ПЕТРОЗАВОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
КАФЕДРА ИНФОРМАТИКИ И МАТЕМАТИЧЕСКОГО ОБЕСПЕЧЕНИЯ

Отчёт по курсу
«Верификация ПО»

Выполнил:
студент 6 курса группы 22609
В. В. Дмитриев

Преподаватель:
к.ф-м.н., доцент К. А. Кулаков

Петрозаводск
2013

Оглавление

Объект исследования

Osmium является быстрым и гибким C++ и Java-инструментарием и системой для работы с OSM данных. Он может, среди прочего, читать и писать OSM файлы в различных форматах, составлять путь и мультиполигональные геометрии, и преобразовывать OSM-данные в шейп-файлы и другие форматы. Он может обрабатывать OSM- данные с или без информации об истории объекта.

Osmium предаставляет C + + разработчикам большой инструментарий OSM связанных блоков, которые он может собрать в точности так, как он должен эффективно работать с данными OSM. Некоторые из этих блоков позволяют настроить использование Javascript, что позволяет конечным пользователям настроить приложение.

Osmium является библиотекой с открытым исходным кодом и доступна под лицензией GNU LGPL или GPL v. 3 или более поздней версии.

Архитектура

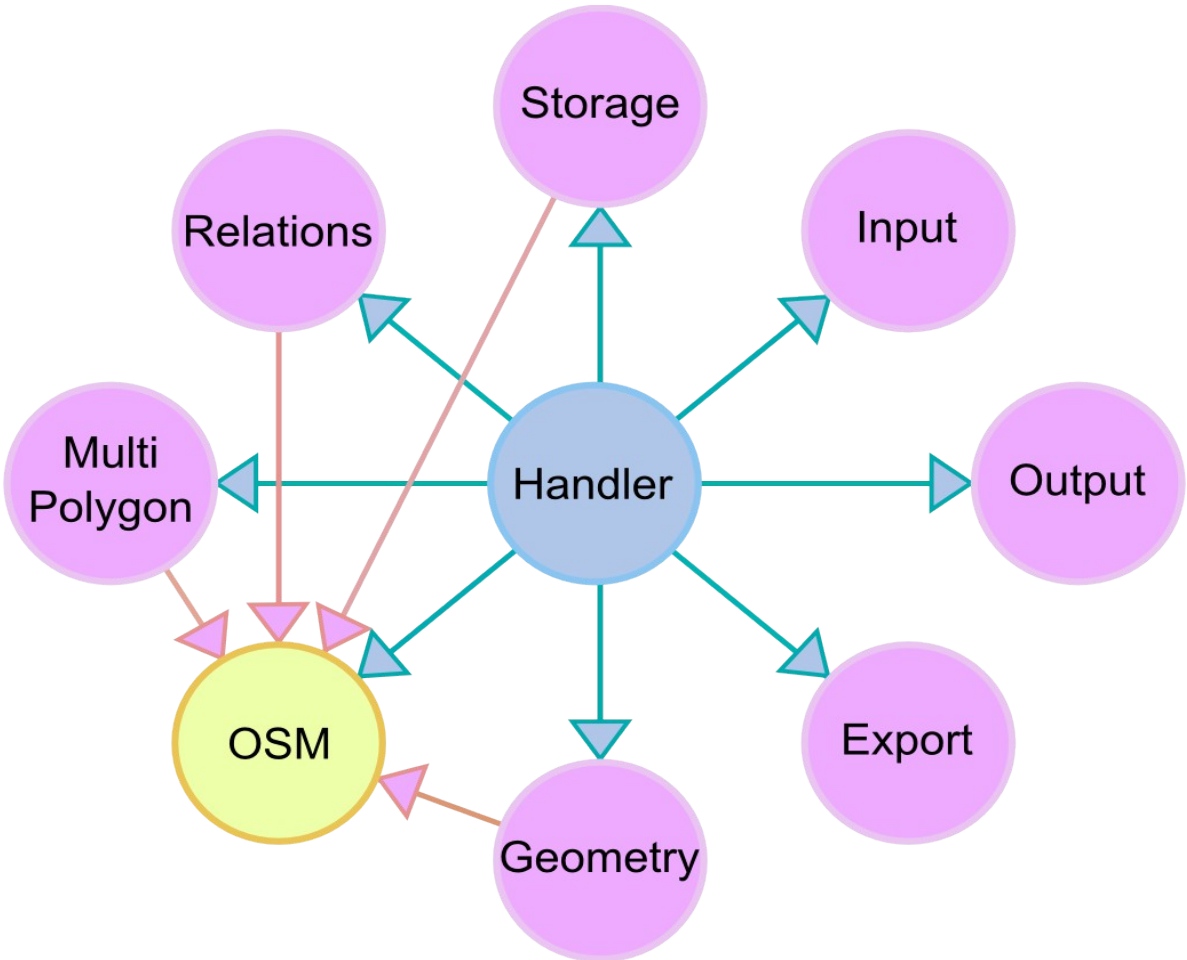

Рис. 1 Диаграмма пространств имён

На Рис. 1 отображаются связи между следующими пространствами имён:

Export — пространство имён, которое содержит классы осуществления экспорта в не-OSM форматы, таких как шейп-файлы.

Geometry — пространство имён, которое содержит классы геометрических элементов.

Handler — пространство имён, которое содержит обработчики оперирующие OSM данными через обратные вызовы.

