерархия классов:



class axipy.**BarThematicLayer**

Базовые классы: ThematicLayer, AllocationThematic, OrientationThematic, StyledByIndexThematic

Тематика в виде столбчатых диаграмм.

Параметры

expressions – Наименования атрибутов или выражений в виде списка list.

Список 22: Создание тематики с последующим добавлением ее к базовому слою.

```
bar = BarThematicLayer(["Население", "Мужское", "Женское"])
# Добавляем к основному слою
world.thematic.add(bar)
```

Классовые методы:

create(dataObject)	Создает слой на базе открытой таблицы или растра.

Свойства:

allocationType	Тип распределения значений.
coordsystem	Координатная система, в которой находятся данные, отображаемые слоем.
data_object	Источник данных для слоя.
isStacked	Расположение столбчатой диаграммы в виде стопки, если True.
is_valid	Проверка на валидность объекта.
max_zoom	Максимальная ширина окна, при котором слой отображается на карте.
min_zoom	Минимальная ширина окна, при котором слой отображается на карте.
opacity	Прозрачность слоя в составе карты.
orientationType	Ориентация относительно центроида.
selectable	Управляет доступностью для выбора объектов слоя, если это поддерживается.
title	Наименование слоя.
visible	Управляет видимостью слоя.
zoom_restrict	Будет ли использоваться ограничение по отображению.

Методы:

get_bounds()	Возвращает область, в которую попадают все данные, которые могут быть отображены на слое.
get_style(idx)	Стиль для указанного выражения.
set_style(idx, style)	Установка стиля оформления для выражения по его индексу в списке выражений.

Сигналы:

data_changed	Сигнал об изменении контента слоя.
need_redraw	Сигнал о необходимости перерисовать слой.

property allocationType: int

Тип распределения значений.

Таблица 6: Допустимые значения.

Константа	Значение	Описание
LINEAR	1	Линейное (по умолчанию)
SQRT	2	Квадратичное
LOG10	3	Логарифмическое

property coordsystem: CoordSystem

Координатная система, в которой находятся данные, отображаемые слоем.

classmethod create(dataObject: DataObject) → Layer

Создает слой на базе открытой таблицы или растра.

Параметры
dataObject – Таблица или растр. В зависимости от переданного объекта будет создан VectorLayer или RasterLayer.

Список 23: Пример создания слоя на базе файла.

```
# Векторный слой
table = provider_manager.openfile(filepath)
vector_layer = Layer.create(table)
# Подпишемся на обновление контента слоя
vector_layer.need_redraw.connect(lambda: print('Update layer'))
```

property data_changed: Signal

Сигнал об изменении контента слоя.

Тип результата
Signal[]

property data_object: DataObject

Источник данных для слоя.

get_bounds() → Rect

Возвращает область, в которую попадают все данные, которые могут быть отображены на слое.

get_style(idx: int) → Style

Стиль для указанного выражения.

Параметры
idx – Порядковый номер выражения.

property isStacked: bool

Расположение столбчатой диаграммы в виде стопки, если True.

property is_valid: bool

Проверка на валидность объекта. Слой мог быть удален, как пример, в связи с закрытием таблицы

property max_zoom: float

Максимальная ширина окна, при котором слой отображается на карте. Учитывается только при установленном zoom_restrict=True

property min_zoom: float

Минимальная ширина окна, при котором слой отображается на карте. Учитывается только при установленном zoom_restrict=True

property need_redraw: Signal

Сигнал о необходимости перерисовать слой.

Тип результата
Signal[]

property opacity: int

Прозрачность слоя в составе карты. Доступные значения от 0 до 100.

property orientationType: int

Ориентация относительно центроида.

Таблица 7: Допустимые значения.

Константа	Значение	Описание
CENTER	0	Диаграмма рисуется по центру (по умолчанию)
LEFT_UP	1	Диаграмма выравнивается по левому верхнему краю
UP	2	Диаграмма выравнивается по верхнему краю
RIGHT_UP	3	Диаграмма выравнивается по верхнему правому краю
RIGHT	4	Диаграмма выравнивается по правому краю
RIGHT_DOWN	5	Диаграмма выравнивается по нижнему правому краю
DOWN	6	Диаграмма выравнивается по нижнему краю
LEFT_DOWN	7	Диаграмма выравнивается по нижнему левому краю
LEFT	8	Диаграмма выравнивается по левому краю

property selectable

Управляет доступностью для выбора объектов слоя, если это поддерживается.

set_style(idx: int, style: Style)

Установка стиля оформления для выражения по его индексу в списке выражений.

Параметры

- **idx** – Индекс.
- **style** – Назначаемый стиль.

Список 24: Пример установки стиля для значения с индексом 2 первого тематического слоя.

```
style_new = Style.from_mapinfo("Brush (2, 255, 0)")
world.thematic[0].set_style(2, style_new)
```

property title: str

Наименование слоя.

property visible

Управляет видимостью слоя.

Выключение видимости верхнего слоя для активной карты:

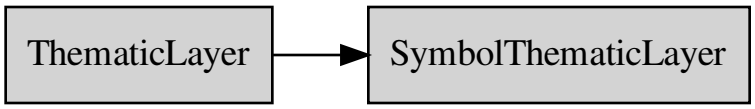
```
if view_manager.active is not None:
    view_manager.active.map.layers[0].visible = False
```

property zoom_restrict: bool

Будет ли использоваться ограничение по отображению. Если установлено True, то для ограничения отображения слоя в зависимости от масштаба используются значения свойств zoom_min и zoom_max

23.7.6 SymbolThematicLayer - Знаки

Иерархия классов:



class axipy.**SymbolThematicLayer**

Базовые классы: ThematicLayer

Тематический слой с распределением по интервалам и с градуировкой символа по размеру.

Параметры

expression – Наименование атрибута или выражение.

Список 25: Создание тематики с последующим добавлением ее к базовому слою.

```
symbol = SymbolThematicLayer("Население")
symbol.defaultStyle = Style.from_mapinfo("Symbol (33, 255,14)")
symbol.maxHeight = 34
world.thematic.add(symbol)
```

Классовые методы:

create(dataObject)	Создает слой на базе открытой таблицы или растра.

Свойства:

coordsystem	Координатная система, в которой находятся данные, отображаемые слоем.
data_object	Источник данных для слоя.
defaultStyle	Стиль по умолчанию для оформления знаков.
is_valid	Проверка на валидность объекта.
maxHeight	Максимальная высота символа.
max_zoom	Максимальная ширина окна, при котором слой отображается на карте.
minHeight	Минимальная высота символа.
min_zoom	Минимальная ширина окна, при котором слой отображается на карте.
opacity	Прозрачность слоя в составе карты.
selectable	Управляет доступностью для выбора объектов слоя, если это поддерживается.
title	Наименование слоя.
visible	Управляет видимостью слоя.
zoom_restrict	Будет ли использоваться ограничение по отображению.

Методы:

get_bounds()	Возвращает область, в которую попадают все данные, которые могут быть отображены на слое.

Сигналы:

data_changed	Сигнал об изменении контента слоя.
need_redraw	Сигнал о необходимости перерисовать слой.

property coordsystem: CoordSystem

Координатная система, в которой находятся данные, отображаемые слоем.

classmethod create(dataObject: DataObject) → Layer

Создает слой на базе открытой таблицы или растра.

Параметры

dataObject – Таблица или растр. В зависимости от переданного объекта будет создан VectorLayer или RasterLayer.

Список 26: Пример создания слоя на базе файла.

```
# Векторный слой
table = provider_manager.openfile(filepath)
vector_layer = Layer.create(table)
# Подпишемся на обновление контента слоя
vector_layer.need_redraw.connect(lambda: print('Update layer'))
```

property data_changed: Signal

Сигнал об изменении контента слоя.

Тип результата
Signal[]

property data_object: DataObject

Источник данных для слоя.

property defaultStyle: Style

Стиль по умолчанию для оформления знаков.

get_bounds() → Rect

Возвращает область, в которую попадают все данные, которые могут быть отображены на слое.

property is_valid: bool

Проверка на валидность объекта. Слой мог быть удален, как пример, в связи с закрытием таблицы

property maxHeight: float

Максимальная высота символа.

property max_zoom: float

Максимальная ширина окна, при котором слой отображается на карте. Учитывается только при установленном zoom_restrict=True

property minHeight: float

Минимальная высота символа.

property min_zoom: float

Минимальная ширина окна, при котором слой отображается на карте. Учитывается только при установленном zoom_restrict=True

property need_redraw: Signal

Сигнал о необходимости перерисовать слой.

Тип результата
Signal[]

property opacity: int

Прозрачность слоя в составе карты. Доступные значения от 0 до 100.

property selectable

Управляет доступностью для выбора объектов слоя, если это поддерживается.

property title: str

Наименование слоя.

property visible

Управляет видимостью слоя.

Выключение видимости верхнего слоя для активной карты:

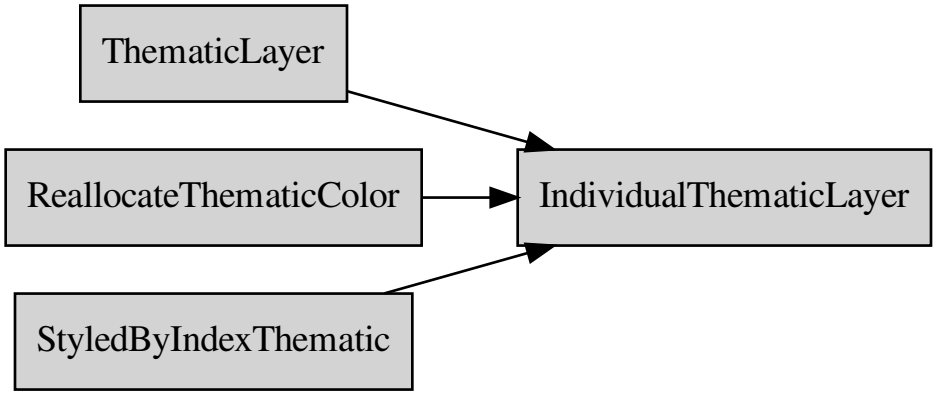
```
if view_manager.active is not None:
    view_manager.active.map.layers[0].visible = False
```

property zoom_restrict: bool

Будет ли использоваться ограничение по отображению. Если установлено True, то для ограничения отображения слоя в зависимости от масштаба используются значения свойств zoom_min и zoom_max

23.7.7 IndividualThematicLayer - Индивидуальные значения

Иерархия классов:



class axipy.**IndividualThematicLayer**

Базовые классы: ThematicLayer, StyledByIndexThematic, ReallocateThematicColor

Тематический слой с распределением стилей по индивидуальным значением.

Параметры

expression – Наименование атрибута или выражение.

Список 27: Создание тематики с последующим добавлением ее к базовому слою.

```
individual = IndividualThematicLayer("Страна")
individual.assign_rainbow()
world.thematic.add(individual)
# Поменяем стиль оформления
individual.set_style(0, PolygonStyle(45, Qt.blue))
```

Классовые методы:

create(dataObject)	Создает слой на базе открытой таблицы или растра.

Свойства:

coordsystem	Координатная система, в которой находятся данные, отображаемые слоем.
count	Количество значений в тематике.
data_object	Источник данных для слоя.
is_valid	Проверка на валидность объекта.
max_zoom	Максимальная ширина окна, при котором слой отображается на карте.
min_zoom	Минимальная ширина окна, при котором слой отображается на карте.
opacity	Прозрачность слоя в составе карты.
selectable	Управляет доступностью для выбора объектов слоя, если это поддерживается.
title	Наименование слоя.
visible	Управляет видимостью слоя.
zoom_restrict	Будет ли использоваться ограничение по отображению.

Методы:

assign_gray([minV, maxV])	Распределение в виде градации серого.
assign_monotone(color[, minv, maxv])	Монотонная заливка разной яркости (оттенки красного, синего и т.п.).
assign_rainbow([sequential, saturation, value])	Распределение цветов по спектру.
assign_three_colors(colorMin, colorMax, ...)	Цвет, распределенный между тремя заданными цветами (с разрывом).
assign_two_colors(colorMin, colorMax[, useHSV])	Равномерно распределяет оформление по заданным крайним цветам.
get_bounds()	Возвращает область, в которую попадают все данные, которые могут быть отображены на слое.
get_style(idx)	Стиль для указанного выражения.
get_value(idx)	Выражение по указанному индексу.
set_style(idx, style)	Установка стиля оформления для выражения по его индексу в списке выражений.

Сигналы:

data_changed	Сигнал об изменении контента слоя.
need_redraw	Сигнал о необходимости перерисовать слой.

assign_gray(minV: int = 20, maxV: int = 80)

Распределение в виде градации серого. Значение задается в интервале (0..100) от черного до белого.

Параметры

- **minV** – Минимальное значение.

- **maxV** – Максимальное значение.

assign_monotone(color: QColor, minv: int = 20, maxv: int = 80)

Монотонная заливка разной яркости (оттенки красного, синего и т.п.). Цветовая схема HSL. Максимальное и минимальное значения задаются в интервале (0..100).

Параметры

- **color** – Базовый цвет.
- **minV** – Минимальное значение.
- **maxV** – Максимальное значение.

assign_rainbow(sequential: bool = True, saturation: float = 90, value: float = 90)

Распределение цветов по спектру. Цветовая схема HSV.

Параметры

- **sequential** – Если True, то последовательное распределение цветов. В противном случае распределение случайно.
- **saturation** – Яркость. Задается в интервале (0..100)
- **value** – Насыщенность. Задается в интервале (0..100)

assign_three_colors(colorMin: QColor, colorMax: QColor, colorBreak: QColor, br: int, useHSV: bool = True)

Цвет, распределенный между тремя заданными цветами (с разрывом).

Параметры

- **colorMin** – Цвет нижнего диапазона.
- **colorMax** – Цвет верхнего диапазона.
- **colorBreak** – Цвет на уровне разрыва.
- **br** – Индекс интервала, на на котором используется цвет разрыва.
- **useHSV** – Если True, то будет использоваться схема HSV. В противном случае - RGB.

assign_two_colors(colorMin: QColor, colorMax: QColor, useHSV: bool = False)

Равномерно распределяет оформление по заданным крайним цветам.

Параметры

- **colorMin** – Цвет нижнего диапазона.
- **colorMax** – Цвет верхнего диапазона.
- **useHSV** – Если True, то будет использоваться схема HSV. В противном случае - RGB.

property coordsystem: CoordSystem

Координатная система, в которой находятся данные, отображаемые слоем.

property count

Количество значений в тематике.

classmethod create(dataObject: DataObject) → Layer

Создает слой на базе открытой таблицы или растра.

Параметры
dataObject – Таблица или растр. В зависимости от переданного объекта будет создан VectorLayer или RasterLayer.

Список 28: Пример создания слоя на базе файла.

```
# Векторный слой
table = provider_manager.openfile(filepath)
vector_layer = Layer.create(table)
# Подпишемся на обновление контента слоя
vector_layer.need_redraw.connect(lambda: print('Update layer'))
```

property data_changed: Signal

Сигнал об изменении контента слоя.

Тип результата
Signal[]

property data_object: DataObject

Источник данных для слоя.

get_bounds() → Rect

Возвращает область, в которую попадают все данные, которые могут быть отображены на слое.

get_style(idx: int) → Style

Стиль для указанного выражения.

Параметры
idx – Порядковый номер выражения.

get_value(idx: int) → Any

Выражение по указанному индексу.

Параметры
idx – Индекс.

property is_valid: bool

Проверка на валидность объекта. Слой мог быть удален, как пример, в связи с закрытием таблицы

property max_zoom: float

Максимальная ширина окна, при котором слой отображается на карте. Учитывается только при установленном zoom_restrict=True

property min_zoom: float

Минимальная ширина окна, при котором слой отображается на карте. Учитывается только при установленном zoom_restrict=True

property need_redraw: Signal

Сигнал о необходимости перерисовать слой.

Тип результата
Signal[]

property opacity: int

Прозрачность слоя в составе карты. Доступные значения от 0 до 100.

property selectable

Управляет доступностью для выбора объектов слоя, если это поддерживается.

set_style(idx: int, style: Style)

Установка стиля оформления для выражения по его индексу в списке выражений.

Параметры

- **idx** – Индекс.
- **style** – Назначаемый стиль.

Список 29: Пример установки стиля для значения с индексом 2 первого тематического слоя.

```
style_new = Style.from_mapinfo("Brush (2, 255, 0)")
world.thematic[0].set_style(2, style_new)
```

property title: str

Наименование слоя.

property visible

Управляет видимостью слоя.

Выключение видимости верхнего слоя для активной карты:

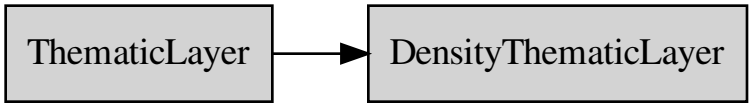
```
if view_manager.active is not None:
    view_manager.active.map.layers[0].visible = False
```

property zoom_restrict: bool

Будет ли использоваться ограничение по отображению. Если установлено True, то для ограничения отображения слоя в зависимости от масштаба используются значения свойств zoom_min и zoom_max

23.7.8 DensityThematicLayer - Плотность точек

Иерархия классов:



class axipy.**DensityThematicLayer**

Базовые классы: ThematicLayer

Тематический слой с заполнением полигональных объектов точками, плотность которых зависит от вычисленного значения по выражению.

Параметры

expression – Наименование атрибута или выражение.

Список 30: Создание тематики с последующим добавлением ее к базовому слою.

```
density = DensityThematicLayer('Население')
density.pointForMaximum = 500
density.color = Qt.red
density.size = 1
world.thematic.add(density)
```

Классовые методы:

create(dataObject)	Создает слой на базе открытой таблицы или растра.

Свойства:

color	Цвет точек.
coordsystem	Координатная система, в которой находятся данные, отображаемые слоем.
data_object	Источник данных для слоя.
is_valid	Проверка на валидность объекта.
max_zoom	Максимальная ширина окна, при котором слой отображается на карте.
min_zoom	Минимальная ширина окна, при котором слой отображается на карте.
opacity	Прозрачность слоя в составе карты.
pointForMaximum	Количество точек для максимального значения.
selectable	Управляет доступностью для выбора объектов слоя, если это поддерживается.
size	Размер точек.
title	Наименование слоя.
visible	Управляет видимостью слоя.
zoom_restrict	Будет ли использоваться ограничение по отображению.

Методы:

get_bounds()	Возвращает область, в которую попадают все данные, которые могут быть отображены на слое.

Сигналы:

data_changed	Сигнал об изменении контента слоя.
need_redraw	Сигнал о необходимости перерисовать слой.

property color: QColor

Цвет точек.

property coordsystem: CoordSystem

Координатная система, в которой находятся данные, отображаемые слоем.

classmethod create(dataObject: DataObject) → Layer

Создает слой на базе открытой таблицы или растра.

Параметры
dataObject – Таблица или растр. В зависимости от переданного объекта будет создан VectorLayer или RasterLayer.

Список 31: Пример создания слоя на базе файла.

```
# Векторный слой
table = provider_manager.openfile(filepath)
vector_layer = Layer.create(table)
# Подпишемся на обновление контента слоя
vector_layer.need_redraw.connect(lambda: print('Update layer'))
```

property data_changed: Signal

Сигнал об изменении контента слоя.

Тип результата
Signal[]

property data_object: DataObject

Источник данных для слоя.

get_bounds() → Rect

Возвращает область, в которую попадают все данные, которые могут быть отображены на слое.

property is_valid: bool

Проверка на валидность объекта. Слой мог быть удален, как пример, в связи с закрытием таблицы

property max_zoom: float

Максимальная ширина окна, при котором слой отображается на карте. Учитывается только при установленном zoom_restrict=True

property min_zoom: float

Минимальная ширина окна, при котором слой отображается на карте. Учитывается только при установленном zoom_restrict=True

property need_redraw: Signal

Сигнал о необходимости перерисовать слой.

Тип результата
Signal[]

property opacity: int

Прозрачность слоя в составе карты. Доступные значения от 0 до 100.

property pointForMaximum: int

Количество точек для максимального значения.

property selectable

Управляет доступностью для выбора объектов слоя, если это поддерживается.

property size: float

Размер точек.

property title: str

Наименование слоя.

property visible

Управляет видимостью слоя.

Выключение видимости верхнего слоя для активной карты:

```
if view_manager.active is not None:
    view_manager.active.map.layers[0].visible = False
```

property zoom_restrict: bool

Будет ли использоваться ограничение по отображению. Если установлено True, то для ограничения отображения слоя в зависимости от масштаба используются значения свойств zoom_min и zoom_max

23.7.9 AllocationThematic - Метод распределения значений для диаграмм

class axipy.**AllocationThematic**

Метод распределения значений для диаграмм.

Свойства:

allocationType	Тип распределения значений.

property allocationType: int

Тип распределения значений.

Таблица 8: Допустимые значения.

Константа	Значение	Описание
LINEAR	1	Линейное (по умолчанию)
SQRT	2	Квадратичное
LOG10	3	Логарифмическое

23.7.10 OrientationThematic - Ориентация для диаграмм

class axipy.**OrientationThematic**

Ориентация тематического представления относительно центроида объекта.

Свойства:

orientationType	Ориентация относительно центроида.

property orientationType: int

Ориентация относительно центроида.

Таблица 9: Допустимые значения.

Константа	Значение	Описание
CENTER	0	Диаграмма рисуется по центру (по умолчанию)
LEFT_UP	1	Диаграмма выравнивается по левому верхнему краю
UP	2	Диаграмма выравнивается по верхнему краю
RIGHT_UP	3	Диаграмма выравнивается по верхнему правому краю
RIGHT	4	Диаграмма выравнивается по правому краю
RIGHT_DOWN	5	Диаграмма выравнивается по нижнему правому краю
DOWN	6	Диаграмма выравнивается по нижнему краю
LEFT_DOWN	7	Диаграмма выравнивается по нижнему левому краю
LEFT	8	Диаграмма выравнивается по левому краю

23.7.11 StyledByIndexThematic - Стиль заливки

class axipy.**StyledByIndexThematic**

Поддержка набора индексированных стилей.

Методы:

get_style(idx)	Стиль для указанного выражения.
set_style(idx, style)	Установка стиля оформления для выражения по его индексу в списке выражений.

get_style(idx: int) → Style

Стиль для указанного выражения.

Параметры
idx – Порядковый номер выражения.

set_style(idx: int, style: Style)

Установка стиля оформления для выражения по его индексу в списке выражений.

Параметры

- **idx** – Индекс.
- **style** – Назначаемый стиль.

Список 32: Пример установки стиля для значения с индексом 2 первого тематического слоя.

```
style_new = Style.from_mapinfo("Brush (2, 255, 0)")
world.thematic[0].set_style(2, style_new)
```

23.8 Отчет

23.8.1 Report - Отчет

class axipy.**Report**

План отчета для последующей печати.

Список 33: Пример создания пустого отчета и вывод его в pdf.

```
printer = QPrinter()
printer.setPageSize(QPageSize(QPageSize.A4))
printer.setOutputFormat(QPrinter.PdfFormat)
printer.setOutputFileName(filepath)
painterReport = QPainter(printer)
contextReport = Context(painterReport)
report = Report(printer)
report.horisontal_pages = 2
# Здесь добавляются элементы отчета
report.draw(contextReport)
```

Свойства:

horisontal_pages	Количество страниц отчета по горизонтали.
items	Элементы отчета.
name	Наименование отчета.
page_size	Размеры одного листа отчета.
unit	Единицы измерения в отчете.
vertical_pages	Количество страниц отчета по вертикали.

Методы:

draw(context)	Выводит отчета в заданном контексте.
fill_on_pages()	Максимально заполняет страницу(ы) отчета.
fit_pages()	Подгоняет число страниц отчета под размер существующих элементов отчета.

Сигналы:

need_redraw	Сигнал о необходимости перерисовки части или всего отчета.

draw(context: Context)

Выводит отчета в заданном контексте.

Параметры
context – Контекст, в котором будет отрисован отчет.

fill_on_pages()

Максимально заполняет страницу(ы) отчета. При этом элементы отчета пропорционально масштабируются.

fit_pages()

Подгоняет число страниц отчета под размер существующих элементов отчета. При этом параметры элементов отчета не меняются.

property horisontal_pages: int

Количество страниц отчета по горизонтали.

property items: ReportItems

Элементы отчета.

property name: str

Наименование отчета.

property need_redraw: Signal

Сигнал о необходимости перерисовки части или всего отчета.

Параметры
rect – Часть отчета, которую необходимо обновить.

Тип результата
Signal[QRectF]

property page_size: QSizeF

Размеры одного листа отчета.

property unit: LinearUnit

Единицы измерения в отчете.

property vertical_pages: int

Количество страниц отчета по вертикали.

23.8.2 ReportItems - Список элементов отчета

class axipy.**ReportItems**

Список элементов отчета.

Свойства:

count	Количество элементов отчета в текущем отчете на данный момент.

Методы:

add(item)	Добавляет новый элемент в отчет.
at(idx)	Возвращает элемент отчета по его индексу.
remove(idx)	Удаляет элемент по его индексу.

add(item: ReportItem)

Добавляет новый элемент в отчет.

Параметры
item – Вставляемый элемент

at(idx: int) → ReportItem

Возвращает элемент отчета по его индексу.

Параметры
idx – Индекс.

Результат
Элемент отчета. Возвращает None в случае, если не найдено.

property count: int

Количество элементов отчета в текущем отчете на данный момент.

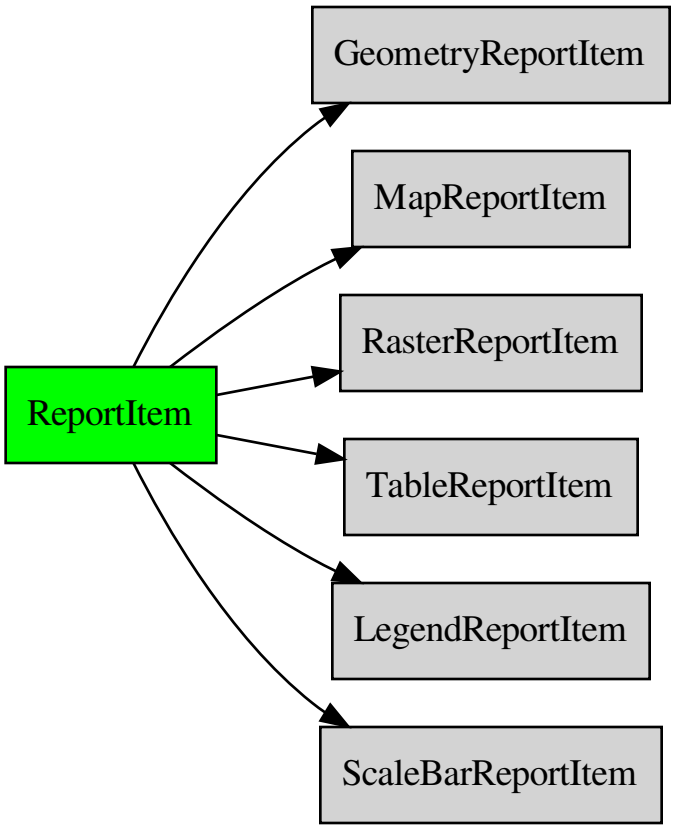
remove(idx: int)

Удаляет элемент по его индексу. Если индекс корректен, элемент будет удален.

Параметры
idx – Индекс удаляемого элемента.

23.8.3 ReportItem - Элемент отчета

Элементы отчета:



class axipy.**ReportItem**

Базовый класс элемента отчета.

Свойства:

border_style	Стиль обводки элемента отчета.
fill_style	Стиль заливки элемента отчета.
rect	Размер (ограничивающий прямоугольник) элемента отчета в единицах измерения отчета.

Методы:

intersects(checkRect)	Пересекается ли с переданным прямоугольником.

property border_style: Style

Стиль обводки элемента отчета.

property fill_style: Style

Стиль заливки элемента отчета.

intersects(checkRect: Union[Rect, QRectF])

Пересекается ли с переданным прямоугольником.

Параметры
checkRect – Прямоугольник для анализа.

property rect: Rect

Размер (ограничивающий прямоугольник) элемента отчета в единицах измерения отчета.

23.8.4 GeometryReportItem - Элемент отчета: геометрия

class axipy.**GeometryReportItem**

Базовые классы: ReportItem

Элемент отчета типа геометрия.

Список 34: Пример создания полигона и добавления его в отчет.

```
geomItem = GeometryReportItem()
geomItem.geometry = Polygon((10, 10), (10, 100), (100, 100), (10, 10))
geomItem.style = PolygonStyle(45, Qt.red)
report.items.add(geomItem)
```

Список 35: Пример создания текста и добавления его в отчет.

```
r = Rect(8, 6, 14, 7)
txt = Text("Пример текста", r)
txt.angle = 20
style = Style.from_mapinfo('Font ("Times New Roman", 512, 0, 16711680, 16776960)')
geomItem = GeometryReportItem()
geomItem.style = style
geomItem.geometry = txt
report.items.add(geomItem)
```

Свойства:

border_style	Стиль обводки элемента отчета.
fill_style	Стиль заливки элемента отчета.
geometry	Геометрическое представление объекта.
rect	Размер (ограничивающий прямоугольник) элемента отчета в единицах измерения отчета.
style	Стиль геометрического представления объекта.

Методы:

intersects(checkRect)	Пересекается ли с переданным прямоугольником.

property border_style: Style

Стиль обводки элемента отчета.

property fill_style: Style

Стиль заливки элемента отчета.

property geometry: Geometry

Геометрическое представление объекта.

intersects(checkRect: Union[Rect, QRectF])

Пересекается ли с переданным прямоугольником.

Параметры
checkRect – Прямоугольник для анализа.

property rect: Rect

Размер (ограничивающий прямоугольник) элемента отчета в единицах измерения отчета.

property style: Style

Стиль геометрического представления объекта.

23.8.5 MapReportItem - Элемент отчета: карта

class axipy.**MapReportItem**

Базовые классы: ReportItem

Элемент отчета, основанный на созданной ранее карте.

Примечание: Перед созданием элемента отчета необходимо предварительно создать карту, на основе которой будет создан элемент отчета.

Параметры

- **rect** – Размер элемента отчета в единицах измерения отчета.
- **map** – Карта, на базе которой будет создан элемент отчета.

Список 36: Пример создания карты и добавления ее в отчет.

```
map_ = Map([world])
mapItem = MapReportItem(Rect(10, 110, 200, 210), map_)
mapItem.center = (100, 100)
mapItem.scale = 200000000
report.items.add(mapItem)
```

Конструктор класса:

__init__(rect, map)	Создает экземпляр класса.

Свойства:

border_style	Стиль обводки элемента отчета.
center	Центр карты в координатах карты.
fill_style	Стиль заливки элемента отчета.
map_rect	Прямоугольник карты в единицах измерения карты.
rect	Размер (ограничивающий прямоугольник) элемента отчета в единицах измерения отчета.
scale	Текущее значение масштаба карты.

Методы:

intersects(checkRect)	Пересекается ли с переданным прямоугольником.
map()	Возвращает элемент типа карта, на основании которой создается элемент отчета.
show_all()	Меняет масштаб карты чтобы показать ее полностью.

__init__(rect: Union[Rect, QRectF], map: Map)

Создает экземпляр класса.

property border_style: Style

Стиль обводки элемента отчета.

property center: Pnt

Центр карты в координатах карты.

property fill_style: Style

Стиль заливки элемента отчета.

intersects(checkRect: Union[Rect, QRectF])

Пересекается ли с переданным прямоугольником.

Параметры
checkRect – Прямоугольник для анализа.

map() → Map

Возвращает элемент типа карта, на основании которой создается элемент отчета.

property map_rect: Rect

Прямоугольник карты в единицах измерения карты.

property rect: Rect

Размер (ограничивающий прямоугольник) элемента отчета в единицах измерения отчета.

property scale: float

Текущее значение масштаба карты.

show_all()

Меняет масштаб карты чтобы показать ее полностью.

Пример замены масштаба для всех элементов отчета:

```
for item in reportView.report.items:
    if isinstance(item, MapReportItem):
        item.show_all()
```

23.8.6 RasterReportItem - Элемент отчета: растр

class axipy.**RasterReportItem**

Базовые классы: ReportItem

Элемент отчета, основанный на растре.

Примечание: В качестве источника может быть как локальный файл, расположенный в файловой системе, так и базе растра, размещенного на Web ресурсе.

Параметры

- **rect** – Размер элемента отчета в единицах измерения отчета.
- **data** – Путь к растровому файлу или его URL.

Список 37: Пример элемента на базе URL.

```
report = create_report()
rasterReportItem = RasterReportItem(
    Rect(10, 10, 140, 70),
    'https://upload.wikimedia.org/wikipedia/commons/thumb/3/34/Gall%E2%80
↪%93Peters_projection_SW.jpg'
    '/1280px-Gall%E2%80%93Peters_projection_SW.jpg'
)
report.items.add(rasterReportItem)
```

Список 38: Пример элемента на базе локального файла.

```
rasterReportItem = RasterReportItem(Rect(10, 10, 140, 70), filename)
report.items.add(rasterReportItem)
```

Конструктор класса:

__init__(rect, data)	Создает экземпляр класса.

Свойства:

border_style	Стиль обводки элемента отчета.
fill_style	Стиль заливки элемента отчета.
preserve_aspect_ratio	Сохранять пропорции при изменении размеров элемента.
rect	Размер (ограничивающий прямоугольник) элемента отчета в единицах измерения отчета.

Методы:

intersects(checkRect)	Пересекается ли с переданным прямоугольником.