Input — пространство имён, которое содержит классы входных данных, которые разобрают OSM-файлы и вызывают обработчик.

MultiPolygon — пространство имён, которое содержит пространство имен для кода, связанного с построением мультиполигонов из соотношений.

OSM — пространство имён, которое содержит классы основных OSM-данных.

Output — пространство имён, которое содержит классы для записи OSM-данных.

Relations — пространство имён, которое содержит классы OSM-связей.

Storage — пространство имён, которое содержит классы хранения данных.

В качестве основных пространств выступают Handler и OSM. Handler обрабатывает и преобразует данные из классов других пространств имён, а OSM содержит классы которые являются основными для многих классов других пространств имён.

Для тестирования мы будем исследовать несколько основных классов. Во-первых это будут классы пространства имён OSM, содержащие основные OSM-данные. Также рассмотрим класс MultiPolygon пространства имён Geometry, который основывается на данных из классов пространства имён OSM. Так как основная работа идёт с OSM-файлами, то рассмотрим класс OSMFile, не принадлежащий ни к одному из пространств имён. Для чтения из OSM-файлов используем функцию read из пространства имён Input. Для хранения данных будем использовать класс ObjectStore из пространства имён Storage, а для вывода и преобразований данных — класс Debug из пространства имён Handler. Взаимодействие перечисленных классов отображено на Рис. 2.