__init__(rect: Union[Rect, QRectF], data: str)

Создает экземпляр класса.

property border_style: Style

Стиль обводки элемента отчета.

property fill_style: Style

Стиль заливки элемента отчета.

intersects(checkRect: Union[Rect, QRectF])

Пересекается ли с переданным прямоугольником.

Параметры
checkRect – Прямоугольник для анализа.

property preserve_aspect_ratio: bool

Сохранять пропорции при изменении размеров элемента.

property rect: Rect

Размер (ограничивающий прямоугольник) элемента отчета в единицах измерения отчета.

23.8.7 TableReportItem - Элемент отчета: таблица

class axipy.**TableReportItem**

Базовые классы: ReportItem

Элемент отчета табличного представления данных.

Примечание: Позволяет отображать как таблицу целиком, так и накладывая дополнительные ограничения при отображении.

Параметры

- **rect** – Размер элемента отчета в единицах измерения отчета.
- **table** – Таблица.

Список 39: Пример.

```
table = provider_manager.openfile(filename)
tableReportItem = TableReportItem(Rect(210, 150, 480, 100), table)
tableReportItem.columns = table.schema.attribute_names[:3]  # Берем для показа␣
↪первые три атрибута
tableReportItem.row_from = 5  # С 5-й сторки
tableReportItem.row_count = 4  # Показываем 4 строки
tableReportItem.start_number = 5  # Нумерация с 5
tableReportItem.border_style = LineStyle(3, Qt.red)  # Стиль рамки
tableReportItem.fill_style = PolygonStyle(8, 65535)  # Стиль фона
report.items.add(tableReportItem)
```

Конструктор класса:

`__init__`(rect, table)	Создает экземпляр класса.

Свойства:

`border_style`	Стиль обводки элемента отчета.
`columns`	Перечень наименований для отображения.
`fill_style`	Стиль заливки элемента отчета.
`rect`	Размер (ограничивающий прямоугольник) элемента отчета в единицах измерения отчета.
`row_count`	Количество записей.
`row_from`	Номер первой строки из таблицы или запроса.
`show_row_number`	Показывать ли номера строк.
`start_number`	Нумерация записей.

Методы:

`intersects`(checkRect)	Пересекается ли с переданным прямоугольником.
`refreshValues`()	"Обновление данных из таблицы.
`table`()	Базовая таблица или запрос.

__init__(rect: Union[Rect, QRectF], table: Table)

Создает экземпляр класса.

property border_style: Style

Стиль обводки элемента отчета.

property columns: list

Перечень наименований для отображения. Если задать пустой список, будут отображены все поля таблицы.

property fill_style: Style

Стиль заливки элемента отчета.

intersects(checkRect: Union[Rect, QRectF])

Пересекается ли с переданным прямоугольником.

Параметры
checkRect – Прямоугольник для анализа.

property rect: Rect

Размер (ограничивающий прямоугольник) элемента отчета в единицах измерения отчета.

refreshValues()

«Обновление данных из таблицы.

property row_count: int

Количество записей. Если указано -1, то берутся все оставшиеся записи.

property row_from: int

Номер первой строки из таблицы или запроса.

property show_row_number: bool

Показывать ли номера строк.

property start_number: int

Нумерация записей. Порядковый номер первой записи.

table() → Table

Базовая таблица или запрос.

23.8.8 LegendReportItem - Элемент отчета: легенда

class axipy.**LegendReportItem**

Базовые классы: ReportItem

Элемент отчета, основанный на легенде векторного или тематического слоя.

Параметры

- **rect** – Размер элемента отчета в единицах измерения отчета.
- **legend** – Предварительно созданная легенда. Она может относиться как к векторному, так и к тематическому слою.

Список 40: Пример создания легенды для тематического слоя.

```
range_ = RangeThematicLayer("Население")
world.thematic.add(range_)
legend = Legend(range_)
legend.columns = 2  # Разобьем на 2 колонки
legendReportItem = LegendReportItem(Rect(100, 230, 50, 70), legend)  # Элемент␣
↪отчета
report.items.add(legendReportItem)  # Добавляем в отчет
```

Конструктор класса:

__init__(rect, legend)	Создает экземпляр класса.

Свойства:

border_style	Стиль обводки элемента отчета.
fill_style	Стиль заливки элемента отчета.
legend	Легенда на базе которой создан элемент отчета.
rect	Размер (ограничивающий прямоугольник) элемента отчета в единицах измерения отчета.

Методы:

intersects(checkRect)	Пересекается ли с переданным прямоугольником.

__init__(rect: Union[Rect, QRectF], legend: Legend)

Создает экземпляр класса.

property border_style: Style

Стиль обводки элемента отчета.

property fill_style: Style

Стиль заливки элемента отчета.

intersects(checkRect: Union[Rect, QRectF])

Пересекается ли с переданным прямоугольником.

Параметры
checkRect – Прямоугольник для анализа.

property legend: Legend

Легенда на базе которой создан элемент отчета.

property rect: Rect

Размер (ограничивающий прямоугольник) элемента отчета в единицах измерения отчета.

23.8.9 ScaleBarReportItem - Элемент отчета: масштабная линейка

class axipy.**ScaleBarReportItem**

Базовые классы: ReportItem

Элемент отчета - масштабная линейка для карты.

Список 41: Пример создания масштабной линейки на базе существующего элемента - карты.

```
scaleBarReportItem = ScaleBarReportItem(Rect(120, 130, 80, 50), mapItem)
report.items.add(scaleBarReportItem)
```

Конструктор класса:

__init__(rect, map)	Создает экземпляр класса.

Свойства:

border_style	Стиль обводки элемента отчета.
fill_style	Стиль заливки элемента отчета.
rect	Размер (ограничивающий прямоугольник) элемента отчета в единицах измерения отчета.

Методы:

intersects(checkRect)	Пересекается ли с переданным прямоугольником.

__init__(rect: Union[Rect, QRectF], map: MapReportItem)

Создает экземпляр класса.

property border_style: Style

Стиль обводки элемента отчета.

property fill_style: Style

Стиль заливки элемента отчета.

intersects(checkRect: Union[Rect, QRectF])

Пересекается ли с переданным прямоугольником.

Параметры
checkRect – Прямоугольник для анализа.

property rect: Rect

Размер (ограничивающий прямоугольник) элемента отчета в единицах измерения отчета.

23.9 Context - Контекст рисования

class axipy.**Context**

Контекст рисования.

Содержит информацию о том, куда производится рисование (QPainter), а так же о необходимых преобразованиях, которые необходимо применить к объекту непосредственно перед его отрисовкой.

Параметры
painter – Объект QPainter для рисования.

Пример создания контекста на базе растра. Далее его можно использовать для отрисовки карты Map, отчета Report или легенды Legend:

```
image = QImage(1600, 800, QImage.Format_ARGB32_Premultiplied)
image.fill(Qt.white)
painter = QPainter(image)
context = Context(painter)
```

Свойства:

coordsystem	Координатная система.
dpi	Количество точек на дюйм, с которым происходит рисование.
rect	Прямоугольник в координатах карты, который будет отрисован.

property coordsystem: CoordSystem

Координатная система.

Если она не задана, берется наиболее подходящая исходя из текущего контента.

property dpi: float

Количество точек на дюйм, с которым происходит рисование.

Влияет на отрисовку в «реальных» единицах измерения (мм, см, пункты).

property rect: Rect

Прямоугольник в координатах карты, который будет отрисован.

23.10 CustomLabels - Пользовательские метки карты

class axipy.**CustomLabels**

Пользовательские метки. Используется для задания параметров через свойство Map.custom_labels.

Методы:

get(layer, id)	Производит запрос параметров.
set(layer, id, properties)	Устанавливает параметры.

get(layer: VectorLayer, id: int) → Optional[CustomLabelProperties]

Производит запрос параметров. Если для данного id не определены, возвращает None.

Параметры

- **layer** – Слой карты
- **id** – Идентификатор записи

set(layer: VectorLayer, id: int, properties: Optional[CustomLabelProperties])

Устанавливает параметры.

Параметры

- **layer** – Слой карты
- **id** – Идентификатор записи
- **properties** – Устанавливаемые свойства. Если задать None, существующие параметры будут сброшены

24

axipy.gui - Модуль пользовательского интерфейса.

Модуль пользовательского интерфейса.

В данном модуле содержатся классы связанные с пользовательским интерфейсом.

24.1 MapTool - Инструмент окна карты

class axipy.**MapTool**

Инструмент окна карты. При создании своего инструмента новый инструмент наследуется от этого класса, и переопределяет необходимые обработчики событий.

См.также:

axipy.ObserverManager.

Пример:

```
MyTool(MapTool):

    def mousePressEvent(self, event):
        print('mouse pressed')
        return self.PassEvent
```

Классовые методы:

reset()	Переключает текущий инструмент на инструмент по умолчанию.

Свойства:

cursor	Текущий курсор для данного инструмента.
view	Отображение данных в окне.

Атрибуты:

BlockEvent	Прекратить обработку события.
PassEvent	Передать событие дальше.
enable_on	Идентификатор наблюдателя для определения доступности инструмента.

Методы:

canDeactivate(reason)	Обрабатывает причину выключения инструмента.
canUnload(reason)	Обрабатывает причину выключения инструмента.
deactivate()	Выполняет действия непосредственно перед выключением инструмента и перед его удалением.
get_select_rect(device[, size])	Возвращает прямоугольник в координатах карты для точки на экране.
handleEvent(event)	Первичный обработчик всех событий инструмента.
is_snapped()	Проверяет, сработала ли привязка к элементам карты или отчета для текущего положения указателя мыши.
keyPressEvent(event)	Обрабатывает событие нажатия клавиши клавиатуры.
keyReleaseEvent(event)	Обрабатывает событие отпускания клавиши клавиатуры.
load()	Выполняет действия непосредственно перед включением инструмента.
mouseDoubleClickEvent(event)	Обрабатывает событие двойного клика мыши.
mouseMoveEvent(event)	Обрабатывает событие перемещения мыши.
mousePressEvent(event)	Обрабатывает событие нажатия клавиши мыши.
mouseReleaseEvent(event)	Обрабатывает событие отпускания клавиши мыши.
paintEvent(event, painter)	Обрабатывает событие отрисовки.
redraw()	Перерисовывает окно карты.
snap([default_value])	Возвращает исправленные координаты, если сработала привязка к элементам в единицах измерения карты или отчета.
snap_device([default_value])	Возвращает исправленные координаты, если сработала привязка к элементам в единицах измерения окна карты (виджета).
to_device(scene)	Переводит точки из координат на карте в координаты окна(пиксели).
to_scene(device)	Переводит точки из координат окна(пикселей) в координаты на карте.
unload()	Выполняет действия непосредственно перед выключением инструмента и перед его удалением.
wheelEvent(event)	Обрабатывает событие колеса мыши.

BlockEvent

Прекратить обработку события. Значение True.

PassEvent

Передать событие дальше. Значение False.

canDeactivate(reason: DeactivationReason) → bool

Обрабатывает причину выключения инструмента.

Переопределите этот метод, чтобы задать свой обработчик.

Параметры
reason – причина выключения.

Результат
False чтобы прервать выключение, иначе True.

Предупреждение: Не рекомендуется, начиная с версии 3.6: Используйте `canUnload()`.

canUnload(reason: DeactivationReason) → bool

Обрабатывает причину выключения инструмента.

Переопределите этот метод, чтобы задать свой обработчик.

Параметры
reason – причина выключения.

Результат
False чтобы прервать выключение, иначе True.

property cursor: QCursor

Текущий курсор для данного инструмента.

Первоначально курсор для инструмента можно установить, переопределив метод `load()`:

```
class MyTool(MapTool):
    def load(self):
        self.cursor = QCursor(Qt.SizeAllCursor)
```

Если же требуется устанавливать различный типы курсора в зависимости от статуса нажатия ПКМ, следует переопределить методы `mousePressEvent()` и `mouseReleaseEvent()` и установить нужное значение там:

```
class MyTool(MapTool):
    def mousePressEvent(self, event) -> bool:
        if event.button() == Qt.LeftButton:
            self.cursor = QCursor(Qt.SizeAllCursor)
```

deactivate()

Выполняет действия непосредственно перед выключением инструмента и перед его удалением.

Предупреждение: Не рекомендуется, начиная с версии 3.6: Используйте `unload()`.

enable_on

Идентификатор наблюдателя для определения доступности инструмента. По умолчанию отсутствует.

get_select_rect(device: QPoint, size: int = 3) → Rect

Возвращает прямоугольник в координатах карты для точки на экране. Удобно для использования при поиске объектов.

Параметры

- **device** – Точка в координатах окна.
- **size** – Размер квадрата в пикселях.

Результат

Прямоугольник в координатах карты.

Пример:

```
device_point = event.pos()
bbox = self.get_select_rect(device_point, 30)
features = table.items(bbox=bbox)
```

handleEvent(event: QEvent) → Optional[bool]

Первичный обработчик всех событий инструмента.

Если событие не блокируется этим обработчиком, то оно будет передано дальше в соответствующий специализированный обработчик `mousePressEvent()`, `keyReleaseEvent()` и прочие в зависимости от типа.

Параметры

event – Событие.

Результат

`BlockEvent`, чтобы блокировать дальнейшую обработку события.

is_snapped() → bool

Проверяет, сработала ли привязка к элементам карты или отчета для текущего положения указателя мыши.

См.также:

`snap()`, `snap_device()`.

keyPressEvent(event: QKeyEvent) → Optional[bool]

Обрабатывает событие нажатия клавиши клавиатуры.

Параметры

event – Событие нажатия клавиши клавиатуры.

Результат

`PassEvent`, чтобы пропустить событие дальше по цепочке обработчиков. `None` или `BlockEvent`, чтобы блокировать дальнейшую обработку события.

keyReleaseEvent(event: QKeyEvent) → Optional[bool]

Обрабатывает событие отпускания клавиши клавиатуры.

Параметры

event – Событие отпускания клавиши клавиатуры.

Результат

`PassEvent`, чтобы пропустить событие дальше по цепочке обработчиков. `None` или `BlockEvent`, чтобы блокировать дальнейшую обработку события.

load()

Выполняет действия непосредственно перед включением инструмента.

Переопределите этот метод, чтобы задать свои действия.

См.также:

unload().

mouseDoubleClickEvent(event: QMouseEvent) → Optional[bool]

Обрабатывает событие двойного клика мыши.

Параметры
event – Событие двойного клика мыши.

Результат
PassEvent, чтобы пропустить событие дальше по цепочке обработчиков. None или BlockEvent, чтобы блокировать дальнейшую обработку события.

mouseMoveEvent(event: QMouseEvent) → Optional[bool]

Обрабатывает событие перемещения мыши.

Параметры
event – Событие перемещения мыши.

Результат
PassEvent или None, чтобы пропустить событие дальше по цепочке обработчиков. BlockEvent, чтобы блокировать дальнейшую обработку события.

mousePressEvent(event: QMouseEvent) → Optional[bool]

Обрабатывает событие нажатия клавиши мыши.

Параметры
event – Событие нажатия клавиши мыши.

Результат
PassEvent, чтобы пропустить событие дальше по цепочке обработчиков. None или BlockEvent, чтобы блокировать дальнейшую обработку события.

mouseReleaseEvent(event: QMouseEvent) → Optional[bool]

Обрабатывает событие отпускания клавиши мыши.

Параметры
event – Событие отпускания клавиши мыши.

Результат
PassEvent, чтобы пропустить событие дальше по цепочке обработчиков. None или BlockEvent, чтобы блокировать дальнейшую обработку события.

paintEvent(event: QPaintEvent, painter: QPainter)

Обрабатывает событие отрисовки.

Параметры

- **event** – Событие отрисовки.
- **painter** – QPainter для рисования поверх виджета

redraw()

Перерисовывает окно карты.

Создает событие PySide2.QtGui.QPaintEvent и помещает его в очередь обработки событий. Аналогично PySide2.QtWidgets.QWidget.update().

static reset()

Переключает текущий инструмент на инструмент по умолчанию.

Обычно инструментом по умолчанию является Выбор.

snap(default_value: Optional[Pnt] = None) → Optional[Pnt]

Возвращает исправленные координаты, если сработала привязка к элементам в единицах измерения карты или отчета.

Параметры
default_value – Значение по умолчанию.

Возвращает значение по умолчанию, если не сработала привязка к элементам карты или отчета.

Пример:

```
point = self.to_scene(event.pos())
current_point = self.snap(point)
```

См.также:

is_snapped(), snap_device(), to_scene().

snap_device(default_value: Optional[QPoint] = None) → Optional[QPoint]

Возвращает исправленные координаты, если сработала привязка к элементам в единицах измерения окна карты (виджета).

Параметры
default_value – Значение по умолчанию.

Возвращает значение по умолчанию, если не сработала привязка к элементам карты или отчета.

Пример:

```
device_point = event.pos()
current_device_point = self.snap_device(device_point)
```

См.также:

is_snapped(), snap().

to_device(scene: Union[Pnt, Rect]) → Union[QPoint, QRect]

Переводит точки из координат на карте в координаты окна(пиксели).