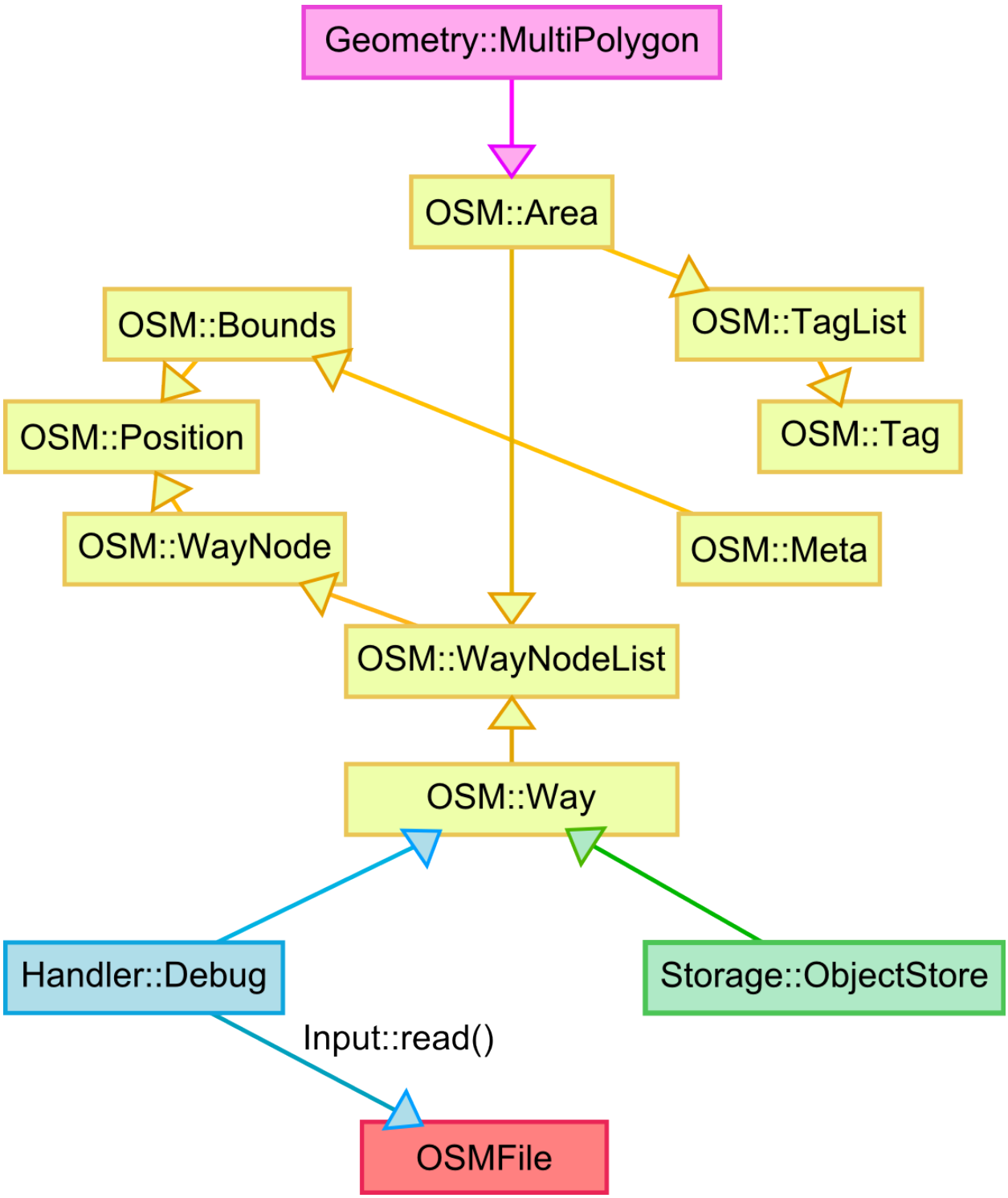

Рис. 2 Диаграмма классов

План тестирования

Тестирование будут проходить следующие классы:

1. Geometry::MultiPolygon (Класс для создания мультиполигона)

2. Handler::Debug (Класс вывода OSM данных)
   1. void init (Osmium::OSM::Meta &meta)
   2. void way (const shared_ptr< Osmium::OSM::Way const > &way) const

3. Input (Классы входных данных)
    1. void read (const Osmium::OSMFile & file, T & handler)

4. OSM::Area (Класс для хранения данных об области)
    1. bool from_way () const
    2. osm_object_id_t orig_id () const

5. OSM::Bounds (Класс для хранения данных о границе области)
    1. Bounds & extend (const Position &position)
    2. bool defined () const
    3. Position bl () const
    4. Position tr () const

6. OSM::Meta
    1. Meta & has_multiple_object_versions (bool h)

7. OSM::Position (Класс для хранения данных позиции)
    1. bool defined () const
    2. int32_t x () const
    3. int32_t y () const
    4. double lon () const
    5. double lat () const
    6. Position & lon (double lon)
    7. Position & lat (double lat)

8. OSM::Tag (Класс для хранения данных тега)
    1. const char * key () const
    2. const char * value () const

9. OSM::TagList (Класс для составления списка тегов)
    1. void add (const char *key, const char *value)
    2. const char * get_tag_by_key (const char *key) const
    3. const char * get_tag_key (unsigned int n) const
    4. const char * get_tag_value (unsigned int n) const

10. OSM::Way (Класс для храния данных пути)
    1. void add_node (osm_object_id_t ref)
    2. osm_object_id_t get_first_node_id () const
    3. osm_object_id_t get_last_node_id () const

11. OSM::WayNode (Класс для храния данных узла пути)
    1. osm_object_id_t ref () const
    2. WayNode & ref (osm_object_id_t ref)
    3. bool has_position () const
    4. double lon () const

5. double lat () const

12. OSM::WayNodeList (Класс для составления списка узлов пути)
    1. void clear ()
    2. WayNodeList & add (const WayNode &way_node)
    3. WayNodeList & add (osm_object_id_t ref)

13. Storage::ObjectStore (класс для хранения данных)

14. OSMFile (Класс для работы с OSM-файлами)

Тесты

Блочное тестирование будет применено к основным функциям классов.

Интеграционное тестирование будет проходить в следующем порядке:

1) Handler::Debug → OSMFile
2) ObjectStore::ApplyHandler → Storage::ObjectStore + Handler::Debug debug + OSM::Meta meta;
3) Geometry::MultiPolygon → OSM::Area → OSM::TagList →OSM::Tag
4) Handler::Debug → OSM::Meta + OSM::Way

Типы тестов:

- П – простой
- О – общий
- С – специальный
- Н – негативный
- К – краевой

Блочные тесты

Класс Area

*Area::f*rom_way
Описание: Создана ли эта область из пути
Входные данные: нет
Косвенные данные: id – идентификатор области
Алгоритм:
1. Создать объект класса *Area*
2. У объекта вызвать функцию *f*rom_way

Результат: true если создана из пути, иначе false

№	Исходные данные	Результат	Тип
1	id – значение для созданных из пути	true	О
2	id – значение для несозданных из пути	false	О
3	id не определено	*сообщение об ошибке*	Н

*Area::*orig_id
Описание: Получить личный идентификатор области
Входные данные: нет
Косвенные данные: id – идентификатор области
Алгоритм:
1. Создать объект класса *Area*
2. У объекта вызвать функцию orig_id

Результат: личный идентификатор области

№	Исходные данные	Результат	Тип
4	id – любое значение	true	О
5	id не определено	*сообщение об ошибке*	Н

Класс Bounds

Bounds::extend
Описание: расширяет границу добавлением точки
Входные данные: Position & position — координаты
Косвенные данные:
int32_t m_min_x — минимальное значение по оси x,
int32_t m_max_x — максимальное значение по оси x,
int32_t m_min_y — минимальное значение по оси y,
int32_t m_max_y — максимальное значение по оси y
Алгоритм:
1. Создать объект класса *Bounds*
2. У объекта вызвать функцию extend

Результат: изменение переменных максимальной или минимальной точки, если точка не входит в ограниченную область

№	Исходные данные	Результат	Тип
1	position – значения превышающие значения максимальной точки	Изменились значения максимальной точки	О
2	position – значения находящиеся в границе	Ничего не изменилось	О
3	position – пустой элемент в результате некорректного создания	Ничего не изменилось	Н

Bounds::defined
Описание: проверяет существование объекта
Входные данные: нет
Косвенные данные:
int32_t m_min_x — минимальное значение по оси x,
int32_t m_max_x — максимальное значение по оси x,
int32_t m_min_y — минимальное значение по оси y,
int32_t m_max_y — максимальное значение по оси y
Алгоритм:
1. Создать объект класса *Bounds*
2. У объекта вызвать функцию defined
Результат: true если объект существует и не пустой, иначе false

№	Исходные данные	Результат	Тип
4	Для int32_t m_min_x, int32_t m_max_x, int32_t m_min_y, int32_t m_max_y заданы значения	true	O
5	нет	false	O

Класс Meta

Meta::has_multiple_object_versions
Описание: проверяет имеет ли объект несколько версий, а так же может может изменить эту информацию
Входные данные: bool h — булева переменная задающая новое значение внутренней переменной
Косвенные данные: bool m_has_multiple_object_versions — внутренняя переменная показывающая имеет ли объект несколько версий
Алгоритм:
1. Создать объект класса *Meta*
2. У объекта вызвать функцию has_multiple_object_versions
Результат: true если у объекта несколько версии, иначе false, при изменении данных возвращает изменёный объект класса *Meta*