Параметры
scene – Точки в координатах карты.

Результат
Точки в координатах окна.

to_scene(device: Union[QPoint, QRect]) → Union[Pnt, Rect]

Переводит точки из координат окна(пикселей) в координаты на карте.

Параметры
device – Точки в координатах окна.

Результат
Точки в координатах карты.

unload()

Выполняет действия непосредственно перед выключением инструмента и перед его удалением.

Переопределите этот метод, чтобы задать свои действия.

См.также:

load().

property view: MapView

Отображение данных в окне.

wheelEvent(event: QWheelEvent) → Optional[bool]

Обрабатывает событие колеса мыши.

Параметры
event – Событие колеса мыши.

Результат
PassEvent, чтобы пропустить событие дальше по цепочке обработчиков. None или BlockEvent, чтобы блокировать дальнейшую обработку события.

class axipy.**DeactivationReason**

Причина выключения инструмента.

Атрибуты:

ActionClick	Нажатие на действие
ActionShortcut	Вызов действия комбинацией клавиш
LayerClick	Нажатие на свойства слоя
ObjectClose	Закрытие объекта данных
Unknown	Не определено
WindowClose	Закрытие окна

24.2 ActiveToolPanel - Панель активного инструмента

class axipy.**ActiveToolPanel**

Сервис предоставляющий доступ к панели активного инструмента.

Список 1: Пример использования.

```
service = ActiveToolPanel()
# Любой пользовательский графический элемент
widget = QWidget()

# Создаём обработчик для панели активного инструмента через который
# будем управлять панелью.
```

(continues on next page)

(продолжение с предыдущей страницы)

```
tool_panel = service.make_acceptable(
    title="Мой инструмент",
    observer_id=DefaultKeys.SelectionEditable,
    widget=widget)

# Подписываемся на сигнал отправляемый после нажатия на кнопку "Применить" в␣
↪панели
tool_panel.accepted.connect(lambda: print("Применяем изменения"))
```

Чтобы отобразить переданный ранее графический элемент нужно вызвать `activate()`. Например при нажатии на пользовательскую кнопку.

Панель активного инструмента созданная через `make_acceptable()` по умолчанию содержит кнопку «Применить». По нажатию на эту кнопку отсылается сигнал `accepted()` который можно обработать в пользовательском инструменте.

Панель активного инструмента созданная через `make_custom()` представляет из себя пустой контейнер в который можно поместить пользовательский графический элемент. Это дает больше свободы для реализации управления панелью активного инструмента.

Переданный идентификатор наблюдателя используется для управления видимостью и доступностью панели. Панель активного инструмента сразу закроется как только наблюдатель вернет False.

Методы:

`make_acceptable`(title, observer_id[, widget])	Создает экземпляр обработчика через который можно взаимодействовать с панелью активного инструмента.
`make_custom`(title, observer_id[, widget])	Создает экземпляр обработчика панели активного инструмента.

make_acceptable(title: str, observer_id: Observer, widget: Optional[QWidget] = None) → AxipyAcceptableActiveToolHandler

Создает экземпляр обработчика через который можно взаимодействовать с панелью активного инструмента. По умолчанию добавляются кнопки «Применить/Отменить».

Параметры

- **title** – Заголовок панели активного инструмента. Обычно это название инструмента.
- **observer_id** – Идентификатор наблюдателя для управления видимостью и доступностью.
- **widget** – Пользовательский виджет который будет отображаться в панели активного инструмента.

make_custom(title: str, observer_id: Observer, widget: Optional[QWidget] = None) → AxipyCustomActiveToolPanelHandler

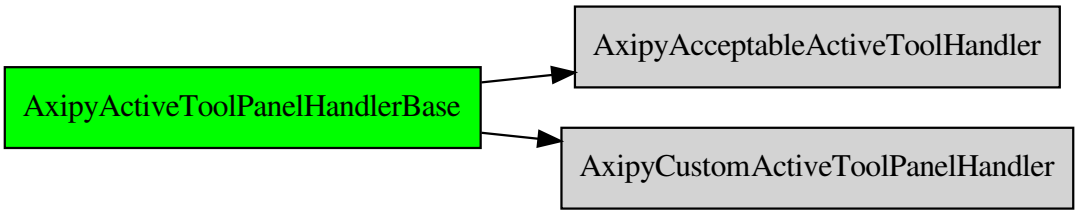
Создает экземпляр обработчика панели активного инструмента. В который можно установить любой пользовательский графический элемент. Используется когда пользователю не нужны предустановленные элементы управления.

Параметры

- **title** – Заголовок панели активного инструмента. Обычно это название инструмента.
- **observer_id** – Идентификатор наблюдателя для управления видимостью и доступностью.
- **widget** – Пользовательский виджет который будет отображаться в панели активного инструмента.

24.3 AxipyActiveToolPanelHandlerBase - Базовый класс обработчика панели активного инструмента

Схема наследования



class axipy.**AxipyActiveToolPanelHandlerBase**

Базовый класс обработчика панели активного инструмента.

Свойства:

widget	Возвращает пользовательский графический элемент.

Методы:

activate()	Показывает пользовательский графический элемент в панели активного инструмента.
deactivate()	Скрывает пользовательский графический элемент из панели активного инструмента.
set_observer(observer_id)	Метод устанавливает наблюдателя.
set_panel_title(title)	Устанавливает заголовок панели активного инструмента.
set_widget(widget)	Пользовательский графический элемент будет помещен в панель активного инструмента при активации обработчика.

Сигналы:

activated	Сигнал испускается когда обработчик панели активного инструмента становится активным.
deactivated	Сигнал испускается перед тем как обработчик панели активного инструмента перестает быть активным.
panel_was_closed	Сигнал испускается после закрытия панели активного инструмента.

activate()

Показывает пользовательский графический элемент в панели активного инструмента.

property activated: Signal

Сигнал испускается когда обработчик панели активного инструмента становится активным.

Тип результата
Signal[]

deactivate()

Скрывает пользовательский графический элемент из панели активного инструмента.

property deactivated: Signal

Сигнал испускается перед тем как обработчик панели активного инструмента перестает быть активным.

Тип результата
Signal[]

property panel_was_closed: Signal

Сигнал испускается после закрытия панели активного инструмента.

Тип результата
Signal[]

set_observer(observer_id: Observer)

Метод устанавливает наблюдателя. Если наблюдатель сигнализирует, что условия доступности кнопки нарушены, то панель активного инструмента сразу же закроется.

Параметры
observer_id – Идентификатор наблюдателя для управления видимостью и доступностью

См.также:

Наблюдатели за состоянием инструмента axipy.ObserverManager

set_panel_title(title: str)

Устанавливает заголовок панели активного инструмента.

Параметры
title – Новый заголовок.

set_widget(widget: QWidget)

Пользовательский графический элемент будет помещен в панель активного инструмента при активации обработчика. Владение графическим элементом передаётся обработчику. Это значит, что не следует использовать и сохранять где-либо ссылку на этот объект. Для получения графического элемента обратно используйте `widget()`.

property widget: QWidget

Возвращает пользовательский графический элемент.

Результат

Переданный ранее пользовательский графический элемент.

24.4 AxipyAcceptableActiveToolHandler - Управление панелью активного инструмента с предустановленными кнопками

class axipy.**AxipyAcceptableActiveToolHandler**

Базовые классы: AxipyActiveToolPanelHandlerBase

Обработчик панели активного инструмента, который предоставляет по умолчанию кнопку Применить. При нажатии на эту кнопку испускается сигнал `axipy.AxipyAcceptableActiveToolHandler.accepted()`.

Конструктор класса:

`__init__`(self[, parent])	Создает экземпляр класса.

Методы:

`disable`()	Отключает доступность блока с кнопкой Применить.
`try_enable`()	Включает доступность блока с кнопкой Применить если наблюдатель, связанный с панелью активного инструмента, подтверждает доступность.

Сигналы:

`accepted`	Отсылается после того как пользователь нажал кнопку "Применить" в панели активного инструмента.

__init__(self, parent: Union[PySide2.QtCore.QObject, NoneType] = None)

Создает экземпляр класса.

Initialize self. See help(type(self)) for accurate signature.

property accepted: Signal

Отсылается после того как пользователь нажал кнопку «Применить» в панели активного инструмента.

Тип результата
Signal[]

disable()

Отключает доступность блока с кнопкой Применить. Если инструмент запускает фоновые задачи с использованием `TaskManager`, то следует вызвать эту функцию перед началом выполнения задачи. Иначе у пользователя может быть возможность добавить множество одинаковых задач, несколько раз нажав на кнопку.

try_enable()

Включает доступность блока с кнопкой Применить если наблюдатель, связанный с панелью активного инструмента, подтверждает доступность.

24.5 AxipyCustomActiveToolPanelHandler - Управление панелью активного инструмента без предустановленных кнопок управления

class axipy.**AxipyCustomActiveToolPanelHandler**

Базовые классы: AxipyActiveToolPanelHandlerBase

Обработчик панели активного инструмента который не предоставляет никаких встроенных по умолчанию элементов управления.

Конструктор класса:

__init__(self[, parent])	Создает экземпляр класса.

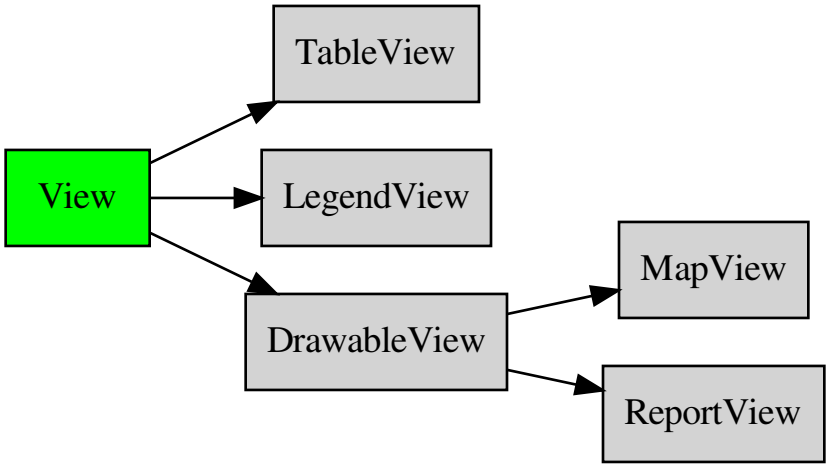
__init__(self, parent: Union[PySide2.QtCore.QObject, NoneType] = None)

Создает экземпляр класса.

Initialize self. See help(type(self)) for accurate signature.

24.6 View - Базовый класс для отображения данных в окне

Иерархия классов:



class axipy.**View**

Базовый класс для отображения данных в окне.

Свойства:

position	Размер и положение окна.
rect	Размер и положение окна.
show_type	Возвращает тип состояния окна.
title	Заголовок окна просмотра.
widget	Виджет, соответствующий содержимому окна.

Методы:

close()	Закрывает окно.
show([type])	Показывает окно в соответствие с приведенным типом.

close()

Закрывает окно.

property position: QRect

Размер и положение окна.

property rect: QRect

Размер и положение окна.

> **Предупреждение:** Не рекомендуется, начиная с версии 4.0: Используйте position.

show(type: int = SHOW_NORMAL)

Показывает окно в соответствие с приведенным типом.

Таблица 1: Допустимые значения:

Константа	Значение	Описание
SHOW_NORMAL	1	Обычный показ окна (по умолчанию).
SHOW_MINIMIZED	2	Показ окна в режиме минимизации.
SHOW_MAXIMIZED	3	Показ окна в режиме распахивания.

property show_type: int

Возвращает тип состояния окна. Подробнее см. show()

property title: str

Заголовок окна просмотра.

property widget: QWidget

Виджет, соответствующий содержимому окна.

Результат

Qt5 виджет содержимого.

24.7 TableView - Таблица просмотра атрибутивной информации

class axipy.**TableView**

Базовые классы: View

Таблица просмотра атрибутивной информации. Для создания экземпляра необходимо использовать axipy.ViewManager.create_tableview() через экземпляр view_manager

Свойства:

data_object	Таблица, на основании которой создается данное окно просмотра.
position	Размер и положение окна.
rect	Размер и положение окна.
show_type	Возвращает тип состояния окна.
table_view	Ссылка на объект таблицы просмотра.
title	Заголовок окна просмотра.
widget	Виджет, соответствующий содержимому окна.

Методы:

close()	Закрывает окно.
show([type])	Показывает окно в соответствие с приведенным типом.

close()

Закрывает окно.

property data_object: DataObject

Таблица, на основании которой создается данное окно просмотра.

Результат
Таблица.

property position: QRect

Размер и положение окна.

property rect: QRect

Размер и положение окна.

> **Предупреждение:** Не рекомендуется, начиная с версии 4.0: Используйте position.

show(type: int = SHOW_NORMAL)

Показывает окно в соответствие с приведенным типом.

Таблица 2: Допустимые значения:

Константа	Значение	Описание
SHOW_NORMAL	1	Обычный показ окна (по умолчанию).
SHOW_MINIMIZED	2	Показ окна в режиме минимизации.
SHOW_MAXIMIZED	3	Показ окна в режиме распахивания.

property show_type: int

Возвращает тип состояния окна. Подробнее см. show()

property table_view: QTableView

Ссылка на объект таблицы просмотра.

Пример установки сортировки для таблицы в текущем окне по второй колонке по возрастанию:

```
if isinstance(view_manager.active, TableView):
    view_manager.active.table_view.sortByColumn(2, Qt.AscendingOrder)
```

property title: str

Заголовок окна просмотра.

property widget: QWidget

Виджет, соответствующий содержимому окна.

Результат
Qt5 виджет содержимого.

24.8 DrawableView - Базовый класс с поддержкой визуального редактирования геометрий

class axipy.**DrawableView**

Базовые классы: View

Базовый класс для визуализации геометрических данных.

Свойства:

can_redo	Возможен ли откат на один шаг вперед.
can_undo	Возможен ли откат на один шаг назад.
is_modified	Есть ли изменения в окне.
position	Размер и положение окна.
rect	Размер и положение окна.
show_type	Возвращает тип состояния окна.
snap_mode	Включает режим привязки координат при редактировании геометрии в окне карты или отчета.
title	Заголовок окна просмотра.
widget	Виджет, соответствующий содержимому окна.

Методы:

close()	Закрывает окно.
offset(dx, dy)	Производит сдвиг окна карты или отчета.
redo()	Производит откат на один шаг вперед.
scale_with_center(scale, center)	Установка нового центра с заданным масштабированием.
show([type])	Показывает окно в соответствие с приведенным типом.
undo()	Производит откат на один шаг назад.

Сигналы:

scene_changed	Сигнал об изменении контента окна.

property can_redo: bool

Возможен ли откат на один шаг вперед.

property can_undo: bool

Возможен ли откат на один шаг назад.

close()

Закрывает окно.

property is_modified: bool

Есть ли изменения в окне.

offset(dx: float, dy: float)

Производит сдвиг окна карты или отчета. Особенностью является то, что при этом сохраняется прежний центр (актуально для карты).

Параметры

- **dx** – Смещение по горизонтали в координатах экрана (пикселях)
- **dy** – Смещение по вертикали в координатах экрана (пикселях)

property position: QRect

Размер и положение окна.

property rect: QRect

Размер и положение окна.

> **Предупреждение:** Не рекомендуется, начиная с версии 4.0: Используйте position.

redo()

Производит откат на один шаг вперед. При этом возвращается состояние до последней отмены.

scale_with_center(scale: float, center: Pnt)

Установка нового центра с заданным масштабированием.

Параметры

- **scale** – Коэффициент масштабирования по отношению к текущему.
- **center** – Устанавливаемый центр.

property scene_changed: Signal

Сигнал об изменении контента окна.

Тип результата
Signal[]

show(type: int = SHOW_NORMAL)

Показывает окно в соответствие с приведенным типом.

Таблица 3: Допустимые значения:

Константа	Значение	Описание
SHOW_NORMAL	1	Обычный показ окна (по умолчанию).
SHOW_MINIMIZED	2	Показ окна в режиме минимизации.
SHOW_MAXIMIZED	3	Показ окна в режиме распахивания.

property show_type: int

Возвращает тип состояния окна. Подробнее см. show()

property snap_mode: bool

Включает режим привязки координат при редактировании геометрии в окне карты или отчета.

property title: str

Заголовок окна просмотра.

undo()

Производит откат на один шаг назад.

property widget: QWidget

Виджет, соответствующий содержимому окна.

Результат

Qt5 виджет содержимого.

24.9 MapView - Окно просмотра карты

class axipy.**MapView**

Базовые классы: DrawableView

Окно просмотра карты. Используется для проведения различных манипуляций с картой. Для создания экземпляра необходимо использовать axipy.ViewManager.create_mapview() через экземпляр view_manager (пример см. ниже).

Примечание: При создании „MapView“ посредством axipy.ViewManager.create_mapview() производится клонирование экземпляра карты axipy.render.Map и для последующей работы при доступе к данному объекту необходимо использовать свойство map.