№	Исходные данные	Результат	Тип
1	нет	false	O
2	h - true	Изменёный объект класса *Meta*	O
3	нет	true	O

Класс Position

Position::defined
Описание: проверяет определена ли позиция
Входные данные: нет

Косвенные данные: int32_t m_x — текущая позиция по оси x
Алгоритм:
1. Создать объект класса *Position*
2. У объекта вызвать функцию defined
Результат: true если определена, иначе false

№	Исходные данные	Результат	Тип
1	нет	false	O
2	m_x — любое значение	*true*	O

Position::x
Описание: возвращает переменную x
Входные данные: нет
Косвенные данные: int32_t m_x — текущая позиция по оси x
Алгоритм:
1. Создать объект класса *Position*
2. У объекта вызвать функцию x
Результат: число

№	Исходные данные	Результат	Тип
3	m_x — любое значение	число	O
4	нет	*Бесконечно малое или большое число*	H

Position::y
Описание: возвращает переменную y
Входные данные: нет
Косвенные данные: int32_t m_y — текущая позиция по оси y
Алгоритм:
1. Создать объект класса *Position*
2. У объекта вызвать функцию y
Результат: число

№	Исходные данные	Результат	Тип
5	m_y — любое значение	число	O
6	нет	*Бесконечно малое или большое число*	H

Position::lon
Описание: возвращает долготу, или записывает новое значение
Входные данные: double lon
Косвенные данные: int32_t m_x — текущая позиция по оси x
Алгоритм:
1. Создать объект класса *Position*
2. У объекта вызвать функцию lon

Результат: *число или объект класса Position, если изменили значение*

№	Исходные данные	Результат	Тип
7	нет	Бесконечно малое или большое число	Н
8	lon — какое-то значение	Изменёный объект класса *Position*	О
9	нет	число	О

Position::lat
Описание: возвращает широту, или записывает новое значение
Входные данные: double lat
Косвенные данные: int32_t m_y — текущая позиция по оси y
Алгоритм:
1. Создать объект класса *Position*
2. У объекта вызвать функцию lat
Результат: *число или объект класса Position, если изменили значение*

№	Исходные данные	Результат	Тип
10	нет	Бесконечно малое или большое число	Н
11	lat — какое-то значение	Изменёный объект класса *Position*	О
12	нет	число	О

Класс Tag

Tag::key
Описание: возвращает ключ тэга
Входные данные: нет
Косвенные данные: std::string m_key – текущий ключ тэга
Алгоритм:
1. Создать объект класса *Tag*
2. У объекта вызвать функцию key
Результат: строка

№	Исходные данные	Результат	Тип
1	m_key — набор символов	строка	О
2	нет	*пустая строка*	Н

Tag::value
Описание: возвращает значение тэга
Входные данные: нет
Косвенные данные: std::string m_value — текущее значение тэга
Алгоритм:
1. Создать объект класса *Tag*
2. У объекта вызвать функцию value

Результат: строка

№	Исходные данные	Результат	Тип
3	m_value — набор символов	строка	О
4	нет	*пустая строка*	Н

Класс TagList

TagList::add
Описание: добавляет элемент в список тэгов
Входные данные: osm_object_id_t ref — значение нового элемента
Косвенные данные: *WayNodeList m_node_list — текущий список элементов*
Алгоритм:
1. Создать объект класса *TagList*
3. У объекта класса *TagList* вызвать функцию add
Результат: добавлен новый элемент

№	Исходные данные	Результат	Тип
1	*ref — целое значение*	добавлен новый элемент	О
2	*ref — пустой*	добавлен новый элемент	Н

TagList::get_tag_by_key
Описание: получение тэга по ключу
Входные данные: const char* key — ключ тэга
Косвенные данные: *нет*
Алгоритм:
1. Создать объект класса *TagList*
2. Добавить объекты класса Tag в объект класса TagList
3. У объекта класса *TagList* вызвать функцию get_tag_by_key
Результат: возвращает тэг, если элемент не найден возвращает 0

№	Исходные данные	Результат	Тип
3	*key — ключ существующего тэга*	тэг	О
4	*key — ключ несуществующего тэга*	*0*	Н
5	*key — пустой*	0	Н