Свойство device_rect определяет размер самого окна карты, а свойство scene_rect - прямоугольную область, которая умещается в этом окне в СК карты.

Преобразование между этими двумя прямоугольниками производится с помощью матриц трансформации scene_to_device_transform и device_to_scene_transform.

К параметрам самой карты можно получить доступ через свойство map. Единицы измерения координат (unit) также берутся из наиболее подходящей СК, но при желании они могут быть изменены. К примеру, вместо метров могут быть установлены километры.

Рассмотрим пример создания карты с последующим помещением ее в окно ее просмотра. Далее, попробуем преобразовать объект типа полигон из координат окна экрана в координаты СК слоя посредством axipy.Geometry.affine_transform().

```
# Откроем таблицу, создадим на ее базе слой и добавим в карту
table_world = provider_manager.openfile('world.tab')
world = Layer.create(table_world)
map = Map([ world ])
# Для полученной карты создадим окно просмотра
mapview = view_manager.create_mapview(map)
# Выведем полученные параметры отображения
print('Прямоугольник экрана:', mapview.device_rect)
print('Прямоугольник карты:', mapview.scene_rect)
# Установим ширину карты и ее центр
mapview.center = (1000000, 1000000)
mapview.set_zoom(10e6 )

>>> Прямоугольник экрана: (0.0 0.0) (300.0 200.0)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
>>> Прямоугольник карты: (-16194966.287183324 -8621185.324024437) (16789976.
↪633236416 8326222.646170927)
```

```
#Создадим геометрический объект полигон в координатах экрана и преобразуем его в␣
↪СК карты
poly_device = Polygon([(100,100), (100,150), (150, 150), (150,100)])
# Используя матрицу трансформации, преобразуем его в координаты карты.
poly_scene = poly_device.affine_transform(mapview.device_to_scene_transform)
# Для контроля выведем полученный полигон в виде WKT
print('WKT:', poly_scene.to_wkt())

>>> WKT: POLYGON ((-5199985.31371008 -147481.338926755, -5199985.31371008 -
↪4384333.3314756, 297505.173026545 -4384333.3314756, 297505.173026545 -147481.
↪338926755, -5199985.31371008 -147481.338926755))
```

Свойства:

can_redo	Возможен ли откат на один шаг вперед.
can_undo	Возможен ли откат на один шаг назад.
center	Центр окна карты.
coordsystem	Система координат карты.
coordsystem_visual	Система координат карты с учетом поправки цены градуса по широте.
device_rect	Видимая область в координатах окна (пиксели).
device_to_scene_transform	Объект трансформации из координат окна в координаты карты.
editable_layer	Редактируемый слой на карте.
is_modified	Есть ли изменения в окне.
map	Объект карты.
position	Размер и положение окна.
rect	Размер и положение окна.
scale	Масштаб карты.
scene_rect	Видимая область в координатах карты (в единицах измерения СК).
scene_to_device_transform	Объект трансформации из координат карты в координаты окна.
selected_layer	Выделенный слой на карте.
show_type	Возвращает тип состояния окна.
snap_mode	Включает режим привязки координат при редактировании геометрии в окне карты или отчета.
title	Заголовок окна просмотра.
unit	Единицы измерения координат карты.
widget	Виджет, соответствующий содержимому окна.

Методы:

close()	Закрывает окно.
mouse_moved(x, y)	Сигнал при смещении курсора мыши.
offset(dx, dy)	Производит сдвиг окна карты или отчета.
redo()	Производит откат на один шаг вперед.
scale_with_center(scale, center)	Установка нового центра с заданным масштабированием.
set_zoom(zoom[, unit])	"Задание ширины окна карты.
set_zoom_and_center(zoom, center[, unit])	"Задает новый центр и ширину окна карты.
show([type])	Показывает окно в соответствие с приведенным типом.
show_all()	Полностью показывает все слои карты.
show_selection()	Перемещает карту к группе выделенных объектов, максимально увеличивая масштаб, но так, чтобы все объекты попадали.
undo()	Производит откат на один шаг назад.
zoom([unit])	Ширина окна карты.

Сигналы:

coordsystem_changed	Сигнал о том, что система координат изменилась.
editable_layer_changed	Сигнал о том, что редактируемый слой сменился.
scene_changed	Сигнал об изменении контента окна.

property can_redo: bool

Возможен ли откат на один шаг вперед.

property can_undo: bool

Возможен ли откат на один шаг назад.

property center: Pnt

Центр окна карты.

close()

Закрывает окно.

property coordsystem: CoordSystem

Система координат карты.

property coordsystem_changed: Signal

Сигнал о том, что система координат изменилась.

Пример:

```
layer_world = Layer.create(table_world)
map = Map([ layer_world ])
mapview = view_manager.create_mapview(map)
mapview.coordsystem_changed.connect(lambda: print('СК была изменена'))
csLL = CoordSystem.from_prj("1, 104")
```

(continues on next page)

(продолжение с предыдущей страницы)

```
mapview.coordsystem = csLL

>>> СК была изменена
```

property coordsystem_visual: CoordSystem

Система координат карты с учетом поправки цены градуса по широте. Отличается от coordsystem только лишь в случае, когда основная система координат - это широта/долгота и широта имеет ненулевое значение. При этом в диапазоне широты (-70...70) градусов вводится поправочный коэффициент, растягивающий изображение по широте и равный 1/cos(y).

property device_rect: Rect

Видимая область в координатах окна (пиксели).

Результат
Прямоугольник в координатах окна.

property device_to_scene_transform: QTransform

Объект трансформации из координат окна в координаты карты.

Результат
Объект трансформации.

property editable_layer: Optional[VectorLayer]

Редактируемый слой на карте.

Результат
Редактируемый слой. Если не определен, возвращает None.

property editable_layer_changed: Signal

Сигнал о том, что редактируемый слой сменился.

property is_modified: bool

Есть ли изменения в окне.

property map: Map

Объект карты.

Результат
Карта.

mouse_moved(x: float, y: float) → Signal

Сигнал при смещении курсора мыши. Возвращает значения в СК карты.

Параметры

- **x** – X координата
- **y** – Y координата

Пример:

```
mapview.mouse_moved.connect(lambda x,y: print('Coords: {} {}'.format(x, y)))

>> Coords: -5732500.0 958500.0
>> Coords: -5621900.0 847900.0
```

offset(dx: float, dy: float)

Производит сдвиг окна карты или отчета. Особенностью является то, что при этом сохраняется прежний центр (актуально для карты).

Параметры

- **dx** – Смещение по горизонтали в координатах экрана (пикселях)
- **dy** – Смещение по вертикали в координатах экрана (пикселях)

property position: QRect

Размер и положение окна.

property rect: QRect

Размер и положение окна.

> **Предупреждение:** Не рекомендуется, начиная с версии 4.0: Используйте position.

redo()

Производит откат на один шаг вперед. При этом возвращается состояние до последней отмены.

property scale: float

Масштаб карты.

scale_with_center(scale: float, center: Pnt)

Установка нового центра с заданным масштабированием.

Параметры

- **scale** – Коэффициент масштабирования по отношению к текущему.
- **center** – Устанавливаемый центр.

property scene_changed: Signal

Сигнал об изменении контента окна.

Тип результата
Signal[]

property scene_rect: Rect

Видимая область в координатах карты (в единицах измерения СК).

Результат
Прямоугольник в координатах карты.

property scene_to_device_transform: QTransform

Объект трансформации из координат карты в координаты окна.

Результат
Объект трансформации.

property selected_layer: Optional[VectorLayer]

Выделенный слой на карте.

Результат
Выделенный слой. Если выделение отсутствует, возвращает None.

set_zoom(zoom: float, unit: Optional[LinearUnit] = None)

«Задание ширины окна карты.

Параметры

- **zoom** – Значение ширины карты.
- **unit** – Единицы измерения. Если не заданы, берутся текущие для карты.

set_zoom_and_center(zoom: float, center: Pnt, unit: Optional[LinearUnit] = None)

«Задает новый центр и ширину окна карты.

Параметры

- **zoom** – Значение ширины карты.
- **center** – Центр карты.
- **unit** – Единицы измерения. Если не заданы, берутся текущие для карты.

show(type: int = SHOW_NORMAL)

Показывает окно в соответствие с приведенным типом.

Таблица 4: Допустимые значения:

Константа	Значение	Описание
SHOW_NORMAL	1	Обычный показ окна (по умолчанию).
SHOW_MINIMIZED	2	Показ окна в режиме минимизации.
SHOW_MAXIMIZED	3	Показ окна в режиме распахивания.

show_all()

Полностью показывает все слои карты.

show_selection()

Перемещает карту к группе выделенных объектов, максимально увеличивая масштаб, но так, чтобы все объекты попадали.

property show_type: int

Возвращает тип состояния окна. Подробнее см. `show()`

property snap_mode: bool

Включает режим привязки координат при редактировании геометрии в окне карты или отчета.

property title: str

Заголовок окна просмотра.

undo()

Производит откат на один шаг назад.

property unit: LinearUnit

Единицы измерения координат карты.

property widget: QWidget

Виджет, соответствующий содержимому окна.

Результат

Qt5 виджет содержимого.

zoom(unit: Optional[LinearUnit] = None) → float

Ширина окна карты.

Параметры

unit – Единицы измерения. Если не заданы, берутся текущие для карты.

24.10 ReportView - Окно просмотра отчета

class axipy.**ReportView**

Базовые классы: DrawableView

Окно с планом отчета. Для создания экземпляра необходимо использовать axipy.ViewManager.create_reportview() через экземпляр view_manager. До параметров самого отчета `axipy.render.Report` можно доступиться через свойство ReportView.report()

Пример создания отчета:

```
reportview = view_manager.create_reportview()

# Добавим полигон
geomItem = GeometryReportItem()
geomItem.geometry = Polygon((10,10), (10, 100), (100, 100), (10, 10))
geomItem.style = PolygonStyle(45, Qt.red)
reportview.report.items.add(geomItem)

# Установим текущий масштаб
reportview.view_scale = 33
```

Свойства:

can_redo	Возможен ли откат на один шаг вперед.
can_undo	Возможен ли откат на один шаг назад.
is_modified	Есть ли изменения в окне.
mesh_size	Размер ячейки сетки.
position	Размер и положение окна.
rect	Размер и положение окна.
report	Объект отчета.
show_borders	Показывать границы страниц.
show_elements_size	Показывать размер элементов.
show_mesh	Показывать сетку привязки.
show_ruler	Показывать линейку по краям.
show_type	Возвращает тип состояния окна.
snap_mode	Включает режим привязки координат при редактировании геометрии в окне карты или отчета.
snap_to_guidelines	Включение режима притяжения элементов отчета к направляющим.
snap_to_mesh	Включение режима притяжения элементов отчета к узлам сетки.
title	Заголовок окна просмотра.
view_scale	Текущий масштаб.
widget	Виджет, соответствующий содержимому окна.
x_guidelines	Вертикальные направляющие.
y_guidelines	Горизонтальные направляющие.

Методы:

clear_guidelines()	Удаляет все направляющие.
clear_selected_guidelines()	Очищает выбранные направляющие.
close()	Закрывает окно.
fill_on_pages()	Наиболее эффективно заполняет пространство отчета масштабированием его элементов.
get_printer()	Ссылка на используемый текущий принтер.
mouse_moved(x, y)	Сигнал при смещении курсора мыши.
offset(dx, dy)	Производит сдвиг окна карты или отчета.
redo()	Производит откат на один шаг вперед.
scale_with_center(scale, center)	Установка нового центра с заданным масштабированием.
set_printer(printer)	Устанавливает для отчета объект PySide2.QtPrintSupport.QPrinter
show([type])	Показывает окно в соответствие с приведенным типом.
undo()	Производит откат на один шаг назад.

Сигналы:

scene_changed	Сигнал об изменении контента окна.

property can_redo: bool

Возможен ли откат на один шаг вперед.

property can_undo: bool

Возможен ли откат на один шаг назад.

clear_guidelines()

Удаляет все направляющие.

clear_selected_guidelines()

Очищает выбранные направляющие.

close()

Закрывает окно.

fill_on_pages()

Наиболее эффективно заполняет пространство отчета масштабированием его элементов.

get_printer() → QPrinter

Ссылка на используемый текущий принтер. Для того, чтобы изменить настройки, нужно запросить существующий объект, поменять необходимые значения и снова назначить посредством ReportView.set_printer(). Или же установить другой объект PySide2.QtPrintSupport.QPrinter.

Список 2: Пример смены свойств у текущего окна отчета

```
from PySide2.QtGui import QPageLayout, QPageSize
import axipy
# Получение текущего окна отчета
reportview = axipy.view_manager.active
if isinstance(reportview, axipy.ReportView):
    # Получение текущего принтера
    printer = reportview.get_printer()
    # Именение размера страницы
    printer.setPageSize(QPageSize(QPageSize.A3))
    # Изменение ориентации страницы
    printer.setPageOrientation(QPageLayout.Landscape)
    # Установление нового значения
    reportview.set_printer(printer)
```

property is_modified: bool

Есть ли изменения в окне.

property mesh_size: float

Размер ячейки сетки.

mouse_moved(x: float, y: float) → Signal

Сигнал при смещении курсора мыши. Возвращает значения в координатах отчета.

Параметры

- **x** – X координата
- **y** – Y координата

Пример:

```
reportview.mouse_moved.connect(lambda x,y: print('Coords: {} {}'.format(x,␣
↪y)))
```

offset(dx: float, dy: float)

Производит сдвиг окна карты или отчета. Особенностью является то, что при этом сохраняется прежний центр (актуально для карты).

Параметры

- **dx** – Смещение по горизонтали в координатах экрана (пикселях)
- **dy** – Смещение по вертикали в координатах экрана (пикселях)

property position: QRect

Размер и положение окна.

property rect: QRect

Размер и положение окна.

> **Предупреждение:** Не рекомендуется, начиная с версии 4.0: Используйте position.

redo()

Производит откат на один шаг вперед. При этом возвращается состояние до последней отмены.

property report: Report

Объект отчета.

Результат
Отчет.

scale_with_center(scale: float, center: Pnt)

Установка нового центра с заданным масштабированием.

Параметры

- **scale** – Коэффициент масштабирования по отношению к текущему.
- **center** – Устанавливаемый центр.

property scene_changed: Signal

Сигнал об изменении контента окна.

Тип результата
Signal[]

set_printer(printer: QPrinter)

Устанавливает для отчета объект PySide2.QtPrintSupport.QPrinter

Параметры
printer – Новое значение принтера или измененное запрошенное ранее через ReportView.get_printer()

show(type: int = SHOW_NORMAL)

Показывает окно в соответствие с приведенным типом.

Таблица 5: Допустимые значения:

Константа	Значение	Описание
SHOW_NORMAL	1	Обычный показ окна (по умолчанию).
SHOW_MINIMIZED	2	Показ окна в режиме минимизации.
SHOW_MAXIMIZED	3	Показ окна в режиме распахивания.

property show_borders: float

Показывать границы страниц.

property show_elements_size: float

Показывать размер элементов.

property show_mesh: bool

Показывать сетку привязки.

property show_ruler: float

Показывать линейку по краям.

property show_type: int

Возвращает тип состояния окна. Подробнее см. `show()`

property snap_mode: bool

Включает режим привязки координат при редактировании геометрии в окне карты или отчета.

property snap_to_guidelines: bool

Включение режима притяжения элементов отчета к направляющим.

property snap_to_mesh: bool

Включение режима притяжения элементов отчета к узлам сетки.

property title: str

Заголовок окна просмотра.

undo()

Производит откат на один шаг назад.

property view_scale: float

Текущий масштаб.

property widget: QWidget

Виджет, соответствующий содержимому окна.

Результат
Qt5 виджет содержимого.

property x_guidelines

Вертикальные направляющие. Значения содержатся в единицах измерения отчета.

Рассмотрим на примере:

```
# Добавление вертикальной направляющей
reportview.x_guidelines.append(20)
# Изменение значения направляющей по индексу
reportview.x_guidelines[0] = 80
# Удаление всех направляющих.
reportview.clear_guidelines()
```

property y_guidelines

Горизонтальные направляющие. Работа с ними производится по аналогии с вертикальными направляющими.

24.11 LegendView - Окно просмотра легенд карты

class axipy.**LegendView**

Базовые классы: View

Легенда для карты. Для создания экземпляра необходимо использовать axipy.ViewManager.create_legendview() через экземпляр view_manager. В качестве параметра передается открытое ранее окно с картой:

```
legendView = view_manager.create_legendview(map_view)
```

Список легенд доступен через свойство legends:

```
for legend in legendView.legends:
   print(legend.caption)
```

Состав может меняться посредством вызова соответствующих методов свойства legends.

Добавление легенды для слоя карты:

```
legend = Legend(map_view.map.layers[0])
legend.caption = 'Легенда слоя'
legendView.legends.append(legend)
legendView.arrange()
```

Доступ к элементу по индексу. Поменяем описание четвертого оформления у первой легенды axipy.render.Legend окна:

```
legend = legendView.legends[1]
item = legend.items[3]
item.title = 'Описание'
legend.items[3] = item
```

Удаление первой легенды из окна:

```
legendView.legends.remove(0)
```

Свойства:

legends	Перечень добавленных в окно легенд.
position	Размер и положение окна.
rect	Размер и положение окна.
show_type	Возвращает тип состояния окна.
title	Заголовок окна просмотра.
widget	Виджет, соответствующий содержимому окна.

Методы:

arrange()	Упорядочивает легенды с целью устранения наложений легенд друг на друга.
close()	Закрывает окно.
show([type])	Показывает окно в соответствие с приведенным типом.

arrange()

Упорядочивает легенды с целью устранения наложений легенд друг на друга.

close()

Закрывает окно.

property legends: ListLegend

Перечень добавленных в окно легенд.

property position: QRect

Размер и положение окна.

property rect: QRect

Размер и положение окна.

> **Предупреждение:** Не рекомендуется, начиная с версии 4.0: Используйте position.

show(type: int = SHOW_NORMAL)

Показывает окно в соответствие с приведенным типом.

Таблица 6: Допустимые значения:

Константа	Значение	Описание
SHOW_NORMAL	1	Обычный показ окна (по умолчанию).
SHOW_MINIMIZED	2	Показ окна в режиме минимизации.
SHOW_MAXIMIZED	3	Показ окна в режиме распахивания.

property show_type: int

Возвращает тип состояния окна. Подробнее см. show()

property title: str

Заголовок окна просмотра.

property widget: QWidget

Виджет, соответствующий содержимому окна.

Результат

Qt5 виджет содержимого.

24.12 ListLegend - Список легенд

class axipy.**ListLegend**

Базовые классы: object

Список добавленных в окно легенд.

24.13 SelectionManager - Доступ к выделенным объектам

class axipy.**SelectionManager**

Класс доступа к выделенным объектам.

Примечание: Получить экземпляр сервиса можно в атрибуте axipy.selection_manager.

Свойства:

count	Размер выделения, то есть количество выделенных записей (количество элементов в списке идентификаторов).
ids	Список идентификаторов выделенных записей.
table	Таблица, являющаяся источником текущего выделения.

Методы:

add(table, ids)	Добавляет выборку записи таблицы по их идентификаторам.
clear()	Очищает выборку.
get_as_cursor()	Возвращает выборку в виде итератора по записям.
get_as_table()	Возвращает выборку в виде таблицы.
remove(table, ids)	Удаляет из выборки записи таблицы по их идентификаторам.
set(table, ids)	Создает выборку из записей таблицы по их идентификаторам.

Сигналы:

changed	Выделение было изменено.

add(table: Table, ids: Union[List[int], int])

Добавляет выборку записи таблицы по их идентификаторам.

Параметры

- **table** – Таблица.
- **ids** – Идентификаторы записей.

property changed: Signal

Выделение было изменено.

Тип результата
Signal[]

clear()

Очищает выборку.

property count: int

Размер выделения, то есть количество выделенных записей (количество элементов в списке идентификаторов).

get_as_cursor() → Iterator[Feature]

Возвращает выборку в виде итератора по записям.

Пример:

```
for f in selection_manager.get_as_cursor():
    print('Feature id={}. Страна={}'.format(f.id, f['Страна']))
```

Предупреждение: Не рекомендуется, начиная с версии 3.5: Используйте axipy.DataManager.selection.

get_as_table() → Optional[Table]

Возвращает выборку в виде таблицы.

Результат
Таблица или None, если выборки нет.

Содержимое таблицы основывается на текущей выборке на момент вызова данного метода. При последующем изменении или сбросе выборки контент данной таблицы не меняется. Результирующей таблице присваивается наименование в формате data*, которое в последствии можно изменить. При закрытии базовой таблицы данная таблицы так-же закрывается.

Пример:

```
# Получаем таблицу из выборки.
tbl = selection_manager.get_as_table()
# Задаем желаемое имя таблицы (необязательно)
tbl.name = 'my_table'
# Регистрация в каталоге (необязательно)
app.mainwindow.catalog().add(tbl)
for f in tbl.items():
    print('Feature id={}. Страна={}'.format(f.id, f['Страна']))
```

Предупреждение: Не рекомендуется, начиная с версии 3.5: Используйте axipy.DataManager.selection.

property ids: List[int]

Список идентификаторов выделенных записей.

remove(table: Table, ids: Union[List[int], int])

Удаляет из выборки записи таблицы по их идентификаторам.

Параметры

- **table** – Таблица.
- **ids** – Идентификаторы записей.

set(table: Table, ids: Union[List[int], int])

Создает выборку из записей таблицы по их идентификаторам.

Параметры

- **table** – Таблица.
- **ids** – Идентификаторы записей.

property table: Optional[Table]

Таблица, являющаяся источником текущего выделения. Если ничего не выделено, то None

24.14 ViewManager - Менеджер содержимого окон

class axipy.**ViewManager**

Менеджер содержимого окон.

Примечание: Используйте готовый экземпляр этого класса `view_manager`.

Список 3: Пример использования

```
table = provider_manager.openfile(filepath)
m = Map([table])
view_manager.create_mapview(m)
```

Свойства:

active	Текущее активное окно.
count	Количество окон.
global_parent	Может использоваться как 'parent' при использовании стандартных диалогов Qt.
legendviews	Список всех окон с легендами.
mapviews	Список всех окон с картами.
reportviews	Список всех окон с отчетами.
tableviews	Список всех окон с таблицами просмотра.
views	Список всех известных окон.

Методы:

activate(view)	Делает заданное окно активным.
close(view)	Закрывает окно.
close_all()	Закрывает все окна.
create_legendview(mapview)	Создает окно легенды для карты.
create_mapview(map)	Создает окно из для переданного объекта карты.
create_reportview([report])	Создает окно с планом отчета.
create_tableview(table)	Создает окно в виде табличного представления из объекта данных.

Сигналы:

active_changed	Активное окно изменилось.
count_changed	Количество окон изменилось.
mainwindow_activated	Возникает когда главное окно приложения инициализировано и активировано.

activate(view: View)

Делает заданное окно активным.

Параметры
view – Содержимое окна.

property active: Optional[View]

Текущее активное окно.

Результат
None, если нет активных окон.

property active_changed: Signal

Активное окно изменилось.

Тип результата
Signal[]

close(view: View)

Закрывает окно.

Параметры
view – Содержимое окна.

close_all()

Закрывает все окна.

property count: int

Количество окон.

property count_changed: Signal

Количество окон изменилось.

Тип результата
Signal[]

create_legendview(mapview: MapView) → LegendView

Создает окно легенды для карты.

Параметры
mapview – Окно с картой.

Результат
Окно с легендой.

create_mapview(map: Map) → MapView

Создает окно из для переданного объекта карты.

Параметры
map – Карта.

Примечание: Переданная карта копируется.

Результат
Окно карты.

create_reportview(report: Report = None) → ReportView

Создает окно с планом отчета.

Параметры
report – План отчета. Если не передан, то создается по умолчанию.

Результат
Окно отчета.

create_tableview(table: Table) → TableView

Создает окно в виде табличного представления из объекта данных.

Параметры
table – Таблица.

Результат
Окно таблицы.

property global_parent: QWidget

Может использоваться как „parent“ при использовании стандартных диалогов Qt. Использование данного свойства решает проблему, когда окно показывается в панели задач как отдельное приложение.

Пример:

```
if QMessageBox.question(view_manager.global_parent, 'Вопрос', 'Отменить␣
↪действие?') == QMessageBox.Yes:
    pass
```

property legendviews: List[LegendView]

Список всех окон с легендами.

property mainwindow_activated: Signal

Возникает когда главное окно приложения инициализировано и активировано. Данное событие может использоваться в плагинах когда есть необходимость обратиться к главному окну приложения. Но ввиду того, что сами плагины загружаются до инициализации главного окна, данную процедуру можно выполнить используя данное событие (`axipy.AxiomaInterface.window`).

Тип результата
Signal[]

property mapviews: List[MapView]

Список всех окон с картами.

Пример:

```
for v in view_manager.mapviews:
    print('Widget:', v.title)

>>> Widget: Карта: world
>>> Widget: Карта: rus_obl
```

property reportviews: List[ReportView]

Список всех окон с отчетами.

property tableviews: List[TableView]

Список всех окон с таблицами просмотра.

property views: List[View]

Список всех известных окон.

24.15 ActionManager - Менеджер системных действий и инструментов

class axipy.**ActionManager**

Менеджер системных действий и инструментов. Класс является статическим словарем, доступным только для чтения (collections.abc.Mapping). Поддерживает обращение по индексу.

Классовые методы:

activate(name)	Делает активным инструмент по его идентификатору если это возможно.
get(key[, default_value])	Возвращает значение по ключу.
icon_by_name(name)	Создание иконки по ее внутреннему идентификатору.
items()	Возвращает список кортежей ключ-значение, где ключи это идентификаторы действий, а значения это объекты класса PySide2.QtWidgets.QAction.
keys()	Возвращает список ключей, где ключи это идентификаторы действий.
values()	Возвращает список значений, где значения это это объекты класса PySide2.QtWidgets.QAction.

Атрибуты:

all_icon_names	Перечень доступных идентификаторов иконок, присутствующих в ресурсах Аксиомы.

static activate(name: str)

Делает активным инструмент по его идентификатору если это возможно. Если действие не найдено, генерируется исключение. Если действие недоступно в настоящий момент (неактивно), установка игнорируется.

Параметры
name – Идентификатор действия

Активация инструмента „Сдвиг“:

```
ActionManager.activate('Pan')
```

Вызов диалога „Стиль символа“:

```
ActionManager.activate('SymbolStyle')
```

all_icon_names: List[str]

Перечень доступных идентификаторов иконок, присутствующих в ресурсах Аксиомы.

На основании данного идентификатора можно создать иконку для действия PySide2.QtWidgets.QAction. Использовать выбранный идентификатор предлагается с помощью функции icon_by_name().

Для создания иконки для PySide2.QtWidgets.QAction также доступен стандартный путь указания имени файла.

classmethod get(key: str, default_value: Optional[Any] = None) → Optional[QAction]

Возвращает значение по ключу.

static icon_by_name(name: str) → QIcon

Создание иконки по ее внутреннему идентификатору. Подробнее см. all_icon_names

Параметры
name – Идентификатор действия

classmethod items() → List[Tuple[str, QAction]]

Возвращает список кортежей ключ-значение, где ключи это идентификаторы действий, а значения это объекты класса PySide2.QtWidgets.QAction.

classmethod keys() → List[str]

Возвращает список ключей, где ключи это идентификаторы действий.

classmethod values() → List[QAction]

Возвращает список значений, где значения это это объекты класса PySide2.QtWidgets.QAction.

24.16 Workspace - Рабочее пространство

class axipy.**Workspace**

Сохранение рабочего пространства в файл рабочего набора *.mws.

Рабочее пространство можно как сохранять в файл так и читать из него. При чтении из файла рабочего пространства посредством метода load_file() все источники данных (таблицы) открываются и добавляются в каталог axipy.DataManager, доступный через переменную axipy.data_manager. А окна с наполнением добавляются в менеджер окон axipy.ViewManager, доступный через переменную view_manager.

В случае записи текущего состояния в файл рабочего пространства последовательность обратная рассмотренной. Состояние каталога и менеджера окон записывается в рабочее пространство посредством метода save_file().

Пример чтения данных в рамках отдельного приложения:

```
from axipy import init_axioma, data_manager, view_manager, Workspace, open_file_
↪dialog

# инициализация ядра
app = init_axioma()
print('Before: tables({}), views({})'.format(len(data_manager), len(view_
↪manager)))
# Чтение рабочего набора
file_name = open_file_dialog("Рабочий набор (*.mws)")
Workspace().load_file(file_name)
print('After: tables({}), views({})'.format(len(data_manager), len(view_manager)))
# Если есть хотя бы одно окно с картой, показываем его
if len(view_manager.mapviews):
    mapview = view_manager.mapviews[0]
    mapview.show()
app.exec_()

# Результат:
# Before: tables(0), views(0)
# After: tables(5), views(3)  # В зависимости от рабочего набора, числа могут␣
↪отличаться
```

Пример записи рабочего пространства:

```
from pathlib import Path

from PySide2.QtCore import QStandardPaths

from axipy import Workspace

path = Path(QStandardPaths.writableLocation(QStandardPaths.DocumentsLocation))
path = path / "ws_out.mws"
Workspace().save_file(path)
print("Файл рабочего набора сохранен.", path)

# Результат:
# Файл рабочего набора сохранен. C:\Users\User\Documents\ws_out.mws
```

Методы:

load_file(file_name)	Читает из файла рабочего пространства и заносит данные в текущее окружение.
load_string(workspace_str)	Читает из строки описание рабочего пространства и заносит данные в текущее окружение.
save_file(file_name[, overwrite])	Сохраняет текущее состояние в файл рабочего пространства.
save_string()	Сохраняет текущее состояние рабочего пространства в виде строки.

load_file(file_name: Union[str, Path])

Читает из файла рабочего пространства и заносит данные в текущее окружение.

Параметры
file_name – Имя входного файла.

load_string(workspace_str: str)

Читает из строки описание рабочего пространства и заносит данные в текущее окружение.

Параметры
workspace_str – Строка с контентом рабочего пространства.

save_file(file_name: Union[str, Path], overwrite: bool = False)

Сохраняет текущее состояние в файл рабочего пространства.

Параметры

- **file_name** – Имя выходного файла.
- **overwrite** – Перезаписывать существующий файл. По умолчанию False.

save_string() → str

Сохраняет текущее состояние рабочего пространства в виде строки.

Результат
Строка с контентом рабочего пространства.

24.17 ChooseCoordSystemDialog - Диалог выбора СК

class axipy.**ChooseCoordSystemDialog**

Диалог выбора координатной системы.

Args:

coordsystem: Система координат по умолчанию.
Если не указана, то задается как значение по умолчанию `axipy.cs.CoordSystem.current()`

Пример:

```
csMercator = CoordSystem.from_prj('10, 104, 7, 0')
dialog = ChooseCoordSystemDialog(csMercator)
if dialog.exec() == QDialog.Accepted:
    result_cs = dialog.chosenCoordSystem()
    print("Выбрано:", result_cs.title)
```

Методы:

chosenCoordSystem()	Возвращает выбранную в диалоге систему координат.

chosenCoordSystem() → CoordSystem

Возвращает выбранную в диалоге систему координат.

24.18 PasswordDialog - Диалог аутентификации пользователя.

class axipy.**PasswordDialog**

Диалог задания или корректировки данных аутентификации пользователя.

Пример:

```
dialog = PasswordDialog()
dialog.address = 'proxy.server'
dialog.user_name = 'user'
dialog.password = 'pass'
if dialog.exec() == QDialog.Accepted:
    print(dialog.user_name, dialog.password)
```

Свойства:

address	Описание ресурса.
password	Пароль.
user_name	Имя пользователя.

property address: str

Описание ресурса. Если значение указано, то в диалоговом окне оно будет присутствовать под именем „Адрес“

property password: str

Пароль.

property user_name: str

Имя пользователя.

24.19 BoundingRectDialog - Диалог задания параметров прямоугольника

class axipy.**BoundingRectDialog**

Диалог задания параметров прямоугольника. Например, охвата для таблицы или системы координат.

Пример:

```
dlg = BoundingRectDialog()
dlg.unit = 'км'
dlg.is_square = True
dlg.rect = Rect(-100, -120, 200, 200)
if dlg.exec() == QDialog.Accepted:
    print('>>>>', dlg.rect)
```

Свойства:

is_square	Контроль соблюдения равенства ширины и высоты.
rect	Параметры прямоугольника
unit	Наименование единиц измерения.

property is_square: bool

Контроль соблюдения равенства ширины и высоты. При этом после ввода какого-либо значения производится коррекция таким образом, чтобы в результате получился квадрат.

property rect: Rect

Параметры прямоугольника

property unit: str

Наименование единиц измерения.

24.20 StyleButton - Кнопка выбора стиля

class axipy.**StyleButton**

Кнопка, отображающая стиль и позволяющая его менять.

Пример добавления кнопки на диалог:

```
from PySide2.QtWidgets import QDialog
from axipy import Style, StyleButton

style = Style.from_mapinfo("Pen (1, 2, 8421504)  Brush (2, 255, 0)")


class Dialog(QDialog):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.pb = StyleButton(style, parent=self)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        self.pb.style_changed.connect(self.style_result)
        self.pb.setGeometry(100, 100, 100, 50)

    def style_result(self):
        print("Стиль изменен", self.pb.style)


dialog = Dialog()
dialog.open()
```

Конструктор класса:

`__init__`([style, parent])	Создает экземпляр класса.

Свойства:

`style`	Устанавливает или возвращает стиль кнопки.

Сигналы:

`style_changed`	Сигнал испускается при смене стиля.