TagList::get_tag_key
Описание: получение ключа тэга по позиции
Входные данные: unsigned int n — позиция в списке тегов
Косвенные данные: *нет*
Алгоритм:
1. Создать объект класса *TagList*

2. Добавить объекты класса Tag в объект класса TagList
3. У объекта класса *TagList* вызвать функцию get_tag_key
Результат: возвращает ключ, если элемент не найден, то выводит сообщение об ошибке

№	Исходные данные	Результат	Тип
6	*n — номер существующего элемента*	ключ	О
7	*n — номер существующего элемента*	*сообщение об ошибке*	Н

TagList::get_tag_value
Описание: получение значение тэга по позиции
Входные данные: unsigned int n — позиция в списке тегов
Косвенные данные: *нет*
Алгоритм:
1. Создать объект класса *TagList*
2. Добавить объекты класса Tag в объект класса TagList
3. У объекта класса *TagList* вызвать функцию get_tag_value
Результат: возвращает тэг, если элемент не найден, то выводит сообщение об ошибке

№	Исходные данные	Результат	Тип
8	*n — номер существующего элемента*	Значение тэга	О
9	*n — номер существующего элемента*	*сообщение об ошибке*	Н

Класс Way

Way::add_node
Описание: добавляет элемент в список узлов пути
Входные данные: osm_object_id_t ref — значение нового элемент
Косвенные данные: *нет*
Алгоритм:
1. Создать объект класса *Way*
3. У объекта класса *Way* вызвать функцию add_node
Результат: добавлен элемент в список узлов пути

№	Исходные данные	Результат	Тип
1	*ref — какое-то значение*	Значение тэга	О

Way::get_first_node_id
Описание: возвращает идентификатор первого элемента
Входные данные: нет

Косвенные данные: WayNodeList m_node_list — текущий список узлов пути
Алгоритм:
1. Создать объект класса *Way*
2. У объекта вызвать функцию get_first_node_id
Результат: идентификатор первого элемента

№	Исходные данные	Результат	Тип
2	m_node_list — набор узлов пути	идентификатор первого элемента	О
3	m_node_list — пуст	*Бесконечно малое или большое число*	Н

Way::get_last_node_id
Описание: возвращает идентификатор последнего элемента
Входные данные: нет
Косвенные данные: WayNodeList m_node_list — текущий список узлов пути
Алгоритм:
1. Создать объект класса *Way*
2. У объекта вызвать функцию get_last_node_id
Результат: идентификатор последнего элемента

№	Исходные данные	Результат	Тип
4	m_node_list — набор узлов пути	идентификатор последнего элемента	О
5	m_node_list — пуст	*Бесконечно малое или большое число*	Н

Класс WayNode

WayNode::ref
Описание: возвращает значение, или записывает новое значение
Входные данные: osm_object_id_t ref — новое значение
Косвенные данные: m_ref — текущие значение
Алгоритм:
1. Создать объект класса *WayNode*
2. У объекта вызвать функцию ref
Результат: *число или объект класса WayNode, если изменили значение*

№	Исходные данные	Результат	Тип
1	нет	Бесконечно малое или большое число	Н
2	ref — какое-то значение	Изменёный объект класса *WayNode*	О
3	нет	число	О

WayNode::has_position
Описание: проверяет есть ли позиция у узла
Входные данные: нет
Косвенные данные: Position m_position — текущая позиция
Алгоритм:
1. Создать объект класса *WayNode*
2. У объекта вызвать функцию has_position
Результат: true если определена, иначе false

№	Исходные данные	Результат	Тип
4	m_position — не пустой	true	O
5	нет	*false*	O

WayNode::lon
Описание: возвращает долготу
Входные данные: нет
Косвенные данные: Position m_position — текущая позиция
Алгоритм:
1. Создать объект класса *WayNode*
2. У объекта вызвать функцию lon
Результат: *число*

№	Исходные данные	Результат	Тип
6	нет	Бесконечно малое или большое число	H
7	m_position — не пустой	*число*	O