__init__(style: Optional[Style] = None, parent: Optional[QWidget] = None)

Создает экземпляр класса.

Параметры

- **style** – Стиль по умолчанию.
- **parent** – Родительский виджет.

property style: Style

Устанавливает или возвращает стиль кнопки.

property style_changed: Signal

Сигнал испускается при смене стиля.

24.21 Виджеты Аксиомы

24.21.1 LayerControlWidget - Виджет управления слоями карты

class axipy.**LayerControlWidget**

Виджет управления слоями карты.

Свойства:

`widget`	Виджет, соответствующий содержимому окна.

Сигналы:

mapview_activated	Сигнал об изменении активной карты.

property mapview_activated: Signal

Сигнал об изменении активной карты.

Тип результата
Signal[MapView]

property widget: QWidget

Виджет, соответствующий содержимому окна.

Результат
Qt5 виджет содержимого.

24.21.2 DataManagerWidget - Виджет управления списком открытых данных

class axipy.**DataManagerWidget**

Виджет управления списком открытых данных.

Свойства:

list_widget	Виджет, соответствующий содержимому списка таблиц.
objects	Список выделенных объектов.
widget	Виджет, соответствующий содержимому окна.

Сигналы:

selection_changed	Выделение в списке было изменено.

property list_widget: QListView

Виджет, соответствующий содержимому списка таблиц.

Результат
Qt5 виджет списка.

property objects: List[DataObject]

Список выделенных объектов.

property selection_changed: Signal

Выделение в списке было изменено.

Тип результата
Signal[]

property widget: QWidget

Виджет, соответствующий содержимому окна.

Результат
Qt5 виджет содержимого.

24.21.3 ViewManagerWidget - Виджет списка окон

class axipy.**ViewManagerWidget**

Виджет списка окон.

Свойства:

widget	Виджет, соответствующий содержимому окна.

Сигналы:

view_changed	Сигнал об изменении выделения в списке.

property view_changed: Signal

Сигнал об изменении выделения в списке.

Тип результата
Signal[View]

property widget: QWidget

Виджет, соответствующий содержимому окна.

Результат
Qt5 виджет содержимого.

24.21.4 NotificationWidget - Виджет уведомлений

class axipy.**NotificationWidget**

Виджет уведомлений.

Свойства:

widget	Виджет, соответствующий содержимому окна.

property widget: QWidget

Виджет, соответствующий содержимому окна.

Результат
Qt5 виджет содержимого.

24.21.5 PythonConsoleWidget - Виджет ввода команд python

class axipy.**PythonConsoleWidget**

Виджет ввода команд python.

Свойства:

widget	Виджет, соответствующий содержимому окна.

property widget: QWidget

Виджет, соответствующий содержимому окна.

Результат

Qt5 виджет содержимого.

24.22 Notifications - Отправление уведомлений

class axipy.**Notifications**

Отправление уведомлений в виде всплывающего окна с его последующей регистрацией в окне уведомлений.

Классовые методы:

push(title, text[, type_message])	Отправляет уведомление.

static push(title: str, text: str, type_message: int = 0)

Отправляет уведомление.

Параметры

- **title** – Заголовок
- **text** – Текст сообщения.
- **type_message** – Тип сообщения. В зависимости от типа сообщения в окне уведомлений оно помечается соответствующим цветом.

Таблица 7: Допустимые значения для типа сообщения:

Атрибут	Наименование
Information	Информационное сообщение. Устанавливается по умолчанию
Warning	Предупреждение
Critical	Критическая ошибка
Success	Успешное выполнение процесса

Пример:

```
from axipy import Notifications
Notifications.push('Предупреждение', 'Сообщение', Notifications.Warning)
```

25

axipy.menubar - Модуль меню главного окна ГИС Аксиома.

Модуль меню главного окна ГИС Аксиома.

25.1 Button - Кнопка

class axipy.menubar.**Button**

Кнопка с инструментом для добавления в меню. Абстрактный класс.

Для создания объекта этого класса используйте ActionButton, ToolButton.

Свойства:

action	Ссылка на объект QAction.
observer_id	Идентификатор наблюдателя для определения доступности инструмента.

Методы:

remove()	Удаляет кнопку из меню.

property action: QAction

Ссылка на объект QAction. Через него можно производить дополнительные необходимые действия через объект Qt.

Пример задания всплывающей подсказки, используя метод класса QAction:

```
button.action.setToolTip("Всплывающая подсказка")
```

property observer_id: str

Идентификатор наблюдателя для определения доступности инструмента.

remove()

Удаляет кнопку из меню.

25.2 ActionButton - Кнопка с действием

class axipy.**ActionButton**

Базовые классы: Button

Кнопка с действием.

Параметры

- **title** – Текст.
- **on_click** – Действие на нажатие. Делегируется функция, которая будет вызвана при активации инструмента.
- **icon** – Иконка. Может быть путем к файлу или адресом ресурса.
- **enable_on** – Идентификатор наблюдателя для определения доступности кнопки. Если это пользовательский наблюдатель, то указывается его наименование при создании.
- **tooltip** – Строка с дополнительной короткой информацией по данному действию.

См.также:

axipy.ObserverManager.

Список 1: Пример со встроенным наблюдателем.

```
from axipy import menubar, ObserverManager

button = menubar.ActionButton(
    'Мое действие',
    on_click=lambda: print('clicked'),
    enable_on=ObserverManager.HasTables
)
```

Список 2: Пример со пользовательским наблюдателем.

```
from axipy import ObserverManager, menubar

my_observer = ObserverManager.create('MyStateManager', False)
button = menubar.ActionButton(
    'Мое действие',
    on_click=lambda: print('clicked'),
    enable_on='MyStateManager'
)
```

Конструктор класса:

__init__(title, on_click[, icon, enable_on, ...])	Создает экземпляр класса.

Свойства:

action	Ссылка на объект QAction.
observer_id	Идентификатор наблюдателя для определения доступности инструмента.

Методы:

remove()	Удаляет кнопку из меню.

__init__(title: str, on_click: Callable[[], Any], icon: Union[str, QIcon] = '', enable_on: Optional[Union[str, Observer]] = None, tooltip: Optional[str] = None)

Создает экземпляр класса.

property action: QAction

Ссылка на объект QAction. Через него можно производить дополнительные необходимые действия через объект Qt.

Пример задания всплывающей подсказки, используя метод класса QAction:

```
button.action.setToolTip("Всплывающая подсказка")
```

property observer_id: str

Идентификатор наблюдателя для определения доступности инструмента.

remove()

Удаляет кнопку из меню.

25.3 SystemActionButton - Действие, присутствующее в системе

class axipy.**SystemActionButton**

Действие, присутствующее в системе.

Конструктор класса:

__init__(name)	Создает экземпляр класса.

Методы:

remove()	Удаляет кнопку из меню.

__init__(name: str)

Создает экземпляр класса.

Параметры

name – идентификатор действия Полный список доступных идентификаторов можно получить посредством axipy.ActionManager.keys().

remove()

Удаляет кнопку из меню.

25.4 ToolButton - Кнопка с инструментом

class axipy.**ToolButton**

Базовые классы: Button

Кнопка с инструментом.

Параметры

- **title** – Текст.
- **on_click** – Класс инструмента, наследник от axipy.MapTool.
- **icon** – Иконка. Может быть путем к файлу или адресом ресурса.
- **enable_on** – Идентификатор наблюдателя для определения доступности кнопки.

См.также:

axipy.ObserverManager.

Список 3: Пример

```
from axipy import MapTool, ToolButton

# Класс инструмента
class MyTool(MapTool):
    pass
param = 'Передаваемый параметр'
# Передача имени класса MapTool как параметр
button = ToolButton('Мой инструмент', MyTool)
# Если необходимо передавать параметры в конструктор, то можно передать как␣
↪конструктор
# внутри lambda функции
button = ToolButton('Мой инструмент', lambda: MyTool(param))
```

Конструктор класса:

__init__(title, on_click[, icon, enable_on, ...])	Создает экземпляр класса.

Свойства:

action	Ссылка на объект QAction.
observer_id	Идентификатор наблюдателя для определения доступности инструмента.

Методы:

remove()	Удаляет кнопку из меню.

__init__(title: str, on_click: Union[Callable[[], MapTool], MapTool], icon: Union[str, QIcon] = '', enable_on: Optional[Union[str, Observer]] = None, tooltip: Optional[str] = None)

Создает экземпляр класса.

property action: QAction

Ссылка на объект QAction. Через него можно производить дополнительные необходимые действия через объект Qt.

Пример задания всплывающей подсказки, используя метод класса QAction:

```
button.action.setToolTip("Всплывающая подсказка")
```

property observer_id: str

Идентификатор наблюдателя для определения доступности инструмента.

remove()

Удаляет кнопку из меню.

25.5 Separator - Разделитель

class axipy.menubar.**Separator**

Базовые классы: Button

Разделитель.

Свойства:

action	Ссылка на объект QAction.
observer_id	Идентификатор наблюдателя для определения доступности инструмента.

Методы:

remove()	Удаляет кнопку из меню.

property action: QAction

Ссылка на объект QAction. Через него можно производить дополнительные необходимые действия через объект Qt.

Пример задания всплывающей подсказки, используя метод класса QAction:

```
button.action.setToolTip("Всплывающая подсказка")
```

property observer_id: str

Идентификатор наблюдателя для определения доступности инструмента.

remove()

Удаляет кнопку из меню.

25.6 Position - Положение кнопки

class axipy.menubar.**Position**

Положение кнопки в меню.

Параметры

- **tab** – Название вкладки.
- **group** – Название группы.

Пример:

```
pos = Position("Основные", "Команды")
```

Конструктор класса:

__init__(tab, group)	Создает экземпляр класса.

Методы:

add(button[, size])	Добавляет кнопку в текущее положение.

__init__(tab: str, group: str)

Создает экземпляр класса.

add(button: Button, size: int = 2)

Добавляет кнопку в текущее положение.

Параметры

- **button** – Кнопка.
- **size** – Размер кнопки. Маленькая кнопка - 1. Большая кнопка - 2.

26

Глоссарий

Метод экземпляра
: Функция, определенная в классе. Вызывается из экземпляра класса (объекта).

  При определении функции, первый аргумент функции содержит ссылку на экземпляр класса и обычно называется self. При вызове функции, ссылка на экземпляр класса передается автоматически (неявно).

Классовый метод
: Функция, определенная в классе и обернутая декоратором `classmethod()`. Вызывается из класса (типа).

  При определении функции, первый аргумент функции содержит ссылку на класс (тип) и обычно называется cls. При вызове функции, ссылка на класс (тип) передается автоматически (неявно).

27

История изменений

27.1 5.2 Изменения

27.1.1 Исправления

- Функция callback для методов `axipy.Destination.export()` и `axipy.Destination.export_from_table()` теперь игнорирует все возвращаемые значения, отличные от `False`.

27.2 5.1 Изменения

Июль 2023

27.2.1 Исправления

- Методы `axipy.Workspace.load_file()` и `axipy.Workspace.save_file()` теперь работают независимо от наличия главного окна Аксиомы. Методы `axipy.MainWindow.load_workspace()` и `axipy.MainWindow.save_workspace()` отмечены как устаревшие.

27.3 5.0.1 Изменения

Июль 2023

27.3.1 Исправления

- Для классов (словарей `dict`):
  - `axipy.ObserverManager`
  - `axipy.CurrentSettings`
  - `axipy.DefaultSettings`
  - `axipy.LinearUnits`
  - `axipy.AreaUnits`
  - Исправлена ошибка с неправильной работой оператора принадлежности `in`;
  - Исправлена некорректная работа метода `get()`, где не учитывалось значение по умолчанию;
  - Исправлена ошибка, где при обращение по индексу `[]` к несуществующему элементу, не генерировалось исключение `KeyError`.

27.4 5.0 Изменения

Июнь 2023

27.4.1 Новое

- Раздел Формирование пользовательского приложения.
- Классы виджетов Аксиомы:
  - `axipy.DataManagerWidget`
  - `axipy.ViewManagerWidget`
  - `axipy.LayerControlWidget`
  - `axipy.NotificationWidget`
  - `axipy.PythonConsoleWidget`
- Класс `axipy.ActionManager` - Менеджер системных действий и инструментов.
- Классы `axipy.LinearUnits`, `axipy.AreaUnits`.
- Классы `axipy.ObserverManager`, `axipy.Observer` вместо устаревших классов `axipy.StateManager` и `axipy.ValueObserver`.
- Классы `axipy.FloatCoord`, `axipy.AngleCoord` вместо устаревших классов `axipy.FloatFormatter` и `axipy.CoordFormatter`.
- Классы `axipy.CurrentSettings`, `axipy.DefaultSettings` вместо устаревшего класса `axipy.Settings`.
- Класс `axipy.StyleButton` вместо устаревшего класса `axipy.StyledButton`.
- Класс `axipy.Plugin` вместо устаревших классов `axipy.AxiomaPlugin` и `axipy.AxiomaInterface`.
- Метод `axipy.execfile()`.

- Метод axipy.open_file_dialog().

27.4.2 Исправления

- Класс axipy.app.Notifications перенесен в axipy.gui как axipy.gui.Notifications.
- В соответствии с **PEP 8#package-and-module-names** переименованы модули:
  - Модуль axipy.concurrent.AxipyProgressHandler переименован в axipy.concurrent.axipy_progress_handler.
  - Модуль axipy.concurrent.Task переименован в axipy.concurrent.task.
  - Модуль axipy.concurrent.TaskManager переименован в axipy.concurrent.task_manager_.
  - Модуль axipy.concurrent.TaskUtils переименован в axipy.concurrent.task_utils.
  - Модуль axipy.cs.CoordSystem переименован в axipy.cs.coord_system.
  - Модуль axipy.cs.CoordTransformer переименован в axipy.cs.coord_transformer.
  - Модуль axipy.da.attribute_schema переименован в axipy.da.schema.
  - Модуль axipy.da.DataManagerWrapper переименован в axipy.da.data_manager_.
  - Модуль axipy.da.DataObjectWrapper переименован в axipy.da.data_object.
  - Модуль axipy.da.FeatureWrapper переименован в axipy.da.feature.
  - Модуль axipy.da.Geometry переименован в axipy.da.geometry.
  - Модуль axipy.da.Style переименован в axipy.da.style.
  - Модуль axipy.da.TabFile переименован в axipy.da.tab_file.
  - Модуль axipy.gui.ActiveToolPanel переименован в axipy.gui.active_tool_panel.
  - Модуль axipy.gui.DialogWrapper переименован в axipy.gui.dialog.
  - Модуль axipy.gui.Notifications переименован в axipy.gui.notifications.
  - Модуль axipy.gui.SelectionManagerWrapper переименован в axipy.gui.selection_manager_.
  - Модуль axipy.gui.ToolWrapper переименован в axipy.gui.map_tool.
  - Модуль axipy.gui.view_manager_wrapper переименован в axipy.gui.view_manager_.
  - Модуль axipy.gui.ViewWrapper переименован в axipy.gui.view.
  - Модуль axipy.gui.WidgetWrapper переименован в axipy.gui.widgets.
  - Модуль axipy.gui.Workspace переименован в axipy.gui.workspace.
  - Модуль axipy.mi.MIGeometry переименован в axipy.mi.mi_geometry.
  - Модуль axipy.render.map переименован в axipy.render.map_.

27.5 4.4 Изменения

Февраль 2023

27.5.1 Исправления

- Методы `axipy.da.CollectionStyle.point()`, `axipy.da.CollectionStyle.line()`, `axipy.da.CollectionStyle.polygon()`, `axipy.da.CollectionStyle.text()` реализованы как свойства.

27.6 4.3 Изменения

Декабрь 2022

27.6.1 Новое

- Модули устанавливаются в папку `installed_modules`.
- Явное указание секции `[general]` в файле метаданных модуля `manifest.ini` необязательно.
- Модули с зависимостями.
- Обход ошибки при импорте библиотеки Matplotlib.
- Раздел Среда разработки.
- Методы `select_by_mouse()` и `select_by_rect()` для выделения геометрий как в инструменте Выбор.
- Доступ к объектам данных в `axipy.da.DataManager` по имени как в словаре `dict`.
- Редактируемый атрибут `axipy.da.Table.schema`.
- Свойства подписей.
- Список загруженных провайдеров `axipy.da.ProviderManager.providers()`.

27.6.2 Исправления

- При выполнении конвертации `axipy.da.MifMidDataProvider.convert_to_tab()` терялись пространственные данные.
- Из-за похожести с `axipy.gui.MapView.device_rect` свойство `axipy.gui.View.rect` переименовано в `axipy.gui.View.position`.
- `axipy.gui.Map.to_image()` не учитывал ограничивающий прямоугольник.
- Метод `axipy.da.Geometry.from_json()` переименован в `axipy.da.Geometry.from_geojson()`.
- Класс `axipy.utl.Printer` переименован в `axipy.utl.FloatFormatter`.
- Для `axipy.da.DataManager.sql_dialect` сменен тип данных с str на enum.

- Свойство axipy.render.Label.placementPolicy вынесено как enum axipy.render.LabelOverlap.

27.7 4.0 Изменения

Июнь 2022

27.7.1 Новое

- Метод создания и показа главного окна axipy.app.MainWindow.show().

27.7.2 Исправления

- Свойство axipy.gui.ReportView.scale() переименовано в axipy.gui.ReportView.view_scale.

27.8 3.7.0 Изменения

Март 2022

27.8.1 Новое

- Методы load()/unload() класса axipy.gui.MapTool; метод axipy.gui.MapTool.deactivate() отмечен как устаревший.
- Метод axipy.gui.MapTool.canDeactivate() переименован в axipy.gui.MapTool.canUnload().
- Функция поиска перевода axipy.tr().

27.8.2 Исправления

- Изменены пределы для свойств axipy.render.RasterLayer.brightness и axipy.render.RasterLayer.contrast на диапазон (-100...100).

27.9 3.5.0 Изменения

Август 2021

27.9.1 Новое

- Новые вспомогательные методы в `axipy.gui.MapTool`.
- Объектно-ориентированный стиль создания кнопок `axipy.menubar.Button`.
- Механизм слежения за значениями `axipy.da.state_manager`.
- Распространение модулей в архивах.
- Объявление модулей с наследованием от `axipy.AxiomaPlugin`.
- Каталог данных содержит таблицу выборки `axipy.da.DataCatalog.selection`.
- Менеджер для запуска и управления пользовательскими задачами `axipy.concurrent.TaskManager`.
- Добавлена панель активного инструмента `axipy.gui.ActiveToolPanel` в которую можно поместить графический элемент упрощающий работу с пользовательским инструментом.

27.9.2 Исправления

- Класс `axipy.da.Collection` переименован в `axipy.da.GeometryCollection`.
- Методы `axipy.da.DataCatalog.tables()`, `axipy.da.DataCatalog.objects()`, `axipy.da.DataCatalog.count()` реализованы как свойства. Метод `axipy.da.Schema.attribute_names()` так-же переделан как свойство.
- Убраны класс `axipy.cs.UnitService` и его экземпляр `axipy.cs.unit`. Их функционал перенесен в базовый класс `axipy.cs.EarthUnit`, который переименован в `axipy.cs.Unit`. Переименованы методы `axipy.cs.LinearUnit.list_all()`, `axipy.cs.AreaUnit.list_all()`.
- Переименован класс `axipy.da.DataCatalog` в `axipy.da.DataManager`
- Переименован класс `axipy.gui.ViewService` в `axipy.gui.ViewManager`
- Переименован класс `axipy.gui.SelectionService` в `axipy.gui.SelectionManager`
- Переименован класс `axipy.da.DataProviders` в `axipy.da.ProviderManager`
- Экземпляр класса `axipy.render.Map` `axipy.render.Map.unit` перенесен в класс `axipy.gui.MapView`.

27.10 3.0.0 Изменения

Апрель 2021

27.10.1 Новое

- Руководство разработчика объединено со справочником функций.
- Свойство временной таблицы `axipy.da.Table.is_temporary`.
- Менеджер контекста `with` для `axipy.da.DataObject`.
- Транзакционная модель редактирования таблиц: `axipy.da.Table.restore()`, `axipy.da.Table.commit()`, `axipy.da.Table.is_modified`, `axipy.da.Table.insert()`, `axipy.da.Table.update()`, `axipy.da.Table.delete()`.
- Каталог объектов данных `axipy.app.MainWindow.catalog` по умолчанию. Открываемые объекты данных автоматически попадают в каталог главного окна. Запросы `axipy.da.DataCatalog.query()` производятся к этому каталогу без явного указания конкретных таблиц.
- Создаваемые окна `axipy.gui.ViewService.create_view()` автоматически добавляются в главное окно программы.
- Настройки ГИС Аксиома `axipy.Settings`.
- Провайдеры данных `axipy.da.DataProviders` со специализированными параметрами для открытия/создания и импорта/экспорта: `tab`, `shp` и другие.
- Раздельные типы стилей: `axipy.da.PointStyle`, `axipy.da.PolygonStyle` и другие.
- Раздельные типы геометрий: `axipy.da.Point`, `axipy.da.Polygon` и другие.
- Загрузка/сохранение рабочих наборов `axipy.app.MainWindow.load_workspace()`, `axipy.app.MainWindow.save_workspace()`.
- Удаление кнопок `axipy.menubar.remove()` приводит к удалению групп и вкладок `axipy.menubar.Position`, если они стали пустыми.

27.10.2 Исправления

- Ошибка при попытке закрытия временной таблицы с изменениями.
- Ошибка при задании разделителя в формате CSV `axipy.da.CsvDataProvider`.

27.11 2.9.0 Изменения

Декабрь 2020

27.11.1 Новое

- Первоначальный релиз.

28

Лицензия

Важно! Нижеприведённые условия являются офертой на заключение безвозмездного Лицензионного договора на предоставление права использования Программы для ЭВМ для указанного в оферте неопределённого круга лиц, намеренных осуществлять использование Программы на личном компьютере не для целей выполнения своих обязанностей по трудовому договору, а также для лиц, использующих Программу: для оценки возможностей Программы, для регистрации Лицензии на Программу в соответствии с ранее заключённым Лицензионным договором, для образовательных и научных целей.

Прочитайте внимательно приведённые ниже условия Лицензионного договора, прежде чем устанавливать, копировать или иным образом использовать Программу для ЭВМ.

Начало использования Программы для ЭВМ, как оно определено условиями приведённого ниже Лицензионного договора, означает Ваше присоединение к приведённому ниже Лицензионному договору и безоговорочное согласие с его условиями в соответствии со ст.1286 ГК РФ.

В случае если Вы не согласны с условиями приведённого ниже Лицензионного договора, не устанавливайте и не используйте Программу или её компоненты.

28.1 БЕЗВОЗМЕЗДНЫЙ ЛИЦЕНЗИОННЫЙ ДОГОВОР

Настоящий Лицензионный договор («Договор») заключён между лицом, присоединившимся к настоящему Договору путём начала использования Программы («Пользователь»), и ООО «ЭСТИ», являющимся обладателем исключительного права на Программу («Правообладатель»).

28.2 ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ, ИСПОЛЬЗУЕМЫЕ В ДОГОВОРЕ

Программа – представленная в объективной форме совокупность данных и команд, предназначенных для функционирования ЭВМ и других компьютерных устройств в целях получения определённого результата, включая подготовительные материалы, полученные в ходе разработки Программы для ЭВМ, и порождаемые ею аудиовизуальные отображения, а именно Программа для ЭВМ ГИС «Аксиома» (зарегистрирована в Едином реестре российских программ для электронных вычислительных машин и баз данных под номером №2174, свидетельство о государственной регистрации Программы для ЭВМ №2016614626).

Правообладатель – обладатель исключительного права на Программу – ООО «ЭСТИ».

Пользователь - физическое лицо (-Вы), которое устанавливает и/или использует Программу от своего лица или правомерно владеет копией Программы. Если Программа была загружена или иным способом использована от имени юридического лица, то под термином Пользователь (-Вы) далее подразумевается юридическое лицо, для которого Программа была загружена или иным способом использована, и которое поручило отдельному физическому лицу принять данное соглашение от своего лица.

Официальный сайт Правообладателя – интернет-сайт, находящийся по адресу: https://axioma-gis.ru

Инсталляция (установка) – запись Программы в память ЭВМ, обеспечивающая возможность её хранения и использования по функциональному назначению.

Программный дистрибутив – комплект файлов, загруженный с Официального сайта Правообладателя и предназначенный для Инсталляции и использования Программы Пользователем. Загрузка Программного дистрибутива означает загрузку самой Программы.

Лицензия - неисключительное непередаваемое право использования Программы по прямому функциональному назначению.

28.3 СТАТЬЯ 1. ОБЩИЕ ПОЛОЖЕНИЯ

1.1. Настоящий Договор устанавливает условия предоставления Правообладателем Пользователю прав использования Программы.

1.2. Настоящий Договор в соответствии со ст.1286 ГК РФ и ст.434 ГК РФ является договором присоединения и заключается в электронной форме путём начала использования Программы Пользователем, как такое начало определено в настоящем Договоре.

1.3. Началом использования Программы Пользователем является совершение Пользователем действий, направленных на воспроизведение Программы, в том числе загрузка Программного дистрибутива с Официального сайта Правообладателя и инсталляция Программы в память ЭВМ или на иной носитель.

1.4. Пользователь присоединяется к настоящему Договору с момента совершения любого из действий, перечисленных в пункте 1.3 настоящего Договора.

28.4 СТАТЬЯ 2. ПРЕДМЕТ ДОГОВОРА

2.1. Правообладатель безвозмездно, на условиях простой (неисключительной) лицензии, предоставляет Пользователю непередаваемое право использования Программы по прямому функциональному назначению путём воспроизведения на всей территории Российской Федерации. Для использования Программы Пользователю разрешается скопировать Программный дистрибутив и инсталлировать Программу.

2.2. Права на использование Программы предоставляются на весь срок действия исключительного права.

2.3. Пользователями Программы на условиях настоящего Договора могут быть следующие юридические лица:

2.3.1. Лица, использующие Программу исключительно в ознакомительных целях для оценки её функциональных возможностей.

2.3.2. Лица, использующие Программу исключительно для регистрации Лицензии на Программу в соответствии с ранее заключённым Лицензионным договором.

2.3.3. Бюджетные учреждения высшего и среднего образования.

2.3.4. Автономные учреждения высшего и среднего образования.

2.3.5. ФГБУ «Российская академия наук».

2.3.6. Научные организации, находящиеся под научно-методическим руководством президиума ФГБУ «Российская академия наук».

2.4. Пользователями Программы на условиях настоящего Договора могут быть следующие физические лица:

2.4.1. Лица, использующие Программу исключительно на личном компьютере не для целей выполнения своих обязанностей по трудовому договору.

2.4.2. Лица, использующие Программу исключительно в ознакомительных целях для оценки её функциональных возможностей.

2.4.3. Лица, использующие Программу исключительно для регистрации Лицензии на Программу в соответствии с ранее заключённым Лицензионным договором.

2.4.4. Лица, являющиеся учащимися бюджетных учреждений высшего и среднего образования.

2.4.5. Лица, являющиеся учащимися автономных учреждений высшего и среднего образования.

2.4.6. Сотрудники бюджетных учреждений высшего и среднего образования.

2.4.7. Сотрудники автономных учреждений высшего и среднего образования.

2.4.8. Сотрудники ФГБУ «Российская академия наук».

2.4.9. Сотрудники научных организаций, находящихся под научно-методическим руководством президиума ФГБУ «Российская академия наук».

28.5 СТАТЬЯ 3. ДОПОЛНИТЕЛЬНЫЕ УСЛОВИЯ

3.1. Правообладатель гарантирует, что он обладает всеми законными основаниями для предоставления Пользователю права использования Программы в пределах, установленных настоящим соглашением.

3.2. Права на использование Программы предоставляется Пользователю на условиях «как есть». Правообладатель отказывается от любых гарантий на Программу, прямо указанных или подразумеваемых, включая, но не ограничиваясь гарантиями пригодности для определённой Пользователем цели, а также не гарантирует, что Программа будет отвечать персональным требованиям и ожиданиям Пользователя, предъявляемым им Программе, аналогам, стандартам, не описанным в сопроводительной документации или на Официальном сайте Правообладателя. Правообладатель не гарантирует, что Программа полностью свободна от ошибок и будет работать с иной конфигурацией ЭВМ, чем указанные на Официальном сайте Правообладателя и/или будет совместима с иными программами, установленными на ЭВМ Пользователя.

3.3. Правообладатель не несёт ответственности за прямой и/или косвенный ущерб, причинённый Пользователю, а также не возмещает Пользователю убытки (включая упущенную выгоду), понесённые Пользователем, в том числе, в результате ненадлежащего качества каналов связи общего пользования, политику обмена трафиком между провайдерами, нормальное функционирование сети Интернет, её частей или за качество линий связи, не имеющих отношения к собственным ресурсам Правообладателя, и за их доступность для Пользователя.

3.4. Пользователь гарантирует, что конечный пользователь (физическое лицо) использует Программу не для целей получения прибыли в рамках коммерческой деятельности юридических лиц, а исключительно для образовательных или учебных целей, либо для получения дохода исключительно физическим лицом, использующим Программу на личном компьютере.

3.5. Пользователь гарантирует, что до присоединения к настоящему Договору он ознакомился с описанием Программы, требованиями к системно-аппаратной платформе, и иной информацией, относящейся к Программе, размещённой на Официальном сайте Правообладателя.

3.6. Пользователь не вправе продавать, отчуждать, передавать иным образом права на использование Программы, Программный дистрибутив третьим лицам.

3.7. Пользователь соглашается с тем, что Программа, документация, как и все другие объекты авторского права, а также системы, идеи и методы работы, другая информация, которая содержится в Программе, товарные знаки являются объектами интеллектуальной собственности Правообладателя. Данный Договор не даёт Пользователю никаких прав на использование объектов интеллектуальной собственности, включая товарные знаки и знаки обслуживания Правообладателя, за исключением прав, предоставляемых настоящим Лицензионным договором.

3.8. Пользователь не вправе модифицировать или изменять Программу никаким способом. Запрещается удалять или изменять наименование Программы, присутствующие в Программе, документации или иных материалах, распространяемых

с Программой, знаки охраны авторского права или иные указания на Правообладателя, уведомления об авторских правах на любой копии Программы.

3.9. Использование Программы в составе программных продуктов Пользователя возможно исключительно с согласия Правообладателя, выраженного в отдельном письменном соглашении. При этом указание названия Программы и Правообладателя является обязательным.

3.10. Использование несколькими физическими лицами экземпляра Программы, в том числе инсталлированной на серверной ЭВМ, возможно только с согласия Правообладателя, выраженного в отдельном письменном соглашении.

3.11. При разработке Программы использовались открытые лицензии. Авторские права на открытое программное обеспечение сохраняются за соответствующими правообладателями. Указание на правообладателей и лицензионные соглашения, по которым предоставляется открытое программное обеспечение, имеются в папке, входящей в состав Программного дистрибутива.

28.6 СТАТЬЯ 4. ЗАКЛЮЧИТЕЛЬНЫЕ ПОЛОЖЕНИЯ

4.1. Настоящее Лицензионное соглашение регулируется в соответствии с законодательством Российской Федерации.

4.2. Досудебный порядок урегулирования споров является обязательным для Сторон. Претензия, направляемая в рамках досудебного порядка разрешения споров, должна быть мотивирована и содержать в себе описание конкретных достоверных обстоятельств неисполнения/нарушения существенных условий настоящего Договора. Претензия, оформленная без соблюдения условий, содержащихся в настоящем пункте, не признаётся Сторонами в качестве документа, направленного на соблюдение досудебного порядка разрешения спора. Сторона, получившая претензию, обязана направить мотивированный ответ на неё другой Стороне в срок не позднее 50 (Пятидесяти) дней с момента получения претензии. В случае недостижения согласия спор передаётся на рассмотрение в судебный орган в соответствии с установленной законом подведомственностью по месту нахождения Правообладателя.

4.3. Все изменения и дополнения к настоящему Договору признаются действительными только в случае составления их в письменной форме и подписания уполномоченными представителями каждой из Сторон, если возможность иного изменения условий настоящего Договора не предусмотрена самим Договором.

4.4. Настоящий Договор может изменяться Правообладателем в одностороннем порядке. Уведомление Пользователя о внесённых изменениях в условия настоящего Соглашения публикуется на Официальном сайте Правообладателя. Указанные изменения в условиях Договора вступают в силу с даты их публикации, если иное не оговорено в соответствующей публикации.

4.5. В случае нарушения Пользователем какого-либо из условий настоящего Договора, Правообладатель вправе отказаться от настоящего Договора в одностороннем порядке и заблокировать использование Программы.

4.6. Если какое-либо положение настоящего Лицензионного договора будет признано аннулированным, недействительным, не имеющим юридической силы или незаконным, то остальные положения настоящего Лицензионного договора сохраняют свою полную силу и действие.

4.7. Условия настоящего Договора являются превалирующими над условиями какого-либо иного лицензионного договора в отношении той же Программы между Пользователем и Правообладателем, если только в таком лицензионном договоре прямо не предусмотрено, что его положения имеют приоритет над настоящим Договором.

28.7 СТАТЬЯ 5. РЕКВИЗИТЫ ПРАВООБЛАДАТЕЛЯ

Правообладатель	ООО «ЭСТИ»		
ИНН	7704613038	КПП	770401001
Юридический адрес	119002, г. Москва, пер. Сивцев Вражек, д. 29/16, пом.2		
Адрес для почтовой корреспонденции	119002, г. Москва, пер. Сивцев Вражек, д. 29/16, пом.2		
Телефон, E-mail	+7 499 241-42-06; info@axioma-gis.ru		

Содержание модулей Python

a

Алфавитный указатель

Non-alphabetical

K

M

П

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z