WayNode::lat
Описание: возвращает широту
Входные данные: нет
Косвенные данные: Position m_position — текущая позиция
Алгоритм:
1. Создать объект класса *WayNode*
2. У объекта вызвать функцию lat
Результат: *число*

№	Исходные данные	Результат	Тип
8	нет	Бесконечно малое или большое число	H
9	m_position — не пустой	*число*	O

Класс WayNodeList

WayNodeList:: add
Описание: добавляет новую точку пути
Входные данные: osm_object_id_t ref или const WayNode & way_node — новый элемент
Косвенные данные: нет
Алгоритм:
1. Создать объект класса *WayNodeList*
2. *Создать объект класса WayNode (необязательный пункт)*
3. У объекта вызвать функцию add

Результат: новый элемент в списке

№	Исходные данные	Результат	Тип
1	ref — целое значение	Новый элемент в списке	О
2	way_node — объект с целым значением	Новый элемент в списке	О
3	way_node — пустой объект	Новый элемент в списке со значением 0	Н

WayNodeList::clear
Описание: очищает список
Входные данные: нет
Косвенные данные: std::vector<WayNode> m_list — текущий список элементов
Алгоритм:
1. Создать объект класса *WayNodeList*
2. У объекта класса *WayNodeList* вызвать функцию clear

Результат: пустой список элементов

№	Исходные данные	Результат	Тип
4	Список элементов в объекте класса *WayNodeList*	Пустой список элементов	О
5	Пустой список элементов в объекте класса *WayNodeList*	Пустой список элементов	С

Интеграционные тесты

1. Handler::Debug → OSMFile

1) Создаётся объект класса OSMFile в нём записывается OSM-файл
2) Создаётся объект класса Handler::Debug
3) По средствам функции read из пространства имён Input вся информация из объекта класса OSMFile записывается в объект класса Handler::Debug

№	Исходные данные	Результат	Тип
1	Корректный OSM-файл	Преобразованная информация из OSM-файла	O
2	Некорректный OSM-файл	Неправильное преобразование информации	C
3	Несуществующий OSM-файл	Ошибка в лог	H

2. Storage::ObjectStore::ApplyHandler → Storage::ObjectStore + Handler::Debug + OSM::Meta meta;

1) Создаётся объект класса OSMFile в нём записывается OSM-файл
2) Создаётся объект класса Storage::ObjectStore
3) В объект класса Storage::ObjectStore записываем данные из OSM-файла
4) Создаётся объект класса Handler::Debug
5) Создаётся объект класса OSM::Meta meta
6) Создаётся объект класса ObjectStore::ApplyHandler в качестве параметров задаются объекты классов Handler::Debug, OSM::Meta meta, Storage::ObjectStore

№	Исходные данные	Результат	Тип
1	Корректный OSM-файл	Создается объект класса ApplyHandler с преобразованными данными	O
2	Некорректный OSM-файл	Неправильное преобразование	C
3	Несуществующий OSM-файл	Пустой класс	H

3. Geometry::MultiPolygon → OSM::Area → OSM::TagList →OSM::Tag

1) Создаётся список тегов (объект класса OSM::TagList)
2) Добавляем в объект класса OSM::TagList объекты класса OSM::Tag
3) Создаётся область (объект класса OSM::Area)
4) Заполняем объект класса OSM::Area необходимыми переменные (идентификатор, пользователь, версия и т.д.)
5) Добавляем в объект класса OSM::Area объект класса OSM::TagList
6) Далее создаем мультиполигон (объект класса Geometry::MultiPolygon) задавая в качестве параметра объект класса OSM::Area

№	Исходные данные	Результат	Тип
1	Список тегов	Мультиполигон	O
2	Список тегов с присутствием пустых элементов	Некорректный мультиполигон	C
3	Пустой список тегов	Пустой объект класса Geometry::MultiPolygon	H

4. Handler::Debug → OSM::Meta + OSM::Way

1. Создаётся объект класса Handler::Debug
2. Создаётся объект класса OSM::Meta
3. Инициализируется объект класса Handler::Debug при помощи функии init, в параметр которой помещаем объект класса OSM::Meta
4. Создаётся объект класса OSM::Way в него помещается путь

   Добавляем объект класса OSM::Way в объект класса Handler::Debug при помощи функции way для дальнейшей обработки информации

№	Исходные данные	Результат	Тип
1	Путь	объект класса Handler::Debug обладает данными для дальнейшей обработки	O
2	Пустой путь	объекту класса Handler::Debug не имеет данных для дальнейшей работы	H

Аттестационные тесты

В аттестационном тесте запускается тест, осуществляющий следующий сценарии:

1) Считываются данные из OSM-файла
2) Извлекаются все объекты
3) Извлекаются все связи между объектами
4) Строятся дополнительные объекты и пути
5) Отображается вся информация об объектах и связях
6) Информация сохраняется об объектах и связях во всех других форматах

Примеры реализации тестов

Для проведения тестов использовали Boost Unit Test Framework. Рассмотрим пример написания блочного теста для класса `WayNodeList`.

```
#include <boost/test/unit_test.hpp>
// Подключение библиотеки тестирования

#include <osmium/osm/way_node_list.hpp>
// Подключение класса  WayNodeList

BOOST_AUTO_TEST_SUITE(WayNodeList)
// Инициализации дерева тестов

BOOST_AUTO_TEST_CASE(set_position) {
     // Последовательность тестов «set_position»
     Osmium::OSM::WayNodeList wnl;
     //Создаётся объект класса WayNodeList
     //Он является списком узлов пути

     BOOST_CHECK_EQUAL(wnl.size(), 0);
     // Проверяем что список пуст

     Osmium::OSM::WayNode wn(5);
     //Создаётся объект класса WayNode со значением  5
     //Он является узел пути
     wnl.add(wn);
     //Добавляем узел в список
     BOOST_CHECK_EQUAL(wnl.size(), 1);
     // Проверяем что список содержит один элемент
     BOOST_CHECK_EQUAL(wnl[0].ref(), 5);
     // Проверяем что значение элемента списка равно значению 5

     wnl.add(17);
     //Добавляем напрямую значение в список
     BOOST_CHECK_EQUAL(wnl.size(), 2);
     // Проверяем что список содержит два элемента
     BOOST_CHECK_EQUAL(wnl[1].ref(), 17);
     // Проверяем что значение элемента списка равно значению 17

     wnl.clear();
```

```
    // Очищаем список от элементов
    BOOST_CHECK_EQUAL(wnl.size(), 0);
    // Проверяем что список пуст
}
BOOST_AUTO_TEST_SUITE_END()
// Завершаем тестирование
```

Резутьтаты тестирования:

```
Running 4 test cases...

*** No errors detected
```

По умолчанию результаты тестирования выводятся в терминал, но библиотека имеет возможность вывода отчета в различные форматы файла.

Результаты

В ходе разработки и выполнения тестов были получены следующие результаты:

Блочные тесты

Класс	Количество тестов	Количество ошибок
Area	5	0
Bounds	6	0
Meta	3	0
Position	12	4
Tag	4	0
TagList	9	0
Way	5	2
WayNode	9	3
WayNodeList	5	0

Перечень обнаруженных ошибок:

Position::x, Position::y, Position::lon, Position::lat, Way::get_first_node_id, Way::get_last_node_id, WayNode::ref, WayNode::lon, WayNode::lat — в объектах, которым принадлежат эти функции, нет проверки на их существование. Если в программе выполнится очистка объекта, то при запросе к данным этого объекта каким-нибудь модулем будут получено бесконечно большое или малое число.

1.

Интеграционные тесты

Связка	Количество тестов	Количество ошибок
Handler::Debug → OSMFile	3	1
ObjectStore::ApplyHandler → Storage::ObjectStore + Handler::Debug debug + OSM::Meta meta;	3	0

Geometry::MultiPolygon→ OSM::Area→ OSM::TagList →OSM::Tag	3	0
Handler::Debug → OSM::Meta + OSM::Way	2	0

Перечень обнаруженных ошибок:

1. OSMFile – не отслеживаются случаи, в которых файл не существует.

Аттестационный тест был пройден библиотекой успешно.